Faculdade de Engenharia da Universidade do Porto

Mini Tanx

Projeto Final

Laboratório de Computadores

Turma 5 – Grupo 2

Realizado por:

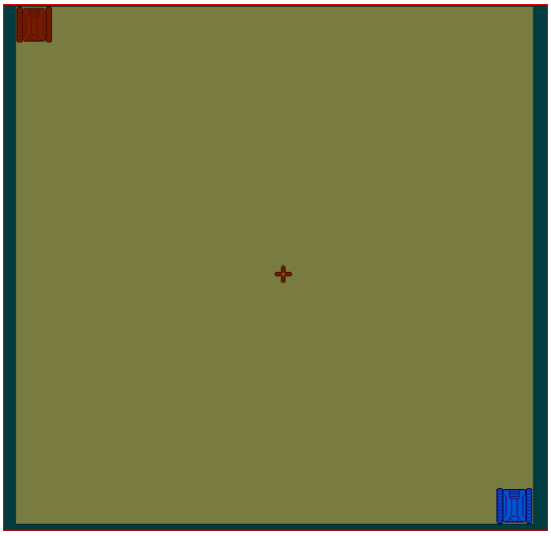
- Gonçalo Pedro Nadais de Pinho (up202108672@fe.up.pt)
- João Afonso Carvalho Viveiros (up202108691@fe.up.pt)
- Artur José Albuquerque Oliveira (up202108663@fe.up.pt)
- Tomás de Campos Sucena de Sequeiros Lopes (up202108701@fe.up.pt)

1. Instruções de utilização do programa

1.1. Modo de Jogo

O jogo consiste num shooter com vista de cima para baixo entre dois tanques controlados por 2 utilizadores no mesmo computador.

Ao entrarmos no modo de jogo os utilizadores são apresentados com este ecrã:

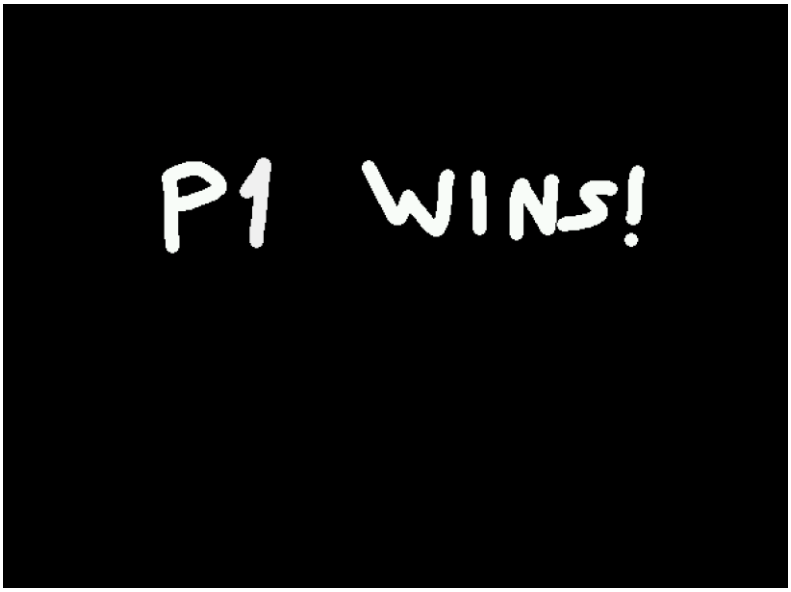


O utilizador consegue ver dois tanques, um deles controlado com WASD e que dispara no rato e o outro controlado com as setas e que dispara no numpad.

