DOCUMENTAÇÃO TRABALHO PRÁTICO PDS I - 2020/1

“DATA INVADERS”

Aluno: Thiago Cleto Miarelli Piedade
Professor: Pedro Olmo Stancioli Vaz de Melo

1. Descrição

Data Invaders é um jogo com temática espacial inspirado no jogo de ATARI “Space Invaders”. Na versão desenvolvida, uma nave representa a Lei Geral de Proteção de Dados, ou LGPD, e luta contra “aliens” personificados por veículos espaciais personalizados com a identidade visual de grandes empresas que fazem uso de dados.

Na tela principal de jogo, o usuário deve utilizar as setas direcionais do teclado para direcionar a nave para esquerda e para direita. Para atirar, deve-se clicar na tecla espaço.

A cada vez que o usuário consegue destruir uma grade de aliens invasores, uma nova grade é formada com uma velocidade de deslocamento lateral 20% maior. Tal mecânica dá ao jogo uma característica de duração infinita, limitada apenas pela habilidade do jogador de se defender contra inimigos cada vez mais rápidos.

2. Menus

O jogo é constituído de três menus principais:

a) Splash Screen: essa tela abre o jogo e permite que o usuário escolha entre jogar o jogo ou ir para a loja de naves.
b) Loja de Naves: na loja de naves, o usuário pode escolher entre diversas naves com características e visuais diferentes. As naves podem ser mais rápidas e atirar mais vezes por segundo.
c) Fim de jogo: o usuário pode escolher jogar novamente, ir para a loja ou sair do jogo.

3. Structs

a) NAVE

A *struct* NAVE define a cor da nave, a ponta_x, referente ao ponto em que a nave deve atirar os projéteis, uma variável do tipo ALLEGRO_BITMAP que representa a imagem que a nave está usando no momento. Além disso, há uma *array* com as imagens sequenciais da animação sprite da nave.

b) ALIEN

A *struct* ALIEN define a cor do alien, junto à coordenada dos seus cantos superiores esquerdos. Além disso, há uma variável inteira com carácter booleano “exist”. Essa variável determina se o alien já foi eliminado do jogo ou não. Além disso, há um parâmetro contendo a imagem atual do alien e as opções a serem sorteadas de imagem.

c) TIRO

A *struct* TIRO define as características básicas de um projétil como o float x e y, referente ao seu centro, o raio e sua existência booleana, bem como o alien.

4. Main

A função *Main* pode ser dividida entre alguns blocos:

Line 145 to 132: Rotinas de Inicialização
Line 243 to 256: Abertura das fontes
Line 261 to 276: Cria as variáveis das structs como os aliens, nave e tiro.
Line 279 to 291: Abre as imagens estáticas do jogo.
Line 296 to 311: Carrega os sons e música do jogo e configura o mixer.
Line 317 to 256: Abertura das fontes
Line 324 : Início do Loop Principal de jogo
Line 328 to 345: Programa abre os arquivos de recorde e armazena o valor em uma variável. Caso a pontuação no jogo tenha sido maior que o recorde, substitui-se o valor no arquivo e na variável.

Line 348 to 357: Configuração do evento de fechar janela e indicador de clique do mouse.

Line 360 to 381: Loop da splash screen

Line 362 to 367: Carrega o background da splashscreen e atualiza a tela.

Line 343 to 356: Cria a condição para evento de teclado em caso do usuário clicar na letra ‘s’. Muda o modo de jogo para jogatina(g)

Line 376 to 378: Muda o modo de jogo para loja(p) em caso de clicar na tecla ‘p’.

***Line 385 to 432:* Loop de jogo**

Line 389 to 401: Em caso do evento ser uma mudança no timer, o programa deve converter o inteiro de pontuação em string, desenhar o background do jogo, desenhar a pontuação na lateral inferior esquerda, desenhar a nave, desenhar a matriz de aliens, atualizar a matriz de aliens conforme o tempo, atualizar a posição do tiro, caso exista, verificar se houve alguma colisao entre o tiro e a matriz de aliens, caso todos os aliens tenham sido destruidos, o jogo recomeça com a velocidade mais rápida. Caso o usuário tenha perdido, o modo de jogo é alterado para end (e) ou recorde(r) dependendo da sua pontuação.

Line 412 to 428: Estabelece o controle da nave. Caso o jogador aperte o botão de direita ou esquerda no teclado, a posição da nave é somada a sua velocidade. Caso o jogador aperte espaço, a função atirar( ) é executada.

