U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório do Projeto

"Chopper Drop"

Laboratório de Computadores

30 de Dezembro de 2014

T6G08:

Miguel Pereira - up201305998
Luís Oliveira – up201304515

Índice

1. Chopper Drop

1.1. Descrição

O funcionamento do jogo é bastante simples. Tem-se um helicóptero, que tem a possibilidade de largar bombas, a movimentar-se na horizontal da esquerda para a direita e um conjunto de edifícios de diferentes alturas. O helicóptero vai diminuindo a sua altitude ao longo do jogo. O objetivo é destruir todos os edifícios antes do helicóptero embater neles. Assim que o helicóptero embate num dos edifícios ou todos os edifícios são destruídos o jogo acaba.

1.2. Instruções de utilização

No menu inicial existem apenas duas opções – “play” e “exit” – que fazem exatamente o que o nome indica. Durante o jogo, utiliza-se o botão esquerdo do rato ou a barra de espaço para largar bombas. Existe, ainda, a possibilidade de pausar o jogo carregando na tecla “P” ou sair para o menu principal carregando em ESC. No canto superior direito do ecrã é possível ver o número de blocos destruídos até ao momento.

2. Dispositivos de entrada/saída

Na realização deste projeto foram utilizados todos os dipositivos estudados nas aulas, à exceção da porta de série. De seguida, apresenta-se uma tabela que contém a informação das funcionalidades usadas de cada dispositivo e maneira como cada um foi utilizado (se se usaram interrupções ou não).

Dispositivo	Para que foi utilizado	Interrupções
Timer	Controlar o *frame rate*	Sim
Teclado	Navegar entre menus, pausar o jogo e largar bombas	Sim
Rato	Selecionar opções e largar bombas	Sim
Placa gráfica	Menus e interface do jogo	Não
RTC	Apresentar a data e a hora	Sim

Os dispositivos com interrupções foram implementados recorrendo a um ciclo que recebe as interrupções de cada dispositivo e chama os respetivos *handlers*. A verificação das interrupções é feita no ficheiro “ChopperDrop.c” nas linhas 60-87.

2.1. Timer

Foram utilizadas as interrupções do *timer* 0 para atualizar o jogo. A estrutura do *timer* que guarda a informação de quantas interrupções teve este dispositivo durante a execução do programa e se existiu ou não uma interrupção do *timer* é atualizada nas linhas 73-74.

2.2. Teclado

É utilizado um módulo de baixo nível – o KBC – para fazer a leitura dos *scancodes* das teclas pressionadas. Esta leitura é feita nas linhas 69-70.

2.3. Rato

Tal como para o teclado, utiliza-se o KBC para ler os *packets* do rato. Destes *packets*, retira-se a informação do deslocamento do rato, de modo a calcular a posição atual, e o estado dos botões direito e esquerdo (se estão ou não pressionados). A leitura dos *packets* e a atualização da estrutura do rato que contém esta informação é feita com a chamada de uma função nas linhas 77-78.

2.4. Placa gráfica

Utilizou-se a placa gráfica para manipular o que era apresentado no ecrã em modo gráfico, escrevendo para a memória de vídeo. Isto é feito no ficheiro “Bitmap.c” nas linhas 143-145 e em “GameState.c” na linha 271.

2.5. RTC

Lê-se os registos do RTC, de modo a atualizar a estrutura que contém a hora e data atuais. A estrutura é atualizada com a chamada de uma função nas linhas 81-82.

3. Organização do código

3.1. main.c

Como o nome indica, é por este módulo que começa a ser executado o programa. Primeiro, inicializa-se o modo gráfico pretendido e cria-se um objeto do tipo ChopperDrop (será explicado no tópico seguinte). De seguida, tem-se o ciclo principal, onde, a cada iteração, se atualiza e desenha, caso necessário, o estado atual e se verifica se o jogo acabou ou não. Por último, assim que o jogo acaba, destrói-se o objeto criado, libertando todos os recursos por ele usados e sai-se do modo gráfico, voltando ao modo de texto.

