ALGORITMOS E PROGRAMAÇÃO
Engenharia de Controle e Automação
Turma H

Juan Marcelo Trabaina

Vitor Dal Bó Abella

Trabalho Final

The Binding of Isaac

PORTO ALEGRE

2023

Introdução

Este trabalho visa a implementação de um jogo baseado em *The Binding of Isaac* em C. Neste documento será descrito como os elementos do jogo foram representados, como foi implementada a interação dos componentes interativos, bem como as estruturas e funções utilizadas e uma explicação de como usar o programa.

Instruçẽs para Execução

Windows:

Compilar o programa

Compilar o arquivo main.c.

Executar

Executar TheBindingOfIsaac_Windows.exe.

Ubuntu 22.04:

Se desejar somente executar o programa, pular para a etapa executar.

Instalar raylib

```
sudo apt-get update
sudo apt install build-essential git
```

```
sudo apt install libasound2-dev libx11-dev libxrandr-dev libxi-dev libgl1-mesa-dev libglu1-mesa-dev libxcursor-dev libxinerama-dev
```

```
git clone https://github.com/raysan5/raylib.git raylib
cd raylib/src/
make PLATFORM=PLATFORM_DESKTOP # To make the static version.
make PLATFORM=PLATFORM_DESKTOP RAYLIB_LIBTYPE=SHARED # To make the dynamic shared version.
make PLATFORM=PLATFORM_WEB # To make web version.
```

```
sudo make install # Static version.
sudo make install RAYLIB_LIBTYPE=SHARED # Dynamic shared version.
```

Compilar o Programa

```
gcc -o TheBindingOfIsaac_Ubuntu main.c -lraylib -lGL -lm -pthread -ldl;
```

Executar

```
./TheBindingOfIsaac_Ubuntu;
```

Controles do Jogo

Durante o jogo

A,S,D,W: mover, respectivamente, para esquerda, baixo, direita e cima;
Setas (←, →, ↓ e ↑): atirar na direção correspondente;
E: Posiciona Bomba;
G: god mode;
O: Save game (durante o jogo);
L: Load Game (durante o jogo);
R: Novo jogo;
ESC: abre Menu.

Menu

Setas (↓ e ↑): Controlar Menu;
N: Novo jogo;
C: Carregar Jogo;
S: Salvar Jogo;
Q: Sair;
V: Voltar para o estado atual do jogo.
Esc: Voltar para o estado atual do jogo.

Recursos utilizados

- **Codeblocks** (IDE de programação)
- **Linguagem de Programação C**
- **Bibliotecas em C:**
    - **raylib.h** (interface grafica)
    - **string.h** (manipulação de strings)
    - **time.h** (manipulação de tempo)
    - **stdio.h** (realização de inputs e outputs)
    - **stdlib.h** (alocação de memória)
- **gitHub e git** (plataforma e programa para controle de versão de arquivos)

Desenvolvimento

Overview dos arquivos:
O programa é composto de vários arquivos e diretórios.

- main.c (arquivo principal para a compilação)
- main (executavel linux)
- main.exe (executável windows)
- Codes (arquivos de funções)
    - *constants.h*
    - *gameFunctions.c*
    - *gameLoop.c*
    - *globals.h*
    - *initializeVariables.h*
    - *keyboardFunctions.h*
    - *manageFiles.h*
    - *mapToArray.c*

    - *mazeSolverFloydWarshall.h*
    - *movements.h*
    - *shoot.c*
    - *stopwatch.h*
    - *structs.h*
    - *visualInterface.h*
- maps (arquivos de mapas)
    - map01.txt
    - … (demais mapas)
- sprites (imagens das estruturas do jogo)
    - isaac.png
    - … (demais sprites)

Descrição dos arquivos

1. ***constants.h***
   São declaradas as constantes do jogo, tais como dimensões do mapa e tamanho das arrays.

2. ***gameFunctions.c***
    - NewGame()
      Contém loop do jogo para cada mapa.
    - QuickLoadGame()
      Modifica variável que ordena carregar último jogo salvo.
    - QuickSaveGame()
      Modifica variável que ordena salvar jogo no estado atual.
3. ***gameLoop.c***
    - *gameLoop()*
      Contém loop que coordena o jogo em determinada fase. Aqui estão as são executadas as funções que leem o teclado para movimento e tiro do isaac, função para movimento do inimigo, verificação de gameOver, morte do isaac
4. ***globals.h***
    - Aqui estão as variáveis globais. *Como por exemplo map.*
5. ***initializeVariables.h***
    - *initializeIsaacEnemiesBullets()*
      Inicializa variáveis das estruturas Isaac e Enemy
    - *initializeMapElement()*
      Inicializa vetor de estruturas mapElement

