

CENTRO UNIVERSITÁRIO DE JOÃO PESSOA - UNIPÊ
CIÊNCIA DA COMPUTAÇÃO

BRUNO RODRIGUES DOS SANTOS
LEONARDO CUNHA LUNA FREIRE
LUKAS DE SOUSA FIGUEIRÊDO

PONGC SDL2

JOÃO PESSOA-PB
2020

BRUNO RODRIGUES DOS SANTOS - 23046317
LEONARDO CUNHA LUNA FREIRE - 23148764
LUKAS DE SOUSA FIGUEIRÊDO - 23724501

PONGC SDL2

Trabalho desenvolvido para a disciplina Técnicas de Desenvolvimento de Algoritmos do Centro Universitário de João Pessoa - UNIPÊ, como requisito para obtenção de nota referente a segunda avaliação, sob a orientação do Prof. Leandro Figueiredo Alves.

JOÃO PESSOA-PB

2020

SUMÁRIO

RESUMO	4
INTRODUÇÃO	5
RESULTADOS	6
CONSIDERAÇÕES FINAIS	16
BIBLIOGRAFIA.....	17
APÊNDICE	18

RESUMO

O projeto que será apresentado trata-se de um clone do famoso jogo PONG. Para isso foi utilizado a linguagem de programação C, conforme requisitado e também foi utilizado a biblioteca Simple DirectMedia Layer (SDL 2.0), a qual permite um acesso mais simplificado para os dispositivos de gráfico, som e entrada.

INTRODUÇÃO

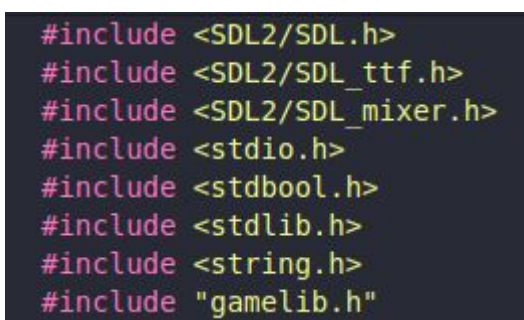
PONGC é um jogo baseado no famoso jogo PONG desenvolvido pela Atari, o qual deu início ao mercado de videogames nos anos 70. Esse jogo consiste em duas dimensões que simula o tênis de mesa, onde dois jogadores controlam uma paleta (barra vertical), movendo-a verticalmente conforme desejado, de cima para baixo ou de baixo para cima.

O objetivo do jogo é acertar com a paleta a esfera que se move em diversas direções ao longo do jogo e impedir que ela ultrapasse as bordas laterais. Caso o jogador não consiga rebater a esfera e a mesma ultrapasse qualquer uma das bordas laterais da esquerda ou da direita, o jogador adversário pontuará, vencendo temporariamente a partida e o jogo reiniciará automaticamente. O jogador que somar os cinco primeiros pontos, vencerá a partida em definitivo.

RESULTADOS

O projeto PONGC foi implementado exclusivamente em linguagem C, utilizando a biblioteca SDL, a qual é considerada uma biblioteca de baixo nível, uma vez que ela somente facilita o acesso aos dispositivos de entrada e saída da máquina, ou seja, é necessário que o programador desenvolva toda a lógica da aplicação (física da aplicação, gráficos, movimentação, sistema de rankings e etc). Isso difere da utilização de uma engine de jogo (ex.: godot ou Unity), que já possui essas lógicas feitas. Além do SDL, foram usadas as bibliotecas comuns da linguagem C: `stdbool.h` (cria o tipo booleano no c), `stdlib.h` (permite alocação dinâmica da memória), `string.h` (permite a manipulação de strings) e o `stdio.h` (permite o I/O pelo terminal).

Figura 1: Bibliotecas utilizadas no trabalho



```
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_mixer.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include "gamelib.h"
```

Fonte: Elaboração própria

Sobre a biblioteca `gamelib.h`, foi criada especialmente para esse projeto para cuidar das lógicas do jogo. Ela permite o processamento dos eventos do jogo, o processamento gráfico, bem como a leitura e escrita de um arquivo de texto, o qual guarda os resultados de todas as partidas da aplicação em tela.

A organização para concepção do jogo foi feita na elaboração de 3 pilares: criação de estruturas especiais, loop de eventos (cria a ilusão de movimento de animação) e leitura / escrita dos resultados das partidas.

