U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Daniel dos Santos Ferreira (up202108771)

Marco Filipe Gonçalves Vilas Boas (up202108774)

Francisco Miguel Correia Mariano Pinheiro Cardoso (up202108793)

José António Pereira Martins (up202108794)

LCOM 2023

1 - Introdução

1.1 - FireBoy & Watergirl

Figura 1: O Jogo

FireBoy & WaterGirl é um jogo de plataformas com puzzle, 2D, adaptado de um jogo real, lançado em 2009, no âmbito do projeto da Unidade Curricular de Laboratórios de Computadores pelo grupo 1 da turma 2L.EIC02: Daniel Ferreira, Marco Vilas Boas, Francisco Cardoso e José Martins.

Neste jogo, o *FireBoy* não pode andar sobre a água, nem sobre o veneno, mas pode andar em lava. Por sua vez, a *WaterGirl* também não pode caminhar sobre o veneno, porém, ao contrário do *FireBoy*, consegue caminhar sobre a água, e não sobre a lava.

É possível jogar o modo *Single Player* ou *Multiplayer*. Ambos consistem em controlar os dois personagens para estes se ajudarem a concluir os puzzles e chegarem às suas respectivas portas, sendo este o objetivo do jogo.

1.2 - Objetivos

Neste projeto conseguiu-se concluir o esperado e pedido para o mesmo:

- Foram utilizados os **dispositivos** trabalhados nas aulas teórico-práticas, mas também aqueles que apenas se abordam nas aulas teóricas:
    - Timer
    - Rato
    - Teclado
    - Placa Gráfica
    - RTC
    - Uart

- Utilizou-se o **GitLab** como ***Version Control System***, criando branches para a implementação de cada uma das features, de modo a minimizar conflitos enquanto no desenvolvimento de cada parte do jogo, e de commits para conseguirmos manter incrementos funcionais de código guardadas.

- Desenvolveu-se o jogo com a linguagem C para o desenvolvimento deste projeto, estruturando o código em diferentes ficheiros, estando estes em diferentes pastas, de modo a que seja facilmente perceptível onde está cada parte do código. Por exemplo, dentro da pasta **src/maps** encontram-se não apenas os mapas do jogo como também os ficheiros **map.c** e **map.h**

2 - Instruções para o Utilizador

2.1 - Menu Principal

Figura 2: Menu Principal

Figura 3: Opção a ser escolhida

No **Menu Principal** (Figura 2) o utilizador tem 3 opções disponíveis, as quais podem ser acedidas **movendo o Cursor** do rato até à opção e pressionando com o **Botão Esquerdo** em cima da mesma. Para o jogador saber a opção que está a ser escolhida, quando o mesmo coloca o cursor sobre a opção, esta fica envolta de uma borda azul, de modo que o jogador perceba que vai selecionar a opção (Figura 3). As opções disponíveis são as seguintes:

- **Single-Player** - O modo single Player levará o jogador para a Interface do jogo [2.4]. Neste modo o jogador irá jogar apenas no seu pc, controlando os dois personagens, ou pode jogar com outra pessoa, cada um controlando um personagem nas suas devidas teclas de movimento.

- **CO-OP** - CONTINUAR DEPOIS

- **EXIT** - Ao selecionar a última opção o jogo irá encerrar. Também é possível fazer isso através da tecla "**ESC**", apenas neste menu.

2.2 Menu de espera *Multiplayer*

Figura 4: O Menu de espera *MultiPlayer*

Este menu é muito simples e intuitivo, serve apenas para 'mostrar' alguma coisa ao utilizador quando este está à espera da conexão ***multiplayer***. Tem também um botão de **cancel,** como podemos ver na figura, para poder voltar ao menu principal.

2.3 - Movimentação

Existem duas personagens controláveis, **FireBoy** e ***WaterGirl***. Assim, no modo *Single Player* seria necessário dois locais onde os jogadores pudessem controlar no teclado. Devido a esta

implicação decidiu-se adotar um estilo de jogo simples, que fosse fácil jogar, sem teclas extras para outras ações:

- Ambos os personagens são capazes de saltar, de modo a que se possam mover no mapa, uma vez que necessitam de alcançar plataformas mais altas. O *FireBoy* executa esta ação através da tecla **W**, enquanto a *FireGirl* o faz com a tecla relativa à **Seta para Cima**.

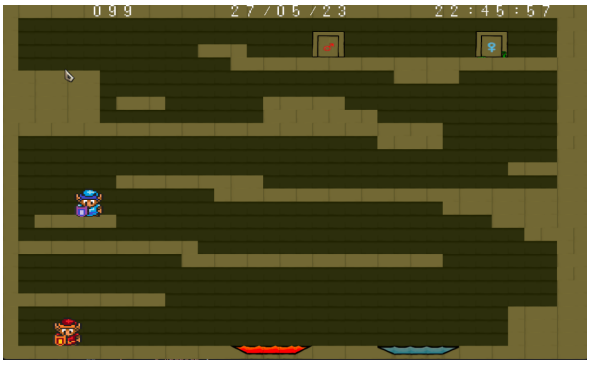

Figura 5: Personagem antes do salto

Figura 6: Personagem após o salto

- Ambos os jogadores são capazes de se mover para a esquerda. O **FireBoy** executa esta ação através da tecla **A**, enquanto a *FireGirl* o faz com a tecla relativa à **Seta para a Esquerda**.

