U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

JANEIRO 2018

pilldrop

Laboratório de Computadores

Grupo T4G08

Mariana Barbosa Aguiar - up201605926

Tiago José Viana Fragoso - up201606040

Índice

1. Instruções de utilização

1.1 Menu principal

Ao iniciar o programa, é apresentado o menu principal, onde o utilizador poderá utilizar o rato para selecionar qualquer uma das quatro opções apresentadas:

Singleplayer: Inicia o modo de jogo singleplayer;
Multiplayer: Inicia o modo de jogo multiplayer, utilizando serial port;
Options: Abre um menu que permite selecionar o tema e a dificuldade de jogo;
Exit: Fecha o programa.

1.2 Menu “Options”

O menu “Options” permite configurar o tema e a dificuldade do jogo. Por defeito, as opções selecionadas são “automatic” e “easy”. Em modo automático, o programa utiliza o RTC para determinar o tema, dentro dos 3 possíveis, abaixo representados, sendo que o programa utilizará o tema “day” entre as 7h e as 18h e o tema “night” nas restantes horas do dia. O tema de Natal, “xmas”, é utilizado a qualquer hora do dia entre 1 de Dezembro e 6 de Janeiro. A mudança do tema reflete-se apenas nos *sprites* e fundo do jogo e não nos menus. Quanto à dificuldade, esta reflete-se apenas no modo “singleplayer”, sendo que no modo “multiplayer”, é imposta a dificuldade “easy”, já que esta permite tirar melhor partido da funcionalidade desse modo.

1.3 “Singleplayer”

Neste modo de jogo, o objetivo é destruir os *sprites* (pills) que caem do topo do ecrã antes que estes passem o fundo. Para isso, é necessário que a “mira” (cursor) tenha a cor da pill a destruir.

Alguns comandos básicos:

- **Botão esquerdo do rato:** utilizado para destruir as pills
- **Botão direito do rato:** utilizado para mudar a cor do cursor
- **Tecla “ESC”:** coloca o jogo no modo de pausa

1.3.1. Descrição aprofundada do jogo

No topo do ecrã, são geradas pills de 2 cores possíveis, com igual probabilidade de *spawn,* ou duma terceira cor, nas dificuldades “medium” ou “hard”, com menor probabilidade de *spawn*. Esta pill, denominada de “death pill” não deve ser destruída, já que resulta na perda imediata do jogo.

A mira controlada pelo rato tem 2 cores possíveis, que podem ser trocadas pressionando o botão direito deste. Para poder destruir uma pill, é necessário combinar as cores da mira com a pill e pressionar o botão esquerdo do rato em cima desta. Um clique falhado resulta na perda do contador do bónus, ou, na dificuldade “hard” resulta na perda de uma vida , caso se tente destruir uma pill da cor errada.

O contador de bónus no canto superior esquerdo conta o número de cliques certeiros nas pills e assim que completo leva o jogador a um nível de bónus. Para alcançar esse nível, o jogador deve realizar 10, 20 ou 30 cliques certeiros, sem deixar que nenhuma pill passe o fundo do ecrã, nas dificuldades “easy”, “medium” e “hard”, respetivamente.

Sempre que uma pill passa o fundo do ecrã, o jogador perde uma vida, no entanto recupera uma sempre que atinge o bónus.

O jogo possui também uma evolução praticamente linear, limitada tanto inferior como superiormente, isto é para cada dificuldade a *spawn rate* aumenta linearmente com o score, mas com diferentes mínimos e máximos, assim como a velocidade em x e y das pills.

1.3.2 "Singleplayer bonus"

Quando o jogador atinge este modo, que dura apenas 10 segundos, tem a possibilidade de "apanhar" o maior número de pills possível, que será adicionado ao seu score no fim do bónus, assim como 1 vida, até a um máximo de 10 vidas. Para "apanhar" as pills, o jogador deve mover o *sprite* com as teclas "A" e "D", de modo a que este colida com as pills.

1.3.2 "Singleplayer pause menu"

Carregando na tecla ESC durante o jogo em modo "singleplayer" o jogador entra no menu de pausa.

Resume: Volta ao jogo.

Restart: Reinicia o jogo.

Exit: Volta ao menu inicial.

1.3.3 “Singleplayer game over menu”

O jogador entra no menu de “game over” quando chega ao fim das vidas, mostrando a pontuação obtida.

