UNIVERSIDADE FEDERAL DE SÃO CARLOS

CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO 3
LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES II

Bruna Fernandes Prates 743513

Esther Calderan Hoffmann 743529

Professor: Luciano Neris

Sumário

1. Introdução

1.1 Histórico sobre jogos

Atualmente, videogames existem em milhares de casas ao redor do mundo, contudo inicialmente, eram encontrados apenas em laboratórios de pesquisa científicas.

A criação dos primeiros jogos foi relacionada a testes e demonstrações de teorias em áreas como: interação humano-computador, aprendizagem adaptativa e estratégia militar. Devido à falta de documentação destes testes, torna-se difícil determinar qual foi, de fato, o primeiro jogo eletrônico criado.

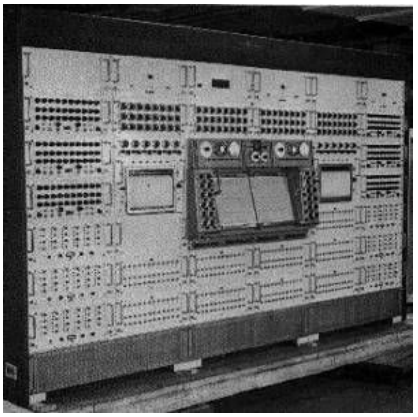
Um dos jogos já considerado como pioneiro é o ***OXO***, desenvolvido por A. S. Douglas, em 1952, para o computador Electronic Delay Storage Automatic Calculator (EDSAC), no Laboratório Matemático da Universidade de Cambridge. Este era um jogo da velha, em que cada partida realizava-se entre um usuário e seu oponente artificialmente inteligente, o qual poderia efetuar um jogo "perfeito". O jogador fazia seu movimento utilizando o controle de um telefone de disco, selecionando qual dos nove quadrados no quadro desejava mover em seguida.

Figura 1.1 - OxO em uma emulação no Windows do computador EDSAC.

Fonte: http://www.gameclassification.com/files/games/Oxo.png

Em 1955, foi desenvolvido o Hutspiel, um jogo de guerra construído pelo exército americano para simular um conflito com a União Soviética na Europa. Criado para dois jogadores, um azul e um vermelho, representando o comandante da OTAN e da URSS respectivamente.

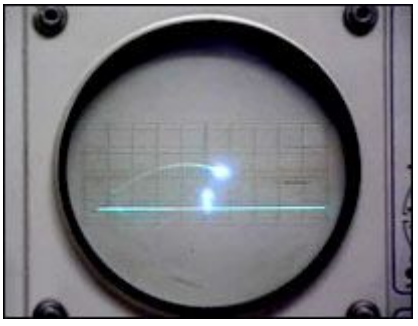
Figura 1.2 - Hutspiel



Fonte: http://www.gameclassification.com/files/games/Hutspiel-.jpg

Três anos depois, o físico norte-americano William Higinbotham criou o "Tennis Programming", também conhecido como "Tennis For Two". Consistia em uma simulação simples de uma partida de tênis, exibida em um osciloscópio e processada por um computador analógico. O objetivo do programador foi simplesmente entreter os convidados no dia da visita anual realizada pelo Laboratório Nacional de Brookhaven. No jogo, a bola é representada por um ponto, e os jogadores controlavam seu movimento por cima de uma linha vertical que representava a rede.

Figura 1.3 - Tennis For Two



Fonte: http://www.gameclassification.com/files/games/Tennis-for-two.jpg

Em 1962 “Spacewar!” foi finalizado. Desenvolvido em Assembly pelos estudantes do MIT Stephen Russell, Peter Samson, Dan Edwards, Martin Graetz, Alan Kotok, Steve Piner e Robert A. Saunders, era composto por duas naves espaciais - controladas por jogadores - e possuía como objetivo acertar torpedos uns nos outros. Este tornou-se um dos primeiros jogos eletrônicos a ter uma distribuição nacional, através de sua inclusão como programa de testes nos computadores comercializados na época.

Figura 1.4 - Spacewar! sendo jogado por dois de seus criadores

Fonte: http://mstatic.mit.edu/mit150/025p.jpg

Em 1966, o engenheiro Ralph Baer pensava em criar uma TV interativa com jogos, mas falhou. O protótipo se chamava "Brown Box" e, em 1971, Magnavox (subsidiária da Philips) comprou tal projeto de Baer e começou a desenvolver o Odyssey, primeiro console de videogame conectado à TV. O aparelho foi um fracasso nas vendas, contudo um de seus 28 jogos serviu como inspiração para o jogo “Pong”, do Atari, o qual seria criado posteriormente, marcando uma nova era na história dos video games.