No caso dessas estruturas especiais, elas consistem na utilização do `typedef struct`, a qual cria variáveis que comportam mais de um variável, possibilitando o planejamento de

tipos complexos, os quais facilitam a leitura do código, bem como a implementação de funcionalidades.

Figura 2: Estruturas utilizadas no jogo

```
// Jogador
typedef struct
{
    // posição dos jogadores
    int x, y, score, win, lost, foundPlayer;
    char name[100];
} Player;

// Bola
typedef struct
{
    // posição da bola
    int x, y;
    // atributo de velocidade horizontal e vertical
    int velx, vely;
} Ball;

// Texto
typedef struct
{
    int texW;
    int texH;
    TTF_Font * font;
    SDL_Color color;
} Texto;

// Rank
typedef struct
{
    char playerName[100];
    int v, l;
} rank;
```

Fonte: Elaboração própria

Sobre o próximo pilar, o loop principal de eventos jogo cuida de todos os eventos dinâmicos que acontecem durante a execução da partida.

Figura 3: Loop principal

```

//Loop principal do jogo
while(stage > 0)
{
    //Observar os eventos do jogo
    stage = processEvents(window, &playerA, &playerB, &ball, hitFx, &stage);

    if(stage == 1)
    {
        // Renderiza o texto do menu
        doRenderMenu(renderer, &textMenu);
    }
    else
    {
        //Renderiza no display os eventos do jogo
        doRenderGame(renderer, &playerA, &playerB, &ball, &textVictory, &stage);
    }

    //Controla o tempo do interno do jogo
    SDL_Delay(10);
}

```

Fonte: Elaboração própria

Analisando a imagem anterior, percebe que o jogo consiste num loop controlado pelo `SDL_DELAY()`, o qual consiste na criação de ilusão de animação, já que essas funções serão acionadas até o evento que finaliza o jogo (algum jogador alcançar 5 pontos).

Em síntese, sobre o funcionamento do loop, primeiro, existe o processamento dos eventos(movimentação dos elementos e etc) e, logo após, os acontecimentos serão dispostos na tela para o usuário.

No último pilar da organização, há a leitura e escrita do arquivo de texto que possui todo o histórico de partidas da aplicação. A solução implementada foi a criação de um ciclo de leitura, atualização dos valores, escrita e impressão do ranking atualizado. Ou seja, o programa primeiramente carrega todas as informações do histórico para a memória, analisa aqueles jogadores que precisam de atualização e, no final, todas as informações das partidas serão despejadas no arquivo, reescrevendo totalmente o arquivo original e impressas para os usuários esse novo arquivo.

Além desses pilares que sintetizam o funcionamento do jogo, faz necessário uma breve explicação de certas lógicas utilizadas.

Sobre o controle de movimento dos jogadores, utilizamos a função `processEvents` para processar os eventos do jogo. O `SDL_PollEvent()` é uma função do próprio SDL que pesquisa eventos pendentes, recebendo eventos do sistema. A `SDL_KEYDOWN` é uma estrutura pertencente `SDL_KeyboardEvent`, usada especificamente quando um evento de pressionamento de tecla ocorre. O auxílio da tabela `SDL_Keycode`, presente na própria documentação da biblioteca, pode ser necessária. A `SDL_QUIT` possibilita o fechamento de tela no console.

Figura 4: Bloco de leitura de input do teclado

```
int processEvents(SDL_Window *window, Player *playerA, Player *playerB, Ball *ball, Mix_Chunk * hitFx)
{
    SDL_Event event;
    float playerCenter, diag;
    while(SDL_PollEvent(&event))
    {
        switch(event.type)
        {
            case SDL_WINDOWEVENT_CLOSE:
            {
                if(window)
                {
                    SDL_DestroyWindow(window);
                    window = NULL;
                    stage = -1;
                }
            }
            break;
            case SDL_KEYDOWN:
            {
                switch(event.key.keysym.sym)
                {
                    case SDLK_ESCAPE:
                    {
                        stage = -1;
                        break;
                    }
                }
            }
            break;
            case SDL_QUIT:
            {
                stage = -1;
                break;
            }
        }
    }
}
```

Fonte: Elaboração própria

Na imagem abaixo temos os eventos ainda da função `processEvents`. `SDL_GetKeyboardState` é usada para obter o estado atual do teclado. Stage refere-se a variável responsável pelo estado do jogo, a qual controla se o jogo está no menu ou foi iniciado. `SDL_SCANCODE_` são códigos pertencentes a tabela Scancode.

