U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Laboratório de Computadores

3 de janeiro de 2019

Chicken Run

Mark Meehan – up201704581

Luís Gonçalves – up201706760

Turma 7 grupo 06

Índice

1. Instruções de utilização

1.1. Main Menu

Ao iniciar o programa, o utilizador depara-se com o menu inicial. Neste menu, é apresentado o título do jogo, com uma interface gráfica, bem como duas opções: **Play Game**, que permite inicializar o jogo propriamente dito, e **Exit**, que termina o programa.

A opção pretendida neste menu, bem como em todos os outros, deve ser escolhida utilizando o rato, sendo que a interface gráfica reage à escolha selecionada, tornando-a clara para o utilizador.

1.2. Play Game

Selecionada a opção **Play Game**, é ativado o estado de jogo. O objetivo consiste em evitar os obstáculos, de dois tipos: cones, que surgem numa linha inferior, e os enxames de abelhas, que surgem numa linha superior. Sendo assim, cabe ao utilizador pressionar as respetivas teclas do teclado: caso pretenda que a galinha salte, deve pressionar a tecla ‘’w’’, e caso pretenda que a galinha se baixe, deve pressionar a tecla ‘’s”. Ao saltar, é efetuada sobre a galinha uma força de acelaração no sentido de cima para baixo, que contraria a sua velocidade inicial, sendo o movimento da mesma fluido e natural. Quando se baixa, a galinha toma uma forma diferente, passando por baixo dos enxames, mas não lhe é aplicada qualquer força externa.

São gerados obstáculos de forma aleatória. Contudo, a probabilidade de ser gerado um obstáculo no nível superior é igual à probabilidade de ser gerado um inferior. No entanto, os obstáculos não podem ser gerados muito próximos dos outros obstáculos, dado que isto tornaria o jogo impossível de completar, já que o utilizador nada poderia fazer para sobreviver.

São apresentadas, na tela, algumas informações ao utilizador. Primeiramente, é visualizada a hora (no formato de 24h) e a data atuais. Para além disso, é apresentado também o score atual. O score é representado por um inteiro que incrementa uma unidade a cada 20 interrupções do timer, ou seja, é incrementado a cada terço de segundo. Por fim, é representada uma barra referente ao número de Power Ups que o utilizador possúi, podendo variar de 0 até 3.

As imagens seguintes representam as capacidades da galinha face a objetos superiores e inferiores, respetivamente.

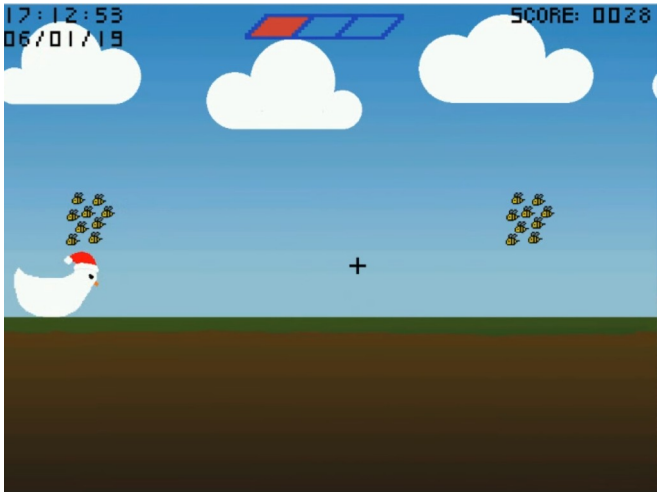


Figura 1 – Galinha após pressionar tecla ‘’s’’

