Trabalho Final

Algoritmos e Programação – 2024/2

Prof. Dr. Thiago L. T. da Silveira
tltsilveira@inf.ufrgs.br

Instruções

- Leia este documento **atentamente**. Nele, estão descritos os requisitos mínimos e desejáveis (extras) quanto à implementação do Trabalho Final. Também são apresentadas diretrizes para entrega e critérios de avaliação.

Objetivo

O objetivo principal do Trabalho Final é exercitar conceitos de programação estudados ao longo do semestre através da implementação de um **jogo de computador** utilizando a linguagem C. Objetivos secundários incluem exercitar as habilidades de pesquisa – uma vez que o aluno poderá utilizar bibliotecas e conceitos que não foram trabalhados em aula para implementar certas funcionalidades – e de trabalho em equipe – uma vez que este deve ser realizado **por duplas ou trios**. O Trabalho Final *não* pode ser implementado individualmente.

Temática e Funcionamento Básico

O jogo a ser implementado é inspirado em ***Jetpack Joyride*** (acesse https://www.youtube.com/watch?v=0hU7tLtOIgE para ver uma *gameplay*). Nesse jogo, temos um personagem – controlado pelo jogador – que corre em um laboratório. O personagem deve utilizar um *jetpack* em suas costas para evitar obstáculos enquanto se move verticalmente. O jogo possui um mapa infinito. Enquanto o personagem não encosta em obstáculos, o mapa continua "correndo" (com a velocidade que aumenta com o passar do tempo). O personagem se depara com espinhos e moedas, que se alternam de forma aleatória. Além disso, ele encontra foguetes que aparecem de tempos em tempos e lasers, que quando ativados, impedem o personagem de trafegar. O personagem tem apenas uma vida; ou seja, ao ser acertado por algum obstáculo, o jogo acaba. O desempenho do jogador é registrado na forma de uma pontuação, levando em conta o número de moedas coletadas e a distância percorrida. O jogador passa de fase ao completar uma determinada pontuação.

Requisitos Mínimos

Implementações do Trabalho Final devem respeitar os seguintes requisitos mínimos:

- Os elementos visuais do jogo devem ser implementados e exibidos em modo texto (através de caracteres).
- O jogo não deve ter *delay*, ou seja, ao ser disparada uma ação do jogador, o jogo deve responder imediatamente (exceto quando o *delay* é deliberado). Por exemplo: se o jogador movimentar o personagem para cima, o deve ser feito imediatamente.
- O mapa do jogo será carregado por um arquivo texto nomeado `"mapa<número>.txt"` (Ex: `"mapa1.txt"`), onde `<número>` corresponde à fase atual. Cada mapa é composto por uma matriz de 12×240 caracteres. O jogo utiliza seções dessa matriz, de tamanho 12×30, que são exibidas em um determinado momento. À medida que o tempo de jogo passa, deve-se selecionar uma partição aleatória do tamanho dessa seção a partir da matriz lida do arquivo correspondente àquela fase.
- Os itens do jogo devem ser representados no arquivo do mapa com os seguintes caracteres:

Caractere	Significado
X	Parede
C	Moeda coletável
Z	Espinho
Espaço em branco	Área de trânsito (posição vazia)

Um exemplo de mapa pode ser visto na Figura 1.

- O jogador sempre se mantém em uma determinada coluna de uma seção carregada do mapa.
- A interação do jogador com o jogo se dá ao pressionar teclas específicas. As ações e teclas são as seguintes:

Tecla	Significado
Esc	Pausa o jogo se estiver rodando
Espaço (ou, alternativamente, o clique do *mouse*)	Move o jogador para cima com a *jetpack*

- O personagem não deve atravessar as paredes.
- Se o personagem entrar em contato com algum obstáculo, seja ele algum tipo espinho ou laser, o jogo termina.
- A pontuação do jogo se baseia no quão longe o jogador consegue ir sem em encostar em obstáculos, ou seja, a distância percorrida a partir do começo da fase. Essa pontuação deve ser mostrada na tela.
- As moedas coletadas contabilizam na pontuação conforme $P_t = 10 \times D_m + M$, onde P_t é a pontuação do jogador, D_m é a distância percorrida e M é a quantidade de moedas coletadas.
- O jogo deve ser capaz de funcionar com diferentes arquivos texto, devendo ter, ao menos, três fases diferentes. Você tem a liberdade para determinar qual a pontuação necessária para que o jogador passe de uma fase para outra.
- O menu (exibido sobre a tela do jogo, no cabeçalho ou rodapé da tela) deve ser navegável com o pressionar de teclas ou o *mouse* e possuir as seguintes opções:

Tecla	Significado
N	Novo Jogo: quando o jogador seleciona esta opção, um novo jogo é iniciado.
L	*Leaderboard*: quando o jogador seleciona esta opção, são apresentadas as melhores pontuações.
Q	Sair do jogo: quando o jogador seleciona esta opção, o jogo é encerrado.

- O *leaderboaord* será lido a partir de um arquivo binário (`"leaderboard.bin"`, por exemplo), contendo as informações de outras jogadas. Você deve armazenar e exibir, ao menos, os três melhores *scores*. Tais *scores* devem ser ordenados decrescentemente.

Você deverá pesquisar a utilização de bibliotecas para criação de uma interface para o programa. Lembre-se que o requisito mínimo é uma representação visual em modo caractere. A escolha da biblioteca e sua utilização ficarão a cargo dos grupos. Algumas bibliotecas sugeridas são a `conio2` e a `ncurses`, usadas para desenvolver interfaces em modo texto, para Windows e para Linux respectivamente. O aluno é encorajado a desenvolver o jogo utilizando interfaces gráficas. Neste caso, algumas bibliotecas que poderiam ser utilizadas seriam: a `Allegro`, a `raylib` (altamente recomendável) e a `sdl2`. O aluno também pode utilizar bibliotecas que não foram mencionadas aqui.

As Figura 1 apresenta um mapa de exemplo em arquivo texto. As seções desse mesmo mapa podem ser exibidas em um programa em modo texto (terminal) de diferentes formas (veja a Figura 2). A Figura 3 apresenta o uma seção do mapa de exemplo sendo exibida em um programa que usa a biblioteca `raylib`.

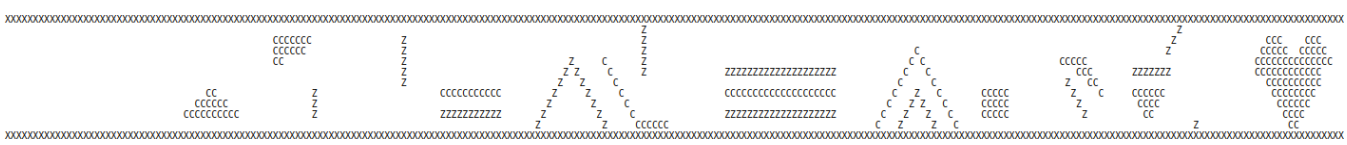

Figura 1: Exemplo de mapa.

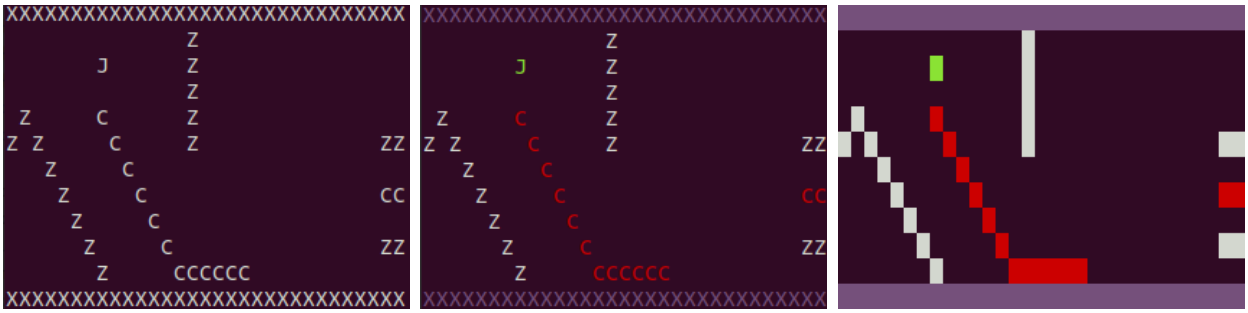

Figura 2: Representações em texto do mapa de exemplo.