Figura 5: Bloco de controle movimento

```

const Uint8 *state = SDL_GetKeyboardState(NULL);
if(state[SDL_SCANCODE_SPACE])
{
    stage = 2;
}
if(state[SDL_SCANCODE_W] && playerA->y > 5)
{
    playerA->y -= 5;
}
if(state[SDL_SCANCODE_S] && playerA->y < 405)
{
    playerA->y += 5;
}
if(state[SDL_SCANCODE_UP] && playerB->y > 5)
{
    playerB->y -= 5;
}
if(state[SDL_SCANCODE_DOWN] && playerB->y < 405)
{
    playerB->y += 5;
}

```

Fonte: Elaboração própria

O trecho de código abaixo cuida da lógica de colisão, uma parte central do jogo como um todo, uma vez que é necessário a ciência precisa desse evento para o jogo ocorrer plenamente. Essa parte foi desafiadora, já que precisamos criar uma lei de física que só se aplica no mundo delimitado do código, bem como raciocinar sobre todos os eventos possíveis dessa concepção para que não ocorra nenhum evento não previsto.

Sobre a solução desenvolvida da colisão, foi criada uma checagem através de condicionais sobre o posicionamento da bola e o jogador atingido. Conforme a posição do impacto da bola no jogador, será adicionada uma velocidade no eixo y, criando um movimento diagonal, essa mesma lógica ocorre quando a bola bate os cantos superiores e inferiores, criando uma ilusão de quadra. Dentro dessa checagem de colisão, foi implementada uma chamada para um arquivo de áudio, o qual permite uma imersão maior do jogador na aplicação.

Figura 6: Lógica de colisão

```
// colisão com o jogador
if ((ball->y >= playerB->y && ball->y <= playerB->y + 100 && ball->x == playerB->x - 10)
|| (ball->y >= playerA->y && ball->y <= playerA->y + 100 && ball->x == playerA->x + 5))
{
    if (ball->x > 320)
    {
        Mix_PlayChannel(-1, hitFx, 0);
        playerCenter = playerB->y + 35;
        diag = playerCenter - ball->y;
        ball->vely += diag * -0.1;
        ball->velx *= -1;
    }
    else
    {
        Mix_PlayChannel(-1, hitFx, 0);
        printf("Bx: %d\n", playerB->x);
        printf("By: %d\n", playerB->y);
        printf("Ballx: %d\n", ball->x);
        printf("Bally: %d\n", ball->y);
        playerCenter = playerA->y + 35;
        diag = playerCenter - ball->y;
        ball->vely += diag * -0.1;
        ball->velx *= -1;
    }
}
```

Fonte: Elaboração própria

A figura abaixo refere-se a estrutura de fim de jogo. Por regra, ocorre quando um dos jogadores pontua cinco pontos.

Figura 7: Código de fim do jogo

```
if (playerA->score >= 5 || playerB->score >= 5 )
{
    stage = -1;
}
```

Fonte: Elaboração própria

Abaixo vemos a função de renderização de menu com escolha das cores, cor da tela e de novos desenhos.

Figura 8: Trecho do código de renderização gráfica

```

void doRenderMenu(SDL_Renderer *renderer, Texto * textoVic)
{
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0);

    SDL_RenderClear(renderer);

    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);

    SDL_Surface * msgA = TTF_RenderText_Solid(textoVic->font, "PONG C", textoVic->color);
    SDL_Texture * msgTextA = SDL_CreateTextureFromSurface(renderer, msgA);
    SDL_QueryTexture(msgTextA, NULL, NULL, &textoVic->texW, &textoVic->texH);

    SDL_Surface * msgB = TTF_RenderText_Solid(textoVic->font, "ESPACO - Para comecar", textoVic->color);
    SDL_Texture * msgTextB = SDL_CreateTextureFromSurface(renderer, msgB);
    SDL_QueryTexture(msgTextB, NULL, NULL, &textoVic->texW, &textoVic->texH);

    SDL_Surface * msgC = TTF_RenderText_Solid(textoVic->font, "ESC - Para finalizar jogo", textoVic->color);
    SDL_Texture * msgTextC = SDL_CreateTextureFromSurface(renderer, msgC);
    SDL_QueryTexture(msgTextC, NULL, NULL, &textoVic->texW, &textoVic->texH);

    SDL_Rect bemvindo = { 90, 40, textoVic->texW, textoVic->texH };
    SDL_Rect instrucaoA = { 90, 160, textoVic->texW, textoVic->texH };
    SDL_Rect instrucaoB = { 90, 200, textoVic->texW, textoVic->texH };

    SDL_RenderCopy(renderer, msgTextA, NULL, &bemvindo);
    SDL_RenderCopy(renderer, msgTextB, NULL, &instrucaoA);
    SDL_RenderCopy(renderer, msgTextC, NULL, &instrucaoB);
    SDL_RenderPresent(renderer);
}