Line 434 to 468: Loop da end screen

Line 437 to 444: Carrega o background da endscreen, imprime o número de pontos no topo, o recorde, as moedas e uma frase cômica aleatória sobre privacidade.

Line 454 to 458: Muda para o modo de jogo, redefinindo os valores das variáveis de jogo.

Line 464 to 466: Muda o modo de jogo para loja(p) em caso de clicar na tecla ‘p’.

Line 470 to 496: Loop da record screen

Line 475 to 479: Carrega o background da record screen, imprime o número de pontos no topo, o recorde, as moedas e uma frase cômica aleatória sobre privacidade.

Line 488 to 496 Muda para o modo de jogo, redefinindo os valores das variáveis de jogo.

Line 497 to 499: Muda o modo de jogo para loja(p) em caso de clicar na tecla 'p'.

Line 502 to 537: Loop da loja

Line 504 to 517: Se o evento for um timer, então atualize as informações de powerups, desenhe o background da loja, imprima a quantidade de moedas e complete a barra de powerups conforme a quantidade já comprada. Teste se o valor de moedas que o usuários possui é maior ou igual ao preço de um powerup. Caso negativo, então exiba um botão que informa que o usuário não tem dinheiro suficiente.

Line 509 to 516 Detecta o clique do usuário, de modo que se o clique estiver dentro da área do botão, então execute o método compraPowerup( );

Line 529 to 534: Muda para o modo de jogo, redefinindo os valores das variáveis de jogo.

Line 542 to 575: Rotinas de destruição

5. Funções

```
void drawSpace(ALLEGRO_BITMAP *background);
```

É responsável por carregar o fundo e os elementos estáticos da tela de jogo. Recebe como parâmetro a imagem de fundo.

```
void criaNave(NAVE *nave);
```

Recebe como parâmetro a struct NAVE, iniciando-a com os valores de posição inicial e carregando as imagens que compõem seu sprite na array nave_animation[].

```
void drawNave(NAVE *nave, int frame);
```

