

Amanda Pereira da Silva Santos, Dhayane dos Santos Pinheiro, Erik Rodrigues dos Santos, Letícia Amaral Xavier e Riquelvi dos Santos Passos

Apostila para linguagem Lua

APRENDA A PROGRAMAR EM LUA

Guia Prático



```
mainScene:onCreate()  
btn = ccui.Button:create("res/Button_Normal.png")  
btn.scale9Enabled(true)  
btn.contentSize(cc.size(70,70))  
btn.setPosition(100,100)  
btn.setTitleColor(cc.c3b(0,0,0))  
btn.setTitleFontSize(30)  
btn.setTitleText("Add")  
btn.addEventListener(function(ref,typ)  
    typ == ccui.TouchEventType.ended then  
        self.sprite = cc.Sprite:create("res/Button_Normal.png")  
        for i=1,1000000 do  
            self.sprite["s_"..i] = {"abc",123}  
        end  
        self.sprite:setPosition(100, 200)  
        self:addChild(self.sprite)  
    end  
end)  
self:addChild(btn)  
  
btn2 = ccui.Button:create("res/Button_Normal.png")  
btn2.scale9Enabled(true)  
btn2.contentSize(cc.size(70,70))  
btn2.setPosition(200,100)  
btn2.setTitleColor(cc.c3b(0,0,0))  
btn2.setTitleFontSize(30)  
btn2.setTitleText("remove")  
btn2.addEventListener(function(ref,typ)  
    typ == ccui.TouchEventType.ended then  
        self:removeFromParent(true)  
        for k, v in pairs(self) do  
            self.sprite[k] = nil  
        end  
        self.sprite = nil  
    end  
end)  
self:addChild(btn2)
```

Sumário

PREFÁCIO.....	3
DOWNLOAD E INSTALAÇÃO.....	4
Source.....	4
IDE.....	9
Conhecendo a IDE.....	10
Novo Projeto.....	10
CONHECENDO A LINGUAGEM.....	11
Variáveis.....	12
Operações aritméticas.....	13
Operadores relacionais.....	14
Estruturas condicionais.....	15
Laços de repetição.....	15
Primeiro Projeto.....	16
TIBIA.....	17
BOT.....	18
Macros.....	18
Scripts.....	18
DESENVOLVIMENTO.....	18
CASE 01.....	19
Editor de Macros.....	19
Magias de cura.....	20
Poções e runas de cura.....	22
Magias Utilitárias.....	25
CASE 02.....	31
Iniciando o TargetBot.....	31
Programando os Ataques.....	39
Editor da Lista de Criaturas.....	48
Controle de Foco do Bot.....	54
Programação do Sistema de Loot.....	57
Criando as funções andar para o targetbot.....	73
Finalizando o TargetBot.....	75
CASE 02.1.....	88
Criando a interface.....	88
Desenvolvendo o menu suspenso.....	91
Finalizando a interface.....	95
CASE 03.....	99
Iniciando o CaveBot.....	99

Criando o arquivo de Configurações.....	111
Criando o editor de ações.....	115
Exemplo de função.....	123
Criando o Gravador.....	125
Criando as funções para andar.....	128
Criando as funções de depósito de item.....	132
Criando as funções de “supply”.....	133
Finalizando o CaveBot.....	134
CASE 3.1.....	144
Criando a interface.....	144
Interface de configuração.....	146
Criando a interface do editor.....	148
Finalizando a interface.....	149
ENCERRAMENTO DO APRENDIZADO.....	151
REFERÊNCIAS.....	152

PREFÁCIO

Print("Olá, Mundo!")

Criada em meados de 1993 por Roberto Ierusalimsky, Luiz Henrique de Figueiredo e Waldemar Celes, membros do *Computer Graphics Technology Group* na PUC-Rio, a Pontifícia Universidade Católica do Rio de Janeiro, no Brasil, a linguagem de programação lua foi desenvolvida por este grupo de alunos e hoje é utilizada em diferentes aplicações, incluindo até mesmo jogos de videogame, inclusive, poucas pessoas sabem que a linguagem de programação Lua é utilizada no *Roblox*, por exemplo. Após três décadas desde seu desenvolvimento, a linguagem Lua é responsabilidade do laboratório LabLua do Departamento de Informática da PUC-Rio, eles são responsáveis pelas atualizações.

Um fato curioso sobre a linguagem Lua é que inicialmente era um projeto para a empresa Petrobras, além disso, a linguagem se chamava Sol, um dos criadores inclusive chegou a explicar que anteriormente se chamava sol por ser uma linguagem grande e pesada, até que uma nova versão menor e mais leve foi criada e foi nomeada como Lua.

A linguagem Lua possui um alto nível de portabilidade, justamente por ser leve e apresentar alguns arquétipos da programação, que são: uma linguagem orientada a objetos, linguagem que permite a programação funcional e uma linguagem orientada a dados.

A tipagem de dados do Lua trabalha de forma dinâmica, cujo valores jamais terão tratamento com um tipo errado, como normalmente acontece com as linguagens não tipadas. Contudo, Lua é uma linguagem interpretada e precisa de um compilador para executar trechos de códigos existentes.

Em Lua, também é possível ter um autogerenciamento da memória, ou seja, significa que ela pode ser gerenciada automaticamente, logo facilitando alguns métodos como a criação de interfaces e precaução de erros que possam acontecer na alocação de memória, erros complexos e de difícil resolução. Dessa maneira, a linguagem Lua oferece o tratamento para campos do tipo texto (*strings*) e estruturas de dados como tabelas

DOWNLOAD E INSTALAÇÃO

Lua é uma das linguagens de programação mais versáteis, por isso a linguagem é usada em várias plataformas, incluindo aplicativos móveis, servidores e outros dispositivos.

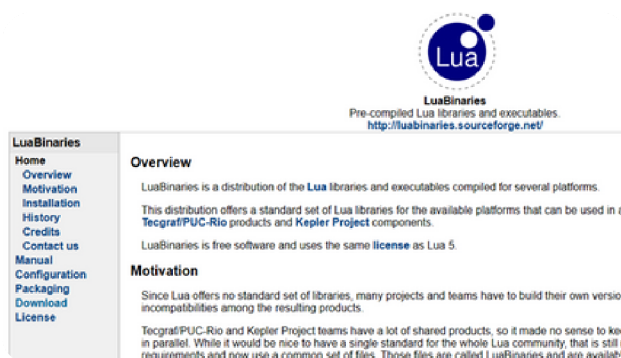
Vamos mostrar como realizar o *download* e a instalação de maneira rápida e fácil, passando por cada etapa de forma clara e direta.

Source

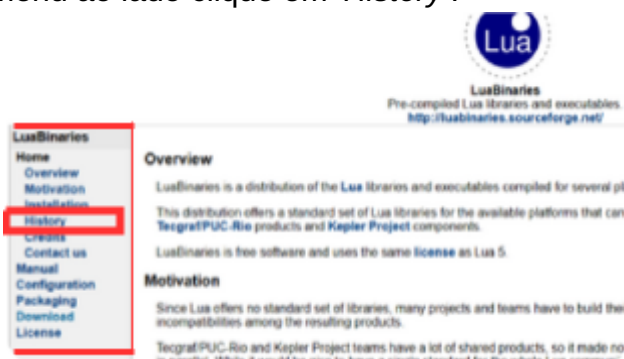
O primeiro passo para baixar a *Source* do Lua é entrar no link abaixo:

<https://luabinaries.sourceforge.net/>

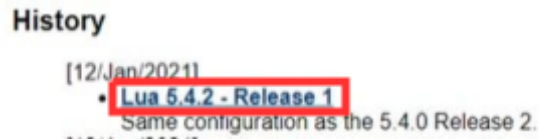
O link irá direcionar para uma página de bibliotecas e executáveis do Lua já pré-compilados.



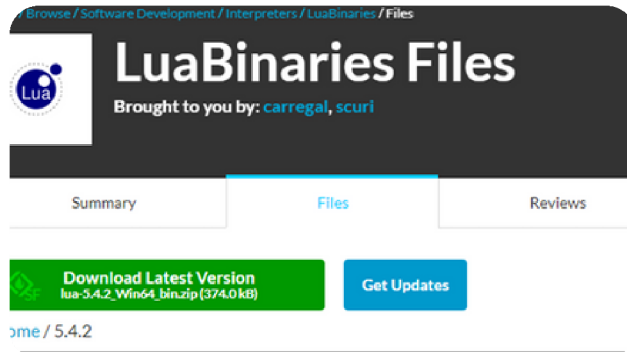
No menu ao lado clique em '*History*'.



Escolha a opção mais recente ou a que utilizaremos nesta apostila.



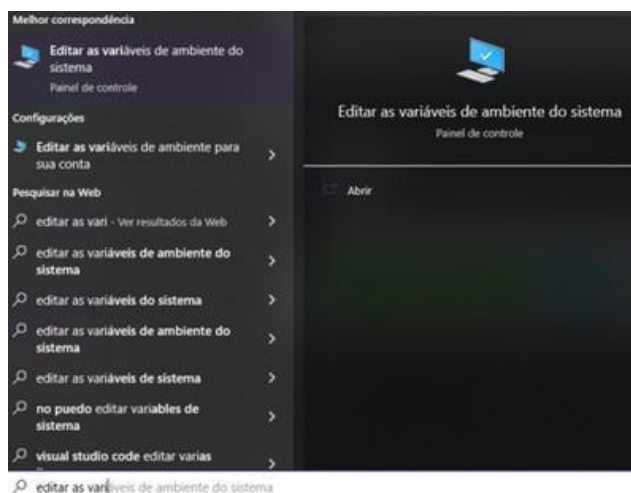
Ao clicar no link, será direcionado a página de download.



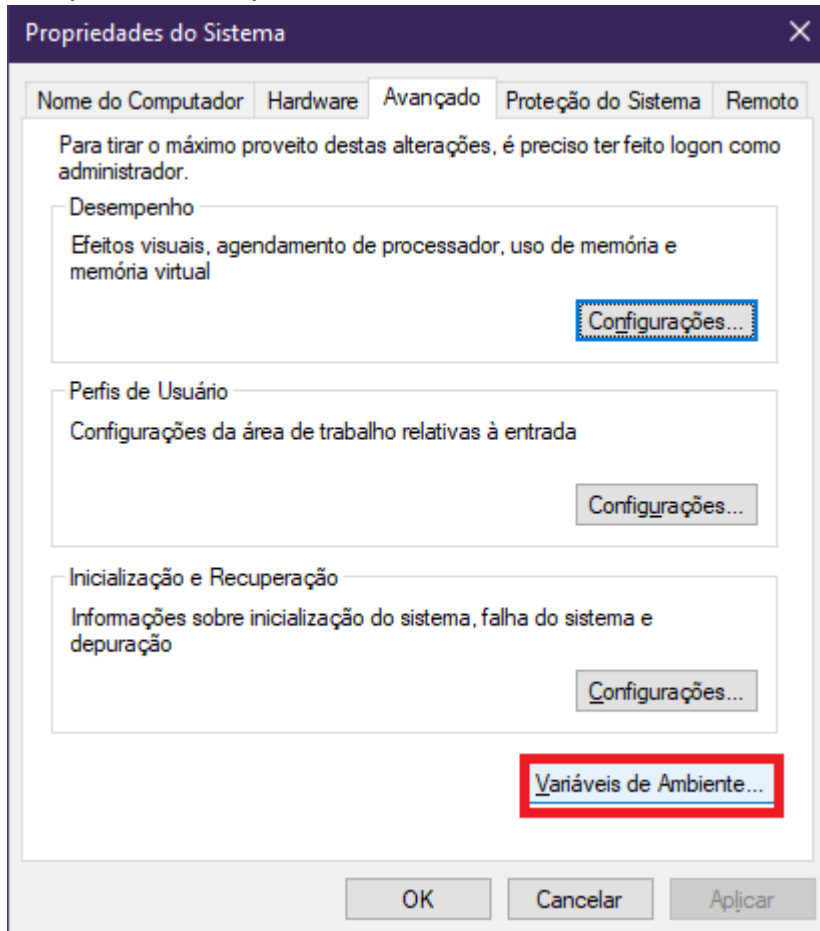
Após fazer o download, localize o arquivo em sua máquina e o descompacte.

Nome	Data de modificação	Tipo	Tamanho
lua-5.4.2_Win64_bin	04/03/2023 21:14	Arquivo ZIP do Wi...	366 KB
lua54.dll	12/01/2021 14:35	Extensão de aplica...	348 KB
lua54	12/01/2021 14:38	Aplicativo	120 KB
lua54	12/01/2021 14:38	Aplicativo	293 KB
wlua54	12/01/2021 14:38	Aplicativo	123 KB

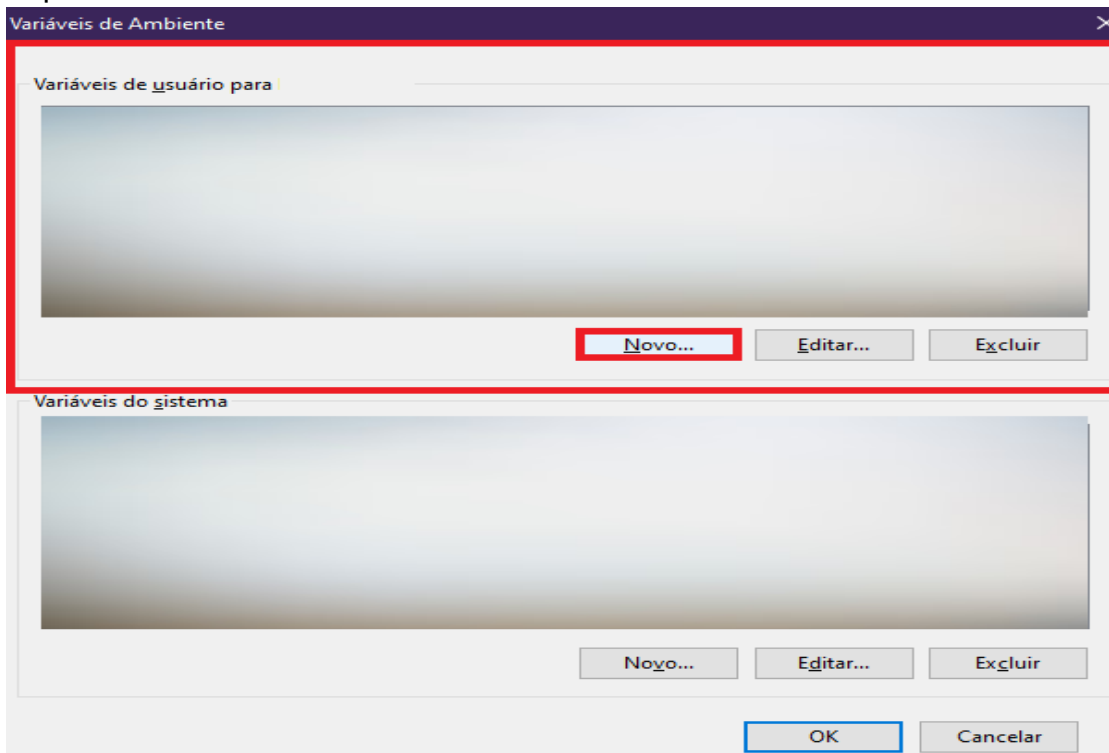
Em sua máquina pesquise por 'Editar as variáveis de ambiente do sistema'.



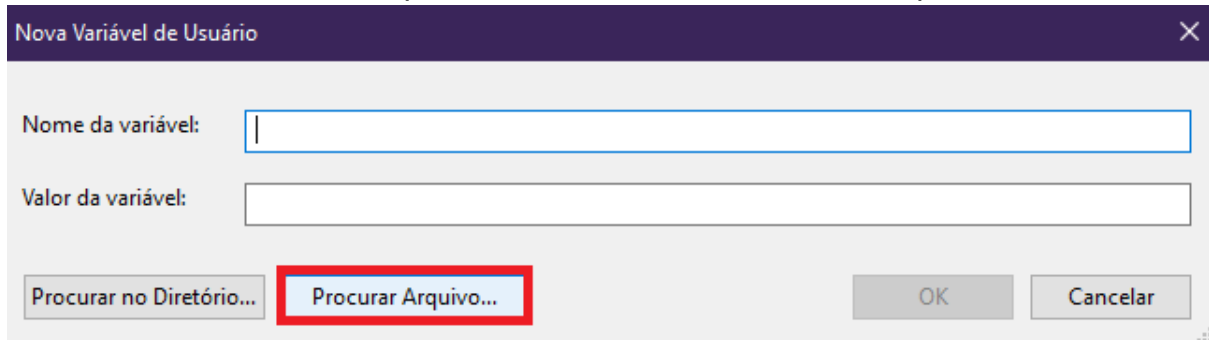
Após abrir, clique em 'Variáveis de Ambiente'.



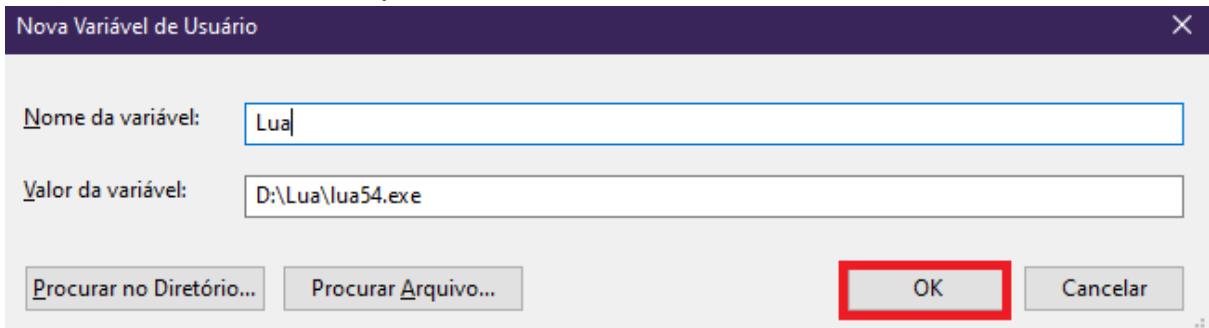
No menu de Variáveis de Ambiente, na sessão 'Variáveis de usuário para ____', clique em 'Novo'.



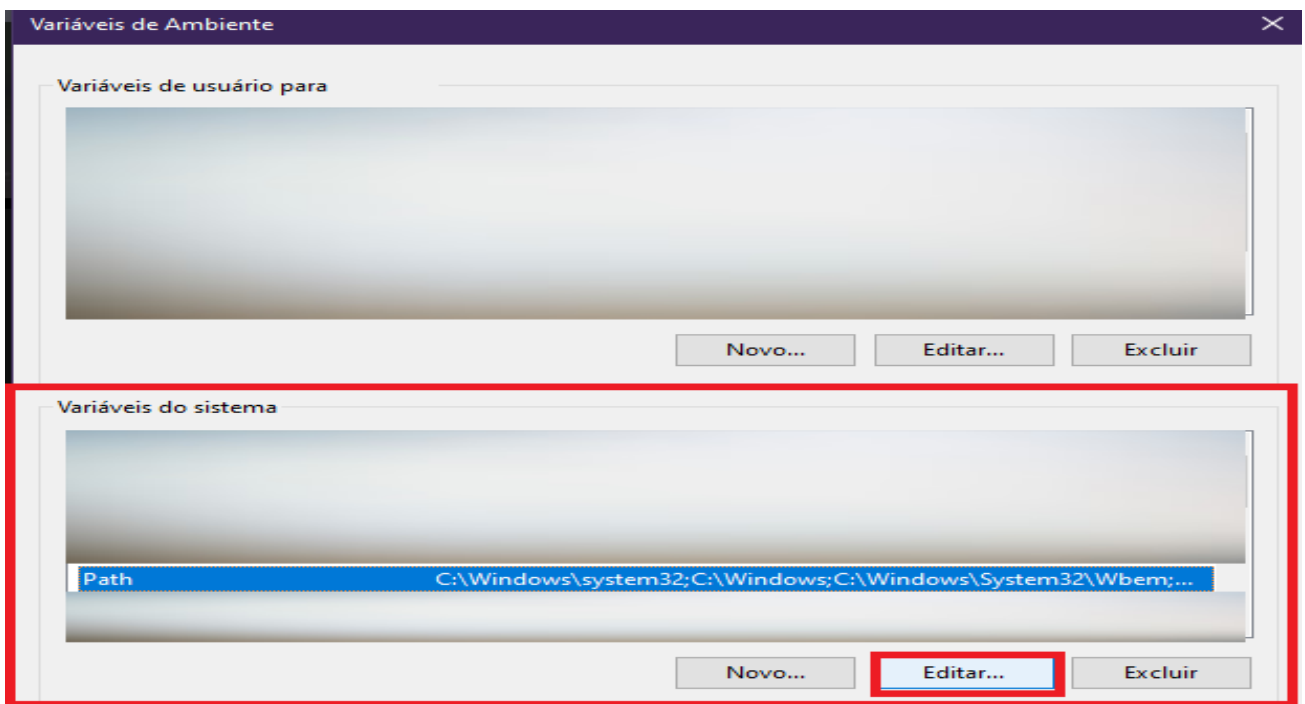
Procure o diretório do arquivo executável do Lua em sua máquina.



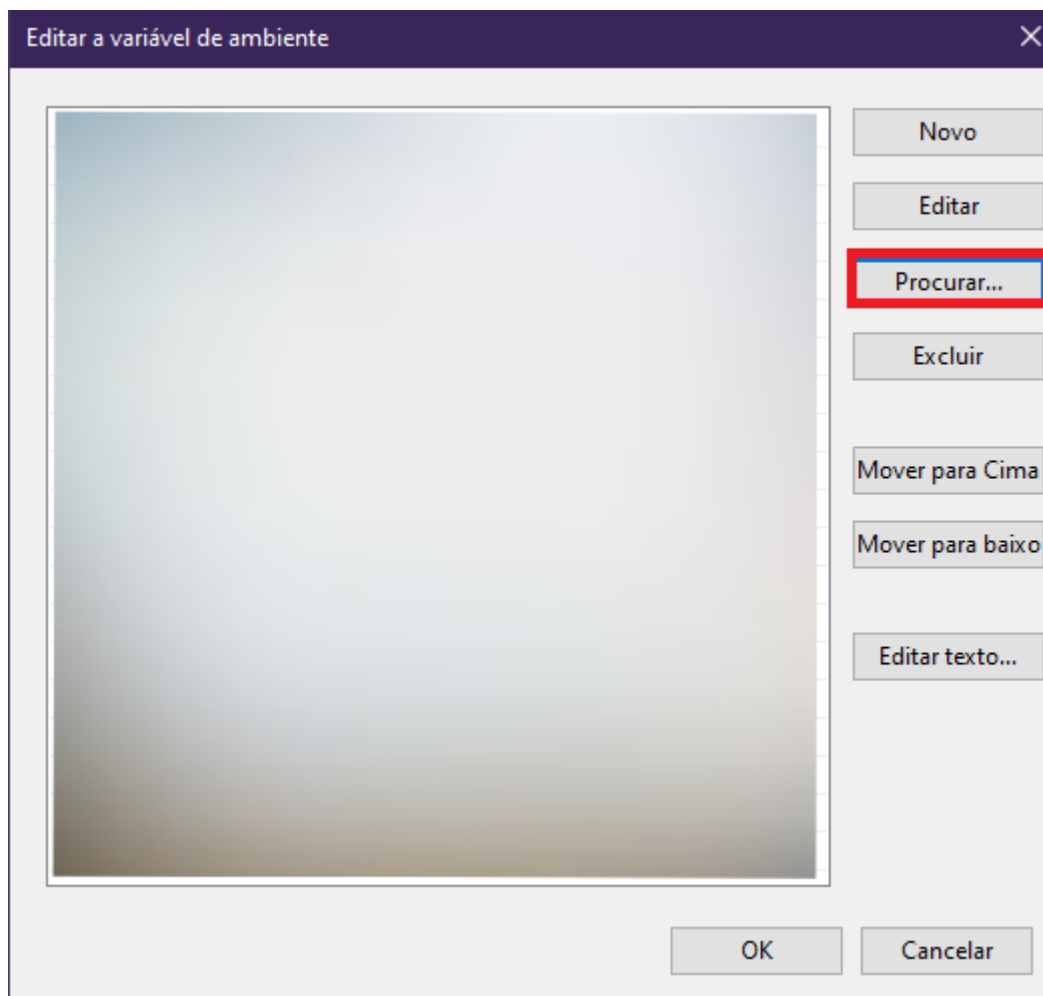
Quando o encontrar, clique em 'OK'.



Ainda no menu de Variáveis de Ambiente, clique no botão 'Editar' que está na seção 'Variáveis do Sistema'



Após isso clique em 'Procurar'.



Encontre os arquivos de download do Lua e clique em 'Ok'.

Pronto, agora já podemos fechar o aplicativo.

IDE

O primeiro passo para baixar a IDE é clicar no link abaixo:

<https://www.geany.org/download/releases/>

O link irá direcionar para a página de download da IDE

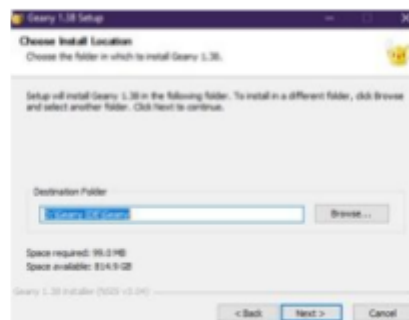
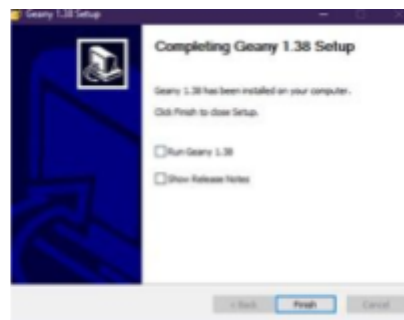


Distribution	File	GPG Signature	GPG Key
Source (tar.gz)	geany-1.38.tar.gz	geany-1.38.tar.gz.gpg (instructions)	codebears.pubkey.txt
Source (tar.bz2)	geany-1.38.tar.bz2	geany-1.38.tar.bz2.gpg (instructions)	codebears.pubkey.txt
Windows (64-bit)	geany-1.38_setup.exe	geany-1.38_setup.exe.gpg (instructions)	48d16.pubkey.txt
macOS	geany-1.38_mac4.dmg geany-1.38_mac_arm64.dmg	-	-

Release notes for Geany 1.38.0

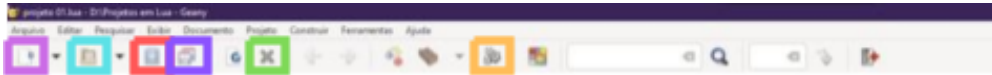
Deve-se então baixar a versão que é recomendável para o seu Sistema Operacional. Após o download, será possível encontrar o arquivo na pasta 'Downloads' da sua máquina.







Execute o arquivo .exe baixado e siga o passo a passo de instalação da IDE.



Conhecendo a IDE

Neste capítulo, vamos explicar a função de cada botão nesta IDE, para que seja possível aproveitar ao máximo todas as ferramentas disponíveis. Vamos começar com uma visão geral dos principais elementos da interface, explicando seu uso e finalidade. Ao final deste capítulo, teremos um conhecimento sólido sobre como usar os botões, dessa forma aumentando a produtividade.

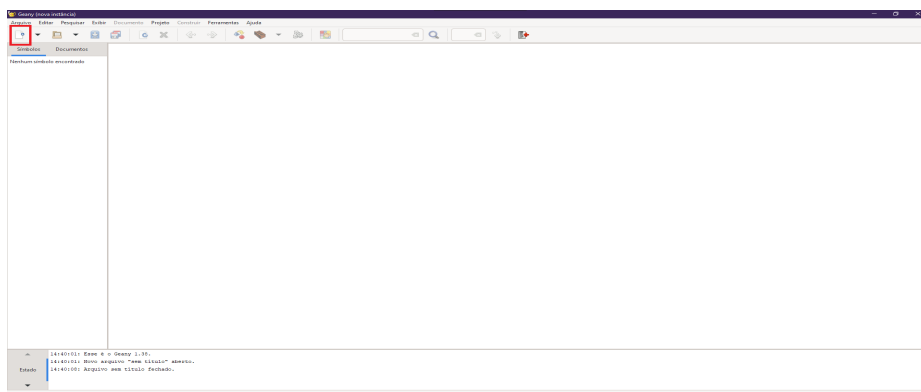


-  Cria um arquivo em branco
-  Abre arquivos
-  Salva o arquivo atual
-  Salva todos os arquivos abertos
-  Fecha o arquivo atual
-  Compila o projeto

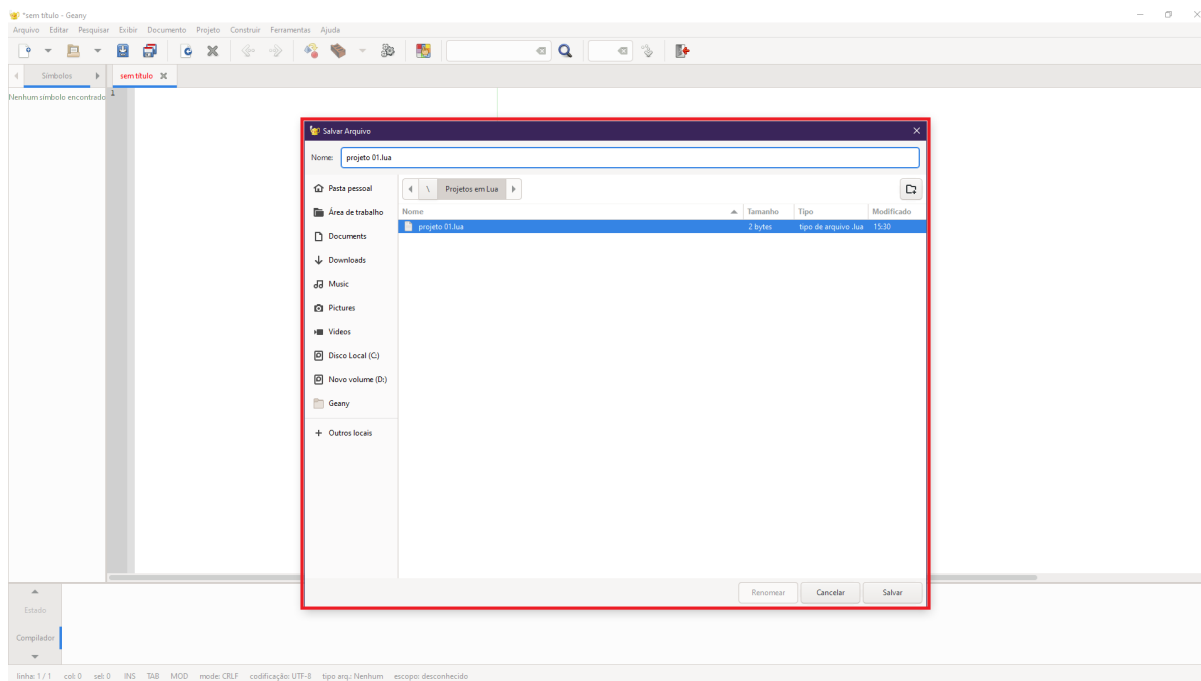
Novo Projeto

Aqui será ensinado como iniciar um novo projeto. Serão apresentadas dicas úteis para facilitar o processo de criação do projeto e torná-lo mais eficiente. Ao final deste capítulo, o leitor terá adquirido conhecimentos essenciais para iniciar um novo projeto de forma autônoma e eficaz.

Para começar, clique no ícone de criar um arquivo branco.



Salve o arquivo ou aperte Ctrl + S.
Em seguida, nomeie o arquivo com a extensão '.lua'.



Para compilar um arquivo é necessário salvá-lo antes.

CONHECENDO A LINGUAGEM

Independentemente do nível de experiência em programação, este capítulo foi desenvolvido para ajudar a entender a linguagem LUA de uma maneira clara e simples.

Serão apresentados os principais conceitos da linguagem LUA, desde as variáveis mais básicas, até os mais diversos operadores aritméticos, passando pelos laços de repetição e outros elementos fundamentais. Através de exemplos práticos, será possível aprender a escrever programas simples em LUA, explorando todo o potencial da linguagem.

Variáveis

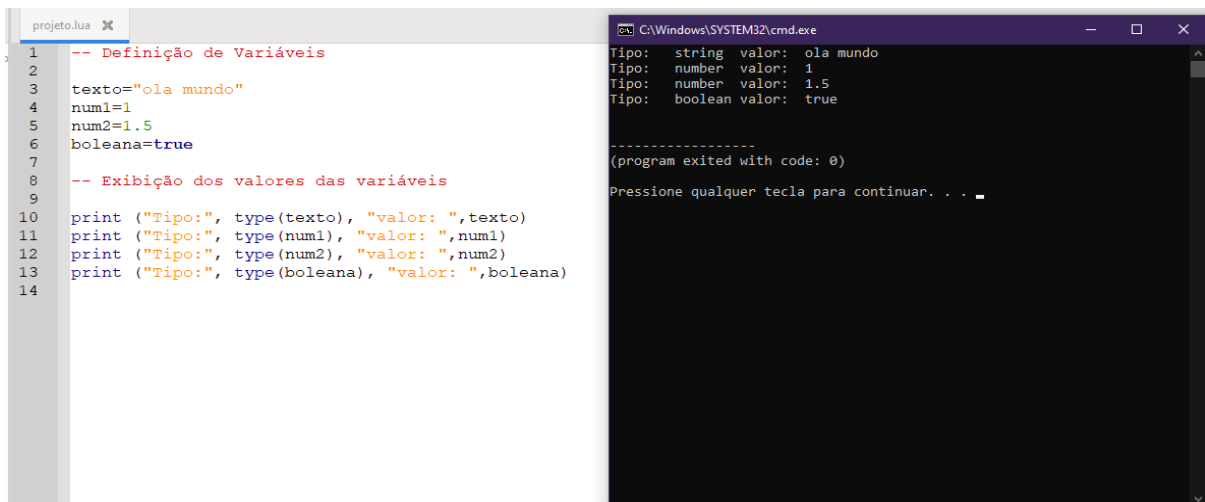
Em Lua, as variáveis são utilizadas para armazenar dados em memória, como números, textos e valores lógicos, que podem ser utilizados posteriormente em cálculos, comparações e outras operações.

Ao contrário de outras linguagens de programação, Lua é uma linguagem dinamicamente tipada, o que significa que as variáveis não precisam ser declaradas com um tipo específico.

Ao atribuir um valor a uma variável, o tipo da variável é definido automaticamente pelo tipo do valor atribuído. Por exemplo, se você atribuir um número inteiro a uma variável, essa variável será automaticamente definida como um número inteiro.

Ainda assim, é importante entender os tipos de dados disponíveis em Lua para utilizá-los corretamente em suas variáveis. Os principais tipos de dados em Lua são:

- Números: podem ser números inteiros ou números de ponto flutuante, representados sem o uso de separadores de milhares.
- Strings: são sequências de caracteres, delimitadas por aspas simples ou duplas.
- Booleanos: podem ter apenas dois valores possíveis, verdadeiro ou falso.
- Tabelas: são estruturas de dados que podem armazenar outros valores e são utilizadas para criar arrays, listas e dicionários.

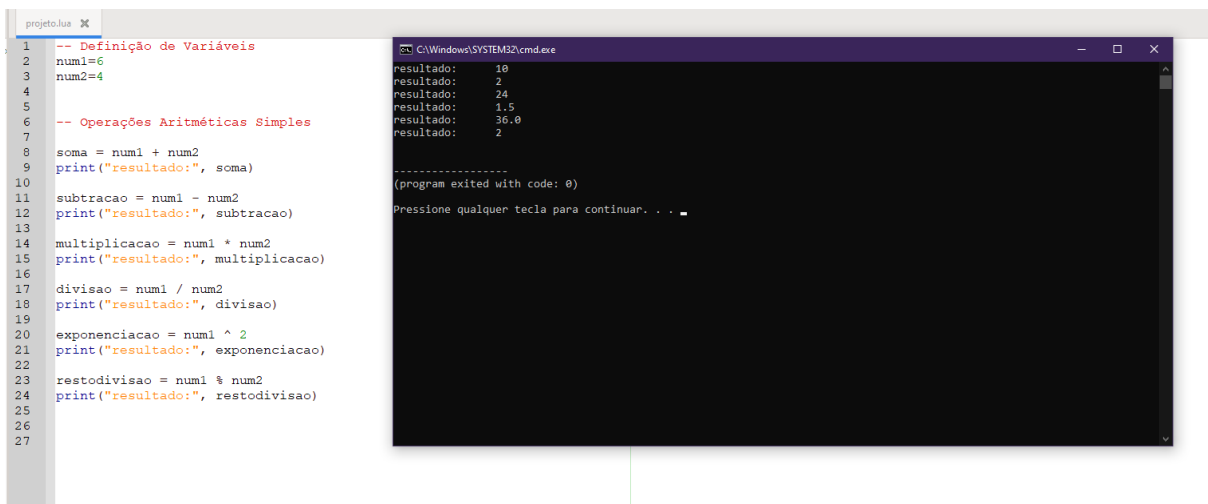


```
projeto.lua X
1  -- Definição de Variáveis
2
3  texto="ola mundo"
4  num1=1
5  num2=1.5
6  booleana=true
7
8  -- Exibição dos valores das variáveis
9
10 print ("Tipo:", type(texto), "valor: ", texto)
11 print ("Tipo:", type(num1), "valor: ", num1)
12 print ("Tipo:", type(num2), "valor: ", num2)
13 print ("Tipo:", type(booleana), "valor: ", booleana)
14
```

```
C:\Windows\SYSTEM32\cmd.exe
Tipo:  string  valor:  ola mundo
Tipo:  number  valor:  1
Tipo:  number  valor:  1.5
Tipo:  boolean  valor:  true
-----
(program exited with code: 0)
Pressione qualquer tecla para continuar. . .
```

Operações aritméticas

- + Soma: Operador utilizado para realizar a adição entre valores;
- Subtração: Operador utilizado para encontrar valores ao diminuir um número por outro, sendo o oposto da adição;
- * Multiplicação: Operador utilizado para realizar uma soma sucessiva de um número por ele mesmo.
- / Divisão: Operador utilizado para realizar a repartição de um valor por um outro valor.
- ^ Exponenciação: Operador utilizado para representar uma multiplicação por fatores iguais.
- % Resto da divisão: Operador utilizado para retornar o resto da divisão entre dois valores.