Figura 1.5 - Brown Box

Fonte: http://www.museumofplay.org/online-collections/images/110_4140/Z0049414.jpg

Ainda em 1971, Nolan Bushnell adaptou o Spacewar!, criando o Computer Space, primeiro arcade do mundo. No ano seguinte, Bushnell fundou a empresa “Atari”.

Em 1972, na Califórnia, ele inventou o Pong (um jogo de ping-pong para arcade), juntamente com Ted Dabney e Larry Bryan.

Figura 1.6 - Pong

Fonte: https://www.audienciaelectronica.net/wp-content/uploads/2012/11/pong_cabbig-prev.jpg

“Maze War”, o primeiro jogo em três dimensões que usou avatares representando os jogadores, foi elaborado em 1973, por Steve Colley, no “Imlac PSD-1” no centro de pesquisas da NASA.

Figura 1.7 - Maze War

Fonte: http://contembits.com.br/imagens/Games/1973-lmdac.jpg

A Apple em 1977 lança o “Apple II”, primeiro computador pessoal feito em série. Tornou-se pioneiro em sua época, em virtude de ser o único que possuía cores e, devido à este fato, foi muito utilizado para jogos.

Figura 1.8 - Apple II

Fonte: http://oldcomputers.net/pics/appleii-system.jpg

Em 1978, é criado o primeiro MUD (Multi-User Dungeon): um RPG multiplayer no qual os jogadores assumem o papel de uma personagem em um mundo virtual, recebendo informações textuais que descrevem cenários, personagens e outras criaturas controladas pelo computador. Ele originou um subgênero de RPGs (Role-Playing Games) e inspirou os MMORPGs (Massive Multiplayer Online Role-Playing Games).

Em 1979, foi lançado o Atari 800, o qual redefiniu as expectativas relacionadas ao que um computador pessoal poderia apresentar em matéria de gráficos e som. O foco das vendas do aparelho foi como video-game; contudo, também podia ser usado como computador. Embora viesse, originalmente, apenas com entrada para os cartuchos, era possível adquirir uma unidade de disquetes e um teclado, transformando-o em um computador completo.

Não existiram muitos programas para o Atari, uma vez que o foco eram os jogos. Seu principal uso como computador era desenvolver programas em BASIC.

Figura 1.9 - Atari 800

Fonte: https://i.redd.it/nhtgehld2i501.jpg

Paralelamente, a Mattel fabrica seu primeiro console, o Intellivision; que continha os melhores gráficos. Era possível, também, comprar o seu teclado de acessório e usá-lo como um computador. O teclado possuía como microprocessador o 6502 da MOS Technology (8 bits), 16 Kb de RAM (expansível até 8 Mb), gravador cassete controlado digitalmente para gravação/leitura de dados e leitura de áudio, portas de expansão e entrada para impressora. Além disso, alguns anos depois foi lançado o Intellivoice, acessório que possuía um chip sintetizador de voz, sendo então possível adicionar vozes ao videogame. Contudo, apenas 13 jogos eram compatíveis com o mesmo, e sua fabricação foi cortada no ano seguinte. O Intellivision vendeu 3 milhões de unidades.

Figura: 1.10 - Intellivision

Fonte: https://tecnoblog.net/wp-content/uploads/2018/05/Intellivision-700x394.jpg

Todo ano lançava-se um novo computador e/ou console de melhor qualidade. Os computadores pessoais se popularizaram pelo mundo todo, garantindo o mercado dos jogos para PC e, assim, tornando possível o desenvolvimento de muitos que viriam marcar a vida de milhares de pessoas.

A quantidade de categorizações para jogos eletrônicos consequentemente aumentou: desde jogos extremamente elaborados, com gráficos realistas nos quais as pessoas pagam caro para se divertir, até jogos online casuais como Neopets e Club Penguim.

Uma pesquisa realizada na Game Developers Conference deste ano (2018) mostra que 59% dos desenvolvedores de jogos possui muito interesse em desenvolver para PC ao invés de outras plataformas, mostrando que o computador ainda possui muita importância em um dos mercados que mais fatura no mundo.