Membro responsável: Miguel Pereira
Peso relativo: 5%

3.2. ChopperDrop.h e ChopperDrop.c

Este é o módulo base do jogo. É onde se verifica se existiram interrupções de algum dos dispositivos subscritos – chamando o *handler* respetivo, caso necessário – e onde se atualiza e desenha o jogo de acordo com o estado atual. É, também, aqui que se muda de estado, assim que o estado atual termina. A estrutura principal do jogo é composta pelos bits dos vários dispositivos na máscara de interrupção, duas *flags* que dizem quando o jogo termina e quando é preciso desenhar o ecrã, o *scancode* da tecla pressionada (que é 0, se nenhuma tecla foi pressionada), um apontador para a estrutura do *timer*, da data e do bitmap do cursor do rato e ainda um atributo que diz o estado atual e um apontador para a estrutura do estado atual.

Membro responsável: Miguel Pereira
Peso relativo: 10%

3.3. Bitmap.h e Bitmap.c

Este módulo contém a função que carrega os bitmaps e a função que os escreve no ecrã, bem como a função que liberta todos os recursos usados por um bitmap assim que ele deixa de ser usado.

Membro responsável: Luís Oliveira
Peso relativo: 10%

3.4. MainMenuState.h e MainMenuState.c

Neste módulo está implementado o menu principal do jogo, ou seja, as funções que o criam, atualizam, desenham e apagam. Aqui criaram-se duas estruturas. Uma que representa um retângulo, com a sua posição, altura e largura. E outra correspondente a este estado. Esta última estrutura é composta por três *flags* – uma que indica se o estado acabou e outras duas que indicam se o rato está sobre o botão “play” ou “exit” –, três bitmaps – um para o fundo e outros dois para o *hover* dos botões – e, por fim, dois

retângulos que representam os limites dos botões. Este estado acaba assim que alguma das opções é selecionada com o botão esquerdo do rato ou a tecla ESC é pressionada.

Membro responsável: Miguel Pereira
Peso relativo: 3%

3.5. GameState.h e GameState.c

É neste módulo que estão implementadas as funções que controlam o jogo propriamente dito. A estrutura deste estado inclui três *flags* que indicam se o estado terminou, se o jogo está em pausa ou se está uma bomba a ser largada, três apontadores para bitmaps – um para o solo, um para o cabeçalho e um para quando o jogo está em pausa –, um apontador para a estrutura do helicóptero e um para a da bomba, um *array* de apontadores para explosões e um para os edifícios, e, por último, o número de blocos destruídos. É aqui se chamam as funções que atualizam as posições do helicóptero e da bomba (caso esta esteja no ecrã) e se detetam e tratam as colisões existentes, seja entre a bomba e os edifícios ou entre o helicóptero e os edifícios.

Membro responsável: Miguel Pereira
Peso relativo: 15%

3.6. GameOverState.h e GameOverState.c

Este módulo contém as funções que criam, atualizam, desenham e apagam o estado de *game over*. Assim que se pressiona ENTER, este estado termina. A estrutura deste estado tem uma *flag* que diz se o estado terminou ou não e um apontador para o bitmap do fundo que deve ser desenhado neste estado.

Membro responsável: Luís Oliveira
Peso relativo: 2%

3.7. GameWonState.h e GameWonState.c

Este módulo contém as funções que criam, atualizam, desenham e apagam o estado de *game won*. Assim que se pressiona ENTER, este estado termina. A estrutura deste estado é igual à do estado de *game* over: tem uma *flag* que diz se o estado terminou ou não e um apontador para o bitmap do fundo que deve ser desenhado neste estado.

Membro responsável: Luís Oliveira
Peso relativo: 2%

3.8. Chopper.h e Chopper.c

Neste módulo estão implementadas as funções que criam, atualizam, desenham e apagam o helicóptero. A estrutura do helicóptero é composta pela sua posição, a largura e a altura do bitmap que o representa e duas *sprites*: uma com a pá esquerda do rotor e outra com a pá direita. Estas *sprites* vão ser desenhadas alternadamente de modo a fazer um efeito de movimento nas pás do rotor. A cada atualização a coordenada x do

helicóptero aumenta num valor defino previamente; assim que o movimento horizontal está completo, a coordenada y aumenta também num valor definido anteriormente e a coordenada x é “reiniciada”. Se o helicóptero embater num edifício, a posição é aumentada em x e em y até este chegar ao solo ou desaparecer do ecrã.

