Universidade Federal de São Carlos
Engenharia de Computação

Laboratório de Arquitetura e Organização de Computadores 2

Terceiro Relatório

Alunos: Bruna Zamith Santos (RA: 628093)
Marcos Augusto Faglioni Junior (RA: 628301)

Professor: Dr. Luciano Neris

07/2017
São Carlos - SP, Brasil

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

O presente relatório objetiva a apresentação do jogo “Clock Colors”, desenvolvido na disciplina Laboratório de Arquitetura e Organização de Computadores 2, ministrada pelo docente Dr. Luciano Neris, no primeiro semestre de 2017, na Universidade Federal de São Carlos.

O jogo foi desenvolvido em linguagem Assembly e utilizando-se a biblioteca Irvine32 [1]. Como objetivo secundário, destaca-se o aprofundamento do conhecimento na linguagem anteriormente citada e na referida biblioteca, assim como nos recursos que esta oferece. Não obstante, foi utilizado o montador *Microsoft Macro Assembler* (MASM) [2], o qual suporta as arquiteturas IA-32 e x86-64 para MS-DOS e Microsoft Windows. O ambiente de desenvolvimento - *Integrated Development Environment* (IDE) - utilizado foi o Visual Studio.

1.1 Histórico dos Jogos de Computadores

A história dos jogos de computadores podem ser considerada também, de certa forma, como uma história da evolução da tecnologia. Isto porque os jogos de computadores exigem tecnologias capazes de armazenar grande quantidade de dados e a capacidade de representá-los.

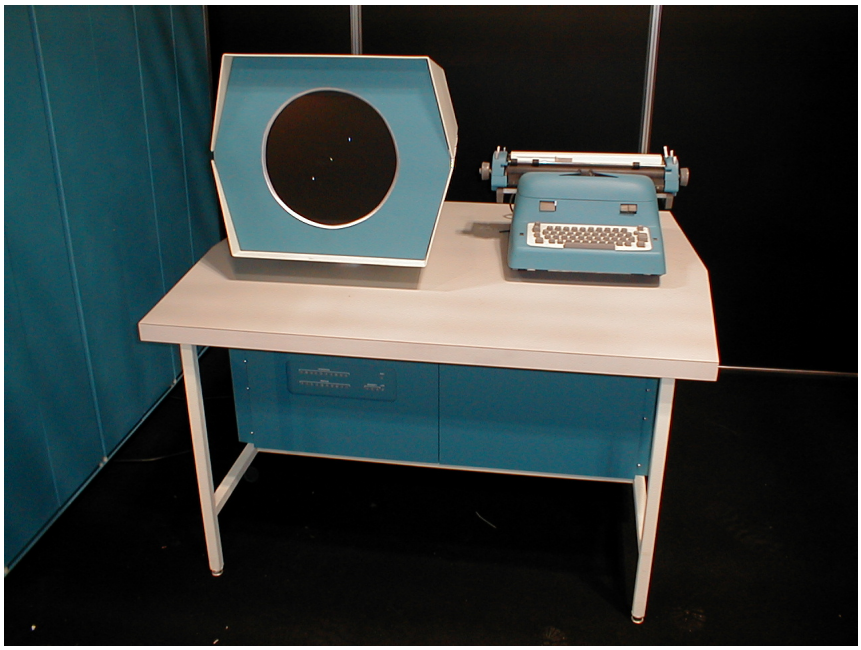
Um dos primeiros jogos de computadores já desenvolvidos é o jogo “Spacewar!”, o qual foi lançado em 1962, no MIT (*Massachusetts Institute of Technology*), por Stephen Russell [3] (Figura 1). “Spacewar”originalmente rodava em computadores PDP-1 (Figura 2), muito maiores que os computadores pessoais atuais. O jogo permitia que duas pessoas jogassem uma contra a outra, pilotando, cada uma, um foguete. O objetivo principal do jogo era derrubar o foguete do jogador adversário.

Figura 1: Spacewar!, 1962

Fonte: [3]

Entre 1971 e 1974, surgiram os primeiros jogos de computadores comerciais. Em 1971, Nolan Bushnell desenvolveu o “Computer Space”(Figura 3), o primeiro jogo de

Figura 2: PDP-1



Fonte: [4]

arcade comercial. Ele era baseado no “Spacewar!”, com gráficos vetoriais, mas era muito sofisticado para o mercado e por isso não obteve sucesso [5].

Figura 3: Computer Space, 1971

Fonte: [5]

Em 1973, foi lançado o jogo “Pong”(Atari), 11 anos após o lançamento de “Spacewar!”. Pong obteve incrível sucesso em meio ao público, mesmo tendo gráficos extremamente simples (retângulos brancos e fundo preto). Inicialmente, “Pong”foi posto em feiras de jogos e lojas de entretenimento, mesmos tipos de locais onde o jogo “Space Invaders”(Figura 4) viria a ser colocado a partir de 1977. O sucesso de “Space Invaders”pode ser atribuido ao fato de ser um jogo de tempo-real que requer reflexos rápidos, possui um

número finito de vidas, e níveis de dificuldade crescentes.

Figura 4: Space Invaders, 1977

Fonte: [6]

Desde então, milhares de novos jogos e emuladores surgiram e atraíram público de todas as idades. De fato, existem diversos tipos de jogos de computadores. No clássico jogo de ação, nunca se ganha, o jogo apenas vai ficando mais difícil e quem "sobreviver"por mais tempo, sai vitorioso. A coisa mais geral a dizer sobre a evolução dos jogos de computadores é que provavelmente eles se tornoram gradualmente mais baseados em gêneros. Quase todos os primeiros jogos de computadores introduziram novos elementos de jogabilidade; Os jogos posteriores tendem a ser exemplos de gêneros específicos, trazendo os traços dos jogos anteriores.

O próprio termo "jogo de computador"concorre fortemente com jogos de videogame, jogos de console e jogos de arcade. Videogames e jogos de console normalmente significam jogos conectados a uma TV, enquanto jogos arcade significam jogos colocados em espaços públicos (e gabinetes individuais). Os jogos de computador são, ocasionalmente, considerados jogos jogados em um computador pessoal. Uma vez que todas essas áreas foram desenvolvidas em paralelo (e porque todos esses jogos são jogados em computadores), é possível usar o mesmo termo ("jogo de computador") para denominar todas essas áreas como um todo [3].

1.2 Etapas de Desenvolvimento

As etapas de desenvolvimento do jogo estão enumeradas a seguir:

1. Construção das Bordas da Tela
2. Construção das Plataformas
3. União das Plataformas e Bordas com o texto do Menu Inicial
4. Implementação do Menu Inicial
5. Criação da Página de Como Jogar
6. Criação da Página de Créditos
7. Contador de Tempo Regressivo
8. Criação do Personagem Controlável pelo Jogador
9. Geração dos Obstáculos na Plataforma, randomicamente
10. Detecção de Colisão
11. Sorteio das Cores Iniciais
12. Variação das Cores da Plataforma
13. Atualização da Altura das Plataformas
14. Criação da Tela de Resultado do Jogo
15. Implementação das Diferentes Dificuldades do Jogo
16. Testes
17. Correção de Eventuais Erros

1.3 Divisão de Tarefas

Para a implementação do projeto, ficou decidido que todos teriam igual participação no desenvolvimento. Assim, ambos os desenvolvedores puderam ter contato com as estruturas de dados do jogo e as principais rotinas. Não obstante, evitou-se a segmentação do código.

Primeiramente, a dupla se reuniu para decidir a lógica do jogo, as variáveis e as rotinas. Então, para o desenvolvimento, se reuniram presencialmente e discutiram juntos.

1.4 Controle de Versões

Foi feito o controle de versões do código, visando-se comparar mudanças feitas ao decorrer do tempo, ver quem foi o último a modificar o arquivo, e facilitar a detecção de problemas. O controle de versões foi possível através da ferramenta Github, sob o link [1]. O projeto foi licenciado com GLP-3.0 (*General Public Licence, version 3.0*).

[1] https://github.com/MarcosFagli/Clock-Collors

2 Descrição

O jogo, entitulado “Clock Colors”, é *single-player*, possui um personagem que se movimenta na tela (horizontalmente e verticalmente), um contador de tempo regressivo, plataformas que variam de cor e armadilhas nessas plataformas. O jogador controla o personagem, chamado de “Etevaldo”, sendo que o detalhamento dos comandos encontra-se na Seção 2.3.

O objetivo principal do jogo é atingir o maior número de plataformas sem morrer ou antes que o tempo acabe. As condições de *Game Over* serão detalhadas na Seção 2.6. O contador de tempo começa em N segundos (sendo que N depende da dificuldade do jogo, explicitada na Seção 2.5) e é decrementado de 1 em 1 segundo. Assim, o jogo caracteriza-se como sendo um jogo de ação e velocidade.

2.1 Características Gerais

Como características gerais do jogo, destacam-se:

- São 4 plataformas na tela do jogo, de posições fixas;
- Quando o personagem desloca-se verticalmente, ao invés de subir nas plataformas, essas que descem, de modo que o jogador sempre permanece no chão;
- A primeira plataforma (imediatamente acima do personagem) pode variar entre 8 cores disponíveis: amarelo, azul, verde, ciano, vermelho, magenta, branco e vermelho claro;
- As demais plataformas possuem cor fixa verde;
- A variação das cores é feita de maneira randômica, e a cada segundo.
- Serão sorteadas duas cores, dentre as 8, consideradas cores “permitidas”, isto é, validam o salto do personagem - estas cores serão exibidas na tela do jogo;
- Cada plataforma, exceto a mais baixa (que não possui armadilhas), contém um número máximo de M armadilhas, com M variando de acordo com a dificuldade do jogo, em posições sorteadas randomicamente;
- Como exposto anteriormente, o tempo inicial varia de acordo com a dificuldade do jogo, e este é decrementado de 1 em 1 segundo;
- O “Etevaldo”só tem 1 vida.

