Faculdade de Engenharia da Universidade do Porto

Byte Battles

Projeto Final - LCOM - Turma 14 - Grupo 05

Realizado por:

João Parada (201405280)

Marta Martins (202206369)

Miguel Santos (202008450)

Índice

1. Introdução

Para o nosso projeto, críamos um jogo em 2D chamado Byte Battles, em que o utilizador joga com uma vista de cima para a arena.

O objetivo do nosso jogo é sobreviver o máximo de tempo possível contra os ataques dos inimigos e pontuar o máximo possível, eliminando os inimigos. Os inimigos aparecem em “waves”, e cada “wave” é mais difícil que a anterior, contendo mais e mais perigosos inimigos.

2. Instruções de Utilização

2.1. Main Menu

Ao abrir o jogo, é mostrado o Initial Screen, onde é possível iniciar o jogo. Ao clicar na spacebar aparece o Main Menu, com as seguintes opções START, HELP e EXIT.

2.2. Help Menu

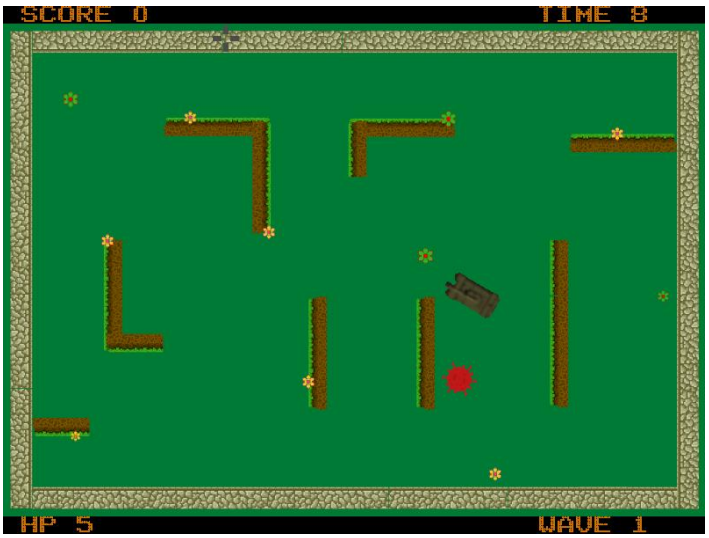
Ao selecionar o botão HELP aparece o HELP MENU com as instruções de jogo e com a opção de voltarmos ao MAIN MENU.

2.3. Highscores

Ao selecionar o botão SCORE no MAIN MENU aparece o menu HIGHSCORES com as classificações dos jogadores que conseguiram jogar o jogo durante o maior tempo possível. (Não implementado)

2.4. Game Arena

Ao selecionar o botão START do MAIN MENU damos início ao jogo e jogamos o máximo de tempo possível até o HP do tanque chegar a 0.



2.5. Pause Menu

Para colocar o jogo em pausa basta clicar na spacebar e aparece o PAUSE MENU.

2.6. Game Over

Quando perdemos, aparece o GAME OVER menu. Com as opções de ver a SCORE, PLAY AGAIN, MAIN MENU e EXIT.

2.7. New Highscore

Quando perdemos se conseguirmos um tempo mais longo que o 1º classificado do menu SCORE aparece o menu NEW HIGHSCORE com parâmetros a preencher (Não implementado).

2.8. Como jogar o jogo

Para movimentar o tanque, o jogador tem que pressionar no teclado as seguintes teclas:

- 'W' para andar para cima;
- 'S' para andar para baixo;
- 'A' para andar para a esquerda;
- 'D' para andar para a direita;
- utilizar o rato para alterar a direção do tanque;
- utilizar o botão esquerdo do rato para disparar.