The image shows a code editor window on the left and a terminal window on the right. The code editor contains the following Lua code:

```
1 -- Definição de Variáveis
2 num1=6
3 num2=4
4
5
6 -- Operações Aritméticas Simples
7
8 soma = num1 + num2
9 print("resultado:", soma)
10
11 subtracao = num1 - num2
12 print("resultado:", subtracao)
13
14 multiplicacao = num1 * num2
15 print("resultado:", multiplicacao)
16
17 divisao = num1 / num2
18 print("resultado:", divisao)
19
20 exponenciacao = num1 ^ 2
21 print("resultado:", exponenciacao)
22
23 restodivisao = num1 % num2
24 print("resultado:", restodivisao)
25
26
27
```

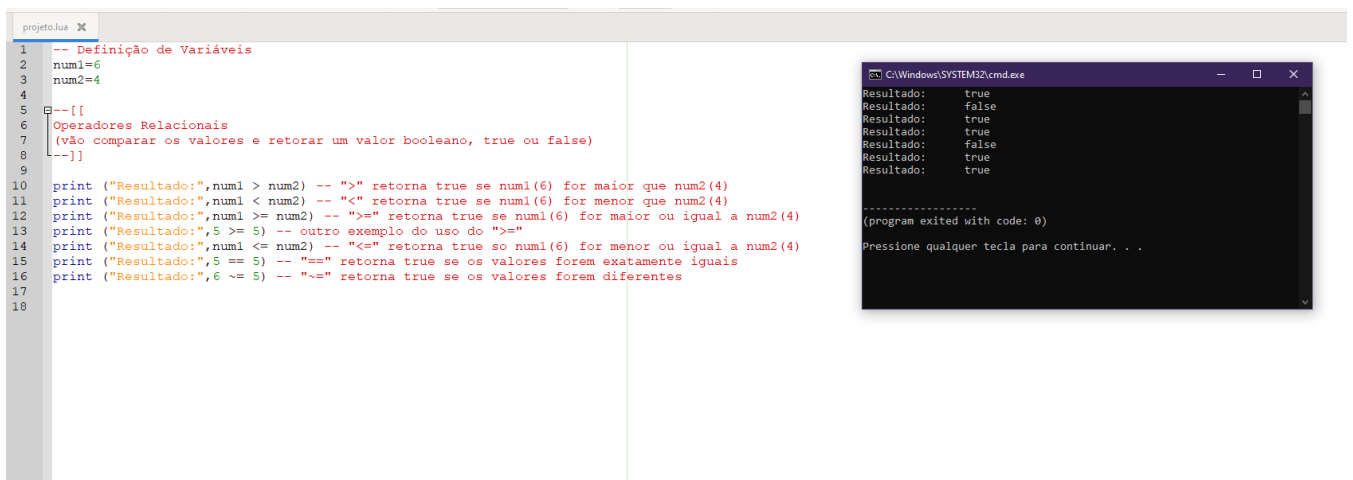
The terminal window shows the output of the code:

```
C:\Windows\SYSTEM32\cmd.exe
resultado: 10
resultado: 2
resultado: 24
resultado: 1.5
resultado: 36.0
resultado: 2
-----
(program exited with code: 0)
Pressione qualquer tecla para continuar. . . .
```

Operadores relacionais

Os operadores relacionais são uma classe de operadores em linguagens de programação que permitem comparar dois valores e verificar se eles satisfazem uma determinada condição. Em Lua, os operadores relacionais são os seguintes:

- > Maior que: verifica se o primeiro valor é maior que o segundo valor.
- < Menor que: verifica se o primeiro valor é menor que o segundo valor.
- >= Maior ou igual a: verifica se o primeiro valor é maior ou igual ao segundo valor.
- <= Menor ou igual a: verifica se o primeiro valor é menor ou igual ao segundo valor.
- == Igual a: verificar se os dois valores são iguais.
- ~= Diferente de: verificar se os dois valores são diferentes.



The image shows a code editor window titled 'projeto.lua' and a terminal window. The code in the editor defines two variables, num1=6 and num2=4, and then prints the results of several relational operations. The terminal window shows the output of these operations, confirming the results: true for '>', '>=', and '=='; false for '<', '<=', and '!='.

```
1 -- Definição de Variáveis
2 num1=6
3 num2=4
4
5
6 Operadores Relacionais
7 (vão comparar os valores e retornar um valor booleano, true ou false)
8 ---]
9
10 print ("Resultado:", num1 > num2) -- ">" retorna true se num1(6) for maior que num2(4)
11 print ("Resultado:", num1 < num2) -- "<" retorna true se num1(6) for menor que num2(4)
12 print ("Resultado:", num1 >= num2) -- ">=" retorna true se num1(6) for maior ou igual a num2(4)
13 print ("Resultado:", 5 > 5) -- outro exemplo do uso do ">="
14 print ("Resultado:", num1 <= num2) -- "<=" retorna true se num1(6) for menor ou igual a num2(4)
15 print ("Resultado:", 5 == 5) -- "==" retorna true se os valores forem exatamente iguais
16 print ("Resultado:", 6 ~= 5) -- "~=" retorna true se os valores forem diferentes
17
18
```

```
Resultado: true
Resultado: false
Resultado: true
Resultado: true
Resultado: false
Resultado: true
Resultado: true
(program exited with code: 0)
Pressione qualquer tecla para continuar. . .
```

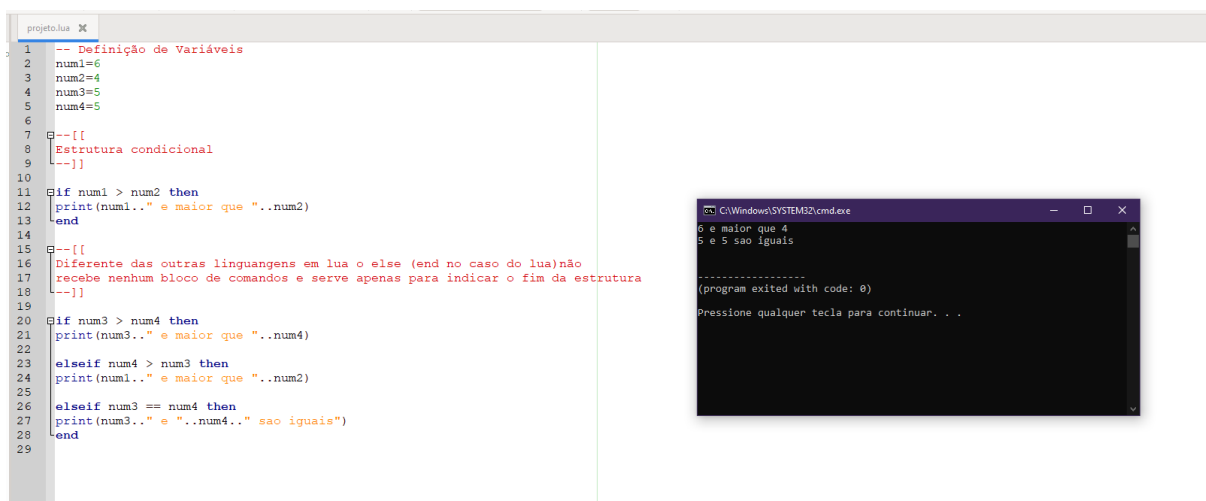
Estruturas condicionais

As estruturas condicionais em Lua são comandos que dependem de uma condição específica para serem executados e compilados pelo programa.

If-then: A estrutura "if-then" é semelhante ao "if-then-else", mas não inclui um bloco de código para ser executado se a condição for falsa.

If-elseif-then: A estrutura "if-elseif then" permite testar várias condições em sequência e executar o bloco de código correspondente à primeira condição verdadeira.

A sintaxe é a seguinte:



The image shows a code editor window titled 'projeto.lua' on the left and a command prompt window on the right. The code in the editor defines variables num1=6, num2=4, num3=5, and num4=5. It uses an 'if' statement to print '6 e maior que 4' and an 'elseif' chain to print '5 e 5 sao iguais'. The command prompt shows the output of these print statements and a message '(program exited with code: 0)'. The code in the editor is as follows:

```
1 -- Definição de Variáveis
2 num1=6
3 num2=4
4 num3=5
5 num4=5
6
7
8 Estrutura condicional
9 ---]]
10
11 if num1 > num2 then
12 print(num1.." e maior que "..num2)
13 end
14
15
16 Diferente das outras linguagens em lua o else (end no caso do lua) não
17 recebe nenhum bloco de comandos e serve apenas para indicar o fim da estrutura
18 ---]]
19
20 if num3 > num4 then
21 print(num3.." e maior que "..num4)
22
23 elseif num4 > num3 then
24 print(num1.." e maior que "..num2)
25
26 elseif num3 == num4 then
27 print(num3.." e "..num4.." sao iguais")
28 end
29
```

The command prompt output is:

```
C:\Windows\SYSTEM32\cmd.exe
6 e maior que 4
5 e 5 sao iguais
-----
(program exited with code: 0)
Pressione qualquer tecla para continuar. . .
```

Laços de repetição

Laços de repetição, também conhecidos como "loops", são estruturas de controle de fluxo que permitem executar um bloco de código várias vezes, até que uma determinada condição seja atendida.

O loop "while" é usado para executar um bloco de código enquanto uma determinada condição for verdadeira.

O loop "repeat" executa o bloco de código pelo menos uma vez e, em seguida, verifica a condição especificada pelo "until". Se a condição for verdadeira, o loop é encerrado. Caso contrário, o bloco de código é executado novamente e o processo é repetido até que a condição seja verdadeira.


```

projeto.lua x
1  -- Definição de Variáveis
2  num1=0
3  num2=0
4
5  --[[
6  Laços de repetição
7  --]]
8
9  --Loop Infinito pois não tem um incrementador para o num1
10 while num1< 10 do
11   print (num1)
12 end
13
14 -- Loop Finito com incrementação
15 num1=0
16 while num1 <10 do
17   print (num1)
18   num1 = num1 + 1
19 end
20
21 -- Loop Infinito com Repeat, tambem por falta da incrementação
22 repeat
23   print(num2)
24 until num2 >10
25
26
27 -- Loop Finito com Repeat
28 num2=0
29 repeat
30   print(num2)
31   num2 = num2 + 1
32 until num2

```

Primeiro Projeto

Para colocar o que foi aprendido em prática será feito um exercício.

O exercício consiste em criar um programa que solicita ao usuário a medida da altura e largura de um retângulo calculando assim a área. Para isso, o programa deve ler as medidas digitadas pelo usuário, calcular a área do retângulo (altura x largura) e exibir o resultado para o usuário.

```

projeto.lua x
1  --[[
2  Como calcular a area de um quadrado a partir dos dados
3  enviados pelo usuario
4  --]]
5
6  print("Digite a base do quadrado:")
7  base=io.read("*number")
8  print("Digite a altura do quadrado:")
9  altura=io.read("*number")
10 area = base * altura
11
12 print ("A area desse quadrado e: ".. area)
13
14
15
16

```

```

C:\Windows\SYSTEM32\cmd.exe
Digite a base do quadrado:
5
Digite a altura do quadrado:
10
A area desse quadrado e:50
-----
(program exited with code: 0)
Pressione qualquer tecla para continuar. . .

```

TIBIA

Tibia é um jogo online de MMORPG (Massively Multiplayer Online Role-Playing Game) que foi lançado em 1997 e durante um certo tempo foi um dos jogos mais populares do gênero. Com um vasto mundo de fantasia, repleto de criaturas míticas, magia e mistério, Tibia oferece aos jogadores a oportunidade de viver aventuras emocionantes em um universo rico e detalhado.

Os jogadores podem escolher entre quatro classes de personagens, cada uma com suas habilidades e pontos fortes, e trabalhar em equipe com outros jogadores para enfrentar desafios e derrotar inimigos poderosos.

Tibia também é conhecido por sua comunidade ativa de jogadores, com fóruns e bate-papo em tempo real que permitem aos jogadores compartilhar suas experiências e dicas, além de fazer novas amizades.

A escolha da linguagem Lua para o desenvolvimento do bot de Tibia foi feita por sua eficiência, flexibilidade e facilidade de integração com outras linguagens de programação. Lua é uma linguagem leve e rápida, ideal para jogos online, e permite a criação de scripts e extensões personalizadas que podem ser adicionados ao jogo de forma dinâmica e sem interromper o funcionamento do servidor.

Nosso projeto visa confeccionar, realizar a programação de um bot de forma que seja possível se interligar com um cliente alternativo do jogo, que foi desenvolvido por fãs do jogo, e possui o código aberto e alterável.

O intuito com a criação desse bot será, principalmente, automatizar as mecânicas já existentes dentro do jogo, que por conta de ser um jogo de turnos demanda tempo até mesmo para realizar as ações mais básicas, com isso, temos em mente automatizar essas ações(coleta de itens, interação com mobs, movimentação, uso de spells, etc) para que torne a gameplay mais fluida, consuma menos tempo e seja mais prática para quem joga.

BOT

O objetivo deste projeto será um BOT programado utilizando a linguagem Lua, que tornará possível realizar ações de cura, reposição de mana, e a utilização de magias utilitárias dentro do jogo Tibia, a fim de tornar a forma de jogar automatizada e mais simples.

Com isso em mente, é ideal que seja entendido como cada elemento presente no BOT será confeccionado, de forma progressiva e detalhada, começando com:

Macros

Os macros são entendidos como uma sequência de comandos de um aplicativo ou como um conjunto de instruções de uma linguagem de programação, podendo armazená-los tanto em disco, quanto na memória como entes independentes que, quando solicitados, executam os comandos na exata sequência que foram armazenados previamente.

Scripts

Os scripts são uma sequência de comandos ou um conjunto de instruções, similar ao macro, porém, os scripts não são armazenados na memória.

DESENVOLVIMENTO

O bot irá englobar grande parte da jogabilidade do jogo, e para isso será necessário a divisão em três partes essenciais, sendo elas:

- A aba principal - Responsável por realizar a cura, pela utilização de magias secundárias e outras utilidades.
- A aba secundária - Responsável por toda a parte relacionada aos ataques dentro do jogo.
- A terceira aba - Será o setor responsável por realizar a movimentação do personagem dentro do jogo.

CASE 01

Editor de Macros

O editor de macros é uma ferramenta que deve ser utilizada para que o jogador possa adicionar e configurar novas funcionalidades para dentro de jogo, por meio do uso de macros.

A primeira linha do código utiliza um método para definir uma aba a ser selecionada dentro do jogo, utilizando da biblioteca interna dentro do mesmo.

setDefaultTab("Main")

No próximo bloco de comandos, será utilizado um método para criação de um botão, onde por meio da passagem de parâmetros irá ser atribuído o texto explicitado no botão, juntamente com a definição da sua funcionalidade.

Dentro dessa função, será utilizado mais um método presente no cliente do jogo, responsável por abrir uma "janela de edição" para que seja possível definir os parâmetros para:

- Retornar e armazenar os macros que possam ser adicionados ao BOT
- O título da janela a ser aberta
- E a descrição da janela a ser aberta.

Por meio desta, também será adicionada mais uma função como parâmetro a ser seguido, onde dentro dessa função haverá a passagem de um parâmetro de texto, ou string, e nesse texto estará presente o valor do que está sendo exibido na janela de edição, e por fim haverá um comando para "recarregar" a janela.

```
3 UI.Button("Editor de macros em jogo", function(NovoTexto)
4   UI.MultilineEditorWindow(storage.ingame_macros or "",
5     {title="Editor de macros", description="descricao"}, function(texto)
6     storage.ingame_macros = texto
7     reload()
8   end)
9 end)
10
```

Em sequência, no próximo bloco de comandos é utilizado um método para criação de um botão, onde por meio da passagem de parâmetros irá ser atribuído o texto explicitado no botão, juntamente com a definição da sua funcionalidade.

Após isso, deve ser criado um **for**, utilizando um **in ipairs** para que o programa execute ou tente executar a lista de comandos de forma sequencial, onde dentro deste, haverá um **if** com intuito de verificar se o texto executado é ou não uma string, ao mesmo tempo que analisa o tamanho da mesma.

Dessa forma, caso as condições do **if** sejam atendidas, serão criadas duas variáveis, sendo uma de status e a outra de resultado.

Caso as condições do **if** não sejam atendidas, aparecerá uma mensagem de erro.

```
for _, scripts in ipairs({storage.ingame_macros}) do
  if type(scripts) == "string" and scripts:len() > 3 then
    local status, result = pcall(function()
      assert(load(scripts, "Editor"))()
    end)
    if not status then
      error("Erro no editor de macros\n" .. result)
    end
  end
end
```

Magias de cura

Esse será o setor responsável por desenvolver as ações que irão curar o personagem.

Para tal, deve-se utilizar dois métodos, sendo um deles para separar o editor de macros dessa nova seção, e o outro para criar uma **label**.

UI.Separator()

UI.Label("Magias de cura")

Após isso, deve ser criado dois **if** para verificar se o tipo da variável que estará armazenada será de um tipo específico.

```
if type (storage.cura1) ~= "table" then
  storage.cura1 = {on=false, title="HP%", text="", min=0, max=100}
end
if type (storage.cura2) ~= "table" then
  storage.cura2 = {on=false, title="HP%", text="", min=0, max=100}
end
```

Feito isso, deve-se criar um **for** utilizando para declarar uma variável que armazenará as informações sobre a cura, e nesse mesmo código passará como parâmetro as duas variáveis de armazenamento feitas anteriormente, além de ser necessário criar uma variável local responsável pelo MACRO de cura, já que inserido neste código será criado o MACRO de cura.

```
for _, curaInfo in ipairs ({storage.cura1, storage.cura2}) do
  local curaMacro = macro(20, function()
```

Dentro desse macro, deverá haver uma função, na qual estará presente uma variável responsável por representar a % de vida do jogador, e para isso será utilizado o método interno **getHealthPercent()**.

```
local curaMacro = macro(20, function()
  local hp = player:getHealthPercent()
```

Para que essa função seja executada, deve haver um **if** para verificar duas condições: o máximo de vida ser, ou não, maior que o HP do personagem e o mínimo de vida ser, ou não, menor que o HP do personagem.

```
local curaMacro = macro(20, function()
  local hp = player:getHealthPercent()
  if curaInfo.max >= hp and hp >= curaInfo.min then
```

Com isso, dentro desse **if** deverá ser criado um novo **if** para selecionar um objeto interno de dentro do cliente do jogo, onde será utilizado o método **saySpell()** que terá como parâmetro o texto armazenado pelas variáveis de armazenamento, além de ser preciso o uso de um **else** junto do método **say()** para passar o parâmetro de texto das variáveis de armazenamento.

Em resumo, a totalidade desse bloco de comandos ficará assim:

```
for _, curaInfo in ipairs ({storage.cura1, storage.cura2}) do
  local curaMacro = macro(20, function()
    local hp = player:getHealthPercent()
    if curaInfo.max >= hp and hp >= curaInfo.min then
      if TargetBot then
        TargetBot.saySpell(curaInfo.text)
      else
        say(curaInfo.text)
      end
    end
  end
end)
```

Poções e runas de cura

Finalizando isso, pode-se seguir para a segunda parte do MACRO, utilizando o método **setOn()** para realizar a ativação da função.

setOn(curaInfo.on)

Dessa forma, deve se utilizar outro método interno para criação de dois **scrolls**. O **UI.DualScrollPainel()**, que servirão para utilizar uma função com os parâmetros de mínimo ou máximo de HP.

```
curaMacro.setOn(curaInfo.on)
UI.DualScrollPainel(curaInfo, function(widget, novoParametro)
  curaInfo = novoParametro
  curaMacro.setOn(curaInfo.on)
end)
end
```

Após, é criada uma nova separação e dado um título para ela utilizando métodos internos do **Client**.

UI.Separator() UI.Label("pots/runas de cura")

Em seguida, pode-se repetir a criação das variáveis de armazenamento que foram feitas na parte inicial do bot, mas dessa vez será para armazenar itens que servirão para curar vida e mana.

```
if type (storage.hpItem1) ~= "table" then
  storage.hpItem1 = {on=false, title="HP%", item=266, min=0, max=100}
end

if type (storage.hpItem2) ~= "table" then
  storage.hpItem2 = {on=false, title="HP%", item=3160, min=0, max=100}
end

if type (storage.manaItem1) ~= "table" then
  storage.manaItem1 = {on=false, title="MP%", item=268, min=0, max=100}
end

if type (storage.manaItem2) ~= "table" then
  storage.manaItem2 = {on=false, title="MP%", item=3157, min=0, max=100}
end
```

A seguir, cria-se um **for** semelhante ao anterior com uso da variável de informação de cura e utilizando o **in ipairs**, onde também é usado como parâmetro as variáveis de armazenamento criadas anteriormente.

Dentro desse **for**, cria-se uma variável para o macro de cura, na qual é passado o valor de uma função, onde dentro desta é inaugurada uma outra variável local que irá armazenar o valor da variável de controle do **for** e o valor da porcentagem de vida do jogador utilizando o método interno **getHealthPercent()**.

Aqui é trabalhado um operador lógico **or**, onde é passado um método matemático **math.min()**, e dentro dele passa-se como parâmetro o valor "100" que remete à porcentagem de mana. Também é passado o método matemático **math.floor()** e dentro dele será passado novamente o valor "100" remetendo à porcentagem de mana, e então, é feita uma multiplicação desse valor pela junção de dois valores que são retornados outros dois métodos internos, o **getMana()** dividido pelo **getMaxMana()**.

```
for ii, curaInfo in ipairs({storage.hpItem1, storage.hpItem2, storage.manaItem1, storage.manaItem2}) do
  local curaMacro = macro(20, function()
    local hp = ii <= 2 and player:getHealthPercent() or math.min(100, math.floor(100 * (player:getMana() / player:getMaxMana())))
```

Agora deve-se elaborar um **if**, onde dentro dele serão feitas duas comparações com um operador lógico. Na primeira é utilizada a variável de informação de cura e o valor máximo definido pelo jogador com intuito de comparar se esse valor é maior ou igual ao valor da variável de HP, já a segunda comparação, checa se essa variável de HP será maior ou igual a variável de informação de cura no valor mínimo definido.

É gerado um novo **if**, e nele é selecionado um objeto interno do **client**, onde junto a ele, utiliza-se o método interno **useItem()**. Deve-se passar três parâmetros dentro do mesmo, sendo o primeiro uma variável de informação de cura contendo o valor do item, o segundo, utilizando novamente essa variável, porém, contendo um valor do subtipo dela, e por fim, utilizando mais um objeto interno do **client**.

```
if curaInfo.max >= hp and hp >= curaInfo.min then
  if TargetBot then
    TargetBot.useItem(curaInfo.item, curaInfo.subType, player)
```

Agora, é definido um **else**, e nele criado duas variáveis locais onde a primeira recebe o valor através do método **g_things.getThingType()**, nesse método é passado o parâmetro com a variável que contém as informações de cura do tipo do item que será armazenado pelo jogador. Após isso, é criada uma segunda variável que irá verificar através de um método interno a versão do **client** do jogo (isso é necessário para que o script consiga identificar que o id do item que será passado esteja relacionado ao item certo).

```
local thing = g_things.getThingType(curaInfo.item)
local subType = g_game.getClientVersion() >= 860 and 0 or 1
```

Finalizando essa parte, cria-se mais um **if**, e dentro dele deve-se passar a primeira variável local criada e usar o operador lógico **and**. Após isso, é passada novamente essa mesma variável com um método interno **isFluidContainer()**.

Dentro desse novo **if** deve-se pegar a segunda variável definida e igualar ela ao subtipo da variável que contém as informações de cura.

Fora do terceiro **if** utiliza-se o método **useInventoryItemWith()** que servirá para utilizar os itens de cura, nele são passados três parâmetros que serão: o item da variável de informação de cura; o objeto interno do client; e a segunda variável local definida anteriormente.

```
        if thing and thing.isFluidContainer() then
            subType = curaInfo.subType
        end
        g_game.useInventoryItemWith(curaInfo.item, player, subType)
    end
end
end)
curaMacro.setOn(curaInfo.on)
```

Fora dessa função, é colocada a variável que foi definida para o macro de cura, utilizaremos nela o método interno **setOn()** e dessa vez utilizando a variável de informação de cura para passar como parâmetro para esse método. Além disso, deve-se gerar os **scrolls** de controle exatamente iguais aos gerados previamente. Por fim, cria-se um **if** que servirá para verificar novamente a versão **client** do jogo e dentro dele é colocada uma **label** de aviso que servirá para alertar o jogador de que ele precisará manter sua mochila aberta para as poções funcionarem.

```
curaMacro.setOn(curaInfo.on)

UI.DualScrollItemPanel(curaInfo, function(widget, novoParametro)
    curaInfo = novoParametro
    curaMacro.setOn(curaInfo.on and curaInfo.item > 100)
end)
end

if g_game.getClientVersion() < 780 then
    UI.Label("Abra sua backpack para pocoes e runas funcionarem")
end
```

Em sua totalidade, o bloco de comandos ficará da seguinte forma:

```
for ii, curaInfo in ipairs({storage.hpItem1, storage.hpItem2, storage.manaItem1, storage.manaItem2}) do
  local curaMacro = macro(20, function()
    local hp = ii <= 2 and player:getHealthPercent() or math.min(100, math.floor(100 * (player:getMana() / player:getMaxMana())))
    if curaInfo.max >= hp and hp >= curaInfo.min then
      if TargetBot then
        TargetBot.useItem(curaInfo.item, curaInfo.subType, player)
      else
        local thing = g_things.getThingType(curaInfo.item)
        local subType = g_game.getClientVersion() >= 860 and 0 or 1
        if thing and thing:isFluidContainer() then
          subType = curaInfo.subType
        end
        g_game.useInventoryItemWith(curaInfo.item, player, subType)
      end
    end
  end)
  curaMacro.setOn(curaInfo.on)
  UI.DualScrollItemPanel(curaInfo, function(widget, novoParametro)
    curaInfo = novoParametro
    curaMacro.setOn(curaInfo.on and curaInfo.item > 100)
  end)
end

if g_game.getClientVersion() < 780 then
  UI.Label("Abra sua backpack para pocoes e runas funcionarem")
end
```

Magias Utilitárias

Novamente, é feita uma nova separação na aba do bot usando o método interno **UI.Separator()**, onde dentro dessa nova aba coloca-se algumas spells utilitárias.

Deve-se colocar uma **label** que irá conter o título da magia do escudo de mana.

É então utilizado um método interno **UI.TextEdit()** que servirá para criar uma caixa de texto, dentro desse método passará dois parâmetros, sendo o primeiro uma variável de armazenamento para o escudo de mana ou o texto "utamo vita", além disso, será passado uma função e dentro desta função serão passados outros dois parâmetros, o primeiro é um objeto interno e o segundo é uma variável para armazenar texto. Dentro dessa função, a variável de armazenamento do escudo de mana e a variável de texto serão igualadas.

```
UI.Separator()

UI.Label("Mana Shield Spell:")
UI.TextEdit(storage.manaShield or "utamo vita", function(widget, novoTexto)
  storage.manaShield = novoTexto
end)
```

Fora da função, será definida uma variável local que terá a função de cronômetro.

É criado um macro no qual serão passados três parâmetros: um timer, uma string e uma função. Dentro dessa função, será criado um **if** no qual será utilizado um método interno **hasManaShield()** e a variável definida como cronômetro, com o operador lógico **or**.

O método interno verificará se o jogador já possui o escudo de mana ativado. Caso contrário, o valor numérico de 90 mil será atribuído à variável de cronômetro.

```
local lastManaShield = 0
macro(20, "manaShield", function()
  if hasManaShield() or lastManaShield + 90000 > now then return end
```

Um segundo **if** será criado, no qual um objeto interno do **client** será utilizado e o método interno **saySpell()** será passado, juntamente com a variável de armazenamento do escudo de mana. Após isso, o **else** será definido e dentro dele o método interno **say()** será utilizado, passando a variável de armazenamento do escudo de mana.

```
  if hasManaShield() or lastManaShield + 90000 > now then return end
  if TargetBot then
    TargetBot.saySpell(storage.manaShield)
  else
    say(storage.manaShield)
  end
end)
```

Esse bloco de comandos ficará da seguinte da forma:

```
UI.Separator()

UI.Label("Mana Shield Spell:")
UI.TextEdit(storage.manaShield or "utamo vita", function(widget, novoTexto)
  storage.manaShield = novoTexto
end)

local lastManaShield = 0
macro(20, "manaShield", function()
  if hasManaShield() or lastManaShield + 90000 > now then return end
  if TargetBot then
    TargetBot.saySpell(storage.manaShield)
  else
    say(storage.manaShield)
  end
end)
```

Em seguida, uma label será colocada para conter o título da magia de velocidade.

O método interno **UI.TextEdit()** será utilizado para criar uma caixa de texto e dentro desse método dois parâmetros serão passados: uma variável de armazenamento para a magia de velocidade ou o texto "utani hur" e uma função. Dentro dessa função, outros dois parâmetros serão passados: um objeto interno e uma variável para armazenar texto. Dentro dessa função, a variável de armazenamento da magia de velocidade será igualada à variável de texto.

```
UI.Label("Haste spell:")
UI.TextEdit(storage.hasteSpell or "utani hur", function(widget, novoTexto)
    storage.hasteSpell = novoTexto
end)
```

Um macro será criado e três parâmetros serão passados dentro dele: um timer, uma string e uma função. Dentro dessa função, será criado um **if** no qual será utilizado o método **hasHaste()**, que verificará se o jogador já possui a magia de velocidade ativada ou não.

```
macro(500, "haste", function()
    if hasHaste() then return end
```

É criado então um segundo **if**, onde ele deve ser utilizado um objeto interno do **client** e passado o método interno **saySpell()**, e nele agindo com a variável de armazenamento da magia de velocidade. Após isso, deve-se definir o **else** do **if**, e dentro dele utilizar o método interno **say()**, nele passando a variável de armazenamento do feitiço.

```
    if TargetBot then
        TargetBot.saySpell(storage.hasteSpell)
    else
        say(storage.hasteSpell)
    end
end)
```

Esse bloco de comandos em sua totalidade ficará assim:

```
UI.Label("Haste spell:")
UI.TextEdit(storage.hasteSpell or "utani hur", function(widget, novoTexto)
    storage.hasteSpell = novoTexto
end)

macro(500, "haste", function()
    if hasHaste() then return end
    if TargetBot then
        TargetBot.saySpell(storage.hasteSpell)
    else
        say(storage.hasteSpell)
    end
end)
```

Uma label será colocada para conter o título da magia de anti paralisia.

O método interno **UI.TextEdit()** será utilizado para criar uma caixa de texto e dentro desse método dois parâmetros serão passados: uma variável de armazenamento para a magia de anti paralisia ou o texto "utani hur" e uma função. Dentro dessa função, outros dois parâmetros serão passados: um objeto interno e uma variável para armazenar texto. Dentro dessa função, a variável de armazenamento da magia de anti paralisia será igualada à variável de texto.

```
UI.Label("Anti paralyze spell:")
UI.TextEdit(storage.antiParalyze or "utani hur", function(widget, novoTexto)
    storage.antiParalyze = novoTexto
end)
```

Um macro será criado e três parâmetros serão passados dentro dele: um timer, uma string e uma função. Dentro da função um **if** juntamente com um **not** será criado e o método **isParalyzed()** será utilizado para verificar se o jogador já está com a magia de anti paralisia ativada ou não.

```
macro(100, "anti paralyze", function()
    if not isParalyzed() then return end
```

Cria-se então um segundo **if**, nele sendo aplicado um objeto interno do **client**, passando o método interno **saySpell()** e nele deve-se passar a variável de armazenamento da magia de anti paralisia. Dentro deste **if** utilizado o método interno **say()**, no qual passa-se a variável de armazenamento do feitiço.

```
    if TargetBot then
        TargetBot.saySpell(storage.antiParalyze)
        say(storage.antiParalyze)
    end
end)
```

Em sua totalidade o bloco de comandos ficará da seguinte forma:

```
UI.Label("Anti paralyze spell:")
UI.TextEdit(storage.antiParalyze or "utani hur", function(widget, novoTexto)
    storage.antiParalyze = novoTexto
end)

macro(100, "anti paralyze", function()
    if not isParalyzed() then return end
    if TargetBot then
        TargetBot.saySpell(storage.antiParalyze)
        say(storage.antiParalyze)
    end
end)
```

Em seguida, será criada uma **label** que servirá de título da nova sessão do bot. Nessa sessão será criada uma variável de armazenamento semelhante às citadas anteriormente, com o intuito de armazenar o id das comidas definidas pelo jogador.

```
if type(storage.comidas) ~= "table" then
    storage.comidas = {3582, 3577}
end
```

A variável local será definida em seguida, essa variável tem a função de armazenar as comidas. Para definir essa variável utilizaremos um método interno do **client**, o **UI.Container()**, onde será passado uma função como parâmetro. Essa função terá dois parâmetros que serão dois objetos internos do **client**. Dentro da função a variável de armazenamento será igualada ao parâmetro de itens da função.

```
local mochilaComidas = UI.Container(function(widget, Items)
    storage.comidas = Items
end, true)
```

Fora da função é utilizado a variável local para operar dois métodos separadamente. O primeiro método tem como objetivo definir o tamanho da variável utilizando o **setHeight()**, onde será passado um valor numérico como parâmetro, sendo o número utilizado 35. O segundo método deve utilizar o **setItems()**, onde o parâmetro será a variável de armazenamento.

```
mochilaComidas:setHeight(35)
mochilaComidas:setItems(storage.comidas)
```

Finalizando os passos anteriores, é criado um novo macro, que receberá três parâmetros: Um timer, uma string e uma função que irá delimitar a quantidade de tempo para consumo da "comida".

Dentro dessa função criada, é então posicionado um **if not** que servirá para passar a variável de armazenamento das comidas, agindo como se fosse um vetor na primeira posição.

Tendo realizado os passos anteriores é criado então um **for**, onde dentro dele deve ser declarado uma variável "mochila", utilizando o método **pairs()**, e como parâmetro para este é utilizado o comando **g_game.getContainers()**.

Dentro dessa linha de comando, é criado novamente outro **for**, que utilizará o método **ipairs()**, onde como parâmetro será utilizada a variável “mochila” criada anteriormente juntamente do método interno **getItems()**.

```
macro(10000, "Alimentar-se", function()  
  if not storage.comidas[1] then return end  
  for _, mochila in pairs(g_game.getContainers()) do  
    for _, item in ipairs(mochila:getItems()) do
```

Ainda trabalhando dentro desse segundo laço de repetição, é feito por fim um último **for**, onde será criada uma variável “comidas”, utilizando o método **ipairs()**, no qual será passado como parâmetro a variável de armazenamento de comidas.

Ainda nesta estrutura de repetição, é criado então um **if**, utilizando a variável “item” feita previamente com uso do método **getId()**, no qual deve-se verificar se o valor retornado será igual ao ID das comidas, onde caso essa condição seja verdadeira, o comando **g_game.use(item)** deve ser retornado.

```
    for jj, comidas in ipairs(storage.comidas) do  
      if item:getId() == comidas.id then  
        return g_game.use(item)
```

Fora das estruturas de repetição anteriores, é arquitetado um **if** que deve passar uma instrução para verificar a versão do **client** do jogo.

```
if g_game.getClientVersion() < 780 then return end
```

Feito isso, é então criada uma variável local que deve receber o valor da variável de armazenamento de comidas, fazendo uso do método **math.random()**, é passado como parâmetro o valor 1 juntamente com a variável de armazenamento de comida. Feitos os passos anteriores, é esquematizado mais um **if**, no qual é utilizado a variável criada anteriormente, também utilizando o comando: **g_game.useInventoryItem(comer.id)**.

```
  if g_game.getClientVersion() < 780 then return end  
  local comer = storage.comidas[math.random(1, #storage.comidas)]  
  if comer then g_game.useInventoryItem(comer.id) end  
end)
```

Em suma, a totalidade das linhas do comando:

```
if type(storage.comidas) ~= "table" then
    storage.comidas = {3582, 3577}
end

local mochilaComidas = UI.Container(function(widget, Items)
    storage.comidas = Items
    end, true)
mochilaComidas:setHeight(35)
mochilaComidas:setItems(storage.comidas)

macro(10000, "Alimentar-se", function()
    if not storage.comidas[1] then return end
    for _, mochila in pairs(g_game.getContainers()) do
        for _, item in ipairs(mochila:getItems()) do
            for jj, comidas in ipairs(storage.comidas) do
                if item:getId() == comidas.id then
                    return g_game.use(item)
                end
            end
        end
    end
    if g_game.getClientVersion() < 780 then return end
    local comer = storage.comidas[math.random(1, #storage.comidas)]
    if comer then g_game.useInventoryItem(comer.id) end
end)
```

CASE 02

Iniciando o TargetBot

O target é a parte do bot que será responsável por alvejar os inimigos dentro do jogo de forma automática.

Primeiramente, é necessário criar um arquivo que carregue os arquivos necessários para o Target Bot, juntamente com a criação de uma pasta para os arquivos do Target Bot, incluindo um arquivo .lua novo, que pode ser renomeado como por exemplo: "criatura.lua", que contará com informações sobre os inimigos que serão encontrados dentro do jogo.

Após isso, ainda dentro desse arquivo recém criado, devemos definir três valores dentro dos seguintes objetos:

```
TargetBot.Creature = {}
TargetBot.Creature.configsCache = {}
TargetBot.Creature.cached = 0
```

Feito isso, é definido uma função dentro do objeto que será utilizado para redefinir as configurações do Target Bot. Dentro desta função, devem ser criados dois métodos, sendo um deles uma lista de de elementos relacionados ao Target

Bot, tendo como intuito alvejar e destruir todos os elementos citados pela lista que posteriormente será criada.

Após o passo anterior, é necessário a criação de uma função para a redefinição de cache temporário das criaturas citadas para o estado inicial.

```
TargetBot.Creature.resetConfigs = function()
    TargetBot.targetList:destroyChildren()
    TargetBot.Creature.resetConfigsCache()
end
```

Em sequência, deve-se fazer parecido como nos passos anteriores, porém agora para a manutenção do cache do Target Bot, para realizar o controle da memória temporária que será utilizada.

```
TargetBot.Creature.resetConfigsCache = function()
    TargetBot.Creature.configsCache = {}
    TargetBot.Creature.cached = 0
end
```

Após isso, é criada uma nova função para adicionar uma variável de configuração ao bot de target de uma criatura, utilizando o comando focus como parâmetro para a utilização.

Logo, para realizar a iniciação, deve-se fazer uma verificação utilizando as funções **type()**, para que seja possível identificar o tipo dos elementos presentes.

Caso a verificação falhe, deve ser exibido uma mensagem de erro, juntamente com a interrupção da execução da função. Porém, caso a verificação seja bem sucedida, será feito um retorno para a função **TargetBot.Creature.resetConfigsCache()** para realizar a atualização ou redefinição do cache de configuração de uma criatura.

```
TargetBot.Creature.addConfig = function(config, focus)
    if type(config) ~= 'table' or type(config.name) ~= 'string' then
        return error("Configuração invalida")
    end
    TargetBot.Creature.resetConfigsCache()
```

Feito o passo anterior, é criado então uma estrutura condicional utilizando **if** para checar se o propriedade **regex**, o método **regex** é utilizado para identificar uma série de caracteres, está ou não ativo nas configurações pré-estabelecidas anteriormente.

Em sequência, é criado um **for** para percorrer todas as partes da string criada anteriormente utilizando a função **string.gmatch()**, onde dentro dessa função são passados dois parâmetros: **config.name** e **"[^,]+"**.

```
if not config.regex then
    config.regex = ""
    for part in string.gmatch(config.name, "[^,]+") do
```

Após isso, é necessário criar um **if** para verificação da existência de algum padrão pré-definido da propriedade **regex**, através do tamanho do mesmo.

Ainda dentro do **if**, é relacionado a variável com ela mesma, para realizar a ligação com o caracter “|” para fazer a separação dos padrões dessa propriedade.

```
if config.regex:len() > 0 then
    config.regex = config.regex .. "|"
end
```

Dessa forma, devem se definir os padrões da propriedade citada acima, utilizando o mesmo processo feito anteriormente, porém dessa vez com o “^” para definir qual será o início da definição dos padrões.

Para tal, deve-se utilizar o comando **part** para realizar a leitura por partes da string de forma sequencial, juntamente do uso do método **trim()**, que remove os espaços em branco da string, em adição ao método **lower()**, que converte todo o texto para letra minúscula, e por fim é feito uso do método **gsub()**, que substituirá caracteres especiais pelos caracteres correspondentes especificados: “%*”, “.*”.

Deve-se utilizar o **gsub()** novamente, para realizar a mesma ação, porém agora substituindo os caracteres especiais pelos seguintes: “%?”, “.?”.

E por fim, para terminar o loop do padrão **regex**, devemos concatenar essa expressão inteira com um “\$” para indicar o final dessa definição de padrões.

```
config.regex = config.regex .. "^" .. part:trim():lower():gsub("%*", ".*"):gsub("%?", ".?") .. "$"
```

Em sua totalidade, o bloco deve ficar da seguinte forma:

```
if not config.regex then
    config.regex = ""
    for part in string.gmatch(config.name, "[^,]+") do
        if config.regex:len() > 0 then
            config.regex = config.regex .. "|"
        end
        config.regex = config.regex .. "^" .. part:trim():lower():gsub("%*", ".*"):gsub("%?", ".?") .. "$"
    end
end
```

O próximo passo a ser seguido é criar uma variável local do tipo “widget”, tendo seu valor atribuído a uma método interno do **cliente(UI.createWidget())**, onde serão passados dois parâmetros, um texto “TargetBotEntry” e a lista de destino.