1.2 Estratégia de Implementação

Em virtude da nítida extensão do projeto, este foi dividido em determinadas etapas para melhor entendimento dos progressos obtidos. Dessa forma, segue abaixo descrição detalhada do que foi feito em cada uma das 3 etapas.

Etapa 1

- ➔ Pesquisa e reflexão sobre possibilidades de jogos em assembly;
- ➔ Escolha da categoria desejada;

- ➔ Definição do funcionamento do jogo, restrições, interface e regras;
- ➔ Elaboração do Relatório 1.

Etapa 2

- ➔ Ajuste de projeto após apresentação do primeiro relatório;
- ➔ Análise da biblioteca Irvine para examinar a gama de possibilidades a oferecer;
- ➔ Definição da lógica do jogo como um todo;
- ➔ Esquematização das rotinas e diagrama de estados;
- ➔ Início da implementação do código;
- ➔ Elaboração do Relatório 2.

Etapa 3

- ➔ Ajustes após apresentação do segundo relatório;
- ➔ Implementação do código restante;
- ➔ Realização de testes;
- ➔ Correção de possíveis erros;
- ➔ Realização de mudanças necessárias;
- ➔ Elaboração do Relatório 3;
- ➔ Preparação do grupo para apresentar o jogo.

Além disso, foram estabelecidas pequenas divisões de tarefas entre os membros do grupo para melhor elaboração do projeto, buscando qualidade e entrega dentro do prazo estipulado. Tendo como objetivo que cada membro adquira o máximo de conhecimento na elaboração do jogo, ambos estarão presentes em todos os procedimentos necessários. Entretanto, para melhor organização e minimização de riscos, determinou-se a seguinte divisão: Bruna Fernandes será encarregada de elaborar a lógica para que as letras apareçam em ordem aleatória e desçam pela tela paulatinamente, realizando a leitura do teclado e verificação de resposta correta ou errônea. Esther Hoffmann será responsável por distinguir a dificuldade de cada nível, selecionando as palavras pertencentes a cada um destes e organizando-as em um arquivo auxiliar; além disso, irá elaborar o design das telas principais.

2. Descrição

2.1 Descrição Geral

Catch Words assemelha-se a um jogo de caça-palavras, porém funciona de forma mais alternativa. As letras que formam uma palavra específica caem verticalmente pela tela, embaralhadas e separadas umas das outras, tornando mais difícil a identificação da mensagem. O objetivo principal é conseguir identificá-la e digitá-la através do teclado antes que todas as letras cheguem até o fim da tela e, se isto ocorrer, o jogo é encerrado. Caso o jogador consiga digitar a palavra a tempo e corretamente, todas as letras somem e um outro conjunto aparece no topo da tela. Se alguma dessas ações não forem efetuadas, o usuário perde o jogo e este encerra-se. Determinou-se também que, em cada fase do jogo, ocorrerá o surgimento aleatório de letras coloridas. A palavra quando formada corretamente acrescentará uma pontuação correspondente ao nível seguinte, representando um "bônus" do nível atual.

Além disso, conforme a quantidade de acertos aumenta, o score também eleva sua quantidade de pontos e os níveis tornam-se mais avançados, fazendo com que as palavras sejam maiores e mais difíceis de serem identificadas. Para isto, determinou-se pontuações específicas a serem acrescentadas, assim como quantidade de palavras por nível.

- Nível 1: 5 pontos por palavra correta - Total de 10 palavras
- Nível 2: 10 pontos por palavra correta - Total de 7 palavras
- Nível 3: 20 pontos por palavra correta - Total de 5 palavras

Para o bom funcionamento do jogo, será utilizada a estrutura de dados Lista. Ao passo que uma número aleatórios são gerados para indicar tanto a posição das letras na tela quanto a palavra selecionada no arquivo, estes são inseridos em uma lista, por ordem de ncessidade. Tal vetor é utilizado para garantir que nenhum valor se repita. Além disso, utiliza-se tal estrutura de dados para armazenar corretamente as letras digitadas pelo usuário para posterior verificação com o resultado desejado.