2.2 Telas do Jogo

O jogo conta com 8 telas:

- 1 Tela do Menu Inicial (Figura 5);
- 1 Tela de Dificuldades (Figura 6);
- 2 Telas de Instruções/Como Jogar (Figura 7);
- 1 Tela de Créditos (Figura 8);
- 1 Tela de Jogo (Figura 9);
- 2 Telas Diferentes de Resultado (Figura 10).

Figura 5: Tela do Menu Inicial do Jogo

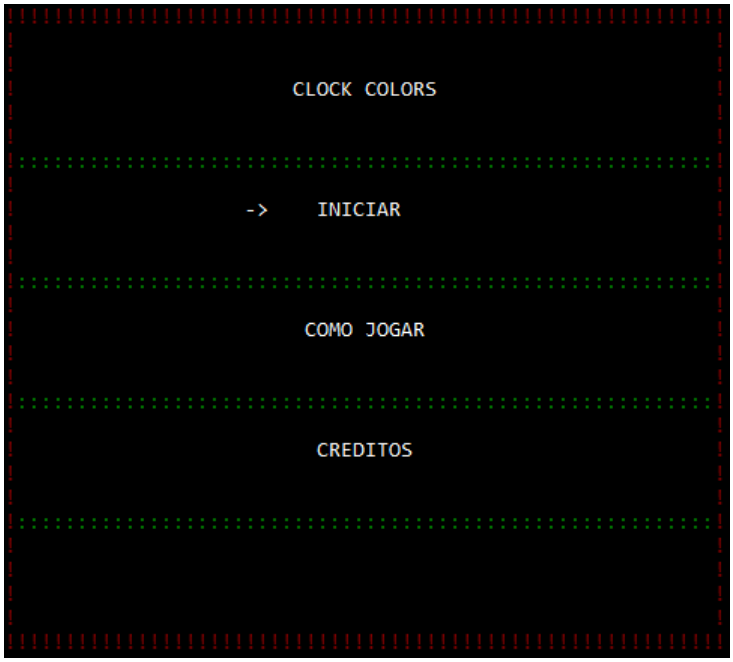


Figura 6: Tela de Seleção da Dificuldade do Jogo

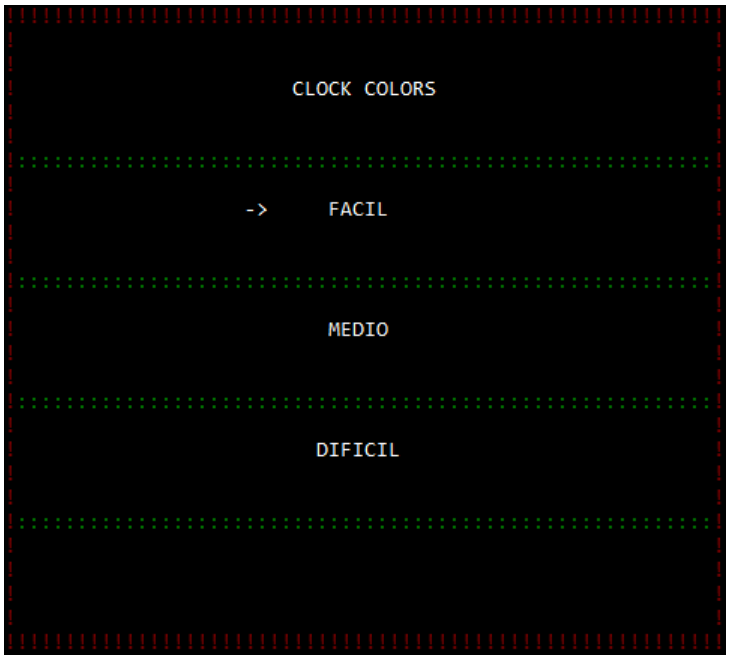


Figura 7: Telas de Instruções/Como Jogar

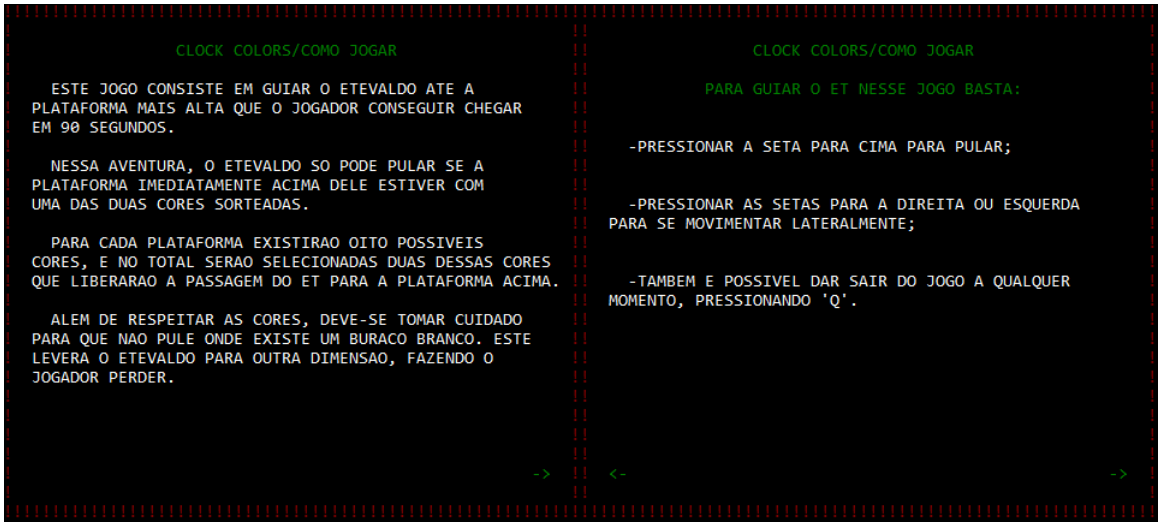


Figura 8: Tela de Créditos do Jogo

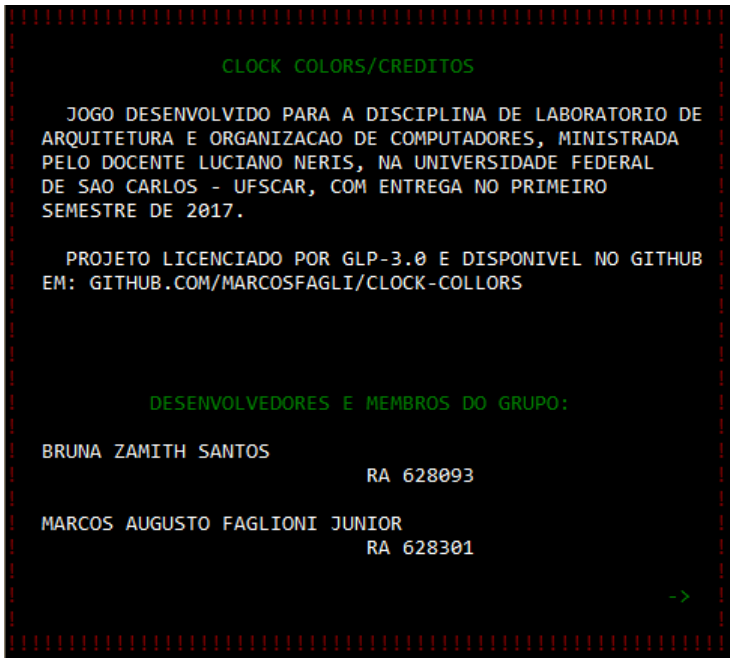


Figura 9: Tela do Jogo

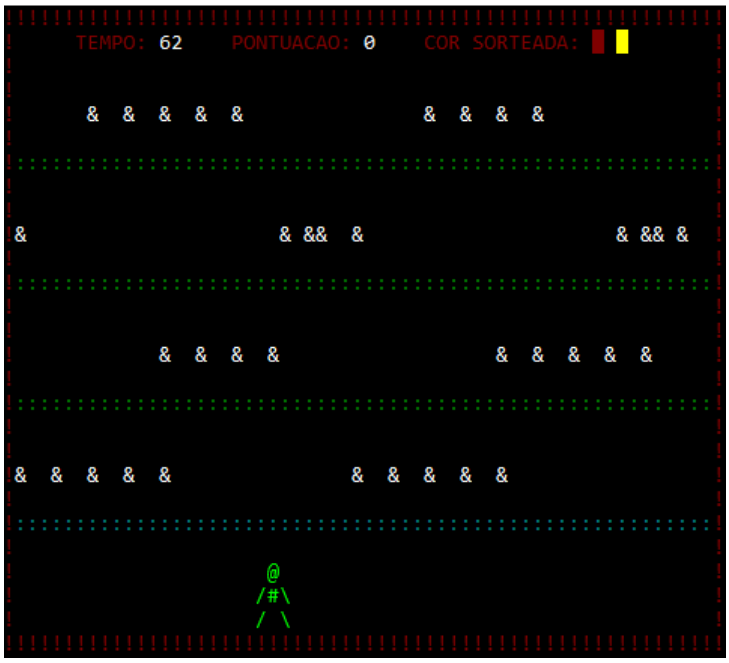
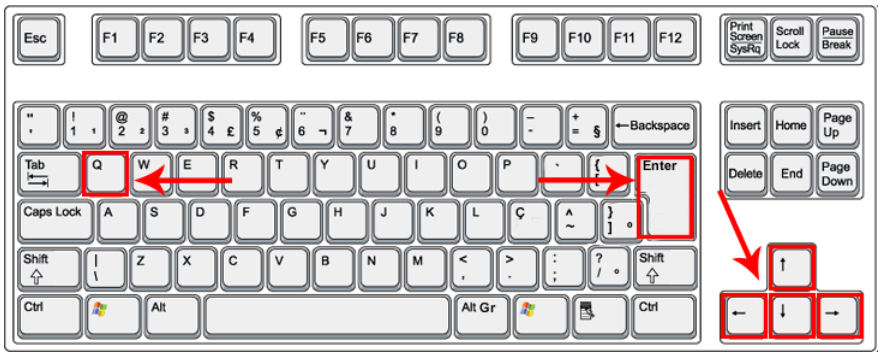


Figura 10: Telas do Resultado

2.3 Comandos do Usuário

São 6 teclas utilizadas no jogo: Seta direita, seta esquerda, seta para cima, seta para baixo, “enter”e “q”/“Q”. A Figura 11 exibe as referidas teclas.