Restart: Reinicia o jogo.

Exit: Volta para o menu inicial.

1.4 “Multiplayer”

No modo “multiplayer”, ambos os jogadores entram num menu de “lobby” onde é efetuada a sincronização entre eles através de serial port. Neste momento, é gerada uma *seed* por um dos jogadores e introduzida na função *srand()*, que é utilizada para gerar todos os valores numéricos que dão a aleatoriedade ao jogo. Deste modo, ambos os jogadores jogam um jogo exatamente igual, já que os números pseudo-aleatórios gerados pela função *rand()* serão os mesmos. Neste modo, a dificuldade é sempre mudada automaticamente para “easy”, dado que o modo de bónus é, neste caso, um período de 2 segundos em que o rato do adversário fica bloqueado. Como a dificuldade é a mais baixa, é mais fácil chegar ao bónus e o jogo torna-se mais divertido. Os restantes princípios do jogo no modo “singleplayer” mantêm-se, exceto o menu de pausa que foi desativado no modo “multiplayer”. Além disso, o jogo termina quando qualquer um dos jogadores ficar sem vidas.

1.4.1 "Multiplayer game over menu"

Este menu apresenta os scores dos 2 jogadores, bem como o resultado do jogo (vitória ou derrota). O botão **Exit** permite voltar ao menu inicial.

2. Estado do projeto

2.1 Dispositivos utilizados

Dispositivo	Utilização	Interrupts
Timer	Atualização do estado do jogo	Sim
Keyboard	Movimentação de *sprites* e atalhos	Sim
Mouse	Seleção de menus e controlo do jogo	Sim
Placa gráfica	Desenho de imagens	Não
RTC	Atualização da data/tema automático	Sim
UART	Modo multiplayer	Sim

2.1.1 Timer

A função principal do timer é a utilização dos seus *interrupts*, mais precisamente 60 por segundo, para atualizar o estado do jogo. A cada *interrupt*, é desenhado o background e são testadas todas as condições que levam a atualização de estados ou realização de eventos, como, por exemplo, colisões com com o rato, atualizações de coordenadas, *spawn* de sprites, etc. É o dispositivo mais importante do jogo, já que todas as atualizações necessárias dependem dos seus *interrupts,* sendo estes tratados na função "timerIH", do ficheiro **GameCtrl**.

2.1.2 Keyboard

O keyboard possui 2 funcionalidades distintas no programa: utilização de atalhos e movimentação de *sprites.* No menu principal, é possível pressionar a tecla "q" para sair do jogo e no modo "singleplayer", um clique na tecla "ESC" coloca o jogo em modo de pausa.
No modo "singleplayer bonus", as teclas "A" e "D" são utilizadas para mover o *sprite.* Ambas

as funcionalidades utilizam os *interrupts* do keyboard, que são tratados na função "keyboardProcess", do ficheiro **GameCtrl**. É apenas utilizado para jogar/controlar a aplicação.

2.1.3 Mouse

O mouse é o dispositivo principal, no que diz respeito ao *input* do utilizador. Os *interrupts* do mouse são utilizados em todos os modos exceto "singleplayer bonus", quer seja para selecionar opções dos menus, ou movimentar a mira e destruir pills. A função que realiza o tratamento da informação obtida a cada *interrupt* é "mouseProcess", do ficheiro **GameCtrl**. É utilizada a posição do rato, assim como os 2 botões principais.

2.1.4 Placa gráfica

A placa gráfica é utilizada no modo 117h, com resolução 1024x768, modo de cor RBG(5:6:5), sensivelmente 64K cores. A implementação utiliza double buffering para melhorar o desempenho do desenho dos gráficos, que se encontra no ficheiro **video_gr**. Quanto a movimentação de objetos, há teste de colisões com os limites do ecrã e entre *sprites*, mas não implementamos animação de *sprites*. Utilizamos também uma *font*, "Aerolite", para impressão de *score* e da data em 2 tamanhos diferentes, que se encontram em "/res/fonts/", realizada pela função "drawAerolite" e "drawDate", do ficheiro **GameModel**. A deteção de colisões com os botões é realizada a partir de coordenadas, no entanto a deteção de colisões dentro do jogo é feita utilizando colisões entre retângulos no mesmo eixo, com um simples algoritmo que testa se os retângulos se tocam, na função "testCollision", do ficheiro **GameCtrl**.