- Ambos os jogadores são capazes de se mover para a direita. O *FireBoy* executa esta ação através da tecla **D**, enquanto a *FireGirl* o faz com a tecla relativa à **Seta para a Direita**.

- Para além disto, ambos os utilizadores podem usar o mesmo rato (algo que não acontece no jogo original) para cortar cordas existentes nos mapas, de modo a que sejam capazes de resolver os puzzles. Para isto, o jogador deve pressionar o **Botão Esquerdo do rato** logo após **mover o Cursor** até a corda.

2.4 - Interface do Jogo

Figura 7: Interface Gráfica do Jogo

Ao longo dos 3 mapas que o utilizador irá encontrar diferentes obstáculos e objetos:

- **Lava** - Para o *FireBoy* isto não será um problema, porém, para a *WaterGirl*, é um obstáculo fatal. Se existir um mínimo contacto entre a lava e a *WaterGirl*, esta irá morrer e os jogadores irão perder o jogo, mesmo que o *FireBoy* já tenha alcançado a sua porta.

- **Água** - Se o *FireBoy* entrar em contacto com este obstáculo, o mesmo irá morrer e ambos os jogadores irão perder o jogo. No entanto, a *WaterGirl* consegue caminhar sobre este obstáculo, de modo a aceder a partes inacessíveis para o *FireBoy*.

- **Veneno** - Assim como as poças de lava e de água, existem poças de veneno. Porém, estas poças são fatais para ambos as personagens. Caso algum das personagens pise no veneno, irá morrer e os jogadores vão perder o jogo.

- **Cordas**: Estes objetos seguram blocos que podem ajudar os jogadores a completar o nível. Basta um dos jogadores chegar perto da corda e **arrastar** o rato por cima dela enquanto **carrega no Botão Esquerdo** para a cortar, fazendo com que o bloco caia..

- **Portas das personagens** - para completar os níveis, as personagens devem alcançar ambas as suas respectivas portas.

2.5 - Menu de Pausa

Figura 8: Menu de Pausa

Como o projeto desenvolvido é um jogo, e um jogo é principalmente uma atividade de lazer, decidiu-se que seria possível os jogadores poderem pausar o jogo carregando na tecla **ESC**. Quando isto acontece, o jogo fica suspenso por trás do menu, e o timer e data e hora também irão parar. No modo multiplayer basta um dos jogadores pausar o jogo e ambos os jogadores serão levados para este menu.

Adicionalmente a permitir ao jogador a possibilidade de pausar o jogo o menu de pausa contém as seguintes opções:

- END - Selecionando esta opção, o jogador irá voltar ao Menu Principal [2.1], e o jogo será terminado. Em caso do jogo estar a ser jogado em modo multiplayer, ambos os jogadores serão levados para o Menu Principal [2.1].

- RETRY - ao selecionar esta opção os jogadores irão recomeçar o nível atual, com as personagens nas suas posições iniciais e o mapa estando no seu formato inicial. Se o jogo estiver a ser jogado em multiplayer, isto acontecerá para ambos os jogadores.

- RESUME - Esta opção retoma o jogo a partir do ponto em que o mesmo foi pausado. No modo multiplayer, basta um dos jogadores selecionar a opção, e o jogo será retomado

Para selecionar uma opção, o jogador deverá clicar com o **Botão Esquerdo** do rato sobre a opção pretendida ou premir a tecla **ESC** do teclado para voltar ao Main Menu.

2.6 - Tela de *Game Over*

Figura 9: Tela de *Game Over*

Como foi dito na secção 2.3, ambos os personagens podem morrer, e, quando isto acontece, os jogadores irão deparar-se com um tela de *Game Over*. Aqui os jogadores terão duas opções de escolha:

- MENU - ao selecionar esta opção o jogo irá voltar ao Main Menu [2.1], e o jogo irá encerrar. Se o jogo estiver a ser jogado em multiplayer e algum dos jogadores selecionar esta opção, ambos os jogadores irão para este menu.

- RETRY - ao selecionar esta opção os jogadores irão recomeçar o nível atual, com as personagens nas suas posições iniciais e o mapa estando no seu formato inicial. Se o jogo estiver a ser jogado em multiplayer, isto acontecerá para ambos os jogadores.

Para selecionar uma opção, o jogador deverá clicar com o **Botão Esquerdo** do *rato* sobre a opção pretendida ou premir a tecla **ESC** do teclado para voltar ao Menu principal.

