INF1202 Trabalho Final de Algoritmos
Turmas C e D - 2021/1

Objetivo

Exercitar as habilidades e conceitos de programação desenvolvidos ao longo da disciplina através da implementação de uma aplicação em C, desenvolvida por um grupo de 2 alunos da mesma turma ou de turmas diferentes (C e D). A coordenação de um trabalho em equipe faz parte da habilidade de programar, então não serão aceitos trabalhos individuais.

O programa deve ser estruturado de forma modular em funções parametrizadas que realizam as entradas, processamento e geram as saídas da aplicação proposta. É aconselhado a criação de bibliotecas para decompor o programa, mas não obrigatório.

Produto do trabalho e datas de entrega

O trabalho será entregue em 3 etapas:

a) Relatório de Andamento: dia 5 de Outubro (tarefa no Moodle)

Formalização da dupla que fará o trabalho e submissão do código inicial do trabalho no moodle. O código deve mostrar a área do jogo e o movimento controlado pelo usuário funcionando.

b) Entrega do código: dia 22/11 até as 10:00 horas pelo Moodle

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

1) Programa executável: o programa deve rodar em qualquer plataforma Windows ou Linux, não apenas na máquina do aluno. Verifique se executará sem exceções antes de entregar. Cuide para que o programa não precise de bibliotecas que não foram entregues.
2) Código documentado (arquivos .c ou .cpp). Inclua o nome dos autores no cabeçalho do programa. **Descreva no cabeçalho como comentário, os requisitos do enunciado que não puderam ser atendidos ou que não executam por erro, para ciência dos avaliadores.**
3) Bibliotecas adicionais às que estão disponíveis por padrão, exceto conio2 e ncurses. 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: dia 22/11

O programa será apresentado no dia 22/11 na aula teórica. O arquivo a ser apresentado será aquele carregado no Moodle. Nenhuma alteração será permitida. Ambos alunos devem dominar o código para explicá-lo e serem avaliados em relação ao domínio do código. 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 sem adaptações, não deve apresentar erros de compilação e executar sem erros. Pontos serão deduzidos 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 apresentação do trabalho prático, mesmo que rodando parcialmente, é pré-requisito para realizar a recuperação, caso o aluno não obtenha nota para aprovação sem recuperação.

CONTEXTUALIZAÇÃO

O jogo a ser implementado é uma variação do jogo **Atari FROGGER** [1], cuja tela na versão original vemos na Figura 1 e iremos adaptar na nossa aplicação.

Figura 1 – Interface original do jogo FROGGER no Atari.

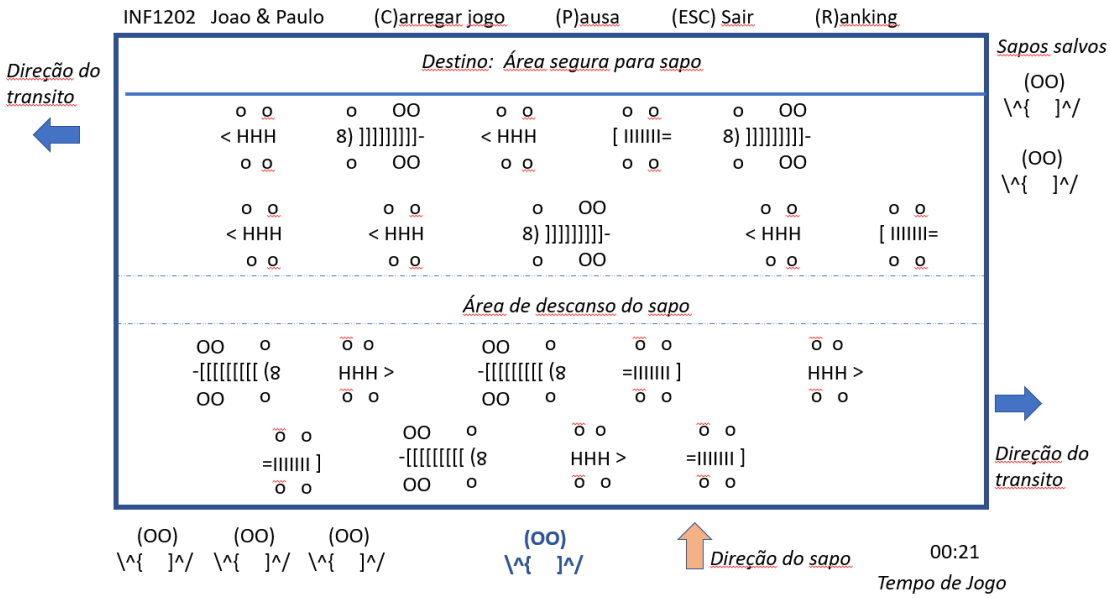
[1] Exemplo do jogo em https://www.youtube.com/watch?v=0lEyapqYGrU