3. Estado do Projeto

3.1. Tabela de Funcionalidades

Dispositivo	Funcionalidade	Interrupções
Timer	Controlar a frame rate da placa gráfica. Contagem do tempo de jogo.	Sim
Keyboard	Selecionar as várias opções do Menu e mover o tanque/pausar o jogo.	Sim
Mouse	Alterar a direção do tanque, disparar e selecionar as várias opções dos menus.	Sim
Video Card	Display dos menus e do jogo	Não

3.2. Video Card

O ficheiro video_gr.c implementa a configuração e as funcionalidades gráficas do jogo. Logo no começo do programa, a Placa Gráfica é inicializada, pela função vg_init(), no VBE Mode 0x115 (800x600, direct color com 24 bits per pixel (8:8:8),permitindo 2^24 cores diferentes.). O jogo permanece neste modo até ser terminado pelo utilizador, e qualquer texto presente no jogo é desenhado pela utilização de diferentes fonts no formato xpm. Nenhum texto está pre-existente em forma de imagem.

Ao todo, a função vg_init() aloca espaço suficiente para 3 buffers (800x600x3 Bytes), first_buffer,second_buffer e arena_buffer. Utilizamos estes buffers para executar Double buffering via page flipping. O nosso page flipping consiste num quarto apontador, drawing_buffer, que define qual dos buffers (first_buffer ou second_buffer) está a ser desenhado e, consequentemente, qual está a ser exibido utilizando a VBE function 0x07. Em cada frame, o conteúdo de arena_buffer é copiado para o drawing_buffer, evitando assim atrasos causados pelo redesenho da arena.
O drawing buffer alterna entre os dois buffers principais em cada frame, o que acontece a cada 2 interrupções do Timer 0, que está programado por default para efetuar 60 interrupções por segundo. Logo, o nosso jogo corre bastante fluidez a 30 FPS, sem qualquer tipo de “screen tearing” ou presença de artefactos.

Para a renderização dos gráficos do jogo utilizamos as structs “Sprite”(sprite.c) e “AnimSprite”(asprite.c) também adaptados dos slides/trabalho feito para o lab5. Nos casos em que o conteúdo a exibir não é um elemento do jogo dinâmico (i.e. texto e arena), desenhamos o xpm respetivo diretamente para o drawing buffer, sem a utilização de sprites.

Utilizamos sprites animados para representar um tipo de inimigo mais forte e para as explosões causadas pelos disparos. Estes sprites animados são compostos por múltiplas imagens (XPMs), permitindo a criação de sequências animadas. Todos os XPMs usados neste projeto foram gerados através do GIMP. As animações são atualizadas com base em um contador de velocidade, permitindo controlar a taxa de atualização das mesmas. No caso das explosões, o AnimSprite é programado de forma a se autodestruir quando chega ao último XPM do array.

O ficheiro sprite.c é responsável pela criação e gestão dos sprites estáticos no jogo, como o tanque do jogador, o cursor/crosshair e o inimigo mais fraco. Sprites são elementos gráficos que se movem pela tela mas não mudam de aparência durante o movimento. Para criar e manipular esses sprites, utilizamos XPMs que representam as imagens estáticas dos objetos. A posição e a velocidade dos sprites são controladas(sempre que possível utilizando as variáveis “xspeed” e “yspeed” de cada sprite) para garantir que se movam corretamente no ambiente de jogo. Este ficheiro também cuida do desenho dos sprites na tela.

3.3. Keyboard

O keyboard controla praticamente todo o jogo. Nos menus, tal como o mouse, pode ser usado para navegação, usando as setas para cima e para baixo para escolher a opção e a barra de espaço para a confirmar. Na arena é responsável pelos movimentos do tanque e para pausar o jogo. Para mover o tanque são usadas as seguintes teclas: ‘W’ para andar para cima; ‘S’ para andar para baixo; ‘A’ para andar para a esquerda; ‘D’ para andar para a direita. Para pausar o jogo enquanto estamos a jogar basta pressionarmos a barra de espaço e aparece o Pause Menu.

Por cada tecla pressionada, o keyboard controller gera uma interrupção com a informação do make/break code da tecla que foi premida/largada, que é processada de acordo com o estado atual do jogo.