Membro responsável: Miguel Pereira
Peso relativo: 5%

3.9. Bomb.c e Bomb.h

Aqui, estão presentes as funções para criar, atualizar, desenhar e apagar uma bomba. A estrutura que representa uma bomba é constituída pela sua posição, a largura e a altura do seu bitmap, a sua velocidade e o número de edifícios com os quais a bomba colidiu. De cada vez que a bomba é atualizada, a sua velocidade aumenta num valor equivalente ao do definido para a gravidade e a sua coordenada y aumenta no valor da velocidade.

Membro responsável: Luís Oliveira.
Peso relativo: 5%

3.10. Block.h e Block.c

Neste módulo estão as funções para criar, desenhar e apagar os blocos que constituem os edifícios. Cada bloco é uma estrutura que contém a sua posição no ecrã. É, ainda, neste módulo que está o apontador para o bitmap dos blocos, que é utilizado sempre que é preciso desenhar um bloco. Desta maneira, evita-se carregar o bitmap sempre que se desenha um bloco.

Membro responsável: Luís Oliveira
Peso relativo: 5%

3.11. Building.h e Building.c

As funções para criar, desenhar e apagar edifícios estão neste módulo. A estrutura de um edifício inclui a sua altura e um *array* de apontadores para os blocos que o constituem.

Membro responsável: Miguel Pereira
Peso relativo: 5%

3.12. Explosion.h e Explosion.c

Aqui implementaram-se as funções para criar, desenhar e apagar explosões e definiu-se o apontador para o bitmap das explosões, uma vez que, tal como para os blocos, não faz sentido estar constantemente a carregar o bitmap. A estrutura de uma explosão é formada pela sua posição e o tempo há que a explosão está no ecrã, contabilizado em número de interrupções do *timer*.

Membro responsável: Luís Oliveira
Peso relativo: 3%

3.13. Graphics.h e Graphics.c

As funções para entrar e sair do modo gráfico estão neste módulo, bem como as funções para "pintar" o ecrã de uma determinada cor e copiar o *buffer* da memória virtual para a memória física.

Membro responsável: Miguel Pereira
Peso relativo: 5%

3.14. VBE.h e VBE.c

Neste módulo está implementada a função que vai buscar a informação sobre o modo de vídeo atual.

Membro responsável: Luís Oliveira
Peso relativo: 2%

3.15. KBC.h e KBC.c

Neste módulo estão as funções de leitura/escrita do KBC e de limpeza do *buffer*. A função de leitura do KBC está implementada em C e em Assembly.

Membro responsável: Luís Oliveira
Peso relativo: 5%

3.16. Keyboard.h e Keyboard.c

Este módulo tem apenas as funções de *subscribe* e *unsubscribe* do teclado.

Membro responsável: Luís Oliveira
Peso relativo: 3%

3.17. Mouse.h e Mouse.c

Este módulo contém as funções que inicializam, atualizam e apagam a estrutura do rato, bem como as funções para escrever para o rato, ativar o *stream mode*, verificar se o rato está dentro de um determinado retângulo e fazer *subscribe* e *unsubscribe*. A estrutura do rato é constituída pela sua posição, o *packet* que está a ser lido e um atributo que indica o próximo *byte* a ser lido, uma *flag* que indica quando um *packet* acabou de ser lido e, ainda, a informação sobre se os botões direito e esquerdo estão ou não pressionados, bem como se é preciso desenhar o rato.

Membro responsável: Miguel Pereira
Peso relativo: 5%

3.18. RTC.h e RTC.c

Neste módulo está implementada a estrutura da data/hora e as funções que retiram essa informação do RTC e criam e atualizam essa estrutura.

Membro responsável: Miguel Pereira
Peso relativo: 3%

3.19. Timer.h e Timer.c

Aqui implementaram-se as funções que criam, atualizam e apagam a estrutura do *timer*. Esta estrutura é composta por uma *flag*, que indica se existiu uma interrupção do *timer* e um atributo que contém a informação de quantas interrupções do *timer* já existiram.

