INF01202 – ALGORITMOS E PROGRAMAÇÃO – 2019-2

TRABALHO PRÁTICO

Objetivo:

Exercitar as habilidades e conceitos de programação desenvolvidos ao longo da disciplina pela implementação de uma aplicação em C, desenvolvida por um grupo de 2 alunos da mesma turma ou de turmas diferentes. A coordenação de um trabalho em equipe faz parte da habilidade de programar que será demonstrada neste trabalho, portanto não serão aceitos trabalhos individuais.

O programa deve ser estruturado de forma a receber um conjunto de entradas (no início da execução ou durante o uso do programa), cuja consistência deve ser verificada, processá-lo, e fornecer uma ou mais saídas.

Produto do trabalho e datas de entrega:

O trabalho será entregue em 3 etapas:

a) Relatório de Andamento: dias 31 de Outubro

Apresentação na aula prática do código do trabalho onde será formalizada a dupla, e mostrada a tela de jogo com o movimento dos personagens funcionando e as estruturas de dados necessárias.

b) Entrega do código: 29 de Novembro (Sexta-feira) até as 9 horas pelo Moodle.

Upload no Moodle em tarefa própria de um ÚNICO arquivo compactado cujo nome do arquivo é o nome dos alunos contendo:

- Programa executável: o programa deve rodar nas máquinas do laboratório das aulas práticas. Verifique se executará sem exceções naquele local antes de entregar.
- Código documentado (arquivos .c, .cpp). Inclua o nome dos autores no cabeçalho do programa.
- Bibliotecas adicionais às que estão disponíveis no laboratório, exceto as conio e curses.

Se houver exceções aos requisitos listados neste enunciado, descreva como comentário no início do arquivo fonte.

Você pode fazer upload de diferentes versões e ir aperfeiçoando o programa. Faça o upload assim que tiver uma versão executável, de modo a garantir a entrega e precaver-se de problemas com servidor, redes, internet, etc.

c) Apresentação: 29 de Novembro

Programa será apresentado no dia 29 de Novembro. O arquivo a ser apresentado será aquele carregado no Moodle. Nenhuma alteração será permitida. **Ambos** alunos devem dominar o código para explicá-lo. A ausência de um dos alunos na apresentação acarretará decréscimo da nota para aquele.

Avaliação:

O programa deve atender todos os requisitos listados neste enunciado, não deve apresentar erros de compilação e rodar normalmente, pontos serão reduzidos caso contrário.

A aplicação desenvolvida deverá demonstrar os seguintes conteúdos que serão avaliados:

1. (2 pontos) Habilidade em estruturar programas pela decomposição da tarefa em subtarefas, utilizando subprogramação para implementá-las.
2. (2 pontos) Documentação de programas (indentação, utilização de nomes de variáveis, abstração dos procedimentos para obter maior clareza, uso de comentários no código).
3. (2 pontos) Domínio na utilização de tipos de dados simples e estruturados (arranjos, estruturas) e passagem de parâmetros. Uso exclusivo de variáveis locais.
4. (1 ponto) Formatação e controle de entrada e saída, com construção de interfaces que orientem corretamente o usuário sem instruções ou intervenção adicional do programador.
5. (1 ponto) Utilização de arquivos binários e de texto.
6. (2 pontos) Atendimento aos requisitos do enunciado do programa: modelo de estrutura de dados, de interação e de relatórios, ordenação dos dados, opções do programa, etc..

A nota do trabalho prático corresponderá a 10% da nota final. A apresentação do trabalho prático, mesmo que rodando parcialmente, é pré-requisito para realizar a recuperação.

Contextualização:

Deverá ser desenvolvido um jogo que consiste em controlar um jogador que pode mover-se de forma vertical ou horizontal por uma área de jogo implementada em modo texto (25x80 caracteres). O Jogador pode se deslocar nas direções vertical e horizontal em todo o espaço de jogo controlado pelas setas do teclado auxiliar. O objetivo do jogo é derrotar os inimigos que surgem em cada sala, acumulando pontos e passando para as salas seguintes. As salas podem se suceder indefinidamente e os obstáculos e inimigos são gerados aleatoriamente e de forma crescente. O jogo termina quando é atingida uma pontuação limite, a ser definida pelo programador ou com a morte do jogador.

O jogo aqui proposto é uma versão modificada de *The Legend of Zelda* do NES[https://www.youtube.com/watch?v=RQ6hgzzk_o8].