3 - Estado do Projeto

Foram implementadas todas as funcionalidades apresentadas ao longo das Instruções para o Utilizador [2]. Ao longo de todo o projeto, utilizamos os dispositivos apresentados na seguinte tabela:

Nome do dispositivo	Utilização	Uso de interrupções
Placa Vídeo	Desenhar os menus e o jogo, incluindo todos os seus sprites.	**Não**
Timer	Estabelecer a taxa de atualização do jogo e atualizar o jogo/ecrã de acordo com essa taxa.	**Sim**
Teclado	Mover as personagens e 'voltar atrás' com ESC	**Sim**
Rato	Mover o cursor. Selecionar as opções nos menus. Cortar as cordas dentro do jogo.	**Sim**
RTC	Ler a data e hora do sistema.	**Não**
UART	Permitir o modo multiplayer com dois jogadores.	**Sim**

Tabela 1: Dispositivos utilizados.

3.1- Timer

O Timer é essencialmente utilizado para atualizar o jogo a uma taxa fixa de 60Hz, taxa definida através da configuração da frequência do timer para 60. A cada interrupção do timer o ***game state*** é analisado e atualizado se necessário, assim como a tecla que será atualizada a cada interrupção de maneira a tornar as animações suaves e agradáveis.

3.2 - Teclado

O teclado é fundamentalmente usado no nosso programa para fazer mover as personagens. Desta forma as teclas **WAD** fazem mover o **fireboy,** em que A e D servem para movimentar a personagem para a esquerda e para a direita, respetivamente, e o W serve para o fazer saltar. As setas por sua vez servem para mover a **watergirl** de maneira semelhante a seta para a esquerda e a seta para a direita servem para movimentá-la para a esquerda e para a direita, respetivamente, e a seta para cima serve para a fazer saltar . A tecla **ESC** também é utilizada como funcionalidade de 'voltar atrás'. Se **ESC** for premido no menu principal o programa termina, se for premido durante o estado de jogo este entra em pausa, se premido durante o menu de pausa ou durante a tela de *game over* o jogador será levado para o menu principal. Também de notar que não só usamos os ***make codes*** mas também tiramos proveito dos ***break codes*** para dar ao utilizador uma experiência mais suave:um exemplo seria na utilização de ***break codes*** na movimentação das personagens, onde o movimento é inicializado com um ***make codes*** e terminado com um ***break code*** de maneira a possibilitar que ambos os jogadores se movam ao mesmo tempo, o que seria impossível se utilizássemos apenas make codes.

3.3 - Rato

O rato é usado para manter um cursor visível durante todo o programa, algo que torna todo o jogo muito mais intuitivo para o utilizador. Este cursor serve para escolher as opções nos diversos menus e é também usado durante o jogo: se um jogador estiver perto o suficiente de uma corda, pode passar o rato por cima dela enquanto clica no botão esquerdo para a cortar, fazendo cair a pedra segurada pela corda, o que pode ajudar a completar o nível em questão.

3.4 - Placa Gráfica

A placa gráfica é, claro, a base deste programa, visto que todo o programa corre em modo gráfico. Utilizamos o modo **0x014C**, que tem resolução **1152x864** e utiliza 4 bytes por pixel, 1 byte para transparência, 1 byte para cada componente, ***red***, ***green*** e ***blue***. Os ***building blocks*** do nosso trabalho são os sprites: no menu principal cada componente é um sprite, desde o ***background*** até ao texto das diferentes opções. No modo de jogo cada bloco é um sprite e cada personagem tem vários sprites associados para conseguirmos animar as suas ações. Para desenhar cada um destes sprites recorremos à função ***low-level*** de desenhar apenas um pixel. Além disso, optamos por usar ***double buffering*** neste projeto, isto é, sempre que queremos desenhar algo no ecrã, primeiro colocamos tudo num frame que se encontra em **RAM** e só depois de tudo ser desenhado é que copiamos este frame para o frame que está de facto a ser desenhado (e que está na **VRAM**). Uma outra otimização que fizemos foi desenhar apenas uma única vez o ***background*** do jogo e quando um evento ocorre (por exemplo um personagem move-se) apenas 'redesenhamos' o background na posição prévia da personagem.

3.5 - RTC

O Real Time Clock foi implementado para podermos mostrar a data e hora real no topo do ecrã. Tem as funções necessárias para se começarem a usar interrupções, porém, como quisemos apenas aceder à data e hora do projeto, não usamos interrupção, e simplesmente damos update ao display a cada tick. O

3.6 - UART

O UART foi utilizado para implementar o multiplayer. Foi utilizado para conectar dois computadores, transmitindo informação de um para o outro, com o auxílio de queues e protocolos de comunicação. Foram utilizadas também as interrupções para saber quando há alguma atualização a fazer.