Neste jogo, um jogador controla sapos que tentam atravessar uma avenida movimentada sem serem atropelados. O jogador ganha pontos cada vez que tem sucesso em fazer o sapo atravessar a rua. Cada vez que um sapo atravessa a rua ou morre, o jogador pode ter acesso ao controle de um novo sapo até o número de 6. O jogo inicia com 6 sapos e encerra quando não houver mais sapos para atravessar a rua. Ao terminar o jogo, a pontuação do jogador é produzida pela média de pontos ganhos pelo tempo de jogo. Ou seja, vence o jogador que atravessar mais sapos no menor tempo. Os requisitos do jogo estão numerados abaixo para facilitar a conferência e referência aos mesmos.

1. CENÁRIO: O cenário do jogo será implementado em **modo texto.**

O jogo é composto por uma avenida horizontal com 4 pistas. Nas duas pistas inferiores os veículos se deslocam para a direita e nas duas pistas superiores para a esquerda. Entre as duas pistas, tem um espaço sem veículos onde o sapo pode parar e descansar sem ser atropelado. Cada pista tem uma velocidade de deslocamento de veículos distinta e os espaços entre os veículos nas pistas também é distinto.

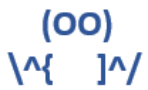
2. A Figura 2 mostra os principais elementos da interface. A área de jogo ocupa praticamente toda a tela sendo mantida apenas 10 colunas na lateral para mostrar os sapos salvos. Não será utilizada matriz para representar o espaço de jogo, mas arranjos e estruturas.



Elementos do jogo:

3. JOGADOR: estrutura de dados que contém o nome do jogador, número de sapos em espera (inicializado com a constante NUMERO de SAPOS do jogo, número de sapos salvos, tempo de jogo e ultimo score.
4. SAPO: estrutura de dados que contêm as coordenadas atuais do envelope do sapo, um campo para denotar “em espera”, “ativo”, “salvo” ou “morto” e um campo com a cor atual do sapo (valor da enumeração COLOR da biblioteca CONIO ou NCURSES). Outros campos podem ser acrescentados para controlar o jogo. O sapo pode andar em 4 direções: CIMA,

BAIXO, ESQ e DIR. O sapo sendo controlado pelo jogador deve ter cor distinta. . O desenho do sapo não precisa acompanhar a direção do movimento.



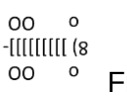
5. LISTA DE SAPOS : Arranjo de SAPOs, com tamanho definido pela constante NUMERO DE SAPOS.
6. CARRO ESPORTE: estrutura de dados que descreve o primeiro tipo de carro. Deve incluir no mínimo, as coordenadas do envelope do carro, a distância (em caracteres) que essa carro mantem com o da frente (entre 10 e 20), a cor do carro, a direção (esquerda ou direita). Você pode definir qualquer configuração de caracteres para o seu carro esporte, mas ele tem que ter, no máximo 6 caracteres e uma configuração de caracteres tem que ser a “frente” do carro. O desenho do carro aponta a frente na direção do movimento.

```
 o o
HHH >
 o o
```
Frente ->

7. CARRO SEDAN: estrutura de dados que descreve o segundo tipo de carro. Deve incluir no mínimo, as coordenadas do envelope do carro, a distância (em caracteres) que essa carro mantem com o da frente (entre 10 e 20), a cor do carro, a direção (esquerda ou direita) e velocidade em que o carro anda. Você pode definir qualquer configuração de caracteres para o seu carro sedan, mas ele deve ter numero de caracteres entre o carro esporte e o ônibus, e uma configuração de caracteres tem que ser a “frente” do carro. O desenho do carro aponta a frente na direção do movimento.

```
  o  o
=IIIIIII ]
  o  o
```
Frente ->

8. ONIBUS: estrutura de dados que descreve o ônibus. Deve incluir no mínimo, as coordenadas do envelope do ônibus, a cor do ônibus, a direção (esquerda ou direita) e velocidade em que o ônibus anda. Você pode definir qualquer configuração de caracteres para o seu ônibus, mas ele tem que ter o dobro do comprimento do carro e uma configuração de caracteres tem que ser a “frente” do ônibus. O desenho do ônibus aponta a frente na direção do movimento.