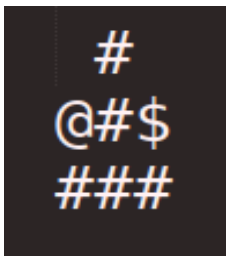
O jogo implementa o personagem do jogador e, no mínimo, 2 tipos de inimigos com diferentes comportamentos. Cada personagem é desenhado na área de jogo em uma área de 3x3 caracteres.

Jogador Link. O personagem possui as seguintes informações:

-Vida (int)

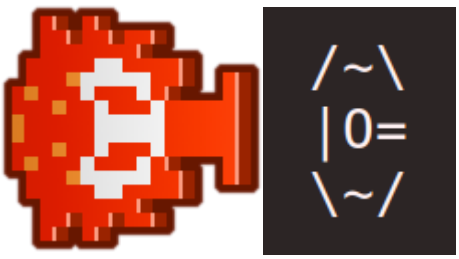
-Nível (int)
-Coordenada na área de jogo x (int)
-Coordenada na área de jogo y (int)
-Pontuação atual (int)
- Chave (boolean) valor que determina troca de sala
-Velocidade (int) Quanto maior o número, maior a velocidade. A velocidade do jogador cresce a cada mudança de sala.

O jogador não tem limite quanto ao número de tiros que pode dar, mas tem um limite de tempo X tiros a ser definido pelo programador, que impede que o usuário fique atirando o tempo todo do jogo. O Jogador perde uma vida quando é atingido pelos inimigos e morre quando não tem mais nenhuma vida.



Inimigo 1 Ghini (Fantasma): Movimenta-se em todas as direções de maneira automática e persegue o jogador pelo mapa, não importando a distância. O Ghini ataca o jogador diretamente, se atingir o jogador, este perde uma vida.
Quando destruído pelo jogador, este acumula 2 pontos, mas o Ghini deixa uma lápide de tamanho 2x2 na sala. A lapide é associada a sala atual e desaparece ao trocar de sala.

Inimigo 2 Octorok: Movimenta-se autonomamente, mas apenas horizontalmente de um lado ou outro da área de jogo. O Octorok ataca o personagem dando tiros somente na direção horizontal, no sentido do lado onde está o "=" , porém com intervalos de tempo entre tiros. Quando destruído pelo jogador, este acumula 1 ponto e deixa uma lapide de tamanho 2x2 na sala. A lapide é associada a sala atual e desaparece ao trocar de sala.



O número de inimigos da primeira sala é constante, e incrementa a cada nova sala. Cada inimigo é inicializado aleatoriamente com um bônus para o jogador que os destruir. Existem 3 opções de bônus:
- 1 vida;
- 1 chave para a próxima sala (uma única chave por sala);
- nenhum bônus.

Interface do jogo:

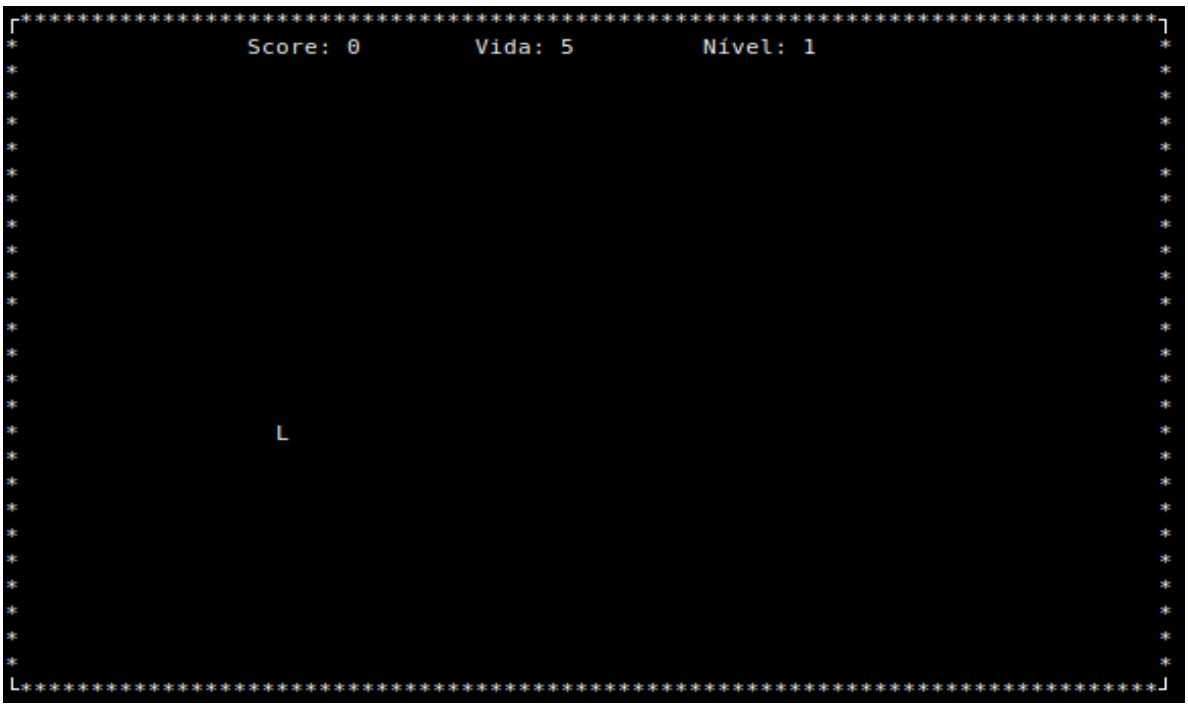
A implementação do espaço do jogo pode ser feita na janela padrão de texto do Code::Blocks (25 linhas por 80 colunas). No topo da área de jogo é apresentado o status do jogo: pontuação, número de vidas, nível ou sala atual