Ademais, as principais variáveis utilizadas estão descritas abaixo:

- **Menu:** variável responsável por guardar todo o desenho correspondente ao menu principal;
- **RULES:** armazena o desenho correspondente às instruções necessárias ao jogo;
- **NextLevel:** contém a tela que indica para qual nível o jogador está se direcionando;
- **FailSpecial:** tela que indica ao jogador que não acertou qual era a palavra especial;
- **Fail:** armazena a tela indicando que o usuário perdeu e o jogo está encerrado;
- **Correct:** tela de acerto da palavra comum;
- **CorrectSPECIALcoming:** tela de acerto e indicando que a próxima palavra a surgir é especial;
- **WIN:** tela de "congrats" indicando a vitória no jogo;
- **ScreenGame:** Tela base do jogo, onde as letras percorrem a tela e estão fixas as frases "DON'T GIVE UP!" e "YOU CAN DO THIS!", bem como SCORE, LEVEL E WORD;
- **fileLevel1, fileLevel2, fileLevel3, fileLevel4:** estas variáveis são responsáveis por armazenar o título dos arquivos texto de cada uma das fases existentes;
- **fileName:** receberá o título de algum dos 4 arquivos existentes, dependendo do nível em que o jogador se encontra;
- **BytesToRead:** quantidade de bytes que devem ser lidos em cada nível. Por exemplo, no nível 1 as palavras possuem tamanho 3, então serão lidos 3 bytes do arquivo;
- **buffer:** guarda o que é lido do arquivo;
- **bytesRead:** variável com foco maior na realização de testes, responsável por guardar a quantidade de bytes que já foram lidos até o momento;
- **SizeLevel:** armazena o múltiplo correspondente a cada nível, para garantir a correta locomoção do ponteiro no arquivo;
- **SizeWord:** tamanho das palavras correspondentes ao nível atual;
- **SizeOfFile:** tamanho do total do arquivo texto, ou seja, desde a primeira letra da primeira palavra até o último sinalizador de término da linha armazenado;

- **RandNum:** vetor que possui tamanho equivalente ao tamanho do arquivo a ser lido. Possui a função de armazenar números aleatórios de acordo com a necessidade para que sejam selecionadas palavras localizadas em tal posição. Além disso, garante a não repetição dos valores aleatórios;
- **Level:** possui como informação o nível em que o jogador se encontra. Porém, os níveis 1, 2 e 3 correspondem a 0, 1 e 2, respectivamente, nesta variável;
- **WordPosition:** variável responsável por armazenar a próxima posição de **RandNum** a ser preenchida. Teoricamente, então, guarda quantas palavras já foram utilizadas;
- **RandWord:** vetor do tipo char que contém a palavra selecionada do arquivo. Ou seja, recebe o conteúdo de **buffer**;
- **RandomPosition:** vetor que armazena posições aleatórias na tela, ou seja, cada valor armazenado neste corresponde a posição de cada letra de uma palavra;
- **ScreenLimit:** valor correspondente ao limite de tela permitido que as letras cheguem antes que o jogo termine;
- **InputPlayer:** armazena a tecla pressionada pelo usuário;
- **InputString:** vetor responsável por armazenar cada uma das letras tecladas pelo jogador quando este está digitando a palavra corretamente;
- **LetterPlayer:** guarda a posição em que a letra digitada pelo usuário se encontra em relação à palavra como um todo;
- **StatusWord:** indica o status da palavra que está descendo pela tela. Quando seu valor corresponde a 0, indica que o vocábulo ainda está incompleto; quando 1, simboliza o acerto por parte do jogador; enquanto 2 pode equivaler a duas situações: 1) jogador errou, porém a palavra era especial, então o jogo não é encerrado. O número altera-se para 1 e o jogo continua. 2) correspondia a uma palavra comum, a qual foi digitada erroneamente e, portanto, o jogo encerra-se;
- **Score:** armazena a pontuação atual do jogador;
- **SpecialWords:** sinaliza se a palavra é especial (0) ou não (1);
- **NumOfWord:** quantidade de palavras totais a serem exibidas no nível correspondente;

- **CurrentNum:** indica em qual das palavras totais o jogo se encontra (primeira, segunda, …);
- **MenuChoice:** simboliza a tela em que o usuário se encontra naquele momento. Quando está no menu principal, armazena 0; quando nas instruções, 1; e, ao chegar a tela de resultado, armazena 2.