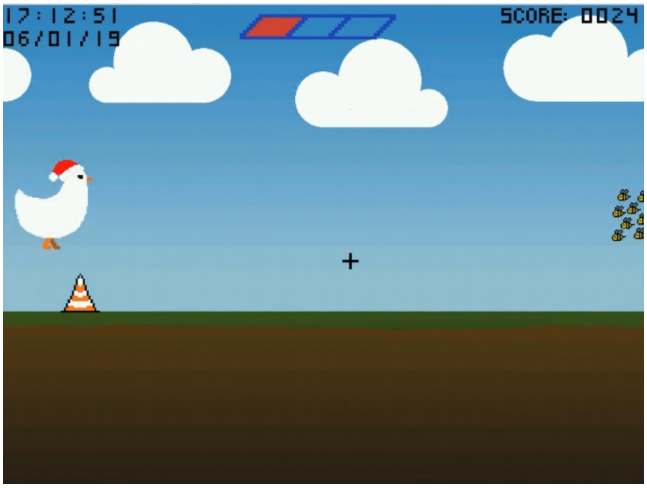


Figura 2 – Galinha após pressionar tecla ‘’w’’

1.3. Pause Menu

Quando o jogador prime a tecla “esc”, é levado para um menu de pausa. Neste menu, são-lhe dadas as opções de continuar a jogar, **Play**, ou sair do jogo, **Exit**. O score do jogador, bem como o número de *Power Ups* que tem e o seu progresso para o *Power Up* seguinte, são representados, e retomam de igual modo se o jogo for continuado.

Figura 3 – Menu de pausa

1.4. Game Over Menu

Quando a galinha colide com um obstáculo, seja ele um cone ou um enxame de abelhas, é aberto um menu de fim de jogo. Neste menu, é apresentado ao utilizador o seu score final, bem como duas opções: **Play**, que inicia um novo jogo, e **Quit**, que termina o programa.

Figura 4 – Menu de Game Over

2. Módulos implementados

Todos os periféricos previstos na especificação do projeto foram implementados, sendo estes o *Timer*, o *Keyboard*, o *Mouse*, a *Video Card* e o *RTC*.

Dispositivo	Utilização	Interrupções
Timer	Controlo de lógica do jogo e do *score*	Sim
Keyboard	Movimento de *sprites*	Sim
Mouse	Escolhas nos menus e controlo do jogo	Sim
Video Card	Representação gráfica	Não
RTC	Representação das horas e data atual	Não

2.1 Timer

O timer é utilizado na atualização da tela de jogo, a um ritmo de 60 *frames* por segundo. Para além disso, é também utilizado no controlo de grande parte da lógica do jogo. O movimento do *background*, o spawn de obstáculos entre determinados intervalos de tempo, a visualização da data e hora atuais, bem como o *score*, e a rapidez de deslocação dos obstáculos, que aceleram ao longo do tempo de jogo, todos dependem intrínsecamente do *timer* e das suas interrupções. Por fim, o controlo de P*ower Ups* é também dependente do *timer*, sendo que é gerado um *Power Up* a cada 5 segundos, caso não esteja atingido o teto máximo de 3 *Power Ups*.

2.2 Keyboard

A principal função do keyboard passa pelo controlo da personagem principal do jogo, a galinha. Ao pressionar as teclas ‘’w’’ e ‘’s’’, a galinha pode, respetivamente, saltar por cima dos obstáculos, ou esquivar-se por baixo dos mesmos. Embora não estivesse inicialmente previsto, de modo a preservar a configuração do jogo original, é também possível usar as teclas “space” e “left ctrl” para saltar e baixar a galinha, respetivamente, podendo o utilizador usar a predifinição mais conforável.

Para além disso, quando o utilizador está no estado de jogo, e pretende parar, pode pressionar a tecla “esc” de modo a surgir o menu de pausa.

2.3 Mouse

O rato é utilizado na navegação e escolha de opções nos menus inicial, de pausa e final. É representada no ecrã uma crosshair que se move com o movimento do rato, e as opções são selecionadas pressionando o botão esquerdo do rato sobre a mesma. Para além disso, o rato é também crucial na eliminação de obstáculos no jogo, quando o utilizador tem *Power Ups* disponíveis, sendo que o botão esquerdo do rato deve ser pressionado quando a crosshair se encontra sobre os obstáculos, de modo a que estes sejam eliminados.

