INF01202 – ALGORITMOS E PROGRAMAÇÃO
Turmas G e H – 2018/2

Xerife

1. Objetivo do trabalho:

Exercitar as habilidades e os conceitos de programação desenvolvidos ao longo da disciplina com a implementação de uma aplicação em C, desenvolvida por um grupo de 2 alunos da mesma turma ou de turmas diferentes (G e H). A coordenação de um trabalho em equipe faz parte da habilidade de programar. 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-las, e fornecer uma ou mais saídas.

2. Produto do trabalho e datas de entrega:

O trabalho será entregue em 3 etapas:

a) Relatório de Andamento: ~~19 de outubro~~ **26 de outubro**

Apresentação na aula prática do código do trabalho onde será formalizada a dupla e mostrada a tela principal do jogo com a movimentação dos elementos do jogo funcionando.

b) Entrega do código até às 12 horas do dia 4 de dezembro pelo Moodle

Upload no Moodle em tarefa própria de um ÚNICO arquivo compactado entregue por um dos alunos do grupo, 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, .h). **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, descreve-as 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

Programa será apresentado no dia **4 de dezembro** na aula prática. O arquivo a ser apresentado será aquele carregado no Moodle sem alterações. Ambos alunos devem dominar o código para explicá-lo. A ausência de um dos alunos na apresentação acarretará decréscimo da nota para o aluno ausente.

3. Avaliação:

O programa deve atender todos os requisitos listados neste enunciado, não deve apresentar erros de compilação e deve 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 coerentes para 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 que sejam necessárias instruções ou intervenções adicionais 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 da disciplina. A apresentação do trabalho prático, mesmo que rodando parcialmente, é pré-requisito para realizar a recuperação.

4. Contextualização:

Deverá ser desenvolvido o jogo “Xerife”. O usuário controla um único jogador (o Xerife) que se movimenta em 4 direções no cenário do jogo correndo atrás de bandidos para prendê-los. O objetivo do jogo é prender todos os bandidos no menor tempo possível.

O cenário do jogo será implementado em modo texto. Inclui a área de movimento entre as coordenadas 2 e 23 (linha) e 2 e 78 (coluna) e a área de informações na linha 25 (veja exemplo na Figura 1). Uma moldura nas linhas 1 e 24 e nas colunas 1 e 80 circunda a área do jogo e limita o movimento dos elementos do jogo. As cores e a representação dos elementos do jogo são de escolha dos alunos, porém espera-se que estejam bem posicionadas, alinhadas e com cores de fácil visualização. **Evitem o uso da cor vermelha, porque os projetores não diferenciam esta cor.**

O jogo deve ter, no mínimo, dois níveis de dificuldade. Além das dificuldades descritas abaixo, outras podem ser criadas pelos alunos.

a) Elementos do jogo:

Xerife: Elemento único. Movimenta-se em 4 direções - para cima, para baixo, para a esquerda, para a direita - e pode prender bandidos. O jogador não pode sair da área estabelecida pelas paredes do cenário; ao chegar em qualquer dos limites, ele não se movimenta mais naquela direção.

Para prender um bandido, o Xerife precisa estar a duas ou menos coordenadas de distância dele e apertar ESPAÇO.

O jogador deve ser implementado como uma Struct, com todas as suas informações: nome, coordenadas, caractere e cor da representação e outras que o jogo exigir.