Será usado um método **setText()** na variável local “widget”, o parâmetro desse método é a variável que conterá a propriedade de nome da configuração. Novamente a variável “widget” será utilizada para definir a sua propriedade value, o mesmo será igual a variável de configuração.

```
local widget = UI.createWidget("TargetBotEntry", TargetBot.targetList)
widget.setText(config.name)
widget.value = config
```

Em seguida será definido um evento de clique para a variável “widget” onde será passada uma função para esse evento. A função terá um parâmetro que conterá uma variável de entrada e uma variável interna.

Dentro desta função será agendado a execução de uma outra função, tendo um atraso de 20 milissegundos.

```
widget.onDoubleClick = function(entry)
    schedule(20, function()
```

Na segunda função será usado um método **edit()** junto do objeto **TargetBot.Creature**, tendo como parâmetro a variável de entrada juntamente com a sua propriedade value. Outra função será passada como parâmetro usando uma nova variável com o objetivo de ter novas configurações.

A variável de entrada junto ao método **setText()** para definir um novo valor, sendo ele a variável para novas configurações com sua propriedade name. A variável de entrada será usada novamente, atribuindo-se à sua propriedade value o valor da variável de novas configurações.

Por fim, será utilizado o método para resetar as configurações de cache do objeto **TargetBot.Creature**, em seguida será usado o método **save()** do objeto **TargetBot**.

```
    TargetBot.Creature.edit(entry.value, function(novaConfig)
        entry.setText(novaConfig.name)
        entry.value = novaConfig
        TargetBot.Creature.resetConfigsCache()
        TargetBot.save()
    end)
end)
end
```

Por último, será criado um **if** que verificará se a variável **focus** tem um valor diferente de nulo, sendo essa condição verdadeira será utilizado um método **focus()** no widget. Será utilizado um método **ensureChildVisible()**, que consta dentro do objeto **TargetBot.targetList**. Em seguida será utilizado a função **return widget**, que retornará o “widget” criado e configurado.

```

if focus then
    widget:focus()
    TargetBot.targetList:ensureChildVisible(widget)
end
return widget

```

Em sua totalidade, o bloco deve ficar da seguinte forma:

```

TargetBot.Creature.addConfig = function(config, focus)
    if type(config) ~= 'table' or type(config.name) ~= 'string' then
        return error("Configuração inválida")
    end
    TargetBot.Creature.resetConfigsCache()

    if not config.regex then
        config.regex = ""
        for part in string.gmatch(config.name, "[^,]+") do
            if config.regex:len() > 0 then
                config.regex = config.regex .. "|"
            end
            config.regex = config.regex .. "^" .. part:trim():lower():gsub("%*", "%"):gsub("%?", "%.") .. "$"
        end
    end

    local widget = UI.createWidget("TargetBotEntry", TargetBot.targetList)
    widget.setText(config.name)
    widget.value = config

    widget.onDoubleClick = function(entry)
        schedule(20, function()
            TargetBot.Creature.edit(entry.value, function(novaConfig)
                entry.setText(novaConfig.name)
                entry.value = novaConfig
                TargetBot.Creature.resetConfigsCache()
                TargetBot.save()
            end)
        end)
    end

    if focus then
        widget:focus()
        TargetBot.targetList:ensureChildVisible(widget)
    end
    return widget
end

```

Em seguida será utilizado o objeto **TargetBot.Creature** e será criada uma função para que seja possível pegar as configurações do bot, dentro desta função haverá uma variável criatura que será passada como parâmetro.

Dentro desta função irá passar um **if not** e a variável “criatura”, o **if** irá retornar em uma tabela vazia caso a condição seja falsa. Depois disso será criada uma variável local que irá armazenar o nome da variável “criatura”, usando o método **getName()** e na formatação do nome será utilizado os métodos **Trim()** e **Lower()**.

```
TargetBot.Creature.getConfigs = function(creature)
    if not creature then return {} end
    local name = creature:getName():trim():lower()
```

Em seguida será criado um **if**, onde será passado o mesmo objeto **TargetBot.Creature**. Em seguida será utilizada a função de configuração da cache criada anteriormente e será passada a variável que armazena o nome da criatura como parâmetro, logo será criado um **return** dentro deste **if** que irá retornar o objeto junto da função.

```
if TargetBot.Creature.configsCache[name] then
    return TargetBot.Creature.configsCache[name]
end
```

A seguir será criada uma variável local que servirá para receber as configurações, essa variável irá receber o valor “{}”, pois deverá ter o valor de uma tabela vazia. Logo após será definido um **for** onde será definido uma segunda variável de configuração e utilizará o comando **in ipairs** e dentro deste **for** será passado uma lista onde será utilizado o método **getChildren()**.

Em seguida será definido um **if**, onde será passada a função **regexMatch()**, a função irá receber como parâmetro a variável “criatura” e em seguida um comando que irá verificar se o nome da criatura corresponde ao padrão **regex** definido anteriormente, dentro deste **if** será passado um **table.insert** e a variável que possui a variável de configuração, será passado também a propriedade **value** da segunda variável de configuração.

```
local configs = {}
for _, config in ipairs(TargetBot.targetList:getChildren()) do
    if regexMatch(name, config.value.regex)[1] then
        table.insert(configs, config.value)
    end
end
```

Em seguida, será criado mais um **if** e será utilizado, novamente, o mesmo objeto e a variável que definimos para cache. Logo após, dentro do **if**, será passado o mesmo objeto mas com a função que será criada para resetar as configurações da cache.

Finalizado este **if**, será utilizado novamente o mesmo objeto e a mesma função de configuração da cache, para essa função será atribuído o valor da variável de configurações, em seguida será utilizado a variável da cache e será incrementado o valor dela em um, logo retornará a variável de configurações.

```
    if TargetBot.Creature.cached > 1000 then
        TargetBot.Creature.resetConfigsCache()
    end
    TargetBot.Creature.configsCache[name] = configs
    TargetBot.Creature.cached = TargetBot.Creature.cached + 1
    return configs
end
```

Em totalidade o bloco de comandos ficará da seguinte maneira:

```
TargetBot.Creature.getConfigs = function(creature)
    if not creature then return {} end
    local name = creature.getName():trim():lower()

    if TargetBot.Creature.configsCache[name] then
        return TargetBot.Creature.configsCache[name]
    end
    local configs = {}
    for _, config in ipairs(TargetBot.targetList:getChildren()) do
        if regexMatch(name, config.value.regex)[1] then
            table.insert(configs, config.value)
        end
    end
    if TargetBot.Creature.cached > 1000 then
        TargetBot.Creature.resetConfigsCache()
    end
    TargetBot.Creature.configsCache[name] = configs
    TargetBot.Creature.cached = TargetBot.Creature.cached + 1
    return configs
end
```

Em seguida será criada uma função que servirá para calcular os parâmetros, dentro desta função será passado dois parâmetros, uma delas é a variável “criatura” que foi criada anteriormente e a outra será uma nova variável que servirá para calcular a distância do personagem até uma determinada criatura.

Dentro desta função será definido quatro variáveis locais, sendo elas, uma variável para configuração, atribuindo a ela o valor do objeto **TargetBot.Creature**, passando também a função para obter as configurações, será utilizado nesta função a variável criatura como parâmetro. A próxima variável servirá para definir o nível de prioridade entre as criaturas e depois uma outra variável irá definir o nível de periculosidade das criaturas, por fim na quarta variável servirá para armazenar a configuração selecionada.

```
TargetBot.Creature.calculaParametros = function(creature, caminho)
    local configs = TargetBot.Creature.getConfigs(creature)
    local prioridade = 0
    local perigo = 0
    local selectedConfig = nil
```

Em seguida será criado um **for** e neste **for** será criado uma segunda variável de configuração, onde será utilizado, novamente, o método **in ipairs** e será passado neste método a primeira variável de configuração que será criada. Dentro deste será criado uma variável local que servirá para configurar a prioridade das criaturas e atribuímos o valor do objeto **TargetBot.Creature** juntamente de uma função que irá calcular a prioridade, essa função será criada posteriormente, e será passado três parâmetros dentro dela, sendo a variável “criatura”, a segunda variável de configuração e variável que servirá para a distância entre o personagem e a criatura.

Será criado um **if** e dentro deste **if** será verificado se esta variável de configuração de prioridade terá um valor maior que a variável de prioridade, caso esta comparação seja verdadeira, esta variável de configuração prioridade irá passar seu valor para a variável de prioridade, após este feito será atribuído o valor do objeto **TargetBot.Creature** junto de uma função que irá calcular o nível de periculosidade, esta função será criada posteriormente, este valor será aderido a variável de mesmo nome. Será atribuído o valor da segunda variável de configuração para variável de configuração selecionada.

```
for _, config in ipairs(configs) do
    local config_prioridade = TargetBot.Creature.calculaPrioridade(creature, config, caminho)
    if config_prioridade > prioridade then
        prioridade = config_prioridade
        perigo = TargetBot.Creature.calculaPerigo(creature, config, caminho)
        selectedConfig = config
    end
end
```

Em seguida será criado o seguinte **return**:

```
return {  
    config = selectedConfig,  
    creature = creature,  
    perigo = perigo,  
    prioridade = prioridade  
}  
end
```

Neste momento será necessário criar a função que irá calcular o nível de periculosidade, para isso será passado o objeto **TargetBot.Creature** juntamente da função com três parâmetros, que devem ser a variável criatura, a segunda variável de configuração, e a variável que detém os valores de distância entre o personagem a criatura.

Em seguida será retornada a segunda variável de configuração juntamente com a variável de periculosidade.

```
TargetBot.Creature.calculaPerigo = function(creature, config, caminho)  
    return config.perigo  
end
```

Programando os Ataques

Será necessário utilizar o arquivo que servirá para carregar os arquivos do bot, dentro deste arquivo será definido uma variável local para criar uma aba dentro do bot, atribuindo-a o título da aba desejado. Em seguida será utilizado a função **setDefaultTab()**, sendo o parâmetro a variável local já criada. Após isso deverá ser criada uma tabela vazia, utilizando em seguida o comando **dofile**, dentro deste comando será passado como argumento o diretório da localização dos arquivos desejados para carregar, não sendo necessário utilizar o diretório desde a pasta raiz.

```
local targetingTab = "Target"  
TargetBot = {}  
dofile("/targetbot/criatura.lua")
```

Logo após deverá ser criado um segundo arquivo .lua que será responsável por gerenciar os ataques às criaturas do jogo. Dentro do arquivo será atribuído uma função ao objeto **TargetBot.Creature**, nesta função serão passados três parâmetros: parametro”; alvos; saquear. Nesta função, será incluído um condicional **if**, no qual a variável “player” será utilizada em conjunto com a função **isWalking()** como condição. Dentro do **if** será utilizado uma variável que

indicará o último passo do jogador, com o valor de uma variável que conterà o momento atual atribuído.

```
TargetBot.Creature.attack = function(parametros, targets, looteando)
  if player:isWalking() then
    lastWalk = now
  end
end
```

Posteriormente será criado duas variáveis locais, sendo a primeira uma variável de configuração, onde receberá o valor da propriedade de configuração da variável “parametro”. A segunda será uma variável “criatura” que será atribuído o valor da propriedade “criatura” da variável “parametro”.

Em seguida será criado um **if** que terá como condição a diferença entre o resultado retornado pela função **g_game.getAttackingCreature()** e a variável “criatura”. Dentro do **if** será utilizado outra função, onde caso a condição anterior seja verdadeira será utilizado a função **g_game.attack()** que será passado como argumento a variável criatura.

```
local config = parametros.config
local creature = parametros.creature

if g_game.getAttackingCreature() ~= creature then
  g_game.attack(creature)
end
```

Em seguida, será criado um **if not**, sendo a condição a variável “saquear”. Dentro do **if** será passado o objeto **TargetBot.Creature**, junto com esse objeto será utilizado uma função **walk()**, que será criada posteriormente. Na função **walk()** serão passados três parâmetros: a variável “criatura”; a variável “configuracao”; a variável “alvos”.

```
if not looteando then
  TargetBot.Creature.walk(creature, config, targets)
end
```

Logo após será criado uma variável local que conterà o valor de mana do jogador. Esta variável local terá como atribuição o valor da variável “player” junto da função **getMana()**.

Em seguida, será criado um **if**, onde passarão três condições. A primeira condição terá o objetivo de ver se a propriedade de usar ataques em grupo, da variável de configuração, está ativa, em seguida será utilizado um operador lógico **and**. A segunda condição verificará se o tamanho da propriedade de ataque em grupo da variável de configuração é maior que um, em seguida será utilizado novamente o operador lógico **and**. A terceira condição verificará se a variável “mana” terá um valor maior do que a propriedade de mana mínima para o ataque em grupo, da variável de configuração.

Dentro do **if** será criado uma variável local “criaturas”, onde dentro dela será atribuído uma função **g_map.getSpectatorsInRange()** que terá como parâmetro a variável “player” junto da função **getPosition()**. Depois será utilizado outro parâmetro com o valor booleano **false**. Em seguida, será utilizado um terceiro parâmetro, que será a propriedade que conterà o valor do raio do ataque em grupo da variável de configurações. O terceiro parâmetro será duplicado como um quarto parâmetro.

```
local mana = player:getMana()
if config.useGroupAttack and config.groupAttackSpell:len() > 1 and mana > config.minManaGroup then
    local creatures = g_map.getSpectatorsInRange(player:getPosition(), false, config.groupAttackRadius, config.groupAttackRadius)
```

Posteriormente serão criadas duas variáveis locais: a primeira será uma variável que receberá valores booleanos, caso haja jogadores em volta; a segunda será uma variável que contará quantos monstros há em volta do jogador.

Um **for** será criado, para passar a variável “criatura”, neste **for** será utilizado o comando **in ipairs** e como argumento será passado a variável “criaturas”. Dentro do **for** será criado um **if not**, onde será usado a variável “criatura” junto com a função **isLocalPlayer()**.

Em seguida será utilizado o operador lógico **and** e a variável “criatura” junto com a função **isPlayer()**. Novamente o operador lógico **and** será utilizado junto com uma expressão que ficará entre parênteses: **(not config.groupAttackIgnoreParty or creature:getShield() <= 2)**. Dentro do **if not** será atribuído o valor **true** para a variável criada anteriormente com o objetivo de receber valores booleanos caso houvesse jogadores em volta.

Em seguida será criado um **else if**, onde passará a variável “criatura” junto com a função **isMonster()**. Dentro do **else if** será utilizado a variável, também criada anteriormente com o objetivo de contar quantos monstros há em volta do jogador. Essa variável terá o seu valor atribuído como ela mesma mais um.

```
local creatures = g_map.getSpectatorsInRange(player:getPosition(), false, config.groupAttackRadius, config.groupAttackRadius)
local playersEmVolta = false
local monstros = 0
for _, creature in ipairs(creatures) do
    if not creature:isLocalPlayer() and creature:isPlayer() and (not config.groupAttackIgnoreParty or creature:getShield() <= 2) then
        playersEmVolta = true
    elseif creature:isMonster() then
        monstros = monstros + 1
    end
end
```

Logo após será criado um **if**, onde passará a variável que tem como objetivo contar quantos monstros há em volta do jogador, comparando se o valor dela é maior ou igual a da propriedade de números de alvos da variável “configuracao”, depois será utilizado um operador lógico **and** e uma expressão: **(not playersEmVolta or config.groupAttackIgnorePlayers)**. Dentro do **if** será utilizado outro **if** onde será utilizado o objeto **TargetBot** junto da função **sayAttackSpell()** que terá duas propriedades da variável de configuração como parâmetro. A primeira será a propriedade de spell de ataque em grupo e a segunda será a propriedade de delay de ataque em grupo.

```
if monstros >= config.groupAttackTargets and (not playersEmVolta or config.groupAttackIgnorePlayers) then
  if TargetBot.sayAttackSpell(config.groupAttackSpell, config.groupAttackDelay) then
    return
  end
end
end
end
```

De início, será criado um **if**, onde será passado a variável de configuração e a propriedade de usar runas em grupo dessa mesma variável. Será usado o operador lógico **and** que passará pelo mesmo processo do **if** citado anteriormente, mas que também irá verificar se o valor dela é maior que 100.

Dentro do **if**, citado acima, será criada uma variável local “criatura” e será utilizado um método **g_map.getSpectatorsInRange()**, no qual serão passados quatro parâmetros. Primeiro será passada a variável “criatura” junto com o método **getPosition()**. Após isso, será passado o segundo método que será o valor booleano **false**, em seguida é passado a variável de configuração e a propriedade de distância da runa de ataque em grupo, o quarto e último parâmetro será exatamente igual ao terceiro.

Será definida uma segunda variável local que será do tipo booleana para caso haja ou não jogadores em volta e após isso será criada uma terceira variável local chamada “monstros” que irá contar a quantidade de monstros presentes em volta do jogador.

```
if config.useGroupAttackRune and config.groupAttackRune > 100 then

  local creatures = g_map.getSpectatorsInRange(creature:getPosition(),
  false, config.groupRuneAttackRadius, config.groupRuneAttackRadius)

  local playersEmVolta = false
  local monstros = 0
```

Após isso, será feito um **for**, onde será passado a variável “criatura” e será usado o método **In Pairs**, dentro desse método será passado a variável “criaturas”. Dentro desse **for** será criado um **if not**, onde será usado a variável “criatura” com o método **isLocalPlayer()**, onde será usado novamente o operador lógico **and** e passado a variável “criatura” junto com o método **isPlayer()**. Novamente, será usado o operador lógico **and** e será passada a expressão **(not config.groupAttackIgnoreParty or creature:getShield() <= 2)**. Dentro do **if not**, será definido o valor **true** para a variável booleana que diz quando há jogadores em volta.

Em seguida, será definido o **else if** no qual será usado a variável “criatura” junto do método **isMonster()**. Dentro dele será atribuído um valor para a variável de contagem de monstros, o valor será o dela mesmo somado mais um.

```
for _, creature in ipairs(creatures) do
  if not creature:isLocalPlayer() and creature:isPlayer() and
    (not config.groupAttackIgnoreParty or creature:getShield() <= 2) then
    playersEmVolta = true
  elseif creature:isMonster() then
    monstros = monstros + 1
  end
end
```

Será definido um **if**, nele será passado a variável de contagem de monstros e irá ser verificado se essa variável tem um valor maior ou igual ao valor da propriedade de alvos da runa de ataque em grupo da variável de configuração. Com isso, será usado um operador lógico **and** e mais uma expressão: (**not playersEmVolta or config.groupAttackIgnorePlayers**). A variável “playersEmVolta” é a variável booleana citada anteriormente.

Dentro do **if** citado acima, será criado outro **if**, nele será passado o objeto **TargetBot** e será usado uma função **useAttackItem()**. Dentro da função, serão passados quatro parâmetros: o primeiro, será a propriedade da runa de ataque em grupo da variável de configuração; o segundo, será o valor numérico zero; o terceiro, a variável “criatura”; e o quarto, será a propriedade de delay da runa de ataque em grupo da variável de configuração.

```
----
if monstros >= config.groupRuneAttackTargets and (not playersEmVolta or config.groupAttackIgnorePlayers) then
  if TargetBot.useAttackItem(config.groupAttackRune, 0, creature, config.groupRuneAttackDelay) then
    return
  end
end
end
```

Agora, será criado um **if**, no qual será passado a variável de configuração juntamente da propriedade dela, que será a propriedade de usar a spell de ataque. Em seguida, será usado o operador lógico **and** e a propriedade de spell de ataque da variável de configuração, também será usado um método que irá retornar o tamanho dessa propriedade e será verificado se o tamanho dela é maior que o valor numérico um.

Será usado o operador lógico **and** para checar se o valor da variável “mana” é maior que o valor da propriedade de mana mínima da variável de configuração. Posteriormente, será criado um **if** dentro de outro **if**, no qual será passado o objeto **TargetBot** junto da função **sayAttackSpell()**, para essa função serão passados dois parâmetros: o primeiro será a propriedade de spell de ataque da variável de configuração e o segundo será o delay da spell de ataque que é uma propriedade da variável de configuração.

```
if config.useSpellAttack and config.attackSpell:len() > 1 and mana > config.minMana then
  if TargetBot.sayAttackSpell(config.attackSpell, config.attackSpellDelay) then
    return
  end
end
```

Será criado um **if**, nele será passado uma propriedade para usar runa de ataque da variável de configuração. A seguir, será usado o operador lógico **and** que será usado para passar a propriedade runa de ataque da variável de configuração e irá ser verificado se essa propriedade tem o valor maior que cem.

Dentro deste **if**, será criado outro **if**, no qual será passado o objeto **TargetBot** e será usado a função **useAttackItem()**, dentro dela serão passados quatro parâmetros: o primeiro será a propriedade de runa de ataque da variável de configuração, o segundo será o valor numérico zero, o terceiro parâmetro será a variável “criatura” e o quarto será a propriedade de delay da runa de ataque da variável de configuração.

```
if config.useRuneAttack and config.attackRune > 100 then
  if TargetBot.useAttackItem(config.attackRune, 0,
    creature, config.attackRuneDelay) then
    return
  end
end
end
```

Feito isso, deve-se fazer uso do objeto **TargetBot.Creature**, definindo para este uma função **walk()**.

Como parâmetro para a utilização, devem ser passadas três variáveis: “criatura”, “configuracao” e “alvo”. Dentro dessa função, deve-se criar algumas variáveis locais: a variável “criaturaPosição”, que receberá o valor da variável “criatura” juntamente do método **getPosition()**; a variável “posicao” que constará com o valor do objeto “player”, juntamente do método **getPosition()**; uma variável “surrounded”, que deve receber o valor booleano true; uma variável “dirs”, recebendo a inserção de uma tabela como valor para ela, onde dentro dela devem ser inseridos os valores: {{-1,1}, {0,1}, {1,1}, {-1, 0}, {1, 0}, {-1, -1}, {0, -1}, {1, -1}}, onde cada um dos valores deve servir para indicar uma direção dentro do jogo, sendo respectivamente: cima esquerda, cima, cima direita, esquerda, direita, baixo esquerda, baixo, baixo direita.

Em sequência, é necessário a criação de um **for** para percorrer todos os valores criados pela tabela “dirs”, onde deve ser criado uma variável local “piso”, que deverá utilizar como valor o método **g_map.getTile({x=pos.x-dirs[i][1],y=pos.y-dirs[i][2],z=pos.z})**, onde “pos” seria o nome da variável de posição.

Feito isso, cria-se um **if** utilizando a variável “piso”, juntamente de um operador lógico **and**, em sequência com a variável “piso” sendo reutilizada com o método **isWalkable()**, constando com o valor booleano **false**. Uma vez que a condição desse **if** seja atendida, será atribuído à variável “surrounded” um valor falso.

```
TargetBot.Creature.walk = function(creature, config, targets)
    local criaturapos = creature:getPosition()
    local pos = player:getPosition()

    local Trapado = true
    local pos = player:getPosition()
    local dirs = {{-1,1}, {0,1}, {1,1}, {-1, 0}, {1, 0}, {-1, -1}, {0, -1}, {1, -1}}
    for i=1,#dirs do
        local tile = g_map.getTile({x=pos.x-dirs[i][1],y=pos.y-dirs[i][2],z=pos.z})
        if tile and tile.isWalkable(false) then
            Trapado = false
        end
    end
end
```

Em seguida, cria-se um **if** que fará uso do objeto **TargetBot.canLure** juntamente do operador lógico **and**, passando uma propriedade “lure” da variável de configuração, ou a propriedade “lureCaveBot” da variável de configuração.

Em sequência, deve-se utilizar a operação lógica **and not**, sequente à propriedade “chase” da variável de configuração, e a variável “criatura” com o método **getHealthPercent()**, que deve comparar o valor dessa variável como sendo maior que 30. Por fim, é utilizado mais uma vez a operação lógica **and not**, juntamente da variável “surrounded”.

É então criada uma variável “monstros”, que receberá valor numérico igual à 0, outro **if** também será criado, onde receberá a variável de “alvos” e verificará se o valor da variável é menor que o valor da propriedade “lureCount” da variável de configuração.

Em seguida, é criado outro **if** que receberá a propriedade “lureCaveBot” da variável de configuração, que também contará com um **return**, para retornar o objeto **targetBot** juntamente da função **allowCaveBot(200)**. A seguir, deve-se definir um **else**, com a criação de uma variável local “caminho” que receberá o valor da função **findPath()**, onde essa função deve receber como parâmetros a variável “posicao”, “criaturaPosicao”, o valor numérico 5 e a seguinte expressão: **{ignoreNonPathable=true, precision=2}**.

Após isso, deve-se criar mais um **if** com intuito de passar a variável “caminho”, onde deve ser criado um return que devolverá o objeto **targetBot**, juntamente da função **walkTo()**, que devem constar com três argumentos para seu funcionamento: a variável “criaturaPosicao”, o valor numérico 10 e a expressão **{marginMin=5, marginMax=6, ignoreNonPathable=true}**.

```
if TargetBot.canLure() and (config.lure or config.lureCavebot) and not
(config.chase and creature:getHealthPercent() < 30) and not Trapado then

    local monstros = 0
    if targets < config.lureCount then
        if config.lureCavebot then
            return TargetBot.allowCaveBot(200)
        else
            local caminho = findPath(pos, criaturapos, 5, {ignoreNonPathable=true, precision=2})
            if caminho then
                return TargetBot.walkTo(criaturapos, 10, {marginMin=5, marginMax=6, ignoreNonPathable=true})
            end
        end
    end
end
end
end
```

Em seguida, será criada uma variável local “distanciaAtual”, que receberá como valor a função **findPath()**, onde dentro dessa função deverão ser passados 4 parâmetros: a variável “posicao”; a variável “criaturaPosicao”; o valor numérico 10 e a seguinte expressão: **{ignoreCreatures=true, ignoreNonPathable=true, ignoreCost=true}**.

Em sequência, será criado um **if** que receberá a propriedade “chase” da variável de configuração, que constará com o auxílio do operador lógico **and**, juntamente da expressão: **(creature:getHealthPercent() < 30 or not config.keepDistance)**. Onde “config” é a variável de configuração citada anteriormente. Ainda dentro desse **if**, deve-se criar outro **if** que servirá para passar a tabela “distanciaAtual”, com intuito de verificar se o valor dela será maior que 1. É criado então um return, juntamente da função **WalkTo()**, sendo passados três parâmetros : a variável “criaturaPosicao”, o valor numérico 10 e a expressão **{ignoreNonPathable=true, precision=1}**.

Sequente a isso, é necessário que se crie um **else if** para o primeiro **if**, que será utilizado a propriedade “keepDistance”, onde dentro desse **else if** será montado outro **if** que deve verificar se o valor da tabela “distanciaAtual” é diferente da propriedade “keepDistanceRange”, seguido do uso do operador lógico **and** para realizar a verificação se o valor da tabela “distanciaAtual” é diferente da propriedade “keepDistanceRange+1”. Ainda dentro desse **if**, será criado então um **return** para devolver o objeto **targetBot.walkTo**, sendo passados três parâmetros: a variável “criaturaPosicao”, o valor numérico 10 e a seguinte expressão: **{ignoreNonPathable=true, marginMin=config.keepDistanceRange, marginMax=config.keepDistanceRange + 1}**.

```

local distanciaAtual = findPath(pos, criaturapos, 10, {ignoreCreatures=true,
ignoreNonPathable=true, ignoreCost=true})

if config.chase and (creature:getHealthPercent() < 30 or not config.keepDistance) then
    if #distanciaAtual > 1 then
        return TargetBot.walkTo(criaturapos, 10, {ignoreNonPathable=true, precision=1})
    end
elseif config.keepDistance then
    if #distanciaAtual ~= config.keepDistanceRange and #distanciaAtual ~= config.keepDistanceRange + 1 then

        return TargetBot.walkTo(criaturapos, 10, {ignoreNonPathable=true, marginMin=config.keepDistanceRange,
marginMax=config.keepDistanceRange + 1})
    end
end
end

```

Feito isso, deve-se criar um **if** com a propriedade “avoidAttacks” da variável de configuração, utilizando três variáveis locais: “diffX”, que receberá os valores de X da variável “criaturaPosicao”, a propriedade “x” da variável “posicao”; a variável local “diffY”, que receberá os valores de Y da variável “criaturaPosicao”, a propriedade “y” da variável “posicao”; e por último, uma variável local que deve ser uma tabela, cujo nome será “candidato”.

Feito isso, é arquitetado um **if** que utilizará o método **math.abs()** que receberá como argumento a variável “diffX”, onde o valor dessa expressão deve ser igual à 1. Após isso, deve-se utilizar o operador lógico **and**, que receberá a variável “diffY” e o valor deve ser igual à 0. Ainda dentro desse **if**, é necessário atribuir um valor para a variável “candidatos”, que será compreendido como a expressão: **{{x=pos.x, y=pos.y-1, z=pos.z}, {x=pos.x, y=pos.y+1, z=pos.z}}**.

Após isso, é feito um processo similar, porém agora invertendo os valores de x e y. É necessário o uso do método **math.abs()**, que dessa vez receberá como argumento a variável “diffY”, onde o valor deverá ser igual a 1, enquanto após utilizar o operador lógico **and** que receberá a variável “diffX”, deve ser igual ao valor 0.

```

if config.avoidAttacks then
    local diffx = criaturapos.x - pos.x
    local diffy = criaturapos.y - pos.y
    local candidates = {}
    if math.abs(diffx) == 1 and diffy == 0 then
        candidates = {{x=pos.x, y=pos.y-1, z=pos.z}, {x=pos.x, y=pos.y+1, z=pos.z}}
    elseif diffx == 0 and math.abs(diffy) == 1 then
        candidates = {{x=pos.x-1, y=pos.y, z=pos.z}, {x=pos.x+1, y=pos.y, z=pos.z}}
    end
end

```

Dados os passos anteriores, é feito a criação de um **for**, onde também constará com a criação da tabela “candidatos”, contando com o método **in**

ipairs, onde deve-se passar a tabela “candidatos” como argumento para esse método. Ainda nesse **for**, é feita a criação de uma variável local “piso”, que receberá como valor a seguinte função: **g_map.getTile()**, onde a variável “candidatos” será passada como argumento, enquanto isso, é criado um **if** para passar a variável “piso”, logo após o uso do operador lógico **and**, onde novamente será passada a variável “piso”, porém dessa vez juntamente da função **isWalkable()**.

Dado os passos anteriores, é criado um **return** para devolver o objeto **targetBot**, juntamente da função **walkTo()**, onde serão retornados também três parâmetros: a variável “candidatos”, o valor numérico 2 e a expressão **{ignoreNonPathable=true}**.

```
for _, candidate in ipairs(candidates) do
    local tile = g_map.getTile(candidate)
    if tile and tile.isWalkable() then
        return TargetBot.walkTo(candidate, 2, {ignoreNonPathable=true})
    end
end
end
end
```

O arquivo aberto anteriormente na parte “programação de ataques” foi encerrado, nesse momento será necessário utilizar novamente o arquivo para carregar os arquivos do Target bot. Como feito no início do “Case 02”, será preciso realizar o mesmo processo e dessa vez adicionar o arquivo criado na programação de ataques a esse arquivo para carregar o Target bot.

Editor da Lista de Criaturas

Nessa parte do case 2 será criado um arquivo responsável pelo editor que irá adicionar os monstros a uma lista de criaturas que serão alvejadas e nesse mesmo editor será configurada algumas propriedades relacionadas às skills dos personagens.

De início, será necessário criar um arquivo a parte para o editor, esse deverá ficar na mesma pasta dos arquivos para o target bot criados em princípio. Na parte da programação, será usado o objeto **TargetBot.Creature**, em seguida será definida uma função **edit** como objeto, como parâmetro para ela serão passadas duas variáveis, sendo a primeira uma de configuração “config” e a segunda uma de retorno chamada “callback”. Dentro da função, será definido o valor da variável “config” como ela mesma e será usado o operador lógico **or** para passar o valor de uma tabela vazia.

Seguidamente, será criada uma variável local “editor”, a qual terá como valor atribuído o método **UI.createWindow()** e será passada como parâmetro a string

'TargetBotCreatureEditorWindow'. Posteriormente, será criada uma tabela local chamada "values" e será atribuída a ela o valor vazio.

```
TargetBot.Creature.edit = function(config, callback)
  config = config or {}

  local editor = UI.createWindow('TargetBotCreatureEditorWindow')
  local values = {}
```

Em seguida, a variável "editor" será usada com a propriedade **name** junto do método **setText()**, como parâmetro para esse método será passada a propriedade **name** da variável "config" e utilizará o operador lógico **or** para passar como parâmetro também uma string vazia.

Logo após, será usado o comando **table.insert()** e como argumento para esse comando será passada a tabela "values" e em seguida a seguinte expressão **{"name", function() return editor.name:getText() end}**.

Nesse momento, será criado uma variável local que terá como função adicionar uma barra de rolagem "addScrollBar". Na função serão passados como argumentos uma variável "id", uma variável "title", a variável "min", a variável "max" e em seguida a variável "defaultValue". Dentro dessa função será criada uma variável local chamada "widget", nela será estabelecida como valor o método **UI.createWidget()**, como parâmetro será passada uma string de valor **'TargetBotCreatureEditorScrollBar'**, após essa string como segundo parâmetro será utilizado o comando **editor.content.left**.

Em seguida, será criado um evento **onValueChange** para a propriedade scroll da variável "widget" através de uma função, como parâmetro para essa função será passada uma variável "scroll" e uma variável "value". Dentro dessa função será usado a propriedade **text** da variável "widget", essa propriedade será utilizada junto com o método **setText()**, como parâmetro para esse método será passada a variável "title" e será concatenada utilizando ".." com uma string com o valor ":", essa string será concatenada utilizando novamente ".." com a variável "value".

```
local addScrollBar = function(id, title, min, max, defaultValue)
  local widget = UI.createWidget('TargetBotCreatureEditorScrollBar', editor.content.left)
  widget.scroll.onValueChange = function(scroll, value)
    widget.text:setText(title .. ": " .. value)
  end
```

Posteriormente, será usada a propriedade **scroll** da variável "widget" novamente, mas dessa vez ela será utilizada juntamente ao método **setRange()**, como parâmetro para esse método serão passados as variáveis "min" e "max". Após isso, será criado um **if** que irá verificar se a variável "max - min" será maior que o valor numérico 1000, caso essa condição seja atendida dentro desse **if** será utilizado o método **setStep()** na propriedade **scroll** da variável "widget" e será passado o valor numérico 100 como parâmetro para o método. Seguidamente, será realizado esse mesmo processo novamente, mas no lugar do valor numérico 1000 será verificado se a subtração resultará no valor 100 e

caso essa condição seja atendida será atribuído o mesmo processo, mas com o valor 10.

```
widget.scroll:setRange(min, max)
if max-min > 1000 then
    widget.scroll:setStep(100)
elseif max-min > 100 then
    widget.scroll:setStep(10)
end
```

Novamente será utilizada a propriedade **scroll** da variável “widget” e dessa vez será utilizado o método **setValue()**, para esse método será passada como parâmetro a variável “config” na posição da variável “id”. Será utilizado o operador lógico **or** e também será passada como parâmetro a variável “defaultValue”. Após isso, serão passados dois parâmetro para o evento **onValueChange** da propriedade **scroll** da variável “widget”, o primeiro parâmetro será a própria propriedade **scroll** da variável “widget” e em seguida ela será passada novamente, porém dessa vez estará junto com o método **getValue()**.

Posteriormente será utilizado um **table.insert**, no qual serão passados como argumentos a variável “value” e a expressão **{id, function() return widget.scroll:getValue() end}**.

```
widget.scroll:setValue(config[id] or defaultValue)
widget.scroll.onValueChange(widget.scroll, widget.scroll:getValue())
table.insert(values, {id, function() return widget.scroll:getValue() end})
end
```

Será criada uma variável local para armazenar texto chamada “addTextEdit”, como valor para ela será adicionada uma função e dentro desta função serão passados três parâmetros: a variável “id”, a variável “title” e por último a variável “defaultValue”. Dentro dessa função, será criada novamente uma variável local chamada “widget” e para essa variável será atribuído como valor para a mesma o método **UI.createWidget()**, como parâmetro para ele será passada a string com valor **'TargetBotCreatureEditorTextEdit'** e depois o comando **editor.content.right**.

Será utilizada a propriedade **text** da variável “widget” junto com o método **setText()**, como parâmetro para esse método será usada a variável “title”. Após isso, será utilizada a propriedade **textEdit** junto com o método **setText()**, como parâmetro será passada a variável “config” na posição da variável “id”, será utilizado o operador lógico **or** e será passada também a variável “defaultValue”, mais uma vez o operador lógico **or** e será passada uma string vazia. Por fim, será utilizado o comando **table.insert**, será passada a variável “value” e a expressão **{id, function() return widget.textEdit:getText() end}** como parâmetro.

```
local addTextEdit = function(id, title, defaultValue)
    local widget = UI.createWidget('TargetBotCreatureEditorTextEdit', editor.content.right)
    widget.text:setText(title)
    widget.textEdit:setText(config[id] or defaultValue or "")
    table.insert(values, {id, function() return widget.textEdit:getText() end})
end
```

Será criado uma variável local nomeada como “addCheckBox”, onde será adicionada uma função que terá três parâmetros: “id”, “title”, “defaultValue”. Dentro desta função será criada uma variável local chamada “widget”, onde será adicionado o valor do método **UI.createWidget**, o método terá como parâmetro uma string contendo o valor **'TargetBotCreatureEditorCheckBox'** e em seguida o comando **editor.content.right**. Após isso a variável “widget” será usada junto com o evento **onClick** que atribuirá o valor de uma função, dentro da função o valor do **checkBox** será alterado. Caso o **checkBox** esteja marcado (**widget.isOn()** ele retornará verdadeiro) e será desmarcado usando **Not widget.isOn()**.

```
local addCheckBox = function(id, title, defaultValue)
    local widget = UI.createWidget('TargetBotCreatureEditorCheckBox', editor.content.right)
    widget.onClick = function()
        widget:setOn(not widget.isOn())
    end
end
```

Após isso será utilizado a propriedade **setText** da variável “widget”, a propriedade conterá o parâmetro “title”. Em seguida será criado um **if**, onde será verificado se a variável “config” junto com a chave **id** é igual a nulo (**nil**). Caso a condição seja verdadeira, será definido o valor da propriedade **setOn()** da widget como **defaultValue**. Caso a condição seja falsa será definido o valor da propriedade **setOn()** do widget como sendo a variável “config” junto com a chave **id**.

Após isso será utilizado o comando **table.insert** onde serão passados os parâmetros: “value”; e uma tabela contendo os valores **id**, uma função que retornará o estado atual da widget.

```
widget:setText(title)
if config[id] == nil then
    widget:setOn(defaultValue)
else
    widget:setOn(config[id])
end
table.insert(values, {id, function() return widget:isOn() end})
end
```

Em seguida será criada uma variável local nomeada como “addItem”, será atribuído a ela uma função que terá como parâmetro: uma variável “id”; uma variável “title”; uma variável “defaultItem”. Dentro desta função será criada uma variável local chamada “widget”, onde será adicionado o valor do método **UI.createWidget**, o método terá como parâmetro uma string contendo o valor **'TargetBotCreatureEditorItem'** e em seguida o comando **editor.content.right**.

