Faculdade de Engenharia da Universidade do Porto

Projeto de laboratório de computadores
Turma 4
Grupo 5

Aerial Ambush

Alexandre Costa : 202207499
Bernardo Costa : 202207579

Índice

1. Instruções de utilização do jogo

1.1. Menu

Fig. 1 - Menu

Ao abrir o jogo, o utilizador encontra a página do menu. Aqui ele pode escolher entre jogar uma partida solo (sozinho), uma partida multiplayer (com outro utilizador noutra máquina virtual), ver a página com os melhores scores já obtidos no jogo e encerrar o jogo. O utilizador pode realizar a sua escolha através do rato com um clique no botão esquerdo em cima do botão desejado na tela.

1.2. Top Scores

Fig. 2 - Scores

Aqui o utilizador pode observar as nove melhores pontuações obtidas no jogo juntamente com a data (dia e hora) em que estes aconteceram. Caso ainda não haja tempos obtidos, os espaços estarão vazios.

A tecla ESC permite voltar ao menu.

1.3. Multiplayer

Fig. 3 - Multiplayer

Neste modo, o usuário pode jogar com outra máquina virtual. O usuário deve escolher entre o player 1 ou 2. Assim que o fizer, o jogador escolhido ficará verde, tal como é possível observar na imagem. Tal acontecimento será visível, também, na outra tela. Logo, para o jogo iniciar, o jogador que não está a verde deve ser escolhido, visto que o outro já foi escolhido pelo outro jogador. Escolhendo o outro jogador, o jogo em conjunto iniciará.

Utilizar a tecla ESC para voltar ao menu.

1.4. GamePlay

Fig. 4 - GamePlay

Dentro do gameplay, o jogador deve afastar-se das bombas quando estas explodem no chão, pois perde uma vida ao passar por uma que esteja a explodir. A tecla ‘A’ move o jogador para a esquerda, a tecla ‘D’ move-o para a direita e o ‘SPACE’ fá-lo saltar. Os aviões aparecem tanto da esquerda como da direita. O jogador pode disparar através do botão esquerdo do rato, e, caso acerte num avião, ganha pontuação extra.

1.5. GamePlay Multiplayer

Fig. 5 - GamePlay Multiplayer

Este modo é bastante parecido com o solo. Porém, os jogadores podem unir-se para combater a ameaça aérea.