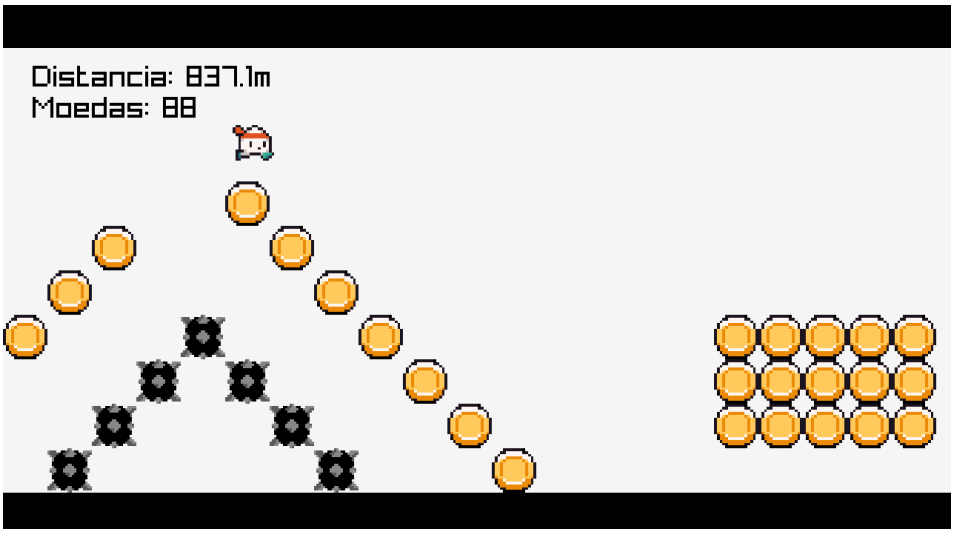

Figura 3: Representação gráfica de uma parte do mapa.

Requisitos Desejáveis (Extras)

Para os alunos motivados, as seguintes tarefas extras são propostas. Embora opcionais, essas tarefas podem melhorar a avaliação final do seu trabalho, caso não tenham conseguido implementar corretamente algum dos requisitos básicos.

- Implemente superpoderes, como, por exemplo, habilitar modo "imune" por alguns segundos – o que permitiria passar pelos espinhos e lasers sem danos.
- Implemente "*cheats*": por exemplo, ao escrever "MUAHAHA" no teclado (diretamente em tempo de jogo, sem abrir menu adicional), o jogador fica invencível naquela fase.
- Faça uma versão com representações visuais mais sofisticadas do jogo do que representação em modo texto. A sugestão do professor e monitores é usar a biblioteca *raylib* (`https://www.raylib.com/`). Você pode adicionar texturas para representar cada elemento do jogo.
- Implemente avisos sonoros quando o personagem receber dano, capturar moedas, passar de fase, perder ou ganhar o jogo, etc.
- Implemente diferentes tipos de coletáveis e diferentes benefícios para cada tipo.
- Implemente diferentes tipos de movimentação do personagem.
- Implemente diferentes tipos de obstáculos e/ou inimigos.

Entrega

O trabalho será entregue em **3 (três) etapas**:

1. Indicação dos grupos: até dia **22 de Novembro de 2024** às 23:59h pelo ambiente virtual Moodle. O trabalho deverá ser realizado em grupos de dois a três alunos. O grupo deve informar os nomes dos integrantes do grupo até o dia 22 de Novembro, através da plataforma Moodle. Os alunos que não tiverem definido grupos até esta data terão um grupo atribuído aleatoriamente.
2. Entrega do código: até dia **13 de Janeiro de 2025** às 23:59h pelo ambiente virtual Moodle. Até o dia 13 de Janeiro, o grupo deverá submeter via Moodle um arquivo `zip` cujo nome deve conter o(s) nome(s) do(s) aluno(s). O arquivo `zip` deve conter:
   - Um relatório contendo a descrição do trabalho realizado, incluindo a especificação de como os elementos do jogo foram representados, como foi implementada a interação dos componentes interativos, bem como as estruturas e funções utilizadas e uma explicação de como usar o programa. O relatório pode ser simples e objetivo, mas deve conter todas essas informações.
   - Os códigos fontes devidamente organizados e documentados (.c).
   - Os códigos executáveis já compilados e prontos para serem executados.
3. Apresentação: dias **14 de Janeiro de 2025** das 13:30h às 15:10h e das 15:30h às 17:10h – Turmas E e F, respectivamente.
   - O trabalho será obrigatoriamente apresentado durante as aulas dos dia 14 de Janeiro. Todos os membros do grupo devem saber responder perguntas sobre qualquer trecho do código e estes serão avaliados separadamente.

