

**WILLIAN AYRES WEB DEVELOPER**  
**RELATÓRIO DO PROJETO**  
**SIMULADOR DE RESTA UM**

Willian Joris Ayres

**RELATÓRIO FINAL DO PROJETO**

**SOFTWARE PARA SIMULADOR DE RESTA UM**

Distração e entretenimento

Curitiba

Novembro / 2019

Willian Joris Ayres

## **RELATÓRIO FINAL DO PROJETO**

### **SOFTWARE PARA SIMULADOR DE RESTA UM**

Distração e entretenimento

Relatório final de projeto, apresentado como  
requisito de confirmação de conhecimentos  
no desenvolvimento de softwares em  
linguagem de programação C.

Curitiba

Novembro / 2019

# Sumário

<b>1. Lista de Ilustrações .....</b>	<b>3</b>
<b>2. Lista de Apêndices .....</b>	<b>4</b>
<b>3. Introdução .....</b>	<b>5</b>
<b>4. Explicação do Simulador .....</b>	<b>6</b>
<b>5. Codificação do Simulador .....</b>	<b>12</b>
<b>6. Considerações Pessoais .....</b>	<b>21</b>
<b>7. Referências .....</b>	<b>22</b>

## Lista de ilustrações

• IMG1 – Tela do Menu Principal .....	7
• IMG2 – Instruções/Regras .....	7
• IMG3 – Menu de histórico .....	8
• IMG4 – Histórico .....	8
• IMG5 – Records .....	8
• IMG6 – Records ➔ Página 1 .....	9
• IMG7 – Records ➔ Página 2 .....	9
• IMG8 – Records ➔ Página 3 .....	9
• IMG9 – Records ➔ Página 4 .....	9
• IMG10 – Menu de tabuleiro .....	10
• IMG11 – Visualização do tabuleiro .....	10
• IMG12 – Leitura do Nome/Apelido .....	11
• IMG13 – Jogada acontecendo .....	11
• IMG14 – Jogada inválida .....	11
• IMG15 – Jogada indesejada .....	12
• IMG16 – Pressionar R .....	12
• IMG17 – Jogada retornada .....	12
• IMG18 – Jogo finalizado .....	12
• IMG19 – Histórico atualizado .....	12

## Lista de apêndices

• Ap1 – Função main .....	13
• Ap2 – Função inicio .....	13
• Ap3 – Leitura regras .....	13
• Ap4 – Reset do arquivo de histórico .....	14
• Ap5 – Ler arquivo de records .....	14
• Ap6 – Struct tabuleiro .....	14
• Ap7 – Escolha de tabuleiro .....	15
• Ap8 – Função auxiliar .....	15
• Ap9 – Declaração do tabuleiro .....	15
• Ap10 – Visualização do tabuleiro .....	16
• Ap11 – Struct tempo .....	16
• Ap12 – Início do jogo .....	16
• Ap13 – Impressão do tabuleiro e informações .....	17
• Ap14 – Inserção do nome do usuário .....	17
• Ap15 – Verificação de jogadas possíveis .....	17
• Ap16 – Verificação de resultado .....	17
• Ap17 – Leitura de dados para a jogada .....	18
• Ap18 – Caso o usuário desista .....	18
• Ap19 – Verificação de jogada inválida .....	18
• Ap20 – Atualiza o campo conforme jogada .....	19
• Ap21 – Continuidade conforme jogada .....	19
• Ap22 – Verificação de novo record .....	19
• Ap23 – Separação em arquivos auxiliares .....	20
• Ap24 – Concatenação de arquivos auxiliares .....	20
• Ap25 – Atualização do arquivo de record .....	20
• Ap26 – Menu auxiliar para fim de jogo .....	21
• Ap27 – Função para abrir arquivo texto .....	21

## Introdução

Este documento apresenta a versão final do simulador do jogo “Resta Um” feito por Willian Ayres. Tendo em vista como aperfeiçoamento de conhecimentos prévios de lógica de programação, algoritmos, desenvolvimento de softwares na linguagem C, além da melhoria na abstração de resolução de problemas pelo meio procedural.

O contexto de codificação deu-se pela IDE “Code Blocks”, sendo de fácil acesso e gratuito. Existem vários motivos para utilização dessa IDE, por exemplo, ela possui várias ferramentas adicionais, como os “debuggers” e os “compilers”.

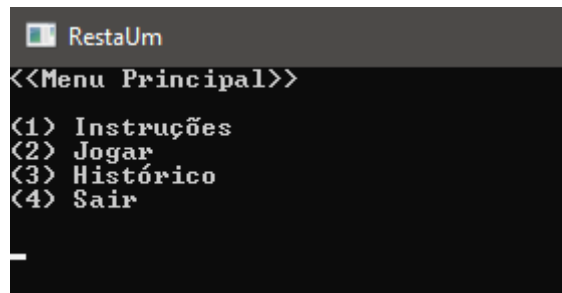
O código, em sua integridade, é escrito completamente em C, sem muitas enrolações. É compreensível e conciso, porém é necessário um nível intermediário de lógica, conhecimentos da linguagem C e abstração. Algumas partes no desenvolver do programa podem se tornar um pouco complexas caso o desenvolvedor não esteja habituado. O programa foi otimizado até o ponto que o conhecimento atual permitiu.

Se você é atraído pelas belezas do mundo da programação, seguir esse relatório para criar o seu próprio simulador de Resta Um não será um grande problema. Basta seguir corretamente e ajustar conforme suas métricas e seus próprios gostos.

## Explicação do Simulador

Por meio de movimentos válidos (horizontais ou verticais), pulando “pinos” e ocupando casas vazias, o pino pulado é retirado do jogo gerando mais casas vazias, assim o jogo termina quando restar apenas um “pino” na posição central do tabuleiro do jogo. O jogador perde quando não houver mais jogadas possíveis e restando mais de um “pino” no tabuleiro.