2.4 Video Card

Este é o dispositivo usado na representação de todas as imagens utilizadas no jogo. O modo gráfico escolhido foi o 0x117, cuja resolução é 1024x768, sendo todas as imagens usadas neste projeto em modo RGB (5:6:5). De modo a tornar a animação de imagens mais fluida e natural, foi implementado *double buffer*. As imagens são visualizadas com acesso à biblioteca de bitmaps do antigo aluno Henrique Ferrolho.

2.5 RTC

O *RTC* é utilizado para a representaçao da data e hora atuais. Toda a informação obtida com origem neste dispositivo é armazenada na struct **Time**, sendo posteriormente acedida para a obtenção desta informação.

3. Estrutura e organização do código

3.1. Módulos implementados

3.1.2. timer

Neste módulo encontram-se as funções utilizadas no acesso ao *timer*, nomeadamente o *timer handler*, bem como o *timer subscribe* e *timer unsubscribe*, essencias para o acesso a interrupções. Todo o código deste módulo foi importado do lab2.

3.1.2. mouse

Neste módulo encontram-se as funções utilizadas no acesso ao *mouse*, especificamente as funções de leitura e análise do *mouse packet*, bem como as funções de *subscribe* e *unsubscribe* das interrupções. Todo o código deste módulo foi importado do lab3.

3.1.3. keyboard

Neste módulo encontram-se as funções utilizadas no acesso ao *keyboard*, nomeadamente para a leitura de *scancodes* de teclas pressionadas e soltas a cada interrupção, bem como, necessariamente, as funções de *subscribe* e *unsubscribe* do *keyboard*. Todo o código deste módulo foi importado do lab3.

3.1.4. video_gr

Neste módulo encontram-se as funções relacionadas com a placa gráfica, como a inicialização do modo gráfico e terminação do mesmo. Para além disso, o *double buffer* foi também implementado neste módulo.

Aluno responsável pelas alterações efetuadas: Mark Meehan.

3.1.4. rtc

Neste módulo foram implementadas funções utilizadas no acesso ao *RTC* para a representação da data e hora atuais.

Aluno responsável: Mark Meehan.

3.1.5. bitmap

Este módulo é responsável pela importação, representação gráfica e eliminação de *bitmaps.* O código contido neste segmento é da autoria do antigo aluno Henrique Ferrolho (http://difusal.blogspot.com/2014/07/minix-posts-index.html), sendo que foram feitas algumas modificações, nomeadamente, uma função que permite a representação de imagens com transparência. Esta capacidade revelou-se de grande importância, dado que possibilita a representação de imagens não-quadrangulares sem um fundo indesejado. A cor de transparência utilizada foi um rosa forte (0xff00ff).

3.1.6. chicken

Neste módulo encontram-se as funções que afetam a principal personagem do jogo, a galinha. Especificamente, estão implementadas funções que desenham, eliminam, e efetuam o salto da mesma. A função relativa ao salto da galinha tem em conta o movimento parabólico que esta deve tomar ao saltar. Sendo assim, ao saltar, é-lhe associada uma velocidade inicial vertical, com sentido de baixo para cima, sendo que a cada interrupção do timer lhe é efetuada uma força em sentido contrário, uma acelaração gravítica. Quando a galinha se baixa, o *bitmap* que lhe está associado é alterado, de modo a que a mesma passe por baixo dos enxames de abelhas.

Alunos responsáveis: Mark Meehan e Luís Gonçalves (50% / 50%).

3.1.7. crosshair

Neste módulo foi implementada uma *crosshair*, que permite ao utilizador navegar e fazer as suas escolhas nos menus do jogo, bem como eliminar obstáculos durante o jogo propriamente dito. Foi criada uma struct **Crosshair**, que armazena a informação relativa à mesma. Sendo assim, no teste de eliminação de obstáculos pelo rato, é utilizada a informação desta struct, comparando as coordenadas da *crosshair* em relação aos limites do obstáculo, bem como a verificação de que o botão esquerdo do rato está a ser pressionado.

Por fim, é também assegurado que a *crosshair* não pode sair da tela de visualização, já que, caso o utilizador perdesse a *crosshair*, era obrigado a terminar o programa forçadamente. As funções implementadas neste módulo constituem funções de representação e atualização da crosshair, sendo que a verificação de colisões foi efetuada num outro módulo, obstacles.c.

