Trabalho Final

Algoritmos e Programação – 2024/1

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, *Infmon*, é fortemente inspirado nas mecânicas dos primeiros jogos da franquia **Pokémon** (`https://www.nintendo.com/pt-pt/Jogos/Game-Boy/Pokemon-Red-Version-266109.html`). O jogo consiste em um *RPG de turno*. Ao iniciar uma batalha contra um inimigo, o jogador poderá escolher, em seu turno, um dentre um máximo de três ataques; enquanto o inimigo irá realizar seu ataque de forma aleatória quando for sua vez. Os turnos se intercalam até que um dos dois, jogador ou computador, seja derrotado. Além disso, ao andar pela grama, existirá uma chance de uma batalha contra um infmon começar, onde o jogador terá as opções de *fugir* da batalha, derrotar o inimigo para ganhar *XP* ou tentar *capturar* o infmon selvagem. O jogador pode ter um máximo de três infmons.

O jogo é separado em fases, sendo que cada fase possui um mapa distinto para ser explorado e um inimigo para derrotar, de modo que ao derrotar o inimigo, o jogador irá avançar para a próxima fase. As informações dos mapas estão contidas em arquivos texto "`mapaX.txt`", onde "`X`" indica o nível atual. No arquivo, há uma matriz de caracteres para indicar a estrutura do mapa (posição da grama, paredes e espaços vazios), e pontos iniciais do jogador e do inimigo. Ao iniciar um novo jogo, o jogador deve poder escolher um dentre três infmons para conduzir, sendo que cada um será de um *tipo* diferente. Quando em uma batalha, (i) caso o jogador derrote seu inimigo, seu infmon atual deve regenerar a vida até o maximo, e o jogador deve retornar para a exata posição do mapa onde estava quando entrou em batalha; (ii) caso o jogador seja derrotado, o jogador deve ver a tela de fim de jogo, podendo escolher entre começar um novo jogo, reiniciar a fase atual ou retomar a partida desde o último *checkpoint*. O jogo chega ao fim após o jogador passar por todos os mapas e derrotar todos o inimigos correspondentes.

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 – embora vocês sejam encorajados a implementá-los em modo gráfico (através a RayLib) ou outra biblioteca de preferência do grupo.
- 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 direita, ele deve imediatamente ir para a direita, se o jogador acionar alguma opção durante a batalha, imediatamente a ação relacionada deve ocorrer, etc.
- O mapa do jogo deverá ser carregado a partir de um arquivo texto nomeado "`mapa<número>.txt`" ("`mapa1.txt`", "`mapa2.txt`", "`mapa3.txt`", etc.), onde o valor numérico indica a fase correspondente ao mapa.
- Cada mapa deve ser estruturado como, no mínimo, uma matriz de caracteres de 30 linhas por 60 colunas. O mapa deve ser fechado (isto é, cercado de paredes).

- O tamanho da janela do jogo é 600 (altura) $\times$ 1200 (largura), portanto com tamanho mínimo de cada bloco de 20 *pixels*. Sinta-se livre para aumentar esse tamanho de janela se desejar ou for necessário.
- Os itens do jogo devem ser representados no arquivo do mapa com os seguintes caracteres:

Caractere	Significado
J	Posição inicial do personagem
E	Posição inicial do inimigo
G	Grama
W	Parede
Espaço em branco	Área de trânsito (posição vazia)

- O programa deve ser capaz de funcionar com qualquer número de fases (até o máximo de 99). Por exemplo, se forem incluídos 99 arquivos de mapa disponíveis para o jogo (seguindo a ordem numérica), seu programa deve ser capaz de possibilitar que o jogador avance pelas 99 fases. O jogo deve ser entregue com pelo menos 4 fases.
- O mapa de cada fase terá no mínimo 1 inimigo e no mínimo 25% da área de transito coberta por grama (o número pode variar conforme a fase).
- Ao andar pela grama, existe uma chance de uma batalha começar. O infmon selvagem deve ter tipo e nível aleatórios. É possível fugir das batalhas contra infmons selvagens, mas não contra inimigos.
- Uma batalha contra um infmon selvagem só deve ocorrer quando o jogador estiver andando pelo grama, ou seja, não deve ocorrer enquanto o jogador estiver parado na grama ou estiver em uma área de trânsito.
- A interação do jogador com o jogo, enquanto o jogador está no mapa, se dá pelas seguintes teclas:

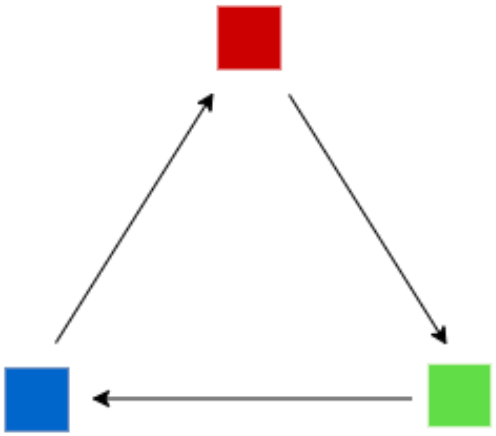
Tecla	Significado
TAB	Pausa o jogo e abre o menu / fecha o menu
Teclas (A/a, D/d, S/s, W/w)	Move o jogador uma posição na direção indicada
Setas ($\leftarrow$, $\rightarrow$, $\downarrow$ e $\uparrow$)	Move o jogador uma posição na direção indicada

- Quando em uma batalha, a interação do jogador com o jogo acontece com base no menu principal de batalha, conforme as seguintes teclas:

Tecla	Significado
A	Abre o **submenu de ataque**, mostrando todos os ataques do infmon atual
C	Tenta capturar o infmon inimigo
T	Abre o **submenu de troca**, mostrando todos infmons que o jogador já capturou
F	Tenta fugir da batalha atual

- O **submenu de ataque** deve monstrar todos os ataques do infmon atual e a opção de retornar ao menu principal da batalha, cada infmon terá entre um e três ataques. Ao escolher um ataque, o dano deve ser calculado e inflingido ao inimigo. Caso o inimigo não seja derrotado, ele irá realizar seu turno, atacando o jogador com um ataque aleatório. Após isso, o jogador irá retornar ao menu principal de batalha para seu turno.
- O **submenu de troca** deve mostrar todos os infmons do jogador, incluindo o que esta atualmente em batalha, além de mostrar a opção de retornar ao menu principal de batalha. O jogador terá um máximo de três infmons, e, ao escolher um deles no submenu de troca, o infmon escolhido se tornara o infmon principal até o final da batalha ou até que ele seja trocado novamente. Depois de trocar de infmon, o jogador fica um turno sem atacar, ou seja, logo após o jogador trocar de infmon, o inimigo irá realizar seu turno, atacando o jogador com um ataque aleatório. Após isso, o jogador irá retornar ao menu principal de batalha para seu turno.
- Quando o jogador estiver lutando contra infmons selvagens, encontrados aleatóriamente ao andar pela grama, ele poderá retornar ao mapa a qualquer momento acionando a opção de *fugir (F)* no menu principal da batalha. Porém, caso o jogador esteja batalhando contra um inimigo, ele não poderá fugir, só podendo sair da batalha ao derrotar o inimigo ou ser derrotado.

- Quando o jogador estiver lutando contra infmons selvagens, encontrados aleatóriamente ao andar pela grama, ele poderá tentar capturar o infmon a qualquer momento da batalha ao acionar a opção de *captura (C)* no menu principal da batalha e, com base na vida atual do inimigo, o jogador terá uma chance de conseguir ou não capturar. Porém caso o jogador esteja batalhando contra um inimigo, ele não poderá capturar o infmon de seu adversário.
- Ao final da batalha, caso o inimigo sejá derrotado, todos os infmons do jogador devem ter suas vidas regeneradas para o máximo.
- Cada infmon deve ter, no mínimo, os seguintes atributos: ***Tipo, Vida, Ataque, Defesa, XP*** e ***Nível***. Esses atributos deverão ser essenciais para a evolução dos infmons e para as batalhas. Atributos adicionais podem ser implementados para enriquecer o jogo.
- Devem existir no mínimo três tipos de infmons, e cada infmon deve ser de um tipo. Cada um dos tipos deve ter vantagem contra pelo menus um tipo e desvantagem contra pelo menos tipo. No exemplo em que os tipos são as cores *Vermelho*, *Azul* e *Verde*, podemos ter as vantagens e desvantagens como na imagem a seguir, onde as setas apontam as vantagens:



- As vantagens e desvantagens devem, respectivamente, aumentar e diminuir o dano causado ou recebido por ataques durante os combates.
- Os atributos de Vida, Ataque e Defesa devem ter relação com o nível do infmon, de forma que sempre que o infmon evoluir de nível, seus atributos devem aumentar.
- Ao derrotar um infmon (pertencente a um inimigo ou selvagem), o infmon atual do jogador deve receber um quantia de XP equivalente ao nível do inimigo. Caso a quantidade de XP do infmon atual do jogador for maior que um número X com base no nível atual, o infmon deve subir de nível. Quanto maior o nível do infmon, mais XP ele precisará para subir de nível.
- Cada mapa deverá ter apenas uma posição inicial do personagem e uma posição para cada inimigo (com um mínimo de um inimigo por fase). Ao derrotar os inimigos, o jogador deve ser levado direto para o início da próxima fase mantendo os infmons atuais e seus atributos.
- Quando em uma batalha, caso a vida do infmon atual se torne menor ou igual a 0, ele será ***derrotado***. Caso o jogador tenha outros infmons, o jogo deve trocar o infmon atual que foi derrotado por outro ainda que ainda não tenha sido derrotado. Caso todos infmons do jogador sejam derrotados, deve ser mostrada a tela de *Fim de Jogo* com as opções de reiniciar da fase, continuar do último *checkpoint* ou voltar ao menu inicial.

- Se o jogador entrar em contato com um inimigo, uma batalha deve iniciar. Todo inimigo deve ter, no mínimo, dois infmons, e, assim como o jogador, caso um infmon seja derrotado, outro deve assumir seu lugar. Isso ocorre até que todos infmons do inimigo (ou do jogador) sejam derrotados e a batalha acabe.
- Caso o jogador complete a última fase, deve ser renderizada uma tela de fim de jogo que o parabenize e com as opções de voltar ao menu ou sair do jogo.
- O menu de início deve ser navegável com o pressionar de teclas e possuir as seguintes opções:

Tecla	Significado
N	Novo Jogo: quando o jogador seleciona esta opção, um novo jogo é iniciado. Todas as configurações do jogo atual são reinicializadas.
C	Carregar jogo: quando o jogador seleciona esta opção, um jogo previamente salvo em um arquivo binário deve ser carregado. Assuma que existe apenas no máximo um jogo salvo e que pode ser carregado.
Q	Sair do jogo: quando o jogador seleciona esta opção, o jogo é encerrado, sem salvar.

- O menu de *pause* (exibido sobre a tela do jogo) deve ser navegável com o pressionar de teclas e possuir as seguintes opções:

Tecla	Significado
C	Continuar: quando o jogador seleciona esta opção, deve-se retomar o do jogo, que continua do ponto em que parou quando o jogador acessou o menu
L	Carregar jogo: quando o jogador seleciona esta opção, um jogo previamente salvo em um arquivo binário deve ser carregado. Assuma que existe apenas no máximo um jogo salvo e que pode ser carregado.
S	Salvar jogo/criar *checkpoint*: quando o jogador seleciona esta opção, deve-se salvar todas as informações pertinentes do jogo em um arquivo binário, de modo que ele possa ser carregado e continuado do ponto em que foi salvo. Essas informações incluem posições do personagem e dos inimigos, número de vidas, fase atual, etc.
B	Voltar ao menu: quando o jogador seleciona esta opção, o jogo volta ao menu inicial.
Q	Sair: Fecha o jogo sem salvar.

- Você pode implementar outras formas além do que está acima para navegar pelo menu, como pelo uso do mouse ou setas do teclado.
- O ato de "pausar" o jogo deve travar totalmente a lógica do jogo e permitir a interação do jogador com o menu. Ou seja, enquanto estiver ativo, nenhuma outra lógica do jogo pode estar sendo executada, incluindo, mas não limitada ao movimento do jogador, ataques do inimigos, animações, etc.

Você deverá pesquisar a utilização de bibliotecas para criação de uma interface para o programa se desejar não utilizar a RayLib. A escolha da biblioteca e sua utilização ficarão a cargo dos grupos. Algumas outras 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.