1.6. GamePlay Lost

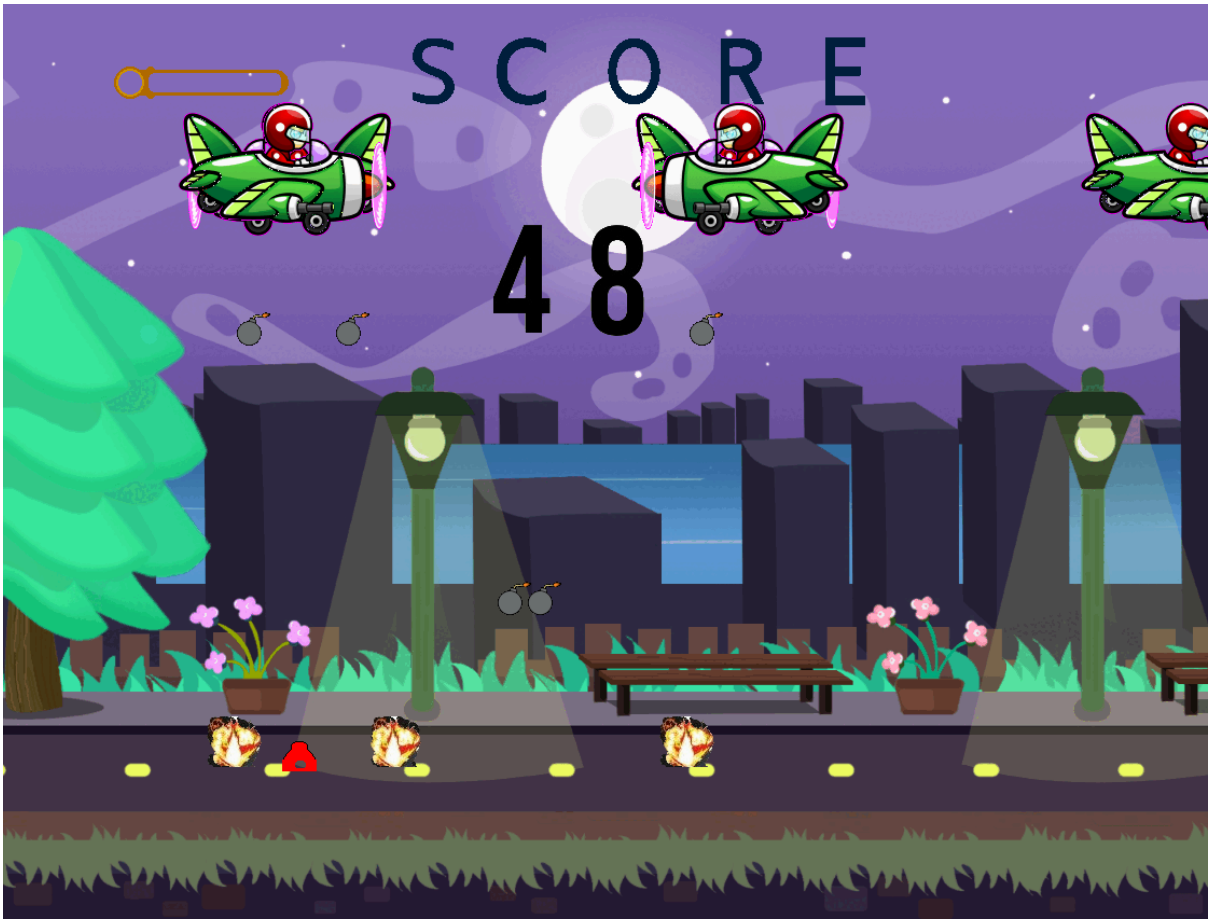


Fig. 6 - GamePlay Lost

Quando o jogador perde todas as vidas, o jogo acaba com a pontuação apresentada. No caso de bater recordes, aparece uma mensagem na tela. Tal recorde deve agora ser visível na página dos ‘Top Scores’. Ao final de uns segundos, o jogo volta ao menu inicial, onde o jogador pode começar outro jogo de novo.

2.Estado do projeto

2.1. Tabela Dispositivos

Dispositivo	Uso	Interrupção
Timer	Score, sprites, loop do jogo, criar bombas e aviões, controlar tempo entre acontecimentos	Sim
Keyboard	Mover player, retroceder nas páginas	Sim
Mouse	Movimento do rato virtual, clicar em botões, disparar	Sim
Video card	Display na tela dos menus e tela de jogo, tal como todos os objetos associados	Não
RTC	Guardar a data dos Top Scores	Não
Serial Port	Possibilita a gameplay conjunta entre players	Sim

2.2. Video card

Estamos a usar o modo 0x14C com a resolução 1152 x 864, 4 bytes de cores por pixel no modo direto, o que permite uma variação de 2^32 cores.

Através do vídeo, conseguimos mostrar a nossa interface de backgrounds(menu, score page, multiplayer page, gameplay background) que é carregada para a nossa classe 'Sprite', que guarda o mapa das cores, tal como a sua posição, altura e largura. Apesar destes últimos parâmetros não fazerem diferença para backgrounds, eles são úteis para objetos, que também são carregados como sprites, e, no caso de objetos que se movem (com velocidade horizontal e/ou vertical), são guardados numa classe 'Object' que possui o 'Sprite' e a velocidade. Os Object estão sujeitos a colisões e alguns deles (aviões, bombas…) possuem ainda pequenas animações causadas por trocas rápidas de sprites. Essas animações são possíveis seja ao utilizar a função void drawMultipleSprite(int index, int wXpm, int hXpm, Sprite *sp), que pinta no buffer apenas uma parte do xpm (útil quando um xpm possui várias sprites do objeto), ou no uso do 'draw' normal, mas trocando o mapa de cores da Sprite por um novo (void changeSprite(Sprite *sp, xpm_map_txpm)). Para criar essas classes, usamos a função Sprite *create_sprite(xpm_map_t xpm, int x, int y), que cria a sprite tendo em conta o xpm que lhe é fornecido e a posição inicial. Para gerar os xpm que temos no projeto, usamos conversores online de png/jpg para o formato pretendido.