A implementação das configurações, funcionalidades e interrupções do teclado estão no ficheiro keyboard c, que já tinhamos implementado para o lab3. Apenas foram adicionadas macros ao ficheiro i8042.h para cada tecla utilizada, fazendo com que o código fique mais legível.

3.4. Mouse

O mouse, tal como o teclado, funciona em todos os menus para selecionar as opções desejadas, movendo o mouse para a opção desejada e clicando no botão esquerdo para confirmar. O mouse também é utilizado durante o jogo para alterar a direção do tanque e para disparar contra os inimigos ao pressionarmos o botão esquerdo do mesmo. Ao mexermos o mouse durante o jogo também conseguimos controlar a mira.

Por cada movimento ou clique do rato, o controller deste gera uma interrupção com toda a informação do movimento/clique na forma de um "packet", que é processada de acordo com o estado atual do jogo.

A implementação das configurações, funcionalidades e interrupções do mouse estão no ficheiro mouse.c, também praticamente inalterado daquele que implementamos para o lab4.

3.5. RTC

Não implementado.

3.6. UART

Não implementado.

3.7. Funcionalidades não implementadas

Apesar de ser possível a escolha do menu "Highscores", devido à falta de tempo não conseguimos acabar a implementação dos highscores. A parte de cálculo do score de cada jogo está feita, e a estrutura base dos menus também, mas a parte de salvar, ler e exibir um ficheiro (i.e. highscores.txt) não foi implementada.

Também, pela mesma razão, não implementamos a opção/menu "Select Arena" que permitiria o utilizador escolher diferentes arenas (que já estão desenhadas em proj/assets/xpm/arenas/).

Outra funcionalidade pensada inicialmente foi a existência de "Power-ups" que apareceriam aleatoriamente pelo mapa e que o jogador poderia apanhar para aumentar a sua HP/pontuação/velocidade/dano.

No futuro, para além das funcionalidades em falta, poderemos também aumentar a personalização do aumento da dificuldade do jogo, que neste momento aumenta linearmente a cada 5 segundos. Para isso, iríamos introduzir novos tipos de inimigos, e desenhamos o código forma a facilitar essa introdução. No entanto, não tivemos tempo/criatividade para desenhar e animar outros inimigos.

4. Organização/Estrutura do Código

4.1. Graphics Module (20%)

- video_gr.c

Neste módulo, para além da função vg_init() que initializa a placa gráfica no modo correto e aloca memória para os buffers, estão diversas funções utilitárias de desenho de xpms (ex. vg_draw_pixmap()) ou de outras figuras, como o vg_draw_rectangle().

Também está presente a função vg_flip_buffers() que está encarregue de trocar os buffers, como explicado anteriormente.

-sprite.c, asprite.c

Nestes ficheiro estão presentes funções de criação, desenho, animação e destruição de sprites estáticos e animados.

Data Structures:

struct vbe_mode_info_t: Estrutura que armazena informações sobre o modo VBE, incluindo resolução, bits por pixel e endereço base da memória de vídeo.

Buffers de vídeo char* (video_mem, first_buffer, second_buffer, arena_buffer, drawing_buffer): Ponteiros de 8 bits para diferentes áreas de memória utilizadas para renderização e manipulação gráfica.

struct Sprite: Contem toda a informação de um objeto associado a uma imagem XPM.

struct AnimSprite: Contem o apontador para um Sprite e um array de apontadores para cada XPM pertencente à animação, assim como informação sobre como é feita esta animação.

4.2. Keyboard Module (10%)

-keyboard.c

Neste módulo estão presentes as funções que desenvolvemos no Lab3, que nos permitem subscrever e cancelar a subscrição de interrupções, configurar e ler o estado do controlador do teclado e ler e processar os scancodes gerados por cada tecla que é pressionada/largada.