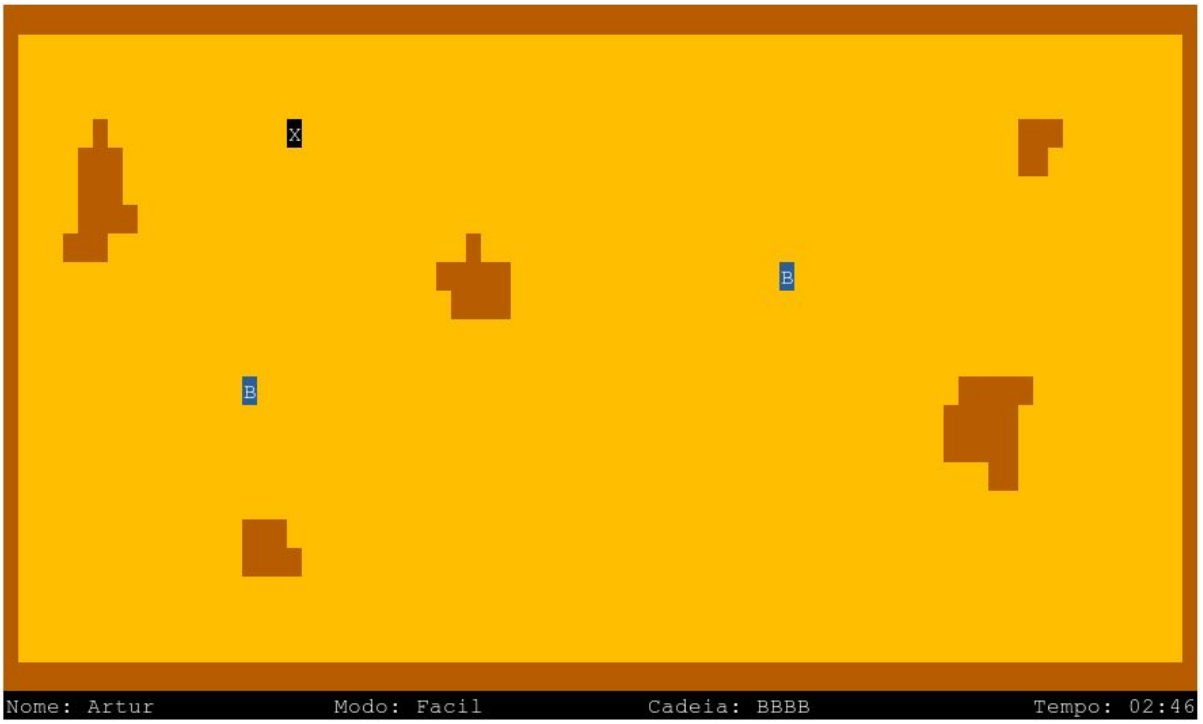

Figura 1: exemplo de cena de jogo.

Bandidos: Qualquer número, dependendo do nível de dificuldade (no mínimo 3 para o modo fácil, e, no mínimo, 7 para o modo difícil). Movimentam-se de forma autônoma (sem intervenção do jogador) e aleatória em 8 direções - para cima, para baixo, para a esquerda, para a direita, diagonal esquerda e acima, diagonal esquerda e abaixo, diagonal direita e acima e diagonal direita e abaixo, sempre em uma posição adjacente e livre. Quando um bandido é preso, ele desaparece do cenário do jogo e o contador da Cadeia na área de informação é incrementado.

Os bandidos devem ser implementados como um arranjo de Struct, onde cada posição do arranjo contém todas as informações de um bandido (posição, cor, velocidade, etc.). Cada bandido pode ter uma cor diferente de desenho.

Especiais: Existirão 5 poderes especiais espalhados pelo mapa. Estes deverão surgir em uma posição aleatória e livre no mapa, assim que o jogo for iniciado. Os especiais ficam invisíveis até o jogador ficar a uma casa de distância deles. Quando o especial aparece o jogador deverá ir para a mesma posição dele para poder coletá-lo, então o especial desaparece do cenário e aumenta a notificação de especiais na área de informação do jogo. Além dos descritos a seguir, outros especiais podem ser adicionados:

- Bota de couro: aumenta a velocidade do Xerife

- Saco de moedas: diminui a velocidade do Xerife
- Relógio de ouro: pausa a contagem do tempo por 5 segundos

Os especiais devem ser implementados como um arranjo de Struct, onde cada posição do arranjo contém todas as informações de cada especial.

Rochas: São distribuídas na área de jogo e atrapalham o movimento de todos. As rochas devem ser definidas por suas posições (x,y) no arquivo texto que representa o mapa. Devem existir pelo menos dois arquivos texto para representar o mapa (modos fácil e difícil), sendo que o mapa do modo difícil deve apresentar maior quantidade de rochas.

As rochas devem ser implementadas como um arranjo unidimensional de coordenadas (você não deve utilizar uma matriz bidimensional para isso), onde cada posição do arranjo representa uma rocha no cenário.