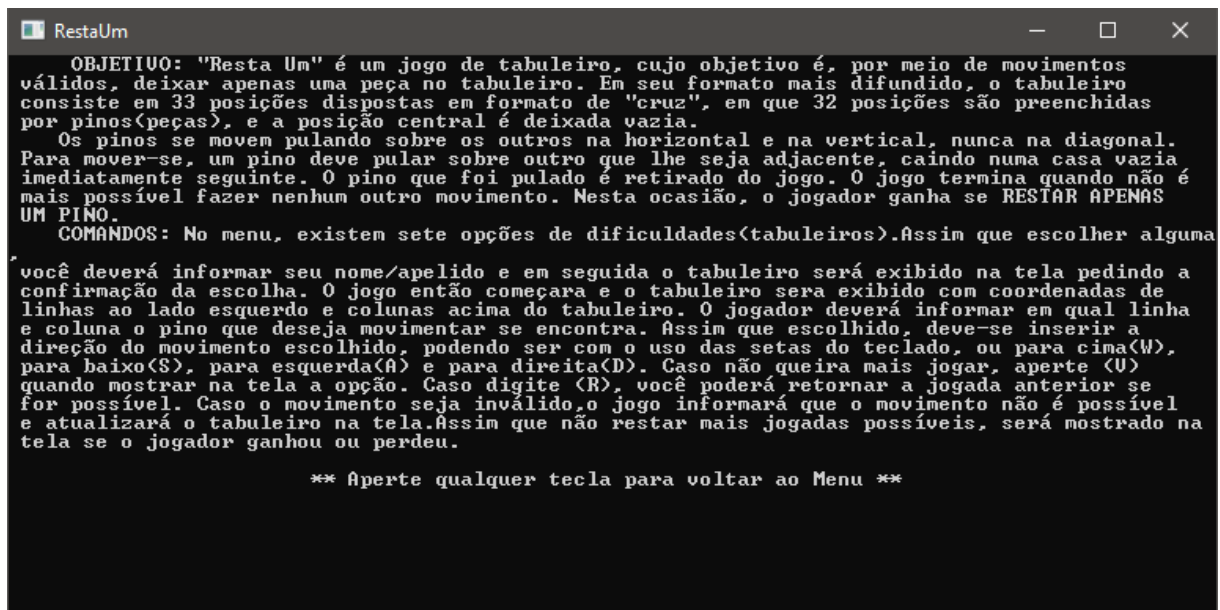
Assim que iniciado o programa com o executável, abre-se a janela do console com nome de “Resta Um”, e printando na tela as opções do menu inicial.



```
RestaUm
<<Menu Principal>>
<1> Instruções
<2> Jogar
<3> Histórico
<4> Sair
```

>> Tela do menu principal

Com a escolha de 4, o console é automaticamente fechado. Caso 1, é aberto a aba de instruções do simulador, contendo todas as regras, funcionamento e controles do jogo.

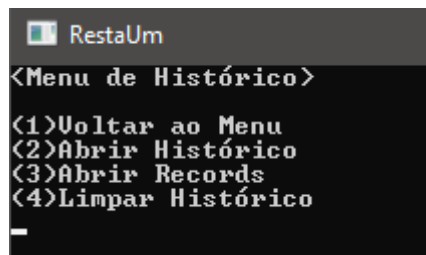


```
RestaUm
OBJETIVO: "Resta Um" é um jogo de tabuleiro, cujo objetivo é, por meio de movimentos válidos, deixar apenas uma peça no tabuleiro. Em seu formato mais difundido, o tabuleiro consiste em 33 posições dispostas em formato de "cruz", em que 32 posições são preenchidas por pinos(peças), e a posição central é deixada vazia.
Os pinos se movem pulando sobre os outros na horizontal e na vertical, nunca na diagonal. Para mover-se, um pino deve pular sobre outro que lhe seja adjacente, caindo numa casa vazia imediatamente seguinte. O pino que foi pulado é retirado do jogo. O jogo termina quando não é mais possível fazer nenhum outro movimento. Nesta ocasião, o jogador ganha se RESTAR APENAS UM PINO.
COMANDOS: No menu, existem sete opções de dificuldades(tabuleiros).Assim que escolher alguma, você deverá informar seu nome/apelido e em seguida o tabuleiro será exibido na tela pedindo a confirmação da escolha. O jogo então começara e o tabuleiro sera exibido com coordenadas de linhas ao lado esquerdo e colunas acima do tabuleiro. O jogador deverá informar em qual linha e coluna o pino que deseja movimentar se encontra. Assim que escolhido, deve-se inserir a direção do movimento escolhido, podendo ser com o uso das setas do teclado, ou para cima(W), para baixo(S), para esquerda(A) e para direita(D). Caso não queira mais jogar, aperte (U) quando mostrar na tela a opção. Caso digite (R), você poderá retornar a jogada anterior se for possível. Caso o movimento seja inválido,o jogo informará que o movimento não é possível e atualizará o tabuleiro na tela.Assim que não restar mais jogadas possíveis, será mostrado na tela se o jogador ganhou ou perdeu.

** Aperte qualquer tecla para voltar ao Menu **
```

>> Instruções/Regras

Caso a escolha seja 3 no menu principal (Histórico), é aberta a aba de opções relativas à leitura de arquivos de histórico e records.



>> Menu de histórico

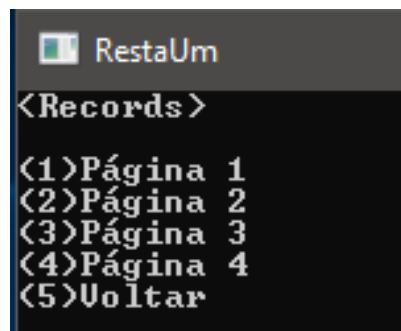
Caso 1, ele volta ao menu principal. Caso 2, abre o arquivo de histórico para a leitura de algumas informações dos jogos finalizados recentemente. Ele sempre mostrará as últimas informações gravadas. Dados obsoletos podem ser desconsiderados.

Nome	Tabuleiro	Resultado	Jogadas	Tempo
will	Tabuleiro L	Ganhou	3	0:14
will	Tabuleiro L	Ganhou	3	0:06
w	Tabuleiro L	Ganhou	3	0:05
ary	Tabuleiro Iniciante	Ganhou	2	0:09
will	Tabuleiro Oito	Perdeu	10	1:09
wii	Tabuleiro Iniciante	Ganhou	8	1:34
Lukinha	Tabuleiro Francês Alternativo	Perdeu	42	6:35
ml	Tabuleiro X	Ganhou	8	2:16
ml	Tabuleiro Padrão Inglês	Perdeu	3	1:36
caue	Tabuleiro Padrão Inglês	Ganhou	31	5:47
WILL	Tabuleiro Padrão Inglês	Perdeu	2	0:20
will	Tabuleiro Mais	Ganhou	10	0:39

\*\* Aperte qualquer tecla para voltar ao Menu \*\*

>> Histórico

Caso a escolha seja 3 no menu de histórico, é aberto a aba para escolher a qual página deseja verificar o record. Cada página possui records referentes a tabuleiros distintos. Nesse menu, caso a escolha seja 5, é retornado ao menu de histórico.