De modo a suavizar a experiência de jogo, implementamos double buffering junto com page flipping. Na função int rotateBuffer(), realizamos a troca entre os dois buffers, logo após desenhar tudo no buffer que está por trás e prestes a entrar. Nesta função, fazemos a chamada à função 7 da VBE, porém, com vista à eficiência, optamos por usar apenas o set display, visto que podemos substituir o get display por uma variável que nos indica qual o buffer que está a ser usado no momento.

O código referente a este dispositivo encontra-se no ficheiro graphic.c que está dentro da pasta dos ‘devices’.

2.3. Keyboard

O teclado é usado para retroceder nas páginas do menu e para controlar o player.

Nas páginas multiplayer e top scores, podemos retroceder para o menu principal utilizando a tecla ESC. A tecla A move para a esquerda, a tecla D move para a direita e o SPACE faz o jogador saltar. A função void (manageInputKeyboard)(Object *player, uint8_t Kscancode, GameState *gs) controla seja o jogador tendo em conta o scancode obtido pelo keyboard, esta função encontra-se na classe controller.c.

O código referente a este dispositivo encontra-se no ficheiro keyboard.c que está dentro da pasta dos devices.

2.4. Mouse

Quanto ao rato, é possível mover pelo menu, tal como é visível na figura 1, de modo a clicar nos botões aí disponíveis.

O rato virtual é uma Sprite, que depois de receber do mouse interrupt a sua variação de posição, atualiza também a posição da Sprite, dando desta maneira a sensação de que temos o nosso rato realmente dentro do jogo. Esta funcionalidade está na função void (mouseMovement)(). Na função void (leftMouseClick)(), realizamos ações quando o botão esquerdo do rato é clicado tendo em conta o estado do jogo em que estamos, como clicar nos botões do menu e disparar uma bala quando estamos no modo de gameplay.

O código referente a este dispositivo encontra-se no ficheiro mouse.c que está dentro da pasta dos devices.

2.5. Timer

O timer é muito usado para o controlo de animações e de tempo. Quando o jogador perde, ficamos na tela de gameover 3 segundos antes de o jogo voltar ao menu automaticamente. Outros controlos de tempo são também a criação de aviões, o tempo de destruição de bombas a partir do contacto com o chão, entre outros. Partindo dessa mesma lógica, também controlamos a animação de sprites, que passando um certo número de frames faz com que o objeto troque a sua sprite para outra - dando assim a sensação de animação. Essas animações podem ser vistas nos aviões e nas bombas. O timer está a ser usado numa frequência de 60, ou seja 60 frames são atualizados a cada segundo.

O código referente a este dispositivo encontra-se no ficheiro timer.c que está dentro da pasta dos devices.

2.6. RTC

Optamos por utilizar o real time clock no projeto apenas para leitura da data e hora em momentos pontuais. No nosso caso, um dos objetivos foi guardar as datas das melhores pontuações.

O código referente a este dispositivo encontra-se no ficheiro ‘rtc.c’, na pasta “devices”.

De modo a mostrar o tempo (e aproveitando para mostrar a pontuação), decidimos criar uma classe ‘Score’, na qual estão todas as funções referentes a guardar as pontuações e as respectivas datas para serem imprimidas na página dos ‘Top Scores’, e noutras ocasiões também.

Há três ocasiões em que a pontuação é apresentada: no estado de jogo “OVER” (void drawScoreGameOver(int score), quando o jogo acaba), “PLAY” (void drawScoreGameOver(), a pontuação aumenta e é apresentada durante o jogo) e “SCORE” (void drawScoreGameScore(), como refere o nome, na página das maiores pontuações).

No ficheiro ‘Bomb.c’, foram adicionadas dentro de algumas funções duas instâncias de ‘update_score(int s)’, que incrementa a variável global “finalScore” no valor de “s” durante o jogo. O jogador recebe 1 ponto por cada bomba explodida e 5 por cada avião abatido.