6. ***keyboardFunctions.h***
   Este arquivo contém algumas funções relacionadas à leitura de entradas do teclado para controlar o jogo. Elas abrangem a detecção de teclas pressionadas para movimento, disparo, deixar bombas e executar comandos especiais.
    - readKeyboardShoot()
      Detecta as teclas pressionadas para realizar disparos. Verifica a direção e a disponibilidade do espaço no mapa antes de criar um projétil.
    - readKeyboardMove()
      Lê as entradas de movimento do teclado e atualiza as coordenadas do personagem Isaac.
    - readKeyboardSpecialKeys()
      Lê as teclas especiais do teclado e executa comandos específicos, como carregar/salvar o jogo e ativar/desativar o GOD mode.
    - readKeyboardLeaveBomb()

Lê o teclado para colocar uma bomba no mapa, desde que haja bombas disponíveis e o espaço ao redor do personagem esteja vazio.

- readKeyboard()

  Função agregada para ler todas as entradas do teclado e executar as ações correspondentes, como movimento, disparo e colocação de bombas.

7. ***manageFiles.h***

Contém funções relacionadas à manipulação de arquivos, especialmente para salvar e carregar jogos.

- *NumberOfMaps()*

  Esta função conta o número de arquivos na pasta "./maps/" e retorna a quantidade total de mapas disponíveis.

- *saveGame()*

  Esta função salva o estado atual do jogo em um arquivo binário. localizado no diretório “savedGame”.

- *loadGame()*

  Esta função carrega um jogo a partir de um arquivo binário localizado no diretório *“savedGame”.*

8. ***mapToArray.c***

Neste arquivo é declarado o map, que é uma matriz que contém cada posição do mapa.

- *ReadMap()*

  Esta função lê um arquivo de mapa, atualiza a matriz 'map' com os caracteres do mapa, define a posição inicial do personagem e dos inimigos de acordo com o conteúdo do arquivo lido.

9. ***mazeSolverFloydWarshall.h***

O inimigo persegue Isaac no menor caminho possível, desviando das paredes. Neste arquivo estão todas as funções para criação da matriz nextMoveMatrix[nVertices][nVertices], onde a linha representa a posição do inimigo, a coluna a de isaac e o valor da matriz o próximo movimento do inimigo (de 0-9):

```
////////////////////
// 1 -- 2 -- 3
// 4 -- 5 -- 6
// 7 -- 8 -- 9
////////////////////
```

5 é a posição do inimigo
nextMoveMatrix[enemyVertex][isaacVertex]=1 -> inimigo deve se mover para esquerda em cima para perseguir Isaac no menor caminho possível.
nextMoveMatrix[enemyVertex][isaacVertex]=2 -> inimigo deve se mover para cima para perseguir isaac no menor caminho possível.
e assim sucessivamente.
nextMoveMatrix[enemyVertex][isaacVertex]=0, inimigo deve ficar parado.

Cada mapa possui a sua nextMoveMatrix.

- *mapToMaze()*

  Cria matriz maze a partir da matriz map, que possui somente 0 e 1, onde 1 significa as paredes e 0 não parede.

- *mazeToGraph()*

  Gera o grafo a partir da matriz maze. Grafo é uma matriz de nVertices x nVertices, onde nVertices=60x30=1800, que representa o custo do movimento direto de uma posição A para uma B.

  1. *Custo=0, A=B*
  2. *Custo=1, A é adjacente a B*
  3. *Custo = 999, Impossível ir de A para B em um único movimento.*

- *floydWarshall()*

Rotina para criar matriz Next[enemyVertex][isaacVertex]. Similar a matriz nextMoveMatrix, mas o valor da matriz significa o vértice para o qual o inimigo deve se movimentar, e não a direção como em nextMoveMatrix.

- *nextVertexToDirection()*
  É capaz de converter um elemento da matriz Next para um elemento da Matriz nextMoveMatrix.
- *buildNextMoveMatrix()*
  Cria nextMoveMatrix a partir de Next com a função *nextVertexToDirection()*.
- *indexToVertex()*
  Dada uma linha e uma coluna do mapa, obtém o índice do vértice.
- *printNextMove()*
  Exibe nextMoveMatrix na tela.
- *saveNextMoveMatrix()*
  Salva nextMoveMatrix na pasta chase. Assim não é preciso recalcular esta matriz em cada execução do programa.
- *loadNextMoveMatrix()*
  Carrega *nextMoveMatrix* de arquivo presente na pasta *chase.*
- *getDxdyFromNextMoveMatrix()*
  Converte um elemento de *nextMoveMatrix para variações em x e y. Exemplo,* para o valor "1"*, dx=-1,dy=-1.*
- *checkIfFileExists()*
  Verifica se há uma cópia do mapa na pasta chase. Isso é o primeiro indício se há uma nextMoveMatrix pré calculada para o mapa. Condição necessária, mas não suficiente.
- *checkIfMapHasChanged()*
  Verifica se o mapa na pasta mapas está igual a sua cópia. Se está igual, significa que há uma *nextMoveMatrix correspondente para esse mapa.*
- *areEnemyMovesPreCalculated()*
  Verifica se há uma *nextMoveMatrix* pré calculada para o mapa correspondente.
- *calculateEnemyMovesIfNeeded()*
  Calcula *nextMoveMatrix* se é necessário, ou seja, se não está pré calculado.
- *saveState()*
  Salva cópia do mapa na pasta *chase*. Esta cópia é usada para verificação se não houve mudança no mapa.