A Figura 1 apresenta um mapa de exemplo em arquivo texto. Esse mesmo mapa pode ser exibido em um programa em modo texto (terminal) de diferentes formas (veja a Figura 2). A Figura 3 apresenta o mapa de exemplo sendo exibida em um programa que usa a biblioteca `raylib`.

```
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
W            W         GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGW
W      E                 GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGW
W            W             GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGW
WGGGGGGGGGGGGW                GGGGGGGGGGGGGGGGGGGGGGGGGGGGGW
WGGGGGGGGGGGGW                  GGGGGGGGGGGGGGGGGGGGGGGGGGGW
WGGGGGGGGGGGGW                    GGGGGGGGGGGGGGGGGGGGGGGGGW
WGGGGGGGGGGGGW                       GGGGGGGGGGGGGGGGGGGGGGW
WGGGGGGGGGGGGW                          GGGGGGGGGGGGGGGGGGGW
WGGGGGWWWWWWWW                               GGGGGGGGGGGGGGW
W                                                GGGGGGGGGGW
W                                                   GGGGGGGW
W                                                     GGGGGW
W                                                       GGGW
WGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG GGGGGGGW
WGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG GGGGGGGW
WGGGGGGGGGGGGWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW WWWWWWWW
WGGGGGGGGGGGG                                              W
WGGGGGGGGGGGG                                              W
WGGGGGGGGGGGG                                              W
WGGGGGGGGGGGG                             WWWWWWWWWWWWWWWWWW
WGGGGGGGGGGGGW                            WGGGGGGGGGGGG   GW
WGGGGGGGGGGGGW                            WGGGGGGGGGGGGGGGGW
WGGGGGGGGGGGGW                             GGGGGGGGGGGGGGGGW
W            W                            GGGGGGGGGGGGGGGGGW
W            WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
W                      GGGGGGGGGGGGGGGGGGGG                W
W                      GGGGGGGGGGGGGGGGGGGG          J     W
W                      GGGGGGGGGGGGGGGGGGGG                W
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
```

Figura 1: Exemplo de mapa.

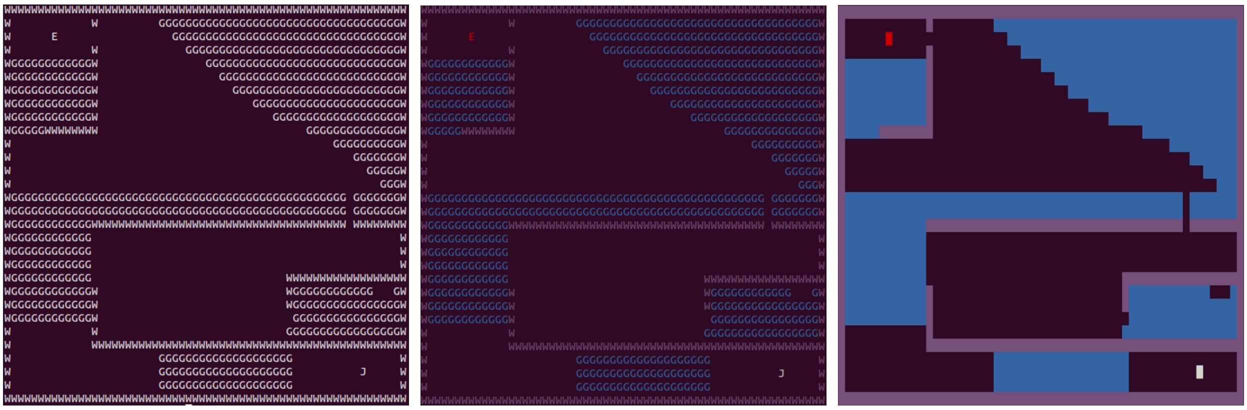


Figura 2: Representações em texto do mapa de exemplo.

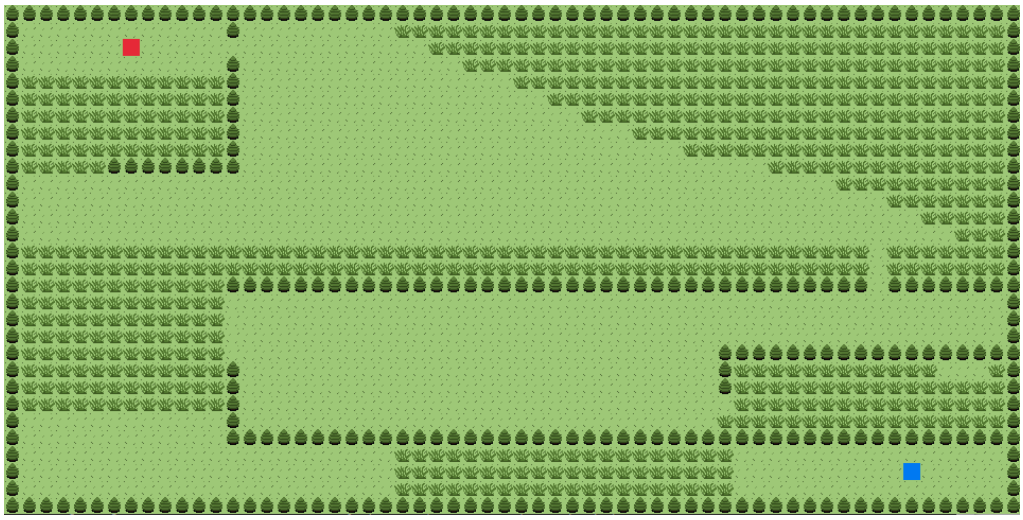


Figura 3: Representação gráfica do mapa de exemplo.