Figura 11: Teclas do Jogo



Os comandos acionados por tais teclas, quando no Menu Inicial ou na Tela de Dificuldades, são:

- “Seta Para Cima”: Muda para a opção acima da atual;
- “Seta Para Baixo”: Muda para a opção abaixo da atual;
- “Enter”: Seleciona a opção atual;
- “q”ou “Q”: Encerra o jogo;

Quando na Tela dos Créditos ou na Tela de Instruções (“Como Jogar”):

- “Seta Para Direita”: Muda para a tela seguinte;

- “Seta Para Esquerda”: Muda para a tela anterior (apenas disponível na Tela de Instruções).

Quando na Tela do Jogo:

- “Seta Direita”: Movimenta o “Etevaldo”para a direita;
- “Seta Esquerda”: Movimenta o “Etevaldo”para a esquerda;
- “Seta Para Cima”: Movimenta o “Etevaldo”verticalmente para a plataforma de cima, fazendo que todas as plataformas desçam;
- “q”ou “Q”: Volta para o Menu Inicial.

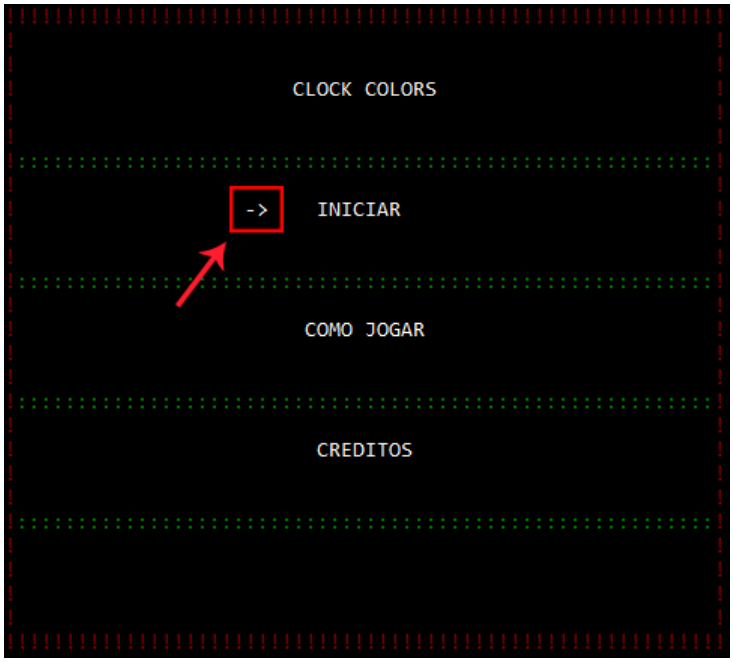
Quando na Tela do Resultado:

- “Seta Para Cima”: Muda para a opção acima da atual;
- “Seta Para Baixo”: Muda para a opção abaixo da atual;
- “Enter”: Seleciona a opção atual.

2.4 Formas de Representação Gráfica

A seleção nas telas, de acordo com especificado na Seção 2.3, é representada por uma seta, como mostrado na Figura 12.

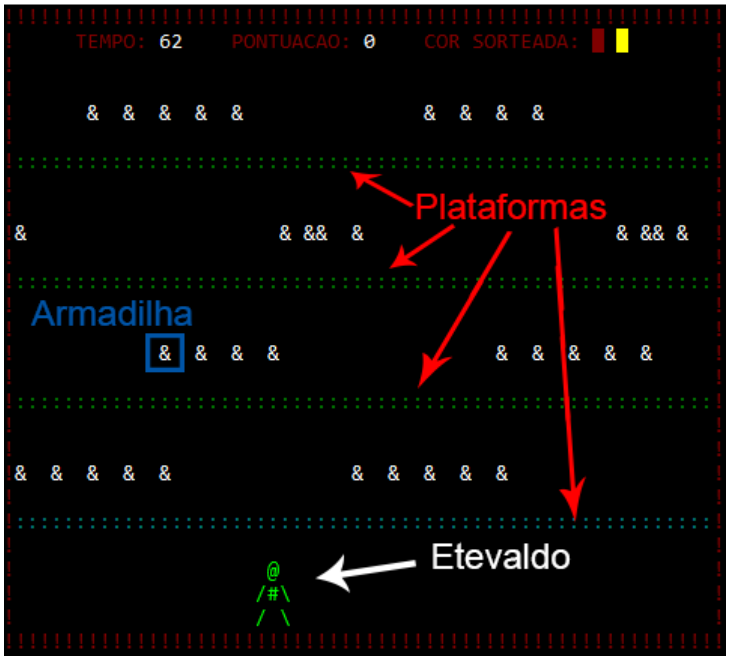
Figura 12: Seta para Seleção das Telas do Jogo



As plataformas possuem posições fixas e são representadas por uma sequência do símbolo dois pontos “:”. As armadilhas, por sua vez, são representadas pelo símbolo “& ”.

O personagem “Etevaldo”tem um desenho próprio. A Figura 13 mostra as representações gráficas da tela do jogo.

Figura 13: Representações Gráficas da Tela do Jogo



2.5 Dificuldades do Jogo

O jogo possui 3 dificuldades diferentes, a qual é escolhida pelo jogador na Tela de Dificuldades. As dificuldades diferem no valor N de tempo máximo e na quantidade máxima M de armadilhas por plataforma:

- Fácil: N = 90 segundos e M = 3 obstáculos;
- Médio: N = 60 segundos e M = 7 obstáculos;
- Difícil: N = 30 segundos e M = 15 obstáculos.

2.6 *Game Over*

O jogo encerra-se caso o tempo acabe ou o jogador perca. O jogador perde se:

- Pular para a plataforma em um momento em que ela não esteja com uma das duas cores sorteadas;
- Ou colidir com alguma das armadilhas da plataforma logo acima de si.

Em ambas as situações, a Tela do Resultado é exibida.

3 Elementos Específicos

3.1 Estrutura de Dados

As estruturas de dados utilizadas no desenvolvimento do jogo são: Vetores, Matrizes e Pilha.

Vetores: Para o projeto, são os *arrays* de caracteres, ou seja, strings. São elementos essenciais ao Menu Inicial, a Tela de Resultado e a qualquer informação que ser a escrita na Tela do Jogo (como o contador regressivo de tempo e as cores sorteadas). Não obstante, também é utilizado como o *array* que armazena as 8 cores disponíveis para sorteio; como o *array* que efetivamente armazenas as 2 cores sorteadas; e como o *array* que armazena as posições das armadilhas.

Matrizes: A tela por si só é uma matriz. Todas as informações exibidas possuem coordenadas X e Y na matriz da tela do jogo.

Pilha: As chamadas de procedimentos são feitas pelas instruções “Call”e “Ret”, que por sua vez, fazem uso de uma pilha para armazenar os endereços de retorno.

3.2 Variáveis Principais

Os principais dados armazenados são:

Tabela 1: Dados Armazenados

Label	Tamanho	Conteúdo	Dado
cor	BYTE	”COR SORTEADA:”,0	Nome Cor
tempo	BYTE	”TEMPO:”, 0	Nome Tempo
pontuacao	BYTE	”PONTUACAO:”, 0	Nome Pontuação
biniciar	BYTE	”INICIAR”,0	Nome Iniciar
bcreditos	BYTE	”CREDITOS”,0	Nome Créditos
bcomoJogar	BYTE	”COMO JOGAR”,0	Nome Como Jogar
nome	BYTE	”CLOCK COLORS”, 0	Nome Jogo
bDificuldade1	BYTE	”FACIL”, 0	Nome Dificuldade 1
bDificuldade2	BYTE	”MEDIO”, 0	Nome Dificuldade 2
bDificuldade3	BYTE	”DIFICIL”, 0	Nome Dificuldade 3
timeMax	BYTE	90	Tempo Máximo do Jogo
time	BYTE	90	Tempo do Jogo
score	BYTE	0	Pontuação
posSeta	BYTE	0	Posição da Seta
tMaxX	BYTE	60	Qntd. Colunas Ecrã
tMaxY	BYTE	26	Qntd. Linhas Ecrã
posXB	BYTE	30	Posição X Personagem
posYB	BYTE	24	Posição Y Personagem
distPlat	BYTE	5	Dist. Entre Plataformas
nArmadilhas	BYTE	10	Num. Armadilhas Plat.
armadilhas	BYTE	40 DUP(?)	Pos. X das Armadilhas
platInicial	WORD	8	Pos. Y da Plat. Mais Alta
cont	BYTE	0	Contador Auxiliar
contTime	BYTE	0	Contador Auxiliar Tempo
coresDisp	WORD	yellow, blue, green,,cyan, red, magenta, white, lightRed	Cores Disponíveis
corSele	WORD	2 DUP(?)	Cores Sorteadas
corPlatAtual	WORD	1 DUP(?)	Cor da Plat. Mais Baixa

3.3 Procedimentos

O jogo faz uso de procedimentos em sua implementação. Todos os procedimentos são declarados entre “PROC”e “ENDP”e fazem uso de registradores para passagem de argumentos. Para a chamada do procedimento, é usado o “call”, que salva o endereço de retorno na pilha.