As três funções principais de desenho usam funções auxiliares (por exemplo, uma função que desenha na tela uma palavra ‘int drawString(char string[], int size, int x, int y)’, e outra que imprime um número ‘void drawNumber(int number, int x, int y)’), que essencialmente fazem uso da função ‘void drawMultipleSprite(int index, int wXpm, int hXpm, Sprite* sp)’ - num ficheiro xpm com várias imagens separadas (como é o caso dos números e das letras), permite apenas selecionar o desenho de uma, num momento desejado.

2.7. Serial Port

Tendo como objetivo implementar um modo de jogo conjunto, decidimos usar o serial port.

Configuramos o UART com uma paridade ímpar, 8 bits por caracter e um bitrate de 115200, ativamos ambos os fifos e usamos interrupts, seja para saber se o canal transmissor está vazio, temos um caracter à espera para ser lido, ou ainda temos pelo menos um char no fifo que não foi lido. Para evitar perdas de informação decidimos implementar uma Queue para guardas bytes para enviar, ou bytes que já chegaram mas ainda não foram tratados.

Para comunicar, usamos sempre a estrutura char. Temos certas ordens, como escolha de player ou início de jogo que já estão associadas à sua char, por outro lado, também temos a necessidade de enviar os movimentos do player obtidos pelo scancode, logo mapeamos o

scancode a uma char para enviar, e ao receber mapeamos de volta para o scancode de modo a ser lido da forma correta. As funções de mapeamento são (char convertScancodeToChar(uint8_t scancode)) e (uint8_t mapCharToKey(char received_char)). Os chars provenientes do outro player são tratados na função (void (manageOutInput)()).

O código referente a este dispositivo encontra-se no ficheiro serialPort.c que está dentro da pasta dos devices.

3.Organização e estrutura do código

3.1. graphic - 5%

Este ficheiro contém as funções desenvolvidas para permitir mostrar uma interface ao utilizador, contém todas as funções referentes ao device do gráfico. Além disso, ainda possui funções de auxílio ao double buffering e page flipping. Como estruturas da classe, são guardados os dois buffers, as informações acerca da resolução e o buffer que está a ser usado.

3.2. keyboard - 5%

Este ficheiro contém as funções referentes ao device keyboard. Aqui o interrupt é ativado, desativado, para além de que controlamos a sua ativação organizando numa estrutura os packages recebidos, e guardando o scancode numa variável.

3.3. kbc - 3%

Este ficheiro tem como objetivo tirar proveito do facto de que ambos o rato e teclado usam o kbc para comunicar. Logo, aqui temos funções de escrita e leitura do kbc. Estas funções são utilizadas nas classes do mouse e do keyboard.

3.4. mouse - 4%

Este ficheiro contém as funções referentes ao dispositivo mouse. Aqui o interrupt é ativado, desativado, para além de que controlamos o seu acontecimento ao organizar numa estrutura os packages recebidos, e guardando o scancode numa variável.

3.5. queue - 2%

Tendo em conta o serial port, o grupo sentiu que era necessário criar uma estrutura para guardar os bytes em espera, Logo, criamos uma simulação de uma queue (fila), visto que em c não existe a estrutura. Podemos adicionar e retirar elementos da fila, tal como verificar se está cheia ou vazia.

3.6. rtc - 3%

Este ficheiro contém as funções referentes ao dispositivo rtc. Aqui o interrupt é ativado, desativado, para além de que controlamos o seu acontecimento.

3.7. serialPort - 16%

Este ficheiro contém as funções referentes ao dispositivo serial port. Aqui podemos escrever/ler nos diversos registos, ativar/desativar os interrupts, e controlá-los ao mandar/receber bytes. Os bytes em espera de serem processados são guardados em duas queues, uma para os recebidos, e outra para os que estão prontos para serem enviados.

3.9. timer - 5%

Este ficheiro contém as funções referentes ao dispositivo timer. Aqui podemos ativar/desativar os interrupts, e controlá-los ao aumentar a contagem. Podemos ainda alterar a frequência do mesmo.