Área de informação: A linha na base do cenário mostra:

- o nome do jogador (instanciado no início do jogo ou carregado do arquivo);
- o nível do jogo (fácil ou difícil);
- a cadeia (contagem dos bandidos presos);
- o tempo de jogo, iniciando em zero e aumentando de acordo com o clock do computador no formato mm:ss (minutos e segundos).

b) Início do jogo:

A tela inicial mostra o nome dos alunos, nome da disciplina e semestre. Ao iniciar, o jogo pergunta o Nome do Jogador e exibe o Menu Principal, que deverá conter pelo menos os seguintes dados e opções:

1. Novo Jogo Modo fácil: carrega arquivo texto de configuração inicial, inicializa as estruturas de dados e mostra o cenário com elementos do jogo.
2. Novo Jogo Modo difícil: carrega arquivo texto de configuração avançada, inicializa as estruturas de dados e mostra o cenário com elementos do jogo. Neste modo de jogo, existirão mais bandidos que precisam ser presos pelo Xerife.
3. Retornar ao jogo: se o jogo foi interrompido com ESC, retorna ao mesmo jogo. Se não houver jogo interrompido, apresenta mensagem e retorna ao Menu Principal.
4. Salvar partida: Todas as variáveis do jogo são salvas em um arquivo binário com os elementos do jogo congelados como estão. Este arquivo deve ser capaz de armazenar as informações de jogo de vários Jogadores distintos, os quais devem ser identificados pelo Nome do Jogador.
5. Carregar partida: abre o arquivo binário de jogos, busca o Nome do Jogador e carrega as configurações de jogo que foram salvas, no ponto em que parou. Se não houver o jogador no arquivo, dá uma mensagem e volta ao Menu Principal.
6. Ajuda: deverá mostrar uma descrição de quais controles deverão ser utilizados para jogar.
7. Ranking: abre o arquivo binário de jogos, e mostra a lista com Nome do Jogador e seus escores (tempo de jogo), ordenada em ordem decrescente pelos escores. Oferece a opção de Resetar ranking, que zera todos os escores e nome de jogadores, ou Sair, que retorna ao Menu Principal.
8. Sair: fecha todos os arquivos, mostra uma mensagem e encerra o programa.

c) Fim do jogo:

Ocorre em 3 situações:

1. *Vitória:* Quando o Xerife prender todos os bandidos, o jogo será encerrado. Na situação de vitória, a pontuação (tempo) deve ser salva na lista de highscores em ordem decrescente. Uma mensagem de parabéns com a lista ordenada de jogadores e pontuações deve ser mostrada. Pode retornar ao Menu Principal.
2. *Derrota:* Se o jogador exceder o tempo máximo de jogo, o jogo será encerrado (o tempo máximo será de escolha dos alunos, dependendo principalmente do número de bandidos; cada modo de jogo deve ter um tempo máximo diferente). Uma mensagem de pesar com a lista ordenada de jogadores e suas pontuações é mostrada, mas seu tempo não é armazenado no ranking. As opções de jogar um novo jogo, carregar um jogo ou sair são oferecidas ao usuário.
3. *Interrupção:* quando o jogador pressionar a tecla ESC, o estado atual do jogo deve pausar e o sistema mostra o Menu Principal.

d) Pontuação do jogo:

A pontuação do jogo é o **tempo** que foi necessário para vencer a partida (ou seja, quanto menor o tempo, maior a pontuação).

e) Controles:

A movimentação do personagem deverá ser feita pelas flechas do teclado **(NÃO deverão ser utilizados os botões "W/A/S/D").** Estes controles deverão estar exibidos no Menu Controle.

f) Salvar estado do jogo:

O estado do jogo deverá ser salvo em um arquivo binário que poderá ser carregado no menu principal, em uma outra execução do jogo. Este arquivo deverá conter informações sobre o jogador (Nome do Jogador, coordenadas na tela, tempo de jogo, caractere e cor da representação), os bandidos (posição, caractere, cor), tempo, entre outras variáveis necessárias para a reinicialização correta da partida.

Dicas úteis:

- Evite utilizar o comando de limpar tela nas movimentações de jogo. O efeito de animação é produzido ao calcular a nova posição do elemento do jogo, plotar com a cor de fundo na posição atual e plotar com a cor do elemento na nova posição.
- Todos os movimentos devem testar a colisão com as bordas, rochas e elementos do jogo.
- Estude a biblioteca time.h para descobrir como manipular o tempo de uma aplicação.
- Implemente os requisitos mínimos do jogo aqui listados, ANTES de fazer os efeitos de animação e outros embelezamentos do sistema.

/* 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 tecla “seta para direita” está pressionada
- if (GetKeyState (VK_LEFT) & 0x80) // se tecla “seta para esquerda” está pressionada
- if (GetKeyState (VK_UP) & 0x80) // se tecla “seta para cima” está pressionada
- if (GetKeyState (VK_DOWN) & 0x80) // se tecla “seta para baixo” está pressionada
- if (GetKeyState (VK_SPACE) & 0x80) // se tecla “espaço” está pressionada
- if (GetKeyState (VK_ESCAPE) & 0x80) // se tecla “ESC” 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*/