10. movements.h

Este arquivo contém um conjunto de funções aos movimentos e interações dos personagens. Ele inclui funções para mover o personagem e os inimigos pelo mapa, verificar a validade desses movimentos, lidar com interações com elementos do ambiente (como fogo e bombas).

- moveIsaac()
  Move o personagem Isaac no mapa e atualiza sua posição.
- verifyMoveIsaac()
  Verifica se o movimento de Isaac é válido e atualiza as informações do personagem conforme a interação com elementos do mapa
- moveAndVerifyIsaac()
  Move e verifica o movimento de Isaac no mapa.
- moveEnemy()
  Move o inimigo no mapa e atualiza sua posição.
- verifyMoveEnemy()
  Verifica se o movimento do inimigo é válido e atualiza as informações do inimigo conforme a interação com elementos do mapa.
- moveAndVerifyIsaac()
  Move e verifica o movimento de Isaac no mapa.
- moveAndVerifyEnemy()

Move e verifica o movimento do inimigo no mapa.

- moveAndVerifyAllEnemies()
  Move e verifica o movimento de todos os inimigos no mapa.
- jumpFireIsaac()
  Evita que Isaac permaneça dentro do fogo após um movimento.
- jumpFireAndBombEnemy()
  Evita que o inimigo permaneça dentro do fogo ou em uma bomba inativa após um movimento.

11. shoot.c

Este arquivo contém funções relacionadas à manipulação dos tiros no jogo.

- *Shoot()*
  Esta função é responsável por disparar uma bala. Ela encontra um slot disponível para a bala e configura suas propriedades, como posição e velocidade.
- *UpdateBullets()*
  Esta função atualiza o estado das balas. Ela verifica se uma bala pode se mover para uma nova posição no mapa, se atinge um inimigo ou se atinge um obstáculo. As balas também são removidas de sua posição anterior no mapa.
- *DamageEnemy()*
  Esta função aplica dano a um inimigo se uma bala atingi-lo. Ela percorre a lista de inimigos e verifica se as coordenadas da bala correspondem à posição de um inimigo vivo. Se o inimigo fica sem vida, ele é desativado e removido do mapa.

12. stopwatch.h

contém a declaração de funções relacionadas ao cronômetro para medir o tempo decorrido.

- *get_elapsed_time()*
  Esta função calcula o tempo decorrido desde que o cronômetro foi iniciado. Ela atualiza a estrutura Stopwatch com o tempo decorrido formatado em horas, minutos e segundos.
- *restart_chronometer()*
  Esta função reinicia o cronômetro, atualizando o tempo inicial para o tempo atual.

13. structs.h

São declaradas as estruturas do jogo:

- *Bullet* (balas lançadas pelo isaac)
- *Isaac* (personagem principal controlado pelo usuário)
- *Enemy* (monstros que querem matar o isaac)
- *Stopwatch* (manipulação do tempo)
- *InformationBarStrings* (armazenamento dos textos a serem exibidos na barra de informações)
- *MapElement* (contém todas as estruturas do mapa: fogueira, chão, parede, isaac, inimigo, bala, bomba ativa, bomba inativa e portal, além das suas interações com isaac e inimigo)
- *GameState* (contém estados do menu: GAME, MENU, WarningMenu e MESSAGE)

14. visualInterface.h

Este arquivo contém funções relacionadas à interface gráfica do jogo.

- *drawMap()*
  Esta função desenha o mapa na tela, percorrendo os elementos do mapa e desenhando os sprites correspondentes.

- *informationBar()*
  Desenha a barra de informações na tela, exibindo informações como vidas, tempo, estágio, etc.
- *createInformationBarStrings()*
  Esta função cria as strings para a barra de informações, formatando os valores das variáveis.
- *drawWindow()*
  Desenha a janela do jogo, chamando as funções anteriores para desenhar o mapa e a barra de informações.
- *DrawMenu()*
  Desenha o menu do jogo, exibindo as opções disponíveis.
- *Menu()*
  Função que lida com a navegação e seleção de opções no menu principal.
- *InGameMenu()*
  Desenha o menu de jogo, que pode ser acessado durante o jogo.
- *AreYouSureMenu()*
  Desenha um menu de confirmação para reiniciar o jogo.
- *drawScore()*
  Desenha a tela de pontuação ao final do jogo.
- *drawGameOver()*
  Desenha a tela de "Game Over".
- *drawYouDied()*
  Desenha a tela de "Você Morreu".
- *missionComplete()*
  Desenha a tela de "Missão Completa", exibida quando um estágio é concluído com sucesso.