Depois será atribuído o valor do parâmetro “title” para o campo text do widget, em seguida será definido um valor para o campo item da variável “widget” usando o método **setItemId()**, onde será passado como parâmetro a variável “config” e a chave **id**, também será utilizado o operador lógico **or** e o parâmetro “defaultItem”. Em seguida será utilizado o comando **table.insert**, onde será passado a variável “values” e uma tabela contendo os valores **id**, e uma função que retorna o id do campo item do widget.

```
local addItem = function(id, title, defaultItem)
    local widget = UI.createWidget('TargetBotCreatureEditorItem', editor.content.right)
    widget.text:setText(title)
    widget.item:setItemId(config[id] or defaultItem)
    table.insert(values, {id, function() return widget.item:getItemId() end})
end
```

Será criado um botão para cancelar, e para isso será utilizado um evento **onClick** e neste evento será atribuído uma função como valor, onde terá o comando **editor:destroy()**. Em seguida, utilizaremos a propriedade **onEscape** que será atribuída o botão cancelar e o evento **onClick**.

```
editor.cancel.onClick = function()
    editor:destroy()
end
editor.onEscape = editor.cancel.onClick
```

Após isso, será definido o botão ‘Ok’ para o editor e será criado um evento **onClick** para o botão. Em seguida será criado uma função para o botão, dentro desta função será criada uma tabela nomeada como “novaConfig”, em seguida

será criado um **for**, onde percorrerá uma tabela “value” usando o comando **in ipars**, que terá como parâmetro a variável “values”. Dentro do **for**, será usada a tabela “novaConfig”.

Dentro do loop, o valor do índice **value[1]** (o primeiro elemento de cada sub-tabela em values) é utilizado como chave para a tabela **novaConfig**, e o valor associado a essa chave é obtido chamando a função **value[2]()**. O resultado dessa função é atribuído à tabela “novaConfig” com a chave correspondente.

```
local novaConfig = {}
for _, value in ipairs(values) do
    novaConfig[value[1]] = value[2]()
end
```

Após isso, será criado um **if**, onde poderá ser verificado o tamanho do nome da tabela “novaConfig”, através do método **len()** que verificará se o tamanho do nome é menor que um. Caso o resultado seja verdadeiro nada será retornado, isso indica que o campo do nome da tabela corresponde a um campo vazio.

Em seguida será utilizado a tabela “novaConfig”, onde será definido a propriedade **regex** como uma string vazia. Depois será criado um **for**, onde será passado a função **string.gmatch()**, que terá como parâmetro a propriedade **name** da tabela “novaConfig” e será passado o padrão **([^\,]+)** para o método **string.gmatch()**. Dentro do **for** será criado um **if**, que verificará se a propriedade **regex** da tabela **novaConfig** não é uma string vazia. Caso a condição seja atendida, será adicionado o caractere “|” na propriedade regex da tabela “novaConfig”.

Em seguida, será atribuído um valor para a propriedade **regex** da tabela “novaConfig”. O valor será o dela, mais o caractere “^”, que significará o início da string. Após isso será utilizado o método **trim()**, que servirá para remover os espaços em branco do início e final da string, em seguida será utilizado **lower()**, que servirá para deixar toda a string em letra minúsculas, será utilizado o método **gsub()**, onde será passado como parâmetro a expressão **(“%*”, “.*”)**. Depois será utilizado o método **gsub()** para passar outra expressão **(“%?”, “.*?”)**.

Em seguida será acionado o método editor: **destroy()** e o método **callback()** onde terá como parâmetro a tabela “novaConfig”.

```
novaConfig.regex = ""
for part in string.gmatch(novaConfig.name, "[^\,]+") do
    if novaConfig.regex:len() > 0 then
        novaConfig.regex = novaConfig.regex .. "|"
    end
    novaConfig.regex = novaConfig.regex .. "^" .. part:trim():lower():gsub("%*", ".*"):gsub("%?", ".*?") .. "$"
end

editor:destroy()
callback(novaConfig)
end
```

Por fim, será definido os valores para os métodos: **addScrollBar(); addCheckBox(); addItem(); textEdit()**.

```
addScrollBar("prioridade", "Prioridade", 0, 10, 1)
addScrollBar("perigo", "Perigo", 0, 10, 1)
addScrollBar("maxDistance", "Distancia Maxima", 1, 10, 10)
addScrollBar("keepDistanceRange", "Manter Distancia", 1, 5, 1)
addScrollBar("lureCount", "Lurar", 0, 5, 1)
```

```
addScrollBar("minMana", "Mana Min. para Spell de Ataque", 0, 3000, 200)
addScrollBar("attackSpellDelay", "Delay para Spell de Ataque", 200, 5000, 2500)
addScrollBar("minManaGroup", "Mana Min. para Spell de Atq em grupo", 0, 3000, 1500)
addScrollBar("groupAttackTargets", "Min. de Alvos para Spell em Grupo", 1, 10, 2)
addScrollBar("groupAttackRadius", "Range para Spell em Grupo", 1, 7, 1)
addScrollBar("groupAttackDelay", "Delay para Spell em Grupo", 200, 60000, 5000)
addScrollBar("runeAttackDelay", "Delay da Runa de Ataque", 200, 5000, 2000)
addScrollBar("groupRuneAttackTargets", "Min. de Alvos para Runa de Ataque", 1, 10, 2)
addScrollBar("groupRuneAttackRadius", "Range da Runa de Ataque", 1, 7, 1)
addScrollBar("groupRuneAttackDelay", "Delay da Runa de Ataque", 200, 60000, 5000)
```

```
addCheckBox("chase", "Chase", true)
addCheckBox("keepDistance", "Manter Distancia", false)
addCheckBox("dontLoot", "Nao Lootear", false)
addCheckBox("lure", "Lure", false)
addCheckBox("avoidAttacks", "Desviar de Ataques em Wave", false)
```

```
addCheckBox("useSpellAttack", "Usar Spell de Ataque", false)
addTextEdit("attackSpell", "Spell de Ataque:", "")
addCheckBox("useRuneAttack", "Runa de Ataque", false)
addItem("attackRune", "Runa de Ataque:", 0)
addCheckBox("useGroupAttack", "Usar Spell de Ataque em Grupo", false)
addTextEdit("groupAttackSpell", "Spell de Ataque em Grupo", "")
addCheckBox("useGroupAttackRune", "Runa de ataque", false)
addItem("groupAttackRune", "Runa de Ataque:", 0)
addCheckBox("groupAttackIgnorePlayers", "Ignorar jogadores em Atq em Grupo", false)
addCheckBox("groupAttackIgnoreParty", "Ignorar PT em Ataque em Grupo", false)
end
```

Assim como feito no processo anterior, será necessário incluir o diretório do arquivo dentro do arquivo “principal” que irá iniciar os arquivos do bot

Controle de Foco do Bot

Para início, é necessário criar um novo arquivo com a extensão .lua, adicionando esse arquivo aos outros arquivos do bot. Neste novo arquivo, será definida uma função que irá alternar o foco do ataque do personagem por meio

de medidas específicas, sendo elas: distância e porcentagem da vida dos Mobs. Também será necessário dar um nome que remeta ao foco do bot, juntamente com a ação de incluir seu diretório dentro dos arquivos do bot para que seja possível iniciá-lo.

Após estabelecer os passos anteriores, deve-se então criar uma função que deve calcular a prioridade do foco, pertencente ao objeto **TargetBot.Creature**, recebendo três parâmetros iniciais: a variável “creature”, “config” e “caminho”. Em seguida, deve-se criar uma variável local nomeada “prioridade”, iniciando a mesma com o valor número 0.

Feito isso, deve-se criar então um **if**, onde nele se passará o comando **g_game.getAttackCreature() == Creature**, de forma que verifique se os parâmetros para o funcionamento sejam atendidos.

Ainda dentro desse mesmo **if**, é necessário passar a variável “prioridade”, na qual deve-se incrementá-la em +1. Sequente a isso, cria-se então um novo **if** para verificar se a variável “caminho” será maior do que o campo de distância máxima da variável “config”, onde caso essa condição seja atendida, deve-se então retornar a variável “prioridade”. Feito isso, é necessário adicionar para a variável “prioridade” o valor do campo prioridade da variável “config”.

```
TargetBot.Creature.calculaPrioridade = function(creature, config, caminho)

    local prioridade = 0

    if g_game.getAttackingCreature() == creature then
        prioridade = prioridade + 1
    end

    if #caminho > config.maxDistance then
        return prioridade
    end

    prioridade = prioridade + config.prioridade
```

Em seguida, cria-se uma variável local, que pode ser nomeada como por exemplo “tamanhoCaminho”, que deve receber o valor da variável “caminho”, e então, deve ser criado um **if** para verificar se o valor da variável “tamanhoCaminho” será igual à 1, e caso essa seja a situação, deve-se incrementar em +3 o valor da variável “prioridade”.

Ainda para este **if**, deve-se criar um **else If** para realizar a verificação de se o tamanho da variável “tamanhoCaminho” será menor ou igual a 3, e em caso desta condição ser atendida, a variável prioridade deve ser incrementada em +1.

```
local caminho_length = #caminho
if caminho_length == 1 then
    prioridade = prioridade + 3
elseif caminho_length <= 3 then
    prioridade = prioridade + 1
end
```


Por fim, é feito a configuração de um outro **if**, contendo o campo **Chase** da variável “config”, utilizando o operador lógico **and** e o método **getHealthPercent()** na variável “creature”, realizando uma comparação para verificar o valor será menor que 30, e que caso essa condição for atendida, o valor da variável “prioridade” deve ser incrementado em +5.

Feito isso, deve-se criar um **else if**, utilizando novamente o método **getHealthPercent()** na variável “creature”, comparando se o valor será menor que 20, e que caso a condição seja atendida, deve-se então incrementar a variável “prioridade” em +2.5.

Feito isso, deve-se criar uma fila de 4 repetições de **else If** para realizar a verificação da porcentagem da vida dos mobs, utilizando o método **getHealthPercent()** na variável “creature”, de forma que:

Caso o valor retornado seja menor que 20, deve-se incrementar o valor da variável “prioridade” em +2.5.

Caso o valor retornado seja menor que 40, deve-se incrementar o valor da variável “prioridade” em 1.5.

Caso o valor retornado seja menor que 60, deve-se incrementar o valor da variável “prioridade” em 0.5.

E por fim, caso o valor retornado seja menor que 80, deve-se incrementar o valor da variável “prioridade” em 0.2.

E para finalizar, já fora dos ifs, deve ser retornado o valor da variável “prioridade”.

```
if config.chase and creature:getHealthPercent() < 30 then
    prioridade = prioridade + 5
elseif creature:getHealthPercent() < 20 then
    prioridade = prioridade + 2.5
elseif creature:getHealthPercent() < 40 then
    prioridade = prioridade + 1.5
elseif creature:getHealthPercent() < 60 then
    prioridade = prioridade + 0.5
elseif creature:getHealthPercent() < 80 then
    prioridade = prioridade + 0.2
end

return prioridade
end
```

Após finalizado o arquivo do controle de foco, deve-se inserir seu diretório dentro do arquivo que irá carregar o arquivo do bot, assim como foi feito nos passos anteriores.

Programação do Sistema de Loot

O sistema de Loot será o arquivo destinando a programação dos scripts para realizar a adição das funcionalidades de coleta dos itens que serão deixados para trás pelos mobs.

Para início, deve-se primeiro criar uma tabela vazia “looting” para o objeto TargetBot, e em seguida criar uma nova tabela vazia “lista” para complementar a tabela “looting” do objeto **TargetBot**.

Feito isso, cria-se uma variável local “ui”, juntamente da criação de uma tabela local vazia “itens”, onde em sequência cria-se uma nova tabela local, também vazia, que será chamada de “mochila”. Em seguida, cria-se uma nova tabela local vazia chamada “itensPorId”, seguinte por uma outra tabela local vazia “mochilasPorId”, juntamente da criação de uma variável local “naoSalva”, a qual receberá o valor falso.

```
TargetBot.Looting = {}
TargetBot.Looting.list = {}

local ui
local itens = {}
local mochilas = {}
local itensById = {}
local mochilasById = {}
local naoSalva = false
```

Para o próximo passo, deve-se definir um campo **setup**, onde nele deve ser atribuída uma função, e dentro dessa, utiliza-se a variável “ui” atribuindo a ela o valor do texto “TargetBotLootingPanel” através do método “UI.createWidget()”, e em seguida deve-se utilizar o método “UI.Container()”, e como parametro é necessário utilizar o campo **onItemsUpdate** da tabela “looting” do objeto **TargetBot**. Em seguida, é necessário passar como parâmetro o valor booleano true, juntamente do valor nil, e como último parâmetro, é realizada a inserção da variável “ui” juntamente da tabela “itens”.

Feito isso, é necessário repetir a linha de código anterior, porém desta vez substituindo **onItemsUpdate** por **onContainersUpdate**, juntamente da substituição de **ui.itens** por **ui.mochila**.

```
TargetBot.Looting.setup = function()
    ui = UI.createWidget("TargetBotLootingPanel")
    UI.Container(TargetBot.Looting.onItemsUpdate, true, nil, ui.itens)
    UI.Container(TargetBot.Looting.onContainersUpdate, true, nil, ui.mochilas)
```

Após a finalização do passo anterior, cria-se um evento **onClick** para a propriedade **everyItem** da variável “ui”, e para esse evento **onClick**, deve-se atribuir uma função para o mesmo. A função deverá passar o método **isOn()** na propriedade **everyItem** da variável “ui”, e como parâmetro, ele receberá a cláusula “not”, e o método **isOn()** da propriedade **everyItem** da variável “ui”. Em seguida, é utilizado o método **save()** no objeto **TargetBot**.

```
ui.everyItem.onClick = function()
  ui.everyItem:setOn(not ui.everyItem:isOn())
  TargetBot.save()
end
```

Em sequência, cria-se outro evento, **onTextChange**, que terá a finalidade de complementar a propriedade **value** que faz parte do campo “painelDePerigo” da variável “ui”, e para esse evento será definida uma função, juntamente da criação de uma variável local “value”, que receberá o método **tonumber()**, e como parâmetro para esse método, será passado o método **getText()**, da propriedade **value** do campo **painelDePerigo** da variável “ui”.

Sequente a isso, cria-se um **If not** que receberá a variável “value”, e dentro desse **If not**, deve ser usado o método **setText()** da propriedade **value** do campo **painelDePerigo** da variável “ui”, e como parâmetro, receberá o valor 0.

Feito isso, deve ser criado um **If**, contando com a variável “naoSalva”, que deverá retornar sem valor algum caso ela não seja marcada como verdadeira. Após isso, é necessário utilizar o objeto “TargetBot” com o método **save()**.

```
ui.maxDangerPanel.value.onTextChange = function()
  local value = tonumber(ui.maxDangerPanel.value:getText())
  if not value then
    ui.maxDangerPanel.value:setText(0)
  end
  if naoSalva then return end
  TargetBot.save()
```

Como próximo passo, é necessário repetir as mesmas linhas de comando que foram citadas no passo anterior, porém substituindo **painelDePerigo** por **painelDeCapacidade**.

```
ui.minCapacityPanel.value.onTextChange = function()
  local value = tonumber(ui.minCapacityPanel.value:getText())
  if not value then
    ui.minCapacityPanel.value:setText(0)
  end
  if naoSalva then return end
  TargetBot.save()
end
end
```

Como próximo passo, é necessário criar um evento **onItemsUpdate** dentro da tabela “looting” do objeto **TargetBot**, e para esse evento deve ser definida uma função que constará com um **if**, recebendo a variável “naoSalva”, que caso essa variável não seja verdadeira, ela não retorna nada. Seguidamente a isso, deve-se colocar o objeto **TargetBot** em conjunto com o método **save()**, em seguida utilizando o método **updateItemsAndContainers()** dentro da tabela “looting” do objeto **TargetBot**.

```
TargetBot.Looting.onItemsUpdate = function()
  if naoSalva then return end
  TargetBot.save()
  TargetBot.Looting.updateItemsAndContainers()
end
```

Após isso, deve-se refazer o passo anterior, porém mudando o evento para qual ele seria direcionado, dessa forma, alterna-se de **onItemsUpdate** para **onContainersUpdate**.

```
TargetBot.Looting.onContainersUpdate = function()
  if naoSalva then return end
  TargetBot.save()
  TargetBot.Looting.updateItemsAndContainers()
end
```

Será definida uma função **update()** dentro da tabela “looting” do objeto **TargetBot**, como parâmetro dessa função será passada a tabela “data”. Dentro dessa função será definida a variável “naoSalva” com o valor **true**, após isso será definida a tabela “lista” que está dentro da tabela “looting” como uma tabela vazia.

Irá ser usado o método **setItems()** no campo **items** da variável “ui”, como parâmetro desse método será passada a tabela “data” e dentro dessa tabela será passado o campo **items**. Em seguida, será utilizado o operador lógico **or** e será passada como parâmetro uma tabela vazia.

Sequencialmente será utilizado o método **setItems()** no campo **mochila** da variável “ui”, a tabela “data” será passada como parâmetro e dentro dessa tabela será passado o campo **mochila**. Em seguida, será usado o operador lógico **or** e será passada como parâmetro uma tabela vazia.

```
TargetBot.Looting.update = function(data)
  naoSalva = true
  TargetBot.Looting.list = {}
  ui.items:setItems(data['items'] or {})
  ui.mochilas:setItems(data['mochilas'] or {})
```

Nesse momento, será utilizado o método **setOn()** no campo **everyItem** que pertence a variável “ui”, como parâmetro para esse método será passada a tabela “data” com o valor **everyItem**. Após isso, será utilizado o método **setText()** no campo **value** que é pertencente ao **paineldeperigo** que pertence a variável “ui”, como parâmetro para ele será passada a tabela “data” com o valor da variável de perigo. Será passado também o operador lógico **or** e o valor numérico 10.

Esse mesmo processo será repetido para o **paineldecapacidade**, as únicas diferenças são que no lugar do **paineldeperigo** será colocado o **paineldecapacidade** e o valor numérico 100 substituirá o 10 colocado no processo anterior.

```
ui.maxDangerPanel.value:setText(data['maxDanger'] or 10)
ui.minCapacityPanel.value:setText(data['minCapacity'] or 100)
```

Nessa linha será usada a função criada anteriormente chamada **updateItemsAndContainers()**, em seguida será atribuído o valor da variável “naoSalva” como falso.

```
TargetBot.Looting.updateItemsAndContainers()
naoSalva = false
end
```

Agora será definido uma função **save()** para a tabela “looting” do objeto **Targetbot**, como parâmetro para ela será passada a tabela “data”. Será definido um valor para o campo **items** da tabela “data”, esse valor será o retorno do método **getItems()** que estará sendo usado no campo **items** da variável “ui”.

Esse processo será repetido, mas agora será utilizado para os campos **mochila**.

```
TargetBot.Looting.save = function(data)
    data['items'] = ui.items:getItems()
    data['mochilas'] = ui.mochilas:getItems()
```

Será definido como valor do campo **perigo** da tabela “data” o método **toNumber()**, como parâmetro para ele será usado o método **getText()** na propriedade **value** do **paineldeperigo** da variável “ui”.

Esse processo será feito novamente para o **paineldecapacidade**.

Em sequência, será definido um valor para o campo **everyItem** da tabela “data” e esse valor será dado pelo método **isOn()** que será usado no campo **everyItem** da variável “ui”.

```
data['maxDanger'] = tonumber(ui.maxDangerPanel.value:getText())
data['minCapacity'] = tonumber(ui.minCapacityPanel.value:getText())
data['everyItem'] = ui.everyItem:isOn()
end
```

Após isso, será definida a função **updateItemsAndContainers()** para a tabela "looting" do objeto **TargetBot**. Dentro dessa função será definida a variável "item" com o valor "getItems" sendo usado no campo **items** da variável "ui".

O mesmo processo será realizado para a variável "mochila".

Em seguida, será definida a tabela "itensPorID" e a tabela "mochilasPorID" como vazias. Após isso, será criado um **for** que irá percorrer a tabela "items" de maneira sequencial, dentro do **for** para cada item da tabela "itensPorID" será definido o valor igual a 1. Esse processo se repetirá para a tabela "mochilasPorID"

```
TargetBot.Looting.updateItemsAndContainers = function()  
  items = ui.items.getItems()  
  mochilas = ui.mochilas.getItems()  
  itemsById = {}  
  mochilasById = {}  
  for i, item in ipairs(items) do  
    itemsById[item.id] = 1  
  end  
  for i, container in ipairs(mochilas) do  
    mochilasById[container.id] = 1  
  end  
end
```

Nesse momento, serão definidas quatro variáveis locais, sendo a primeira a variável "espereAte" sendo iniciada no valor numérico 0. A segunda variável será a "esperaMochila" e ela começará com o valor nil. A terceira é a variável "status" com o valor de uma string vazia. E por último, a quarta que é a variável "ultimoAlimento" que também será iniciada com o valor 0.

```
local waitTill = 0  
local esperaMochila = nil  
local status = ""  
local ultimaFood = 0
```

Após isso, será definida a função **getStatus()** e essa função irá retornar a variável "status".

```
TargetBot.Looting.getStatus = function()  
  return status  
end
```

Dito isso, agora será definida a função processo para a tabela "Looting" e como parâmetro será passado as variáveis "alvos" e a variável que contém o

nível de perigo, após isso será criado um **if**, onde como condição será usado a cláusula **not** e será passada a tabela “itens” com o valor 1. Em seguida será usado o operador lógico **and** e usado novamente a cláusula **not** e será passado o método **isOn()** no campo **everyItem** da variável “ui”, será utilizado o operador lógico **or** e mais uma vez a cláusula **not** e será passado a tabela “mochila” com o valor numérico 1, caso essa condição seja atendida a variável status será definida como uma string vazia e será retornado o valor booleano false.

```
TargetBot.Looting.process = function(targets, dangerLevel)
  if (not items[1] and not ui.everyItem:isOn()) or not mochilas[1] then
    status = ""
    return false
  end
end
```

Agora será criado um **if** onde no mesmo será verificado se a variável que contém o nível de perigo possui um valor maior que o retorno do método **toNumber()** com o parâmetro o retorno do método **getText()** da propriedade **value** do **paineldeperigo** da variável “ui”, caso esta condição seja atendida a variável “status” será definida com o texto “muito perigoso” e retornará o valor booleano false.

```
if dangerLevel > tonumber(ui.maxDangerPanel.value:getText()) then
  status = "Muito Perigoso"
  return false
end
```

A partir de agora será criado mais um **if**, onde será comparado se o retorno do método **getFreeCapacity()** da variável “player” possui um valor menor que o retorno do método **toNumber()** que possuirá como parâmetro o método **getText()** na propriedade **value** do **paineldecapacidade** da variável “ui”, caso esta condição seja atendida a variável “status” será definida com o texto “sem capacidade” e a tabela “lista” será definida como vazia, logo será retornado o valor booleano false.

```
if player:getFreeCapacity() < tonumber(ui.minCapacityPanel.value:getText()) then
  status = "Sem Capacidade"
  TargetBot.Looting.list = {}
  return false
end
```

Será definida uma variável local nomeada de “loot” e receberá o valor da tabela “lista” que pertence a tabela “loot”, após isso será criado um **if** e nesse **if** será verificado se a variável “loot” possui o valor **nil**, caso a condição seja atendida a variável “status” receberá o valor de uma string vazia, logo retornará o valor booleano false.

```
local loot = TargetBot.Looting.list[1]
if loot == nil then
  status = ""
  return false
end
```

Um novo **if** será criado para que seja possível verificar se a variável “espereAte” possui um valor maior que a variável que possui o valor do momento atual - a mesma variável usada na parte de “programando os ataques” do case 02 desta apostila - e caso a condição seja atendida será retornado o valor booleano true.

```
if waitTill > now then
    return true
end
```

Uma variável local será criada e nomeada como “mochila” e será atribuída a mesma o valor do método **g_game.getContainers()** e em seguida será criada uma tabela local chamada “mochilaLoot” e para a mesma será atribuída o método **getLootContainers()** com uma variável “mochila” como parâmetro. Após isso será criado um **if** com a cláusula **not** e será passado a tabela “mochilasLoot” com o valor 1, onde será verificado se o valor é diferente de nulo, caso a condição seja atendida a variável “status” será definida como o texto “sem espaço” e será retornado o valor booleano false.

```
local mochilas = g_game.getContainers()
local mochilasLoot = TargetBot.Looting.getLootContainers(mochilas)

if not mochilasLoot[1] then
    status = "No space"
    return false
end
```

Após o **if**, a variável “status” será definida com o texto “looteando”. Em seguida, será criado um **for** que irá percorrer a tabela “mochilas” de maneira sequencial, dentro desse **for** será criado um **if**, onde será verificado quais mochilas foram marcadas como uma mochila para **loot**. Dentro do **if**, será usada a função **lootContainer()** da tabela “looting”, como parâmetro para esse método serão passadas a tabela “mochilasLoot” e a variável da análise de controle do **for**, além de ser retornado o valor booleano true.

```
status = "Looteando"

for index, container in pairs(mochilas) do
    if container.lootContainer then
        TargetBot.Looting.lootContainer(mochilasLoot, container)
        return true
    end
end
```

Agora iremos criar uma variável local chamada “posicao” e será atribuído o valor do método **getPosition()** na variável “player”, em seguida será criada outra variável local, chamada “distancia”, onde o valor atribuído será **math.max** que terá como parâmetro o método **math.abs()**. O método **math.abs()** terá como

parâmetro o eixo 'x' da variável "posicao", realizando uma subtração com o eixo 'x' da variável "loot". O método **math.abs()** fará a mesma subtração, porém agora utilizando o eixo 'y'.

Em seguida, será criado um **if** para verificar se o número de tentativas de coletar o **loot** é maior que 30 (propriedade **tries** do objeto **loot**). Neste mesmo **if** será utilizado o operador lógico **or** para verificar se a posição do **loot** é diferente da posição do jogador. Em seguida será utilizado novamente o operador lógico **or** para verificar se a variável "distancia" possui um valor maior que 20. Caso as condições sejam atendidas será usado o comando **table.remove**, que terá como parâmetro a tabela lista da tabela "looting", também será passado como parâmetro o valor numérico 1, onde será retornado o valor booleano true.

```
local pos = player:getPosition()
local dist = math.max(math.abs(pos.x-loot.pos.x), math.abs(pos.y-loot.pos.y))
if loot.tries > 30 or loot.pos.z ~= pos.z or dist > 20 then
    table.remove(TargetBot.Looting.list, 1)
    return true
end
```

Logo após será criada uma variável local, chamada de "piso", que terá como atribuição o valor do método **g_map.getTile()** onde será passado como parâmetro a posição da variável "loot". Em seguida será criado um **if**, onde será verificado se a variável "distancia" possui um valor maior ou igual a 3. Depois será utilizado o operador lógico **or** e a cláusula **not** para verificar se a variável "piso" possui um valor diferente de nulo. Caso a condição seja atendida a propriedade "tries" do objeto **loot** será incrementada em mais um.

Após isso, será utilizada a função **andarAte()** e como parâmetro para esta função, será passada a posição do objeto **loot**, o valor numérico 20 e a tabela a seguir: **{ ignoreNonPathable = true, precision = 2 }**. E também será retornado o valor booleano true.

```
local tile = g_map.getTile(loot.pos)
if dist >= 3 or not tile then
    loot.tries = loot.tries + 1
    TargetBot.walkTo(loot.pos, 20, { ignoreNonPathable = true, precision = 2 })
    return true
end
```

Em seguida será criada uma variável local, chamada “container”, onde será utilizado um método **getTopUseThing()** na variável “piso”. Em seguida será utilizado um **if**, com a cláusula **not**, para verificar se a variável “container” é nula.

Logo após, será utilizado o operador lógico **or** junto com a cláusula **not**, onde será utilizado o método **isContainer()** na variável “container”. Caso a condição seja atendida será utilizado o método **table.remove()** e como parâmetro será passado a tabela “lista” da tabela “looting” e o valor numérico 1, também será retornado o valor booleano true.

```
local container = tile:getTopUseThing()
if not container or not container:isContainer() then
    table.remove(TargetBot.Looting.list, 1)
    return true
end
```

Em seguida será utilizado o método **g_game.open()** que terá como parâmetro a variável “container”. Após isso, para a variável “espereAte” será atribuído o valor da variável que contém o valor do momento atual, mais o valor numérico 1000. A variável “esperaMochila” terá o valor atribuído como o do método **getId()** da variável “container”. A variável “tries” do objeto **loot** será incrementada em 10, também será retornado o valor booleano true.

```
g_game.open(container)
waitTill = now + 1000
esperaMochila = container:getId()
loot.tries = loot.tries + 10

return true
end
```

Em seguida será definido uma função **getLootContainers()**, esta função terá como parâmetro a variável “mochila”, criada anteriormente. Após isso, uma primeira tabela vazia, chamada “mochilasLoot”, será criada e uma segunda tabela, também vazia, chamada “abrirMochilaPorId”, e uma variável local “abrir”, que receberá o valor **nil**.

```
TargetBot.Looting.getLootContainers = function(mochilas)
    local mochilasLoot = {}
    local abrirMochilasById = {}
    local toOpen = nil
```

Logo após, será criado um **for** que percorrerá a tabela mochila, de maneira sequencial. Dentro do **for** será utilizado os métodos **getId()** e **getContainerItem()**, onde para cada elemento (variável container definida juntamente ao **for**) da tabela “abrirMochilaPorId” será atribuído o valor numérico 1.

Em seguida, será criado um **if** para verificar duas condições, sendo a primeira, se o id obtido no comando anterior existe na tabela “mochilasPorId”. Para isso será utilizado o operador lógico **and** e a cláusula **not**, para verificar se a mochila não está sendo utilizada como mochila para loot. Isso será feito através da propriedade “lootContainer” de cada elemento da tabela mochila

```
for index, container in pairs(mochilas) do
  abrirMochilasById[container:getContainerItem():getId()] = 1
  if mochilasById[container:getContainerItem():getId()] and not container.lootContainer then
```

Dentro deste **if** criado anteriormente, será criado outro **if**, onde utilizaremos o método **getItemsCount()** no elemento da tabela mochila utilizado no **for** para verificar se o valor é menor que o valor do método **getCapacity()** que também será utilizado no mesmo elemento. Caso a condição seja atendida, será utilizado o método **table.insert()** que terá como parâmetro a tabela “mochilasLoot” e o elemento utilizado anteriormente no **for**.

```
if container:getItemsCount() < container:getCapacity() then
  table.insert(mochilasLoot, container)
```

Em seguida será criado um **else** e um **for** que percorrerá de maneira sequencial todos os itens dentro do “container” e em seguida será criado um **if** para verificar se o item é um “container” e se o id desse item está presente na tabela “mochilasPorId”. Caso a condição seja atendida será utilizado a variável “abrir” que terá como atribuição o par contendo o elemento item do segundo **for** criado e o elemento “container” do primeiro **for**. Para sair do loop será utilizado o comando **break**.

```
else
  for slot, item in ipairs(container:getItems()) do
    if item:isContainer() and mochilasById[item:getId()] then
      toOpen = {item, container}
      break
    end
  end
end
end
end
end
```

Em sequência, é necessário criar um novo **if** utilizando a cláusula **not** para verificar se a tabela “mochilasLoot” está vazia no índice 1. Caso a condição seja atendida como verdadeira, deve-se criar então um outro **if** para verificar se a variável “abrir” tem um valor diferente de nulo, dessa forma, caso a condição seja atendida, é utilizado então o método **g_game.open()**, utilizando como argumento a variável “abrir” tanto no índice 1 quanto no índice 2.

Feito isso, é necessário utilizar a variável “ate” para definir um valor para ela mesma, que será o valor da variável que contém o valor do momento atual + o valor numérico 500, de forma que retorne a tabela “mochilasLoot”.

```
if not mochilasLoot[1] then
  if toOpen then
    g_game.open(toOpen[1], toOpen[2])
    waitTill = now + 500
    return mochilasLoot
  end
end
```

Após isso, cria-se um novo **for** para percorrer de maneira sequencial a tabela “mochila”, onde dentro deste **for**, deverá ser criado um **if not**, que receberá a tabela “mochilasPorId” para verificar se a mochila estará marcada como uma mochila de loot, e para isso, será passado como índice a variável do **for**, juntamente dos métodos **getId()** e **getContainerItem()**, em conjunto do operador lógico **and** e a cláusula **not**, de forma que receba a variável do **for**, e da sua propriedade **lootContainer**, e que caso essa condição inicial seja atendida, é criado um segundo **for** que deverá percorrer de maneira sequencial a variável do índice do **for** anterior, juntamente do método **getItem()**.

Após isso, deve se criar um **if**, de forma que receba a variável de índice do segundo for, utilizando o método **isContainer()**, em conjunto do operador lógico **and**, passando a tabela “mochilaPorId” nesse índice for citado anteriormente, juntamente do método **getId()**, onde caso a condição desse **if** seja atendida, é utilizado então o método **g_game.open()**, que recebe como parâmetro o índice do segundo **for**.

Feito isso, é necessário utilizar a variável “espereAte” para atribuir a ela o valor da variável que contenha o momento atual + o valor numérico 500, de modo que retorne a tabela “mochilasLoot”.

```
for index, container in pairs(mochilas) do
  if not mochilasById[container:getContainerItem():getId()] and not container.lootContainer then
    for slot, item in ipairs(container:getItems()) do
      if item:isContainer() and mochilasById[item:getId()] then
        g_game.open(item)
        waitTill = now + 500
        return mochilasLoot
      end
    end
  end
end
```

Feito isso, cria-se um **for** para configurar que a variável “slot” seja igual a variável “primeiroSlotInventario”, que também deve ser igual a variável “ultimoSlotInventario”. Ainda dentro deste **for**, cria-se uma variável local “item”, que receberá como valor o método **getInventoryItem()**, e como parâmetro, receberá a variável “slot”. Após isso, cria se um **if** para verificar se a variável “item” tem o valor diferente de nulo e se a variável é uma mochila, juntamente com a confirmação de que o id da variável não esteja presente na tabela

“abrirMochilasPorId”, onde caso essa condição seja atendida, é utilizado o método `g_game.open()`, recebendo como parâmetro a variável “item”.

Em sequência, é utilizada a variável “espereAte”, que recebe como valor a variável que contém o momento atual + o valor numérico 500. Feito isso, é retornada a tabela “mochilasLoot”, e por fim, para encerrar a função `getLootContainers()` haverá um segundo retorno a tabela “mochilasLoot”.

```
for slot = InventorySlotFirst, InventorySlotLast do
    local item = getInventoryItem(slot)
    if item and item:isContainer() and not abrirMochilasById[item:getId()] then
        g_game.open(item)
        waitTill = now + 500
        return mochilasLoot
    end
end
end
return mochilasLoot
end
```

Como próximo passo, deve ser definida uma função `lootContainer()` para a tabela “looting”, e como parâmetro é necessário passar a tabela “mochilasLoot” e a variável “container”. Nessa função, deve ser definida uma variável local “proximaMochila”, que terá valor de “nil”. Em sequência, cria-se um `for` para percorrer de maneira sequencial cada “item” da variável “container”, em conjunto com o método `getItems()`.

```
TargetBot.Looting.lootContainer = function(mochilasLoot, container)
    local proxMochila = nil
    for i, item in ipairs(container:getItems()) do
```

Em seguida, deve ser criado um `if` para verificar se essa variável “item” é um container utilizando o método `isContainer()` e se o id dessa variável “item” não está presente na tabela “itemsPorId”, onde caso essa condição seja atendida, a variável “proximaMochila” deve receber o valor da variável “item”.

```
if item:isContainer() and not itemsById[item:getId()] then
    proxMochila = item
```

Seguindo, é criado então um `else if` que receberá a variável “itemsPorId”, e como índice a variável “item”, juntamente do método `getId()`. Para continuar, utiliza-se o operador lógico `or` para realizar a verificação de se a opção “everyItem” da variável “ui” está ativada através do método `isOn()`, além do uso do operador lógico `and` em conjunto com cláusula `not`, para verificar se a variável “item” não é uma mochila. Caso a condição do `else if` seja acatada como verdadeira, é utilizado a propriedade `lootTries` da variável “loot”, atribuindo o valor dela ou o valor numérico 0 somado a 1.

Deve-se também criar um **if**, para realizar a verificar se o tamanho da propriedade **lootTries** é menor que 5, e que caso seja atendida, deverá ser retornada a função **lootItem()** – que será criada futuramente – da tabela “looting”, e que receberá como parâmetro para essa função a tabela “mochilasLooting” e a variável “item”.

```
elseif itemsById[item:getId()] or (ui.everyItem:isOn() and not item:isContainer()) then
    item.lootTries = (item.lootTries or 0) + 1
    if item.lootTries < 5 then
        return TargetBot.Looting.lootItem(mochilasLoot, item)
    end
end
```

Será criado em seguida um **else if**, que será utilizado para verificar se existe algum tipo de comida, armazenada na tabela criada anteriormente para os alimentos. Em seguida será utilizado o operador lógico **and** para verificar se existe algum item no primeiro índice desta tabela, novamente será utilizado o operador lógico **and** para verificar se o valor da variável “ultimoAlimento” somado ao valor numérico 5000 é menor que a variável contém o valor do momento atual. Caso as condições sejam atendidas, será criado um **for** que irá percorrer de maneira sequencial a tabela que armazena os alimentos.

Logo após será criado um **if** onde será verificado se o id da variável “item” é igual ao id da variável alimento que havia sido definida no **for**. Caso as condições sejam atendidas, será utilizado o método **g_game.use()**, como parâmetro, para esse método será utilizado a variável “item”. Também será atribuído a variável “ultimoAlimento” o valor da variável que possui o valor do momento atual. Para essa atribuição o **return** retorna nulo.

```
elseif storage.foodItems and storage.foodItems[1] and ultimaFood + 5000 < now then
    for _, food in ipairs(storage.foodItems) do
        if item:getId() == food.id then
            g_game.use(item)
            ultimaFood = now
            return
        end
    end
end
end
end
```

Em seguida será criado um **if** que verificará se a variável “proximaMochila” possui um valor diferente de nulo. Será atribuído à propriedade **lootTries** da variável “proximaMochila” a seguinte expressão: **(proxMochila.lootTries or 0) + 1**. Logo após será criado um segundo **if** para verificar se a propriedade **lootTries** da “proximaMochila” se ela é menor que 2. Caso a condição seja verdadeira, será utilizado o método **g_game.open()**, como parâmetro para esse método será passado a variável “proximaMochila” e a variável “container”, em seguida será atribuído a variável “espereAte” o valor da variável que contém o valor do momento atual somado ao valor numérico 300. Para a variável “esperaMochila” será atribuído o valor do método **getid()**, sendo usado na variável “proximaMochila”. Em seguida, será criado um **return** que retornará nulo.

Após isso será atribuído o valor booleano **false** para a variável “lootContainer” do objeto **lootContainer**. Em seguida será utilizado o método **g_game.close()**, que terá como parâmetro o objeto **container**. Em seguida será

utilizado o comando **table.remove** que terá como parâmetro a tabela “lista” da tabela “looting” e o valor numérico 1.

```
if proxMochila then
    proxMochila.lootTries = (proxMochila.lootTries or 0) + 1
    if proxMochila.lootTries < 2 then
        g_game.open(proxMochila, container)
        waitTill = now + 300
        esperaMochila = proxMochila:getId()
        return
    end
end
container.lootContainer = false
g_game.close(container)
table.remove(TargetBot.Looting.list, 1)
end
```

Em seguida será definido a função **lootItem()** para a tabela “looting” e como parâmetro para esta função será passado a tabela “mochilasLoot” e a variável “item”. Será criado em seguida um **if** que verificará se o item é empilhável através do método **isStackable()**. Caso a condição seja atendida será criada a variável “contar”, que servirá como contador, será atribuída a essa variável o valor do retorno do método **getCount()** que será utilizada na variável “item”.

Logo após será criado um **for**, onde será criado uma variável “container”, o **for** percorrerá a tabela “mochilasLoot” de maneira sequencial. Em seguida será criado um segundo **for**, onde a variável “contaltem”, que servirá como contador de itens será criada, o **for** percorrerá de maneira sequencial a variável “container” juntamente do método **getItems()**. Dentro deste segundo **for** será criado um **if** que verifica se o id da variável “item” é igual ao id da variável “contaltem”, em seguida será utilizado o operador lógico **and**, utilizando também o método **getCount()** na variável “contaltem”, onde será verificado se ela possui o valor numérico menor que 100. Caso as condições sejam verdadeiras será utilizado o método **g_game.move()**, que terá como parâmetro as variáveis “item” e “container” junto do método **getSlotPosition()**, onde será passado o índice do segundo **for** menos 1. Como último parâmetro para o método **g_game.move()** será passado a variável “contar”.