```

Fonte: Elaboração própria

Na imagem abaixo temos apenas uma parte do código do main do jogo. Nele vemos a declaração da janela, declaração de renderização e inicialização do SDL, assim como também os atributos de inicialização das peças com suas características. Ainda no main teremos comandos para abertura de áudio, configurações referentes a janela como largura e altura de pixels, assim como também limpeza de memória. Por motivos de simplificação não mostraremos o código completo, visto que ele será disponibilizados junto a este trabalho para todos os interessados.

Figura 9: Trecho do código que cria elementos do jogo

```
int main()
{
    SDL_Window *window;
    SDL_Renderer *renderer;

    SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO);
    TTF_Init();

    Player playerA;
    playerA.x = 5;
    playerA.y = 190;
    playerA.score = 0;

    Player playerB;
    playerB.x = 630;
    playerB.y = 190;
    playerB.score = 0;

    Ball ball;
    ball.x = 220;
    ball.y = 120;
    ball.velx = 2;
    ball.vely = 0;

    Texto textMenu;
    textMenu.texW = 0;
    textMenu.texH = 0;
    textMenu.color.r = 255;
    textMenu.color.b = 255;
    textMenu.color.g = 255;
    textMenu.color.a = 255;
    textMenu.font = TTF_OpenFont("Roboto.ttf", 38);

    Texto textVictory;
    textVictory.texW = 0;
    textVictory.texH = 0;
    textVictory.color.r = 255;
    textVictory.color.b = 255;
```

Fonte: Elaboração própria

Após uma análise das imagens acima, podemos perceber a abrangência de possibilidades que o SDL nos permite desenvolver na elaboração e criação de um jogo. No entanto, uma das dificuldades encontradas consiste no material ainda escasso referentes a esta biblioteca. Ainda que seja possível achar facilmente a documentação em seu site oficial, e não há dúvidas de que ele foi de grande utilidade, existem funções que carecem de mais informações, detalhes e exemplos que ajudariam ainda mais os usuários.

Agora uma explicação breve de cada tela existente na aplicação. Na primeira tela, mostra o espaço que os jogadores devem registrar seus primeiros nomes na aplicação, os nomes não podem ser iguais ou possuir um espaço, o código faz um tratamento quando um nome não seguir as regras e pede novamente a reescrita dos nomes.

Figura 10: Primeira tela do jogo

```
PONG C
Insira nome sem espaco
Digite o nome do Jogador A: jogadora
Digite o nome do Jogador B: jogadorB
```

Fonte: Elaboração própria

Na segunda tela, ocorre logo após o registro de nomes válidos, é criada uma janela gráfica, a qual contém o título do jogo, informações básica de controle.

Figura 11: Segunda tela do jogo

```
PONG C

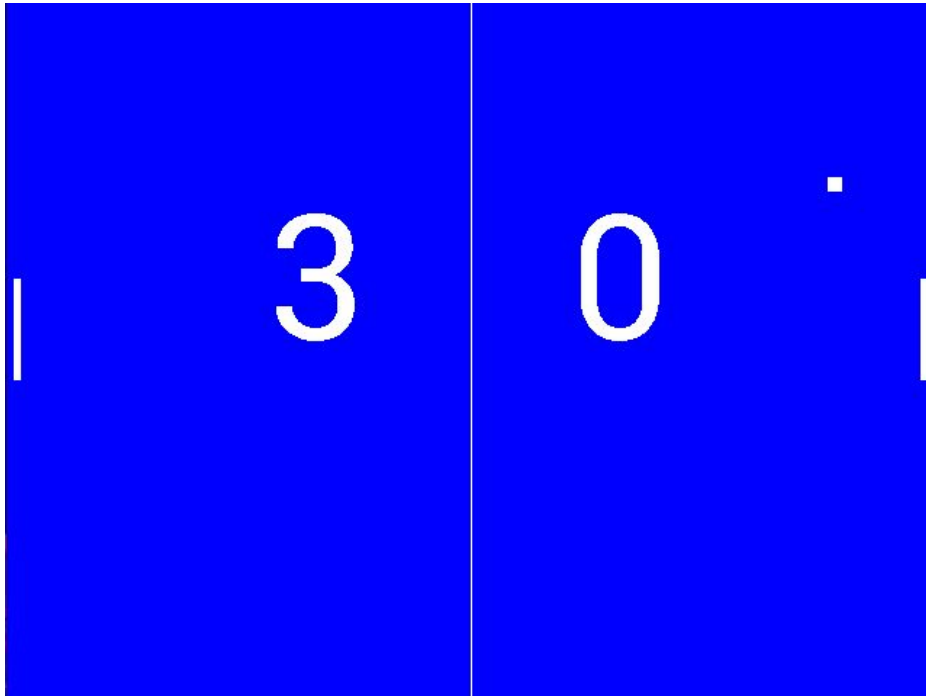
Controle A: W / S
Controle B: Seta Cima / Seta Baixo

ESPACO - Para comecar
ESC - Para finalizar jogo
```