>> Records



RestUm				
	Tabuleiro	Resultado	Jogadas	Tempo
None				
caue	Tabuleiro Padrão Inglês	Ganhou	31	5:47
oi	Tabuleiro Padrão Francês	Ganhou	40	9:59
ary	Tabuleiro Iniciante	Ganhou	2	0:09
hello	Tabuleiro L	Ganhou	3	0:05
21	Tabuleiro P	Ganhou	40	9:59
22	Tabuleiro Cruz	Ganhou	40	9:59
23	Tabuleiro Quadrados	Ganhou	40	9:59
vill	Tabuleiro Mais	Ganhou	10	0:39
ml	Tabuleiro X	Ganhou	8	2:16
v	Tabuleiro Espelho	Ganhou	40	9:59
** Aperte qualquer tecla para voltar ao Menu **_				

## >> Records → Página 1

RestUm				
	Tabuleiro	Resultado	Jogadas	Tempo
None				
vi	Tabuleiro Canto	Ganhou	40	9:59
225	Tabuleiro Oito	Ganhou	40	9:59
ven	Tabuleiro Cartola	Ganhou	40	9:59
wall	Tabuleiro Casinha	Ganhou	40	9:59
35	Tabuleiro Plateia	Ganhou	40	9:59
vill	Tabuleiro Abajur	Ganhou	40	9:59
av	Tabuleiro Alvo	Ganhou	40	9:59
29	Tabuleiro Triângulo	Ganhou	40	9:59
vill	Tabuleiro Anel	Ganhou	40	9:59
w2	Tabuleiro Flecha	Ganhou	40	9:59
** Aperte qualquer tecla para voltar ao Menu **				

## >> Records → Página 2

RestUm				
	Tabuleiro	Resultado	Jogadas	Tempo
None				
ary	Tabuleiro Auto Estrada	Ganhou	40	9:59
caue	Tabuleiro Ampulheta	Ganhou	40	9:59
caue	Tabuleiro Barra	Ganhou	40	9:59
vill	Tabuleiro Círculo	Ganhou	40	9:59
39	Tabuleiro Cristal	Ganhou	40	9:59
04	Tabuleiro Prisão	Ganhou	40	9:59
11	Tabuleiro X Megrito	Ganhou	40	9:59
anna	Tabuleiro Foguete	Ganhou	40	9:59
04	Tabuleiro Caveira	Ganhou	40	9:59
30	Tabuleiro 5 Cruzes	Ganhou	40	9:59
** Aperte qualquer tecla para voltar ao Menu **_				

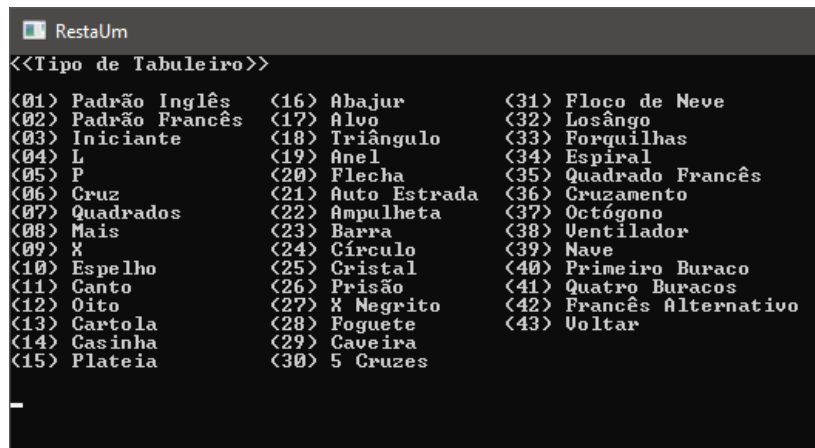
## >> Records → Página 3

RestUm				
	Tabuleiro	Resultado	Jogadas	Tempo
None				
vill	Tabuleiro Floco de Neve	Ganhou	40	9:59
vill	Tabuleiro Losango	Ganhou	40	9:59
vill	Tabuleiro Forquilhas	Ganhou	40	9:59
vill	Tabuleiro Espiral	Ganhou	40	9:59
22	Tabuleiro Quadrado Francês	Ganhou	40	9:59
33	Tabuleiro Cruzamento	Ganhou	40	9:59
44	Tabuleiro Octógono	Ganhou	40	9:59
55	Tabuleiro Ventilador	Ganhou	40	9:59
66	Tabuleiro Nave	Ganhou	40	9:59
vill	Tabuleiro Primeiro Buraco	Ganhou	40	9:59
77	Tabuleiro Quatro Buracos	Ganhou	40	9:59
88	Tabuleiro Francês Alternativo	Ganhou	40	9:59
** Aperte qualquer tecla para voltar ao Menu **_				

## >> Records → Página 4

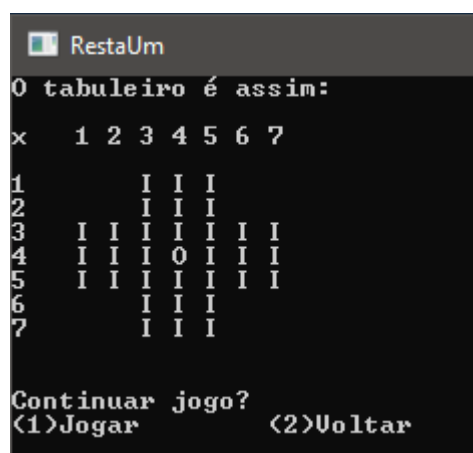
Caso seja escolhida a opção 4 no menu de histórico, as informações salvas dos últimos jogos são apagadas, ou seja, na hora da leitura de histórico, ele estará vazio.