Aluno responsável: Mark Meehan.

3.1.8. game

Este contitui o principal módulo de todo o programa, sendo que todas as outras funções são, diretamente ou indiretamente, aqui utilizadas. Contém as interrupções dos periféricos, a inicialização e terminação do modo gráfico, bem como a chamada de todas as funções das quais decorrem o jogo e a sua respetiva lógica associada. Toda a lógica aqui implementada depende do *GameState* do jogo, implementado num outro módulo, gameState.c.

São inicializados os objetos usados no jogo, especificamente:

- background
- Chicken
- Crosshair
- Time
- mainMenu
- scoreMenu
- PowerUps
- GameState

Alunos responsáveis: Mark Meehan e Luís Gonçalves (50% / 50%).

3.1.9. gameState

Este módulo é responsável pela manipulação do estado de jogo. Contém um enum com os diferentes estados de jogo possíveis, **MAIN**, que representa o menu inicial, **PLAY**, que constitúi o estado de jogo, **PAUSE**, que representa o menu de pausa, **OVER**, estado ativado quando o jogo é perdido, e **EXIT**, responsável pela terminação do programa.

Aluno responsável: Mark Meeahan.

3.1.10. i8042

Módulo que contém macros relativas ao *mouse* e ao *keyboard*.

Alunos responsáveis: Mark Meehan e Luís Gonçalves(50% / 50%).

3.1.11. i8254

Módulo que contém *macros* relativas ao *timer*.

Alunos responsáveis: Mark Meehan e Luís Gonçalves(50% / 50%).

3.1.12. macros

Módulo que contém *macros* relativas a vários elementos do jogo e da sua lógica associada.

Alunos responsáveis: Mark Meehan e Luís Gonçalves(50% / 50%).

3.1.12. menu

Este módulo é responsável para criação, atualização e interação do utilizador com todos os menus, nomeadamente a escolha da opção por parte do utilizador, com acesso aos módulos *crosshair* e *mouse*. Assim, contém funções de desenho, atualização e interação com o utilizador.

Foi também criada a struct **background**, que contém a informação relativa ao *background* do jogo. Existem 2 *bitmaps* que lhe estão associados, um *bitmap* que representa o céu, e outro representativo do chão. Tal como no jogo original, o chão move-se à velocidade dos obstáculos, enquanto o céu se move a uma velocidade relativamente inferior. Sendo assim, todas as funções associadas à visualização, atualização e eliminação do *background* encontram-se aqui implementados.

Para além disso, está também implementado neste módulo a capacidade de retomar o jogo após uma pausa, bem como de começar um jogo novo. Assim, é também associada a este módulo o módulo *gameState*.

Alunos responsáveis: Luís Gonçalves e Mark Meehan (65% / 35%).

3.1.13. obstacles

Neste módulo estão contidas todas as funções de geração, atualização e colisão de obstáculos, bem como a manipulação, alteração e representação de *Power Ups*. É criada uma struct de 10 possíveis obstáculos. Num intervalo de tempo que varia entre um mínimo, que permite que o utilizador tenha sempre capacidade de sobreviver, e um máximo, que permite assegurar a relativa dificuldade do jogo, é gerado um obstáculo. Este, por sua vez, pode ser um obstáculo superior, um enxame de abelhas, ou inferior, um cone, com igual probabildade. Quando um obstáculo sai da tela, o seu espaço no array é libertado, sendo que a posição dos restantes obstáculos neste array é alterada de modo a ocupar sempre os primeiros índices disponíveis. Quando um objeto é gerado, ocupa a primeira opção disponível neste array. Deste modo, otimiza-se ao máximo a geração de obstáculos, evitando problemas de performance ao fim de algum tempo, quando forem gerados vários obstáculos.

É representado um *bitmap* diferente consoante o número de *Power Ups* disponíveis, sendo que essa informação é verficada a cada interrupção do *timer*.