Além disso, para implementação, foram utilizados procedimentos prontos do Irvine [1]. Os principais procedimentos prontos utilizados são estão descritos na Tabela 2.

Tabela 2: Principais Procedimentos Utilizados

Procedimento	Realiza
SETTEXTCOLOR	Muda a Cor do Texto
GOTOXY	Define o Cursor na Posição X e Y
WRITECHAR	Imprime um Caractere na Tela
WRITESTRING	Imprime uma String na Tela
RandomRage	Gera um Número Aleatório em Determinado Intervalo
ReadKey	Leitura do Teclado

A seguir estão descritos os principais procedimentos desenvolvidos para o jogo, junto de seus respectivos pseudo-códigos. Os códigos em *Assembly* encontram-se no Apêndice A. Para a documentação do código, acessar seu repositório no Github [2].

3.3.1 Limpa Tela

Faz a limpeza da tela, ao imprimir espaço (’ ’) de cor preta.

Pseudo-código

```
proc limpaTela(recebe tMaxX, recebe tMaxY)
  cor recebe "black";
  (X,Y) recebe (0,0);
  loop X: de 0 a tMaxX
    loop Y: de 0 a tMaxY
      printaNaTela((X,Y),cor,’ ’);
```

3.3.2 Imprime Personagem

Imprime o personagem “Etevaldo”

[2] https://github.com/MarcosFagli/Clock-Collors

Pseudo-código

```
;    @
;   /#\
;   / \
proc ImpPerso(recebe posXB, recebe posYB)
  cor recebe "lightGreen";
  printaNaTela((posXB,posYB),cor,'@');
  printaNaTela((posXB-1,posYB+1),cor,'/');
  printaNaTela((posXB,posYB+1),cor,'#');
  printaNaTela((posXB+1,posYB+1),cor,'\');
  printaNaTela((posXB-1,posYB+2),cor,'/');
  printaNaTela((posXB+1,posYB+2),cor,'\');
```

3.3.3 Deleta Personagem

Deleta o personagem "Etevaldo", limpando a posição onde se encontra.

Pseudo-código

```
proc delPerso(recebe posXB, recebe posYB)
  cor recebe "black";
  printaNaTela((posXB,posYB),cor,' ');
  printaNaTela((posXB-1,posYB+1),cor,' ');
  printaNaTela((posXB,posYB+1),cor,' ');
  printaNaTela((posXB+1,posYB+1),cor,' ');
  printaNaTela((posXB-1,posYB+2),cor,' ');
  printaNaTela((posXB+1,posYB+2),cor,' ');
```

3.3.4 Printa Seta

Imprime a seta na tela.

Pseudo-código

```
proc PrintSeta(recebe X, recebe Y, recebe posSeta)
  cor recebe "black";
  printaNaTela(posSeta,cor,' ');
  cor recebe "white";
  multiplica Y por distPlat;
  soma Y com platInicial;
```

```
  printaNaTela((X,Y),cor,'-');
  printaNaTela((X+1,Y),cor,'>');
```

3.3.5 Bordas

Desenha a borda do jogo com "!"

Pseudo-código

```
proc Bordas(recebe tMaxX, recebe tMaxY)
  cor recebe "red";
  loop X: de 0 a tMaxX
    printaNaTela((X,0),cor,'!');
    printaNaTela((X,tMaxY),cor,'!');
  loop Y: de 0 a tMaxY
    printaNaTela((0,Y),cor,'!');
    printaNaTela((tMaxX,Y),cor,'!');
```

3.3.6 Plataformas

Imprime as 4 plataformas do jogo.

Pseudo-código

```
proc Plataformas(recebe tMaxX, recebe distPlat, recebe
   platInicial)
  cor recebe "green";
  loop Y: de 1 a 4
     loop X: de 0 a tMaxX
       printaNaTela((X,(platInicial+(Y*dist))),cor,'!');
```

3.3.7 Tela Início

Imprime o texto da tela de início do jogo.

Pseudo-código

```
proc TelaInicio(recebe platInicial, recebe biniciar, recebe
   distPlat)
   cor recebe "white";
   X recebe platInicial;
```

```
   printaNaTela((3,24),cor,"Clock Colors");
   printaNaTela((X,26),cor,biniciar);
   X recebe (X + distPlat);
   printaNaTela((X,25),cor,bcomoJogar);
   X recebe (X + distPlat);
   printaNaTela((X,26),cor,bcreditos);
```

3.3.8 Sorteia Cores

Faz o sorteio das duas cores "possíveis"do jogo. É feita a comparação para que as duas cores não sejam iguais.

Pseudo-código

```
proc SorteiaCores(recebe corSele, recebe corDisp)
   intervalo recebe (1 a 8); #numero de cores no vetor corDisp
   corSele[0] recebe random(intervalo);
   corSele[1] recebe random(intervalo);
   enquanto corSele[1] igual a corSele[0]:
      corSele[1] recebe random(intervalo);
```

3.3.9 Seta Direita

Este procedimento apaga o "Etevaldo"na posição atual e o imprime na posição imediatamente à direita. Antes disso, compara se o personagem já não se encontra na borda direita do jogo. Caso se encontre, não é feita a movimentação.

Pseudo-código

```
proc ProcSetaDir(recebe posXB, recebe tMaxX)
   se posXB igual a (tMaxX-2):
      retorna;
   seno:
      chama delPerso;
      posXB recebe (posXB + 1);
      chama ImpPerso;
```

3.3.10 Seta Esquerda

Este procedimento apaga o "Etevaldo"na posição atual e o imprime na posição imediatamente à esquerda. Antes disso, compara se o personagem já não se encontra na borda

esquerda do jogo. Caso se encontre, não é feita a movimentação.

Pseudo-código

```
proc ProcSetaEsq(recebe posXB)
   se posXB igual a 2:
      retorna;
   seno:
      chama delPerso;
      posXB recebe (posXB - 1);
      chama ImpPerso;
```

3.3.11 Colisão

Verifica se, ao tentar mover o personagem para cima, este não colidiu com alguma das armadilhas da plataforma.

Pseudo-código

```
proc Colisao(recebe posXB, recebe nArmadilhas, recebe
   armadilhas)
   loop i: de 0 a nArmadilhas
      se posXB igual a armadilhas[i]:
         retorna TRUE;
   retorna FALSE;
```

3.3.12 Verifica Posição

Recebe um valor de posição de armadilha e o vetor com as posições das armadilhas. Caso o valor da posição já esteja contido no vetor de posições, não é inserido. Isto evita que armadilhas sejam sobrepostas em uma mesma plataforma.

Pseudo-código

```
proc VerificaPos(recebe posicao, recebe nArmadihas, recebe
   armadilhas)
   loop i: de 0 a nArmadilhas
      se posicao igual a armadilhas[i]:
         retorna TRUE;
   retorna FALSE;
```

3.3.13 Seta Para Cima

Verifica se o movimento vertical é válido, isto é, se não houve colisão e se a cor da plataforma atual é igual a uma das duas cores selecionadas. Senão, o jogador perde o jogo. Caso o movimento seja válido, é feito o sorteio das posições da plataforma superior, e todas as plataformas são deslocadas para baixo.

Pseudo-código

```
proc PrcSetaCima(recebe posXB, recebe corSele, recebe
   corPlatAtual, recebe nArmadilhas, recebe armadilhas)
   colidiu recebe (chama Colisao);
   se colidiu igual a TRUE:
      retorna TRUE;
   se corPlatAtual diferente de corSele[0] e diferente de
      corSele[1]:
      retorna TRUE;
   shift armadilhas em nArmadilhas;
   intervalo recebe (2 a tMaxX);
   loop i: de 0 a nArmadihas:
      posicao recebe random(intervalo);
      enquanto (chama VerificaPos) igual a TRUE:
         posicao recebe random(intervalo);
      armadilhas[i] recebe posicao;
   chama DesenhaArm;
```

3.3.14 Tela Jogo

Desenha a tela do jogo e verifica comando do usuário.

Pseudo-código

```
proc TelaJogo
   chama LimpaTela;
   chama Bordas;
   chama Plataformas;
   chama TempoTela;
   chama ScoreTela;
   chama CriaArmInicio;
   chama ImpPerso;
   chama SorteiaCores;
   enquanto 1:
```

```
chama CorSelPlat;
tempo recebe (tempo - 1);
chama ScoreTela;
chama TempoTela;
se tempo igual a 0:
   retorna 0;
aguarda entrada teclado;
chama proc correspondente PrcSetaCima, PrcSetaDir ou
   PrcSetaEsq;
se PrcSetaCima igual a TRUE:
   retorna 1;
```

3.3.15 Telas Dificuldade, Instruções, Créditos, Perdeu e Acaba Tempo

Imprime cada uma das Telas, com os textos específicos. Além disso, recebe entrada do teclado e permite a seleção da opção desejada pelo usuário (setas).

3.3.16 Tempo Tela

Contador regressivo do jogo. Apaga o tempo anterior na tela, e escreve o tempo atualizado.

Pseudo-código

```
proc TempoTela(recebe time, recebe cor)
   printaNaTela((1,13),cor,time);
```

3.3.17 Score Tela

Pontuação do jogo. Apaga a pontuação anterior na tela, e escreve a pontuação atualizada.