Em seguida será atribuído a variável “espereAte” o valor da variável que contém o valor do momento atual somado ao valor numérico 300. Depois será criado um **return** que retornará nulo.

```
TargetBot.Looting.lootItem = function(mochilasLoot, item)
    if item:isStackable() then
        local count = item:getCount()
        for _, container in ipairs(mochilasLoot) do
            for slot, citem in ipairs(container:getItems()) do
                if item:getId() == citem:getId() and citem:getCount() < 100 then
                    g_game.move(item, container:getSlotPosition(slot - 1), count)
                    waitTill = now + 300
                    return
                end
            end
        end
    end
end
```

Em seguida será criada uma variável local “container”, será atribuído a ela o valor do primeiro índice da tabela “mochilasLoot”, também será utilizado o método **g_game.move()** que terá como parâmetro a variável “item”, a variável container junto do método **getSlotPosition()**, como parâmetro para este método será utilizado a variável “container” junto com o método **getItemsCount()**. Por fim, para o método **g_game.move()** será passado o valor numérico 1. Em seguida será atribuído a variável “espereAte” o valor da variável que contém o valor do momento atual somado ao valor numérico 300.

```
local container = mochilasLoot[1]
g_game.move(item, container:getSlotPosition(container:getItemsCount()), 1)
waitTill = now + 300
end
```

Logo após será utilizado a função **onContainerOpen()**, como parâmetro para esta função será passada outra função, que terá como parâmetro as variáveis “container” e “previousContainer”. Dentro da segunda função será criado um **if**, onde será verificado se a variável “container”, juntamente dos métodos **getContainerItem()** e **getId()** possuem um valor igual ao da variável “esperaMochila”. E então, será definido a propriedade **lootContainer** da variável “container” como **true**, em seguida será definido a variável “esperaMochila” com o valor “nil”.

```
onContainerOpen(function(container, previousContainer)
    if container:getContainerItem():getId() == esperaMochila then
        container.lootContainer = true
        esperaMochila = nil
    end
end)
```

Em seguida será utilizada uma função **onCreatureDisappear()**, que terá como parâmetro uma outra função, esta segunda função terá como parâmetro o objeto **creature**. Dentro da função será criado um **if** com cláusula **not** e será verificado se o objeto **targetBot** está ativo através do método **isOn()**. O **if** não terá bloco de comando, além de um **return** que servirá apenas para parar a execução do código, caso o **targetBot** esteja desativado.

Logo após será criado outro **if** com cláusula **not** e será verificado se o objeto **creature** está visível ou se é uma criatura através do método **isMonster()**. O **if** não terá bloco de comando, além de um **return** que servirá para sair da função.

```
onCreatureDisappear(function(creature)
    if not TargetBot.isOn() then return end
    if not creature:isMonster() then return end
```

Será criada uma variável local que receberá o nome de “configuracao”, para essa variável será atribuído o valor da função **calculaParametro()** para esta função será passado o objeto **creature** e uma tabela vazia, após isso será definido um **if** com a cláusula **not** e para este **if** será usado a propriedade **config** da variável “configuracao”, será usado o operador lógico **or** e o ajuste “dontLoot”

que pertence a propriedade **config** da variável de “configuracao”, este **if**, assim como os anteriores, ele não possuirá um bloco de comandos e terá um **return** que servirá para sair da função.

```
if not config.config or config.config.dontLoot then
    return
end
```

Três variáveis locais serão criadas, sendo a primeira a variável “posicao” que será atribuído o valor do método **getPosition()**, sendo usado no objeto **player**, a segunda variável será a “mobposicao” que receberá o mesmo valor do método **getPosition()**, sendo usada no objeto **creature**, a terceira variável será a “nome”, que possuirá o valor do método **getName()**, sendo usada no mesmo objeto.

```
local pos = player:getPosition()
local mpos = creature:getPosition()
local nome = creature:getName()
```

Um novo **if** será criado, onde será verificado se o jogador da criatura está na mesma camada “z”, além disso, será calculado se a distância vertical entre o jogador e a criatura é maior que seis, caso uma dessas condições seja verdadeira, será utilizado um **return** para sair da função.

```
if pos.z ~= mpos.z or math.max(math.abs(pos.x-mpos.x), math.abs(pos.y-mpos.y)) > 6 then
    return
end
```

Agora será gerado um agendamento de função de atraso de 20 milissegundos, em seguida um **if** com uma cláusula **not** será criado com o intuito de verificar se a tabela “mochila” em seu primeiro índice possui um valor diferente de nulo, se esta condição for atendida, será utilizado um **return** para sair da função. Um novo **if** será desenvolvido, com o intuito de verificar se existe um elemento na posição ou no índice 20 da tabela “lista” que pertence a tabela “looting”, se existir, um **return** será usado para sair da função.

```
schedule(20, function()
    if not mochilas[1] then return end
    if TargetBot.Looting.list[20] then return end
```

Uma variável chamada “piso” será criada que terá como valor o retorno do método **g_map.getTile()** possuindo como parâmetro a variável “mobPosicao”, em seguida um **if** com a cláusula **not** será criado e o mesmo servirá para verificar se a variável “piso” possuirá um valor válido, caso não possua, um **return** será usado para sair da função.

```
local tile = g_map.getTile(mpos)
if not tile then return end
```

Uma variável local “container” será desenvolvida, que receberá o valor do método **getTopUseThing()** sendo utilizado na variável “piso”. Um **if** com a

cláusula **not** será criado que servirá para verificar se a variável “container” possui um **container** válido, caso esta variável não possua um **container** válido, será possível utilizar um **return** para sair da função.

```
local container = tile:getTopUseThing()
if not container or not container:isContainer() then return end
```

Em seguinte, um **if** com uma cláusula **not** será gerado que servirá para verificar se existe um caminho acessível entre a posição atual do jogador e a posição da criatura, com a distância máxima de equivalente ao valor 6, se não houver um caminho válido, será utilizado um **return** para sair da função. Após isso, um **table.insert** será utilizado para inserir na tabela “lista” o seguinte comando:

```
{posicao=mobPosicao, creature=nome, container=container:getId(),
added=now, tries=0}
container:setMarked('#000088')
```

```
3 if not findPath(player:getPosition(), mpos, 6, {ignoreNonPathable=true,
- ignoreCreatures=true, ignoreCost=true}) then
- return
- end
3 table.insert(TargetBot.Looting.list, {pos=mpos, creature=nome,
- container=container:getId(), added=now, tries=0})
- container:setMarked('#000088')
- end)
- end)
```

Será necessário inserir o diretório desse repositório juntamente aos demais arquivos do **boot**.

Criando as funções andar para o targetbot

É necessário criar um novo arquivo com a extensão **.lua** e um nome que o criador consiga associar que ele conterá funções relacionadas ao deslocamento do personagem. De início, serão definidas três variáveis, a primeira se chamará “destino”, a segundo “distanciaMaxima” e a terceira será nomeada “parametro”.

Em seguida, será definida uma função chamada **andarAte()** e três variáveis criadas serão passadas como argumentos privados, nessa função iremos atribuir o valor desses argumentos às variáveis correspondentes criadas anteriormente.

```
local dest
local maxDist
local parametros

TargetBot.walkTo = function(_dest, _maxDist, _parametros)
- dest = _dest
- maxDist = _maxDist
- parametros = _parametros
end
```

Será criada uma função **andar()**, será criado também um **if** com a cláusula **not** e será verificado se a variável “destino” não possui um valor vazio, caso o valor seja vazio será usado um **return** para sair da função.

Após isso, será criado um novo **if** e nele será verificado se o jogador já está andando através do método **isWalking()** que será usado no objeto **player**, caso o jogador já esteja andando é utilizado um **return** para sair da função.

```
]TargetBot.walk = function()  
  if not dest then return end  
  if player:isWalking() then return end
```

Será criada uma variável local denominada “posicao”, ela terá como valor o método **getPosition()** sendo utilizado no objeto **player**. Em seguida, será verificado se a posição atual do jogador e seu destino estão na mesma camada, caso não estejam é usado um **return** para sair da função.

Uma variável local chamada “distancia” será criada, ela terá como valor o cálculo da distância horizontal máxima entre a posição atual do jogador e o seu destino.

```
local pos = player:getPosition()  
if pos.z ~= dest.z then return end  
local dist = math.max(math.abs(pos.x-dest.x), math.abs(pos.y-dest.y))
```

Nesse momento, será criado um **if** que verificará se a propriedade **precision** da variável “parametro” foi definida e se possui um valor maior ou igual da variável “distancia”, caso essa condição seja atendida é utilizado um **return** para sair da função.

Um segundo **if** será iniciado, ele irá verificar se as propriedades **marginMin** e **marginMax** da variável “parametros” foram definidas, se esse caso for verdadeiro se iniciará um terceiro **if**. Esse terceiro **if** irá verificar se a variável “distancia” possui um valor maior ou igual ao da propriedade **marginMin** e se ela possui um valor menor ou igual ao da segunda propriedade, caso essas condições sejam atendidas a função será interrompida pelo **return**.

```
if parametros.precision and parametros.precision >= dist then return end  
if parametros.marginMin and parametros.marginMax then  
  if dist >= parametros.marginMin and dist <= parametros.marginMax then  
    return  
  end  
end
```

Será criada uma tabela “caminho”, nela será atribuído o valor do método **getPath()**, como parâmetro para ela serão passadas as variáveis “posicao”, “destino”, “distanciaMaxima” e “parametro”. Logo após, será criado um **if** para verificar se essa tabela está definida, caso esteja será utilizada a função **andar()** e será passada como parâmetro a tabela “caminho” no índice 1.

```
local caminho = getPath(pos, dest, maxDist, parametros)
if caminho then
    walk(caminho[1])
end
end
```

O diretório deste arquivo precisará ser adicionado ao arquivo que carregará o boot.

Finalizando o TargetBot

É necessário criar um novo arquivo com extensão .lua, ele será responsável por definir as funções responsáveis pelos ataques do jogador.

Serão definidas sete variáveis locais chamadas: “targetBotMacro”, que terá o valor “nil” atribuído; “config”, que também terá o valor “nil” atribuído; “ultimaAcao”, essa terá atribuído o valor 0; “caveBotPermissao”, com valor numérico 0; “lureAtivado”, que terá o valor **true**; “configWidget”, que terá o valor do método **UI.Config()**; e por último a variável “ui”, que receberá o valor do método **UI.CreateWidget()** que terá como parâmetro a string “targetBotPainel”.

```
local targetbotMacro = nil
local config = nil
local lastAction = 0
local cavebotAllowance = 0
local lureEnabled = true

local configWidget = UI.Config()
local ui = UI.createWidget("TargetBotPanel")
```

Será atribuído o valor do atributo **list** do widget “UI.ListPainel” a variável “ui.List”. O valor da variável “ui.List” será atribuído a variável “targetList” do objeto **TargetBot**, após isso, será utilizado o método **setup()** da tabela “looting”.

```
ui.list = ui.listPanel.list
TargetBot.targetList = ui.list
TargetBot.Looting.setup()
```

Agora as seguintes strings serão definidas: “status” e “off”, para os lados esquerdo e direito, respectivamente, no widget “status” da variável “ui”. Em seguida as strings “target:” e a string “-”, para os lados esquerdo e direito respectivamente do widget “target:”. Em diante as strings “config:” e “-” também

para os lados direito e esquerdo do widget “config”, por fim será definido as strings “perigo:” e “0” para os lados esquerdo e direito do widget “perigo”.

```
ui.status.left:setText("Status:")
ui.status.right:setText("Off")
ui.target.left:setText("Target:")
ui.target.right:setText("-")
ui.config.left:setText("Config:")
ui.config.right:setText("-")
ui.perigo.left:setText("Perigo:")
ui.perigo.right:setText("0")
```

Será definido um evento **onClick** para o botão “debug” do widget “editor”, em seguida será criado uma variável local, onde será atribuído a mesma o valor do método o **isOn()**, sendo utilizado no botão “debug”, logo será utilizado o método **setOn()** no botão “debug” e assim com o parâmetro será passado a cláusula **not** e a variável “on”.

Agora um **if** será iniciado, no mesmo será verificado se a variável “on” foi definida, caso esta condição seja atendida um **for** será criado, com o intuito de percorrer de maneira sequênciã o retorno do método **getSpectators()**, dentro deste **for** cada retorno será usado a função **clearText()**.

```
ui.editor.debug.onClick = function()
  local on = ui.editor.debug:isOn()
  ui.editor.debug:setOn(not on)
  if on then
    for _, spec in ipairs(getSpectators()) do
      spec:clearText()
    end
  end
end
```

Nesse momento, será definido um macro que irá se repetir a cada 100 milissegundos, ele será atribuído a variável “targetBotMacro”, além disso, será passada como parâmetro para ele uma função. Após isso, será definida uma variável local chamada “posicao”, ela receberá o valor do objeto **player** usando o método **getPosition()**.

Em seqüência, será definida uma variável local “criaturas” e será usado um método **g_map.getSpectatorsInRange()**, como parâmetro para esse método será passada a variável “posicao” com o valor booleano **false** e também passará duas vezes o valor numérico 6.

```
targetbotMacro = macro(100, function()
  local pos = player:getPosition()
  local creatures = g_map.getSpectatorsInRange(pos, false, 6, 6)
```

Um **if** será criado, nele será verificado se a variável “criaturas” possui um valor maior que 10, caso essa condição seja verdadeira será repetida a definição da variável “criatura” substituindo o valor numérico 6 por 3.

```
if #creatures > 10 then
    creatures = g_map.getSpectatorsInRange(pos, false, 3, 3)
end
```

Quatro variáveis locais serão criadas, sendo chamadas de: “maiorPrioridade”, “nivelPerigo”, “alvos” e “maiorPrioridadeParametro”.

```
local maiorPrioridade = 0
local perigoLevel = 0
local targets = 0
local maiorPrioridadeParametros = nil
```

Agora, será criado um **for**, que percorrerá de maneira sequencial a lista de criaturas da variável “criaturas”. Em seguida, uma variável local denominada “caminho” será criada, o método **findPath()** será atribuído a ela, além disso, o objeto **player** será passado como argumento para esse método juntamente ao método **getPosition()**. Após isso, o objeto **creature** será passado com o método **getPosition()** como argumento para o primeiro método mencionado, para esse mesmo método, também serão passados como argumentos o valor numérico 7 e a seguinte tabela: “{ignoreLastCreature=true,ignoreNonPathable=true, ignoreCost=true}”

```
for i, creature in ipairs(creatures) do
    local caminho = findPath(player:getPosition(), creature:getPosition(),
        7, {ignoreLastCreature=true, ignoreNonPathable=true, ignoreCost=true})
```

Após isso, um **if** será criado, nele será usado o método **isMonster()** para verificar se o objeto **creature** é uma criatura válida. Nesse mesmo **if**, será utilizado o operador lógico **and** para verificar se a variável “caminho” foi definida, se essa condição **for** atendida será criada uma variável local chamada “parametros” e o método **calculaParametros()**, que terá como argumentos o objeto **criatura** e a variável “caminho”, será atribuído a ela.

A variável “nivelPerigo” terá somado ao seu valor o valor da propriedade **perigo** da variável “parametros”

```
if creature:isMonster() and caminho then
    local parametros = TargetBot.Creature.calculaParametros(creature, caminho)
    perigoLevel = perigoLevel + parametros.perigo
```

Um **if** será criado novamente, nele será verificado se a propriedade **prioridade** da variável “parametros” é maior que 0, caso essa condição seja atendida a variável “alvos” será incrementada em +1. Será definido um terceiro **if**, ele verificará se a propriedade **prioridade** da variável “parametros” é maior que a variável “maiorPrioridade”, caso isso seja atendido o valor da variável “maiorPrioridade” será igualado ao valor da propriedade **pioridade** da variável

“parametros”. Além disso, o valor da variável “maiorPrioridadesParametros” será igualado ao da variável “parametros”

```
if parametros.prioridade > 0 then
    targets = targets + 1
    if parametros.prioridade > maiorPrioridade then
        maiorPrioridade = parametros.prioridade
        maiorPrioridadeParametros = parametros
    end
end
```

Será criado, nesse momento, um quarto **if**, que irá verificar se o debug do painel editor está ativado, isso será feito através do método **isOn()**, caso essa condição seja verdadeira será usado o método **setText()** no objeto **creature**. Como parâmetro para o método **setText()** será passado o campo “name” da propriedade **config** da variável “parâmetro” concatenando através de ponto duplo com a string “\n”, concatenando novamente através de ponto duplo com a propriedade **prioridade** da variável “parametro”. Feito isso, será utilizada a função **andarAte()** sendo passado como argumento o valor “nil”.

```
if ui.editor.debug:isOn() then
    creature:setText(parametros.config.name .. "\n" .. parametros.prioridade)
end
end
end
end

TargetBot.walkTo(nil)
```

Em seguida será definida uma variável local, denominada “looting”, onde será atribuído o método **process()** da tabela “looting”, como argumento para o método será utilizado a variável “alvos” e a variável “nivelPerigo”. Logo após será criado a variável “lootingStatus”, sendo atribuído o método **getStatus()** que pertence a tabela “looting”.

```
local looting = TargetBot.Looting.process(targets, perigoLevel)
local lootingStatus = TargetBot.Looting.getStatus()
```

Após isso será utilizado o método **setText()** passando a variável “nivelPerigo” como argumento no lado direito do painel perigo que foi criado anteriormente. Em seguida será criado um **if** onde será verificado se a variável “maiorPrioridadeParametros” existe, em sequência será utilizado o operador lógico **and**, juntamente com a cláusula **not** e o método **isInPz()**. Caso a condição seja atendida será utilizado o método **getName()** na propriedade **creature** da variável “maiorPrioridadeParametros”, para utilizar como argumento no método **setText()** que irá preencher o lado direito do painel de alvo.

Em seguida, será utilizado o campo “name” da propriedade **config** da variável “maiorPrioridadeParametros” como argumento para o método **setText()**, que irá preencher o lado direito do painel de configuração. Após isso, será passado as variáveis “maiorPrioridadeParametros”, “alvos” e “looting”, para a função ataque que foi criada na segunda etapa do Case 02.

```
ui.perigo.right:setText(perigoLevel)
if maiorPrioridadeParametros and not isInPz() then
    ui.target.right:setText(maiorPrioridadeParametros.creature:getName())
    ui.config.right:setText(maiorPrioridadeParametros.config.name)
    TargetBot.Creature.attack(maiorPrioridadeParametros, targets, looting)
```

Após isso, será criado um **if** onde será verificado se a string da variável “lootingStatus” é maior que zero. Caso a condição seja verdadeira sera utilizado o método **setStatus()** no objeto **targetBot**, que terá como argumento a string “ataque &” concatenado por ponto duplo (..) com a variável “lootingStatus”. Em seguida, será criado um **else if** para verificar se a variável “caveBotPermissao” possui um valor maior que a variável que contém o valor do momento atual. Caso a condição seja atendida, será utilizado o método **setStatus()** para o objeto **targetBot** e será passado a seguinte string como argumento: “Lurando usando CaveBot”. O citado **caveBot** será desenvolvido mais a frente.

Em seguida, será definido o **else** que terá em sua estrutura o método **setStatus()** sendo utilizado do objeto **targetBot** e conterà a seguinte string como argumento: “Atacando”. Ainda dentro do **else** será criado um **if** juntamente da cláusula **not**, onde será verificado se a variável “lureAtivo” está definida. Caso a condição seja atendida, será utilizado o método **setStatus()** no objeto **targetBot**, onde conterà a seguinte string como argumento: “Atacando (luring off)”.

Fora das estruturas de repetição será utilizado a função **andar()** no objeto **targetBot** que vai definir o valor da variável “ultimaAcao”, sendo ele, o valor da variável que contém o valor do momento atual. Em seguida, será utilizado um **return** para sair da função.

```
1   if lootingStatus:len() > 0 then
2       TargetBot.setStatus("Ataque & " .. lootingStatus)
3   elseif cavebotAllowance > now then
4       TargetBot.setStatus("Lurando usando CaveBot")
5   else
6       TargetBot.setStatus("Atacando")
7       if not lureEnabled then
8           TargetBot.setStatus("Atacando (luring off)")
9       end
10      end
11      end
12      TargetBot.walk()
13      lastAction = now
14      return
15  end
```

Logo após, será utilizado o método **setText()** que terá como argumento uma string “-” para preencher o lado direito do painel de alvo. A mesma coisa será feita para o painel de configuração. Em seguida, será criado um **if** que verificará se a variável “looting” tem um valor diferente de nulo e caso a condição seja

atendida, será utilizado a função **andar()** no objeto **targetBot** e a variável “ultimaAcao” receberá o valor da variável que contém o valor do momento atual.

```
ui.target.right:setText("-")
ui.config.right:setText("-")
if looting then
    TargetBot.walk()
    lastAction = now
end
```

Após isso, será criado um **if** que verificará se o tamanho da variável “lootingStatus” é maior que zero. Caso a condição seja verdadeira será utilizado o método **setStatus()** contendo como argumento a variável “lootingStatus” para o objeto **targetBot**. Em seguida, será definido um **else**, onde será passado um string, dentro dela estará escrito “Esperando”. A string será usada como argumento para o método **setStatus()** e será usada no objeto **targetBot**.

```
----
if lootingStatus:len() > 0 then
    TargetBot.setStatus(lootingStatus)
else
    TargetBot.setStatus("Esperando")
end
end)
```

Em seguida, será definido como valor para a variável “config” a função **setup()**, que terá como parâmetros a string “targetBot_configs”, a variável “configWidget”, a string “json” e uma função que terá como parâmetro uma variável “name”, uma variável “ativado” e uma variável “data”. Um **if** com uma cláusula **not** será criado, para verificar se a variável “data” tem um valor diferente de nulo. Caso a condição seja atendida será utilizado o método **setText()** que passará uma string “off”, o método será utilizado para preencher o lado direito do painel “status”. Em sequência, será retornado o macro “targetBotMacro”, juntamente do método **setOff()**.

```
config = Config.setup("targetbot_configs", configWidget, "json", function(name, enabled, data)
    if not data then
        ui.status.right:setText("Off")
        return targetbotMacro.setOff()
    end
end
```

Após isso, será utilizado a função **resetConfigs()** e em sequência será criado um **for**, onde será percorrido de maneira sequencial os valores da variável “data” ou então ele percorrerá uma tabela vazia. Dentro do **for** será utilizado o método **addConfig()** e como argumento será passado a variável que é iterada no **for**.

```
for _, value in ipairs(data["targeting"] or {}) do
    TargetBot.Creature.addConfig(value)
end
```

Logo após será utilizado a função **update()** que pertence a tabela “looting” e como argumento será passado a tabela “data” juntamente da string “looting”, o

operador lógico **or** e em sequência será passado uma tabela vazia. Em seguida, será criado um **if** que verificará se a variável “ativado” está definida e possui um valor diferente de nulo. Caso a condição seja atendida será utilizado o método **setText()** com a string “on” para preencher o lado direito do painel “status”. Em seguida será criado um **else** repetindo os mesmos comando, substituindo apenas a string “on” pela string “off”.

```
TargetBot.Looting.update (data["looting"] or {})  
  
if enabled then  
  ui.status.right:setText ("On")  
else  
  ui.status.right:setText ("Off")  
end
```

Após isso será utilizado o método **setOn()** e terá como argumento a variável “ativado”, o método será utilizado no macro “targetBotMacro”. Em seguida será definido o valor “nil” para a propriedade **delay** do macro “targetBotMacro” e depois será definido o valor **true** para a variável “lureAtivo”.

```
targetbotMacro.setOn(enabled)  
targetbotMacro.delay = nil  
lureEnabled = true  
end)
```

Em seguida será definido um evento **onClick** para o botão “add” do painel editor, dentro deste evento será utilizado uma função **edit()** que terá como parâmetro o valor “nil” e uma função que terá como parâmetro a variável “novaConfig”, dentro desta função teremos outra função chamada **addConfig()**, que terá como argumento as variáveis “novaConfig” e o valor booleano **true**, em sequência será utilizado a função **save()**.

```
ui.editor.buttons.add.onClick = function()  
  TargetBot.Creature.edit(nil, function(novaConfig)  
    TargetBot.Creature.addConfig(novaConfig, true)  
    TargetBot.save()  
  end)  
end
```

Em seguida será criado um evento **onClick** para o botão “edit” do painel editor, logo após será criado uma variável local “entrada” que terá como atribuição o valor do método **getFocusedChild()** sendo usado a propriedade **list** da variável “ui”. Logo em seguida será criado um **if** com a cláusula **not** e será passado a variável “entrada” para verificar se ela não possui um valor definido. Caso a condição seja atendida será utilizado um **return** para sair da função.

Logo após será utilizado a função **edit()** que terá como argumento a propriedade **value** da variável “entrada” e uma função que terá como parâmetro a variável “novaConfig”. Em seguida será utilizado o método **setText()** na variável de entrada, onde terá como argumento a propriedade **name** da variável “novaConfig”, em sequência a propriedade **value** da variável “entrada” receberá

o valor da variável “novaConfig”. Também será utilizado a função **resetConfigsCache()** e a função **save()**.

```
ui.editor.buttons.edit.onClick = function()
  local entry = ui.list:getFocusedChild()
  if not entry then return end
  TargetBot.Creature.edit(entry.value, function(novaConfig)
    entry.setText(novaConfig.name)
    entry.value = novaConfig
    TargetBot.Creature.resetConfigsCache()
    TargetBot.save()
  end)
end
```

Em seguida será criado o evento **onClick** para o botão “remove”, logo após será criado uma variável local “entrada” que terá como atribuição o valor do método **getFocusedChild()** sendo usado a propriedade **list** da variável “ui”. Logo em seguida será criado um **if** com a cláusula **not** e será passado a variável “entrada” para verificar se ela não possui um valor definido. Caso a condição seja atendida será utilizado um **return** para sair da função.

Logo após será utilizado o método **destroy()** na variável “entrada” e também será utilizado a função **resetConfigsCache()** e a função **save()**.

```
ui.editor.buttons.remove.onClick = function()
  local entry = ui.list:getFocusedChild()
  if not entry then return end
  entry.destroy()
  TargetBot.Creature.resetConfigsCache()
  TargetBot.save()
end
```

Em seguida será definido uma função **isActive()** e dentro desta função será retornado a comparação entre a variável “ultimaAcao” somada ao valor 300 e a variável que contém o valor do momento atual.

```
TargetBot.isActive = function()
  return lastAction + 300 > now
end
```

Logo após será definido uma função, chamada **isCaveBotActionAllowed()**, onde retornará a verificação se a variável “caveBotPermissao” é maior que a variável que contém o valor do momento atual.

```
TargetBot.isCaveBotActionAllowed = function()
  return cavebotAllowance > now
end
```

Em sequência será definido uma função **setStatus()** que terá como parâmetro uma variável “text”, a função retornará o método **setTexte()** com a variável “text” sendo utilizado no lado direito do painel “status”.

```
TargetBot.setStatus = function(text)
  return ui.status.right.setText(text)
end
```

Em seguida será definido uma função **isOn()** que retornará a variável “config” junto do método **isOn()**.

```
TargetBot.isOn = function()
  return config.isOn()
end
```

O mesmo será feito para a função **isOff()**.

```
TargetBot.isOff = function()
  return config.isOff()
end
```

Em seguida será definido a função **setOn()** que terá como parâmetro a variável “valor”, dentro da função será criado um **if** que verificará se a variável “valor” possui um valor igual a **false**. Caso a condição seja atendida terá um **return** que retornará a função **setOff()** que terá como argumento o valor **true**, caso contrário será utilizado a função **setOn()** na variável “config”.

```
TargetBot.setOn = function(val)
  if val == false then
    return TargetBot.setOff(true)
  end
  config.setOn()
end
```

O mesmo será feito para a função **setOff()**.

```
TargetBot.setOff = function(val)
  if val == false then
    return TargetBot.setOn(true)
  end
  config.setOff()
end
```

Em seguida será definido a função **delay()** que terá como parâmetro a variável “value”. A função será usada no macro “targetBotMacro”, atribuindo o valor da variável que contém o valor do momento atual, que será nomeada como “agora”, essa variável já foi citada anteriormente, somada ao valor da variável “value”.

```
TargetBot.delay = function(value)
  targetbotMacro.delay = now + value
end
```

Logo após será definido uma função **save()** onde será criado uma variável local "data" que terá como valor uma tabela que terá dois campos. O primeiro campo será o "targeting" que terá como valor uma tabela vazia e o segundo campo será "looting" que terá como valor uma tabela vazia também. Em seguida será criado um **for** onde será percorrido de maneira sequencial a propriedade **list** da variável "ui", onde também será usado o método **getChildren()**. Na estrutura deste **for** a variável que vai o iterar deverá ser uma variável chamada "entrada".

Dentro do mesmo **for** será utilizado um **table.insert**, como argumento será utilizado a tabela "targeting" da tabela "data" e a propriedade **value** da variável "entrada". Em sequência será utilizado a função **save()** na tabela "looting" do objeto **targetBot** e será passado como argumento a tabela "looting" da tabela "data". Depois disso, será utilizado a função **save()** na variável "config" e será passado a tabela "data".

```
TargetBot.save = function()
  local data = {targeting={}, looting={}}
  for _, entry in ipairs(ui.list:getChildren()) do
    table.insert(data.targeting, entry.value)
  end
  TargetBot.Looting.save(data.looting)
  config.save(data)
end
```

Em seguida será definida uma função **allowCaveBot()** que terá como parâmetro uma cláusula **time**, dentro desta função será atribuído a variável "caveBotPermissao" o valor da soma entre a variável "agora" e a cláusula **time**.

```
TargetBot.allowCaveBot = function(time)
  cavebotAllowance = now + time
end
```

Logo após será criada uma função **disableLuring()**, dentro desta função será atribuído o valor **false** para a variável "lureAtivo".

```
TargetBot.disableLuring = function()
  lureEnabled = false
end
```

O mesmo será feito para a função **enable()**.

```
TargetBot.enableLuring = function()
  lureEnabled = true
end
```

Em seguida será definido duas variáveis locais. A primeira terá o nome de “ultimaSpell” e terá valor zero, a segunda terá o nome de “ultimaAtaqueSpell” e também receberá o valor zero.

```
local ultimaSpell = 0
local ultimaAtaqueSpell = 0
```

Em seguida será definida uma função **saySpell()** que terá como parâmetro as variáveis “text” e “delay”. Dentro desta função será criado um **if** que verificará se o tipo da variável “text” é diferente de string ou se o tamanho da variável “text” é menor que 1. Caso as condições sejam atendidas, será utilizado um **return** para sair da função. Será criado um **if** com a cláusula **not** que verificará se a variável “delay” foi definida, caso a condição seja atendida a variável receberá o valor 500.

Logo após será criado um **if** onde verificará se o retorno do método **g_game.getProtocolVersion()** é menor que 1090. Caso a condição seja atendida, a variável “ultimaAtaqueSpell” irá receber o valor da variável “agora”. Em seguida será criado mais um **if**, que verificará se a variável “ultimaSpell” somada a variável “delay” é menor que o valor da variável “agora”, caso a condição seja atendida, será utilizado o método **say()** que terá como argumento a variável “text” e a variável “ultimaSpell” receberá o valor da variável “agora”, o **if** retornará o valor **true**. Caso a condição não seja atendida, será retornado o valor **false**.

```
TargetBot.saySpell = function(text, delay)
  if type(text) ~= 'string' or text:len() < 1 then return end
  if not delay then delay = 500 end
  if g_game.getProtocolVersion() < 1090 then
    ultimaAtaqueSpell = now
  end
  if ultimaSpell + delay < now then
    say(text)
    ultimaSpell = now
    return true
  end
  return false
end
```

Em seguida será definido uma função **sayAttackSpell()** que terá como argumento as variáveis “text” e “delay”. Dentro desta função será criado um **if** que verificará se o tipo da variável “text” é diferente de string ou se o tamanho da variável “text” é menor que 1. Caso as condições sejam atendidas, será utilizado um **return** para sair da função. Será criado um **if** com a cláusula **not** que verificará se a variável “delay” foi definida, caso a condição seja atendida a variável receberá o valor 2000.

Logo após será criado outro **if** que irá verificar se a variável “ultimaAtaqueSpell” somada a variável “delay” é menor que a variável “agora”. Caso a condição seja atendida será usado o método **say()** que terá como argumento a variável “text”, a variável “ultimaAtaqueSpell” receberá o valor da

variável “agora” e será retornado o valor **true**. Caso contrário será retornado o valor **false**.

```
]TargetBot.sayAttackSpell = function(text, delay)
  if type(text) ~= 'string' or text:len() < 1 then return end
  if not delay then delay = 2000 end
] if ultimaAtaqueSpell + delay < now then
  say(text)
  ultimaAtaqueSpell = now
  return true
- end
- return false
-end
```

Em seguida será criado duas variáveis locais, a primeira será “ultimoItem” e receberá valor zero, a segunda será “ultimaRuna” e receberá o valor zero. Logo após será definido uma função **useItem()** que terá como parâmetro as variáveis “item”, “subType”, “alvo” e “delay”. Em sequência será criado um **if** com cláusula **not** que verificará se a variável “delay” não possui um valor nulo. Caso a condição seja verdadeira, a variável “delay” receberá o valor 200.

Outro **if** será criado que verificará se a soma da variável “ultimoItem” e da variável “delay” e menor que a variável “agora”. Caso a condição seja verdadeira será criado uma variável local “coisa” que terá como valor o método **g_things.getThingType()** que terá como argumento a variável “item”. Em seguida, será criado um **if** com a cláusula **not** para verificar se a variável “coisa” não possui um valor nulo. Será utilizado o operador lógico **or** e a cláusula **not** e na segunda condição, será utilizado o método **isFluidContainer()** na variável “coisa”. Caso as condições sejam verdadeiras a variável “subType” vai receber o valor da seguinte expressão: **g_game.getClientVersion() >= 860 and 0 or 1**.

```
]TargetBot.useItem = function(item, subType, target, delay)
  if not delay then delay = 200 end
] if ultimoItem + delay < now then
  local thing = g_things.getThingType(item)
] if not thing or not thing.isFluidContainer() then
  subType = g_game.getClientVersion() >= 860 and 0 or 1
- end
```

Em seguida será criado um **if** que verificará se o retorno do método **g_game.getClientVersion()** é menor que 780, caso a condição seja verdadeira será criado uma variável local “tempoltem” que receberá o valor do método **g_game.findPlayerItem()** que terá como argumento as variáveis “item” e “subType”. Logo após será criado um **if not** para verificar se a variável “tempoltem” não foi criada, caso a condição seja atendida, será utilizado um **return** para sair da função.

Em sequência será utilizado o método **g_game.useWith()** que terá como argumento as variáveis “tempoltem”, “alvo” e “subType”. Será criado um **else** onde terá dentro o método **g_game.useInventoryItemWith()** que terá como argumento as variáveis “item”, “alvo” e “subType”.

Após isso, a variável “ultimoItem” receberá o valor da variável agora.

```
if g_game.getClientVersion() < 780 then
    local tmpItem = g_game.findPlayerItem(item, subType)
    if not tmpItem then return end
    g_game.useWith(tmpItem, target, subType)
else
    g_game.useInventoryItemWith(item, target, subType)
end
ultimoItem = now
end
end
```

Em seguida será definido uma função **useAttackItem()** que será igual à função anterior substituindo a variável “ultimoItem” pela variável “ultimaRuna” e o valor do delay será 2000.

```
TargetBot.useAttackItem = function(item, subType, target, delay)
    if not delay then delay = 2000 end
    if ultimaRuna + delay < now then
        local thing = g_things.getThingType(item)
        if not thing or not thing.isFluidContainer() then
            subType = g_game.getClientVersion() >= 860 and 0 or 1
        end
    end
    if g_game.getClientVersion() < 780 then
        local tmpItem = g_game.findPlayerItem(item, subType)
        if not tmpItem then return end
        g_game.useWith(tmpItem, target, subType)
    else
        g_game.useInventoryItemWith(item, target, subType)
    end
    ultimaRuna = now
end
end
```

Em sequência será definido uma função **canLure()** que retornará a variável “lureAtivo”.

```
TargetBot.canLure = function()
    return lureEnabled
end
```

É necessário adicionar o diretório do arquivo juntamente aos demais diretórios dos arquivos do **bot**.

CASE 02.1

Criando a interface

Nesta parte do case 02, será criado os estilos para serem importados pela interface gráfica do bot, isto é necessário para que não ocorra nenhum tipo de bug visual que resulte no não funcionamento do bot.

Será necessário criar um novo arquivo, desta vez com a extensão .otui (extensão gráfica que é lida pelo OT Client), poderá ser usado a IDE Geany para a confecção do mesmo. E para facilitar coloque o nome deste arquivo exatamente igual ao do arquivo criado na Programação do Sistema de Loot.

De início, será criado um panel que será usado como a janela principal, ele precisa receber como nome a string "TargetBotLootingPanel". Para definir como layout para esse panel, será definido para ele o tipo **verticalBox**, ele terá uma propriedade chamada **fit-children** com o valor **true**.

```
TargetBotLootingPanel < Panel
  layout:
    type: verticalBox
    fit-children: true
```

Será criada uma separação horizontal, nela será colocada uma margem de valor 5px no topo.

```
HorizontalSeparator
  margin-top: 5
```

Uma label será criada, ela terá como propriedades a margem do topo igual ao item anterior, terá o alinhamento central para o texto e deverá conter o texto "itens para lootear".

```
Label
  margin-top: 5
  text: Itens para lootear
  text-align: center
```

Em seguida será definido um container de botões que irá receber o id "items" e terá o valor 3px para a margem do topo.

```
BotContainer
  id: items
  margin-top: 3
```

Será criado um botão de alternância, ele deve conter o id “everyItem”, uma margem no topo de 2px e o texto “lootear todos os itens”.

```
BotSwitch
  id: everyItem
  !text: tr("Lootear todos os itens")
  margin-top: 2
```

Sequencialmente será criada uma nova label com as mesmas propriedades da anterior e deverá conter o texto “BPs para loot”.

```
Label
  margin-top: 5
  text: BPs para Loot
  text-align: center
```

Outro container será criado, ele receberá o id “mochilas”, terá a margem do topo com o valor 3px e uma altura de 45px.

```
BotContainer
  id: mochilas
  margin-top: 3
  height: 45
```

Mais um painel será criado, ele receberá o id “maxDangerPanel”, terá uma altura de 20px e uma margem de topo de 5px.

```
Panel
  id: maxDangerPanel
  height: 20
  margin-top: 5
```

Agora será definido o texto do botão “edit”, esse botão receberá o id “value”, uma ancoragem à direita, uma ancoragem no topo e outra no canto inferior, ele terá essas ancoragens em relação a um elemento pai, além dessas propriedades, ele terá uma margem à direita de 6px e uma largura de 80px.

```
BotTextEdit
  id: value
  anchors.right: parent.right
  anchors.top: parent.top
  anchors.bottom: parent.bottom
  margin-right: 6
  width: 80
```

Mais um painel será criado, ele será exatamente igual ao painel anterior e terá como id "minCapacityPanel".

```
Panel
  id: minCapacityPanel
  height: 20
  margin-top: 3
```

Outro texto para o botão "edit" será criado exatamente igual ao anterior.

```
BotTextEdit
  id: value
  anchors.right: parent.right
  anchors.top: parent.top
  anchors.bottom: parent.bottom
  margin-right: 6
  width: 80
```

Agora uma label será criada, ela estará ancorada a parte esquerda e ao centro vertical do elemento pai, ela conterà uma margem a esquerda de 5px e o seguinte texto "Min. capacidade:".

```
Label
  anchors.left: parent.left
  anchors.verticalCenter: prev.verticalCenter
  text: Min. capacidade:
  margin-left: 5
```

Novamente uma label será criada, essa terá uma margem superior de 3px, uma à esquerda de 20px e uma à direita de também 20px. Além disso, ela terá o alinhamento central para o texto, um ajuste automático do tamanho do texto, haverá uma quebra de linha do mesmo e deverá conter o seguinte texto "Arraste a BP ou clique em qualquer slot vazio".

```
Label
  margin-top: 3
  margin-left: 20
  margin-right: 20
  !text: tr("Arraste a BP ou clique em qualquer slot vazio")
  text-align: center
  text-wrap: true
  text-auto-resize: true
```

Este arquivo .otui está finalizado, e assim como foi feito anteriormente deve-se colocar o arquivo junto aos demais arquivos do bot, porém não deve-se usar o comando **importStyle()** e adicionar o diretório deste arquivo, no arquivo que será responsável por carregar o bot.