Avaliação

Os seguintes itens serão considerados na avaliação do trabalho: estruturação do código em módulos, documentação geral do código (comentários, indentação), "jogabilidade" do jogo e atendimento aos requisitos definidos. A divisão da pontuação é a seguinte:

1. Habilidade em estruturar programas pela decomposição da tarefa em subtarefas, utilizando subprogramação para implementá-las (2 pontos).

2. Organização e documentação de programas (indentação, utilização de nomes adequados para variáveis e constantes, abstração dos procedimentos para obter maior clareza, uso de comentários no código) (1 ponto).
3. Domínio na utilização de tipos de dados simples e estruturados (arranjos e estruturas) e passagem de parâmetros por cópia e referência (2 pontos).
4. Atendimento dos requisitos do enunciado do programa: menus, elementos gráficos esperados, interação, movimentação de personagens, funções dos menus, etc. (5 pontos).

Dicas

- Use uma IDE (como o Code::Blocks) para desenvolvimento.
- Alguns alunos relatam problemas com o uso da biblioteca `conio` com as últimas versões do Code::Blocks. Nestes casos, recomenda-se utilizar umas versão anterior o Code::Blocks (como a 17.12).
- Utilizar `structs` para representar os elementos dinâmicos do jogo.
- Alternativas para entrada de dados não bloqueantes (i.e., não esperam o usuário digitar a entrada e pressionar `ENTER`) podem ser encontradas na biblioteca `conio.h`. Veja os exemplos fornecidos na apresentação do enunciado, disponíveis no moodle. Ver funções como `kbhit` e `getch`.
- A biblioteca `windows.h` contém a função `SetConsoleCursorPosition` que pode ser usada para definir a posição do cursor do terminal.
- Além da `conio.h`, existem outras bibliotecas que podem ser usadas, como a `ncurses`, para Linux. Além disso, existem bibliotecas mais avançadas que permitem gerar apresentações visuais mais sofisticadas, como `Allegro`, `raylib` (preferível), `sdl2`, etc., para criação de interfaces gráficas.
- Para o jogo não executar muito rápido, pode-se utilizar a função `sleep`.
- Enquanto ainda não tivermos estudado como manipular arquivos, é possível inicialmente representar o conteúdo do mapa que deveria ser carregado diretamente em código (*hardcoded*), e utilizar essa informação para elaborar a representação visual e movimentação. Depois de estudar a manipulação de arquivos, essa informação do mapa pode ser carregada dos arquivos.
- É comum, à medida que formos estudando novos conteúdos, surgirem ideias de como utilizar esses conteúdos para melhorar o jogo. Isso acaba gerando modificações constantes no jogo. Isso é esperado.
- Verifique o material adicional oferecido no Moodle como suporte para este trabalho.
- Considere o uso e repositórios remotos de código (*Github*, *Gitlab*, *Bitbucket*) com controle e versão para desenvolvimento em equipe.
- Recorra ao monitor para sanar dúvidas gerais sobre instalação de bibliotecas, sobre dúvidas a respeito do uso das biblitotecas adicionais, uso de repositórios (*github*), uso de ferramentas e implementação de funcionalidades associadas ao jogo.

Observações

Este trabalho deve refletir a solução individual do grupo para o problema proposto. Casos de plágio (mais de 75% de similaridade – incluindo entregas de outros semestres) serão tratados com severidade e resultarão em anulação da nota do Trabalho Final (vide programa da disciplina). Para detectar e quantificar o plágio no código, será utilizado o *software* MOSS (`http://theory.stanford.edu/~aiken/moss/`).