2.1.5 Real Time Clock

O RTC é utilizado para alterar a data que é apresentada no menu principal e, caso o tema esteja em modo automático, altera o tema automaticamente, quando o utilizador se encontrar no menu principal, dependendo da hora/data, usando os seus *interrupts.* A função que processa interrupts do RTC é "rtcProcess", do ficheiro **GameCtrl**, complementada por uma função em *Assembly*("rtcIntHandlerASM"*)*, que lê os registos de RTC e identifica a origem do *interrupt*, caso este seja um *Update interrupt,* atualiza a data.

2.1.6 UART

O UART é utilizado para o modo "multiplayer" e foi implementado utilizando *interrupts* e FIFOs. Para complementar a FIFO da UART, implementamos uma queue em C, que nos permite guardar a informação proveniente da FIFO e depois processá-la ou guardar informação a introduzir na FIFO e depois processá-la. Foram implementadas 2 queues diferentes (*Received* e *Send* queue). Toda a implementação foi realizada no ficheiro **serialport**.

2.1.6.1 Sincronização de jogadores com UART

Para a sincronização dos 2 jogadores, usamos a função "lobbyManager" ficheiro **MultiplayerCtrl**, que procura um jogador que esteja no menu "multiplayer" e se ambos se conseguirem conectar, um deles envia uma seed que é usada pelos 2 jogadores e assim o jogo é iniciado.

1. **Estado de sincronização:**
   O primeiro jogador a entrar no "lobby" envia um "convite" ao outro jogador. Quando este entrar no lobby, procura pelo "convite" e caso o tenha recebido envia uma mensagem de confirmação. A partir deste momento, o primeiro jogador fica identificado como "host" e tem a função de gerar a *seed* e o segundo jogador fica identificado com "client" e ambos os jogadores estão conectados. De seguida, o "host" gera e envia a seed e quando o "client" a receber envia a mensagem de que está pronto e ambos começam o jogo.
2. **Durante o jogo:**
   Caso um jogador consiga atingir o "bonus level", este envia uma mensagem ao outro jogador, que bloqueia o seu rato durante 2 segundos.
3. **Fim do jogo:**
   O primeiro jogador a ficar sem vidas, envia uma mensagem ao segundo, informando que perdeu o jogo e qual o seu score final e fica a espera do score do jogador adversário. Por outro lado, caso um jogador receba a mensagem de perda do outro, pode automaticamente passar para o menu de "game over" porque já possui ambos os scores.

3. Estrutura do código

3.1 Módulos

3.1.1 Game

Este módulo contém o **GameModel** e o **GameCtrl**, que serão abordados mais tarde, e é responsável por iniciar o jogo, chamando o **handler**.

Este módulo foi desenvolvido na íntegra pela Mariana Aguiar.

3.1.2 GameModel

Este módulo contém todas as variáveis, objetos e macros de pré-processador que são utilizadas para definir a dificuldade ou evolução do jogo.

Quanto às variáveis, este módulo inicializa:

- **Struct Status** que contém várias variáveis sobre o estado do jogo, dificuldade, tema, etc.
- Variáveis mínimas e máximas para **velocidades**, **frequência de *spawn*** de certos objetos e outras variáveis que definem a evolução do jogo, todas dependentes da **dificuldade** selecionada, usando as macros.

Quanto aos objetos, este módulo inicializa:

- Crosshair/Cursor
- Background e Status bar do jogo
- Main/Pause/Options/GameOver Menu
- Array de Pills
- *Sprite* utilizado no bónus
- Fonts a ser utilizadas

Este módulo foi desenvolvido por: Mariana Aguiar/Tiago Fragoso (70%/30%).

3.1.3 GameCtrl

Este módulo é o mais importante em termos de lógica e atualização do jogo. É também responsável por processar toda a informação proveniente dos periféricos e atualizar as máquinas de estados, variáveis de jogo e posições de objetos, assim como, testar colisões e definir a evolução do jogo. As principais funções deste módulo são "eventManager" e "statusManager".

- **eventManager:** Baseado no estado presente, guardado na struct "Status" do **GameModel**, esta função é responsável por chamar as funções relativas a cada estado, como, por exemplo, desenhar o menu no seu estado, ou desenhar e atualizar coordenadas de uma pill no jogo.
- **statusManager:** Baseado nas variáveis de jogo, guardadas na struct "Status" do **GameModel**, esta função é responsável por atualizar o estado do jogo.