Exemplo:

```
importStyle("/targetbot/looting.otui")
importStyle("/targetbot/target.otui")
importStyle("/targetbot/creature_editor.otui")
```

Desenvolvendo o menu suspenso

Deve-se criar um novo arquivo com extensão .otui, e colocar o mesmo nome do arquivo que foi desenvolvido na parte “Finalizando o Targetbot” do Case 02.

De início será feita uma nova label que receberá o título “TargetBotEntry”, a mesma possuirá um background color com o valor alfa, além de também possui um deslocamento horizontal. Além de tudo, essa label pode receber um foco.

```
TargetBotEntry < Label
  background-color: alpha
  text-offset: 2 0
  focusable: true
```

Em seguida, será definido um estilo para o foco, onde quando algo estiver em foco fará com que o background color mude para #00000055.

```
$focus:
  background-color: #00000055
```

Após isso, será definido um painel que será nomeado como “TargetBotDualLabel”, terá como altura 18px, a margem a esquerda de 3px e uma margem à direita de 4px.

```
TargetBotDualLabel < Panel
  height: 18
  margin-left: 3
  margin-right: 4
```

Uma nova label será criada e a mesma receberá o id “left”, além disso a mesma possuirá uma ancoragem ao topo e a esquerda do elemento pai, contudo também há um auto ajuste de texto.

```
Label
  id: left
  anchors.top: parent.top
  anchors.left: parent.left
  text-auto-resize: true
```

Agora será criada uma label exatamente igual, mas para o lado direito.

```
Label
  id: right
  anchors.top: parent.top
  anchors.right: parent.right
  text-auto-resize: true
```

Sequencialmente será criado um painel que será nomeado como “TargetBotPanel”, como layout do tipo **verticalbox** ele terá uma propriedade chamada **fit-children** com o valor **true**.

```
TargetBotPanel < Panel
  layout:
    type: verticalBox
    fit-children: true
```

Em seguida uma separação horizontal será feita e terá uma margem superior de 2px e uma margem inferior de 5px.

```
HorizontalSeparator
  margin-top: 2
  margin-bottom: 5
```

Serão feitas quatro instâncias da classe “TargetBotDualLabel” que terão os seguintes ids: “status”, “target”, “config” e “perigo”.

```
TargetBotDualLabel
  id: status
TargetBotDualLabel
  id: target
TargetBotDualLabel
  id: config
TargetBotDualLabel
  id: perigo
```

Agora mais um painel será definido com o id nomeado como “listPanel” que possuirá uma altura de 40px.

```
Panel
  id: listPanel
  height: 40
```

Uma lista será criada, a mesma receberá o id “list” que terá uma ancoragem para preencher o elemento pai, também possuirá uma barra de rolagem vertical e uma margem à direita de 15px, não poderá receber foco e o primeiro item desta lista ganhará foco.

```
TextList
  id: list
  anchors.fill: parent
  vertical-scrollbar: listScrollbar
  margin-right: 15
  focusable: false
  auto-focus: first
```

Será definido a barra de rolagem vertical que receberá o id “listScrollbar” e possuirá uma ancoragem inferior, superior e a direita do elemento pai, além de possuir um tamanho de rolagem de 10px e a rolagem da barra será em pixels.

```
VerticalScrollbar
  id: listScrollbar
  anchors.top: parent.top
  anchors.bottom: parent.bottom
  anchors.right: parent.right
  pixels-scroll: true
  step: 10
```

Será criado um botão de alternância que possuirá o id “configbutton” e ele terá um evento **onClick()** como manipulador, onde quando o botão é clicado ele terá seu estado entre ativado e desativado. A altura do painel pai é ajustada dependendo do estado do botão e quando o botão estiver ativado o texto que aparecerá no mesmo será “esconder o editor de Target” e quando desativado exibirá o texto “mostrar o editor de Target”.

```
BotSwitch
  id: configButton
  @onClick: |
    self:setOn(not self.isOn())
    self:getParent().listPanel:setHeight(self.isOn() and 100 or 40)
    self:getParent().editor:setVisible(self.isOn())

  $on:
    text: Esconder o Editor de Targets

  $!on:
    text: Mostrar o Editor de Targets
```

Um painel será criado e receberá o id “editor”, terá sua visibilidade desativada e como layout será do tipo **verticalbox** terá uma propriedade chamada **fit-children** com o valor **true**.

```
Panel
  id: editor
  visible: false
  layout:
    type: verticalBox
    fit-children: true
```

Sequencialmente será iniciado mais um painel que receberá o id “buttons” e possuirá como altura 20px e uma margem superior de 2px.

```
Panel
  id: buttons
  height: 20
  margin-top: 2
```

Em seguida, será criado um button que receberá o ip “add” e terá uma ancoragem superior, inferior e a esquerda do elemento pai irá conter um texto “add” e terá uma largura de 56px.

```
Button
  id: add
  anchors.top: parent.top
  anchors.bottom: parent.bottom
  anchors.left: parent.left
  text: Add
  width: 56
```

Seguindo adiante, será criado um button agora nomeado com o id “edit” e com uma ancoragem inferior, superior e no centro vertical do elemento pai, contendo o texto “edit” e terá 56px de largura.

```
Button
  id: edit
  anchors.top: parent.top
  anchors.bottom: parent.bottom
  anchors.horizontalCenter: parent.horizontalCenter
  text: Edit
  width: 56
```

Agora mais um button será criado com o id “remove” e terá uma ancoragem superior, inferior e a direita do elemento pai, conterá um texto “remove” com uma largura de 56px.

```
Button
  id: remove
  anchors.top: parent.top
  anchors.bottom: parent.bottom
  anchors.right: parent.right
  text: Remove
  width: 56
```

Um botão de alternância será criado e receberá o id “debug” e irá conter o texto “Mostrar Prioridade dos Target”.

```
BotSwitch
  id: debug
  text: Mostrar Prioridade dos Target
```

Este arquivo também deverá ter seu diretório sendo atribuído a um comando **importStyle**, no arquivo responsável por carregar o bot.

Finalizando a interface

Agora será necessário criar novamente um arquivo com extensão .otui, e ele deverá receber o mesmo nome do arquivo criado na etapa “Editor da Lista de Criaturas” do Case 02.

Será criado um painel que deverá conter o nome “TargetBotCreatureEditorScrollBar” que terá uma margem superior de 3px e uma altura de 28px.

```
TargetBotCreatureEditorScrollBar < Panel
  height: 28
  margin-top: 3
```

Em seguida será criado uma label que terá o id “text”, o texto centralizado e uma ancoragem superior à direita e à esquerda do elemento pai.

```
Label
  id: text
  anchors.left: parent.left
  anchors.right: parent.right
  anchors.top: parent.top
  text-align: center
```

Logo após será criado uma barra de rolagem horizontal que deverá conter o id “scroll”, uma ancoragem superior à direita e à esquerda do elemento pai. Terá uma margem superior de 3px, um tamanho mínimo de 0px, um tamanho máximo de 10px e definirá o passo da barra de rolagem como 1.

```
HorizontalScrollBar
  id: scroll
  anchors.left: parent.left
  anchors.right: parent.right
  anchors.top: prev.bottom
  margin-top: 3
  minimum: 0
  maximum: 10
  step: 1
```

Em seguida, será criado outro painel que conterá o título “TargetBotCreatureEditorTextEdit”, o painel deverá ter 40 px de altura e uma margem superior de 7px.

```
TargetBotCreatureEditorTextEdit < Panel
  height: 40
  margin-top: 7
```


Em sequência será criada uma label que conterá o id “text”, uma ancoragem superior à direita e à esquerda do elemento pai e o alinhamento do texto será no centro.

```
Label
  id: text
  anchors.left: parent.left
  anchors.right: parent.right
  anchors.top: parent.top
  text-align: center
```

Também será definido um editor de texto que receberá o id “textEdit”, uma ancoragem inferior a direita e à esquerda do elemento pai, uma margem superior de 5px, um tamanho mínimo de 0px, um tamanho máximo de 10px e a propriedade step terá o valor 1.

```
TextEdit
  id: textEdit
  anchors.left: parent.left
  anchors.right: parent.right
  anchors.top: prev.bottom
  margin-top: 5
  minimum: 0
  maximum: 10
  step: 1
```

Logo após será criado um painel que receberá o nome de “TargetBotCreatureEditorItem” que terá uma altura de 34px, uma margem superior de 7px e uma margem à direita e à esquerda de 25px.

```
TargetBotCreatureEditorItem < Panel
  height: 34
  margin-top: 7
  margin-left: 25
  margin-right: 25
```

Em seguida será criada uma label que terá o id “text”, terá uma ancoragem à esquerda e uma ancoragem no eixo vertical do elemento pai.

```
Label
  id: text
  anchors.left: parent.left
  anchors.verticalCenter: next.verticalCenter
```

Logo após será criado um elemento item que receberá o id “item”, terá uma ancoragem superior e a direita do elemento pai.

```
BotItem
  id: item
  anchors.top: parent.top
  anchors.right: parent.right
```

Em seguida, será criado um botão de alternância que receberá o nome “TargetBotCreatureEditorCheckBox”, terá 20px de altura e uma margem superior de 7px.

```
TargetBotCreatureEditorCheckBox < BotSwitch
  height: 20
  margin-top: 7
```

Em sequência será criado uma janela principal que receberá o nome de “TargetBotCreatureEditorWindow”, terá como texto “TargetBot creature editor”, uma altura de 630px e uma largura de 500px. Será alterado a altura da janela para 300px toda vez que ela for movida.

```
TargetBotCreatureEditorWindow < MainWindow
  text: TargetBot creature editor
  width: 500
  height: 630

  $mobile:
    height: 300
```

Em seguida será criado uma label que terá uma ancoragem a esquerda e a direita superior do elemento pai, terá o alinhamento do texto no centro e conterá o texto “Você pode usar * (qualquer caractere) e ? (qualquer caractere) no nome do alvo”.

```
Label
  anchors.left: parent.left
  anchors.right: parent.right
  anchors.top: parent.top
  text-align: center
  !text: tr('Você pode usar * (qualquer caractere) e ? (qualquer caractere) no nome do alvo')
```

Em seguida, será criado outra label que é idêntica à anterior mudando apenas o texto, que será: “Você também pode inserir vários alvos, separá-los por ,”.

```
Label
  anchors.left: parent.left
  anchors.right: parent.right
  anchors.top: prev.bottom
  text-align: center
  !text: tr('Você também pode inserir vários alvos, separá-los por ,')
```

Logo após será criado um editor de texto que receberá o id **name**, terá uma ancoragem superior a direita e à esquerda do elemento pai, uma margem a esquerda de 90px e uma margem superior de 5px.

```
TextEdit
  id: name
  anchors.top: prev.bottom
  anchors.left: parent.left
  anchors.right: parent.right
  margin-left: 90
  margin-top: 5
```

Em seguida será criada uma label que terá uma ancoragem ao centro do eixo vertical e a direita do elemento pai, terá também o texto “Nome do alvo: “

```
Label
  anchors.verticalCenter: prev.verticalCenter
  anchors.left: parent.left
  text: Target name:
```

Em sequência será criada uma barra de rolagem vertical que terá o id “contentScroll”, uma ancoragem ao topo do elemento com o id “name” e uma ancoragem à direita do elemento pai. A propriedade step terá o valor 28, um scroll em pixels, uma margem à direita de -10px, uma margem superior de 5px e uma margem inferior de 5px.

```
VerticalScrollBar
  id: contentScroll
  anchors.top: name.bottom
  anchors.right: parent.right
  step: 28
  pixels-scroll: true
  margin-right: -10
  margin-top: 5
  margin-bottom: 5
```

Em seguida será criado um painel que poderá ser movido com o scroll, um id “content”, terá uma ancoragem superior a quem tiver o id “name”, terá uma ancoragem à esquerda e à direita do elemento pai. A barra de rolagem vertical do painel será a que contém o id “contentScroll” e terá uma margem inferior de 10px.

```
ScrollablePanel
  id: content
  anchors.top: name.bottom
  anchors.left: parent.left
  anchors.right: parent.right
  vertical-scrollbar: contentScroll
  margin-bottom: 10
```

Logo após será criado um painel que terá o id “left”, terá uma ancoragem à esquerda superior do elemento pai, uma ancoragem do lado direito do elemento que está centralizado no eixo vertical do elemento pai, terá uma margem superior de 5px, uma margem à direita de 10px e uma margem a esquerda de 10px. Na criação do layout será do tipo **verticalBox** e terá a sua propriedade **fit-children** como **true**.

```
Panel
  id: left
  anchors.top: parent.top
  anchors.left: parent.left
  anchors.right: parent.horizontalCenter
  margin-top: 5
  margin-left: 10
  margin-right: 10
  layout:
    type: verticalBox
    fit-children: true
```

Em seguida será criado um painel idêntico ao anterior, mas para o lado direito.

```
Panel
  id: right
  anchors.top: parent.top
  anchors.left: parent.horizontalCenter
  anchors.right: parent.right
  margin-top: 5
  margin-left: 10
  margin-right: 10
  layout:
    type: verticalBox
    fit-children: true
```

Em seguida será criado um botão que terá como id “ok”, conterà o texto “ok”, será ancorado a parte inferior e a esquerda do elemento pai, uma margem à direita de 10px e uma largura de 60px.

```
Button
  id: ok
  !text: tr('Ok')
  anchors.bottom: parent.bottom
  anchors.left: parent.left
  margin-right: 10
  width: 60
```

Em seguida, será criado um segundo botão que terá o id “cancel”, conterà o texto “cancel”, terá um alinhamento inferior e a direita do elemento pai e uma largura de 60px.

```
Button
  id: cancel
  !text: tr('Cancel')
  anchors.bottom: parent.bottom
  anchors.right: parent.right
  width: 60
```

CASE 03

Iniciando o CaveBot

Neste case será desenvolvido a parte do bot que ficará responsável pelo controle do deslocamento do personagem.

Para começar será necessário criar um novo arquivo com extensão .lua, e o mesmo deverá ser nomeado de maneira que seja possível entender que se remete a ações que o bot executará

Será definida uma tabela vazia para a propriedade “actions” do objeto **CaveBot**. Em seguida uma função **addAcao()** será definida para o objeto **CaveBot**, onde terá como argumento uma variável “action”, uma variável “value” e por fim uma variável “focus”, dentro desta função será atribuída a variável “action”, o valor dela mesma, utilizando o método “lower”. Sequencialmente será definida uma nova variável local chamada “reacao” e a mesma será atribuída a tabela “actions”, tendo como índice a variável “action”.

```
CaveBot.Actions = {}  
  
CaveBot.addAcao = function(action, value, focus)  
    action = action:lower()  
    local reacao = CaveBot.Actions[action]
```

Em sequência, um **if not** será criado e será passada a variável “reacao” como condição, caso essa condição seja atendida, será retornado o método **erro()** e como parâmetro desse método será passada uma string escrita “acao invalida” e a variável “action”.

```
if not reacao then  
    return error("Invalid cavebot action: " .. action)  
end
```

Um **if** será desenvolvido, com o intuito de verificar se a variável “value” é do tipo “number”, caso essa condição seja atendida, essa variável será convertida para a string.

```
if type(value) == 'number' then  
    value = tostring(value)  
end
```

Agora uma variável local nomeada “widget” será criada e possuirá como valor o método **createWidget()** do objeto **ui** e terá como parâmetro a string “CaveBotAction” e a propriedade **actionList** do objeto **CaveBot**. O método **setText()** será usado na variável “widget”, seguidamente método **setText()** com os parâmetros a variável “action” seguido de uma string “:” e então será passada como parâmetro a variável “value” juntamente ao método **split()**, que possuirá como parâmetro a string “\n”, sequencialmente o índice de número um será passado.

Será atribuído à propriedade **action** do “widget” para a variável “action”, em sequência será atribuído o valor da variável “value”, a propriedade **value** do “widget”.

```
local widget = UI.createWidget("CaveBotAction", CaveBot.actionList)  
widget:setText(action .. ":" .. value:split("\n")[1])  
widget.action = action  
widget.value = value
```

Prontamente será definido um **if** onde será verificado se a propriedade **color** da variável “reacao” existe, caso essa condição seja verdadeira, essa propriedade será atribuída ao “widget”.

```
if reacao.color then
  widget:setColor(reacao.color)
end
```

Um evento de clique duplo será criado para o “widget” e será passado como parâmetro uma variável “cwidget”, sequencialmente um **if** será desenvolvido para verificar se a propriedade **editor** do objeto **CaveBot** existe, se esta condição for verdadeira, será criado um “schedule” e como parâmetro será passado o valor “20” e uma função, onde será utilizado o método **edit()** na propriedade **editor** e será passado como parâmetro as propriedades **action** e **value** da variável “cwidget, além de ser passado uma função que terá como parâmetro as variáveis “action” e “value”.

Sequencialmente uma função chamada **editacao()** que terá como parâmetro as variáveis “cwidget”, “action” e “value”, em seguida a função **save()** do **CaveBot** será utilizada.

```
widget.onDoubleClick = function(cwidget)
  if CaveBot.Editor then
    schedule(20, function()
      CaveBot.Editor.edit(cwidget.action, cwidget.value, function(action, value)
        CaveBot.editAcao(cwidget, action, value)
        CaveBot.save()
      end)
    end)
  end
end
```

Será criado um novo **if** com o intuito de verificar se a variável “focus” existe, caso esta condição seja atendida, será utilizado o método **focus()** na variável “widget” e em seguida será utilizado o método **ensureChildVisible()** com a variável “widget” como parâmetro na propriedade **actionList** do **CaveBot**, dessa forma o “widget” será retornado.

```
if focus then
  widget:focus()
  CaveBot.actionList:ensureChildVisible(widget)
end
return widget
end
```

Será definida uma função **editAcao()** para o objeto **CaveBot** que terá como parâmetro as variáveis “widget”, “action” e “value”. Após isso, será atribuída a variável “action” o valor dela mesma com o método **lower()**, sequencialmente será definida uma variável local “reacao” que terá como valor a tabela “actions” com o índice “action”.

```
CaveBot.editAcao = function(widget, action, value)
    action = action:lower()
    local reacao = CaveBot.Actions[action]
    ..
end
```

Um **if not** será criado para verificar a existência da variável “reacao” e caso essa condição seja atendida será retornado o método **error()** contando como argumento a string “acao invalida” e a variável “action”.

```
if not reacao then
    return error("Invalid cavebot action: " .. action)
end
```

Em sequência, será criado um **if not** que verificará a existência das propriedades **action** e **value** da variável “widget”, caso essa condição seja verdadeira, será retornado o método **error()** com a seguinte string “acao invalida para o widget, algum valor foi perdido”.

```
if not widget.action or not widget.value then
    return error("Invalid cavebot action widget, has missing action or value")
end
```

Nessa parte, um processo já descrito anteriormente será repetido, uma variável local nomeada “widget” será criada e possuirá como valor o método **createWidget()** do objeto **ui** e terá como parâmetro a string “CaveBotAction” e a propriedade **actionList** do objeto **CaveBot**. O método **setText()** será usado na variável “widget”, seguidamente método **setText()** com os parâmetros a variável “action” seguido de uma string “:” e então será passada como parâmetro a variável “value” juntamente ao método **split()**, que possuirá como parâmetro a string “\n”, sequencialmente o índice de número um será passado.

Será atribuído à propriedade **action** do “widget” para a variável “action”, em sequência será atribuído o valor da variável “value”, a propriedade **value** do “widget”.

```
widget:setText(action .. ":" .. value:split("\n")[1])
widget.action = action
widget.value = value
if reacao.color then
    widget:setColor(reacao.color)
end
return widget
end
```

Será definido uma função **registrarAcao()** que terá como parâmetro três variáveis: “action”, “color” e “callback”. Após isso será atribuída a variável “action” o valor dela mesma e o método **lower()**.

Em seguida, será criado um **if** que verificará se a tabela “actions” com o índice “action” existe. Caso a condição seja verdadeira, será retornado um erro através do método **error()** que terá como argumento a string “acao duplicada” e a variável “action”. Caso a condição não seja atendida a tabela “actions” com índice “action” será preenchida atribuindo o valor da variável “color” a propriedade **color** e o mesmo para a variável “callback”.

```
CaveBot.registrarAcao = function(action, color, callback)
  action = action:lower()
  if CaveBot.Actions[action] then
    return error("Duplicated action: " .. action)
  end
  CaveBot.Actions[action] = {
    color=color,
    callback=callback
  }
end
```

Após isso será utilizado a função **registrarAcao()**, onde será passado como argumento uma string escrito “label”, uma segunda string escrito “yellow” e uma função. A função terá como parâmetro três variáveis: “value”, “tentativas” e “prev”, retornando o valor **true**. O processo será repetido com os parâmetros das duas primeiras string serão **gotolabel()** e no lugar da string “yellow” será utilizado um código hexadecimal #ffff55 e no return será utilizado o método **gotolabel()** que terá como parâmetro a variável “value” da função.

```
CaveBot.registrarAcao("label", "yellow", function(value, retries, prev)
  return true
end)

CaveBot.registrarAcao("gotolabel", "#FFFF55", function(value, retries, prev)
  return CaveBot.gotoLabel(value)
end)
```

Em seguida será utilizado novamente a função **registrarAcao()** que terá como parâmetro a string “delay”, o código #aaaaaa e uma função exatamente igual às usadas anteriormente. Após será criado um **if** que verificará se a variável “tentativas” é igual a zero e caso a condição seja atendida, o valor da variável “value” será convertido para valor numérico na propriedade **delay** do objeto **CaveBot**. Será criado também um return que retornará uma string “tente novamente”. Caso a condição não seja atendida será retornado o valor booleano **true**.

```
CaveBot.registrarAcao("delay", "#AAAAAA", function(value, retries, prev)
  if retries == 0 then
    CaveBot.delay(tonumber(value))
    return "retry"
  end
  return true
end)
```


Será utilizado novamente a função **registrarAcao()** que terá como parametro uma string “função”, uma string “red”, uma função igual às anteriores, criando também uma variável local chamada “prefixo” que receberá o valor da seguinte string: "local retries = " .. tentativas .. "\nlocal prev = " .. tostring(prev) .. "\nlocal delay = CaveBot.delay\nlocal gotoLabel = CaveBot.gotoLabel\n". Será atribuído à variável “prefixo” o valor dela mesma concatenada por ponto duplo com a seguinte string: "local macro = function() error('Macros inside cavebot functions are not allowed') end\n".

```
CaveBot.registrarAcao("function", "red", function(value, retries, prev)
  local prefix = "local retries = " .. tentativas .. "\nlocal prev = " .. tostring(prev) ..
  "\nlocal delay = CaveBot.delay\nlocal gotoLabel = CaveBot.gotoLabel\n"
  prefix = prefix .. "local macro = function() error('Macros inside cavebot functions are not allowed') end\n"
end)
```

Em seguida será criado um **for** onde será definido uma variável extensão para percorrer os **callbacks** em pares do campo **extensions** do objeto caveBot, dentro do **for** será atribuído a variável prefixo o seguinte valor prefixo .. "local " .. extensão .. " = CaveBot.Extensions." .. extensão .. "\n".

```
for extension, callbacks in pairs(CaveBot.Extensions) do
  prefix = prefix .. "local " .. extension .. " = CaveBot.Extensions." .. extension .. "\n"
end
```

Serão criadas duas variáveis locais chamadas de “status” e “resultado”, onde armazenarão um método **pcall()** que irá chamar uma função que terá um return para o método **assert()**, esse terá um método **load()** dentro dele que terá como parâmetro a variável “prefixo” concatenada por ponto duplo a variável “value” e uma string escrita “funcao”.

```
local status, resultado = pcall(function()
  return assert(load(prefix .. value, "cavebot_function"))()
end)
```

Em seguida será criado um **if not** que terá como condição a variável “status” e caso a condição seja atendida será utilizado o método **error()** que terá como parâmetro a string “erro na função:\n” concatenada por ponto duplo a variável “resultado”, será retornado o valor booleano **false**. Caso a condição não seja atendida a variável “resultado” será retornada.

```
if not status then
  error("Error in cavebot function:\n" .. resultado)
  return false
end
return resultado
end)
```

Em seguida será utilizado novamente a função **registrarAcao()** com o argumento será passado a string escrito “go to”, uma string escrito “green” e uma função igual as anteriores. Feito isso, será criado uma variável local “posicao” que será igual ao valor do método **regexMatch()** que terá como

argumento uma variável “value” e em sequência a seguinte string: “\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+),?\s*([0-9]?)”.

Em sequência será feito um **if not** onde será passado a variável “condição” com o índice 1. Caso a condição seja verdadeira será passado o método **error()** com a string “valor de ação inválida, deveria ser a posição: “, passará a variável “value” como parâmetro, que deverá retornar o valor **false**.

```
]CaveBot.registrarAcao("goto", "green", function(value, retries, prev)
  local posicao = regexMatch(value, "\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+),?\s*([0-9]?)"
] if not posicao[1] then
  error("Invalid cavebot goto action value. It should be position (x,y,z), is: " .. value)
  return false
end
```

Em seguida será criado um **if**, onde será utilizado o método **get()** na variável “config” do objeto **CaveBot**, será passada a string “mapclick”, em seguida será criado outro **if** que verificará se a variável “tentativas” possui um valor maior ou igual a 5. Se a condição for verdadeira será retornado o valor false. Logo após será criado um **else**, que terá um **if** para verificar se a variável “tentativas” possui um valor maior ou igual a 100. Caso a condição seja verdadeira deverá retornar o valor **false**.

```
if CaveBot.Config.get("mapClick") then
  if retries >= 5 then
    return false
  end
else
  if retries >= 100 then
    return false
  end
end
```

Logo após será criada uma variável “precisao” que terá o valor igual ao valor do método **toNumber()** que terá como parâmetro a variável “posicao” com os índices 1 e 5. Em seguida será preenchida a tabela “posicao” sendo o campo **x** igual ao método **toNumber()** tendo como argumento o método da variável “posicao” com os índices 1 e 2, em sequência o campo **y** usando o mesmo método mas com a variável “posicao” com os índices 1 e 3. Por último o campo **z**, com o mesmo método mas com a variável “posicao” com os índices 1 e 4.

Em seguida será criada uma variável local “playerPosicao”, que receberá o valor do objeto **player** juntamente do método **getPosition()**. Será criado então um **if** que verificará se o campo **z** da tabela “posicao” é diferente do campo **z** da tabela “playerPosicao”. Caso a condição seja verdadeira, será retornado o valor **false**.

```
local precisao = tonumber(posicao[1][5])
posicao = {x=tonumber(posicao[1][2]), y=tonumber(posicao[1][3]), z=tonumber(posicao[1][4])}
local playerPosicao = player:getPosition()
if posicao.z ~= playerPosicao.z then
  return false
end
```

Em seguida será criado outro **if** que terá o método **math.abs()** que terá como parâmetro a diferença entre os campos **x**, das variáveis “posicao” e “playerPosicao”. Logo após será feito uma soma com o método **math.abs()** que passará a diferença entre os campos **y** das mesmas variáveis, será verificado se essa soma será maior 40. Caso a condição seja verdadeira, será retornado o valor **false**.

```
if math.abs(posicao.x-playerPosicao.x) + math.abs(posicao.y-playerPosicao.y) > 40 then
    return false
end
```

Em seguida será criada uma variável local “miniMapColor” que receberá o valor do método **g_map.getMiniMapColor()** que terá como parâmetro a variável “posicao”. Em sequência será criada uma variável local “escadas”, a variável terá como valor a seguinte expressão: **(minimapColor >= 210 and minimapColor <= 213)**. Será criado então um **if** onde será verificado se a variável “escadas” existe. Caso a condição seja verdadeira, será criado outro **if** onde será utilizado o método **math.abs()**, como parâmetro será passado a diferença do campo **x** das variáveis “posicao” e “playerPosicao”, será verificado se o resultado é igual a 0. O operador lógico **and** e o método **math.abs()** serão utilizados para passar a diferença dos campos **y** das tabelas “posicao” e “playerPosicao”, será verificado se o valor é menor ou igual a 0. Caso a condição seja verdadeira, será retornado o valor **true**.

Em seguida, será criado um **else if** que conterá a expressão utilizada no **if** anterior, com exceção da última comparação do método de diferença dos campos **y**, onde o zero será trocado pela variável “precisao” ou pelo valor numérico 1. Caso a condição seja verdadeira, será retornado o valor booleano **true**.

```
if escadas then
    if math.abs(posicao.x-playerPosicao.x) == 0 and math.abs(posicao.y-playerPosicao.y) <= 0 then
        return true
    end
elseif math.abs(posicao.x-playerPosicao.x) == 0 and math.abs(posicao.y-playerPosicao.y) <= (precisao or 1) then
    return true
end
```

Em seguida, será definido uma variável local “caminho” que irá ter o valor atribuído pelo método **findPath()** que terá como parâmetro a variável “playerPosicao”, “posicao”, o valor numérico 40 e a seguinte expressão: **{ ignoreNonPathable = true, precisao = 1, ignoreCreatures = true }**. Em seguida, será criado um **if not**, onde passará a variável “caminho”. Caso a condição seja verdadeira, irá retornar o valor **false**.

```
local caminho = findPath(playerPosicao, posicao, 40, { ignoreNonPathable = true, precisao = 1, ignoreCreatures = true })
if not caminho then
    return false
end
```

Em seguida será criado um **if not**, onde será utilizado o método **get()** da variável “config” do objeto **CaveBot**, para o método será passado como parâmetro a string “ignoreFields”, também será utilizado o operador lógico **and**

e a função **andarAte()** que terá como parâmetro a variável “posicao” e o valor numérico 40. Caso a condição seja verdadeira, será retornado a string “retry”.

```
if not CaveBot.Config.get("ignoreFields") and CaveBot.andarAte(posicao, 40) then
    return "retry"
end
```

Em seguida será criado um **if** onde será utilizado, novamente, a função **andarAte()**, que terá como parâmetro a variável “posicao”, valor numérico 40 e a seguinte expressão: **{ ignoreNonPathable = true }**. Será retornado a string “retry”.

```
if CaveBot.andarAte(posicao, 40, { ignoreNonPathable = true }) then
    return "retry"
end
```

Um **if** será criado, ele irá verificar se a variável “tentativas” possui um valor maior ou igual a 3, caso essa condição seja atendida, será definida uma variável “precisao” que receberá o valor da variável “tentativas” com o decremento de 1. Em seguida, será definido outro **if**, esse verificará a existência da variável “escadas”, no caso dessa condição ser verdadeira, o valor da variável “precisao” será definido como 0. Após esse passo, será definido um terceiro **if**, nele será passada a função **andarAte()** como condição e essa função terá como parâmetro a variável “posicao”, o valor numérico 50 e a seguinte expressão: **{ ignoreNonPathable = true, precisao = precisao }**, com essa condição sendo verdadeira a string “retry” será retornada.

```
if retries >= 3 then
    local precisao = retries - 1
    if escadas then
        precisao = 0
    end
    if CaveBot.andarAte(posicao, 50, { ignoreNonPathable = true, precisao = precisao }) then
        return "retry"
    end
end
```

Um **if not** será definido, o método **get()** será usado na variável “config” do objeto **CaveBot** e a string “mapClick” será passada como parâmetro para esse método. Após isso, o operador lógico **and** será utilizado e irá verificar se a variável “tentativas” possui um valor maior ou igual a 5, caso essa condição seja verdadeira, o valor **false** será retornado.

```
if not CaveBot.Config.get("mapClick") and retries >= 5 then
    return false
end
```

Um novo **if** será definido e o método **get()** na variável “config” será utilizado novamente, como parâmetro para esse método será passada a string “skipBlocked”, com essa condição sendo atendida o valor **false** será retornado.

```
if CaveBot.Config.get("skipBlocked") then
    return false
end
```

A função **andarAte()** será utilizada, será passada como parâmetro para ela a variável “posicao”, o valor numérico 40 e a expressão { **ignoreNonPathable = true, precisao = 1, ignoreCreatures = true** }. Em sequência, será retornada a string “retry”.

```
CaveBot.andarAte(posicao, 40, { ignoreNonPathable = true, precisao = 1, ignoreCreatures = true })  
return "retry"  
end)
```

Será utilizada a função **registrarAcao()** e como argumento para ela será passada uma string “use”, outra string contendo o código hexadecimal #FFB272 e uma função exatamente igual a usada anteriormente. Com isso, será definida uma variável local “posicao”, que terá seu valor local atribuído através do método **regexMatch()**, como argumento para esse método será passada a variável “value” e a string “\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)”. Seguidamente, será criado um **if not**, nele será passada como condição a variável “posicao” no índice 1, com essa condição sendo atendida, será definida uma variável local chamada “itemid”, que terá seu valor atribuído através do método **toNumber()**, esse terá como parâmetro a variável “value”. Após isso, será criado um novo **if not** e nele será passada a variável “itemid”, no caso dela não possuir um valor válido será utilizado o método **error()**, que conterà a string “essa e uma acao invalida, o id do item deveria ser:”, esse string será concatenada por ponto duplo junto da variável “value”, em sequência será retornado o valor **false**. No caso da variável “itemid” possuir um valor válido será usado o método **use()**, que terá como argumento a variável “itemid” e após isso será retornado o valor **true**.

```
CaveBot.registrarAcao("use", "#FFB272", function(value, retries, prev)  
  local posicao = regexMatch(value, "\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)")  
  if not posicao[1] then  
    local itemid = tonumber(value)  
    if not itemid then  
      error("Invalid cavebot use action value. It should be (x,y,z) or item id, is: " .. value)  
      return false  
    end  
    use(itemid)  
    return true  
  end  
end
```

Agora serão definidos valores que irão preencher a tabela “posicao”, para o campo **x** que terá valor igual ao método **toNumber()** tendo como argumento o método da variável “posicao” com os índices 1 e 2, em sequência o campo **y** utilizando o mesmo método mas com a variável “posicao” com os índices 1 e 3. Por último o campo **z**, com o mesmo método mas com a variável “posicao” com os índices 1 e 4.

Em seguida, será criada uma variável local **“playerPosicao”**, que receberá o valor do objeto **player** juntamente do método **getPosition()**. Será criado então um **if** que verificará se o campo **z** da tabela **“posicao”** é diferente do campo **z** da tabela **“playerPosicao”**. Caso a condição seja verdadeira, será retornado o valor **false**.

```
posicao = {x=tonumber(posicao[1][2]), y=tonumber(posicao[1][3]), z=tonumber(posicao[1][4])}
local playerPosicao = player:getPosition()
if posicao.z ~= playerPosicao.z then
    return false
end
```

Após isso, um passo anterior será repetido, será criado outro **if** que terá o método **math.abs()** que terá como parâmetro a diferença entre os campos **x**, das variáveis **“posicao”** e **“playerPosicao”**. Logo após será feita uma soma com o método **math.abs()** que passará a diferença entre os campos **y** das mesmas variáveis, será verificado se essa soma será maior 7. Caso a condição seja verdadeira, será retornado o valor **false**.

```
if math.max(math.abs(posicao.x-playerPosicao.x), math.abs(posicao.y-playerPosicao.y)) > 7 then
    return false
end
```

Sequencialmente, será definida uma variável local chamada **“piso”**, que terá seu valor atribuído através do método **g_map.getTile()**, como parâmetro para esse método será passada a variável **“posicao”**. Um **if not** será definido, será passada como condição para ele a variável **“piso”**, caso essa condição seja verdadeira, o valor **false** será retornado.

```
local piso = g_map.getTile(posicao)
if not piso then
    return false
end
```

Em seguida será criada uma variável local que receberá o nome de **“topThing”**, o valor atribuído será o da variável **“piso”** juntamente do método **getTopUseThing()**. Em sequência será criado um **if not** onde passará a variável **“topThing”**, caso a condição seja verdadeira irá retornar o valor **false**.

```
local topThing = piso:getTopUseThing()
if not topThing then
    return false
end
```

Depois será utilizado o método **use()** que terá como parâmetro a variável **“topThing”**, será utilizado o método **delay()** que terá como parâmetro a soma do método **get()** da variável **“config”** contendo as strings **“useDelay”** e **“ping”**.

```
use(topThing)
CaveBot.delay(CaveBot.Config.get("useDelay") + CaveBot.Config.get("ping"))
return true
end)
```

Em seguida será utilizado a função **registrarAcao()** que terá como argumento a string “usarCom”, terá um código hexadecimal #EEB292. Em sequência terá uma função exatamente igual as usadas anteriormente.

Após isso, será criado a variável local “posicao” que terá como valor o método **regexMatch()** que terá como argumento a variável “value” e a seguinte string. “\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)”. Em sequência será criado um **if not** onde passará a variável “posicao” no índice 1, caso a condição seja atendida será criado outro **if not** onde será verificado a existência da variável “itemid” onde será utilizado o método **error()** que terá como argumento a string “ação invalida devido as coordenadas (x, y e z) ou pelo itemId”. A string será concatenada por ponto duplo com a variável “value”, onde retornará o valor booleano **false**. Caso nenhuma das condições seja atendida será utilizado o método **use()** e será passada a variável “itemid” e retornará o valor booleano **true**.

```
]CaveBot.registrarAcao("usewith", "#EEB292", function(value, retries, prev)
  local posicao = regexMatch(value, "\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)")
] if not posicao[1] then
]   if not itemid then
     error("Invalid cavebot usewith action value. It should be (itemid,x,y,z) or item id, is: " .. value)
     return false
   end
   use(itemid)
   return true
end
```

Em seguida será criado uma variável local “itemid” onde receberá o valor do método **tonumber()**, que terá como parâmetro a variável “posicao” nos índices 1 e 2. O final da função será idêntico a função anterior.

```
local itemid = tonumber(posicao[1][2])
posicao = {x=tonumber(posicao[1][3]), y=tonumber(posicao[1][4]), z=tonumber(posicao[1][5])}
local playerPosicao = player:getPosition()
if posicao.z ~= playerPosicao.z then
  return false
end

if math.max(math.abs(posicao.x-playerPosicao.x), math.abs(posicao.y-playerPosicao.y)) > 7 then
  return false
end

local piso = g_map:getTile(posicao)
if not piso then
  return false
end

local topThing = piso:getTopUseThing()
if not topThing then
  return false
end

usewith(itemid, topThing)
CaveBot.delay(CaveBot.Config.get("useDelay") + CaveBot.Config.get("ping"))
return true
end)
```

Será utilizado novamente a função **registrarAcao()** que terá como parâmetro a string “say” que terá um código hexadecimal #FF55FF, terá também uma função idêntica a utilizada anteriormente. Depois será utilizado o método **say()** que terá como argumento a variável “value” e retornará o valor booleano **true**.

```
CaveBot.registrarAcao("say", "#FF55FF", function(value, retries, prev)
    say(value)
    return true
end)
```

Agora este arquivo foi finalizado, e deve ter seu diretório adicionado ao dos demais arquivos feitos anteriormente, porém dessa vez será necessário criar uma variável local “cave”, usar o método **setDefaultTab()** tendo como argumento a variável, e a criação de duas tabelas vazias, uma chamada de “cavebot” e a outra de “cavebot.extensions”.

```
local cavebotTab = "Cave"
setDefaultTab(cavebotTab)
CaveBot = {}
CaveBot.Extensions = {}
```

Criando o arquivo de Configurações

Este arquivo será responsável por armazenar as configurações feitas na parte do cavebot.