Data Structures:

int hook_id: Utilizado para armazenar o ID do hook usado nas funções de subscrição e remoção de interrupções do teclado.

uint8_t keyboard_data: Armazena o último byte de dados lido do buffer de saída do teclado.

uint8_t kbd_status_byte: Armazena o byte de status lido do controlador de teclado.

bool discard_keyboard_data: Flag que indica se os dados do teclado devem ser descartados devido a um erro de leitura.

bool skip_print: Flag que indica se a impressão do scancode deve ser ignorada, usada principalmente para tratar scancodes com prefixo.

4.3. Mouse Module (10%)

Semelhante ao anterior, neste módulo estão presentes as funções que desenvolvemos no Lab4, que nos permitem subscrever e cancelar a subscrição de interrupções, configurar e ler o estado do KBC e ler e processar os pacotes recebidos por cada movimento ou botão premido.
Para além disso, também contem uma máquina de estados que foi utilizada no Lab4 mas que não é usada no nosso projeto.

Data structures:

int mouse_hook_id: Utilizado para armazenar o ID do hook usado nas funções de subscrição e remoção de interrupções do mouse.

uint8_t mouse_status_byte: Armazena o byte de status lido do KBC relacionado ao mouse.

uint8_t mouse_data: Armazena o último byte de dados lido do buffer de saída do KBC relacionado ao mouse.

bool discard_mouse_data: Flag que indica se os dados do mouse devem ser descartados devido a um erro de leitura.

uint8_t mouse_count: Contador de bytes recebidos de um pacote do mouse.

int x_sum: Soma acumulada das mudanças na coordenada x, usada na máquina de estados.

int y_sum: Soma acumulada das mudanças na coordenada y, usada na máquina de estados.

enum MouseState: Armazena o estado atual da máquina de estados do mouse.

4.4. Timer Module (10%)

Neste módulo estão presentes as funções que desenvolvemos no Lab2. Com estas funções podemos configurar a frequência de qualquer um dos 3 timers(no projeto só utilizamos o Timer 0), subscrever e cancelar a subscrição de interrupções do timer e configurar e ler o estado/conteúdo do timer.

Data structures:

int timer0_hook_id: Armazena o ID do hook usado nas funções de subscrição e remoção de interrupções do timer.

int timer_counter: Contador de interrupções do timer.

4.5. Model Module (10%)

- game_model.c

Este módulo é responsável por gerenciar todos os elementos do jogo, como o tanque do jogador e os inimigos. Inclui funções para criar, atualizar e destruir esses elementos, caso haja colisão entre um inimigo e o tanque ou seja necessário libertar a memória alocada(i.e. fim do jogo).

O módulo controla a posição "real"(através do Sprite), o hp, direção e o tipo de cada elemento existente no jogo.

Data structures:

struct GameUnit: Estrutura base de cada elemento do jogo, contem todos os dados de cada elemento.
enum SpriteType: define se o Sprite pertencente ao elemento é estático ou animado(necessário para se saber como lidar com o Sprite de cada elemento).

struct Enemy: Estrutura específica para os inimigos, para além de herdar todos os dados existentes em GameUnit (através de um apontador para um GameUnit), contem informação sobre o tipo de inimigo(enum EnemyType) e um apontador Enemy* que armazena a referência para o próximo inimigo na linked-list Enemy *enemy_list.
Esta linked-list de inimigos é fundamental para o funcionamento do jogo, porque para além de nos permitir armazenar diferentes tipos de inimigos na mesma estrutura, também nos dá a referência para qualquer inimigo existente no jogo, sendo possível assim eliminar o inimigo específico que foi atingido/atingiu o tanque.

-arena.c

Este ficheiro é uma extensão do anterior, contendo todas as informações relacionadas com a arena, como quais são as coordenadas que são "caminháveis" ou obstáculos. Esta informação é representada como uma grade/matriz bidimensional 800x560.
Também contem a função create_arena() que cria a arena a partir do XPM, desenha-la para o arena_buffer e ao mesmo tempo preenche a grade pixel a pixel, detetando se é

obstáculo ou não a partir da cor do pixel. Se o pixel for da cor ground_color, a posição deste é marcado com 0 e, caso contrário, com 1.