Alunos responsáveis: Mark Meehan e Luís Gonçalves (80% / 20%).

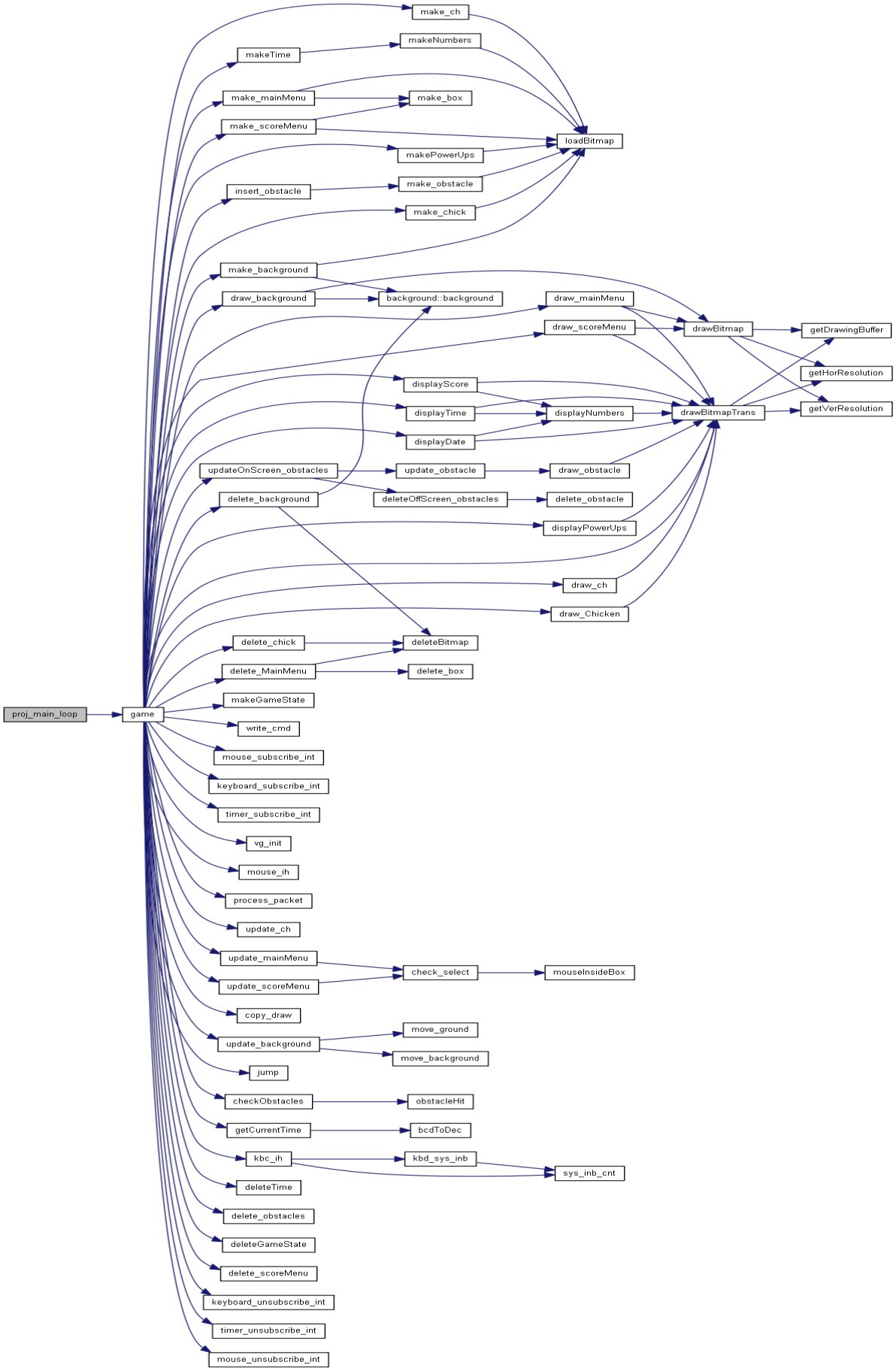
3.1.14. proj.c

A única função deste módulo é a chamar a função game(), que inicia o jogo.