2.2 Descrição da Rotina

Para o bom funcionamento do jogo e otimização do código, este foi implementado utilizando-se de diversas rotinas, como mostrado a seguir:

- **GetRandomScreenPosition:** utiliza o vetor *RandomPosition* para armazenar neste os valores aleatórios gerados (entre 15 e 65), que correspondem a posições arbitrárias na tela;
- **GetRandomNumber:** gera um número aleatório e guarda-o em *RandNum[WordPosition]*, para corresponder à posição da palavra escolhida no arquivo;
- **GetWord:** responsável por armazenar a palavra selecionada do arquivo na variável *RandWord* e, também, por incrementar o valor de *WordPosition* a cada execução;
- **ReadKeyboard:** convoca a função ReadKey da biblioteca Irvine para verificar se alguma tecla foi pressionada. Em caso afirmativo, dirige-se à rotina **VerifyLetter**, senão apenas encerra tal procedimento;
- **VerifyLetter:** compara *InputPlayer* com o que está armazenado em *RandWord[LetterPlayer]*, ou seja, realiza a verificação da letra teclada para classificá-la como correta ou incorreta. No primeiro caso, irá chamar a rotina **AddScore**, incrementar *LetterPlayer* e, em seguida, realiza a comparação deste valor obtido da incrementação com o tamanho da palavra esperada. Caso a comparação seja válida, altera-se *StatusWord* para 1, sinalizando que a palavra não deve mais percorrer a tela e o resultado positivo será exibido. Todavia, quando a letra é incorreta, compara-se, também, *LetterPlayer* ao

tamanho e, em caso de correspondência, *StatusWord* permanece com seu valor igual a 0;

- **AddScore:** realiza a adição de pontos ao score de acordo com o estabelecido por cada nível;
- **PrintInfoPlayer:** responsável por exibir na tela as informações pertencentes ao jogador, como score, nível, letras já digitadas;
- **PrintWordLine:** exibe as letras da palavra correspondente em suas primeiras posições aleatórias, ou seja, corresponde a linha de posição inicial da palavra embaralhada. Além disso, convoca **ReadKeyboard** para verificar se foi inserido algum input pelo usuário;
- **PrintWordMoving:** executa o movimento vertical das letras pela tela, incrementando e decrementando ebx para realizar tal ato. É executada enquanto a linha atual for diferente de *ScreenLimit*, além de verificar o valor de *StatusWord*;
- **PrintResultWord:** realiza a verificação de duas variáveis: *StatusWord* e *SpecialWord*. Se a primeira equivaler a 1 (a palavra toda digitada está correta) então é printado o que está armazenado em *Correct*. Porém, se equivaler a 2, necessita-se a verificação da variável *SpecialWord*. Caso seja uma palavra especial e o jogador tenha errado esta, *FailSpecial* é printada. Caso não, ao errar a palavra é exibida então a tela armazenada em *Fail*;
- **ClearForNewWord:** responsável por zerar variáveis para iniciar uma nova palavra do nível como, por exemplo, *LetterPlayer*, *StatusWord*, *InputString*, *RandWorD*, etc;
- **SetInfoLevels:** possui um switch em seu interior para setar corretamente os valores de variáveis de acordo com os critérios de cada nível;
- **PrintGameLevel:** primeiramente, armazena em ecx o valor de *NumOfWord* do nível atual e subtrai 2 deste valor. Em seguida, realiza o loop em que, a cada execução, chama-se a variável *ScreenGame* e diversos procedimentos: **SetInfoLevels**, **GetWord**, **GetRandomScreenPosition**, **PrintWordMoving**, **PrintResultWord**. Ao fim disto, altera-se *SpecialWord* para 1 (identificando que a palavra seguinte é especial), incrementa-se 1 no nível (para que o vocábulo tenha tamanho equivalente ao nível seguinte), assim como em

CurrentNum, altera as cores e invoca todas as rotinas presentes no loop. Após tal feito, deve-se novamente setar as variáveis para exibir na tela a última palavra correspondente ao nível atual, tendo o usuário errado ou acertado a palavra especial;