4 - Estrutura do Código

4.1 - **Código: Módulos**

4.1.1 - Placa Gráfica

Este módulo implementa algumas funções de baixo nível, como chamadas à VBE para mudar entre modos de vídeo e de texto ou **system calls** para mapear um espaço de endereçamento do nosso programa para **VRAM.** São também implementadas algumas funções de mais alto nível como desenhar um **XPM** no ecrã, funções estas que recorrem à função base de desenhar (mudar a cor) de um pixel no ecrã. Para isto temos dois ficheiros, graphics.h e graphics.c, que estão no diretório src/drivers/graphics.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.2 - Timer

O módulo do timer foi desenvolvido para controlar a taxa de atualização do jogo, como explicado acima. Utilizamos dois ficheiros: timer.h e timer.c, no diretório src/drivers/timer.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.3 - KBC

O módulo do KBC engloba o código que controla o rato e o teclado. Como ambos os dispositivos partilham o KBC decidimos pela seguinte arquitetura: kbc.h e kbc.c definem os **macros** e têm funções gerais para o KBC, que são usadas tanto pelo rato como pelo teclado. Depois temos também mais dois pares de ficheiros: keyboard.h e keyboard.c implementam funções específicas para o teclado, bem como o seu **interrupt handler,** definindo também uma **struct** auxiliar **keyboard_packet** que encapsula todos os dados úteis sobre o pacote atual do teclado. De modo semelhante temos os ficheiros mouse.h e mouse.c onde definimos **mouse_packet,** outra struct auxiliar que engloba todos os dados úteis do pacote do rato e também a sua posição no ecrã;

temos também funções auxiliares que verificam, por exemplo, se o rato está por cima de um **sprite.** Estes ficheiros estão no diretório src/drivers/kbc.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.4 - RTC

O módulo do RTC implementa o controlador para este dispositivo, o que permite subscrever as suas interrupções, ainda que não o faça no projeto, e ler o 'tempo real', que é posteriormente mostrado no ecrã. Os ficheiros rtc.h e rtc.c encontram-se no diretório src/drivers/rtc.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.5 - UART

Este módulo implementa a porta-série, que permite receber e tratar interrupções feitas por este dispositivo, bem como receber e enviar dados para o computador ao qual estamos ligados. Os ficheiros deste módulo (incluindo a estrutura de dados auxiliar queue) está no diretório src/drivers/serial_port.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.6 - Sprite

Os Sprites foram criados com base nos snippets fornecidos nas aulas teóricas (slides). Um sprite guarda a posição, a velocidade e o mapa de cores da figura, bem como a sua altura e largura. Esta 'classe' implementa apenas as funções básicas para os sprites, como a sua criação, destruição e desenho no ecrã. Os ficheiros sprite.h e sprite.c encontram-se no diretório src/sprites.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.7 - Character

Este módulo é o maior e mais importante módulo do jogo. Representa um das personagens, fireboy e watergirl, e é responsável por animar os seus sprites, controlar os seus movimentos (sejam estes devido a input do utilizador ou devido à física), verificar colisões, verificar se um

dos jogadores 'morreu' ou se ambos os jogadores estão na respetiva porta e passaram assim o nível. Outro problema que este módulo trata é a queda de blocos, visto que estes só podem cair quando um personagem está perto dele. Estes ficheiros estão no diretório src/characters.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.8 - Map

O módulo map, como o nome indica, trata da leitura e construção dos mapas. A cada nível está associado um mapa, que é guardado como um ficheiro de texto no formato map{lvl}.txt, onde cada caractere corresponde a um 'bloco' de construção do mapa. Além disto este módulo também tem uma função para avançar para o próximo nível e para restaurar as cordas de um nível caso estas tenham sido cortadas e se tente reiniciar o nível. Estes ficheiros estão no diretório src/maps.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.9 - Count_down

O módulo count_down foi desenvolvido como um cronómetro para podermos associar tempo aos níveis. Isto é, este módulo representa um contador de tempo que vai sendo diminuído até chegar a 0, e se chegar a 0 e os jogadores ainda não tiverem completado o nível, estes perdem. Este módulo trata da lógica deste contador e de o desenhar no ecrã. Estes ficheiros estão no diretório src/count_down.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.10 - Protocolo de Comunicação

Este módulo trata do protocolo de comunicação usado pela porta-série. Neste protocolo de mensagens o primeiro byte enviado é um caractere identificador do tipo de mensagem (por exemplo, 'c' significa uma mensagem relacionada com a conexão e 'k' significa um pacote do teclado). Aqui implementamos várias funções para verificar o estado da conexão e para enviar mensagens de uma máquina para outra. O código relativo a esta secção está no diretório src/communication.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.1.11 - Game

Este é o 'top layer module' do projeto. Este módulo serve para separar as várias funções que devem ser chamadas para o jogo funcionar e guarda um game_state que faz com que o que é desenhado no ecrã seja diferente de acordo com este estado. Estes ficheiros estão no diretório src.

Contribuição: Daniel - 25%, Francisco - 25%, José - 25%, Marco- 25%

4.2 - Gráficos de Chamadas

A função **proj_main_loop** contém a inicialização do jogo no módulo **game_init**, onde ocorre a subscrição de interrupções importantes, como timer, rato, teclado, UART e RTC, além da inicialização da placa gráfica. Também inicializa e carrega todos os sprites necessários para o decorrer do jogo.

