Turmas E e F
Trabalho Prático Final - Semestre 2019/2

MOTIVAÇÃO E OBJETIVOS

O objetivo deste trabalho é exercitar as habilidades e conceitos de programação desenvolvidos ao longo da disciplina pela implementação de um jogo em C, em grupos de 2 alunos.

Deverá ser feito um jogo inspirado no jogo **Agar.io**, em modo texto (ou gráfico, para tal será providenciada uma biblioteca de gráficos). O **Agar.io** é um jogo online em que cada usuário (jogador) controla uma peça (círculo) que se movimenta em um espaço sem limites, centralizando sempre na peça do jogador. Quando um círculo maior intersecta um menor, então o maior consome o menor, o que implica que o maior cresce e o menor desaparece. Uma implementação online deste jogo você encontra em https://agar.io/#ffa

Como o jogo não será desenvolvido online, ele terá um único jogador, cujo objetivo será sobreviver o maior tempo possível. A visão que o jogador terá do mundo será sempre centralizada na posição da sua peça, que poderá se movimentar nas direções norte, sul, leste e oeste (quando a peça se movimenta em uma direção, na prática o que se faz é mostrar o cenário movimentado uma posição naquela direção, pois a visão do jogo é centralizada na peça). De forma aleatória devem entrar (e sair) na visão do jogador outras peças criadas pelo jogo, de diferentes tamanhos, com diferentes direções de movimentos.

REQUISITOS DO PROGRAMA

As duplas terão liberdade para a implementação do programa. Entretanto, os seguintes requisitos devem ser respeitados:

- As peças devem possuir o mesmo tipo de forma geométrica. Para implementações em modo texto, as peças podem ser losangos ou quadrados (pois círculos são mais complicados de desenhar em modo texto), aqueles que usem a biblioteca gráfica que será disponibilizada, desenharão as peças como círculos (pois essa biblioteca tem funções específicas para desenhar círculos);
- Além da peça do jogador, cada peça que entre no jogo deverá ter dois tipos de classificação: pelo movimento e pelo tipo (ambas geradas aleatoriamente, assim como o tamanho da peça e o ponto de entrada no cenário);
- As classificação de movimento pode ser (cada uma das peças que se movimenta tem também uma velocidade gerada aleatoriamente):

  - **Estática** - peças que não se movimentam, elas só aparecem quando o jogador se movimenta;
  - **Aleatória** - peças que escolhem cada movimento de forma aleatória;
  - **Direcionadas** - peças que se movimentam em uma única direção (considerando as direções adicionais: sudeste, sudoeste, nordeste e noroeste);
  - **Perseguidoras** - peças que irão perseguir a peça do jogador.

- As classificação de tipo pode ser:
  - **Normal** - peças que não possuem nenhum atributo especial;
  - **Venenosa** - peças que se consumidas por uma maior a envenenam durante alguns segundos (proporcional ao tamanho dela). Uma peça que foi envenenada não pode consumir nenhuma menor enquanto estiver envenenada;
  - **Explosiva** - peças que ao colidir com outras explodem e explodem aquela com que colidiram.

- Deverá existir uma legenda para explicar
- O jogo deverá possuir uma tabela de pontuação que deverá ser carregada através de um arquivo binário chamado **ganhadores.bin**. Nesse arquivo serão salvos a pontuação e o nome do jogador em ordem decrescente, no máximo 10;
- O jogador poderá salvar o estado atual do jogo e mais tarde carregá-lo para voltar a jogar de onde parou. O estado deverá ser salvo em um arquivo binário chamado **meu_agario.bin**;
- Cada elemento deverá possuir sua própria struct, exemplo: peça, posição, cenário, jogador, etc.;
- As peças são movidas para baixo de modo autônomo dependendo da dificuldade do tabuleiro. Quanto mais difícil, mais rápido as peças se movem para baixo no tabuleiro;
- O jogo deve possuir uma tela que mostre um menu inicial com as seguintes opções: iniciar jogo, carregar jogo, exibir ganhadores e sair (após selecionar uma das opções e sair dela deve voltar à tela de menu inicial);
- O jogo deve reconhecer quando o jogo acabou, ou seja, quando a peça do jogador foi consumida ou explodida;
- O jogo deverá conter um contador de pontos, a ser exibido em uma linha logo abaixo do final do tabuleiro, ou em uma posição que achar mais conveniente. Lembre que a pontoação é dada pelo tempo que o jogador manteve sua peça viva (sem contar os possíveis intervalos em que o jogo esteve pausado);
- Ao final do jogo, o programa deverá verificar se a pontuação do jogador entra da tabela das *Ganhadores*. Caso afirmativo, o programa deve solicitar a leitura do nome do jogador e armazenar seu nome e pontuação no arquivo **ganhadores.bin** (possivelmente excluindo o score mais baixo até então, se a tabela já se encontrar com seu número máximo de entradas);