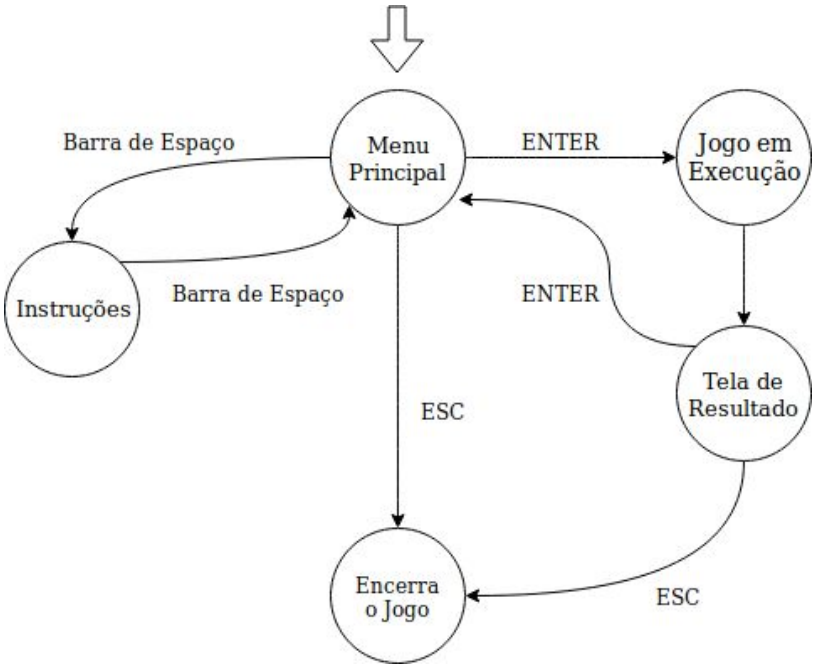
- **Game:** inicia-se exibindo a tela que indica o próximo nível (*NextLevel*) e acionando a função **SetInfoLevels**, **PrintGameLevel** (para ilustrar completamente o nível a ser jogado). Além disso, realiza a comparação de *StatusWord* com o valor 2 para verificar se deve-se encerrar a execução. Em caso negativo, incrementa-se 1 em *Level* e chama as mesmas funções novamente. Quando o jogador localiza-se no último nível e a comparação feita com *StatusWord* continua sendo inválida, então a tela WIN é exibida;
- **Main:** realiza a exibição do menu principal e permanece utilizando-se da função **ReadKeyboard** até que o usuário selecione alguma tecla de caminho. Após isto, confere qual foi pressionada: enter, space-bar ou esc.

3. Elementos Específicos

O usuário, ao iniciar o jogo, poderá escolher dentre 3 opções: Iniciá-lo (tecla Enter), direcionar-se às Instruções (Barra de Espaço) ou Sair do jogo (tecla Esc). Na tela de Instruções poderá ser estudada a condução do jogo: basta digitar no teclado a ordem das letras que correspondem a palavra que deve ser originada. Ao iniciar o Catch Words, será apresentada a primeira fase do jogo, em que as menores palavras caem embaralhadas pelo painel e, conforme acerta-se a ordem das letras, sua pontuação aumenta (a qual permanecerá visível durante todo o tempo). Desvendada todas as palavras do primeiro nível, o jogador é redirecionado ao nível seguinte, em que as palavras resultantes são maiores que as adivinhadas anteriormente. Ao pressionar uma tecla correspondente a alguma letra não desejada pela lógica do jogo, o usuário automaticamente perde, a tela de "fim de jogo" surge e este é encerrado. Em caso de vitória, a tela de resultado corresponderá a mensagem de parabenização. Em ambos os casos, para encerrar o jogo, basta apertar a tecla Esc e, no caso em que desejar-se jogar novamente, "Enter" o direcionará ao Menu Principal. Além disso, vale destacar a aparição de uma "palavra especial" em cada nível. Esta possuirá cor diferenciada e pontuação correspondente ao próximo nível. Caso o usuário não acerte, apenas não ganhará os pontos e o jogo continua normalmente.

O detalhamento acima pode ser resumido em um diagrama de estados como o representado abaixo, permitindo uma melhor visualização dos possíveis eventos presentes no projeto como um todo.

Figura 3.1: Diagrama de Estados de Catch Words



Ademais, para tornar melhor e mais concreta a visualização do planejamento, segue abaixo imagens correspondentes às telas a serem exibidas no jogo.