Pseudo-código

```
proc ScoreTela(recebe score, recebe cor)
   printaNaTela((1,30),cor,score);
```

3.3.18 Seleciona Cor Plataforma

Imprime a plataforma de acordo com a sua cor atual.

3.3.19 Apaga Armadilhas

Apaga as armadilhas, percorrendo o vetor e substiituindo a posição com ' '.

3.3.20 Desenha Armadilhas

Desenha as armadilhas, percorrendo o vetor e printando '&'.

3.3.21 Verifica Posição Inicial

Realiza a mesma função que o procedimento VerificaPos, porém, para as armadilhas sorteadas no início do jogo. A diferença é que este procedimento realiza o processo para as 4 plataformas, e não só a mais elevada.

3.3.22 Cria Armadilhas Início

Faz o sorteio das posições iniciais das armadilhas inicializadas e chama o procedimento VerificaPosIni.

3.3.23 Troca Cor Plataformas

Varia a cor da plataforma logo acima do personagem. A variação é feita de forma randômica, percorrendo o vetor de cores disponíveis.

Pseudo-código

```
proc TrocaCorPlat(recebe coresDisp, recebe corPlatAtual, recebe
   tMaxX)
   intervalo recebe (1 a 8); #numero de cores no vetor corDisp
   corPlatAtual recebe random(intervalo);
   loop X: de 0 a tMaxX
      printaNaTela((X,Y),corPlatAtual,':');
```

3.4 Procedimento Principal (*Main*)

Para evitar processamento desnecessário, foi optado que a borda do jogo, que está presente em todas as telas, fosse impressa uma única vez. Por isso, ela o procedimento que a imprime, é chamado na *Main*. Não obstante, o mesmo acontece com a impressão das plataformas, mas estas estão presentes apenas na Tela de Menu Inicial, na Tela de Dificuldades e na Tela do Jogo. Ressalta-se que o jogo no modo difícil, tem um delay de inicialização devido à elevada quantidade de armadilhas a serem sorteadas e impressas. O procedimento *Main* pode ser melhor representado pelo Diagrama de Estados, na Seção

3.5.

Pseudo-código

```
chama LimpaTela;
chama Bordas;
chama Plataformas;
chama TelaInicio;
score recebe 0;
posSeta recebe 0;
posSeta1 recebe 0;
aguarda igual teclado;
caso tecla seja enter:
   verifica posSeta e chama tela correspondente;
   se tela correspondente igual a de jogo:
      chama TelaDificuldade;
      se facil:
         nArmadilhas recebe 3;
         time recebe 90;
      se medio:
         nArmadilhas recebe 7;
         time recebe 60;
      se dificil:
         nArmadilhas recebe 15;
         time recebe 30;
caso tecla igual a 'q' ou igual a 'Q':
   encerra o jogo;
```

3.5 Diagrama de Estados

Figura 14: Diagrama de Estados do Jogo - Parte 1

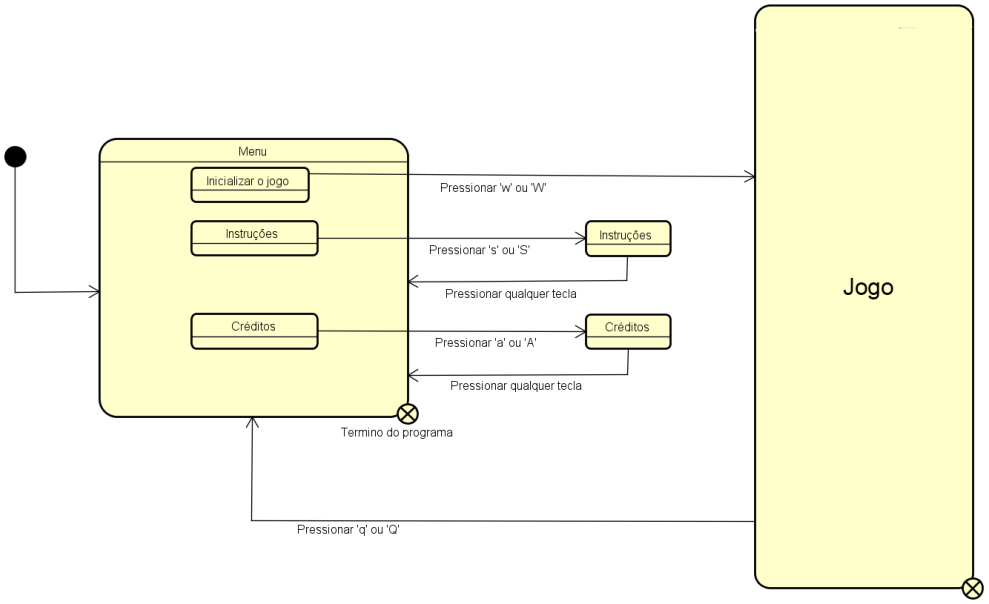
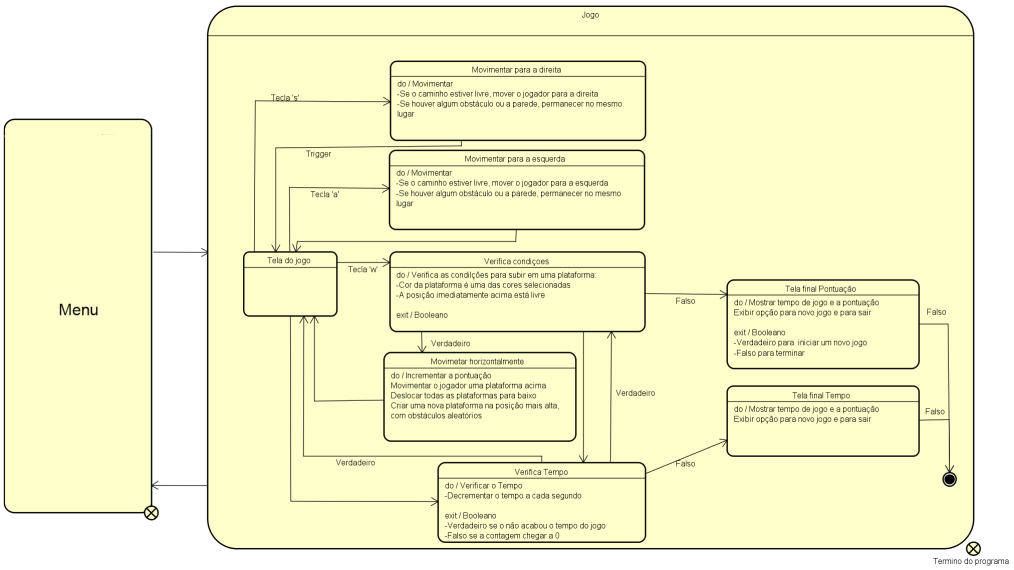


Figura 15: Diagrama de Estados do Jogo - Parte 2



4 Análise Crítica e Discussão

A maior dificuldade encontrada no desenvolvimento do projeto inicial foi o levantamento de requisitos iniciais do jogo, assim como a definição final do seu objetivo. Inicialmente, pretendia-se que o jogo não tivesse duração máxima, sendo a contagem de tempo progressiva. Todavia, chegou-se ao consenso de que o jogo ficaria mais dinâmico se houvesse o incentivo do tempo regressivo.

Definido o projeto, partiu-se para o seu desenvolvimento. Acredita-se que o fato de não ter havido divisão clara de tarefas entre os desenvolvedores, mas sim o aprendizado mútuo e simultâneo, muito contribuiu para que se adquirisse o conhecimento em *Assembly*, Irvine [1] e linguagem de baixo nível.

Melhorias poderiam ser feitas na própria escrita do código, com a definição de protótipos de procedimentos e fazendo-se uso da chamada "Invoke", bem como o uso de *stack frames*. Acredita-se que essas mudanças tornariam o código mais compreensível e curto. Contudo, quando os desenvolvedores tiveram primeiro contato com essas técnicas, já haviam avançado muito na implementação do jogo. Assim, mantiveram o padrão inicial.

Além disso, o jogo nas dificuldades "Medio"e "Difícil"pode apresentar certa lentidão, devido ao número de armadilhas que são impressas, a cada jogada, na tela. Em uma próxima versão do jogo, pensa-se em otimizar essa impressão, de modo a torná-lo mais rápido e fluido. No geral, os resultados alcançados mostraram-se satisfatórios, com bom funcionamento e jogabilidade do jogo.