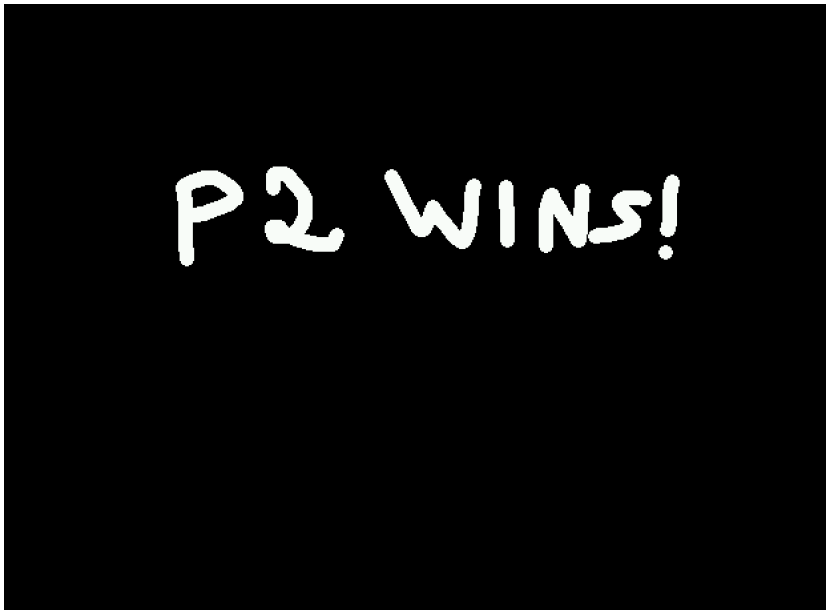
- 1 – Dispara na diagonal inferior esquerda.
- 4 – Dispara na horizontal esquerda.
- 7 – Dispara na diagonal superior esquerda.
- 8 – Dispara na vertical superior.
- 9 – Dispara na diagonal superior direita.
- 6 – Dispara na horizontal direita.
- 3 – Dispara na diagonal inferior direita.
- 2 – Dispara na vertical inferior.

Quando um dos tanques é atingido por uma bala do outro, o jogo para durante 4 segundos, reiniciando após este tempo.

No caso do Jogador 1 ganhar aparece o ecrã fica neste estado:



No caso do Jogador 2 ganhar fica neste estado:



Para sair do jogo basta carregar no ESC.