Retorna ao menu principal, assim que escolhida a opção 2 (Jogar), abre-se a aba de escolha de tabuleiro. Nela você pode escolher entre as 42 opções de tabuleiro jogáveis, basta digitar o número correspondente ao tabuleiro que deseja jogar.

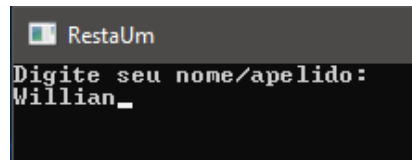


## >> Menu de tabuleiros

Caso a escolha seja 43, ele retorna ao menu principal. Caso escolha de 1 a 42, o simulador imprime o tabuleiro para mostrar ao jogador se realmente deseja jogar nesse tabuleiro. Caso o jogador digite 2, ele retorna ao menu de tabuleiros. Caso 1, ele pede o nome do jogador e então começa o jogo.

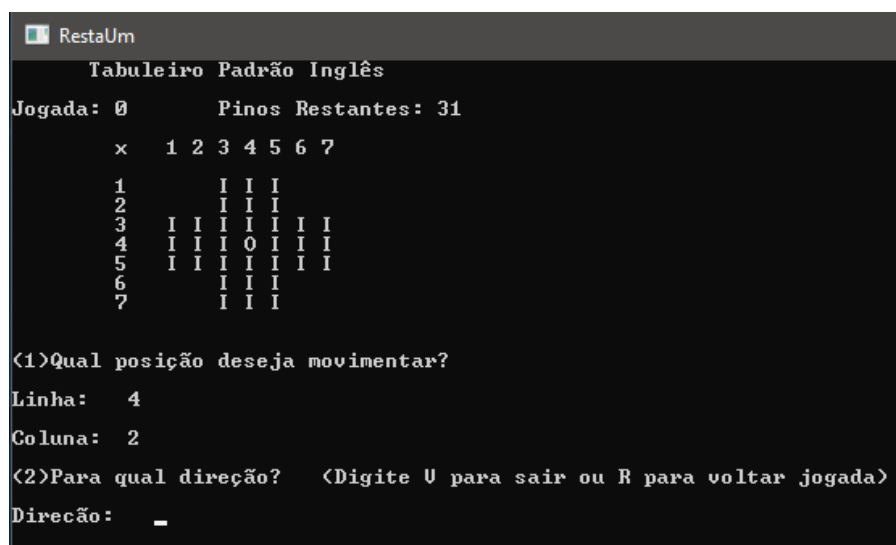


## >> Visualização do tabuleiro

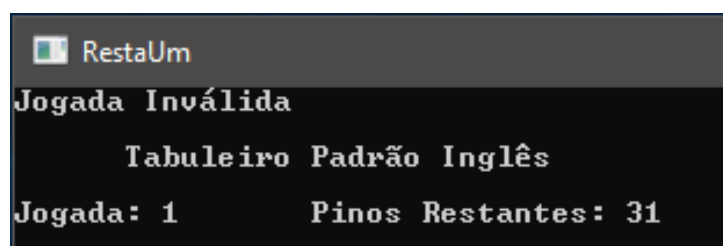


### >> Leitura do Nome/Apelido

Assim que iniciado, o simulador imprime na tela o tabuleiro e seu nome, com número de jogadas, pinos faltantes, e pede que jogador digite a linha e coluna que o pino que deseja movimentar se encontra, além da direção do movimento. Se a teclado V, o jogo retorna ao menu inicial. Caso a jogada seja inválida, aparecerá um aviso e terá de repetir a jogada.



### >> Jogada acontecendo



### >> Jogada inválida

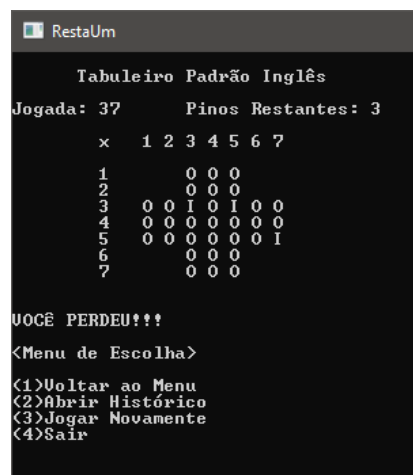
Se você executar uma jogada que não era planejada, mudou sua estratégia ou até mesmo apertou sem querer, na hora de definir a direção do movimento, você pode retornar a jogada apertando a tecla R, voltando ao campo da jogada anterior, porém a contagem de jogadas continua a mesma.



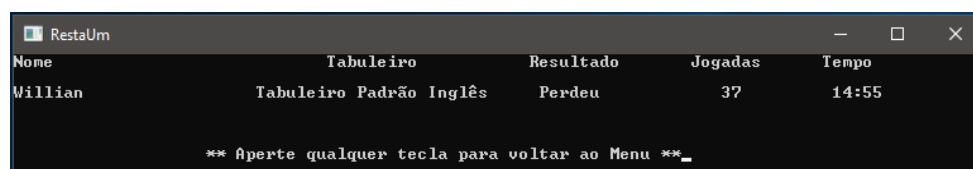
>> Jogada indesejada >> Pressionar R

>> Jogada retornada

Assim que terminado, é mostrado na tela se o jogador venceu ou perdeu e o menu de escolha para escolher entre: sair (o programa é finalizado); voltar ao menu (o programa volta ao menu inicial); abrir o histórico (abre o histórico como na IMG4); e jogar novamente (o programa se repete a partir do menu de tabuleiros).



>>Jogo finalizado



>> Histórico atualizado

## Codificação do Simulador

As bibliotecas utilizadas para o programa foram: stdio.h, stdlib.h, string.h, locale.h, conio.h, ctype.h, time.h. Além dos cabeçalhos .h que foram separados para melhor visualização do programa: Menu.h, Tabuleiro.h, Estatistica.h, Principal.h.

