**Trabalho Prático – MultiTênis**

1. **Manual de uso**

Ao executar o programa são exibidas duas barras, de cores aleatórias, que funcionam como os jogadores. O jogador localizado na parte superior da tela é o jogador 1 (de nome “P1”) e o jogador localizado na parte inferior é o jogador 2 (de nome “P2”), sendo que cada jogador pode se movimentar apenas em uma metade da tela, sem invadir o campo do outro jogador. Os movimentos do jogador 1 são dados pelas teclas A, D, S e W, sendo, respectivamente, para se mover para a esquerda, direita, baixo e cima, com a tecla J para rebater a bola caso esta esteja perto do jogador. Já os movimentos do jogador 2 se dão pelas setas para direita, esquerda, baixo e cima, com a tecla Espaço para rebater a bola caso esta esteja perto do jogador. A bola só é rebatida para o outro campo se o jogador apertar a tecla de rebate, e os pontos são contabilizados apenas se a bola encosta nos limites do eixo Y (0 ou SCREEN\_H (tamanho da tela)). Ao fechar o display, é impresso qual foi o vencedor (ou em caso de empate) e o placar, além do programa gerar um arquivo de nome “historico\_partidas” do tipo .txt, que armazena as partidas jogadas, bem como o vencedor.

1. **Implementação**
   1. **void initplayer1 (player \*player);** e **void initplayer2 (player \*player);**

Inicia os jogadores a partir do parâmetro por referência de uma estrutura do tipo jogador. A estrutura apresenta variáveis do tipo inteiro (right, left, up, down, attack, points, id), flutuante (x, y, veloc) e ALLEGRO\_COLOR para armazenar o código RGB do jogador, que é gerado aleatoriamente. Foi necessário fazer duas funções, pois cada jogador apresenta uma posição inicial (x, y), cores e identidades diferentes.

* 1. **void draw\_players (player player1, player player2);**

Desenha um retângulo preenchido para cada jogador. É usado a função al\_draw\_filled\_rectangle(x, y, x’, y’, color); para determinar o tamanho do jogador, então, x’ – x é o comprimento do jogador e y’ – y é a altura do jogador, e color é uma variável do tipo ALLEGRO COLOR determinada pela struct do jogador.

* 1. **void update\_player1 (player \*player);** e **void update\_player2(player \*player);**

Atualiza as posições do jogador se atende às condições determinadas para não ultrapassar os limites da tela. Se o jogador tiver apertado a tecla para a direita, a função verifica se a posição x do jogador somado ao deslocamento do jogador e à metade do seu comprimento é menor que o tamanho do eixo x da tela. Se sim, é somado à posição x do jogador o deslocamento do jogador. A lógica é parecida para movimentar para a esquerda. Se o jogador tiver apertado a tecla para mover o jogador para cima, a função verifica se a posição y do jogador somado ao deslocamento do jogador e à metade de sua altura é retirado da posição y do jogador o deslocamento do jogador.

* 1. **void init\_ball (ball \*b, int position\_ball);**

Inicializa a bola em algum dos 4 cantos da tela, a depender do parâmetro position\_ball. Para isso, foi implementado a lógica com o switch case, sendo que no case 0 a bola começa no canto superior direito, para o case 1 no canto inferior direito, para o case 2 no canto inferior esquerdo e para o case 3 no canto superior esquerdo. O raio da bola varia de 5 a 10 e o deslocamento da bola tem sinais diferentes a depender de onde ela surge.

* 1. **void collision (ball \*ball, player \*player, int id);**

Função recebe um dado ball por referência do tipo estrutura ball, outro dado player por referência do tipo estrutura player e um dado id do tipo inteiro. A princípio, a função verifica se o botão de rebate foi apertado e qual foi o jogador que fez o ataque, para em seguida verificar se a posição x e y da bola está dentro da área do retângulo do jogador para inverter a direção da bola.

* 1. **void update\_ball (ball \*ball);**

É incrementado na posição x e y da bola o deslocamento dx e dy da bola, e verifica se a posição x está invadindo os limites da tela (maior que SCREEN\_W ou menor que 0) – se sim, inverte o sinal de dx para inverter a direção da bola.

* 1. **void check\_points (ball \*ball, player \*player1, player \*player2);**

Verifica se a bola ultrapassa os limites do eixo y da tela – se ultrapassa o limite superior é contabilizado mais um ponto para o jogador 2 em sua variável points da struct player2, se ultrapassa o limite inferior é contabilizado mais um ponto para o jogador 1 em sua variável points da struct player1.

* 1. **void draw\_ball (ball \*ball);**

Procedimento para imprimir um círculo preenchido que representam as bolas do jogo.

* 1. **void draw\_scenario ( )**;

Procedimento que limpa a tela e imprime dois retângulos preenchidos que representam as quadras de cada jogador. Os retângulos possuem tons de verde diferentes e são separados por uma linha fina branca.

* 1. **void match (player \*player1, player \*player2, int winner);**

Registra em um arquivo chamado “histórico\_partidas.txt” o vencedor da partida e o placar. O arquivo foi aberto como “a”, para, toda vez que a função for chamada, o arquivo for atualizado com as novas informações, sem retirar as anteriores.

* 1. **int main (int argc, char \*\*argv);**

Na função principal foram declaradas as structs, algumas variáveis auxiliares para os laços utilizados durante a implementação do código e uma string para imprimir a tela final com o placar e o vencedor da partida. A princípio o código apresenta as rotinas de inicialização da biblioteca, do módulo de primitivas, do display, do teclado, do temporizador, do mouse (que foi usado apenas para auxiliar no processo de implementação do código, mas não tem função durante o jogo), entre outras inicializações. Em seguida foi criada a fila de eventos e o registro dos eventos de tela nessa fila, para implementar o jogo.

Posteriormente, dentro do while, foram implementadas as funções apresentadas de forma que o jogo funcionasse corretamente gerando 12 bolas (o máximo sendo determinado por um define) que nascem em posições diferentes, somem quando ultrapassam os limites superiores e inferiores e contabilizam os pontos, imprimindo o placar na tela enquanto o jogo roda e também o placar final e o jogador vencedor quando a partida acaba.