Figura 3.2: Tela Inicial

Figura 3.3: Tela de Instruções

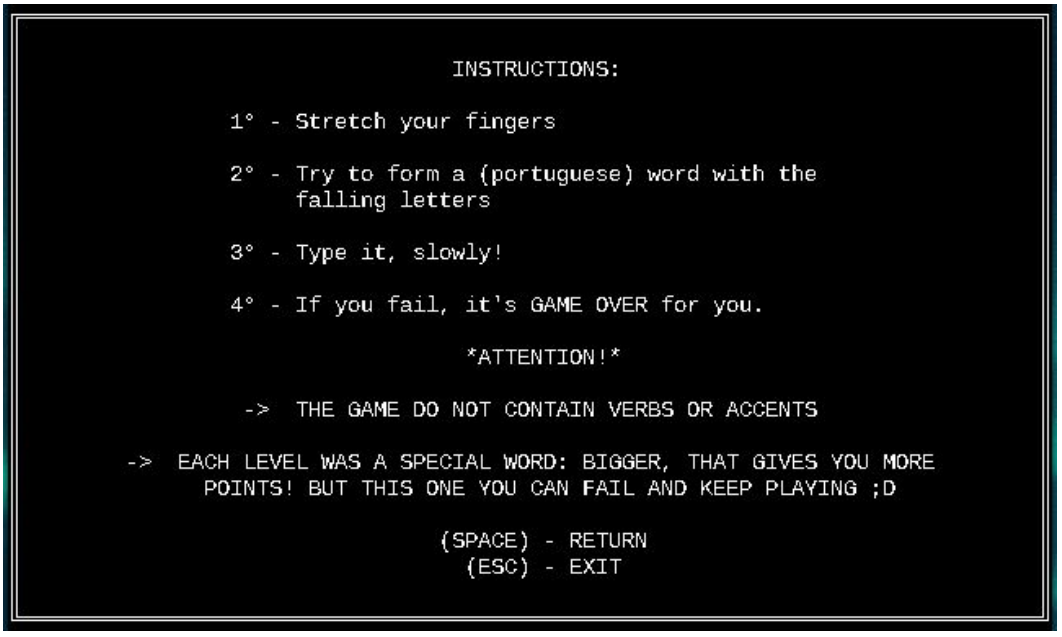


Figura 3.5: Jogo em Execução

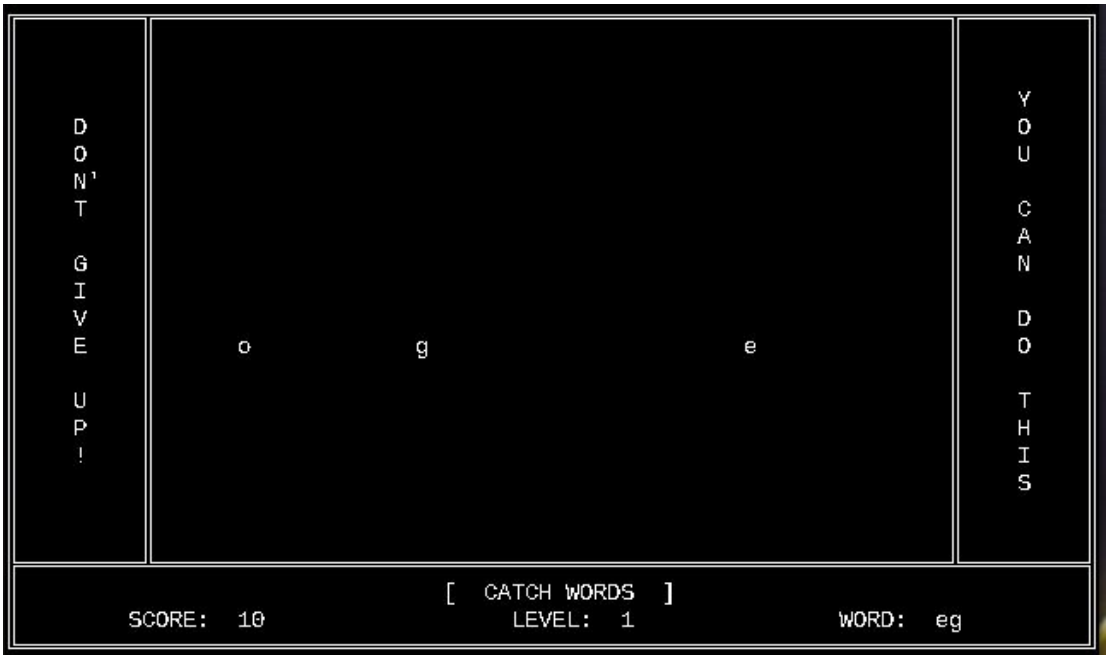


Figura 3.6: Tela de Fim de Jogo

4. Análise Crítica e Discussão

Evidencia-se determinadas limitações ao projeto devido a utilização da biblioteca restrita recomendada e, também, pelo fato de que Assembly trata-se de uma linguagem de máquina, a qual torna-se mais incomum no cotidiano atual de desenvolvimento de jogos. Desse modo, geram maior complexidade para execução de tarefas essenciais e exibição gráfica. Todavia, mesmo com tais dificuldades, o grupo comprometeu-se com uma produção eficiente, atingindo os objetivos objetivos propostos. Ademais, é possível perceber quantia de experiência e aprendizado que foi adquirida; aproximando o aluno do funcionamento interno de um computador, assim como o manuseio da linguagem que o controla de um nível mais baixo.

O grupo encontrou algumas dificuldades diante da elaboração do código e, para que o projeto funcionasse corretamente, algumas mudanças foram necessárias, como, por exemplo, a estrutura de dados utilizada. Ademais, em virtude da dificuldade apresentada em cada nível seguindo as primeiras determinações impostas, concordou-se em diminuir a complexidade das etapas.

Desta forma, com o planejamento e organização presentes no desenvolvimento do projeto, juntamente a dedicação apresentada pelo grupo, o projeto foi concluído dentro do período estipulado e de acordo com o desejado.

5. Referências Bibliográficas

Irvine, K.; Assembly Language for Intel-Based Computers, Prentice Hall, 7 (ou edição mais recente).

Evolução dos Video Games Parte 1 - Primórdios. 2017. Disponível em <https://fourgeeks.com.br/games/evolucao-dos-video-games-parte-1/>. Acesso em 27 de setembro de 2018.

História: Primeiros Jogos Digitais. 2015. Disponível em <http://www.ufpa.br/dicas/net1/int-h-jo.htm>. Acesso em 27 de setembro de 2018.

História dos Jogos de Computadores. Disponível em <http://www.ahistoria.com.br/jogos-de-computadores/>. Acesso em 28 de setembro de 2018.