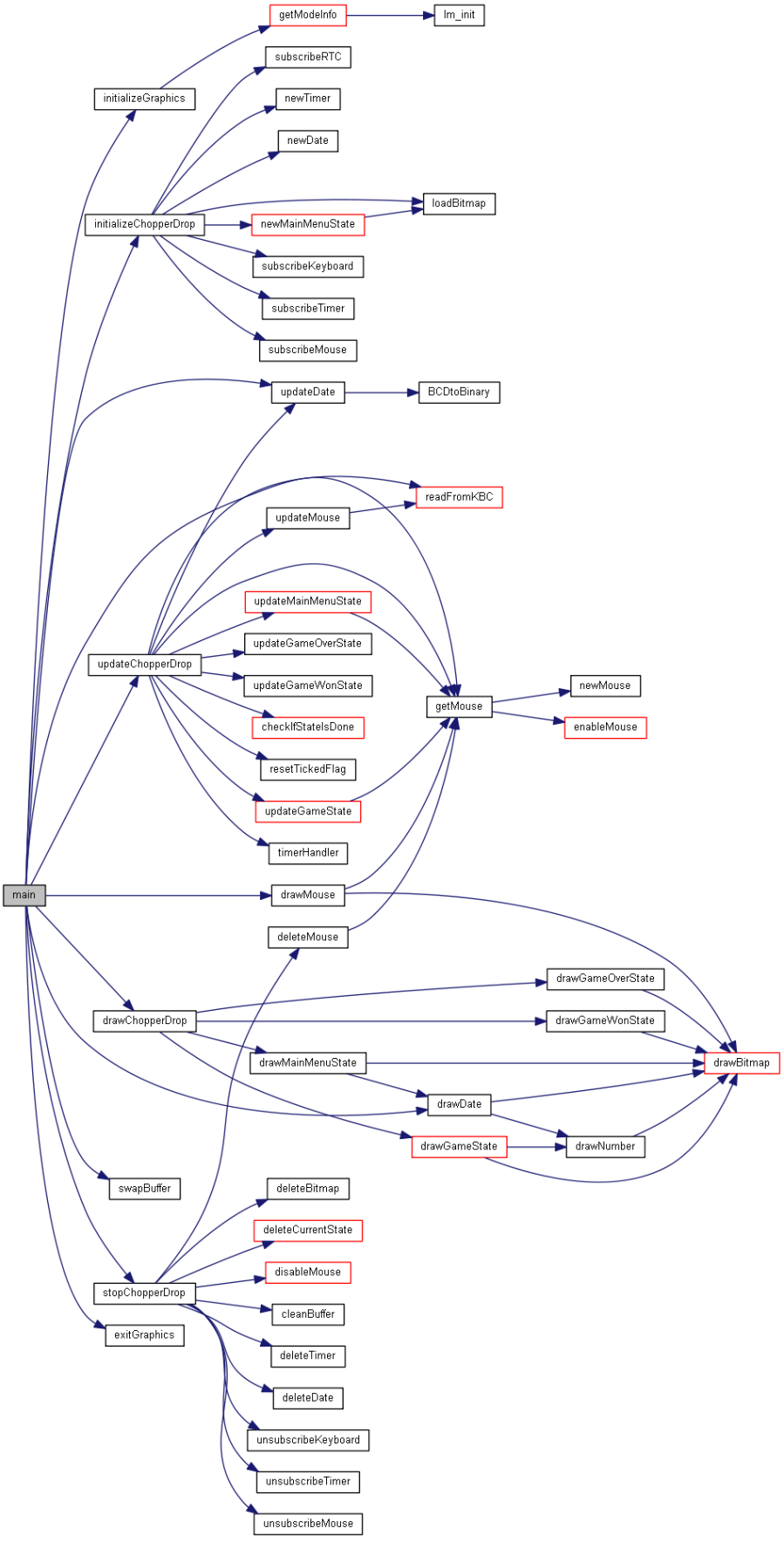
Membro responsável: Luís Oliveira
Peso relativo: 5%

3.20. Utilities.h e Utilities.c

Neste módulo estão definidas algumas variáveis globais – a resolução do ecrã e o número de *bytes* por *pixel*, uma vez que estas informações são necessárias em módulos diferentes do projeto – e várias macros que são utilizadas ao longo da implementação do programa. Contém, ainda, uma função que desenha um dado número no ecrã.