ENTREGA E AVALIAÇÃO

- O trabalho deverá ser feito em duplas. Informar os componentes da dupla até o dia primeiro de novembro.

- Até o dia seis de dezembro (ou treze de dezembro), cada a dupla deverá submeter via Moodle um arquivo zip cujo nome deve conter o(s) nome(s) do(s) aluno(s). O arquivo zip deve conter:
  - Uma descrição do trabalho realizado contendo a especificação completa das estruturas usadas e uma explicação de como usar o programa;
  - Os programas-fonte devidamente organizados e documentados (arquivos.c e arquivos.h);
  - Executável do programa.
- O trabalho será obrigatoriamente apresentado durante a aula prática do dia treze de dezembro (para os que realizaram o envio até o dia seis de dezembro) ou no dia dezessete de dezembro (para os que realizaram o envio até após o dia seis de dezembro). Ambos membros da dupla deverão saber responder perguntas sobre qualquer trecho do código.
- No dia da apresentação serão realizados diferentes testes de condições de jogo para testar o programa.
- Os seguintes itens serão considerados na avaliação do trabalho: estruturação do código em módulos, documentação geral do código (comentários, identação), jogabilidade e atendimento aos requisitos definidos.
- **Importante**: trabalhos copiados não serão considerados. Saibam que há ferramentas que possibilitam a detecção automática de plágio.

BIBLIOTECAS

A seguir são relacionadas algumas constantes, estruturas e funções por bibliotecas que serão de utilidade no desenvolvimento do projeto.

- **time.h**: (empregada para medir o tempo)
  - `clock_t`: tipo para representar número de *clocks* em dado momento (recomendado realizar *casting* para *int* ou *double* na hora de realizar os cálculos).
  - `CLOCKS_PER_SEC`: constante que indica quantos *clokc* acontecem em um segundo.
  - `clock()`: função que retorna a quantidade de *clocks* atuais.

  Exemplo:

  clock_t inicio, fim;

  inicio = clocks();

  //Algum código a ser executado

  fim = clocks();

  printf("Demorou %d segundos", (int) ((fim - inicio) / CLOCKS_PER_SEC));
- **conio.h**: (empregada para algumas operações de I/O)
  - `kbhit()`: função que retorna verdadeiro se alguma tecla foi apertada. Deve ser usada em conjunto com `getch()` para obter a tecla em questão.
  -

  Exemplo:

  if(kbhit()) {

  switch(getch()) {

  case 'a' ... 'z':

```
    case 'A' ... 'Z':
      printf("Apertou uma letra!");
      break;
    case 13:
      printf("Apertou o ENTER!");
      break;
    case 72:
      printf("Apertou o UP!");
      break;
}
```

- **windows.h**: (empregada para algumas operações de visualização)
  - `HANDLE`: tipo usado para controlar o *prompt* de comandos.
  - `GetStdHandle(int)`: função que permite obter o controlador do *prompt* de comandos.
  - `STD_OUTPUT_HANDLE`: constante que permite indicar que se deseja obter o controlador do *prompt* de comandos.
  - `COORD`: tipo que permite representar coordenadas no *prompt*.
  - `SetConsoleCursorPosition(HANDLE, COORD)`: função que coloca o cursor do *prompt* na coordenada indicada.
  - `SetConsoleTextAttribute(HANDLE, int)`: função que permite trocar atributos de texto do *prompt* antes de imprimir (ex. cor da fonte ou cor do fundo).
  - cores principais: (combinando-as com o operador | se podem obter outras cores)
    - `BACKGROUND_RED`, fundo vermelho
    - `BACKGROUND_GREEN`, fundo verde
    - `BACKGROUND_BLUE`, fundo azul
    - `FOREGROUND_RED`, texto vermelho
    - `FOREGROUND_GREEN`, texto verde
    - `FOREGROUND_BLUE`, texto azul