Também estão presentes aqui as funções que tratam da colisão entre um sprite e a arena.

Data structures:

struct Arena: Contém a grade 800x560 e a cor caminhável de cada arena.

4.6. Logic Module (10%)

- **game_logic.c:**

O Logic Module é responsável por gerenciar a lógica principal do jogo, incluindo deteção de colisões entre sprites, movimentação de sprites, cálculo da direção do tanque e a mecânica de disparo. Contém funções para verificar colisões entre sprites, determinar se uma nova posição é válida para movimento, mover sprites para novas posições e calcular a direção do tanque com base na posição do cursor. Este módulo também gere a lógica de disparo, verificando se um tiro atinge um inimigo e, em caso afirmativo, destrói o inimigo e aumenta a pontuação do jogador.

Data Structures:

enum Direction: Enumeração que define as 12 direções possíveis (usando posições do relógio) para o tanque.

4.7. View Module (10%)

- game_view.c

O View Module é responsável pela gestão e exibição da parte visual dos elementos de jogo anteriormente referidos. Este módulo lida com o carregamento dos sprites e das fontes de texto do jogo, a criação de animações, e a renderização dos diversos componentes gráficos, como o tanque, inimigos, explosões, mira e cursor. Ele assegura que todos os elementos sejam desenhados corretamente no screen e atualizados conforme necessário.

Data Structures:

struct Explosion: Estrutura usada para representar explosões no jogo, contendo o AnimSprite para a animação da explosão e um apontador para a próxima explosão na

linked-list de explosões. Esta linked-list tem a mesma utilidade da enemy_list, permitindo-nos identificar corretamente a explosão a ser eliminada, quando esta chega ao fim da sua animação.

4.8. Dispatcher Module (5%)

- dispatcher.c

Este módulo contem as máquinas de estado para o timer, teclado e rato. É responsável por gerenciar o fluxo e a lógica do jogo de acordo com o seu estado atual. Também controla a transição entre os diferentes estados do jogo de acordo com o input do utilizador e coordena as operações de atualização e renderização do jogo. Este módulo assegura que as ações e eventos no jogo sejam tratados corretamente.

Data structures:

enum State: Enumeração de todos os estados possíveis do jogo.

struct GameState: Estrutura que para além de conter o estado atual do jogo, armazena atributos como pontuação, timer, tempo de jogo e dificuldade.