Serão definidas quatro tabelas vazias, uma será a tabela “config”, a segunda será a tabela “value”, a terceira será “default_value” e a quarta será “value_setters”.

```
CaveBot.Config = {}
CaveBot.Config.values = {}
CaveBot.Config.default_values = {}
CaveBot.Config.value_setters = {}
```

Em seguida será definido uma função **setup()** na tabela “config”, sequencialmente será definido um valor para o campo **ui** da tabela “config”, através do método **UI.createWidget()**, que terá como argumento a string “CaveBotConfigPanel”, logo será definido duas variáveis locais, a primeira sendo “ui”, que receberá o valor do campo **ui** da tabela “config” e a segunda será uma variável “add” que receberá o valor do campo **add** da mesma tabela.

```
CaveBot.Config.setup = function()
    CaveBot.Config.ui = UI.createWidget("CaveBotConfigPanel")
    local ui = CaveBot.Config.ui
    local add = CaveBot.Config.add
```


Feito isso, uma função **add()** será usada, a mesma terá como argumento a string “ping”, uma segunda string chamada “server ping” e o valor numérico 100. Após isso, a função **add()** será usada novamente e irá receber os valor conforme é demonstrado na imagem abaixo:

```
add("ping", "Server ping", 100)
add("walkDelay", "Walk delay", 10)
add("mapClick", "Use map click", false)
add("mapClickDelay", "Map click delay", 100)
add("ignoreFields", "Ignore fields", false)
add("skipBlocked", "Skip blocked path", false)
add("useDelay", "Delay after use", 400)
end
```

Agora será definido uma função **show()**, onde será utilizada no método **ui()** da tabela “config”.

```
CaveBot.Config.show = function()
  CaveBot.Config.ui:show()
end
```

Será feita uma função exatamente igual a anteriormente, mas ao invés de mostrar a interface gráfica, a mesma irá esconder a interface gráfica.

```
CaveBot.Config.hide = function()
  CaveBot.Config.ui:hide()
end
```

Sequencialmente, uma nova função **onConfigChange()** será definida, a mesma terá como parâmetros uma variável “configNome”, “ativo” e “configDados”. Em seguida um **for** será definido, com o intuito de percorrer cada valor de maneira sequencial da tabela “default_values” da tabela “config”. Será atribuído cada valor da tabela “default_values” na tabela “value_setters” possuindo como índice o índice do **for**.

```
CaveBot.Config.onConfigChange = function(configNome, Ativo, configData)
  for k, v in pairs(CaveBot.Config.default_values) do
    CaveBot.Config.value_setters[k] (v)
  end
```

Em sequência, um **if not** será criado com o intuito de verificar a variável “configDados”, caso esta variável não exista, haverá um return que será usado para sair da função. Feito isso, mais um **for** será definido que de forma sequencial percorrerá os valores da variável “configDados”, então um novo **if** será criado para verificar a existência da tabela “value_setters” no índice do **for**, com essa condição sendo atendida, um valor será atribuído para essa tabela.

```
] for k, v in pairs(configData) do
]   if CaveBot.Config.value_setters[k] then
      CaveBot.Config.value_setters[k] (v)
    end
  end
end
```

Uma função nomeada **save()** será criada, a mesma deverá retornar para a tabela “values”.

```
☐ CaveBot.Config.save = function()
  return CaveBot.Config.values
end
```

Uma função será criada e definida como **add()**, será passado como parâmetro nessa função como variável “id”, “titulo” e “valorpadrao”, um **if** será criado para verificar se existe algum valor na tabela “values” no índice “id”, caso essa condição seja atendida, será retornado para o método **error()** que irá conter como argumento a string “configuracao duplicada:”, essa string será concatenada por ponto duplo com a variável “id”.

```
☐ CaveBot.Config.add = function(id, title, defaultValue)
☐   if CaveBot.Config.values[id] then
      return error("Duplicated config key: " .. id)
    end
end
```

Será criado uma variável local “panel” e outra variável local “ajuste”, em seguida será criado um **if** que irá verificar se a variável “valorpadrao” é do tipo numérico. A variável “panel” receberá o valor do método **UI.createWidget()**, o método terá como parâmetro a string “CaveBotConfigNumberValuePanel” e em seguida o campo **ui** da variável “config”. Em sequência a variável “ajuste” e receberá uma função que terá como parâmetro uma variável “value”, após isso o campo **values** da tabela “config” receberá o parâmetro **value** e será utilizado o método **setText()** no campo **value** da variável “panel” onde será passado como argumento a variável “value” e o valor **true**.

```
local panel
local setter
☐ if type(defaultValue) == "number" then
  panel = UI.createWidget("CaveBotConfigNumberValuePanel", CaveBot.Config.ui)
☐   setter = function(value)
      CaveBot.Config.values[id] = value
      panel.value:setText(value, true)
    end
end
```

Em seguida será utilizado a função **ajuste()**, definida anteriormente, que terá como argumento a variável “defaultValue”. Será utilizado um método **onTextChange()** no campo **value** da variável “panel”, onde será definida uma função, a função terá como parâmetro a variável “widget” e a variável “novoValor” que será convertida para um tipo numérico através do método **toNumber()**. Em seguida, será criado um **if**, onde será verificado se o valor da variável “novoValor” é diferente de nulo. Caso a condição seja verdadeira será atribuído o valor da variável “novoValor” para o campo **values** do índice **id** da variável “config”, será utilizado o método **save()** em seguida.

```

setter(defaultValue)
panel.value.onTextChange = function(widget, newValue)
  newValue = tonumber(newValue)
  if newValue then
    CaveBot.Config.values[id] = newValue
    CaveBot.save()
  end
end
end

```

Logo após será criado um **else if** onde será verificado se o tipo da variável “defaultValue” é booleano. Caso a condição seja atendida será atribuído a variável “panel” receberá o valor do método **UI.createWidget()**, o método terá como parâmetro a string “CaveBotConfigBooleanValuePanel” e em seguida o campo **ui** da variável “config”. Em sequência a variável “ajuste” e receberá uma função que terá como parâmetro uma variável “value”, após isso o campo **values** da tabela config receberá o parâmetro **value** e será utilizado o método **setText()** no campo value da variável “panel” onde será passado como argumento a variável “value” e o valor **true**.

```

elseif type(defaultValue) == "boolean" then
  panel = UI.createWidget("CaveBotConfigBooleanValuePanel", CaveBot.Config.ui)
  setter = function(value)
    CaveBot.Config.values[id] = value
    panel.value:setOn(value, true)
  end
end

```

Com o final do **else if** será utilizado a função **ajuste()** passando a variável “defaultValues” como parâmetro. Em seguida será definida uma função de evento **onClick()** para o campo **value** da variável “panel”, após isso será utilizado o método **setOn()** na variável “widget” que terá como parâmetro a cláusula **not** e o método **isOn()** na variável “widget”. Em seguida será atribuído o valor do método **isOn()** sendo usado na variável “widget” ao campo **values** no índice **id** que pertence a tabela “config”, após será utilizado o método **save()**. Será definido **else** que conterà um return, que retorna o método **error()** que terá o seguinte texto: “valor padrao da configuração invalido” concatenado a variável “id” por ponto duplo.

```

setter(defaultValue)
panel.value.onClick = function(widget)
  widget:setOn(not widget.isOn())
  CaveBot.Config.values[id] = widget.isOn()
  CaveBot.save()
end
else
  return error("Invalid default value of config for key " .. id .. ", should be number or boolean")
end

```

Em seguida será utilizado o método **setText()** contendo como argumento o método **tr()** que terá como argumento a variável "titulo", sendo concatenado por ponto duplo a uma string que conterá dois pontos. Logo após será definido o valor da variável "ajuste" para a tabela "value_setters" no índice **id**, onde será definido o valor da variável "defaultValues" para a tabela "values" no índice **id** e por fim será definido o valor da variável "defaultValues" para a tabela "defaultValues" no índice **id**.

```
panel.title:setText(tr(title) .. ":")

CaveBot.Config.value_setters[id] = setter
CaveBot.Config.values[id] = defaultValue
CaveBot.Config.default_values[id] = defaultValue
end
```

Em sequência, será definido uma função **get()** que terá como parâmetro a variável "id", criaremos um **if** para verificar se a tabela "values" no índice **id** tem o valor igual a nulo. Caso a condição seja atendida será retornado o método **error()** com a string que conterá o texto "id invalido", caso contrário será retornado a tabela "values" no índice **id**.

```
CaveBot.Config.get = function(id)
if CaveBot.Config.values[id] == nil then
return error("Invalid CaveBot.Config.get, id: " .. id)
end
return CaveBot.Config.values[id]
end
```

Assim como feito anteriormente deverá ser colocado o diretório deste arquivo junto ao diretório dos demais arquivos.

Criando o editor de ações

Nesta etapa será criado um novo arquivo de extensão .lua que deverá remeter ao editor de ações do bot.

Será definido duas tabelas vazias, onde a primeira se chamará "editor" e a segunda se chamará "actions", em seguida será definido uma função **registrarAcao()** e a mesma terá como parâmetro uma variável "action", uma variável "texto" e uma variável "parametro". Será verificado se a variável "texto" é divergente do tipo string, caso a mesma seja divergente, a variável "parametro" receberá o valor da variável "texto" e a variável "texto" receberá o valor da variável "action".

```
CaveBot.Editor = {}
CaveBot.Editor.Actions = {}

CaveBot.Editor.registrarAcao = function(action, text, parametros)
if type(text) ~= 'string' then
parametros = text
text = action
end
```

Uma variável “color” será definida, onde a mesma receberá o valor **nil**, em seguida será verificado se a variável “parametro” é diferente do tipo função, caso a mesma seja diferente do tipo função, será definida uma variável local “reacao” e essa variável “reacao” receberá o valor da tabela “actions” no índice “action”. Será verificado se a variável “reacao” tem o valor igual a nulo, caso essa condição seja verdadeira, o método **error()** será retornado, nele conterá o texto “erro no editor, a acao: ”, em seguida irá concatenar por ponto duplo a variável “action” que também irá concatenar por ponto duplo “nao existe”. Finalizando isso, a tabela “actions” no índice “action” irá receber o valor da variável “parametro” e a variável color receberá a propriedade “color” da variável “reacao”.

```
local color = nil
if type(parametros) ~= 'function' then
    local reacao = CaveBot.Actions[action]
    if not reacao then
        return error("CaveBot editor error: action " .. action .. " doesn't exist")
    end
    CaveBot.Editor.Actions[action] = parametros
    color = reacao.color
end
```

Em sequência, uma variável local “button” será definida e receberá o valor do método **UI.createWidget()** que terá como argumento a string “CaveBotEditorButton”, além do botão da variável “ui”. Sequencialmente será usado o método **setText()** e como parâmetro será passada a variável “texto” e será verificado se a variável “color” possui um valor diferente de nulo, caso seja verdadeiro, será usado o método **setColor()** na variável “button”, onde será passado a variável “color” como argumento.

```
local button = UI.createWidget('CaveBotEditorButton', CaveBot.Editor.ui.buttons)
button:setText(text)
if color then
    button:setColor(color)
end
```

Em seguida, será definido um evento **onClick** e será verificado se a variável “parametro” é do tipo função, caso a mesma seja do tipo função, será usado uma função “parametro”, logo um return será criado mas não irá retornar nada.

```
button.onClick = function()
    if type(parametros) == 'function' then
        parametros()
        return
    end
end
```

Será utilizada uma função **edit()**, como argumento para essa função será usada a variável “action”, o valor nil e uma função, essa segunda função deverá ter como argumento uma variável “action” e uma variável “value”. Em sequência, uma variável local chamada “acaoFoco” será criada para receber o valor do método **getFocusedChild()** sendo usado na variável “actionList” do objeto **CaveBot**, após isso, será definida uma variável “index” que terá como valor o método **getChildCount()** sendo usado na mesma variável “actionList”.

```
CaveBot.Editor.edit(action, nil, function(action, value)
    local acaoFoco = CaveBot.actionList:getFocusedChild()
    local index = CaveBot.actionList:getChildCount()
    --
end)
```

Em seguida um **if** será definido para verificar se a variável “acaoFoco” possui um valor diferente de nulo, caso essa condição seja verdadeira, ela receberá o valor do método **getChildIndex()** que terá como argumento a variável “acaoFoco”.

```
if acaoFoco then
    index = CaveBot.actionList:getChildIndex(acaoFoco)
end
```

Uma variável local “widget” será criada e receberá o valor da função **addAcao()** que terá como argumento a variável “action” e a variável “value”. Sequencialmente será usado o método **moveChildToIndex()** que terá como parâmetro a variável “widget” e a variável “index” sendo incrementada em mais um, esse método será usado na variável “actionList” e na mesma variável “actionList” será usado o método **focusChild()** que possuirá como parâmetro a variável “widget”, em seguida será usado a função **save()** e irá retornar a variável “button”.

```
    local widget = CaveBot.addAcao(action, value)
    CaveBot.actionList:moveChildToIndex(widget, index + 1)
    CaveBot.actionList:focusChild(widget)
    CaveBot.save()
end)
end
return button
end
```

Uma nova função **setup()** será definida, onde será atribuída ao campo **ui** da tabela “editor” o valor do método **UI.CreateWidget()** que possuirá como argumento a string “CaveBotEditorPanel” e em sequência uma variável local “ui” será criada e receberá o valor do campo **ui** da tabela “editor”. Em seguida, uma variável local chamada “registrarAcao” será criada e receberá o valor da função **registrarAcao()**.

```
☐ CaveBot.Editor.setup = function()
    CaveBot.Editor.ui = UI.createWidget("CaveBotEditorPanel")
    local ui = CaveBot.Editor.ui
    local registrarAcao = CaveBot.Editor.registrarAcao
```

Será utilizada a função **registrarAcao()**, a mesma terá como parâmetro uma string “subir” e uma função. Será definida uma variável local chamada “action” que receberá o valor da variável “actionList” juntamente do método **getFocusedChild()**, em seguida será criado um **if not** para verificar o valor da variável “action” e caso o mesmo seja nulo, será usado um return para sair da função. Sequencialmente, uma variável local “index” será criada que terá como valor a variável “actionList” juntamente com o método **getChildIndex()** que terá como parâmetro a variável “action”. Em seguida será verificado por meio de um **if** se a variável “index” possui um valor menor que dois, caso essa condição seja verdadeira, um return será usado para sair da função.

Feito isso, será necessário utilizar o método **moveChildToIndex()** que terá como parâmetro a variável “action” e a variável “index” sendo incrementada em 1, além disso, será necessário usar um método **ensureChildVisible()** que terá como parâmetro a variável “action” e logo em seguida será utilizada a função **save()**.

```
registrarAcao("move up", function()
  local action = CaveBot.actionList:getFocusedChild()
  if not action then return end
  local index = CaveBot.actionList:getChildIndex(action)
  if index < 2 then return end
  CaveBot.actionList:moveChildToIndex(action, index - 1)
  CaveBot.actionList:ensureChildVisible(action)
  CaveBot.save()
end)
```

Mais uma vez será utilizada a função **registrarAcao()** que terá como argumento uma string “editar” e uma função. Será definida uma variável local chamada “action” que receberá o valor da variável “actionList” juntamente do método **getFocusedChild()**. Após isso será definido um **if not** para verificar se a variável “action” possui um valor nulo e será usado o operador lógico **or** juntamente com a cláusula **not** com o intuito de verificar se o evento, caso **onDoubleClick** retorna um valor nulo, caso a condição seja atendida, será usado um return para sair da função, caso contrário, o evento **onDoubleClick** será usado junto com a variável “action” e possuirá a própria variável “action” como parâmetro.

```
registrarAcao("edit", function()
  local action = CaveBot.actionList:getFocusedChild()
  if not action or not action.onDoubleClick then return end
  action.onDoubleClick(action)
end)
```

Mais uma vez, a variável “registrarAcao” será registrada e a mesma possuirá como parâmetro uma string “descer” e uma função. Esse trecho será muito semelhante com o trecho do botão “subir”.

```
-----  
registrarAcao("move down", function()  
    local action = CaveBot.actionList:getFocusedChild()  
    if not action then return end  
    local index = CaveBot.actionList:getChildIndex(action)  
    if index >= CaveBot.actionList:getChildCount() then return end  
    CaveBot.actionList:moveChildToIndex(action, index + 1)  
    CaveBot.actionList:ensureChildVisible(action)  
    CaveBot.save()  
end)
```

Será novamente utilizada a função **registrarAcao()** que terá como parâmetro uma string “remove” e também terá uma função como argumento. Será definida uma variável local chamada “action” que receberá o valor da variável “actionList” juntamente do método **getFocusedChild()**. Em sequência, será criado um **if not** para verificar se a variável “action” tem um valor nulo, com essa condição sendo atendida um return será utilizado para sair da função, do contrário a variável será utilizada juntamente ao método **destroy()** e será usada a função **save()**.

```
registrarAcao("remove", function()  
    local action = CaveBot.actionList:getFocusedChild()  
    if not action then return end  
    action:destroy()  
    CaveBot.save()  
end)
```

Será utilizada uma função **registrarAcao()** onde conterà uma string e uma tabela com os campos: “value”, “title”, “description”, “multiline”, em alguns casos existirá também o campo “validation”. No primeiro caso, a string será uma label, o campo “value” receberá o valor “labelName”, o campo “title” receberá o valor “label”, o “description” o valor “addLabel” e o campo “multiLine” receberá o valor **false**.

```
registrarAcao("label", {  
    value="labelName",  
    title="Label",  
    description="Add label",  
    multiline=false  
})
```


No segundo caso, a string será “delay”, o campo “value” receberá o valor 500, o campo “title” receberá o valor “delay”, o “description” receberá o valor “delay para proxima acao”, o campo “multiLine” receberá o valor **false** e o “validation” receberá o valor “^\s*[0-9]{1,10}\s*\$”.

```
registrarAcao("delay", {
  value="500",
  title="Delay",
  description="Delay next action (in milliseconds)",
  multiline=false,
  validation="^\s*[0-9]{1,10}\s*$"
})
```

O terceiro caso conterà os seguintes valores: a string será “gotolabel”, o campo “value” receberá o valor “labelName”, o campo “title” e o “description” receberão o valor “go to label” e o campo “multiLine” receberá o valor **false**.

```
registrarAcao("gotolabel", "go to label", {
  value="labelName",
  title="Go to label",
  description="Go to label",
  multiline=false
})
```

No quarto caso, a string será “goTo”, o campo “value” terá como valor uma função que irá retornar as coordenadas x,y e z de um ponto, o campo “title” receberá o valor “ir ate”, o campo “description” também receberá o valor “ir ate (x,y,z)”, o campo “multiLine” receberá o valor **false** e o “validation” receberá o valor “^\s*([0-9]+\s*,\s*([0-9]+\s*,\s*([0-9]+)\s*)\$”.

```
registrarAcao("goto", "go to", {
  value=function() return posx() .. "," .. posy() .. "," .. posz() end,
  title="Go to position",
  description="Go to position (x,y,z)",
  multiline=false,
  validation="^\s*([0-9]+\s*,\s*([0-9]+\s*,\s*([0-9]+)\s*)$"
})
```

No quinto caso, a string será “use”, o campo “value” terá como valor uma função que irá retornar as coordenadas x,y e z de um ponto, o campo “title” receberá o valor “use”, o campo “description” também receberá o valor “use o item na posicao (x,y,z)”, por último, o campo “multiLine” receberá o valor **false**.

```
registrarAcao("use", {
  value=function() return posx() .. "," .. posy() .. "," .. posz() end,
  title="Use",
  description="Use item from position (x,y,z) or from inventory (itemId)",
  multiline=false
})
```

No sexto caso, a string será “usarcom”, o campo “value” terá como valor uma função que irá retornar as coordenadas x,y e z de um ponto, o campo “title” terá o valor “usar com”, o campo “description” receberá o valor “use o item na posicao (x,y,z)”, o campo “multiLine” terá o valor **false** e o campo “validation” terá o valor “^\\s*([0-9]+)\\s*,\\s*([0-9]+)\\s*,\\s*([0-9]+)\\s*,\\s*([0-9]+)\$”.

```
registrarAcao("usewith", "use with", {
  value=function() return "itemId," .. posX() .. "," .. posY() .. "," .. posz() end,
  title="Use with",
  description="Use item at position (itemid,x,y,z)",
  multiline=false,
  validation="^\\s*([0-9]+)\\s*,\\s*([0-9]+)\\s*,\\s*([0-9]+)\\s*,\\s*([0-9]+)$"
})
```

No sétimo caso, a string será “say”, o campo “value” receberá o valor “text”, o campo “title” receberá o valor “say”, o “description” receberá o valor “digite o texto para ser dito” e o campo “multiLine” receberá novamente o valor **false**.

```
registrarAcao("say", {
  value="text",
  title="Say",
  description="Enter text to say",
  multiline=false
})
```

No último caso, a string será “function”, o campo “value” receberá o valor da tabela “exampleFunctions” nos índices 1 e 2, o campo “title” receberá o valor “editar funcao do bot”, o campo “multiLine” receberá o valor **true**, esse caso conterà dois campos adicionais, um denominado “example” que receberá a tabela “exampleFunctions” como valor e o campo “width” que receberá o valor 650.

```
registrarAcao("function", {
  title="Edit bot function",
  multiline=true,
  value=CaveBot.Editor.ExampleFunctions[1][2],
  examples=CaveBot.Editor.ExampleFunctions,
  width=650
})
```

Será definido um evento **onClick** para o botão “autoRecording” onde será definido um **if** que verifica se o botão “autoRecording” está ativo. Caso o botão esteja ativo será utilizado uma função **disable()** para desativá-lo, caso contrário será utilizado uma função **enable()** para ativá-lo

```
ui.autoRecording.onClick = function()
  if ui.autoRecording.isOn() then
    CaveBot.Recorder.disable()
  else
    CaveBot.Recorder.enable()
  end
end
```

Em seguida será utilizado a função **onPlayerPositionChange()** que terá como argumento uma função que terá como argumento a variável “posicao”. Em sequência será utilizado o método **setText()** que terá as propriedades **x**, **y** e **z** da variável “posicao” concatenadas por ponto duplo. O método será utilizado na propriedade **pos** da variável “ui”.

Em seguida será feito o mesmo, porém no lugar da variável “posicao” será passado três métodos que retornarão os valores para **x**, **y** e **z**.

```
onPlayerPositionChange(function(posicao)
  ui.pos:setText("Position: " .. posicao.x .. ", " .. posicao.y .. ", " .. posicao.z)
end)
ui.pos:setText("Position: " .. posX() .. ", " .. posY() .. ", " .. posz())
end
```

Depois será definida uma função **show()** e então usaremos o método **show()** na variável “ui”. O mesmo será feito para o método **hide()**.

```
CaveBot.Editor.show = function()
  CaveBot.Editor.ui:show()
end

CaveBot.Editor.hide = function()
  CaveBot.Editor.ui:hide()
end
```

Logo após será definido uma função **edit()** que terá uma variável “action”, “value” e “callback”, também será definido uma variável local “parametros” que terá como valor a tabela “actions” no índice “action”. Após isso será definido um **if not** para verificar se a variável “parametros” tem um valor nulo, caso essa condição seja atendida é usando um **return** para sair da função. Será criado outro **if not** para verificar se a variável “value” tem um valor nulo, caso a condição seja atendida será criado um **if** que verificará se a propriedade **value** da variável “parametros” é do tipo função, sendo do tipo função ela será atribuída a variável “value”. Será definido então um **else if** que verificará se a propriedade **values** da variável “parametros” é do tipo string, caso seja, ela será atribuída a variável “value”.

```
CaveBot.Editor.edit = function(action, value, callback)
  local parametros = CaveBot.Editor.Actions[action]
  if not parametros then return end
  if not value then
    if type(parametros.value) == 'function' then
      value = parametros.value()
    elseif type(parametros.value) == 'string' then
      value = parametros.value
    end
  end
end
```

Em seguida será utilizado o método **editorWindow()** que terá como parâmetro a variável “value”, uma variável “parametros” e uma função. A função terá como parâmetro a variável “novoTexto”, será utilizado um callBack que passar como parâmetro as variáveis actions e “novoTexto”.

```
] UI.EditorWindow(value, params, function(newText)
    callback(action, newText)
end)
end
```

Assim como feito anteriormente deverá ser colocado o diretório deste arquivo junto ao diretório dos demais arquivos.

Exemplo de função

Este arquivo irá conter um exemplo de função, que é utilizado como parâmetro em uma das funções do arquivo do editor de ações. E também deverá ser criado um novo arquivo de extensão .lua que deverá remeter ao editor de ações do bot.

Será definido uma tabela “exampleFunctions” e em sequência será definido uma função local cujo o nome será **addExampleFunction()** que terá como argumento uma variável “title” e uma variável “text”. A função irá retornar o comando **table.insert** que terá como argumento a tabela “exampleFunctions” e retornará uma tabela com os campos preenchidos pelas variáveis “title” e “text”.

```
CaveBot.Editor.ExampleFunctions = {}

local function addExampleFunction(title, text)
    return table.insert(CaveBot.Editor.ExampleFunctions, {title, text:trim()})
end
```

Em seguida será utilizado a função **addExampleFunction()** que terá como argumento a string “clique para pesquisar exemplo de funcoes” e em seguida um comentário escrito “return true”.

```
]addExampleFunction("Click to browse example functions", [[
return true
]])
```

Novamente será utilizado a função **addExampleFunction()**. Essa função deve ser preenchida com os seguintes exemplos de funções abaixo:

```
]addExampleFunction("Disable TargetBot", [[
TargetBot.setOff()
return true
]])
```

```
addExampleFunction("Enable TargetBot", [[
  TargetBot.setOn()
  return true
]])
```

```
addExampleFunction("Enable TargetBot luring", [[
  TargetBot.enableLuring()
  return true
]])
```

```
addExampleFunction("Disable TargetBot luring", [[
  TargetBot.disableLuring()
  return true
]])
```

```
addExampleFunction("Logout", [[
  g_game.safeLogout()
  delay(1000)
  return "retry"
]])
```

```
addExampleFunction("buy 200 mana potion from npc Eryn", [[
  local npc = getCreatureByName("Eryn")
  if not npc then
    return false
  end
  if retries > 10 then
    return false
  end
  local posicao = player:getPosition()
  local npcPosicao = npc:getPosition()
  if math.max(math.abs(posicao.x - npcPosicao.x), math.abs(posicao.y - npcPosicao.y)) > 3 then
    autoWalk(npcPosicao, {precisao=3})
    delay(300)
    return "retry"
  end
  if not NPC.isTrading() then
    NPC.say("hi")
    NPC.say("trade")
    delay(200)
    return "retry"
  end
  NPC.buy(268, 100)
  schedule(1000, function()

    NPC.buy(268, 100)
    NPC.closeTrade()
    NPC.say("bye")
  end)
  delay(1200)
  return true
]])
```

```

addExampleFunction("Say hello 5 times with some delay", [[
  if retries > 5 then
    return true
  end
  say("hello")
  delay(100 + retries * 100)
  return "retry"
]])

```

Assim como feito anteriormente deverá ser colocado o diretório deste arquivo junto ao diretório dos demais arquivos.

Criando o Gravador

O gravador servirá para auxiliar o jogador a criar scripts do bot, gerando um script a partir dos movimentos que a pessoa fizer, de maneira automática.

Será definido uma tabela “recorder” e duas variáveis locais com o valor **nil**, que serão as variáveis “ativo” e “ultimaPosicao”.

```

CaveBot.Recorder = {}

local Ativo = nil
local ultimaPosicao = nil

```

Será definida também uma função local **setup()** onde será definido dentro dela uma função **addPosition()** que terá como argumento a variável “posicao”. Será utilizado uma função **addAcao()** que terá como parâmetros a string “goto” e também terá as propriedades **x**, **y** e **z** da variável “posicao” e o valor booleano **true**. Depois disso, a variável “ultimaPosicao” receberá o valor da variável “posicao”.

```

local function setup()
  local function addPosition(posicao)
    CaveBot.addAcao("goto", posicao.x .. "," .. posicao.y .. "," .. posicao.z, true)
    ultimaPosicao = posicao
  end
end

```

Em sequência, será utilizado um método **onPlayerPositionChange()** que terá como argumento uma função que terá como argumento a variável “novaPosicao” e a variável “antigaPosicao”. Em seguida, será definido um **if** que verificará se o **CaveBot** está ativo e será utilizado o operador lógico **or**, junto da cláusula **not** para verificar se a variável ativo possui um valor nulo. Caso uma das condições sejam atendidas será utilizado um return para sair da função. Depois disso será criado um **if not** para verificar se a variável “ultimaPosicao” tem um valor nulo. Caso a condição seja atendida será utilizado a função **addPositio()** que terá como parâmetro a variável “antigaPosicao”.

Será criado um **else if** que verificará se o eixo z da variável “novaPosicao” é diferente do eixo z da variável “antigaPosicao”. Em sequência será utilizado o

operador lógico **or**, utilizando também o método **math.abs()** onde será passado a diferença entre os eixos x das variáveis “antigaPosicao” e “novaPosicao” e verificará se o resultado é maior que 1. O mesmo será feito para o eixo y. Será utilizado a função **addPosition()** que passará a variável “antigaPosicao”, depois será utilizado o **else if** com o método **math.max()** que verificará se o retorno do método será maior que 5, o método utilizará as duas operações feitas para o eixo x e y no **else if** anterior. Caso a condição seja atendida será utilizado o método **addPosition()** que terá como parâmetro a variável “novaPosicao”

```
onPlayerPositionChange(function(novaPosicao, antigaPosicao)
  if CaveBot.isOn() or not Ativo then return end
  if not ultimaPosicao then
    addPosition(antigaPosicao)
  elseif novaPosicao.z ~= antigaPosicao.z or math.abs(antigaPosicao.x - novaPosicao.x) > 1
  or math.abs(antigaPosicao.y - novaPosicao.y) > 1 then
    addPosition(antigaPosicao)
  elseif math.max(math.abs(ultimaPosicao.x - novaPosicao.x), math.abs(ultimaPosicao.y - novaPosicao.y)) > 5 then
    addPosition(novaPosicao)
  end
end)
```

Será utilizado o método **onUse()** que terá como argumento uma função que terá como argumento uma variável “posicaoItemId”, “stackPosicao” e “subType”. Em seguida, será definido um **if** que verificará se o **CaveBot** está ativo e será utilizado o operador lógico **or**, junto da cláusula **not** para verificar se a variável ativo possui um valor nulo. Caso uma das condições sejam atendidas será utilizado um **return** para sair da função.

Em sequência será criado um **if** que verificará se a propriedade **x** da variável “posicao” é diferente do código 0xFFFF. Caso a condição seja atendida será a variável “ultimaPosicao” receberá o valor da variável “posicao” e será utilizado a função **addAcao()**, onde será passado a string “use”, os eixos x, y e z da variável “posicao” e o valor booleano **true**.

```
onUse(function(posicao, itemId, stackPosicao, subType)
  if CaveBot.isOn() or not Ativo then return end
  if posicao.x ~= 0xFFFF then
    ultimaPosicao = posicao
    CaveBot.addAcao("use", posicao.x .. "," .. posicao.y .. "," .. posicao.z, true)
  end
end)
```

Em seguida, será utilizado a função **onUseWith()** que terá como parâmetro uma função que terá como parâmetro uma variável “posicao”, uma variável “itemId”, uma variável “target” e uma variável “subType”. Em seguida, será definido um **if** que verificará se o **CaveBot** está ativo e será utilizado o operador lógico **or**, junto da cláusula **not** para verificar se a variável “ativo” possui um valor nulo. Caso uma das condições sejam atendidas será utilizado um **return** para sair da função.

Depois será criado um **if not** onde será utilizado o método **isItem()** na variável “target”. Caso a condição seja verdadeira será utilizado um **return** para sair da função. Será criado uma variável “targetPosicao” que receberá o valor da variável “target” e o método **getPosition()**.

Será verificado através de um **if** se o eixo x da variável “targetPosicao” é igual ao código 0xFFFF. Caso a condição seja verdadeira será utilizado um **return** para sair da função. Após isso a variável “ultimaPosicao” receberá o valor da variável “posicao”. Será utilizado novamente a função **addAcao()** que passará a string “usewith”, a variável “itemId”, os eixos x, y e z da variável “posicao” e o valor booleano **true**.

```

3 onUseWith(function(posicao, itemId, target, subType)
  if CaveBot.isOn() or not Ativo then return end
  if not target:isItem() then return end
  local targetPosicao = target:getPosition()
  if targetPosicao.x == 0xFFFF then return end
  ultimaPosicao = posicao
  CaveBot.addAcao("usewith", itemId .. "," .. targetPosicao.x .. "," .. targetPosicao.y .. "," .. targetPosicao.z, true)
end)
-end

```

Em seguida será definida uma função **isOn()** que retornará a variável “ativo”.

```

CaveBot.Recorder.isOn = function()
  return Ativo
end

```

Será definida uma função **enable()** onde inicialmente será utilizado o método **setOff()** no objeto **CaveBot**, será verificado se a variável “ativo” tem o valor igual a **nil**. Caso a condição seja atendida será utilizada a função **setup()**. Em seguida será utilizado o método **setOn()** com o valor **true** no botão “autoRecording”, a variável “ativo” receberá o valor **true** e a variável “ultimaPosicao” receberá o valor **nil**.

```

CaveBot.Recorder.enable = function()
  CaveBot.setOff()
  if Ativo == nil then
    setup()
  end
  CaveBot.Editor.ui.autoRecording:setOn(true)
  Ativo = true
  ultimaPosicao = nil
end

```

Será criada uma função **disable()** onde será verificado se a variável “ativo” tem o valor **true**, caso a condição seja atendida, a variável receberá o valor **false**. Novamente será utilizado o método **setOn()** com o valor **false** no botão “autoRecording” e será utilizado a função **save()**.

```

CaveBot.Recorder.disable = function()
  if Ativo == true then
    Ativo = false
  end
  CaveBot.Editor.ui.autoRecording:setOn(false)
  CaveBot.save()
end

```


Assim como feito anteriormente deverá ser colocado o diretório deste arquivo junto ao diretório dos demais arquivos.

Criando as funções para andar

Este arquivo será responsável por executar as funções de movimento do personagem. Será algo similar ao feito no Case 02.

Será definido três tabelas locais, a primeira receberá o nome de “diretorioEsperado”, a segunda receberá o nome de “estaAndando”, a terceira receberá o nome de “caminhoAndar” e uma variável local “caminhoIterado”.

```
local diretorioEsperado = {}
local estaAndando = {}
local caminhoAndar = {}
local caminhoIterado = 0
```

Será criado um função **resetaAndar()**, onde será esvaziado as tabelas “diretorioEsperado”, “caminhoAndar” e atribuirá o valor **false** para a tabela “estaAndando”.

```
CaveBot.resetaAndar = function()
  diretorioEsperado = {}
  caminhoAndar = {}
  estaAndando = false
end
```

Em seguida será definido uma função **caminhar()** onde será criado um **if** e utilizaremos o método **get()** na variável “config”, será utilizado como argumento para o método uma string contendo o valor “mapClick”. Esse **if** verificará a existência da string “mapClick”, caso a condição seja atendida será retornado o valor booleano **false**.

Através de um **if** será verificado se a tabela “diretorioEsperado” possui o valor igual a zero, caso a condição seja verdadeira, será retornado o valor **false**. Novamente será criado **if** para verificar se o valor da tabela “diretorioEsperado” é maior ou igual a 3, caso essa condição seja atendida será utilizado a função **resetaAndar()**.

```
CaveBot.caminhar = function()
  if CaveBot.Config.get("mapClick") then
    return false
  end
  if #diretorioEsperado == 0 then
    return false
  end
  if #diretorioEsperado >= 3 then
    CaveBot.resetaAndar()
  end
end
```

Em sequência será criada uma variável local “diretorio” que receberá o valor da tabela “caminhoAndar” no índice da variável “caminhoIterado”. Após isso será criado um **if** para verificar se a variável “diretorio” possui um valor diferente de nulo, caso a condição seja verdadeira será utilizado um método **g_game.walk()**, será passado como argumento a variável “diretorio” e o valor booleano **false**. Em sequência será utilizado o comando **table.insert** e será passado como argumento a tabela “diretorioEsperado” e a variável “diretorio”.

A variável “caminhoIterado” será incrementada em +1, será utilizado então a função **delay()** que terá como argumento o método **get()** na variável “config” que terá como argumento a string “walkDelay” o resultado desse método será somado ao resultado do método **getStepDuration()** terá como argumento o valor booleano **false** e a variável “diretorio”. Em seguida será criado um **return** para retornar o valor **true** e um **return** para retornar o valor **false**.

```
----
local diretorio = caminhoAndar[caminhoIterado]
if diretorio then
    g_game.walk(diretorio, false)
    table.insert(diretorioEsperado, diretorio)
    caminhoIterado = caminhoIterado + 1
    CaveBot.delay(CaveBot.Config.get("walkDelay") + player:getStepDuration(false, diretorio))
    return true
end
return false
end
```

Em seguida será utilizado um método **onPlayerPositionChange()** que terá como argumento um função, a função terá como argumento as variáveis “novaPosicao” e “antigaPosicao”. Logo após será criado um **if not** que verificará se a variável “antigaPosicao” ou a variável “novaPosicao” possuem um valor diferente de nulo.

Em seguida será criada uma tabela cujo nome será “diretorios” e será preenchida com os seguintes campos: {NorthWest, North, NorthEast}, {West, 8, East}, {SouthWest, South, SouthEast}. Será definido uma variável local “diretorio” que receberá o valor da tabela “diretorios” tendo como índices o eixo y da variável “novaPosicao”, realizando uma subtração com o eixo y da variável “antigaPosicao”.

Será criado um **if** que verificará se a variável “diretorio” possui um valor diferente de nulo, caso a condição seja atendida a variável “diretorio” irá receber o valor da variável “diretorio” no índice o eixo x da variável “novaPosicao” subtraindo o eixo x da variável “novaPosicao”. Será criado também um **if not** onde caso a variável “diretorio” tenha o valor nulo será atribuído a ela o valor 8.

```
onPlayerPositionChange(function(novaPosicao, antigaPosicao)
    if not antigaPosicao or not novaPosicao then return end

    local diretorios = {{NorthWest, North, NorthEast}, {West, 8, East}, {SouthWest, South, SouthEast}}
    local diretorio = diretorios[novaPosicao.y - antigaPosicao.y + 2]
    if diretorio then
        diretorio = diretorio[novaPosicao.x - antigaPosicao.x + 2]
    end
    if not diretorio then
        diretorio = 8
    end
end
```

Será criado um **if not** onde será verificado se a variável “estaAndando” ou se a tabela “diretorioEsperado” no índice 1 possuem valores nulos. Caso a condição seja verdadeira a tabela “caminhoAndar” será esvaziada e será utilizado a função **delay()** que terá como parâmetro a soma do método **get()** com o parâmetro da string “ping” ao método **getStepDuration()** que terá como argumento o valor booleano **false**, a variável “diretorio” e somará o valor 150.

```
if not estaAndando or not diretorioEsperado[1] then
    caminhoAndar = {}
    CaveBot.delay(CaveBot.Config.get("ping") + player:getStepDuration(false, diretorio) + 150)
    return
end
```

Será definido um **if** que irá verificar se o primeiro índice da tabela “diretorioEsperado” é diferente do valor da variável “diretorio”, caso a condição seja verdadeira, será criado um **if** que irá verificar através do método **get()** se a configuração “mapClick” está ativada. Em seguida será utilizado a função **delay()** que terá como argumento o método **get()** na variável “config” que terá como argumento a string “walkDelay” o resultado desse método será somado ao resultado do método **getStepDuration()** terá como argumento o valor booleano **false** e a variável “diretorio”.