Frente ->

9. PISTAS: Arranjo de estruturas descreve a direção (esq e dir) e velocidade de cada pista. O numero de pistas é definido por uma constante de valor 4. Idealmente o

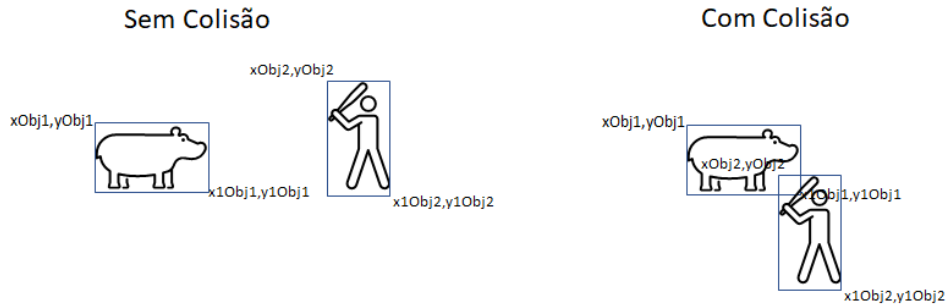
jogo poderia ser modificado alterando a direcao e velocidade das pistas apenas neste arranjo de estruturas.

10. TEMPÓ DE JOGO: A área de jogo tem um relógio no canto inferior direito que mostra em minutos e segundos o tempo de duração do jogo. Leia o clock do computador ao movimentar o primeiro sapo, para que seja possível calcular este valor.

FUNCIONAMENTO DO JOGO:

11. O jogo inicia diretamente na área de jogo, com 5 sapos na espera, um sapo ativo na posição de largada, nenhum sapo salvo e o tempo de jogo zerado.
12. Ao abrir o jogo, o usuário tem acesso a área de jogo e ao menu superior de opções com 5 opções possíveis:
13. Carregar Jogo: abre uma interface onde é solicitado o nome do jogador e um arquivo binário <nomedojogador>.bin é carregado posicionando os elementos do jogo onde estavam quando o jogo foi pausado.
14. Pausa: abre uma interface onde é solicitado o nome do jogador. Um arquivo binário <nomedojogador>.bin com todos os dados do jogo é salvo. No início do jogo, estes dados serão os mesmos de inicialização do jogo. Em qualquer momento do jogo, o jogador pode decidir PAUSAR o jogo, utilizando a tecla “P”. O jogo solicita o nome do jogador e salva o estado atual do jogo (todas as variáveis) em um arquivo binário <nome_do_jogador.bin>. Após pausar o jogador pode continuar jogando, movimentando o sapo ou sair com ESC. O jogador pode retornar ao mesmo momento do jogo quando reabrir o aplicativo, com a opção Carregar jogo.
15. ESC: dá uma mensagem e encerra o jogo. Se esta opção do menu for utilizada durante o jogo, deve salvar a pontuação do jogador no final do arquivo de ranking. Se o arquivo de ranking já tiver 10 linhas, procura o jogador com a menor pontuação insere os dados deste jogador na posição deste jogador no arquivo.
16. Ranking: carrega um arquivo texto onde cada linha contém <nome do jogador>.txt e sua pontuação. Ordena os dados carregados em ordem decrescente de pontuação. Abre uma interface e mostra os dados ordenados para o usuário.
17. Iniciar jogo. Qualquer tecla de setas do teclado auxiliar começa o jogo. O sapo se movimenta na direção da seta (exceto para baixo no primeiro movimento) e o tempo de jogo começa a ser contado.
18. MOVIMENTO DO SAPO: O jogador controla um sapo por vez e pode movimentar o sapo ativo em qualquer das 4 direções, buscando desviar do trânsito e atingir o outro lado da avenida. Ao ser salvo, o sapo é colocado na coluna de sapos salvos a direita da área de jogo e um novo sapo se posiciona na largada na parte inferior da área de jogo. Os sapos atropelados devem gerar um feedback para o jogador (desmontar, explodir, etc..) e depois desaparecer.
19. FUNÇÃO DE COLISAO: A colisão de objetos de jogo é tratada em computação gráfica através da noção de “envelope”. Um envelope de um elemento do jogo é o menor retângulo que envolve aquele elemento, alinhado com os eixos x e y da área de tela. Esse retângulo é representado pelas coordenadas superior esquerda e inferior direita do retângulo, como no exemplo abaixo, que mostra os envelopes do animal e do homem. No modelo mais simples de colisão, que implementaremos neste trabalho, qualquer superposição de envelopes é considerada uma colisão.

Esta função que recebe as coordenadas dos envelopes e verifica se acontece colisão será desenvolvida na aula prática de funções void com parâmetros da disciplina.