O simulador de Resta Um inicia-se com a função setlocale para a adição de caracteres especiais, a função system para alterar o nome do console e então, chamada da função de menu inicial, executado pela função Inicio ().

```
#include <locale.h>
#include "Menu.h"
int main () {
    setlocale (LC_ALL, "portuguese");
    system ("title RestaUm");
    system ("mode con: cols=100 lines=30");
    Inicio ();
    return 0;
}
```

```
void Inicio () {
    int op;
    do {
        printf ("Menu\n");
        printf ("1-\n2-\n3-\n4-");
        op = getch ();
        menu (&op);
    } while (op > 52 || op < 49);
}
```

```
FILE *fileregra = fopen ("regra.txt", "r");
if (fileregra == NULL) exit (1);
int n = 0;
char c = fgetc (fileregra);
while (c != EOF) {
    n++;
    c = fgetc(fileregra);}
rewind (fileregra);
char *leitura = (char *) calloc (n, sizeof(char));
while (fgets (leitura, n, fileregra) != NULL)
    printf ("%s", leitura);
fclose (fileregra);
free (leitura);
getch ();
```

Assim que chamada a função Inicio (), é mostrado na tela as opções e então solicitado ao usuário/jogador que insira o número correspondente a opção escolhida. O endereço desse número ( $1 < *op < 4$ ) é então passado como parâmetro de referência para a função menu (int \*op), onde passa por uma estrutura de decisão do tipo switch. Caso \*op=1, então, na mesma função, é chamado um ponteiro do tipo FILE para a leitura do arquivo texto de regras, presente no mesmo diretório do programa Resta Um. Primeiro é varrido o arquivo para saber seu tamanho, então trabalhado dinamicamente, com o tamanho já lido anteriormente. Usa-se o getch () para esperar entrada aleatório do usuário após ler as informações do arquivo de instruções.

Caso escolhido `*op=3` chama-se a função para abrir um menu de estatísticas. Nessa função de estatísticas, novamente executa-se um switch de uma outra variável qualquer para decidir entre: abrir o histórico de jogos, abrir um menu para páginas de records, ou limpar o arquivo de histórico. Para a leitura do histórico, utiliza-se uma função similar ao adendo de código 3, porém abrindo um arquivo texto diferente, formatado a sua preferência. Se escolher a opção de limpar o arquivo, basta reescrever o arquivo já existente com as informações zeradas, no caso desse simulador seria algo do gênero:

```
FILE *file = fopen ("arquivo.txt", "w");

char aux [] = "Nome          Tabuleiro      Resultado   Jogadas    Tempo\n\n";

fprintf (file, "%s", aux);

fclose (file);
```

```
void abreRecord (int r) {

    char aux [13] = "ranking";

    aux [7] = r;

    strcat (aux, ".txt");

    FILE *file = fopen (aux, "r");

    char c;

    while (c != EOF) {

        if (c != '$') printf ("%c", c);

        if (c == '$') printf (" ");

        c = fgetc (file);

    }

    fclose (file);

    getch ();}
```

Caso a escolha seja o menu de records, precisa-se fazer outro switch para escolher qual das páginas de records (cada uma possui um arquivo texto diferente) deseja abrir para leitura. Uma opção deve ser deixada para retornar a esse menu de histórico. Dependendo da escolha, abre-se o arquivo correspondente a essa opção. Pode-se utilizar uma função a parte, passando o valor do arquivo para identificar qual será aberto.

Primeiramente, declara-se a string auxiliar, depois adiciona o número a ela e por último a extensão. Após isso é só abrir o arquivo com o nome gerado. Os '\$' são para definir marcadores na hora de conferir se os records foram batidos, basta ignorá-los e fazer a leitura normalmente.

Com isso, finaliza-se a parte de leitura dos arquivos textos, e podemos retornar ao menu principal. Nele, caso `*op=4`, o programa chama a função `exit (0)`, para retornar falso e fechar o programa. Caso `*op=2`, imprime-se na tela todas as opções de tabuleiros possíveis (como consta na IMG10), e espera que uma das 42 opções seja escolhida pelo usuário.

Antes de continuar, precisa-se declarar uma struct com algumas informações do tabuleiro como: um contador para jogadas, contador de pinos no campo, resultado da partida, nome do jogador, nome do campo, e um array multidimensional 9x9 para armazenar o campo. É usado typedef para facilitar na hora de chamada de parâmetros e passada de argumentos.

```
typedef struct {

    int contador;

    int pinos;

    char resultado [7];

    char jogador [20];

    char nome [30];

    char campo [9][9];

} tabuleiro;
```

Após escolhida uma das 42 opções, é declarado um ponteiro para struct do tipo tabuleiro, chamada uma função para ler o tabuleiro do arquivo texto, e outra para visualizar previamente o tabuleiro.

```
int opS, p [2];

printTabs ();

do {

    p [0] = getche () - 48;

    p [1] = getche () - 48;

    opS = 10 * p [0] + p [1];

} while (opS < 1 || opS > 43);

if (opS == 43) Inicio ();

else menuS (opS);
```

```
void menuS (int opS) {

    tabuleiro *t1 = (tabuleiro *) malloc (sizeof(tabuleiro));

    int n = opS - 1;

    declaraTabu (t1, n);

    preVisu (t1, n);

}
```

```
t1->contador = 0;

t1->pinos = 0;

char ch, str [82];

int aux = 0;

FILE *file = fopen ("tabuleiro.txt", "r");

if (file == NULL) {printf ("ERRO!"); exit (1);}

while (ch != EOF){

    if (aux == n && ch != '\n')

        fscanf (file, "%s", str);

    if (aux == n+42 && ch != '\n') {

        fgets (t1->nome, 30, file);

        break;}

    if (ch == '\n') aux++;

    ch = getc (file);

}

for (int i = 0; i < (strlen(t1->nome)); i++) {

    if (t1->nome [i] == '\n') t1->nome [i] = '0';}

for (int i = 0; i < strlen(str); i++) {

    if (str [i] == 'T') t1->pinos++;

}

memcpy (t1->campo, str, sizeof(t1->campo));

fclose (file);
```

Cada opção contém a mesma inicialização de jogo, indiferente do tabuleiro, porem o parâmetro n muda conforme a opção de jogo escolhida, para quando chamado a função declaraTabu (tabuleiro \*t1, int n), a leitura do arquivo texto que contém as informações de: nome do tabuleiro e formato do tabuleiro possam ser lidas e armazenadas corretamente, já que cada tabuleiro se encontra em uma linha do arquivo texto, e n serve como parâmetro para o encontro dessa linha no arquivo. Além de puxar essas informações, a função inicia o contador de jogadas com 0. Para o campo é declarado uma string [82] e um ponteiro FILE para leitura do arquivo texto do tabuleiro respectivo e, em seguida, atribuído à string o tabuleiro em formato de palavra, presente na linha n do arquivo, que estava salvo no arquivo. Por meio da função memcpy, o campo é então convertido de uma string para uma matriz com o que havia sido gravado na string. Além do campo, é puxado o nome do tabuleiro, percorrido esse nome para ter certeza de adicionar o caractere de final de string, e percorrido a string do campo para contar a quantidade de pinos. A formatação para leitura do arquivo texto e recuperação das informações, varia conforme métrica do desenvolvedor.