Membro responsável: Miguel Pereira
Peso relativo: 2%

4. Gráfico de chamada de funções



5. Implementação

O jogo foi implementado recorrendo a uma máquina de estados, de modo a facilitar a transição entre as várias fases do jogo. Existem quatro estados: o estado do menu principal, o estado do jogo e os estados de *game over* e *game won*. Os estados são ainda caraterizados por "ações" que indicam a razão do estado ter terminado, o que permite transitar para o estado correto, assim que isso acontece.

Todo o código foi desenvolvido num estilo orientado a objetos, utilizando tipos de dados abstratos. Para tal, recorreu-se a *structs*. Cada estado e cada elemento do jogo (helicóptero, bomba, explosão, bloco e edifício) tem a sua própria "classe" definida.

Um dos atributos do objeto principal do jogo é um apontador para o objeto do estado atual. Uma vez que cada estado é um objeto de uma "classe" diferente, isso foi possível apenas através de um apontador genérico para esse objeto.

Para o modo gráfico, utilizou-se a técnica de *double buffering*. Todos os objetos são desenhados, primeiro, para um *buffer* em memória virtual que é, depois, copiado para a memória física através da função *memcpy()*.

O jogo é atualizado 60 vezes por segundo, de acordo com a frequência por defeito do *timer*. Sempre que existe uma interrupção do *timer* são chamadas as funções de atualização e desenho, caso necessário, do estado atual.

A implementação da leitura e escrita de bitmaps foi baseada na implementação fornecida pelo estudante Henrique Ferrolho. No entanto, foram feitas alterações significativas na função de escrita no ecrã, de maneira a ser possível utilizar bitmaps com fundos "transparentes". Definiu-se uma cor (garantindo que não era usada por nenhum dos bitmaps do jogo) que não vai ser desenhada no ecrã. Isto é, sempre que um determinado *pixel* tem essa cor, ele não é copiado para a memória de vídeo.

A altura dos edifícios é gerada aleatoriamente, mas com algumas restrições, sendo que não podem existir mais de quatro edifícios com a mesma altura.

A deteção de colisões é feita de maneira otimizada. Com base na posição do helicóptero/bomba calcula-se o sítio exato onde devem ser verificadas as colisões, evitando procurar colisões com todos os objetos do ecrã. Cada objeto é interpretado como um retângulo, com base na largura e altura do bitmap que o representa.

A leitura dos *scancodes* das teclas é feita através da chamada de uma função em Assembly que lê o *output buffer* do KBC. Definiu-se o código das teclas utilizando uma enumeração, de maneira a poder usar o nome da tecla no desenvolvimento do código, e duas macros – KEY_UP e KEY_DOWN – que, dado o código de uma tecla, permitem saber se essa tecla está a ser pressionada ou se foi largada.

O RTC foi implementado com recurso às *update interrupts*. Sempre que ocorre uma interrupção, a data e hora são atualizadas e desenhadas no ecrã. No entanto, no início da execução do programa recorre-se a *polling* para ir buscar esta informação, de maneira a que não seja preciso esperar por uma interrupção para apresentar a data e hora.

6. Considerações finais

6.1. Avaliação da unidade curricular

Na nossa opinião, o facto do RTC e da porta de série não terem tido o mesmo tempo "dedicado" nas aulas práticas que os outros dispositivos, mas a sua inclusão no projeto ser avaliada equivalentemente é um ponto negativo, que até pode colocar em vantagem os estudantes que estão a repetir a unidade curricular. De resto, não temos mais nada a referir.

6.2. Instruções de instalação

É necessário invocar os seguintes comandos no diretório de topo do projeto, em *su*:

- sh install.sh
- sh compile.sh
- sh run.sh