● **Menu principal:** A tela inicial do programa deve apresentar ao usuário um menu com as seguintes opções:

a) Novo Jogo: Inicia um novo jogo.
b) Carregar Jogo: Inicia o jogo a partir de um arquivo binário indicado pelo usuário. A qualquer momento durante o jogo, a tecla 'S' deve permitir ao usuário salvar o estado de jogo.
c) Recordes: Apresenta uma lista com as 10 maiores pontuações obtidas, em ordem decrescente. Quando o jogador termina uma partida, sua pontuação deve ser comparada com essa lista e, caso a pontuação seja alta o suficiente, inserida na posição apropriada. O jogo deve pedir um nome de 3 letras ao jogador para associar à pontuação. As pontuações devem ser salvas em um arquivo de texto e carregadas na inicialização do programa.
d) Créditos: Deve apresentar o nome dos alunos.
e) Sair: Encerra o programa.

● **Área de jogo:** A área de jogo deve conter as informações atualizadas do jogador, incluindo, na primeira linha da janela, pelo menos o número de vidas, pontuação e nível, ou seja, o número da sala atual.

Exemplo de interface onde L representa o personagem. Os inimigos não estão representados.



Atenção: o espaço e elementos de jogo NÃO são implementados como uma matriz, mas um conjunto de estruturas. Faremos isso na aula respectiva.

Funcionamento do jogo:

O jogador começa com 3 vidas e com a pontuação zerada. A posição inicial do jogador é no canto inferior esquerdo (x=79,y=24) . Os inimigos são gerados randomicamente, porem nenhum inimigo pode ser gerado a uma distância de menos que 8 coordenadas do jogador. Os inimigos se movimentam e atiram automática e aleatoriamente, de acordo com a velocidade daquela sala.
O jogador morre se esbarrar em alguma lápide e ganha uma vida se destruir um inimigo que tem uma vida de bônus. O jogador é controlado pelo usuário e busca acumular pontos atacando os inimigos até achar a chave e trocar de sala.

Em qualquer momento do jogo, o usuário pode digitar ESC para sair. Neste caso, o programa pergunta se quer salvar o jogo. Se o usuário quiser salvar o jogo será salvo em um arquivo binário, com nome <NomeUsuario>.bin. Em qualquer situação, o ESC encerra o jogo em andamento e retorna ao menu inicial. O usuário pode então sair, ou recarregar o jogo.

FIM DE JOGO: Quando o jogador perder todas as suas vidas, o programa deve mostrar uma mensagem com a pontuação final e a comparar com a lista das maiores pontuações. Caso a pontuação esteja entre as 10 maiores, o jogo deve pedir um nome de 3 letras ao jogador e registrá-lo com a pontuação na posição adequada, descartando o registro da última posição (para manter a lista com tamanho 10). As pontuações devem ordenadas e salvas em um arquivo de texto e carregadas na inicialização do programa.

Controles:

Setas – O JOGADOR se move na direção indicada.

Espaço – Dispara um TIRO em linha reta na direção da “frente” do jogador, onde tem o $. Pra atirar para o outro lado, o jogador muda de direção.