Interface Gráfica

A interface gráfica do jogo contém:

- Menu inicial, Figura 1;
- o jogo em si, Figura 2;
- barra de informações embaixo do jogo, Figura 2;
- Menu durante o jogo, Figura 3;
- mensagens para avisar que Isaac morreu, passou de fase fase,*game over* ou virou o jogo, respectivamente, Figuras 4,5,6 e 7

Figura 1 – Menu inicial do jogo

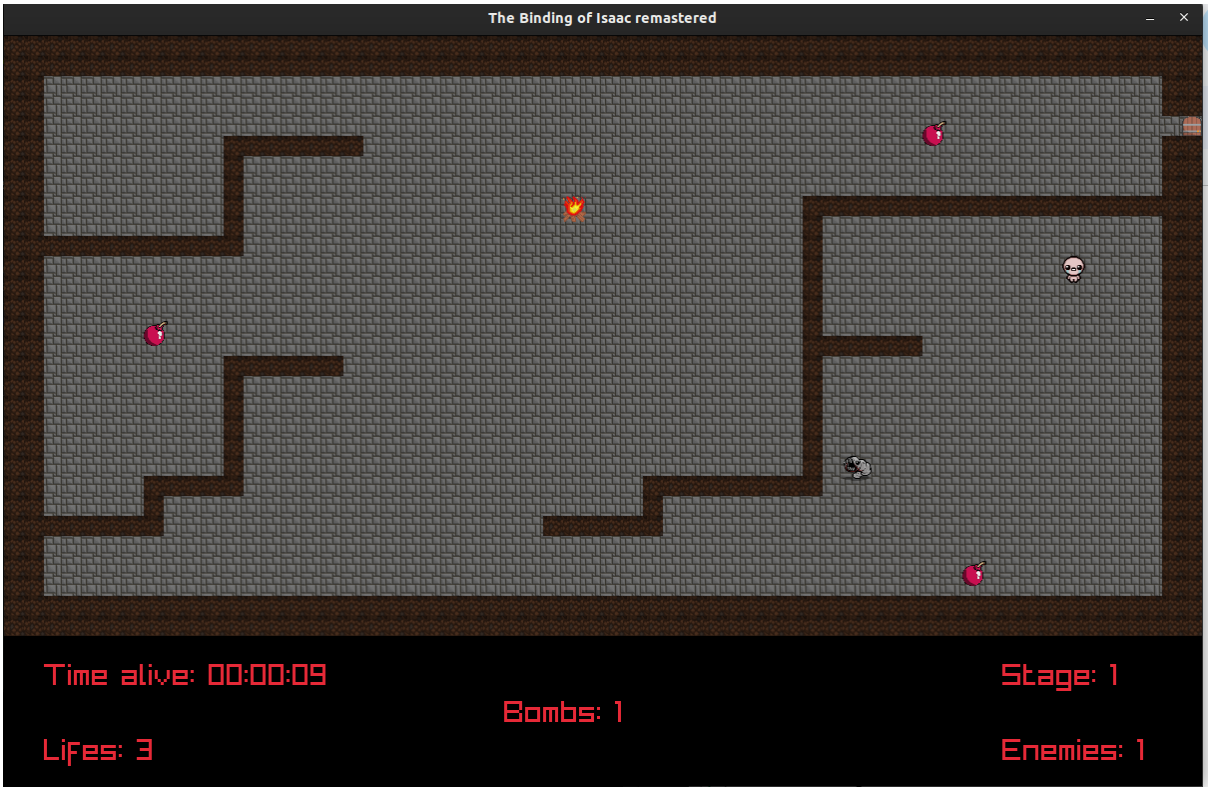

Figura 2 - Jogo em si e barra de informações.

Figura 3 – Menu durante o jogo.

Figura 4 – Mensagem que Isaac morreu mas tem vidas restantes.

Figura 5 – Mensagem que Isaac passou de fase.

Figura 6 – Mensagem de GAME OVER.

Figura 7 – Mensagem de que Isaac completou todas as fases e exibe pontuação.

Conclusão

Esta tarefa permitiu que colocássemos em prática as habilidades técnicas da linguagem C vistas na disciplina de algoritmos e programação. Além disso, nos motivou a usar a plataforma github para controle de versão. O uso da biblioteca raylib permitiu a interface gráfica no desenvolvimento do jogo, o qual parece ser intuitivo para qualquer jogador. O jogo foi entregue sem nenhum bug identificado.

Porto Alegre, 10 de Setembro de 2023.