Fonte: Elaboração própria

Na terceira tela, o jogo é iniciado após o acionamento da tecla espaço, é possível ver todos os elementos do jogo em ação, os jogadores, bola e placar, bem como eventuais efeitos sonoros no impacto da bola com o jogador ou com algum limite de campo.

Figura 12: Terceira tela do jogo



Fonte: Elaboração própria

Na última tela, ocorre quando algum jogador chega no placar de 5 pontos primeiro, a janela gráfica é terminada e o programa imprime no terminal todo histórico de partida.

Figura 13: Última tela do jogo

```
PONG C
Insira nome sem espaco
Digite o nome do Jogador A: jogadorA
Digite o nome do Jogador B: jogadorB

-----Exibindo RANK-----
Jogador: leo          Vitoria: 3          Derrota: 0
Jogador: teste        Vitoria: 0          Derrota: 2
Jogador: outroteste   Vitoria: 0          Derrota: 1
Jogador: jogadorA     Vitoria: 0          Derrota: 1
Jogador: jogadorB     Vitoria: 1          Derrota: 0
```

Fonte: Elaboração própria

CONSIDERAÇÕES FINAIS

Em relação à elaboração do projeto, observando somente a experiência do usuário, a equipe pensa que a aplicação atingiu um estágio de desenvolvimento satisfatório, uma vez que é possível ter uma partida completa do jogo proposto, bem como um registro do histórico da partida.

No tocante às partes da aplicação feitas no terminal, pensamos que é um ponto que poderia ser melhorado num trabalho futuro, uma vez que a mistura de elementos de terminal e elementos gráficos não foi ideal. Outra melhoria futura nessa área gráfica, seria a inclusão de um sistema de rastro da bola, o qual aumentaria a qualidade do jogo.

No aspecto técnico, no geral, o código possui uma boa organização estrutural, utilização de funções, bibliotecas e tipos complexos que facilitaram o desenvolvimento da aplicação. Atingimos o nosso objetivo de deixar a função `main()` mais breve possível. Contudo, a função do processamento de evento do jogo poderia ser refatorada, uma vez que ficou muito grande e genérica, misturando assuntos de física, controle dos jogadores e outras partes.

Por fim, o projeto foi uma ótima oportunidade de colocar em prática todo o conhecimento adquirido pela disciplina, bem como o aprendizado de novos (lógica de colisão, lógica de loop de um jogo e outros), os quais só seriam possíveis de adquirir pela elaboração de um projeto prático.

BIBLIOGRAFIA

SDL: Simple DirectMedia Layer. , Sdl, 2020. Disponível em:<<http://https://www.libsdl.org/>>. Acesso em: 25 nov. 2020.

LEARN video game programming in c. , Vertostudio3d, 2013. Disponível em: <<http://https://www.youtube.com/channel/UCPsK6vduM-ZqbszTuzNJrcQ>>. Acesso em: 23 nov. 2020.

SDL TUTORIALS. , Sdl, 2020. Dispon?vel em: <<http://https://wiki.libsdl.org/Tutorials>>. Acesso em: 22 nov. 2020.

SDL FORUMS. , Sdl, 2020. Dispon?vel em: <<http://https://discourse.libsdl.org/>>. Acesso em: 25 nov. 2020.

APÊNDICE

Todo o código do projeto está disponibilizado no GitHub e pode ser facilmente acessado através do link que se segue: <https://github.com/leonardoclf/pongC>