A Evolução da Indústria dos Computadores & Video Games. 2017. Disponível em <https://crpgbook.files.wordpress.com/2017/06/1975e280932014.pdf>. Acesso em 28 de setembro de 2018.

Video Game History. 2018. Disponível em <https://www.history.com/topics/inventions/history-of-video-games>. Acesso em 28 de setembro de 2018.

Early History of Video Games. Disponível em <https://en.wikipedia.org/wiki/Early_history_of_video_games> . Acesso em 28 de setembro de 2018.

History of Video Games. Disponível em <https://en.wikipedia.org/wiki/History_of_video_games#Early_history_(1948%E2%80%931972)>. Acesso em 28 de setembro de 2018.

A Origem dos Jogos Eletrônicos. 2015. Disponível em <http://ultimateretroplayer.blogspot.com/2015/07/a-origem-dos-jogos-eletronicos-arcades.html>. Acesso em 28 de setembro de 2018.

História dos Jogos Eletrônicos. Disponível em <https://pt.wikipedia.org/wiki/Hist%C3%B3ria_dos_jogos_eletr%C3%B4nicos> . Acesso em 28 de setembro de 2018.

História dos Jogos de Computador. Disponível em <http://www.ahistoria.com.br/jogos-de-computador/>. Acesso em 29 de setembro de 2018.

The Most Important Gaming Platforms in 2018. 2018. Disponível em <https://www.statista.com/chart/4527/game-developers-platform-preferences/>. Acesso em 29 de setembro de 2018.

Irvine Library Help. Disponível em <http://programming.msjc.edu/asm/help/index.html>. Acessado em 25 de Outubro de 2018.

The MASM Forum Archive 2004 to 2012 - Topic: How to read from file in MASM. Disponível em <http://www.masmforum.com/board/index.php?topic=16266.0>. Acessado em 25 de Outubro de 2018.

The MASM Forum Archive 2004 to 2012 - Topic: SetFilePointer. Disponível em <http://www.masmforum.com/board/index.php?PHPSESSID=786dd40408172108b65a5a36b09c88c0&topic=18747.0> . Acessado em 25 de Outubro de 2018.

Text to ASCII Generator (TAAG). Disponível em <http://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20>. Acessado em 26 de Outubro de 2018.

The Complete Table of ASCII. Disponível em <https://theasciicode.com.ar/> . Acessado em 26 de Outubro de 2018.