A Códigos

Código Limpa Tela

```
LimpaTela PROC
   mov eax, black+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, 0
   call GOTOXY
   movzx ecx, tMaxY
LLP1:
   mov dl, 0
   mov dh, cl
   call GOTOXY
   push ecx
   movzx ecx, tMaxX
LLP2:
   mov al, ' '
   call WRITECHAR
   loop LLP2
   pop ecx
   loop LLP1
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, 0
   call GOTOXY
   ret
LimpaTela ENDP
```

Código Imprime Personagem

```
ImpPerso PROC
   mov eax, lightGreen+(black*16)
   call SETTEXTCOLOR
   mov dh, bh
   mov dl, bl
   sub dh, 1
   call GOTOXY
   mov al, '@'
   call WRITECHAR
```

```
   mov dl, bl
   dec dl
   mov dh, bh
   call GOTOXY
   mov al, '/'
   call WRITECHAR
   mov al, '#'
   call WRITECHAR
   mov al, '\'
   call WRITECHAR
   mov dl, bl
   dec dl
   mov dh, bh
   inc dh
   call GOTOXY
   mov al, '/'
   call WRITECHAR
   mov al, ' '
   call WRITECHAR
   mov al, '\'
   call WRITECHAR
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
ImpPerso ENDP
```

Código Deleta Personagem

```
delPerso PROC
   mov eax, black+(black*16)
   call SETTEXTCOLOR
   mov dl, posXB
   mov dh, posYB
   dec dh
   call GOTOXY
   mov al, ' '
   call WRITECHAR
   mov dl, posXB
```

```
   dec dl
   mov dh, posYB
   call GOTOXY
   mov ecx, 3
LDP1:
   mov al, ' '
   call WRITECHAR
   loop LDP1
   mov dh, posYB
   mov dl, posXB
   inc dh
   dec dl
   call GOTOXY
   mov ecx, 3
LDP2:
   mov al, ' '
   call WRITECHAR
   loop LDP2
  mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
delPerso ENDP
```

Código Printa Seta

```
PrintSeta PROC
   push dx
   mov ax, 0
   mov al, posSeta
   movzx dx, distPlat
   mul dx
   add ax, platInicial
   mov dl, 20
   mov dh, al
   call GOTOXY
   mov eax, black+(black*16)
   call SETTEXTCOLOR
   mov al, ' '
```

```
   call WRITECHAR
   call WRITECHAR
   pop dx
   cmp dx, 0026h
   jne LPS1
   cmp posSeta, 0000h
   jbe LPS2
   dec posSeta
LPS1:
   cmp dx, 0028h
   jne LPS2
   cmp posSeta, 0002h
   jae LPS2
   inc posSeta
LPS2:
   mov ax, 0
   mov al, posSeta
   movzx dx, distPlat
   mul dx
   add ax, platInicial
   mov dl, 20
   mov dh, al
   call GOTOXY
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov al, '-'
   call WRITECHAR
   mov al, '>'
   call WRITECHAR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
PrintSeta ENDP
```

Código Bordas

```
Bordas PROC
   mov eax, red+(black*16)
   call SETTEXTCOLOR
   movzx ecx, tMaxX
```

```
   mov al, '!'
L1:
   call WRITECHAR
   loop L1
   movzx ecx, tMaxY
   mov dh, 1
L2:
   mov dl, 0
   call GOTOXY
   call WRITECHAR
   mov dl, tMaxX
   dec dl
   call GOTOXY
   call WRITECHAR
   inc dh
   loop L2
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   movzx ecx, tMaxX
L3:
   call WRITECHAR
   loop L3
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   ret
Bordas ENDP
```

Código Plataformas

```
Plataformas PROC
   mov eax, green
   call SETTEXTCOLOR
   mov ecx, 4
   mov bx, platInicial
   sub bx, 2
LP1:
   mov dl, 1
   mov dh, bl
   call GOTOXY
   add bl, distPlat
```

```
   push ecx
   movzx eax, tMaxX
   sub eax, 2
   mov ecx, eax
LP2:
   mov al, ':'
   call WRITECHAR
   loop LP2
   pop ecx
   loop LP1
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   ret
Plataformas ENDP
```

Código Tela Início:

```
TelaInicio PROC
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 24
   mov dh, 3
   call GOTOXY
   mov edx, OFFSET nome
   call WRITESTRING
   movzx eax, platInicial
   mov dl, 26
   mov dh, al
   call GOTOXY
   mov edx, OFFSET biniciar
   call WRITESTRING
   add al, distPlat
   mov dl, 25
   mov dh, al
   call GOTOXY
   mov edx, OFFSET bcomoJogar
   call WRITESTRING
   add al, distPlat
   mov dl, 26
   mov dh, al
   call GOTOXY
```

```
   mov edx, OFFSET bcreditos
   call WRITESTRING
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
TelaInicio ENDP
```

Código Sorteia Cores

```
SorteiaCores PROC
   call Randomize
   mov eax,9
   call RandomRange
   mov corSele,ax
L1: mov eax,9
   call RandomRange
   cmp ax, corSele
   je L1
   mov corSele[TYPE corSele],ax
   mov bx, corSele
   imul bx, TYPE corSele
   mov ax, [coresDisp + bx]
   mov corSele, ax
   mov bx, corSele[TYPE corSele]
   imul bx, TYPE corSele
   mov ax, [coresDisp + bx]
   mov corSele[TYPE corSele], ax
   ret
SorteiaCores ENDP
```

Código Seta Direita

```
ProcSetaDir PROC
   movzx eax, tMaxX
   sub eax, 3
   cmp posXB, al
   jae fimProcDir
   call delPerso
```

```
   inc posXB
   mov bl, posXB
   mov al, tMaxY
   sub al, 2
   mov bh, al
   call ImpPerso
fimProcDir:
   ret
ProcSetaDir ENDP
```

Código Seta Esquerda

```
ProcSetaEsq PROC
   cmp posXB, 2
   jbe fimProcEsq
   call delPerso
   dec posXB
   mov bl, posXB
   mov al, tMaxY
   sub al, 2
   mov bh, al
   call ImpPerso
fimProcEsq:
   ret
ProcSetaEsq ENDP
```

Código Colisão

```
Colisao PROC
   movzx ecx, nArmadilhas
   mov edi, 0
L1:
   mov dl, armadilhas[edi]
   cmp dl, posXB
   je colidiu
   inc edi
   loop L1
   mov eax, 1
   jmp quit
colidiu:
   mov eax, 0
```

```
quit:
   ret
Colisao ENDP
```

Código Verifica Posição

```
VerificaPos PROC
   push eax
   push edx
   push ecx
   movzx ecx, nArmadilhas
   mov edi, OFFSET armadilhas
   movzx ebx, quantArmadilhas
   sub ebx, ecx
   add edi, ebx
   mov ebx, 1
   cld
   repne scasb
   jnz fim
   mov ebx, 0
fim:
   pop ecx
   pop edx
   pop eax
   ret
VerificaPos ENDP
```

Código Seta Para Cima

```
PrcSetaCima PROC
   call Colisao
   cmp eax, 0
   je diferente
   mov ax, corPlatAtual
   cmp ax, corSele
   je igual
   mov ax, corPlatAtual
   cmp ax, (corSele+2)
   jne diferente
   movzx ecx, nArmadilhas
   mov ebx, OFFSET armadilhas
```

```
igual:
   inc score
   call ApagaArm
   mov ebx, 0
   movzx eax, nArmadilhas
   mov edx, 3
   mul edx
   inc eax
   movzx edx, nArmadilhas
   mov ecx, eax
shiftByte:
   mov al, armadilhas[edx]
   mov armadilhas[ebx], al
   inc edx
   inc ebx
   loop shiftByte
   movzx ecx, nArmadilhas
   movzx edx, quantArmadilhas
   sub edx, ecx
L1:
   call Randomize
   movzx eax, tMaxX
   sub eax, 4
   call RandomRange
   inc al
   call VerificaPos
   cmp ebx, 1
   jne L1
   mov armadilhas[edx], al
   inc edx
   loop L1
   mov al, 07h
   call WriteChar
   call DesenhaArm
   mov eax, 1
   jmp fim
diferente:
   mov eax, 0
fim:
   ret
PrcSetaCima ENDP
```

Código Tela Jogo

```
TelaJogo PROC
   call LimpaTela
   call Bordas
   call Plataformas
   call TempoTela
   call ScoreTela
   call CriaArmInicio
   mov eax, red
   call SETTEXTCOLOR
   mov dl, 6
   mov dh, 1
   call GOTOXY
   mov edx, OFFSET tempo
   call WRITESTRING
   mov dl, 19
   mov dh, 1
   call GOTOXY
   mov edx, OFFSET pontuacao
   call WRITESTRING
   mov dl, 35
   mov dh, 1
   call GOTOXY
   mov edx, OFFSET cor
   call WRITESTRING
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov bl, posXB
   mov bh, posYB
   call ImpPerso
   call SorteiaCores
   call CorSelPlat
LTJ1:
   mov cont, 0
   inc contTime
   cmp contTime, 2
   jnae LTJ3
   call ScoreTela
   call TempoTela
   mov contTime, 0
```

```
LTJ3:
   call TrocaCorPlat
LTJ2:
   mov eax, 50
   inc cont
   cmp cont, 10
   ja LTJ1
   cmp time, 0
   jbe fimTempo
   call Delay
   call ReadKey
   jz LTJ2
   cmp dx, 0026h
   je setaCima
   cmp dx, 0025h
   je setaEsq
   cmp dx, 0027h
   je setaDir
   cmp dx, 0051h
   je fimTelaJogo
   jmp LTJ2
setaCima:
   call PrcSetaCima
   cmp eax, 0
   je fimPerdeuObs
   call CorSelPlat
   jmp LTJ2
setaEsq:
   call ProcSetaEsq
   jmp LTJ2
setaDir:
   call ProcSetaDir
   jmp LTJ2
fimPerdeuObs:
   mov eax, 0
   ret
fimTelaJogo:
   mov eax, 1
   ret
fimTempo:
   mov eax, 2 ret
```

```
TelaJogo ENDP
```

Código Tela Dificuldade

```
TelaDificuldade PROC
   call LimpaTela
   call Bordas
   call Plataformas
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 24
   mov dh, 3
   call GOTOXY
   mov edx, OFFSET nome
   call WRITESTRING
   movzx eax, platInicial
   mov dl, 27
   mov dh, al
   call GOTOXY
   mov edx, OFFSET bDificuldade1
   call WRITESTRING
   add al, distPlat
   mov dl, 27
   mov dh, al
   call GOTOXY
   mov edx, OFFSET bDificuldade2
   call WRITESTRING
   add al, distPlat
   mov dl, 26
   mov dh, al
   call GOTOXY
   mov edx, OFFSET bDificuldade3
   call WRITESTRING
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
TelaDificuldade ENDP
```