4.9. Menu Module (5%)

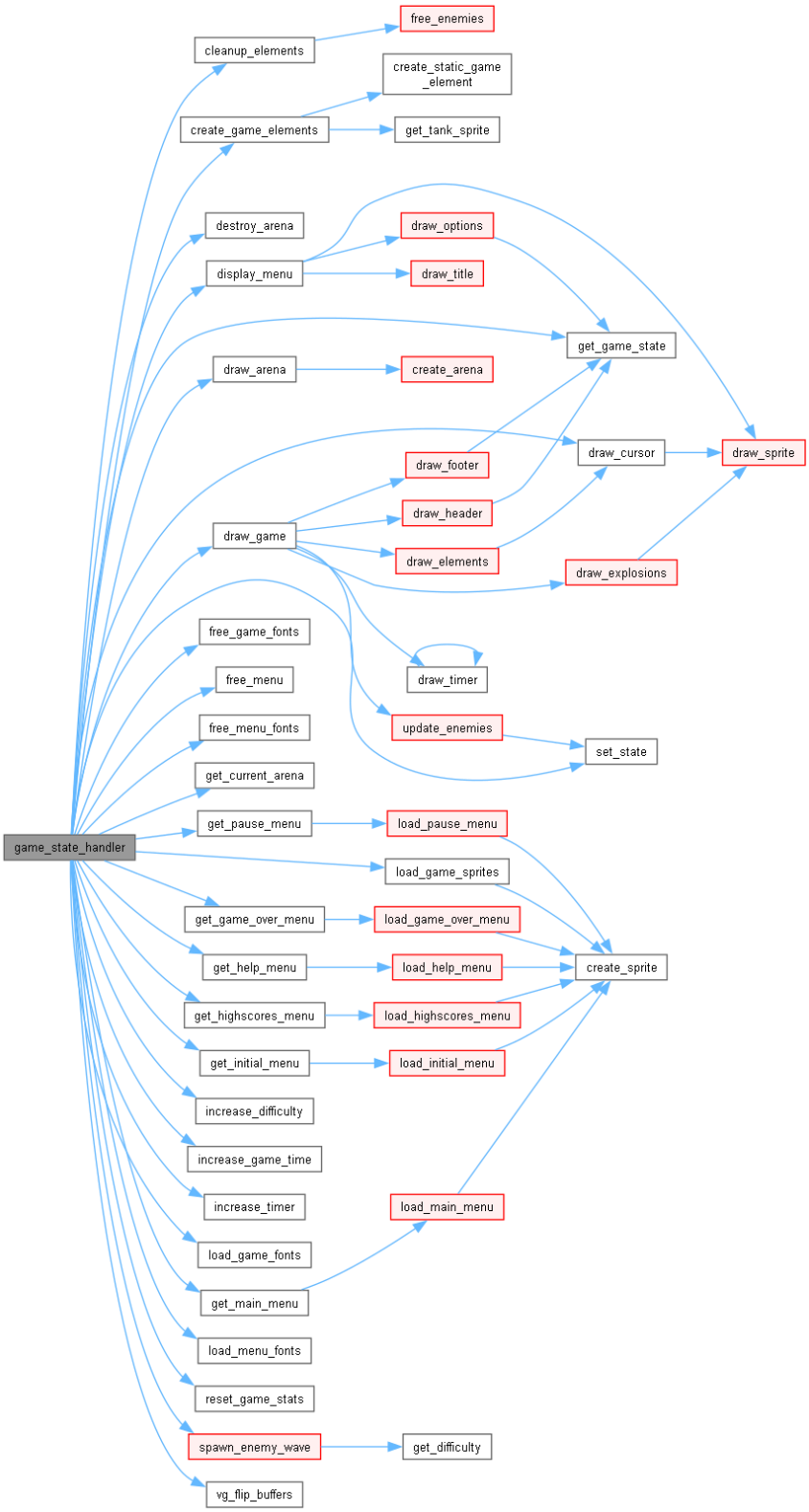
- menu.c

Este módulo define a criação, carregamento e gerenciamento de diferentes menus do jogo, cada um com suas opções e estados. Ele inclui funções para criar menus, carregar fontes gráficas, desenhar títulos e opções no menu, e lidar com interações do usuário como cliques, movimentos do cursor e entradas de teclado. Além disso, há funções específicas para carregar menus pré-definidos como o menu inicial, menu principal, menu de ajuda, e outros.

Data structures:

struct Menu: contem toda a informação pertencente a cada menu, como a posição e tamanho de todas as opções e título, o sprite de fundo, e a funcionalidade de cada opção.

4.10. Function Call Graph



5. Detalhes da Implementação

O facto de termos seguido o padrão model-view-controller (MVC) desde o início, facilitou bastante o desenvolvimento do nosso jogo. Apesar de não estar tão completo como desejamos, a futura implementação de novas funcionalidades, como por exemplo a criação de novos tipos de inimigos, seria bastante simples, onde a maior parte do trabalho requerido seria trabalho artístico.

A primeira dificuldade encontrada foi pensar como fazer com que o tanque se direcionasse em direção à mira. Visto que a struct AnimSprite segue uma ordem específica entre as diferentes imagens, esta abordagem seria impossível, pois o tanque tem que poder ir de uma direção para qualquer outra, dependendo da posição relativa entre este e a mira. Decidimos então criar um array de imagens, cada imagem correspondente a uma certa direção e alterar a imagem do Sprite estático em tempo real. No fim de contas, é uma reimplementação do AnimSprite mas com a possibilidade de se escolher qual imagem é desenhada.