Exemplo:

```
HANDLE prompt = GetStdHandle(STD_OUTPUT_HANDLE);
COORD coordenada;
coordenada.X = 20;
coordenada.Y = 5;
//Posiciona o cursor do prompt na linha 5, coluna 20:
SetConsoleCursorPosition(prompt, coordenada);
//Troca a cor para escrever com letra vermelha
SetConsoleTextAttribute(prompt, FOREGROUND_RED);
printf("Texto vermelho");
coordenada.Y = 2;
//Posiciona o cursor do prompt na linha 2, coluna 20:
SetConsoleCursorPosition(prompt, coordenada);
//Troca a cor para escrever com letra amarela (se obtém juntando o vermelho e o verde)
```

SetConsoleTextAttribute(prompt, FOREGROUND_RED | FOREGROUND_GREEN);

printf("Texto amarelo");

coordenada.Y = 15;

//Posiciona o cursor do prompt na linha 15, coluna 20:

SetConsoleCursorPosition(prompt, coordenada);

//Troca a cor para escrever com letra branca (se obtém juntando o vermelho, o verde e o azul) e fundo roxo (se obtém juntando vermelho e azul)

SetConsoleTextAttribute(prompt, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE | BACKGROUND_RED | BACKGROUND_BLUE);

printf("Texto branco fundo roxo");

- **graphics.h**: biblioteca que permite um trabalho gráfico mais simples que *windows.h*, pois inclui funções prontas para desenhar figuras geométricas.
  - `initgraph(int*,int*,char*)`: função que inicia uma nova janela para dee har em modo gráfico.
  - `setcolor(int)`: função que permite definir o color da fonte.
  - `setbkcolor(int)`: função que permite definir a cor do fundo.
  - `rectangle(int, int, int, int)`: função que desenha um retângulo passando a coordenada esquerda superior e a direita inferior.
  - `circle(int, int, int)`: função que desenha um círculo a partir do centro e o raio.
  - `outtextxy(int, int, char*)`: função que permite imprimir um texto a partir de uma posição dada.
  - `outstreamxy(int, int)`: função que permite imprimir informação no *buffer* de saída da biblioteca a partir de uma posição dada.
  - `bgiout`: *buffer* de saída da biblioteca, é possível passar textos e números para imprimir.
  - `cleardevice()`: limpa a tela com a última cor de fundo atribuída.
  - Usando só as funções anteriores é possível desenvolver o projeto, contudo há funções para trocar a fonte usada, o padrão de desenho de fundo assim como para desenhar outras figuras geométricas que podem ser empregadas para melhorar o jogo.

Exemplo:

int gdriver = DETECT, gmode;

//Inicia o modo gráfico

initgraph(&gdriver, &gmode, "");

//Desenha um círculo azul na posição 100, 200 de raio 50

setcolor(BLUE);

circle(100, 200, 50);

//Imprime texto azul na posição 50, 0

outtextxy(50, 0, "Mensagem azul");

//Desenha um retângulo vermelho com coordenada esquerda superior 10, 20 e direita inferior 60, 100

setcolor(RED);

rectangle(10, 20, 60, 100);

setbkcolor(BLUE);

```
//Imprime texto vermelho com fundo azul na posição 250, 0
outtextxy(250, 0, “Mensagem vermelha com fundo azul”);
//Coloca no buffer de saída o texto: o produto de 37 por 13 eh:
bgiout « “O produto de ” « 37 « “ por ” « 13 « “ eh ” « 37 * 13;
//Imprime a informação no buffer de saída na posição 200, 200
outstreamxy(200, 200);
//Aguarda 20 segundos antes de fechar
delay(20000);
```