UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Ciências de Computação

Nome: Guilherme Ramos Costa Paixão **N°USP:**11796079
Nome: Jônatas Alves Lopes **N°USP:**11796552

Trabalho Final: Sistemas Operacionais - Manual do Jogo

O protagonista do jogo tem como meta pegar os sapos que estão caindo do céu enquanto evita encostar nos tijolos. O personagem tem três vidas, representadas pelos corações no canto superior esquerdo da tela, e a cada tijolo encostado ou sapo que cai no chão ele perde uma dessas vidas; caso perca todas as vidas você perde o jogo. A cada sapo coletado, você ganha um ponto. O contador de pontos fica no canto superior direito da tela e o total de pontos aparece na tela ao perder. O objetivo, então, é fazer o maior número de pontos, coletando sapos, antes de perder todas as vidas.

Controles:
As teclas A, D ou setas para esquerda e direita controlam o movimento do personagem.

Como Instalar:

1. Instalar a biblioteca SDL2 e o plugin SDL_Image. Em sistemas baseados em Debian pode ser feito assim:
   `sudo apt install libsdl2-dev libsdl2-image-dev`
   ou então compilando as bibliotecas já no arquivo do projeto: no arquivo de cada biblioteca – SDL2 e SDL2_Image – execute os seguintes comandos:

   ```
   ./configure
   make
   sudo make install
   ```

2. No diretório, compile o jogo com o comando "*make*".
3. Após isso, o jogo está pronto para ser rodado!

Uso de threads no jogo:

Foi utilizado uma *thread* específica para capturar os comandos do jogador através de uma função chamada "*getInput*". Assim, de forma concorrente, enquanto o jogo renderizava na thread principal em um loop, as entradas de teclado eram capturadas.

```
    graphics = new Graphics();
    points = 0;
    t_input = std::thread(&Game::getInput, this);
    mixer = new Mixer(&quit);
}
```

```
void Game::getInput() {
    while (!quit) {
        while (SDL_PollEvent(&event)) {
            switch (event.type) {
            case SDL_QUIT:
                quit = true;
                break;

            case SDL_KEYUP:
                keyPressed(event.key.keysym.sym, false);
                break;

            case SDL_KEYDOWN:

                keyPressed(event.key.keysym.sym, true);
                break;
            }
        }

    }
}
```

Nessa thread, os semáforos foram representados pelos comandos capturados que realizariam operações diferentes em funções da thread principal.

Também foram utilizadas threads para a saída de áudio do jogo: quando cada som específico é ativado, é criada uma nova thread para reproduzir o som.

```
bool Mixer::play(int song_id) {
    if (song_id > SAMPLES_N || samples[song_id]==NULL)
        return false;

    while (detach_mutex);
    if (threads[song_id] != NULL) {
        detach_mutex = true;
        D std::cout << samples[song_id]->path << " already playing" << std::endl;
        samples[song_id]->stop();
        samples[song_id]->restart();
        threads[song_id]->detach();
        delete threads[song_id];
        threads[song_id] = NULL;
        detach_mutex = false;
    }

    threads[song_id] = new std::thread(&Sound::play, samples[song_id]);

    return true;
}
```

Esse controle é realizado pela classe Mixer. Foi utilizado um semáforo mutex (detach_mutex) para evitar o acesso de mais de uma thread ao mesmo tempo, quando é feita a verificação se ela está ativa.