Pontuação:

A pontuação total será a soma total de:

- **1 ponto** por Octorok abatido;
- **2 pontos** por Ghini abatido;
- **5 pontos** para cada troca de sala;

Dicas:

a) A animação do jogo é feita escrevendo o objeto com a cor de fundo na posição atual e escrevendo com a cor de frente na nova posição. A função de animação será implementada na aula prática.

b) O jogo deve ter uma dificuldade intermediária (nem muito fácil, nem muito difícil).

c) O *gameloop* deve ser implementado **sem** o uso de funções do tipo *clrscr()* (para evitar o efeito de tela piscando).

d) O gameloop de jogo encerra com a tecla ESC acionada pelo usuário ou morte do jogador.

e) Os elementos do jogo são representados em estruturas ou vetores de estruturas, onde dois campos inteiros guardam as coordenadas do elemento na área de jogo. Funções de colisão serão implementadas na aula prática.

/* Estrutura geral do programa */

Menu principal
    Novo jogo
    Carregar jogo
    Recordes
    Créditos
    Sai

Inicializa estrutura de dados do jogador e dos inimigos
Desenha cenário
Laço do jogo
    Laço da sala
        Gera inimigos aleatoriamente
        Move jogador
        Move inimigos
        Gera tiros aleatórios
        Testa colisões
    Até chave == 1
Até ESC ou VIDAS == 0
Se ESC
    Salva jogo
    Volta para menu inicial
Senao Se VIDAS == 0
    Calcula pontos do jogador
    Se pontuação está entre as 10 melhores
        Pede o nome do jogador
        Atualiza lista de recordes
        Salva arquivo de pontuações
    Volta para o menu inicial

Funções e bibliotecas auxiliares: Funções e bibliotecas do C++ podem ser utilizadas para fazer a interface somente. Todas as funções de operação do jogo e manipulação de arquivos devem ser codificadas em C puro.

Uma biblioteca bastante útil é a *conio2*, porém ela é exclusiva para Windows. Abaixo há alguns exemplos de funções úteis dessa e de outras bibliotecas. Para quem desejar implementar em Linux, recomenda-se a biblioteca *ncurses*.

Exemplos e sugestões de funções auxiliares:

```
/*  FUNCOES E DEFINICOES UTEIS DA CONIO2  */

/**   * Colors which you can use in your application.   */
typedef enum
{
   BLACK,                /**< black color */
   BLUE,          /**< blue color */
   GREEN,                /**< green color */
   CYAN,         /**< cyan color */
   RED,          /**< red color */
   MAGENTA,              /**< magenta color */
   BROWN,                /**< brown color */
   LIGHTGRAY,            /**< light gray color */
   DARKGRAY,             /**< dark gray color */
   LIGHTBLUE,            /**< light blue color */
   LIGHTGREEN,     /**< light green color */
   LIGHTCYAN,            /**< light cyan color */
   LIGHTRED,             /**< light red color */
   LIGHTMAGENTA,   /**< light magenta color */
   YELLOW,               /**< yellow color */
   WHITE         /**< white color */
} COLORS;

/* FUNCOES DE POSICIONAMENTO EM TELA E CORES */
void clrscr(); // limpa a tela
void gotoxy (int x, int y); // posiciona o cursor em (x,y)
void cputsxy (int x, int y, char* str);  //imprime str na posicao x, y
void putchxy (int x, int y, char ch);  //imprime char na posicao x, y
void textbackground (int cor); // altera cor de fundo ao escrever na tela
void textcolor (int cor); // altera cor dos caracteres ao escrever na tela

/*  FUNCOES E DEFINICOES UTEIS DA CONIO.H  */

char getch(); // devolve um caractere lido do teclado, sem eco na tela
int kbhit(); // devolve true se alguma tecla foi pressionada, sem eco na tela


 /*  FUNCOES E DEFINICOES UTEIS DA WINDOWS.H  */

void Sleep (int t); // paralisa o programa por t milissegundos
void GetKeyState (int tecla); // pode ser usada para ler as setas do teclado
Exemplo de uso (trecho):
        if (GetKeyState (VK_RIGHT) & 0x80) // se a tecla "seta para direita" está pressionada
```

(…)

/* FUNCOES E DEFINICOES UTEIS DA TIME.H */

clock();

Exemplo de uso (trecho):

```
double tempo;
clock_t inicio, fim;
inicio = clock(); // salva tempo ao iniciar
 (…) // trecho do programa
fim = clock(); // salva tempo ao terminar
tempo_total_segundos = (int) ((fim – inicio) / CLOCKS_PER_SEC); /* calcula o
```

número de segundos decorridos durante a execução do trecho do programa*/