Requisitos Extas

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.

- Implementem diferentes tipos de infmons. Implementar mais de três tipos de infmons é um bônus – desde que os diferentes tipos ajam de maneira diferente e tenham vantagens e desvantagens próprias.
- Implementem "*cheats*": por exemplo, ao escrever "MUAHAHA" no teclado (diretamente em tempo de jogo, sem abrir menu adicional), o jogador fica invencível naquela fase ou o famoso Konami Code (`https://en.wikipedia.org/wiki/Konami_Code`).
- Emitam avisos sonoros ou visuais quando o jogador atacar, receber dano, entrar em uma batalha, subir de nível, etc.
- Implementem uma versão com representações visuais mais sofisticadas do jogo. Você pode adicionar texturas (e animações) para representar cada elemento do jogo, além de criar renderizações que, por exemplo, deem um zoom na posição do jogador e se adaptem ao tamanho da tela.
- Criem um tutorial para seu jogo explicando as mecânicas e habilidades que o jogador possui.
- Criem evoluções para os infmons ao atingirem determinados níveis.
- Façam com que o programa gerencie um arquivo binário "`highscores.bin`". Esse arquivo deve armazenar os cinco tempos mais rápidos de conclusão do jogo. Ao fim de uma partida, se o jogador atingiu um dos cinco melhores tempos, o programa deve requisitar o nome do jogador. Ao fim da partida, o programa sempre exibe o *ranking* dos cinco jogadores com menor tempo, do menor para o maior. Deve-se incluir uma opção no menu do jogo para visualizar o *ranking*.
- Combinem o aumento do número de inimigos, nível dos infmons e tamanho das fases para gerar fases com dificuldade crescente.
- Sejam criativos, conversem com o professor e monitor e implementem suas próprias ideias!

Entrega

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

1. Indicação dos grupos: até dia **12 de Julho de 2023** às 23:59h pelo ambiente virtual Moodle. O trabalho deverá ser realizado em grupos de dois ou três alunos. O grupo deve informar os nomes de seus integrantes até o dia 12 de Julho, 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 **22 de Agosto de 2024** às 23:59h pelo ambiente virtual Moodle. Até o dia 22 de Agosto, 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 *breve* 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 **23 de Agosto de 2024** das 13:30 às 15:10h.
   - O trabalho será obrigatoriamente apresentado durante a aula do dia 23 de Agosto. Todos os membros do grupo devem saber responder perguntas sobre qualquer trecho do código e estes serão avaliados separadamente.
   - Até o dia 9 de Agosto, os alunos deverão fazer um breve relato a respeito do andamento do trabalho. Esse relato deve ser feito através de um vídeo subido no Youtube (como *unlisted*). O *link* deve ser informado em fórum específico. É esperado que, neste momento, os alunos demonstrem pelo menos: a manipulação das informações do mapa e sua exibição e a movimentação dos personagens. Caso o grupo não apresentar este mínimo neste momento, todos os integrantes perdem 5% da nota final do trabalho.

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 (e abusar de) `structs` para representar os elementos dinâmicos do jogo. Lembre-se que você pode colocar uma struct dentro da outra, como por exemplo, uma struct *Posição* com x e y dentro de uma struct *Jogador*.
- Se antecipe e vá desenvolvendo o jogo conforme o tempo passa até a data de entrega. É útil ir implementando item por item e checando se o que está sendo desenvolvido está correto. Por exemplo, primeiramente faça a renderização do mapa, depois a implementação do jogador e seu movimento, depois os monstros e assim por diante. Modularize seu jogo para evitar propagação de erros!

- 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` ou a `SetTargetFPS` da RayLib.
- 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.
- Separe a lógica do jogo com a renderização na tela. Uma dica é sempre primeiro atualizar o estado das variáveis a partir das entradas do usuário e só após isso desenhar o que for necessário.
- 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.
- Jogue e analise os exemplos de jogos disponibilizados no *Moodle*, mas, lembre-se, você é sempre encorajado a criar soluções e apresentações novas.
- Lembre-se: No exemplo de implementação do *Moodle*, nem tudo que está implementado é requisito mínimo. Refira-se a este documento para saber o que é necessário fazer e tire qualquer dúvida com o professor ou monitor. Contudo, adições ao que estão aqui são sempre bem vindas.

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/).