2. Estado do Projeto

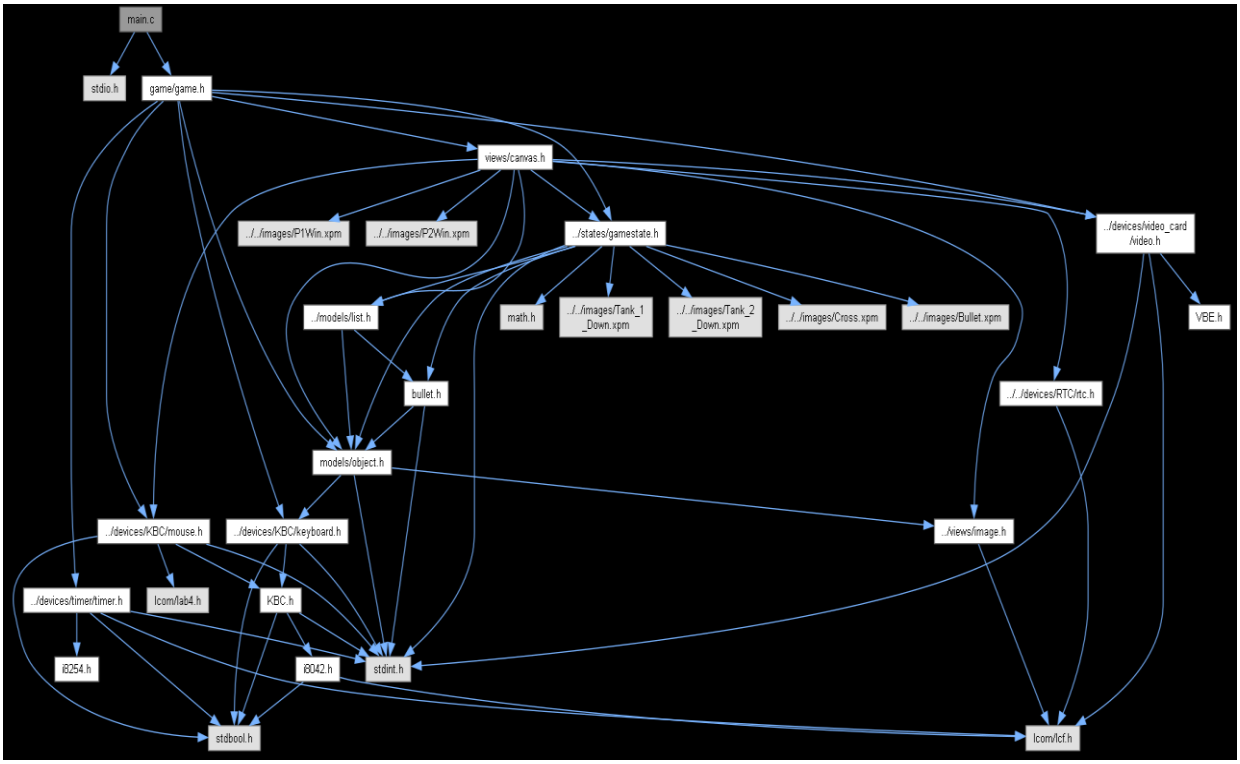
2.1. Tabela de Dispositivos

Dispositivo	Propósito	INT
Teclado	Movimento do Tanque e disparo de balas	**Y**
Timer	Controlar a *frame rate*	**Y**
Mouse	Navegar o menu e disparar balas	**Y**
Placa Gráfica	Desenhar o menu e o jogo	**N**
RTC	Para obter a hora do dia, mudando o ambiente do jogo, consoante	**N**

2.2. Tabela de Funcionalidade

Funcionalidades	Dispositivos	Estado de Implementação
Navegação pelo menu	Mouse	Completo
Movimento	Teclado	Completo
Disparar	Mouse e Teclado	Completo

2.3. Grafo de chamada de Funções



2.4. Teclado

Neste jogo usamos os *interrupts* que provêm do KBC para ler o input do jogador, movendo o tanque ou realizando outra ação consoante esse input.

Para lermos os inputs provenientes do KBC, utilizamos a função: `void (kbd_get_scancode)(kbd_data_t* kbd_data, uint32_t wait_ticks)`

que, num *interrupt*, vai tentar buscar um *scancode* ao output-buffer do KBE.

Para processarmos os inputs provenientes da função anterior utilizamos a função:

`void process_scancode(GameState state, kbd_data_t data)`

que dado uma *struct kbd_data_t* contendo dados proveniente do KBD, altera os objetos dentro da *struct GameState* como, por exemplo, os jogadores.

2.5. Timer

Neste jogo utilizamos os *interrupts* provenientes do timer para controlar a *frame rate* do jogo, mais especificamente, usamos o timer para manter o jogo a 60 *frames* por segundo e para controlar quando uma coisa é desenhada.

A cada *interrupt* do timer utilizamos esta função:

`int (canvas_refresh)(GameState* game)`

para desenharmos no buffer o que vem na *frame* a seguir, considerando os inputs já processados do KBD, e a função:

`void gameStep(GameState *state)`

para processar as balas existentes, se elas colidem com algo e para atualizar da posição delas.

2.6. Mouse

Neste jogo utilizamos os *interrupts* provenientes do Mouse para obtermos a posição do mesmo e atualizarmos a posição dele no ecrã e também o usamos para disparar balas.

Utilizamos a função:

void (mouse_get_data)(mouse_data_t* mouse_data, uint32_t wait_ticks) para buscar a data presente no *output buffer* do KBC, verificando antes se estes dados provêm do rato e a função:

void mouse_parse_packet(mouse_data_t* mouse_data) para darmos analisarmos os dados provenientes da função anterior.

Com os dados passados por esta função, conseguimos atualizar a posição do rato no ecrã e verificar se ele está a premir algum botão.

2.7. Placa Gráfica

Neste jogo utilizamos a placa gráfica no modo 0x11A, com uma resolução de 1280x1024, bem como com um modo de cor direto no qual vermelho tem 5 bits, verde tem 6 bits e azul tem 5 bits, ou seja, tem este formato 5:6:5.

Utilizamos *double-buffering*, com o qual copiamos a informação de um buffer para outro.

Como foi mencionado no *timer*, utilizamos o canvas_refresh para atualizar o que estava desenhado no ecrã. Em termos mais aplicados, atualizamos o *buffer* que não está ligado ao ecrã. A fim de atualizarmos esse buffer, chamamos a função:

int video_switch(), para fazer a copia do buffer atualizado para o buffer do ecrã.

Como temos objetos que se movem (os tanques, por exemplo) temos deteção de colisões para verificar se existe colisões entre esses objetos, por exemplo, entre tanques e paredes, balas e paredes, etc…

Essa deteção é feita através da função:

bool checkCollisions(Object *obj1, Object* obj2), a qual verifica se um objeto está dentro do outro com recurso da função:

bool pointInObject(Object *obj, uint16_t x, uint16_t y) que, verifica se um dado ponto está dentro de um dado objeto.