20. MOVIMENTO DOS VEÍCULOS: Os veículos têm comportamento autônomo (não são controlados pelo jogador) e são gerados initerruptamente em cada pista a partir das laterais da área de jogo. Uma vez associados a uma das pistas, assumem a direção e velocidade daquela pista. Os veículos se deslocam apenas para a esquerda nas pistas de cima e para a direita nas pistas de baixo, cada pista tem velocidade diferente. A distância entre cada veículo e o veículo da frente é controlada pelo parâmetro associado a cada veículo que muda com o decorrer do jogo.
21. O jogo encerra quando não houver mais sapos ou quando o jogador pressionar ESC.
22. AVANÇO DE FASE: se o jogador conseguir salvar todos os sapos, ele troca de fase. A fase 2 se caracteriza por :
    - 22.1 Tem mais sapos para serem salvos;
    - 22.2 Sapo ativo tem outra cor;
    - 22.3 A distância entre os carros é menor;
    - 19.4 A velocidade máxima das pistas pode ser maior.
23. Configure sua fase inicial de jogo para ser seja possível existir uma fase 2 ainda “jogável”. Configure suas funções e variáveis de modo que a fase 2 seja implementada com as mesmas funções com parâmetros ajustados.
24. Os requisitos não definidos nesta lista e aspectos de visualização dos elementos do jogo podem ser tratados como desejado pelos programadores.

Dicas

- Implementar uma função "gameloop()", que possui um laço que encerra ao pressionar a tecla ESC ou quando número de sapos == 0.
- Não utilizar funções como "clrscr()" dentro do laço (para evitar o efeito de tela piscando).
- Implemente uma função com parâmetro estrutura e cor para plotar cada um dos elementos do jogo. Para apagar o objeto da área de jogo, plote novamente o objeto passando como parâmetro a cor de fundo.
- Caso o movimento fique muito rápido, colocar um intervalo (*sleep*) entre cada iteração para evitar que o programa utilize 100% do *core* de CPU o tempo todo.
- Iniciar implementando o básico: cenário do jogo, movimentação do sapo e dos veículos e colisão do sapo. Depois, ir adicionando os demais requisitos.

```
                    /*   Programa principal  FROGGER
/* Main */
/* Essa estrutura de programa é uma sugestão que pode ser alterada e não um
requisito do jogo */
{
    Inicializa cenário do jogo
    Laço de jogo
    Contabiliza pontuação
    Carrega score
    Ordena e mostra score do jogo
    Encerra
}

/* Fluxo geral do jogo */

Inicializa estrutura de dados do jogo
Desenha cenário
Desenha sapo no centro da area inferior
Gera veículos e distancias, e distribui nas pistas aleatoriamente
Laço do jogo : repete
    Le teclas
    Move sapo
    Move veículos aternando entre as pistas para dar sensação de sincronicidade
    Testa colisão do sapo com lista de veículos
    Refresh do cenário
    Se PAUSA
        Salva jogo no arquivo binario
        Le entrada do jogador  (segue o jogo) ou ESC (sai do jogo)
Até ESC ou VIDAS == 0
Pede o nome do jogador e Calcula pontos
Carrega arquivo scores
Inclui jogador
Ordena e mostra os escores
Salva arquivo de pontuações
Sai do jogo

/*    Tipos definidos pelo programador   */
/* Essa estrutura de dados é uma sugestão que pode ser alterada e não um
requisito do jogo, mas os elementos e suas posições no jogo devem ser tratados
como estruturas*/


Tipo COORDENADA
X int
Y int
```

Tipo JOGADOR
    String nome do jogador
    int numero de sapos em espera
int numero de sapos salvos
time tempo de jogo
long int score

Tipo SAPO
    COORDENADA posição
    Int situação (pode ser espera, ativo, salvo ou morto)
    COLORS cor (ver enumeração abaixo)
    Int direção (pode ser cima, baixo , esq , dir)

Tipo VEICULO
    Tipo int (pode ser esporte, sedan ou onibus)
    COORDENADA posição
    Int distancia
    Int orientacao (pode ser esq ou dir)
    Int pista (pode ser 1 , 2 , 3 ou 4) // Para o caso de tratar aqui a pista do carro.
    COLORS cor (opcional)

VEICULO lista veículos [MAX_VEICULOS]

Tipo FASE
    Int numero de sapos
    float fator de velocidade
    float fator de distancia
    COLORS cor do sapo ativo

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 e windows.h* para Windows ou ncurses para o Linux. . Abaixo há alguns exemplos de funções úteis da conio2. Informações sobre as bibliotecas sao facilmente encontradas no StackOverflow ou diversos outros sites.

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*/
```