Após a inicialização do tabuleiro, é chamada a função `preVisu (t1)` para visualizar o tabuleiro e decidir se quer continuar o jogo nesse tabuleiro. Caso 2, o programa volta ao menu secundário para escolha de outro tabuleiro. Na impressão, então, abre-se dois laços para percorrer a matriz, e impresso caractere por caractere, com formatação de preferência do desenvolvedor. Após impressão, é perguntado se o usuário realmente deseja continuar com aquele tabuleiro ou deseja trocar por algum outro e então, lido do teclado a opção do jogador. Caso desista de jogar nesse tabuleiro, o programa retorna à escolha de tabuleiros. Caso 1, é chamada função de sucessão do jogo `jogEmMenu (t1)`.

```
system("cls");

char pos_x [11] = "x 1234567 ", pos_y [10] = " 1234567 ";

printf ("O tabuleiro é assim:\n\n");

for (int i = 0; i < 10; i++) printf ("%c ", pos_x [i]);

printf ("\n");

for (int i = 0; i < 9; i++) {

    printf ("%c ", pos_y [i]);

    for (int j = 0; j < 9; j++) {

        if (t1->campo [i] [j] == '*') printf (" ");

        else printf ("%c ", t1->campo[i] [j]);

    }

    printf ("\n");

}

printf ("\nContinuar jogo?\n(1)Jogar\n(2)Voltar\n");

int no;

do {no = getch ();} while (no > 50 || no < 49);

if (no == 49) jogEmMenu (t1, n); else {free (t1); declaraTabu ();}
```

Para continuar, precisa-se de outra struct com duas variáveis para armazenar os valores do controle do timer do jogo. Usa-se o comando `typedef` para facilitar o uso da struct no decorrer do programa.

```
typedef struct {

    int mint;

    int segt;

} tempo;
```

Na função `jogo (t1)`, primeiramente é inicializado duas structs `tm` (da biblioteca `time.h`) para o controle do cronômetro de jogo. Assim que armazenados corretamente os dados de minuto e segundos, é inicializado o nome do jogador do campo nome da struct tabuleiro. Então, é chamada uma função de impressão do tabuleiro `imprimiInfo (t1)`, que mostra na tela a jogada atual, número de pinos, nome do tabuleiro, as linhas e colunas do tabuleiro e por último os pinos e as posições válidas do tabuleiro.

```
tempo *temp = (tempo *) malloc (sizeof(tempo));

struct tm *hora_atual, *hora_atual2;

time_t sec, sec2;

int seg, seg2;

time (&sec); hora_atual = localtime (&sec);

seg = hora_atual->tm_sec + 60*(hora_atual->tm_hour*60 + hora_atual->tm_min);

leNome (t1);

imprimiInfo (t1);

executaJogada (t1, temp, seg);
```



```

void imprimiInfo (tabuleiro *t1) {

    printf (" ");

    for (int i = 0; t1->nome [i] != '\0'; i++)

        printf ("%c", t1->nome [i]);

    printf ("\n\nJogada: %d\t", t1->contador);

    printf ("Pinos Restantes: %d\n\n\t", t1->pinos);

    imprimiTabu (t1, "\t");

}

```

```

void leNome (tabuleiro *t1) {

    system ("cls");

    printf ("Digite seu nome/apelido: \n");

    setbuf (stdin, NULL);

    fgets (t1->jogador, 20, stdin);

    t1->jogador [strlen (t1->jogador) - 1] = '\0';

    system ("cls");

}

```

Após a impressão, é chamada a função recursiva `executaJogada (tabuleiro *t1, tempo *temp, int segu)` com os parâmetros: ponteiro pra struct `tabuleiro`; ponteiro pra struct `tempo`; e um inteiro `segu` que armazenou a quantidade de segundos para controle de tempo. A função `executaJogada (t1, temp, segu)` é chamada até não existirem mais jogadas possíveis. A primeira coisa a ser executada dentro dela, é a chamada de duas funções auxiliares: `verificaJogo (t1)` para conferir se existem jogadas possíveis (se existe pelo menos um pino ligeiramente próximo ao outro), que retorna 0 se não existir; `GameOver(t1)` para conferir caso o retorno de 0 na função anterior, jogador venceu, e 1, jogador perdeu.

```

int verificaJogo (tabuleiro *t1) {

    int v = 1;

    for (int i = 1; i < 9; i++) {

        for (int j = 1; j < 9; j++) {

            if ((t1->campo [i] [j] != '*') && (t1->campo [i] [j] == 'T') &&

                ((t1->campo [i + 1] [j] == 'T' && t1->campo [i + 2] [j] != '*' && t1->campo [i + 2] [j] != 'T') ||

                 (t1->campo [i] [j + 1] == 'T' && t1->campo [i] [j + 2] != '*' && t1->campo [i] [j + 2] != 'T') ||

                 (t1->campo [i - 1] [j] == 'T' && t1->campo [i - 2] [j] != '*' && t1->campo [i - 2] [j] != 'T') ||

                 (t1->campo [i] [j - 1] == 'T' && t1->campo [i] [j - 2] != '*' && t1->campo [i] [j - 2] != 'T')))

                v = 0; } } return v;}

```

```

int verificaResultado (tabuleiro *t1) {

    int v = 0;

    for (int i = 1; i < 9; i++) {

        for (int j = 1; j < 9; j++) {

            if (t1->campo [i] [j] == 'T') v++;

        }

    }

    if (v > 1) v = 0;

    return v;}

```

Caso a função verificaResultado (t1) retorne 0, significa que o jogador perdeu e atribui-se o resultado “perdeu”, então retorna-se à função jogo (). Caso verificaResultado (t1) retorne 1, atribui-se o resultado “ganhou” e retorna-se à função jogo ().