3.10. utils - 1%

Este ficheiro contém as funções para obter o byte mais/menos significativo e uma adaptação ao sys_inb, de modo a receber o valor num formato encurtado. Estas funções são usadas no desenvolvimento dos devices.

3.11. Bomb - 7%

Este ficheiro possui as funções referentes à estrutura Bombs, que guarda um conjunto de bombas(objetos). Funções como criar a estrutura, adicionar e remover uma bomba, o movimento de cada bomba e o seu desenho na tela.

Numa fase final, esta estrutura acabou por ser aproveitada para designar um conjunto de balas, visto que a lógica é semelhante.

3.12. Object - 1%

Este ficheiro possui as funções referentes à estrutura Object, que guarda um sprite e a velocidade horizontal e vertical. Podemos criar um objeto e desenhá-lo na tela. Esta classe foi bastante aproveitada para criar outras classes, tal como os Bombs, Planes, Player.

3.13. Plane - 4%

Este ficheiro possui as funções referentes à estrutura Plane e à estrutura Planes que é um conjunto da primeira. O Plane guarda um objeto e o seu estado. Podemos criar um Plane e adicioná-lo aos Planes, assim como tratar do movimento e desenho destes.

3.14. Player - 1%

Este ficheiro possui o movimento de um Player, sendo este apenas um objeto.

3.15. Score - 11%

Este ficheiro possui funções que permitem controlar o score ao longo de uma partida, desenhá-lo no ao longo e no final da mesma e atualizar os nossos top scores no caso da pontuação vencer um dos top 9.

3.16. Sprite - 3%

Este ficheiro possui as funções referentes à estrutura Sprite, que guarda as coordenadas da posição, a altura e largura, e o mapa das cores. Podemos criar um sprite e desenhá-lo na tela, seja na sua forma total, ou um corte do mesmo. Também podemos carregar outro xpm de modo a obter um novo mapa de cores.

3.17. controller - 2%

Este ficheiro possui apenas a função para controlar o input vindo do keyboard.