Chama a função **game_loop**, que contém o loop do driver receiver. Essa função verifica o estado atual do jogo e chama as funções para receber dados da UART, caso esteja em modo multiplayer. Também chama as funções de atualização dos jogadores e dos objetos do mapa, e ordena a renderização do ecrã durante as interrupções do timer. Quando ocorrem interrupções de outros periféricos, os respetivos handlers serão chamados.

Por fim, chama a função **game_exit**, que liberta toda a memória alocada pelo projeto e anula todas as subscrições das interrupções anteriormente feitas.

O fluxo de chamadas de funções pode ser observado mais facilmente no Doxygen

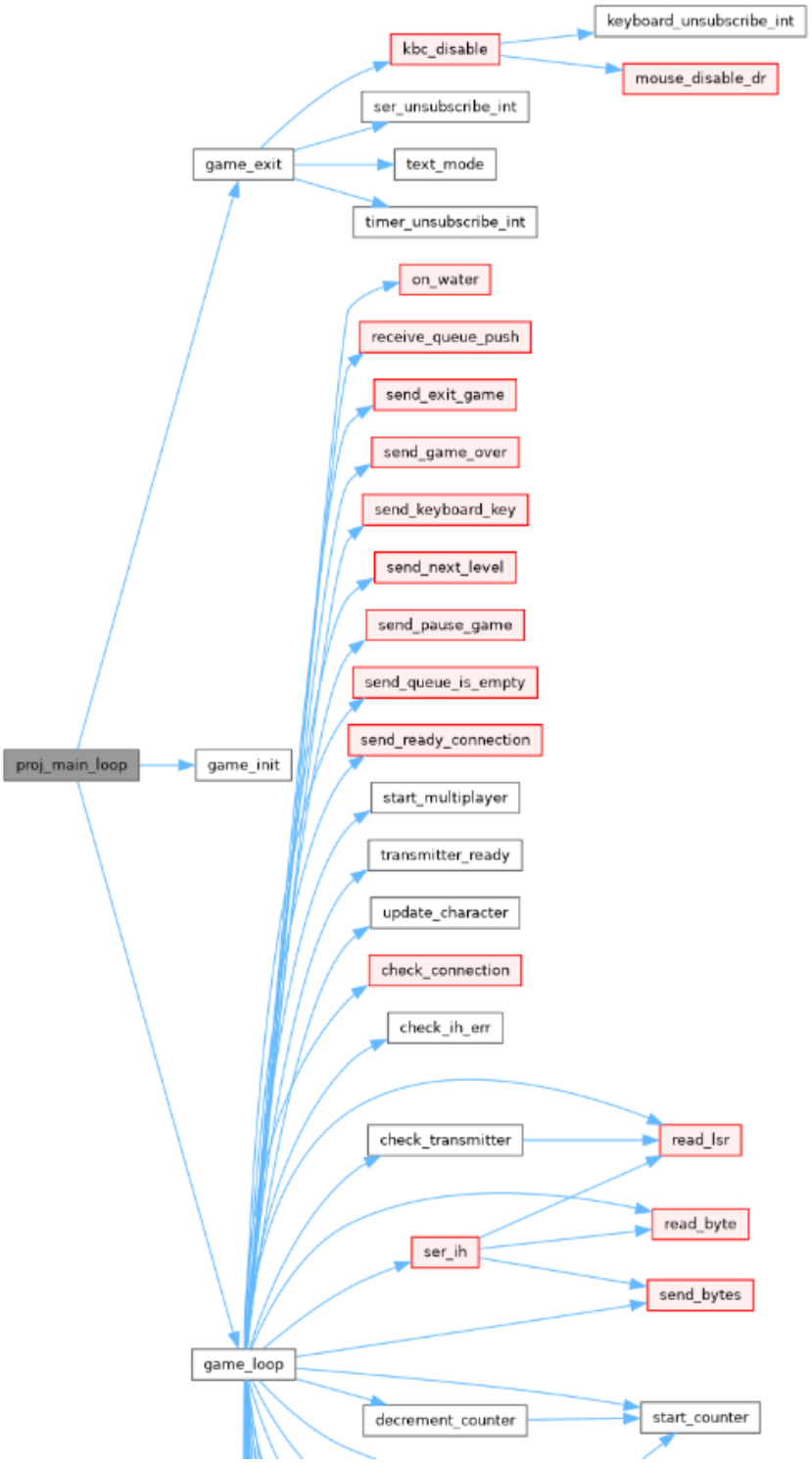

Figura 17.1: Gráfico de chamadas da função

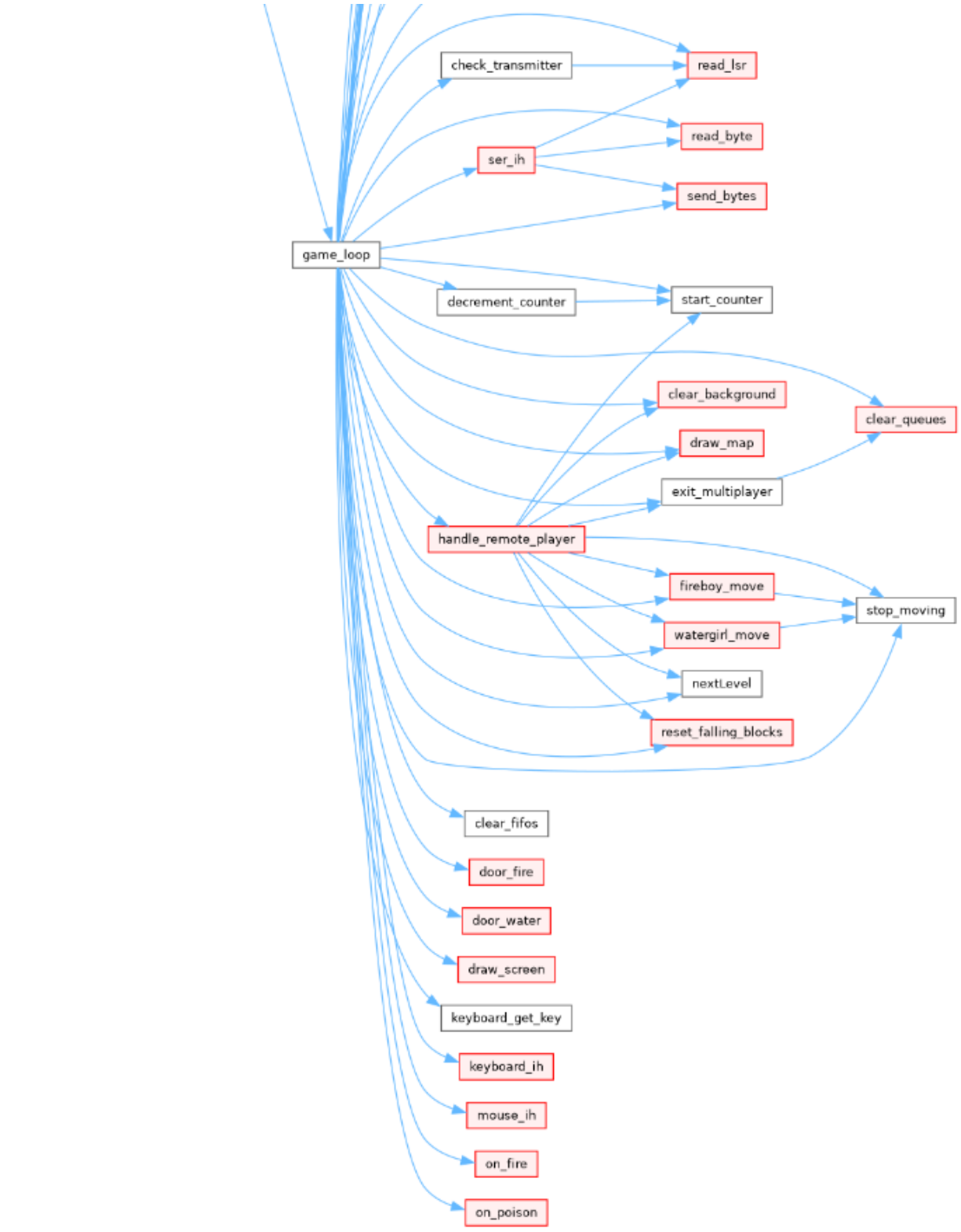

Figura 17.2: Gráfico de chamadas da função

5 - Detalhes de Implementação

5.1 - Implementações de aprendizagem exterior às aulas

5.1.1 - Implementação dos mapas do jogo e do seu desenho

A construção de mapas é algo que pode ser muito facilmente feito através de programas especializados nisso, graças à utilidade que isso tem na indústria de jogos de plataformas hoje em dia. No entanto, para este jogo, foram construídos mapas em ficheiros .txt, atribuindo a cada caracter desses ficheiro um quadrado de 32*32 pixeis na hora de desenhar. Por isso, cada ficheiro tem as dimensões do mapa na primeira linha do ficheiro, seguido de caracteres referentes para cada um desses quadrados, tendo portanto cada ficheiro 28(27+1 de dimensões) linhas e 36 colunas. Cada ficheiro tem o nome mapA.txt sendo A o número do mapa.

A leitura destes mapas é feita caracter a caracter onde cada caractere representa um sprite diferente. Este processo é realizado pela função:

int (draw_map)(Map map)*

Função 1 - src/maps/map.c

Devido a existência de objetos no mapa que são mudados ao longo do jogo, foi necessário a criação de funções que reiniciam esses objetos ao seu estado inicial. Um exemplo disso são os ***falling_blocks*** **,** blocos que inicialmente estão suspensos no ar preços a uma corda que será cortada durante o jogo fazendo assim com que os blocos caiam para outra posição, assim, para reiniciar os blocos as suas posições iniciais é usada a seguinte função:

void (reset_falling_blocks)()

Função 2 - src/maps/map.c

5.1.2 - Movimentação de personagem e Deteção de Colisões

A movimentação das personagens foi um dos passos mais importantes do projeto, tendo sido desenvolvida ao longo de todo o projeto. A função que trata desta movimentação é a seguinte:

void move(Character character)*

Função 3 - src/characters/character.c

1. Primeiro, é verificado se a personagem está no chão ou não usando a função:

wall_down(Character character)*

Função 4 - src/characters/character.c

Esta função verifica se há uma parede embaixo da personagem. Se houver uma parede embaixo, significa que a personagem está no chão.