Se ainda existirem jogadas possíveis, é declarado dois inteiros (linha, coluna), um char direção e então, lidos do teclado (1<linha/coluna<9 e direção=W/A/S/D/V/R).

```
int seg; struct tm *hora_atual3; time_t seg3;

int linha = 0, coluna = 0;

do {linha = getch ();} while (linha > 57 || linha < 49);

linha -= 48;

do {coluna = getch ();} while (coluna > 57 || coluna < 49);

coluna -= 48;

printf (“\n\n(2)Para qual direção?\n(Digite V para sair ou R para voltar jogada)\n\nDireção: “);

char d;

do {d = getch ();} while (d != 'W' && d != 'A' && d != 'S' && d != 'D' && d != 'V' && d != 'R');
```

```
time (&seg3);

hora_atual3 = localtime (&seg3);

seg = hora_atual3->tm_sec;

seg += 60*(hora_atual3->tm_hour*60 + hora_atual3->tm_min);

temp->s = (seg - segu) % 60;

temp->m = (seg - segu) / 60;

strcpy (t1->resultado, "Perdeu");

gravaJogo (t1, temp);

free (t1);

free (temp);

Inicio ();
```

É executado um switch (d) para realizar a lógica da jogada, que dependendo da direção escolhida, o campo[linha][coluna] será alterado. Porém, caso o jogador pressione ‘V’, o programa finalizará o timer, fazendo que com subtraia-se o tempo quando o jogo foi iniciado do tempo quando o jogo foi encerrado. Também irá definir o resultado do jogo com “perdeu”, salvará as informações da partida na tabela de histórico, liberará os espaços alocados pelas matrizes e retornará à função Inicio ().

```
int verificaJogada (tabuleiro *t1, int linha, int coluna, char d) {

    int v = 1;

    switch (d) {

        case 'W':

            if (t1->campo[linha-2][coluna]=='T' || t1->campo[linha-2][coluna]=='*' ||

                t1->campo[linha-1][coluna]=='O' || t1->campo[linha-1][coluna]=='*' ||

                t1->campo[linha][coluna]=='O' || t1->campo[linha][coluna]=='*')

                v = 0; break;

        (...) return v;}

}
```

Antes da jogada ser realizada, é conferido se a jogada é válida, ou seja, se a posição a ser movimentada não é um \*, se onde quero chegar já não exista um pino ou se realmente estou movimentando um pino. Isso é feito para todas as direções:

```

case 'W':
    if (verificaJogada (t1, linha, coluna, d)) {
        check = 1; salvaJogada (t1);
        t1->campo[linha][coluna]='O';
        t1->campo[linha-1][coluna]='O';
        t1->campo[linha-2][coluna]='T';
    }
    break;

```

Caso a jogada seja possível, então o estado de jogada válida é acionado para 1, salvo a jogada anterior em um arquivo texto e por último executa-se a atualização do tabuleiro conforme a sua direção: Onde a peça estava vira espaço vazio, a posição adjacente na direção do movimento também vira vazio e a posição próxima a adjacente na direção ao movimento recebe o pino.

if (!check) {		else {
t1->contador++;		t1->contador++;
system ("cls");		t1->pinos--;
printf ("Jogada Inválida\n\n");		system ("cls");
imprimiInfo (t1);		printf ("\n");
printf ("\n");		imprimiInfo (t1);
executaJogada (t1, temp, segu);		executaJogada (t1, temp, segu);
}		}

Se a jogada for impossível, é mostrado na tela “Jogada Inválida”, o contador de jogadas é acrescentado em 1 e então novamente imprimido o tabuleiro e chamada a função executaJogada (t1, temp, segu). Caso seja válida, incrementa o contador e decrementa os pinos, e novamente chama a função executaJogada.

Caso o jogo tenha chegado ao fim, é retornado à função jogo (t1, n), recupera-se o tempo do sistema para então fazer a diferença desse tempo final pelo inicial e saber a duração total do jogo, depois é salvo em um arquivo texto tabuleiro que foi jogado, o nome do usuário, resultado da partida, número de jogadas e o tempo de duração do jogo conforme formatação definida pelo desenvolvedor. Em seguida, é chamado a função verificaRecord (tabuleiro \*t1, tempo \*temp, int n), na qual abre um ponteiro para arquivo FILE, lê a linha do arquivo de records (que contém 1 recorde para cada tipo de tabuleiro) dependendo de n como parâmetro, compara se o resultado foi de vitória, se o número de jogadas foi o mesmo ou o menor presente no recorde, e por último, compara se o tempo foi menor.

```

char arq [13] = "ranking"; | while (ch != EOF) {
arq [7] = (n / 10) + 49; | if (aux == ((n % 10) * 2 + 2) && ch == '$')
strcat (arq, ".txt"); | fscanf (file, "%s %d %d:%d", re, &nj, &nj2, &nj3);
int aux = 0, nj, nj2, nj3; | if (ch == '\n')
char ch, re [7]; | aux++;
FILE *file = fopen (arq, "r"); | ch = getc (file);}
if ((strcmp (re, t1->resultado) == 0) && nj >= t1->contador && (60*nj2 + nj3) >= (60*temp->m + temp->s))

```

```

char c = '@';
int l = 0;
FILE *file = fopen (arq, "r");
FILE *file2 = fopen ("rankingaux.txt", "w");
FILE *file3 = fopen ("rankingaux2.txt", "w");
while (c != EOF) {
    if (c == '\n') l++;
    if (l <= n + 1 && l != n && c != '@') putc (c, file2);
    if (l > n + 1 && c != '@') putc (c, file3);
    c = getc (file);
}
fclose (file); fclose (file2); fclose (file3);

```

Caso os valores tenham sido menores do que os gravados no record, então é chamada a função `separaArquivos (t1, temp, n, arq)` para apagar a linha do arquivo de recordes correspondente a `n` como parâmetro e então armazenar o que havia antes da linha específica no arquivo `aux1` e o que havia depois no arquivo `aux2`. Para identificar a linha, basta contar as quebras de linhas. Se o número de quebras for igual linha desejada, deve-se desconsiderar ela. Após a varredura do arquivo por completo, fecha-se momentaneamente os 3 arquivos.

```

FILE *file2 = fopen ("rankingaux2.txt", "r");
if (file2 == NULL) {printf ("ERRO!"); exit (1);}
FILE *file = fopen ("rankingaux.txt", "a");
int tam = 1000;
char leitura [tam];
while ((fgets (leitura, tam, file2)) != NULL)
    fputs (leitura, file);
fclose (file2);
fclose (file);

```

Então, chama-se `gravaArquivo (t1, temp)`, para colocar as informações de recordes atualizados no arquivo `aux1`, em seguida, chama-se `concatenaArquivos ()` para concatenar os arquivos `aux2` em `aux1`. O arquivo que contém as informações do topo do record é aberto em formato de apêndice, para por adicionar o conteúdo do segundo arquivo auxiliar.