Código Tela Instruções

```
TelaInstrucoes PROC
LTI3:
   call LimpaTela
   call Bordas
   mov eax, green+(black*16)
   call SETTEXTCOLOR
   mov dl, 18
   mov dh, 2
   call GOTOXY
   mov edx, OFFSET nome
   call WRITESTRING
   mov al, '/'
   call WRITECHAR
   mov edx, OFFSET bcomoJogar
   call WRITESTRING
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 5
   mov dh, 4
   call GOTOXY
   mov edx, OFFSET mInstrucoes1
   call WRITESTRING
   mov dl, 3
   mov dh, 5
   call GOTOXY
   mov edx, OFFSET mInstrucoes2
   call WRITESTRING
   mov dl, 3
   mov dh, 6
   call GOTOXY
   mov edx, OFFSET mInstrucoes3
   call WRITESTRING
   mov dl, 5
   mov dh, 8
   call GOTOXY
   mov edx, OFFSET mInstrucoes4
   call WRITESTRING
   mov dl, 3
   mov dh, 9
   call GOTOXY
   mov edx, OFFSET mInstrucoes5
```

```
 call WRITESTRING
 mov dl, 3
 mov dh, 10
 call GOTOXY
 mov edx, OFFSET mInstrucoes6
 call WRITESTRING
 mov dl, 5
 mov dh, 12
 call GOTOXY
 mov edx, OFFSET mInstrucoes7
 call WRITESTRING
 mov dl, 3
 mov dh, 13
 call GOTOXY
 mov edx, OFFSET mInstrucoes8
call WRITESTRING
 mov dl, 3
 mov dh, 14
 call GOTOXY
 mov edx, OFFSET mInstrucoes9
 call WRITESTRING
 mov dl, 5
 mov dh, 16
 call GOTOXY
 mov edx, OFFSET mInstrucoes10
 call WRITESTRING
 mov dl, 3
 mov dh, 17
 call GOTOXY
 mov edx, OFFSET mInstrucoes11
 call WRITESTRING
 mov dl, 3
 mov dh, 18
 call GOTOXY
 mov edx, OFFSET mInstrucoes12
 call WRITESTRING
 mov dl, 3
 mov dh, 19
 call GOTOXY
 mov edx, OFFSET mInstrucoes13
 call WRITESTRING
```

```
    movzx eax, tMaxX
    sub eax, 5
    mov dl, al
    movzx eax, tMaxY
    sub eax, 2
    mov dh, al
    call GOTOXY
    mov eax, green+(black*16)
    call SETTEXTCOLOR
    mov al, '-'
    call WRITECHAR
    mov al, '>'
    call WRITECHAR
    mov dl, 0
    mov dh, tMaxY
    call GOTOXY
LTI1:
    mov eax,50
    call Delay
    call ReadKey
    jz  LTI1
    cmp dx, 0027h
    jne LTI1
    call LimpaTela
    call Bordas
    mov eax, green+(black*16)
    call SETTEXTCOLOR
    mov dl, 18
    mov dh, 2
    call GOTOXY
    mov edx, OFFSET nome
    call WRITESTRING
    mov al, '/'
    call WRITECHAR
    mov edx, OFFSET bcomoJogar
    call WRITESTRING
    mov dl, 13
    mov dh, 4
    call GOTOXY
    mov edx, OFFSET mInstrucoes14
    call WRITESTRING
```

```
mov eax, white+(black*16)
call SETTEXTCOLOR
mov dl, 5
mov dh, 7
call GOTOXY
mov edx, OFFSET mInstrucoes15
call WRITESTRING
mov dl, 5
mov dh, 10
call GOTOXY
mov edx, OFFSET mInstrucoes16
call WRITESTRING
mov dl, 3
mov dh, 11
call GOTOXY
mov edx, OFFSET mInstrucoes17
call WRITESTRING
mov dl, 5
mov dh, 14
call GOTOXY
mov edx, OFFSET mInstrucoes18
call WRITESTRING
mov dl, 3
mov dh, 15
call GOTOXY
mov edx, OFFSET mInstrucoes19
call WRITESTRING
mov dl, 3
movzx eax, tMaxY
sub eax, 2
mov dh, al
call GOTOXY
mov eax, green+(black*16)
call SETTEXTCOLOR
mov al, '<'
call WRITECHAR
mov al, '-'
call WRITECHAR
movzx eax, tMaxX
sub eax, 5
mov dl, al
```

```
   movzx eax, tMaxY
   sub eax, 2
   mov dh, al
   call GOTOXY
   mov al, '-'
   call WRITECHAR
   mov al, '>'
   call WRITECHAR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
LTI2:
   mov eax,50
   call Delay
   call ReadKey
   jz  LTI2
   cmp dx, 0025h
   je LTI3
   cmp dx, 0027h
   jne LTI2
   ret
TelaInstrucoes ENDP
```

Código Tela Créditos

```
TelaCreditos PROC
   call LimpaTela
   call Bordas
   mov eax, green+(black*16)
   call SETTEXTCOLOR
   mov dl, 18
   mov dh, 2
   call GOTOXY
   mov edx, OFFSET nome
   call WRITESTRING
   mov al, '/'
   call WRITECHAR
   mov edx, OFFSET bcreditos
   call WRITESTRING
   mov eax, white+(black*16)
   call SETTEXTCOLOR
```

```
mov dl, 5
mov dh, 4
call GOTOXY
mov edx, OFFSET mcreditos1
call WRITESTRING
mov dl, 3
mov dh, 5
call GOTOXY
mov edx, OFFSET mcreditos2
call WRITESTRING
mov dl, 3
mov dh, 6
call GOTOXY
mov edx, OFFSET mcreditos3
call WRITESTRING
mov dl, 3
mov dh, 7
call GOTOXY
mov edx, OFFSET mcreditos4
call WRITESTRING
mov dl, 3
mov dh, 8
call GOTOXY
mov edx, OFFSET mcreditos5
call WRITESTRING
mov dl, 5
mov dh, 10
call GOTOXY
mov edx, OFFSET mcreditos6
call WRITESTRING
mov dl, 3
mov dh, 11
call GOTOXY
mov edx, OFFSET mcreditos7
call WRITESTRING
mov eax, green+(black*16)
call SETTEXTCOLOR
mov dl, 12
mov dh, 16
call GOTOXY
mov edx, OFFSET mcreditos8
```

```
call WRITESTRING
mov eax, white+(black*16)
call SETTEXTCOLOR
mov dl, 3
mov dh, 18
call GOTOXY
mov edx, OFFSET mcreditos9
call WRITESTRING
mov dl, 30
mov dh, 19
call GOTOXY
mov edx, OFFSET mcreditos10
call WRITESTRING
mov dl, 3
mov dh, 21
call GOTOXY
mov edx, OFFSET mcreditos11
call WRITESTRING
mov dl, 30
mov dh, 22
call GOTOXY
mov edx, OFFSET mcreditos12
call WRITESTRING
movzx eax, tMaxX
sub eax, 5
mov dl, al
movzx eax, tMaxY
sub eax, 2
mov dh, al
call GOTOXY
mov eax, green+(black*16)
call SETTEXTCOLOR
mov al, '-'
call WRITECHAR
mov al, '>'
call WRITECHAR
mov dl, 0
mov dh, tMaxY
call GOTOXY
mov eax, white+(black*16)
call SETTEXTCOLOR
```

```
LTI1:
   mov eax,50
   call Delay
   call ReadKey
   jz  LTI1
   cmp dx, 0027h
   jne LTI1
   ret
TelaCreditos ENDP
```

Código Seta Tela Perdeu

```
SetaTelaPerdeu PROC
   push dx
   mov dl, 15
   mov dh, 18
   call GOTOXY
   mov eax, black+(black*16)
   call SETTEXTCOLOR
   mov al, ' '
   call WRITECHAR
   call WRITECHAR
   mov dl, 15
   mov dh, 21
   call GOTOXY
   mov eax, black+(black*16)
   call SETTEXTCOLOR
   mov al, ' '
   call WRITECHAR
   call WRITECHAR
   pop dx
   cmp dx, 0026h
   jne LPS1
   cmp posSeta1, 0000h
   jbe LPS2
   dec posSeta1
LPS1:
   cmp dx, 0028h
   jne LPS2
   cmp posSeta1, 0001h
   jae LPS2
```

```
   inc posSeta1
LPS2:
   cmp posSeta1, 0
   je seta1
   mov dh, 21
   jmp setasai
seta1:
   mov dh, 18
setasai:
   mov dl, 15
   call GOTOXY
   mov eax, green+(black*16)
   call SETTEXTCOLOR
   mov al, '-'
   call WRITECHAR
   mov al, '>'
   call WRITECHAR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   ret
SetaTelaPerdeu ENDP
```

Código Tela Perdeu

```
TelaPerdeu PROC
   call LimpaTela
   call Bordas
   mov eax, green+(black*16)
   call SETTEXTCOLOR
   mov dl, 23
   mov dh, 3
   call GOTOXY
   mov edx, OFFSET mPerdeu1
   call WRITESTRING
   mov dl, 15
   mov dh, 18
   call GOTOXY
   mov al, '-'
```

```
call WRITECHAR
mov al, '>'
call WRITECHAR
mov eax, white+(black*16)
call SETTEXTCOLOR
mov dl, 10
mov dh, 7
call GOTOXY
mov edx, OFFSET mPerdeu21
call WRITESTRING
mov dl, 19
mov dh, 8
call GOTOXY
mov edx, OFFSET mPerdeu22
call WRITESTRING
mov dl, 22
mov dh, 11
call GOTOXY
mov edx, OFFSET mPerdeu3
call WRITESTRING
movzx eax, score
call WRITEDEC
mov dl, 24
mov dh, 13
call GOTOXY
mov edx, OFFSET mPerdeu4
call WRITESTRING
movzx eax, timemax
sub al, time
call WRITEDEC
mov dl, 22
mov dh, 18
call GOTOXY
mov edx, OFFSET mPerdeu5
call WRITESTRING
mov dl, 27
mov dh, 21
call GOTOXY
mov edx, OFFSET mPerdeu6
call WRITESTRING
mov dl, 0
```

```
   mov dh, tMaxY
   call GOTOXY
   ret
TelaPerdeu ENDP
```