2.8. RTC

Neste jogo utilizamos RTC para fazermos uma mudança ao cenário, conforme a hora do dia.

Utiliza-se a função:

int getRTCOutput(uint8_t cmd, uint8_t *output) para ir buscar as horas. Se o seu resultado for maior que 21, mudamos o cenário do jogo, mudando o cenário através da função:

int (nightTransform)(uint32_t* color), que altera as cores do jogo para ficarem num tom mais escuro.

3. Estrutura/Organização de código

3.1. Gamestate.c – 15%

Neste ficheiro definimos um novo *data type GameState* que contém todos os dados pertinentes ao jogo, por exemplo, os dois jogadores, a lista de paredes, a lista de balas, entre outros. Definimos, também, funções para a criação do *GameState* e alteração do mesmo.

Contribuidores:

- João Afonso Carvalho Viveiros
- Artur José Albuquerque Oliveira

3.2. Image.c – 5%

Neste ficheiro definimos um novo *data type Image*, constituído pela informação relevante a uma imagem: se é um sprite ou uma forma, o tamanho dela e as cores dela. Também definimos funções para criar este *data type* consoante se queremos criar um *sprite* ou uma forma.

Contribuidores:

- Tomás de Campos Sucena de Sequeiros Lopes

3.3. Canvas.c – 7%

Utilizamos este ficheiro para tudo o que tenha a ver com desenhar no ecrã. É neste ficheiro que desenhamos tudo, sejam os tanques, as paredes, as balas, *a crosshair* do rato e onde se invoca a função para copiar os conteúdos de um buffer para o outro.

Contribuidores:

- Tomás de Campos Sucena de Sequeiros Lopes
- João Afonso Carvalho Viveiros

3.4. Object.c - 8%

Neste ficheiro definimos um novo *data type Object* que consiste na informação relevante a um objeto, seja ele um tanque ou uma parede. Também se definiu funções para a sua criação, de forma a mudar as informações relevantes a um objeto, por exemplo, a sua posição, e para verificar colisões com outros objetos.

Contribuidores:

- Tomás de Campos Sucena de Sequeiros Lopes
- Gonçalo Pedro Nadais de Pinho
- João Afonso Carvalho Viveiros

3.5. List.c - 7%

Neste ficheiro definimos um novo *data type List* que consiste numa *double linked list* e *ListElement* que , através do *void**, pode conter qualquer tipo de informação. Também se definiu funções para a criação desta lista e para a remoção dela, para retirar elementos desta lista, tanto na *Front* como na *Back* e para adicionar elementos à lista.

Contribuidores:

- Tomás de Campos Sucena de Sequeiros Lopes
- João Afonso Carvalho Viveiros

3.6. Bullet.c – 6%

Neste ficheiro definimos um novo *data type Bullet* que contem toda a informação relevante a uma bala. Também definimos funções para eliminar uma bala e para causar o ricochete de uma bala, quando há colisões com paredes.

Contribuidores:

- João Afonso Carvalho Viveiros
- Artur José Albuquerque Oliveira

3.7. Game.c – 10%

Neste ficheiro definimos funções para controlar o jogo, uma função para iniciar o jogo, outra função para o ciclo do jogo, que consiste no *interrupt loop*, e uma função para parar o jogo, que consiste em desalocar memória previamente alocada.

Contribuidores:

- Tomás de Campos Sucena de Sequeiros Lopes
- Gonçalo Pedro Nadais de Pinho
- João Afonso Carvalho Viveiros
- Artur José Albuquerque Oliveira

3.8. Keyboard.c – 9%

Neste ficheiro criamos o driver para o KBD, criando funções para ler *scancodes* provenientes do *output buffer*, funções para verificar o status do KBC e funções para ativar e desativar *interrupts*.

Contribuidores:

- Gonçalo Pedro Nadais de Pinho
- Tomás de Campos Sucena de Sequeiros Lopes

3.9. KBC.c – 2%

Neste ficheiro definimos funções auxiliares que são usadas em keyboard.c, por exemplo, uma função para ir buscar o status do KBC e uma função escrever comandos para o KBC.