E, por último, chama-se `arrumaRanking ()` colocar o que havia em `aux1` no arquivo de recordes padrão, e limpar os dois arquivos auxiliares.

<code>FILE *file = fopen ("rankingaux.txt", "r");</code>	<code>  char *leitor = (char *) calloc (n, sizeof (char));</code>
<code>if (file==NULL) {</code>	<code>  while (fgets (leitor, n, file) != NULL)</code>
<code>    printf ("ERRO!"); exit (1);}</code>	<code>     fputs (leitor, file2);</code>
<code>FILE *file2 = fopen (arq, "w");</code>	<code>  free (leitor); fclose (file); fclose (file2);</code>
<code>char c;</code>	<code>  FILE *limpa = fopen ("rankingaux.txt", "w"); fclose (limpa);</code>
<code>int n = 0;</code>	<code>  FILE *limpa = fopen ("rankingaux2.txt", "w"); fclose (limpa);</code>
<code>while (c != EOF) {n++; c = getc (file);}</code>	<code> </code>
<code>rewind (file);</code>	<code> </code>

Após essa sucessão de funções, o programa volta a função jogo, assim como se nenhum recorde foi batido no jogo. Então, é liberado o espaço alocado por t1 e temp. É declarado um inteiro e lido do teclado ( $1 < opt < 3$ ) e chamado uma função de menuT ().

```
int opT;
printf("<Menu de Escolha>\n\n(1) Voltar ao Menu\n(2) Abrir Histórico\n");
printf("(3) Jogar Novamente\n(4) Sair\n");
do {opT = getch ();} while (opT < 49 || opT > 52);
system ("cls");
switch (opT) {
    case 49: Inicio (); break;
    case 50: leArquivo ("historico.txt"); menuT (); break;
    case 51: escolheTab (); break;
    case 52: exit (0); break;
}
```

Nela, é escolhido se o jogador deseja ver o histórico salvo, se deseja jogar novamente, se deseja retornar a tela principal ou se sair do programa diretamente. Caso a opção seja jogar novamente, o programa volta ao menuS() e continua a ser executado, até que em algum momento o jogador deseje sair do jogo, procurando alguma função de escape exit (0) espalhado pelo programa.

Além dessas funções, foram utilizadas funções auxiliares para leitura de arquivos:

```
FILE *abreArquivo (char nome [], char t []) {
    FILE *file = fopen (nome, t);
    if (! strcmp (t, "r")) {
        if (file == NULL) {printf ("\n\nERRO AO ABRIR O ARQUIVO!\n\n"); exit (1);}
    }
    return file;
}
```

Basicamente, isso cobre a maior parte das funções implementas no simulador, pois a partir disso, é tudo repetição e recursividade. Basta abusar das funções livremente.

## Considerações Pessoais

Desde o início não foi um trabalho fácil, muitas das vezes se tornando complexo, e de difícil entendimento. Vários erros foram acumulados e para administração e exclusão deles foi necessário muito tempo investido. Aos poucos, erros de leitura de caractere, erros de retorno em funções do tipo void, ou até erros de passagem por parâmetros. O domínio de funções foi elevado com sucesso. Funções com e sem retorno, funções com e sem parâmetros, recursividade. Tudo nesse quesito foi muito melhorado no desenvolvimento do simulador. Todos os erros foram completamente removidos, resultando em programa que compila sem nenhum warning da IDE.

Aprender a concatenar arrays de caractere, vulgo strings, comparar e usufruir de várias funções da biblioteca string.h, foi um dos resultados ao desenvolver esse simulador. Em linguagem C, a utilização de strings é um pouco limitada, tornando o trabalho com elas um pouco árduo e dificultoso. Mas com a boa prática a abstração e lógica de strings foram muito bem aperfeiçoadas.

Outra melhoria no aprendizado foi sobre arquivos texto. Abertura para leitura, confirmação de existência do arquivo, recuperação e gravação de dados, retorno de ponteiros para arquivos. Foram vários os conhecimentos adquiridos e bem trabalhos sobre o assunto. Aprender a abrir e fechar o arquivo no momento correto, manuseá-lo da maneira correta, foram habilidades adquiridas na hora de comparação dos records. Separar um arquivo em dois, gravar novas informações, concatenar e então substituir no principal, não foi fácil.

Além do projeto ambientar e familiarizar mais com a linguagem de programação, a melhoria em ter que correr atrás por conta própria das resoluções dos erros, implementação de funções por maneira correta, ou até o uso de funções que até então eram desconhecidas. Tudo isso ajudou para o crescimento profissional, no sentido de correr atrás das respostas sozinho. Afinal, um engenheiro precisa detectar um problema e apontar uma solução para ele, tendo ajuda ou não.

## Referências

AKSTEIN e HEINE, Liliana e Evelyn. Divertido – Resta Um. Disponível em:  
<https://www.divertudo.com.br/restaum/restaum.html> Acesso em: 10 de Novembro de 2019.

BACKES, André. Linguagem C Descomplicada. Disponível em:  
[https://www.youtube.com/watch?v=GiCt0Cwcp-U&ab\\_channel=LinguagemCPrograma%C3%A7%C3%A3oDescomplicada](https://www.youtube.com/watch?v=GiCt0Cwcp-U&ab_channel=LinguagemCPrograma%C3%A7%C3%A3oDescomplicada) Acesso em: 5 de Julho de 2019.

BACKES, André. Linguagem C – Completa e Descomplicada. 2.ed, GEN LTC, 2018.

SCHILDT, Herbert. C Completo e Total. 3.ed, Makron Books, 1996.