Faz uso da posição atribuída ao struct NAVE, e coloca-a no espaço. Com uso do inteiro frame, que deve receber o inteiro correspondente ao timer do jogo, a função troca de imagem na array 3 vezes por segundo, dando uma impressão de movimento.

```
int randInt(int min, int max);
```
Função genérica de sorteio que gera um número entre *min* e *max*.

```
void drawAlien(ALIEN *alien);
```
Imprime o bitmap do alien em questão.

```
void criaAlien(ALIEN *alien, float x, float y);
```
Inicia o alien com a posição x e y enviada pelos parâmetros da função, muda a existência do alien para 1 e carrega as diversas opções de skins para os aliens. Após isso, sorteia um número de 0 à quantidade de skins disponíveis e atribui uma skin ao alien. O número da skin é equivalente ao número de pontos que destruir aquele alien vale.

```
void criaMatrizAliens(int linhas, int colunas, ALIEN
aliens[linhas][colunas]);
```
Usada para iniciar e reiniciar o jogo, a função atribui aos struct ALIEN, os valores iniciais do jogo por meio da função CriaAlien( ), podendo recomeçar a jogatina.

```
int testaCanto(ALIEN *alien);
```
Testa se o alien enviado por parâmetro está com seu lado direito encostando na extremidade direita, ou com seu lado esquerdo encostando na extremidade esquerda da tela. A função retorna 0 caso o alien não esteja encostando e 1 caso este esteja encostando.

```
void BuildAlienGrid(int linhas, int colunas, ALIEN
alien[linhas][colunas], int seconds);
```
Cria uma matriz de aliens de n linhas e m colunas. Como fica dentro do loop de execução do jogo, a cada execução, a função soma o valor da velocidade à posição do alien, fazendo com que este se movimente horizontalmente. Além disso, a função faz uso do método *testaCanto( )*, que identifica se algum dos aliens que estão sendo exibidos atingiu alguma das extremidades da tela. Caso a afirmação seja positiva, a função inverte o valor da velocidade, multiplicando-a por -1 e soma um valor referente ao deslocamento vertical para baixo, contribuindo para uma maior dificuldade do jogo.

```
void updateTiro(TIRO *tiro);
```
Fica sendo executado constantemente no Loop. Se o tiro existe, ele é desenhado com a função drawTiro( ). A cada nova execução, ele subtrai a velocidade de sua posição, fazendo com que o tiro fique subindo.

```
void atirar(TIRO *tiro, NAVE *nave, ALLEGRO_SAMPLE *tiro_sound);
```
Testa se o tiro existe. Em caso positivo, ele não atira, uma vez que pode haver apenas um tiro no ar por vez. Caso o tiro não exista, ele reposiciona o tiro existente na ponta da nave e torna sua existência = 1.

```
void drawTiro(TIRO *tiro);
```
Desenha uma circunferência branca preenchida na posição indicada pelo objeto.

```
void criaTiro(TIRO *tiro);
```
Inicia os valores do tiro no começo do jogo.

```
void colisao(TIRO *tiro, int linhas, int colunas, ALIEN
alien[linhas][colunas], ALLEGRO_SAMPLE *hit_audio);
```
Testa em cada um dos aliens com estado de existência igual à 1, se a função bateu( ) retorna 1.

```
int bateu(ALIEN *alien, TIRO *tiro);
```
Testa se a posição x do tiro está entre as posições canto_x do alien e canto_x + ALIEN_W. Além disso, testa se a posição y do tiro está acima da altura da base do alien, mas abaixo do topo. Issa verificação dupla do y é importante para evitar que a movimentação posterior do alien em relação ao tiro não elimine o alien mesmo que ele não tenha sido atingido. Além das verificações de posição, a função também checa a existência do tiro e do alien. Caso todas as condições sejam verdadeiras, a função **retorna 1,** indicando uma colisão. Caso contrário, ela **retorna 0.**

```
int perdeu_nave(ALIEN *alien, NAVE *nave);
```
Testa se o alien bateu na nave. A função checando se a altura da nave é menor ou igual à do alien e se eles ocupam o mesmo espaço horizontalmente. Caso exista uma colisão, a função **retorna 1**. Caso contrário, ela **retorna 0.**

```
void perdeu(int linhas, int colunas, ALIEN alien[linhas][colunas],
NAVE *nave);
```
Testa se algum alien da matriz retorna 1 quando submetido a função perdeu_nave( ) ou se algum alien atingiu a parte inferior da tela. Caso alguma das condições seja verdadeira, a função muda o modo de jogo para ‘e’, ou seja, end screen.

```
void reinicia(int linhas, int colunas, ALIEN alien[linhas][colunas]);
```
Caso na tela de endgame, o jogador escolha jogar novamente, a função reposiciona a matriz de aliens no estado inicial, redefine a velocidade para padrão e atribui o valor 0 aos pontos.

```
void testRecord(FILE *file, int recorde, int pontos);
```
Testa se o valor da pontuação é maior que o recorde estabelecido. Caso positivo, o recorde é substituído pela pontuação atual.

```
void repopulate(int linhas, int colunas, ALIEN
alien[linhas][colunas]);
```
Testa se o valor da pontuação é maior que o recorde estabelecido. Caso positivo, o recorde é substituído pela pontuação atual.

```
void atualizaMoedas(FILE *moedas_file, int valor, char modo);
```
Quando o jogador perde, a função lê o arquivo de quantas moedas o jogador tem, soma o valor referente aos pontos obtidos no jogo, abre o arquivo em modo escrita e escreve o valor atualizado. A função atualizaMoedas é usada tanto para somar as moedas ganhas em uma partida quando para subtrair as moedas gastas em compras de powerup. Dessa forma, o primeiro argumento refere-se à variável cujo arquivo de moedas ficará armazenado, a segunda o valor a ser somado ou subtraído e a terceira o modo. Há dois modos de operação: 'e', de "earn" ou "ganhar", e "s" de "spend" ou "gastar".

```
void drawLoja(ALLEGRO_BITMAP *background);
```
Atualiza a quantidade de moedas, de modo que qualquer operação de compra exiba o saldo atualizado. Recebe um argumento de imagem que funciona como o seu background.

```
void getPowerupData(FILE *powerups);
```
Lê os dois valores armazenados no arquivo de powerups para que as compras realizadas na loja fiquem armazenadas mesmo depois que o usuário feche o arquivo.

```
int compraPowerup(FILE *powerups, char tipo);
```
A função de comprar powerups recebe a variável de arquivo dos powerups e um tipo, sendo 'h' referente a compra de um powerup de velocidade horizontal e 't' de um powerup de tiro. A função aumenta a quantidade de powerups no arquivo de powerups em uma unidade na categoria indicada e subtrai o valor padrão de um powerup da quantidade de moedas.

```
int buttonClick(int mouse_x, int mouse_y, int x1, int y1, int x2, int y2);
```
Identifica se o clique nas coordenadas mouse_x e mouse_y ficaram dentro da área do retângulo delimitado por (x1, y1) e (x2, y2).

```
void preenchePowerUp();
```
Obtém os valores de power ups comprados por meio da função getPower upData( ) e adiciona