O cálculo da direção na função calculate_tank_direction() também foi algo complexa matematicamente, tivemos que repartir 360º por 12 imagens, em que a imagem 1 aponta para o 1 num relógio tradicional(i.e. a imagem 1 é escolhida quando o tanque está entre os 15º e os 45º para a mira), a 2 para o 2(45º-75º), etc. Após termos implementado o cálculo da direção reparamos que o tanque por vezes ficava a alternar entre 2 imagens indefinidamente, devido ao facto de cada imagem ter largura e comprimento diferente, e, por isso, tivemos que usar um “limite de histerese”, não alterando a imagem a não ser que a diferença do ângulo anterior para o atual seja superior a esse limite.

Em relação à deteção de colisões entre o sprite do tanque e os inimigos, a utilização da linked-list de inimigos facilitou muito a verificação, bastando iterar pela lista a cada frame e comparar as posições+dimensões.

Quanto à colisão com a arena, a nossa abordagem também foi bastante simples, escolhendo para cada arena a sua “ground_color” e ao carregar a arena, todos os pixéis que não forem dessa cor são tratados como obstáculos, e em cada movimento de cada elemento é feita a verificação se este está a “pisar” um obstáculo. Esta abordagem permite-nos adicionar novas arenas muito facilmente, tendo apenas a restrição de só

poder ter 1 cor de espaço caminhável (talvez no futuro se altere esta ground_color para um array de ground_colors, permitindo assim a criação de arenas mais complexas e agradáveis ao olho).

Inicialmente, para a representação de texto no modo gráfico, tinhamos pensado em criar um ficheiro .xpm diferente para cada carácter. Rapidamente nos apercebemos que ter apenas 1 ficheiro .xpm para cada tipo de letra diferente fazia muito mais sentido, e por isso mesmo implementamos a função “draw_character()” que leva como argumento uma font, a posição e o carácter a ser desenhado. Esta calcula a posição do carácter no ficheiro XPM da font e adicioná-la ao apontador da fonte, depois de a multiplicar pelo comprimento de cada carácter.

Apesar de não termos necessidade (visto que não tivemos tempo para implementar os highscores) de receber input de texto do utilizador a partir do teclado, a sua implementação está praticamente feita, pois temos tanto a parte de processamento de scancode para cada carácter como a parte de desenho de cada carácter.

6. Conclusões

O projeto foi uma oportunidade valiosa de aprendizagem, que permitiu ao grupo adquirir conhecimentos práticos na implementação e integração de vários módulos e dispositivos essenciais para o desenvolvimento de um jogo. Durante o desenvolvimento, tivemos a oportunidade de aprimorar as nossas habilidades de programação em C e de compreender a importância da gestão eficiente do tempo.

A implementação bem-sucedida dos diferentes módulos foi crucial para a funcionalidade geral do jogo. A integração destes módulos permitiu um controlo completo do jogo, desde a movimentação e direção do tanque até a deteção de colisões e a animação de explosões. O uso de sprites animados e estáticos, bem como a implementação de diferentes direções para o tanque, enriqueceu a experiência visual e estratégica do jogo. Além disso, a capacidade de desenhar e atualizar elementos como a mira e o cursor aumentou a interatividade e a precisão do controle por parte dos jogadores.

Além das funcionalidades implementadas com sucesso, também foram identificados alguns "likes to have" que poderiam ser adicionados ao jogo em versões futuras. Entre eles, destacam-se os power-ups e o menu de highscores. Essas melhorias sugeridas apresentam potencial para aumentar a complexidade e o desafio do jogo.

Em resumo, o projeto permitiu que o grupo aplicasse os conhecimentos adquiridos durante as aulas práticas, para criar um jogo funcional e oferecer aos jogadores uma experiência de jogo satisfatória.