2. Se a personagem estiver no chão, significa que ela não está a saltar nem a cair.. Neste caso, a velocidade vertical (yspeed) da personagem é definida como zero, e a personagem é movida verticalmente até que esteja exatamente no chão. Isto é feito através de um loop while que verifica se a personagem ainda não está no chão através da função:

int (is_on_ground)(Character character)*

Função 5 - src/characters/character.c

3. Se a personagem não estiver no chão, significa que ela está no ar, a saltar ou a cair. Nesse caso, a velocidade vertical (yspeed) da personagem é ajustada adicionando o valor de GRAVITY para simular a gravidade. Em seguida, a personagem é movida verticalmente de acordo com a sua velocidade vertical atual.

4. Em seguida, é verificado se a personagem se está a mover para a esquerda ou para a direita. Para isto são usadas as funções:

wall_left(Character character)* e *wall_right(Character* character)*

Funções 4 e 5 - src/characters/character.c

5. Se a direção da personagem for LEFT e não houver uma parede à esquerda da personagem, a personagem é movida para a esquerda subtraindo a velocidade horizontal (xspeed) da personagem pela posição X.

6. Da mesma forma, se a direção da personagem for RIGHT e não houver uma parede à direita da personagem (verificado pela função wall_right()), a personagem é movida para a direita adicionando a velocidade horizontal (xspeed) da personagem à posição X.

Esta é a lógica básica de movimentação da personagem. A personagem pode-se mover horizontalmente para a esquerda ou direita desde que não haja uma parede a bloquear o caminho. Além disso, a movimentação vertical é controlada pela gravidade quando a personagem está no ar e pelo contato com o chão quando a personagem está no chão.

As funções wall_right, wall_left e wall_right funcionam verificando se o caracter diretamente baixo da personagem, no mapa atual, são caracteres relativos a blocos, paredes ou as poças de lava ou água.

Outro ponto importante quanto às colisões das personagens são as poças de água, lava e veneno. Estas, na verificação de colisões, contam como 'chão', é feita a verificação entre o tipo do personagem e o tipo de poça noutra função (o fireboy pode andar na lava e a watergirl na água).

5.1.3 - Animações de sprites

A animação de sprites foi algo relativamente desafiante neste projeto, visto que os nossas personagens têm vários 'estados' (parado, correr, saltar) e portanto havia que definir uma maneira de consistentemente animar estas ações. Acabamos por encontrar a seguinte solução: definimos uma struct que guarda a direção na qual a personagem se está a mover de momento (através de um enum), bem como o número de frames que precisam de ocorrer até à próxima animação e o sprite atual. Além disto, guardamos um array com os color maps de quando a personagem se está a movimentar para a esquerda, outro com os maps de quando se está a movimentar para a direita, e outro (que guarda apenas um elemento) para quando a personagem está parado/de frente. Assim, com mais uma variável que guarda o 'índice' atual no array atual de sprites, conseguimos percorrer estes arrays de sprites a cada determinado intervalo de tempo e conseguir o efeito de animação da personagem a correr.

5.2 - Implementações de aprendizagens relativas às aulas

5.2.1 - Double Buffering

Double buffering consiste em construir uma réplica do frame da memória de vídeo, que reside na RAM. Isto permite uma experiência visual muito mais suave devido ao seguinte: em vez de desenharmos os sprites diretamente na VRAM, podemos desenhá-los neste buffer auxiliar da RAM e, depois de desenhar tudo o que queremos, basta copiar de uma vez só este buffer auxiliar para o frame em VRAM através do memcpy(), fazendo com que o ecrã não apareça 'riscado' ou com artefactos. Além disso, implementamos mais uma técnica de otimização que nos surgiu da natural observação: o background do menu e de um nível são estáticos, ou seja, não precisam de ser desenhados mais do que uma vez. Assim o que fazemos é, por exemplo, ao iniciar um nível, desenhamos uma única vez o background no buffer e, posteriormente, sempre que há algum evento, por exemplo o movimento de um personagem, apenas substituímos os pixeis que eram 'ocupados' pela personagem pelos respetivos pixeis do background, otimizando assim imenso a performance (a cada evento, no pior caso, só temos de redesenhar no máximo dois personagens e uns quantos blocos em queda, o que corresponde a cerca de 5% da área do ecrã, o que é uma melhoria tremenda em comparação com desenhar 100% do ecrã a cada evento).