Será definido um **else**, que será utilizado então a função **delay()** que terá como argumento o método **get()** na variável “config” que terá como argumento a string “mapClickDelay” o resultado desse método será somado ao resultado do método **getStepDuration()** terá como argumento o valor booleano **false** e a variável “diretorio”. Em seguida, será criado um **return** para sair da função.

```
if diretorioEsperado[1] ~= diretorio then
    if CaveBot.Config.get("mapClick") then
        CaveBot.delay(CaveBot.Config.get("walkDelay") + player:getStepDuration(false, diretorio))
    else
        CaveBot.delay(CaveBot.Config.get("mapClickDelay") + player:getStepDuration(false, diretorio))
    end
    return
end
```

Em seguida será utilizado o comando **table.remove** que tirará o primeiro índice da tabela “diretorioEsperado” e será criado um **if** que através do método **get()** verificará se a configuração de “mapClick” está ativada e se a tabela “diretorioEsperado” possui o valor maior que zero, caso a condição seja verdadeira, será utilizado então a função **delay()** que terá como argumento o método **get()** na variável “config” que terá como argumento a string “mapClickDelay” o resultado desse método será somado ao resultado do método **getStepDuration()** terá como argumento o valor booleano **false** e a variável “diretorio”.

```
table.remove(diretorioEsperado, 1)
if CaveBot.Config.get("mapClick") and #diretorioEsperado > 0 then
    CaveBot.delay(CaveBot.Config.get("mapClickDelay") + player:getStepDuration(false, diretorio))
end
end)
```

Será definido uma função **andarAte()** que terão como parâmetros uma variável “destino”, uma “distanciaMaxima” e uma variável “parametro”. Será criado em seguida uma variável local “caminho” que terá atribuído o valor do método **getPath()** que terá como parâmetro o método **getPosition()** no objeto **player** com as variáveis “destino”, “distanciaMaxima” e “parametro”. Em seguida será definido um **if not** que terá como condição a variável “caminho” ter um valor igual a nulo ou a tabela “caminho” no índice um ter um valor nulo, caso seja verdadeiro será retornado o valor booleano **false**.

```
CaveBot.andarAte = function(destino, maxDist, parametros)
  local caminho = getPath(player:getPosition(), destino, maxDist, parametros)
  if not caminho or not caminho[1] then
    return false
  end
end
```

Em seguida será criado uma variável local “diretorio”, que receberá o valor do primeiro índice da tabela caminho, em sequência será definido um **if** que verificará através do método **get()** se a configuração do “mapClick” está ativada, caso a condição seja atendida, será criado uma variável local “ret”. A variável “ret” receberá o valor do método **autoWalk()** tendo como parâmetro a variável “caminho”.

Logo após será verificado se a variável “ret” possui valor diferente de nulo, caso a condição seja verdadeira a variável “estaAndando” receberá o valor booleano **true** e a tabela “diretorioEsperado” receberá o valor da tabela “caminho”. Em seguida será utilizado a função **delay()** que terá como argumento o método **get()** na variável “config” que terá como argumento a string “mapClickDelay”, o resultado desse método será somado ao resultado do método **math.max()** que terá como parâmetro o método **get()** verificando se a configuração ping está ativa, em sequência a soma do método **getStepDuration()** que terá como argumento o valor booleano **false** e a variável “diretorio”. O método **getStepDuration()** será utilizado novamente e multiplicado por dois e retornará a variável “ret”.

```
if CaveBot.Config.get("mapClick") then
  local ret = autoWalk(caminho)
  if ret then
    estaAndando = true
    diretorioEsperado = caminho
    CaveBot.delay(CaveBot.Config.get("mapClickDelay") + math.max(CaveBot.Config.get("ping") +
      player:getStepDuration(false, diretorio), player:getStepDuration(false, diretorio) * 2))
  end
  return ret
end
```

Por fim, será utilizado o método **g_game.walk()** que terá como parâmetros a variável “diretorio” e o valor booleano **false**. Após isso a variável “estaAndando” receberá o valor **true**, a variável “caminhoAndar” receberá o valor da variável “caminho”, a variável “caminholterado” receberá o valor dois, a tabela de “diretorioEsperado” receberá o valor da variável diretorio e será utilizado a função **delay()** que terá como argumento o método **get()** na variável “config” que terá como argumento a string “walkDelay” o resultado desse método será

somado ao resultado do método **getStepDuration()** terá como argumento o valor booleano **false** e a variável “diretorio”, no fim será retornado o valor **true**.

```
g_game.walk(diretorio, false)
estaAndando = true
caminhoAndar = caminho
caminhoIterado = 2
diretorioEsperado = { diretorio }
CaveBot.delay(CaveBot.Config.get("walkDelay") + player:getStepDuration(false, diretorio))
return true
end
```

Assim como feito anteriormente deverá ser colocado o diretório deste arquivo junto ao diretório dos demais arquivos.

Criando as funções de depósito de item

Agora será necessário criar mais um arquivo .lua e este arquivo será responsável por definir algumas funções que estão relacionadas com o depósito de itens.

Será definido uma tabela “depositer” que estará vazia, em sequência será criado uma variável local “ui” e para a tabela será criado um função **setup()**, que não terá parâmetros e linhas de código. Será criado uma função **onConfigChange()** que terá como parâmetro a variável “configName”, uma variável “ativo”, uma variável “configData” e será definido um **if** onde será verificado se a variável “configData” tem um valor igual a nulo, caso a condição verdadeira haverá um return para sair da função.

Logo após será criada uma função **onSave()** que retornará uma tabela vazia e será criado uma função **run()** que terá como parâmetro uma variável “retries”, uma variável “prev” e retornará o valor **true**.

```
CaveBot.Extensions.Depositer = {}

local ui

CaveBot.Extensions.Depositer.setup = function()
end

CaveBot.Extensions.Depositer.onConfigChange = function(configNome, Ativo, configData)
    if not configData then return end
end

CaveBot.Extensions.Depositer.onSave = function()
    return {}
end

CaveBot.Extensions.Depositer.run = function(retries, prev)
    return true
end
```

Assim como feito anteriormente deverá ser colocado o diretório deste arquivo junto ao diretório dos demais arquivos.

Criando as funções de “supply”

Será necessário criar um novo arquivo de extensão .lua. Este arquivo ficará responsável por algumas funções que tem a finalidade de fazer o abastecimento.

Será definido uma tabela “supply” que estará vazia, em sequência será criado uma variável local “ui” e para a tabela será criado um função **setup()**, que não terá parâmetros e linhas de código. Será criado uma função **onConfigChange()** que terá como parâmetro a variável “configName”, uma variável “ativo”, uma variável “configData” e será definido um **if** onde será verificado se a variável “configData” tem um valor igual a nulo, caso a condição verdadeira haverá um return para sair da função.

Logo após será criada uma função **onSave()** que retornará uma tabela vazia e será criado uma função **run()** que terá como parâmetro uma variável “retries”, uma variável “prev” e retornará o valor **true**.

```
CaveBot.Extensions.Supply = {}

local ui

CaveBot.Extensions.Supply.setup = function()
end

CaveBot.Extensions.Supply.onConfigChange = function(configNome, Ativo, configData)
    if not configData then return end
end

CaveBot.Extensions.Supply.onSave = function()
    return {}
end

CaveBot.Extensions.Supply.run = function(retries, prev)
    return true
end
```

Assim como feito anteriormente deverá ser colocado o diretório deste arquivo junto ao diretório dos demais arquivos.

Finalizando o CaveBot

Será necessário criar um arquivo de extensão .lua, este arquivo será responsável por finalizar o CaveBot.

Para começar será definido duas variáveis locais, onde a primeira se chamará “config” e a segunda se chamará “CaveBotMacro”, ambas irão receber o valor **nil**, feito isso, será definido uma variável para configuração do widget, chamada “configWidget” e a mesma irá receber o valor do método **ui.Config()**. Em sequência, será criado uma variável “ui.CreateWidget” que possuirá como parâmetro a string **CaveBotPanel**.

```
local cavebotMacro = nil
local config = nil

local configWidget = UI.Config()
local ui = UI.createWidget("CaveBotPanel")
```

Inicialmente, será definido para o campo **list** da variável “ui” e será atribuído a ele o valor do campo **list** do objeto **listPanel**, após isso, para a propriedade **actionList** do objeto **CaveBot**, será atribuído o valor do campo **list** da variável “ui”. Sequencialmente será definido um **if** onde será verificado se a propriedade **editor** do objeto **CaveBot** possui um valor diferente de nulo, caso essa condição seja atendida, será usada uma função **setup()** na propriedade **editor** do mesmo objeto. Esse processo será repetido na variável “config” do objeto **CaveBot**.

```
ui.list = ui.listPanel.list
CaveBot.actionList = ui.list

if CaveBot.Editor then
    CaveBot.Editor.setup()
end

if CaveBot.Config then
    CaveBot.Config.setup()
end
```

Será definido um **for** que irá iterar a tabela “extensions” através das variáveis locais, sendo elas “extension” e “callback”, a partir disso, será criado um **if** onde será verificado se a variável “callback” possui um valor diferente de nulo, caso a condição seja verdadeira, será utilizado o método **setup()** na variável “callback”.

```
]for extension, callbacks in pairs(CaveBot.Extensions) do
]   if callbacks.setup then
       callbacks.setup()
       end
end
```

Em seguida será definido duas variáveis locais, sendo a primeira delas “acaoTentiva” e a segunda “resultadoAcao”. A primeira receberá o valor de zero

e a segunda receberá o valor **true**. Sequencialmente, será atribuído um valor para a variável “CaveBotMacro”, onde será usado um macro, que possuirá vinte milissegundos de delay e terá uma função como parâmetro. Agora será definido um **if** e o mesmo irá verificar se o objeto **targetBot** está ativo, caso essa condição seja atendida, será usado a função **resetaAndar()** do objeto **CaveBot**.

```
local actionRetries = 0
local prevActionResult = true
cavebotMacro = macro(20, function()
  if TargetBot and TargetBot.isActive() and not TargetBot.isCaveBotActionAllowed() then
    CaveBot.resetaAndar()
    return
  end
end
```

Em sequência, será definido um **if**, que terá como objetivo verificar se o retorno da função **caminhar()** é diferente de nulo, caso essa condição seja atendida, haverá um return que não retornará nada, o mesmo servirá apenas para sair da função.

```
if CaveBot.caminhar() then
  return
end
```

Logo será criada uma variável local “actions”, que receberá o valor do método **getChildCount()**, sendo utilizado na propriedade **list** da variável “ui”. Será definido um **if** que irá verificar se a variável “actions” possui valor igual a zero, caso essa condição seja verdadeira, será usado um return para sair da função. Além disso, será criada uma variável local chamada de “acaoAtual”, a mesma receberá o valor do método **getFocusedChild()**, que será utilizado da propriedade **list** da variável “ui”, após isso, será definido um **if not** onde será verificado se a variável “acaoAtual” possui um valor nulo, caso essa condição seja atendida, será atribuído a variável o valor do método **getFirstChild()** sendo utilizado na variável “list” da propriedade “ui”.

```
local actions = ui.list:getChildCount()
if actions == 0 then return end
local acaoAtual = ui.list:getFocusedChild()
if not acaoAtual then
  acaoAtual = ui.list:getFirstChild()
end
```


Serão criadas três variáveis locais, sendo a primeira “action”, que receberá o valor da tabela “actions” e possuirá como índice a variável “acaoAtual”, já a segunda variável se chamará “value”, a mesma receberá o valor da propriedade “value” da variável “acaoAtual” e a terceira variável será “retry”, que receberá o valor false.

```
local action = CaveBot.Actions[acaoAtual.action]
local value = acaoAtual.value
local retry = false
```

Um **if** será definido, com o intuito de verificar se a variável “actions” possui um valor diferente de nulo, caso a condição seja verdadeira, será criada duas variáveis locais chamadas “status” e “result”, para essas variáveis será atribuído o valor de um método **pCall()** que terá como parâmetro uma função, dentro dessa função será utilizado o método **resetAndar()** e será retornado um callback que possuirá três argumentos, sendo eles uma variável “value”, uma variável “acaoTentativa” e o último sendo “resultadoAcao”.

```
if action then
  local status, result = pcall(function()
    CaveBot.resetaAndar()
    return action.callback(value, actionRetries, prevActionResult)
  end)
```

Será definido um **if**, onde terá como condição a variável “status” possuindo um valor diferente de nulo, em seguida será definido um **if**, o mesmo servirá para verificar se a variável “result” possui o valor da variável “retry”, caso essa condição seja atendida, a variável “acaoTentativa” será incrementada em um e a variável “retry” possuirá o valor booleano **true**.

```
if status then
  if result == "retry" then
    actionRetries = actionRetries + 1
    retry = true
```

Nesse momento, será definido um **else if**, que verificará se a variável “result” será do tipo booleano, com essa condição sendo atendida, a variável “acaoTentativa” receberá o valor 0 e a variável “resultadoAcao” irá receber o valor da variável “result”.

```
-----
elseif type(result) == 'boolean' then
  actionRetries = 0
  prevActionResult = result
-----
```

Serão definidos alguns **elses** que irão retornar alguns erros.

```
----
    error("Acao invalida (" .. acaoAtual.action .. "), deveria \"retry\", falso ou verdadeiro, mas é:: " ..
        tostring(result))
end
else
    error("Erro ao executar ação (" .. acaoAtual.action .. "):\n" .. result)
end
else
    error("Acao Invalida: " .. acaoAtual.action)
end
```

Será criado um **if**, que irá verificar se a variável “retry” possui um valor diferente de nulo, no caso dessa condição ser verdadeira, um return será usado para sair da função.

```
if retry then
    return
end
```

Feito isso, será criado outro **if** que terá como condição a variável “acaoAtual” tendo um valor diferente da propriedade **list** da variável “ui” juntamente do método **getFocusedChild()**, caso essa condição seja atendida a variável “acaoAtual” receberá o valor do método **getFocusedChild()** sendo usado na propriedade **list** da variável “ui”. Além disso, a variável “acaoTentativa” receberá o valor 0 e a variável “resultadoAcao” receberá o valor true.

```
if acaoAtual ~= ui.list:getFocusedChild() then
    acaoAtual = ui.list:getFocusedChild() or ui.list:getFirstChild()
    actionRetries = 0
    prevActionResult = true
end
```

Uma variável local chamada “proximaAcao” será criada, ela receberá o valor da propriedade **list** da variável “ui” juntamente do método **getChildIndex()**, que terá como parâmetro a variável “acaoAtual” sendo incrementada em mais um. Após isso, um **if** será definido, esse terá como condição a variável “proximaAcao” possuindo um valor maior que o da variável “action”, caso isso seja atendido, a variável “proximaAcao” receberá o valor numérico 1. Em sequência, será usado o método **focusChild()** na propriedade **list** da variável “ui”, ele terá como argumento o método **getChildByIndex()** que terá como argumento a variável “proximaAcao”.

```
local proximaAcao = ui.list:getChildIndex(acaoAtual) + 1
if proximaAcao > actions then
    proximaAcao = 1
end
ui.list:focusChild(ui.list:getChildByIndex(proximaAcao))
end)
```

Será definida uma variável local “ultimaConfig”, essa receberá o valor de uma string vazia. Após isso, a variável “config” receberá como valor a função **setup()**, que terá como parâmetro uma string “CaveBot_Config”, a variável “configWidget”, uma string “CFG” e uma função, que receberá como parâmetro uma variável “nome”, uma variável “ativado” e por último uma variável “data”.

Um **if** será criado, ele servirá para verificar se o gravador do bot está ativo, caso isso seja atendido, será utilizada a função **disable()** e **setOff()** para desligar o gravador, além disso, um **return** será usado para sair da função.

```
local ultimaConfig = ""
config = Config.setup("cavebot_configs", configWidget, "cfg", function(nome, ativado, data)
  if ativado and CaveBot.Recorder.isOn() then
    CaveBot.Recorder.disable()
    CaveBot.setOff()
  return
end
```

Uma variável local será criada, ela receberá o nome de “indexAcaoAtual” e terá como valor o método **getChildIndex()**, que terá como parâmetro a propriedade **list** da variável “ui” juntamente ao método **getFocusedChild()**. Sequencialmente, será criado o método **destroyChildren()**, um **if not** será utilizado para verificar se a variável “data” possui um valor igual a nulo, caso essa condição seja verdadeira, será retornado o método **setOff()**.

```
local indexAcaoAtual = ui.list:getChildIndex(ui.list:getFocusedChild())
ui.list:destroyChildren()
if not data then return cavebotMacro.setOff() end
```

Será então definida uma nova variável local “cavebotConfig”, recebendo o valor “nil”. Em sequência, será definido um **for** que conterá um chave “K” e uma chave “v”, onde haverá uma verificação se a chave “v” é do tipo “table” e se o id do primeiro índice será igual a 2. Também deve-se verificar se a chave “v” em seu primeiro índice consta com o valor “config”.

Deve-se agora ser criadas duas outras variáveis: “status” e “result”, onde ambas receberão o valor do método **pcall()**, que receberá como parâmetro uma função, esta que deve retornar o código “json.decode(v[2])”.

```
local cavebotConfig = nil
for k,v in ipairs(data) do
  if type(v) == "table" and #v == 2 then
    if v[1] == "config" then
      local status, result = pcall(function()
        return json.decode(v[2])
      end)
    ..
  ..
end
```

Em seguida, deverá ser definido um **if not** para verificar se a variável “status” possui um valor nulo, utilizando o método **error()**, onde como parâmetro haverá uma string contendo o texto “erro ao analisar as extensões do bot”, que será

concatenado por pontos duplos à variável “result”, e após isso, defini-se um **else** que irá atribuir o valor da variável “result” à variável “cavebotConfig”.

```
if not status then
    error("Error while parsing CaveBot extensions from config:\n" .. result)
else
    cavebotConfig = result
end
```

Como próximo passo, deverá ser definido um **else if** para verificar se a chave “v” em seu primeiro índice possui o valor “extensions”, onde caso a condição seja atendida deve-se repetir o processo feito acima, criando outras duas outras variáveis: “status” e “result”, onde ambas receberão o valor do método **pcall()**, que receberá como parâmetro uma função, esta que deve retornar o código “json.decode(v[2])”.

Em seguida, continuaremos repetindo um dos processos acima, definindo um **if not** para verificar se a variável “status” possui um valor nulo, utilizando o método **error()**, onde como parâmetro haverá uma string contendo o texto “erro ao analisar as extensões do bot”, que será concatenado por pontos duplos à variável “result”, e após isso, defini-se um **else** que irá atribuir o valor da variável “result” à variável “cavebotConfig”.

```
elseif v[1] == "extensions" then
    local status, result = pcall(function()
        return json.decode(v[2])
    end)
    if not status then
        error("Error while parsing CaveBot extensions from config:\n" .. result)
    else
```

Deverá agora ser criado um **for** que receberá como índice a variável “extension”, e que como chave receberá a variável “callbacks”, onde esse **for** vai percorrer a tabela “extensions”. Dentro disso, deverá ser criado um **If** para verificar se o evento **onConfigChange()** acontecerá, e caso essa condição seja atendida, deverão ser passados 3 parâmetros para ele: a variável “nome”, “ativado” e “result”, tendo como índice a chave “extension”, onde também deverá ser definido um **else** utilizando a função **addAção()** que receberá como parâmetro o primeiro e segundo índice da chave “v”.

```
for extension, callbacks in pairs(CaveBot.Extensions) do
    if callbacks.onConfigChange then
        callbacks.onConfigChange(nome, ativado, result[extension])
    end
end
else
    CaveBot.addAcao(v[1], v[2])
end
end
```

Em sequência, deverá ser definido 3 parâmetros para o evento **onConfigChange()** da variável “config”: As variáveis “nome”, “ativado” e “cavebotConfig”.

Agora, deve-se definir para a variável “acaoTentativa” o valor numérico 0, em seguida utilizaremos a função **resetaAndar()**, e para a variável “resultadoAcao” será atribuído o valor true, além de utilizar o método **setOn()** do “cavebotMacro”, que receberá como parâmetro a variável “ativado”, e posteriormente, deve ser utilizada a variável “delay” do “cavebotMacro”, atribuindo a ela o valor “nil”.

```
CaveBot.Config.onConfigChange(nome, ativado, cavebotConfig)

actionRetries = 0
CaveBot.resetaAndar()
prevActionResult = true
cavebotMacro.setOn(ativado)
cavebotMacro.delay = nil
```

Deve-se agora definir um **if** para verificar se a variável “ultimaConfig” possui o mesmo valor da variável “nome”, onde caso essa condição seja verdadeira, será utilizado o método **focusChild()** na propriedade **list** na variável “ui”, e que como parâmetro para esse método, será utilizado o método **getChildByIndex()** e esse segundo método também receberá um parâmetro, a variável “indexAcaoAtual”.

Caso a condição do **if** não seja atendida, a variável “ultimaConfig” receberá o valor da variável “nome”.

```
actionRetries = 0
CaveBot.resetaAndar()
prevActionResult = true
cavebotMacro.setOn(ativado)
cavebotMacro.delay = nil
if ultimaConfig == nome then
    ui.list:focusChild(ui.list:getChildByIndex(indexAcaoAtual))
end
ultimaConfig = nome
end)
```

Cria-se então um evento **onClick()** para a propriedade **showEditor**, que constará com um **if not** para verificar se o editor possui um valor nulo, onde caso ele possua, deverá ser utilizado um return para que o evento não aconteça.

Em seguida, cria-se um **if** para verificar se a propriedade **showEditor** já está ativa, onde caso esteja, utiliza-se então o método **hide()** no editor. Deverá ser

utilizado também o método **setOn()** recebendo como parâmetro o valor booleano false.

Agora, deverá ser definido um **else** onde dentro dele, deverá ser utilizado método **show()** do editor, e o método **setOn()**, com o valor booleano true.

```
ui.showEditor.onClick = function()
  if not CaveBot.Editor then return end
  if ui.showEditor.isOn() then
    CaveBot.Editor.hide()
    ui.showEditor:setOn(false)
  else
    CaveBot.Editor.show()
    ui.showEditor:setOn(true)
  end
end
```

Em sequência, deverão ser realizados os mesmos passos da instrução anterior, porém, dessa vez, substituindo **showEditor** por **showConfig**.

```
ui.showConfig.onClick = function()
  if not CaveBot.Config then return end
  if ui.showConfig.isOn() then
    CaveBot.Config.hide()
    ui.showConfig:setOn(false)
  else
    CaveBot.Config.show()
    ui.showConfig:setOn(true)
  end
end
```

Agora, deverá ser definida uma função para o caveBot chamada **isOn()**, que deverá retornar a função **isOn()**, e o mesmo será feito para uma nova função **isOff()**.

```
CaveBot.isOn = function()
  return config.isOn()
end

CaveBot.isOff = function()
  return config.isOff()
end
```

Feito isso, deverá ser definida uma função **setOn()**, que receberá como parâmetro uma variável “val”, onde dentro do parâmetro deverá ser verificado se ela possui o valor false, onde caso isso seja atendido, ela deverá retornar o método **setOff()**, e caso contrário, ela deverá retornar o método **setOn()**. O mesmo será feito com uma função **setOff()**.

```
☐ CaveBot.setOn = function(val)
☐   if val == false then
       return CaveBot.setOff(true)
     end
     config.setOn()
  end

☐ CaveBot.setOff = function(val)
☐   if val == false then
       return CaveBot.setOn(true)
     end
     config.setOff()
  end
```

Deverá ser definido agora uma função **goToLabel()** que receberá como parâmetro uma variável “label”, esta mesma variável que receberá como valor ela mesma, juntamente do método **lower()**. Feito isso, deverá ser criado um **for** para percorrer a listas da variável “Ui”, juntamente do método **getChildren()**, onde, após isso, cria-se um **if** para verificar se o objeto “child” e sua variável “action” possuem o valor “label”, e verificar se a variável “value” do objeto “child” possui o mesmo valor da variável “label”.

Em seguida, deverá ser atribuída a propriedade **list** da variável “ui” o método **focusChild()**, passando o objeto “child” como parâmetro. Deve-se também criar dois returns, sendo um para realizar o retorno do valor true, e o outro para retornar o valor false.

```
☐ CaveBot.goToLabel = function(label)
  label = label:lower()
☐   for index, child in ipairs(ui.list:getChildren()) do
☐     if child.action == "label" and child.value:lower() == label then
         ui.list:focusChild(child)
         return true
       end
     end
  end
  return false
end
```

Tendo realizado o passo anterior, deverá ser criada uma função **save()**, onde dentro dela será definida uma tabela local “data”, criando juntamente um **for** que percorrerá a propriedade **list** da variável “ui”, junto do método **getChildren()**, utilizando, logo em seguida, um “table.insert” para inserir na tabela “data” o objeto “child”, junto de sua propriedade **action** e **value**.

Deverá agora ser criado um **if**, para verificar se a variável “config” possui um valor diferente de nulo, onde caso essa condição seja verdadeira, um “table.insert” será utilizado para preencher a tabela “data”, preenchendo-a com o texto “config” e com o método **json.encode()**, que receberá com parametro a variável “config” juntamente do método **save()**.

```
if CaveBot.Config then
    table.insert(data, {"config", json.encode(CaveBot.Config.save())})
end
```

Em seguida, cria-se uma tabela local chamada “extensionData”, juntamente de um **for** que irá percorrer a tabela “extensions”, além de definir um **if** para verificar se o evento **onSave()** do objeto “callbacks” irá retornar um valor diferente de nulo, e caso essa condição seja verdadeira, deverá ser criada uma variável local “extData”, recebendo o valor da variável “callBack” juntamente da função **onSave()**.

Logo após, cria-se um novo **if** para verificar se a variável “extData” é do tipo table, e que caso essa condição seja atendida, a variável “extData” terá seu valor atribuído para a tabela “Extension_data”.

Deve-se utilizar um “table.insert” para preencher a tabela “data” com uma string “extensions”, utilizando o método **json.encode()** recebendo como parâmetro a tabela “Extension_data” e o valor numérico 2. Em seguida, deve-se utilizar a tabela “save”, passando como parâmetro a tabela “data”.

```
local extension_data = {}
for extension, callbacks in pairs(CaveBot.Extensions) do
    if callbacks.onSave then
        local ext_data = callbacks.onSave()
        if type(ext_data) == "table" then
            extension_data[extension] = ext_data
        end
    end
end
table.insert(data, {"extensions", json.encode(extension_data, 2)})
config.save(data)
end
```

Assim como feito em todos os arquivos, será necessário colocar o diretório deste arquivo junto ao dos demais.

CASE 3.1

Criando a interface

Como feito no “Case 2.1”, neste também iremos desenvolver uma parte gráfica para o bot, assim será necessário criar um arquivo de extensão .otui, e importar o seu estilo no arquivo onde fica os diretórios de todos os arquivos que formam o bot.

De início, deverá ser criada uma label com o título “caveBotAction”, a qual terá um background color “alpha” e haverá um “text offset” de 2 e 0, que existirá juntamente da propriedade **focusable** que terá a valor true, e o foco terá um background color de seguinte código hexadecimal: #00000055.

```
CaveBotAction < Label
  background-color: alpha
  text-offset: 2 0
  focusable: true

  $focus:
    background-color: #00000055
```

Feito isso, deverá ser criado um painel, o “caveBotPanel” que contará com um layout do tipo “verticalBox”, e a propriedade **fit-children**, que contará com o valor true, além de ter que se criar uma separação horizontal que deverá ter uma margem ao seu topo de 2px, e a margem inferior 5px. Em seguida, é criado um painel que terá id “listPanel” e a altura de 100px, além de constar com uma margem superior de 100px.

Cria-se então uma “textList” com id “list” que terá uma ancoragem de preenchimento, uma barra de scroll vertical que terá o nome “listScrollBar”, uma margem à direita de 15px, sem a capacidade de ser focada manualmente, ela contará com uma função de auto foco que se concentrará no primeiro elemento.

```
CaveBotPanel < Panel
  layout:
    type: verticalBox
    fit-children: true

  HorizontalSeparator
    margin-top: 2
    margin-bottom: 5

  Panel
    id: listPanel
    height: 100
    margin-top: 2

  TextList
    id: list
    anchors.fill: parent
    vertical-scrollbar: listScrollBar
    margin-right: 15
    focusable: false
    auto-focus: first
```

Para a configuração da barra de rolagem lateral, é necessário que ela receba o id “listScrollBar”, que terá ancoragem superior, inferior e à direita em relação ao elemento pai, além de possuir como verdadeira a propriedade **pixels-scroll**, e possuirá um “step” de 10px.

```
VerticalScrollBar
  id: listScrollbar
  anchors.top: parent.top
  anchors.bottom: parent.bottom
  anchors.right: parent.right
  pixels-scroll: true
  step: 10
```

Em seguida, será criado um botão de alternância, cujo id será “showEditor” e possuirá uma margem superior de 2px, e que caso o botão esteja ativo, haverá de mostrar o texto “esconder waypoints”, e que caso esteja inativo, mostrará o texto “mostrar waypoints”.

```
BotSwitch
  id: showEditor
  margin-top: 2

  $on:
    text: Hide waypoints editor

  $!on:
    text: Show waypoints editor
```

Agora é criado um novo botão de alternância, que o id será “showConfig” e terá uma margem superior de 2px. Enquanto ativo, exibirá o texto “ocultar config”, e enquanto inativo, mostrará “mostrar config”.

```
BotSwitch
  id: showConfig
  margin-top: 2

  $on:
    text: Hide config

  $!on:
    text: Show config
```

Interface de configuração

Será necessário criar mais um arquivo de extensão .otui, desta vez para que seja criada a interface gráfica do menu de configuração

Para início, deve-se criar um painel “caveBotConfigPanel” que terá o id “caveBotEditor”, possuindo visibilidade igual a false, com um layout do tipo “verticalBox”, com a propriedade **fit-children** igual a true. Será necessário criar uma separação horizontal, com a margem superior de 5px.

Em seguida, deve-se criar uma label com texto alinhado ao centro “caveBotConfig” e terá uma margem superior de 15px.

```
CaveBotConfigPanel < Panel
  id: cavebotEditor
  visible: false

  layout:
    type: verticalBox
    fit-children: true

  HorizontalSeparator
    margin-top: 5

  Label
    text-align: center
    text: CaveBot Config
    margin-top: 5
```

Agora, deverá ser definido um painel “CaveBotConfigNumberValuePanel”, que terá altura de 20px e margem superior de 5p. Após isso, deverá ser criado um editor de texto “value”, que possuirá ancoragem superior, inferior e a direita ao elemento pai, que terá uma margem à direita de 5px e a largura de 50px.

```
CaveBotConfigNumberValuePanel < Panel
  height: 20
  margin-top: 5

  BotTextEdit
    id: value
    anchors.right: parent.right
    anchors.top: parent.top
    anchors.bottom: parent.bottom
    margin-right: 5
    width: 50
```

Em seguida, deverá ser definida uma label com a id “title”, com ancoragem à esquerda em relação ao elemento pai, e uma ancoragem vertical ao centro, e uma margem à esquerda de 5px.

```
Label
  id: title
  anchors.left: parent.left
  anchors.verticalCenter: prev.verticalCenter
  margin-left: 5
```

Deve-se agora criar um segundo painel, cujo nome deverá ser “CaveBotConfigBooleanValuePanel”, que terá altura de 20px, uma margem superior de 5px, e um botão de alternância com id “Value” que deverá conter um alinhamento superior, inferior e à direita do elemento pai, além de possuir margem a direita de 5px e largura de 50px, quando o botão estiver ativo constará com o texto “on” e quando inativo o texto “off”.

```
CaveBotConfigBooleanValuePanel < Panel
  height: 20
  margin-top: 5

  BotSwitch
    id: value
    anchors.right: parent.right
    anchors.top: parent.top
    anchors.bottom: parent.bottom
    margin-right: 5
    width: 50

    $on:
      text: On

    $!on:
      text: Off
```

Seguindo isso, deve-se criar uma nova label, que será idêntica à última label com nome “title”.

```
Label
  id: title
  anchors.left: parent.left
  anchors.verticalCenter: prev.verticalCenter
  margin-left: 5
```

Criando a interface do editor

Será necessário criar um novo arquivo de extensão .otui, neste será programado a interface do editor das configurações do bot.

Deve ser criado um novo botão, “CaveBotEditorButton”, juntamente da criação de um painel “CaveBotEditorPanel”, tendo a id “CaveBotEditor”, com visibilidade falsa, layout “verticalBox” e com a propriedade **fit-children** igual a true. Deverá então ser criada uma label “pos”, com texto alinhado ao centro “-”.

Será necessário criar um painel “buttons” que terá margem superior de 2px, layout do tipo “grid”, com “cell-size” de 86 e 20 px, um “cellspacing” de 1px, um “flow” igual a true e com a propriedade **fit-children** igual a true.

```
CaveBotEditorButton < Button

CaveBotEditorPanel < Panel
  id: cavebotEditor
  visible: false
  layout:
    type: verticalBox
    fit-children: true

Label
  id: pos
  text-align: center
  text: -

Panel
  id: buttons
  margin-top: 2
  layout:
    type: grid
    cell-size: 86 20
    cell-spacing: 1
    flow: true
    fit-children: true
```

Após isso, cria-se uma label com o texto: “Dê um duplo clique na ação para editá-la”, com alinhamento de texto no centro, possuindo propriedades de texto iguais a “auto resize”, “wrap”, ambas essas propriedades com valor true, possuindo margem superior, à direita e à esquerda de 2px.

```
Label
  text: Double click on action from action list to edit it
  text-align: center
  text-auto-resize: true
  text-wrap: true
  margin-top: 3
  margin-left: 2
  margin-right: 2
```

Por fim, deverá ser criado um botão de alternância que irá conter o id “Auto Recording”, possuindo o texto “Auto Recording”, tendo margem superior de 3px.

```
BotSwitch
  id: autoRecording
  text: Auto Recording
  margin-top: 3
```

Finalizando a interface

Será necessário criar mais um arquivo de extensão .otui, este irá conter os ajustes finais em relação a confecção da interface gráfica.

Para essa etapa, deverá ser criado um painel “SupplyItem”, possuindo 34px de altura. Em seguida, deverá ser definido um “BotItem” e “Item”, com tamanho de 32x32px, ancoragem superior e à esquerda do elemento pai, além de uma margem superior de 1px.

Após isso, haverá de definir um painel “Fields”, tendo ancoragem nos quatro sentidos (direita, esquerda, cima e baixo) em relação ao elemento pai, além de ter uma margem tanto a direita quanto à esquerda de 2px.

```
SupplyItem < Panel
  height: 34

  BotItem
    id: item
    size: 32 32
    anchors.left: parent.left
    anchors.top: parent.top
    margin-top: 1

  Panel
    id: fields
    anchors.top: parent.top
    anchors.bottom: parent.bottom
    anchors.left: prev.right
    anchors.right: parent.right
    margin-left: 2
    margin-right: 2
```

Feito isso, é criada então uma label “minLabel”, com ancoragem à direita e à esquerda do elemento pai, possuindo sua ancoragem à direita centralizada em relação ao elemento pai, tendo margem à direita de 2px, e um texto alinhado ao centro “Min”.

Em sequência, deverá ser criada uma label idêntica a essa última criada, porém alternando os valores “min” para “max”.

```
Label
  id: minLabel
  anchors.top: parent.top
  anchors.left: parent.left
  anchors.right: parent.horizontalCenter
  margin-right: 2
  text-align: center
  text: "Min"

Label
  id: maxLabel
  anchors.top: parent.top
  anchors.left: parent.horizontalCenter
  anchors.right: parent.right
  margin-left: 2
  text-align: center
  text: "Max"
```

Como próximo passo, deverá ser criado um editor de texto “Min”, com ancoragem superior em relação à parte superior da label “min”, com ancoragem à direita e à esquerda também em relação a essa mesma label. Seu texto estará alinhado no centro, contendo o texto “1”.

O mesmo processo será feito de novo, porém, alterando novamente, o texto “min” para “max”.

```
BotTextEdit
  id: min
  anchors.top: minLabel.bottom
  anchors.left: minLabel.left
  anchors.right: minLabel.right
  text-align: center
  text: 1
```

```
BotTextEdit
  id: max
  anchors.top: maxLabel.bottom
  anchors.left: maxLabel.left
  anchors.right: maxLabel.right
  text-align: center
  text: 100
```

Agora, deverá ser definido um painel “SupplyItemList”, com altura de 102px, além de ser necessário criar um painel que seja possível utilizar o scroll, com id “List”, tendo ancoragem de preenchimento em relação ao elemento pai, além de possuir uma barra de scroll vertical “scroll”, com margem à direita de 7px, layout do tipo “verticalBox”, com altura de 34px.

Também é necessário definir uma outra barra de rolagem, que possuirá o id “scroll”, com ancoragem superior, inferior e a direita do elemento pai, com “step” de 10px, além de possuir a propriedade “pixelsScroll” como true.

```
SupplyItemList < Panel
  height: 102

  ScrollablePanel
    id: list
    anchors.fill: parent
    vertical-scrollbar: scroll
    margin-right: 7
    layout:
      type: verticalBox
      cell-height: 34

  BotSmallScrollBar
    id: scroll
    anchors.top: prev.top
    anchors.bottom: prev.bottom
    anchors.right: parent.right
    step: 10
    pixels-scroll: true
```

ENCERRAMENTO DO APRENDIZADO

“A simplicidade da Lua reside na sua capacidade de expressar poderosos conceitos com clareza e concisão, tornando cada linha de código uma obra-prima funcional.” - Roberto Ierusalimsky, co-criador da linguagem Lua.

Ao longo desta jornada, você não apenas adquiriu conhecimentos práticos sobre a configuração de ambientes de desenvolvimento, mas também desvendou os mistérios da sintaxe Lua, habilmente aplicando esses conhecimentos na construção de um bot personalizado para aprimorar suas experiências no mundo virtual do Tibia.

Essa conquista representa mais do que apenas um tutorial concluído; é a manifestação de sua dedicação em compreender os fundamentos da programação e da automação. Você já possui um entendimento firme dos princípios que embasam o assunto e está preparado para descobrir novas oportunidades.

Considere este momento como um trampolim para aventuras mais avançadas. Experimente com novos projetos, desafie-se a alcançar novos patamares e compartilhe seu conhecimento com a comunidade. Lembre-se de que a programação é uma jornada constante de aprendizado, e cada desafio superado contribui para sua evolução como desenvolvedor.

Enquanto avança, mantenha-se atualizado com as tendências da programação e explore outras linguagens que possam ampliar ainda mais seu repertório. Você acabou de dar os primeiros passos em uma jornada fascinante pelo amplo universo da tecnologia. Parabéns por alcançar esta etapa inicial!

REFERÊNCIAS

PARTE I – Básico do básico de Lua. 21 dez. 2010. Fórum TibiaKing. Disponível em: https://tibiaKing.com/forums/topic/451-tutorial-b%C3%A1sico-sobre-lua/?_fromLogin=1. Acesso em: 16 ago. 2023.

OTCLIENTV8 BOT. 26 out. 2019. Fórum OtLand. Disponível em: <https://otland.net/threads/otclientv8-bot.266958/>. Acesso em: 10 ago. 2023.

PRE-COMPILED Lua libraries and executables. [S. l.]. Disponível em: <https://luabinaries.sourceforge.net/>. Acesso em: 17 abr. 2023.

IERUSALIMSCHY, Roberto. Uma Introdução à Programação em Lua. Artigo, [S. l.]. Disponível em: <https://www.lua.org/doc/jai2009.pdf>. Acesso em: 15 maio 2023.

IERUSALIMSCHY, Roberto; CELES, Waldemar; FIGUEIREDO, Luiz Henrique de. A Linguagem Lua e suas Aplicações em Jogos. Artigo [S. l.]. Disponível em: <https://www.lua.org/doc/wjogos04.pdf>. Acesso em: 27 abr. 2023.

LABLUA. Lua: Conceitos Básicos e API C. 2008. Artigo - PUC-Rio, [S. l.], 2008. Disponível em: https://www.lua.org/doc/apostila_lua_2008.pdf. Acesso em: 18 maio 2023.

NOÇÕES de Lua 3.1: Noções básicas da linguagem de programação Lua. Artigo [S. l.]. Disponível em: <http://profs.ic.uff.br/~otton/graduacao/informatical/noco-es-3.1.pdf>. Acesso em: 26 maio 2023.