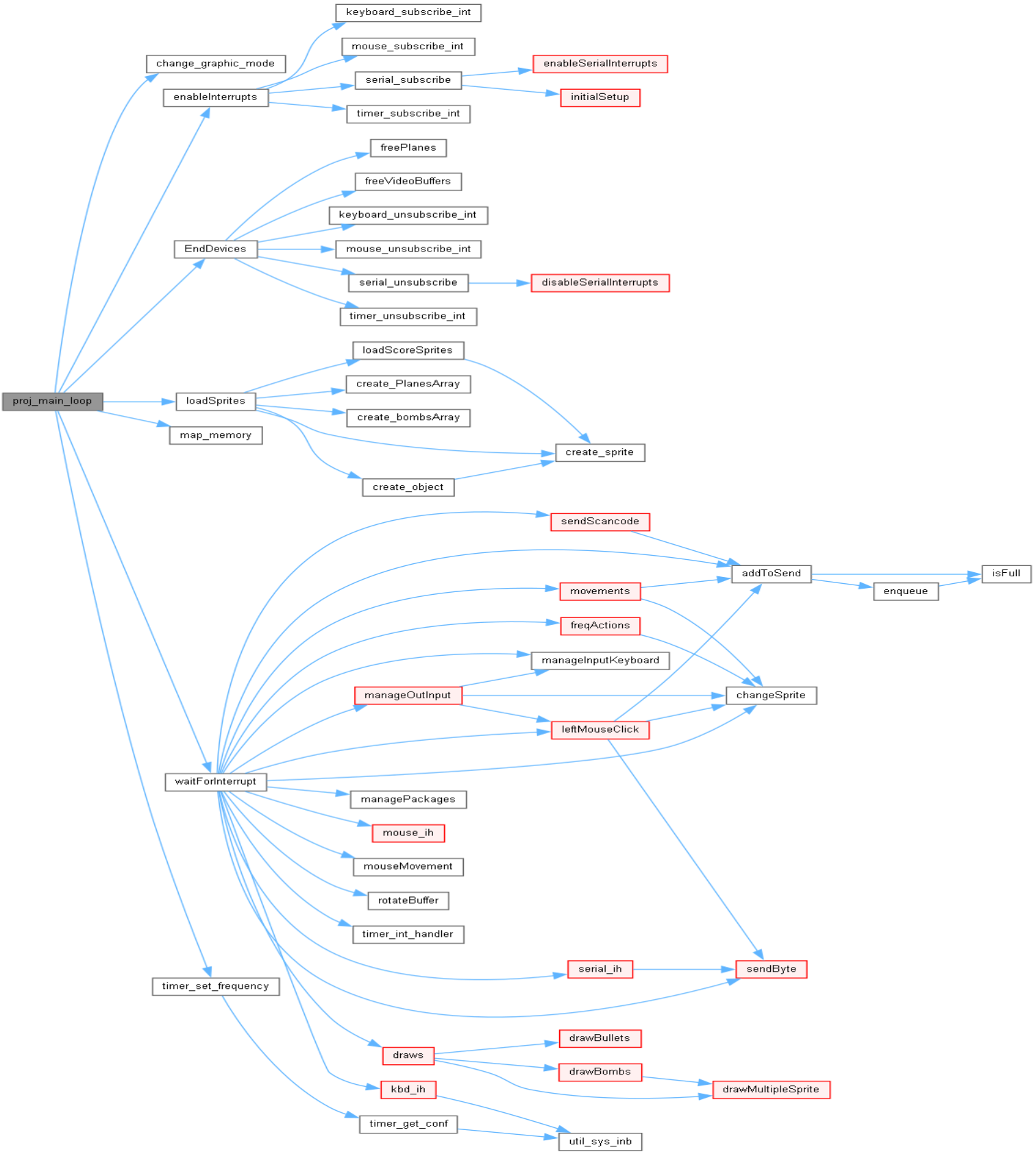
3.18. model - 26%

Este ficheiro possui toda a lógica por trás do jogo. Aqui carregamos os sprites todos, ativamos/desativamos os interrupts e, tendo em conta o device do interrupt, controlamos os acontecimentos.

3.19. proj - 1%

Este é o ficheiro base, onde mapeamos a memória, mudamos o modo gráfico e chamamos os interrupts do model.c.

Gráfico de chamada de funções



4.Detalhes de implementação

Cada frame nosso é um interrupt do timer, ou seja, tendo uma frequência de 60, temos 60fps, fomos obrigados a fazer assim, porque o nosso jogo têm que decorrer mesmo que não haja nenhum interrupt nos devices como keyboard ou mouse, visto que temos objetos que se movem sozinhos.

O nosso estado de jogo funciona como uma máquina de estados. Tendo em consideração o estado realizamos diferentes ações, como vemos nas funções (void draws() , void movements()), onde tendo em conta o estado de jogo pinta diferentes sprites na tela / causa diferentes movimentos de sprites.

Temos colisões para manter o player dentro da tela de jogo, e o rato também. Entre player e bomba, observamos se o player está dentro de um raio de 50 px em relação ao centro da bomba.

Tivemos bastantes problemas na implementação do serial port, inicialmente tínhamos apenas queues de receção e transmissão, e os interrupts não estavam a funcionar corretamente, então decidimos ativar os fifos que parece que corrigiram os problemas de interrupts perdidos.

5.Conclusão

Inicialmente, tínhamos como objetivo fazer um jogo mais complexo, mas com a falta de tempo e de membros de grupo acabamos por fazer um jogo mais simples, mas que utiliza todos os devices possíveis.

Algumas das features que poderiam ser implementadas com mais tempo:

- Melhorar a experiência de multiplayer
- Ataque especial aos aviões
- Aviões especiais
- Sistema de níveis
- Sprite do player melhorada
- Melhorar o sistema de vida (possibilidade de ganhar vida)

Apesar dos desafios encontrados ao longo do caminho, a execução deste projeto foi incrivelmente gratificante, pois tivemos total controle sobre as funções de baixo nível, permitindo-nos criar as bases do nosso projeto de forma precisa e adaptada às nossas necessidades específicas.