Contribuidores:

- Gonçalo Pedro Nadais de Pinho
- Tomás de Campos Sucena de Sequeiros Lopes

3.10. Mouse.c - 8%

Neste ficheiro criamos o driver para o Mouse, criando funções para ler os dados presentes no *output buffer* do KBC, funções para ativar e desativar *interrupts* do *Mouse* e uma função para gerir a informação proveniente do output buffer.

Contribuidores:

- Artur José Albuquerque Oliveira

3.11. Rtc.c – 3%

Neste ficheiro criamos o *driver* para o RTC (Real Time Clock), criando funções para ler o *output* do RTC e funções para verificar e converter para binário caso necessário.

Contribuidores:

- João Afonso Carvalho Viveiros

3.12. Timer.c – 4%

Neste ficheiro criamos o driver para o *Timer*, criando uma função para criar a *control word*, outra para enviar essa *control word* para o registo de controlo do *timer*, outra para mudar a frequência do *timer* e outras para ativar e desativar os *interrupts* do *timer*.

Contribuidores:

- Tomás de Campos Sucena de Sequeiros Lopes

3.13. Vídeo.c – 10%

Neste ficheiro criamos o driver da placa gráfica, utilizando o VBE, onde criamos funções para buscar as informações sobre um dado modo de vídeo, uma função para pôr o minix num dado modo gráfico e outra para meter em modo de texto, na memória.

Contribuidores:

- Gonçalo Pedro Nadais de Pinho
- Tomás de Campos Sucena de Sequeiros Lopes

3.14. Utils.c – 1%

Neste ficheiro definimos funções genericamente úteis, como uma função para ir buscar o *most significant byte*, outra para ir o *least significant byte* e uma função para invocar um *sys_inb* que devolve data com 8 *bits*.

Contribuidores:

- Gonçalo Pedro Nadais de Pinho
- Tomás de Campos Sucena de Sequeiros Lopes

3.15. Main.c – 5%

Neste ficheiro define-se a função chamada no terminal do *minix*. A função cria o jogo e chama o ciclo de jogo onde o jogo se situa.

Contribuidores:

- Gonçalo Pedro Nadais de Pinho
- Tomás de Campos Sucena de Sequeiros Lopes

4. Detalhes de Implementação

Ao elaborarmos o projeto, procurámos implementar diversos conceitos distintos ou aplicados de forma diferente dos tópicos abordados nas aulas. Entre eles estão incluídos o Real Time Clock e deteção de colisões.

A implementação do Real Time Clock procura aumentar a imersão do jogo ao mudar o ambiente conforme a hora real. Uma vez que este dispositivo não foi estudado nas aulas laboratoriais, mostrou ser difícil de implementar, pelo que recorremos a uma maior pesquisa e auxílio de colegas de outros grupos.

As colisões foram uma implementação essencial para o ciclo de jogo, pelo que são elas que ditam o vencedor do jogo. A sua implementação foi auxiliada maioritariamente pela *struct Object,* a qual simplificou os cálculos necessários.

A implementação dos dispositivos já abordados nas aulas procuraram simplificar a lógica do jogo, permitindo qualquer colega do grupo que estivesse menos familiarizado com um certo dispositivo a trabalhar com ele de uma forma simples e eficiente.

Todos estes aspetos levaram a que fossemos capazes de criar, eficientemente, um projeto completo e que procura aplicar tudo o que a unidade curricular tem para oferecer.

5. Conclusões

Neste projeto não conseguimos fazer totalmente o previsto na sugestão inicial, isto pois falhámos a implementar a porta-série (UART). Devido a isto tivemos de optar por uma aproximação local ao multijogador. Para além disto, não nos foi possível implementar o menu, devido a problemas de memória, os quais não conseguimos identificar a causa a tempo da entrega.

Depressa nos apercebemos da variedade de conteúdo que nos seria possível incluir no projeto, contudo, devido à falta de tempo e não estar previsto na proposta inicial, acabámos por não incluir. Isto incluiria diferentes modos de jogo bem como uma maior variedade de mecânicas.

Apesar disto, conseguimos desenvolver a parte mais importante daquilo que foi proposto, inclusive implementar o Real Time Clock, mesmo sem termos praticado com o dispositivo nas aulas laboratoriais. Ainda conseguimos entender de que forma a combinação dos dispositivos abordados nas aulas atuam harmoniosamente num projeto.