5.2.2 - Protocolo de Comunicação

A comunicação entre dois computadores é feita através da porta série e para tal foi preciso definir um protocolo cuidado de comunicação. Neste protocolo usamos o modelo apresentado nas aulas teóricas do envio de mensagens com uma chave inicial.

Para estabelecer a conexão entre os dois computadores enviamos inicialmente o byte 'c', e de seguida os bytes 0xF9, que significa pedir para dar início à conexão, e o byte 0xFA, que significa que a conexão foi aceite. Quando um dos computadores envia pedido de conexão este fica em 'busy waiting' pelo outro computador até que este envie os bytes necessários, e quando isto acontecer entram ambos no jogo. Um dos computadores pode apenas mover o fireboy e outro apenas a watergirl. De notar que enquanto um dos computadores espera temos um ecrã de espera e um botão para cancelar, para voltar ao menu principal, de modo a não bloquear o programa.

Para transmitir dados do teclado, envia-se primeiro o header 'k' e depois envia-se não o scancode do teclado (até porque isto poderia requerer dois bytes) mas sim o byte já associado ao **enum keyboard_key.** Assim o tratamento do byte do lado que recebe é mais fácil e direto.

Para transmitir a informação de que uma corda foi cortada, o protocolo diz que temos primeiro de enviar o header 'r', e depois dois bytes: um com a posição x e outro com a posição y da corda. De notar que estas coordenadas não representam as coordenadas do ecrã mas sim as coordenadas na grelha do mapa: o mapa tem dimensões 37 x 26 e assim podemos enviar a informação de modo mais compacto.

Finalmente temos mais uma regra, para o caso de uma mudança do estado de jogo. Esta mensagem deve ser enviada com o header 'x' e de seguida é enviado um byte que representa uma mudança, por exemplo **pause** ou **game over.**

5.2.3 Filas UART e FIFO

Para transmissão de dados utilizamos as filas da UART, que são ativadas e limpas quando a inicializamos. Utilizamos também filas FIFO (mais conhecidas por queues) para emular as filas UART em software, de modo a podermos tratar a informação de maneira mais regrada, a cada interrupção do timer. As queues foram implementadas da maneira tradicional com um apontador para o início da fila e um para o fim, mantendo uma variável size que guarda o tamanho atual da fila.

6. Conclusão

6.1. Problemas

Durante a realização do projeto, enfrentamos vários problemas que resultaram em ***features*** planeadas que ficaram incompletas ou em um estado inferior ao planeado. Alguns exemplos desses problemas foram:

- Resetar as cordas sempre que um nível era reiniciado. Uma das funcionalidades planejadas para este projeto foi a presença de blocos suspensos por cordas que caem quando a corda é cortada pelo jogador. Embora essa funcionalidade tenha sido implementada no projeto final, problemas surgiram ao tentar resetar as cordas quando o mapa reiniciava sem ter que recriar o mapa outra vez .Eventualmente chegamos a uma solução , mas esta resultou em um código muito extenso e confuso, dificultando a leitura, compreensão e *debug* do mesmo.

- Perda de informações na porta serial. Uma das funcionalidades do nosso projeto é a possibilidade de dois jogadores jogarem o jogo em computadores diferentes, graças à porta serial. No entanto, devido a um erro que resulta na perda de informações enviadas, a funcionalidade de jogar em dois computadores diferentes torna-se inconsistente.

6.2. Funcionalidades não implementadas

Durante a realização do projeto, várias funcionalidades discutidas e planejadas foram deixadas de fora devido à falta de tempo. Estas incluem:

- Objetos móveis: objetos com os quais as personagens podem interagir, empurrar, pegar, atirar, etc.

- Botões ativadores: botões espalhados pelo mapa que, quando um personagem fica em cima deles, desbloqueiam uma seção do mapa que antes era inacessível.

- Música e efeitos sonoros: músicas de fundo diferentes, dependendo do estado do jogo, e efeitos sonoros para as personagens e objetos do mapa.

- Mais animações: aumentar a quantidade de animações no projeto, incluindo animações de morte, conclusão de nível, etc.

6.3. Conquistas do projeto

Apesar de não termos conseguido concluir tudo o que planeamos para o projeto, alcançamos diversas conquistas durante sua realização, incluindo:

- Porta serial;

- Um jogo semelhante ao original;

- Implementação de novas funcionalidades não presentes no jogo original.

6.4. O que aprendemos:

Durante o projeto, adquirimos conhecimentos valiosos, tais como:

- Utilização de drivers em contextos reais;

- Organização e implementação de um jogo;

- Realização de um projeto em uma linguagem de baixo nível;

- *Debug* em baixo nível.

7 Apêndice

7.1 Anexos

7.1.1 Anexo A - Instruções de Instalação

Para poder instalar o jogo *Fireboy and Watergirl*, deve dirigir-se à pasta **proj/src** e fazer os seguintes comandos.

- make clean
- make depend && make
- lcom_run proj 1|2

7.1.2 Anexo B - Imagens e fontes

- Sprites:
  - https://www.1001games.com/adventure/fireboy-and-watergirl-1-forest-temple