Código Tela Acaba Tempo

```
TelaAcabaTempo PROC
   call LimpaTela
   call Bordas
   mov eax, green+(black*16)
   call SETTEXTCOLOR
   mov dl, 21
   mov dh, 6
   call GOTOXY
   mov edx, OFFSET mPerdeuTempo1
   call WRITESTRING
   mov dl, 15
   mov dh, 18
   call GOTOXY
   mov al, '-'
   call WRITECHAR
   mov al, '>'
   call WRITECHAR
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 22
   mov dh, 11
   call GOTOXY
   mov edx, OFFSET mPerdeu3
   call WRITESTRING
   movzx eax, score
   call WRITEDEC
   mov dl, 24
   mov dh, 13
   call GOTOXY
   mov edx, OFFSET mPerdeu4
   call WRITESTRING
   movzx eax, timemax
   call WRITEDEC
   mov dl, 22
```

```
   mov dh, 18
   call GOTOXY
   mov edx, OFFSET mPerdeu5
   call WRITESTRING
   mov dl, 27
   mov dh, 21
   call GOTOXY
   mov edx, OFFSET mPerdeu6
   call WRITESTRING
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
TelaAcabaTempo ENDP
```

Código Tempo Tela

```
TempoTela PROC
   mov dl, 13
   mov dh, 1
   call GOTOXY
   movzx eax, time
   call WRITEDEC
   dec eax
   mov time, al
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
TempoTela ENDP
```

Código Score Tela

```
ScoreTela PROC
   mov dl, 30
   mov dh, 1
   call GOTOXY
   movzx eax, score
   call WRITEDEC
   mov dl, 0
   mov dh, tMaxY
```

```
   call GOTOXY
   ret
ScoreTela ENDP
```

Código Seleciona Cor Plataforma

```
CorSelPlat PROC
   mov dl, 49
   mov dh, 1
   call GOTOXY
   movzx eax, corSele
   mov edx, 00000016
   mul edx
   call SETTEXTCOLOR
   mov al, ' '
   call WRITECHAR
   mov dl, 51
   mov dh, 1
   call GOTOXY
   movzx eax, (corSele+2)
   mov edx, 00000016
   mul edx
   call SETTEXTCOLOR
   mov al, ' '
   call WRITECHAR
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
CorSelPlat ENDP
```

Código Apaga Armadilhas

```
ApagaArm PROC
   mov eax, black+(black*16)
   call SETTEXTCOLOR
   mov ecx, 4
   mov bx, 21
   sub bx, 2
```

```
   mov esi, 0
LP1:
   mov dh, bl
   push ecx
   movzx ecx, nArmadilhas
LP2:
   mov dl,armadilhas[esi]
   call GOTOXY
   inc esi
   mov al, ' '
   call WRITECHAR
   loop LP2
   pop ecx
   sub bl, distPlat
   loop LP1
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
ApagaArm ENDP
```

Código Desenha Armadilhas

```
DesenhaArm PROC
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov ecx, 4
   mov bx, 21
   sub bx, 2
   mov esi, 0
LP1:
   mov dh, bl
   push ecx
   movzx ecx, nArmadilhas
LP2:
   mov dl,armadilhas[esi]
   call GOTOXY
   inc esi
   mov al, '&'
```

```
   call WRITECHAR
   loop LP2
   pop ecx
   sub bl, distPlat
   loop LP1
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
DesenhaArm ENDP
```

Código Verifica Posição Inicial

```
VerificaPosIni PROC
   push eax
   push ecx
   push edi
   movzx ecx, nArmadilhas
   mov ebx, 1
   cld
   repne scasb
   jnz fim
   mov ebx, 0
fim:
   pop edi
   pop ecx
   pop eax
   ret
VerificaPosIni ENDP
```

Código Cria Armadilhas Início

```
CriaArmInicio PROC
   mov ecx, 4
   mov ebx, 1
   mov edx, 0
   mov edi, OFFSET armadilhas
L2:
   push ebx
```

```
    push ecx
    movzx ecx, nArmadilhas
L1:
    call Randomize
    movzx eax, tMaxX
    sub eax, 4
    call RandomRange
    inc al
    call VerificaPosIni
    cmp ebx, 1
    jne L1
    mov armadilhas[edx], al
    inc edx
    loop L1
    pop ecx
    pop ebx
    movzx eax, nArmadilhas
    push edx
    mul ebx
    pop edx
    mov edi, OFFSET armadilhas
    add edi, eax
    inc edi
    inc ebx
    loop L2
    call DesenhaArm
    ret
CriaArmInicio ENDP
```

Código Troca Cor Plataforma

```
TrocaCorPlat PROC
    call Randomize
    mov eax, 9
    call RandomRange
    imul ax, TYPE coresDisp
    mov bx, [coresDisp + ax]
    mov corPlatAtual, bx
    movzx eax, bx
    call SETTEXTCOLOR
    mov al, tMaxY
```

```
   sub al, 5
   mov dl, 1
   mov dh, al
   call GOTOXY
   movzx eax, tMaxX
   sub eax, 2
   mov ecx, eax
LTCP1:
   mov al, ':'
   call WRITECHAR
   loop LTCP1
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
   ret
TrocaCorPlat ENDP
```

Código Procedimento Principal

```
main PROC
   call CLRSCR
start:
   call LimpaTela
   call Bordas
   call Plataformas
   call TelaInicio
   mov posSeta, 0
   mov posSeta1, 0
   mov score, 0
   mov dl, 20
   mov dh, BYTE PTR [platInicial]
   call GOTOXY
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov al, '-'
   call WRITECHAR
   mov al, '>'
   call WRITECHAR
   mov dl, 0
```

```
   mov dh, tMaxY
   call GOTOXY
AguardaTecla1:
   mov eax,50
   call Delay
   call ReadKey
   jz  AguardaTecla1
   cmp dx,000Dh
   je LS1
   cmp dx, 0051h
   je fim
   call PrintSeta
   jne AguardaTecla1
LS1:
   cmp posSeta, 0000h
   je jogo
   cmp posSeta, 0001h
   je instrucoes
   cmp posSeta, 0002h
   je creditos
jogo:
   call TelaDificuldade
   mov dl, 20
   mov dh, BYTE PTR [platInicial]
   call GOTOXY
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   mov al, '-'
   call WRITECHAR
   mov al, '>'
   call WRITECHAR
   mov dl, 0
   mov dh, tMaxY
   call GOTOXY
AguardaTecla2:
   mov eax,50
   call Delay
   call ReadKey
   jz  AguardaTecla2
   cmp dx,000Dh
   je LS2
```

```
    cmp dx, 0051h
    je fim
    call PrintSeta
    jne AguardaTecla2
LS2:
    cmp posSeta, 0000h
    je facil
    cmp posSeta, 0001h
    je medio
    cmp posSeta, 0002h
    je dificil
facil:
    mov nArmadilhas, 3
    mov quantArmadilhas, 12
    mov time, 90
    jmp play
medio:
    mov nArmadilhas, 7
    mov quantArmadilhas, 28
    mov time, 60
    jmp play
dificil:
    mov nArmadilhas, 15
    mov quantArmadilhas, 60
    mov time, 30
play:
    call TelaJogo
    cmp eax, 0
    je fim1
    cmp eax, 1
    je start
    cmp eax, 2
    je fim2
    jmp start
instrucoes:
    call TelaInstrucoes
    jmp start
creditos:
    call TelaCreditos
    jmp start
fim1:
```

```
   call TelaPerdeu
AguardaTecla3:
   mov eax,50
   call Delay
   call ReadKey
   jz  AguardaTecla3
   cmp dx,000Dh
   je saiAguardaTecla3
   call SetaTelaPerdeu
   jne AguardaTecla3
saiAguardaTecla3:
   cmp posSeta1, 0
   je start
   jmp fim
fim2:
   call TelaAcabaTempo
AguardaTecla4:
   mov eax,50
   call Delay
   call ReadKey
   jz  AguardaTecla4
   cmp dx,000Dh
   je saiAguardaTecla4
   call SetaTelaPerdeu
   jne AguardaTecla4
saiAguardaTecla4:
   cmp posSeta1, 0
   je start
   jmp fim
fim:
   movzx eax, tMaxY
   inc eax
   mov dl, 0
   mov dh, al
   call GOTOXY
   mov eax, white+(black*16)
   call SETTEXTCOLOR
   exit
main ENDP
END main
```

Referências

[1] "Irvine Library Help." Disponível em: http://programming.msjc.edu/asm/help/index.html?\\page=source%2Fabout.htm. . Acesso em: 26 abr. 2017.

[2] "The MASM32 SDK." Disponível em: http://www.masm32.com/. Acesso em: 26 abr. 2017.

[3] "A history of the computer game." Disponível em: https://www.jesperjuul.net/thesis/2-historyofthecomputergame.html. Acesso em: 28 jun. 2017.

[4] "The Dot Eaters." Disponível em: http://thedoteaters.com/?attachment_id=7550. Acesso em: 28 jun. 2017.

[5] "History of computer games." Disponível em: http://www.emunix.emich.edu/~evett/GameProgramming/History.pdf. Acesso em: 28 jun. 2017.

[6] "Space Invaders." Disponível em: https://en.wikipedia.org/wiki/Space_Invaders. Acesso em: 28 jun. 2017.