Alunos responsáveis: Mark Meehan e Luís Gonçalves.

3.1.15 Peso dos módulos

Módulo	Peso
game	18%
obstacles	18%
chicken	8%
timer	8%
mouse	8%
menu	6%
gameState	4%
rtc	4%
bitmap	4%
crosshair	4%
background	4%
video_gr	4%
keyboard	4%
proj	1%
macros	1%
i8254	1%
i8042	1%


proj_main_loop
game
make_ch
makeNumbers
makeTime
make_mainMenu
make_box
make_scoreMenu
loadBitmap
makePowerUps
insert_obstacle
make_obstacle
make_chick
make_background
draw_background
background::background
draw_mainMenu
draw_scoreMenu
drawBitmap
getDrawingBuffer
getHorResolution
getVerResolution
displayScore
displayTime
displayNumbers
drawBitmapTrans
displayDate
updateOnScreen_obstacles
update_obstacle
draw_obstacle
delete_background
deleteOffScreen_obstacles
delete_obstacle
displayPowerUps
draw_ch
draw_Chicken
delete_chick
deleteBitmap
delete_MainMenu
delete_box
makeGameState
write_cmd
mouse_subscribe_int
keyboard_subscribe_int
timer_subscribe_int
vg_init
mouse_ih
process_packet
update_ch
update_mainMenu
check_select
mouseInsideBox
update_scoreMenu
copy_draw
update_background
move_ground
move_background
jump
checkObstacles
obstacleHit
getCurrentTime
bcdToDec
kbc_ih
kbd_sys_inb
sys_inb_cnt
deleteTime
delete_obstacles
deleteGameState
delete_scoreMenu
keyboard_unsubscribe_int
timer_unsubscribe_int
mouse_unsubscribe_int

4. Detalhes de implementação

De um modo geral, os nossos conhecimentos das aulas da unidade curricular foram suficientes para a implementação do projeto. No entanto, algumas tarefas exigiram alguma pesquisa e trabalho da nossa parte. Entre elas:

- A verificação de colisões. No nosso projeto, utilizamos dois tipos de deteção de colisão: colisões por cor pixel a pixel, para as colisões entre a galinha e os obstáculos, e colisões AABB, para as colisões entre a crosshair, quando o botão esquerdo do rato é premido, e os obstáculos. De um modo geral, foram relativamente fáceis de aprender e implementar, pelo que não trouxeram grandes dificuldades.
- O uso de máquinas de estado. Não trouxe grandes dificuldades embora tivesse sido um desafio a sua implementação eficiente e bem interligada com o restante código efetuado.
- A programação orientada a objetos. De início causou alguma confusão, devido à sua peculiariedade na linguagem C, contudo, facilmente nos adaptamos e tentamos segmentar ao máximo o código, maximizando o uso da orientação a objetos.

5. Conclusão

Em conclusão, consideramos que este projeto nos permitiu aprender matérias diversas e completamente novas, adquirindo novos conhecimentos. Embora tenha sido um projeto desafiante e muito trabalhoso, a grande liberdade de escolha que nos foi dada tornou a sua aplicação genuinamente interessante e divertida, para além de todos os benefícios técnicos que lhe estão associados.

Em relação à unidade curricular de Laboratório de Computadores, consideramos a disciplina de grande importância naquilo que são as competências de um Engenheiro Informático, devido às bases que aprendemos. No entanto, não consideramos a carga de trabalho adequada ao número de créditos da disciplina, sendo os *labs* a cada duas semanas muito exigentes tanto a nível de pesquisa, como de implementação, como de *debugging*. Por fim, não consideramos, também, o teste de programação adequado à unidade curricular, sendo que inúmeros alunos que efetuaram todos os labs autonomamente, e certamente terão também sucesso no seu projeto, não conseguiram completar o 1º exercício do mesmo, estando agora em vias de não obter aprovação. Não consideramos justo a colocação de alunos nesta situação por um teste de peso de 10% e com todos os critérios que lhes estão associados.