Este módulo foi desenvolvido por: Mariana Aguiar/Tiago Fragoso (60%/40%).

3.1.4 Handler

Este módulo apenas contém as funções que subscrevem os *interrupts* dos periféricos utilizados, o loop de receção de *interrupts,* que, por sua vez, chama cada *interrupt handler*, presentes no **GameCtrl**. Em caso de saída do jogo, é neste módulo que é a subscrição de *interrupts* é desativada. Além disso é esta a função que chama a função "driver_receive".

Este módulo foi desenvolvido por: Mariana Aguiar/Tiago Fragoso (70%/30%).

3.1.5 Timer_modular

Módulo importado do código desenvolvido para o lab 2.

3.1.6 Keyboard

Módulo importado do código desenvolvido para o lab 3.

3.1.7 Mouse

Módulo importado do código desenvolvido para o lab 4.

3.1.8 Video_GR

Módulo importado do código desenvolvido para o lab 5, com implementação de double buffer.

As alterações foram feitas por Mariana Aguiar.

3.1.9 Vbe

Módulo importado do código desenvolvido para o lab 5.

3.1.10 Bitmap

Módulo responsável por importar, desenhar e apagar imagens bitmaps utilizados para toda a parte gráfica do jogo. O código deste módulo é da autoria de Henrique Ferrolho publicado no seu blog (source), com algumas modificações: a adição da deteção de transparência, alteração de tudo de *char* para *short*, e uma pequena alteração ao desenho de bitmaps com alinhamento central, já que não correspondia com as nossas necessidades em termos de implementação. Com a adição da deteção de transparência conseguimos desenhar só a parte dos bitmaps que pretendemos, o que se torna muito útil para imagens não quadradas ou rectangulares. A deteção de transparência é feita com a identificação da cor rosa puro (0xff40ff) em todos os pixéis da imagem, não desenhando certo pixel se a sua cor corresponder à de transparência.

As alterações a este módulo foram desenvolvidas por Mariana Aguiar e Tiago Fragoso (50%/50%).

3.1.11 MultiplayerCtrl

Este módulo contém a máquina de estados utilizada para o estado de sincronização de dois jogadores no modo “multiplayer”. A funcionalidade deste módulo já foi explicada em cima.

Este módulo foi desenvolvido por Mariana Aguiar/Tiago Fragoso (10%/90%)

3.1.12 rtcIntHandlerASM

Este módulo contém a função que lê o registo que contém a origem de *interrupts* do rtc.

Este módulo foi desenvolvido por Tiago Fragoso.

3.1.13 RTC

Este módulo é reponsável por ler os registos do Real Time Clock, auxiliado pelo módulo “rtcIntHandlerASM”, e atualiza a data a cada interrupt de *Update* do RTC. É realizada uma leitura individual de cada registo do RTC correspondente a cada parte da data e se necessária é realizada a conversão para binário.

Este módulo foi desenvolvido por Tiago Fragoso.

3 .1.14 Serialport

O módulo “Serialport” contém todas as funções que comunicam diretamente com o UART, assim como a implementação de duas SW queues (recQueue e sndQueue). Este módulo contém funções capazes de subscrever ou desativar a subscrição dos *interrupts* do UART, mudar a configuração do registo LCR e a bitrate a que o UART opera. Além disso, realiza todo o processamento de *interrupts* e leitura/escrita de informação das SW queues para o UART ou em sentido inverso. Para facilitar o uso do UART para transmitir informação, implementamos uma variável que contém o número de palavras completas na SW queue de receção, que é incrementada quando recebe o caractere ‘$’, que nós definimos como sendo o caractere de terminação das nossas mensagens. Esta implementação permitiu-nos ter uma função que retorna a primeira palavra completa na queue de receção, sem ser necessário operar sobre ela para descobrir se contém alguma palavra completa. Por outro

lado, qualquer mensagem enviada pela *serial port*, contém como último caractere o caractere de terminação.

Este módulo foi desenvolvido por Tiago Fragoso.

3.1.15 Proj

Este módulo é o que chama a função que inicia o jogo.

Este módulo foi desenvolvido por Mariana Aguiar.

3.1.16 I8042

Este módulo foi importado do código do lab 2/3/4.

3.1.17 I8254

Este módulo foi importado do código do lab 2.

3.1.18 Debug

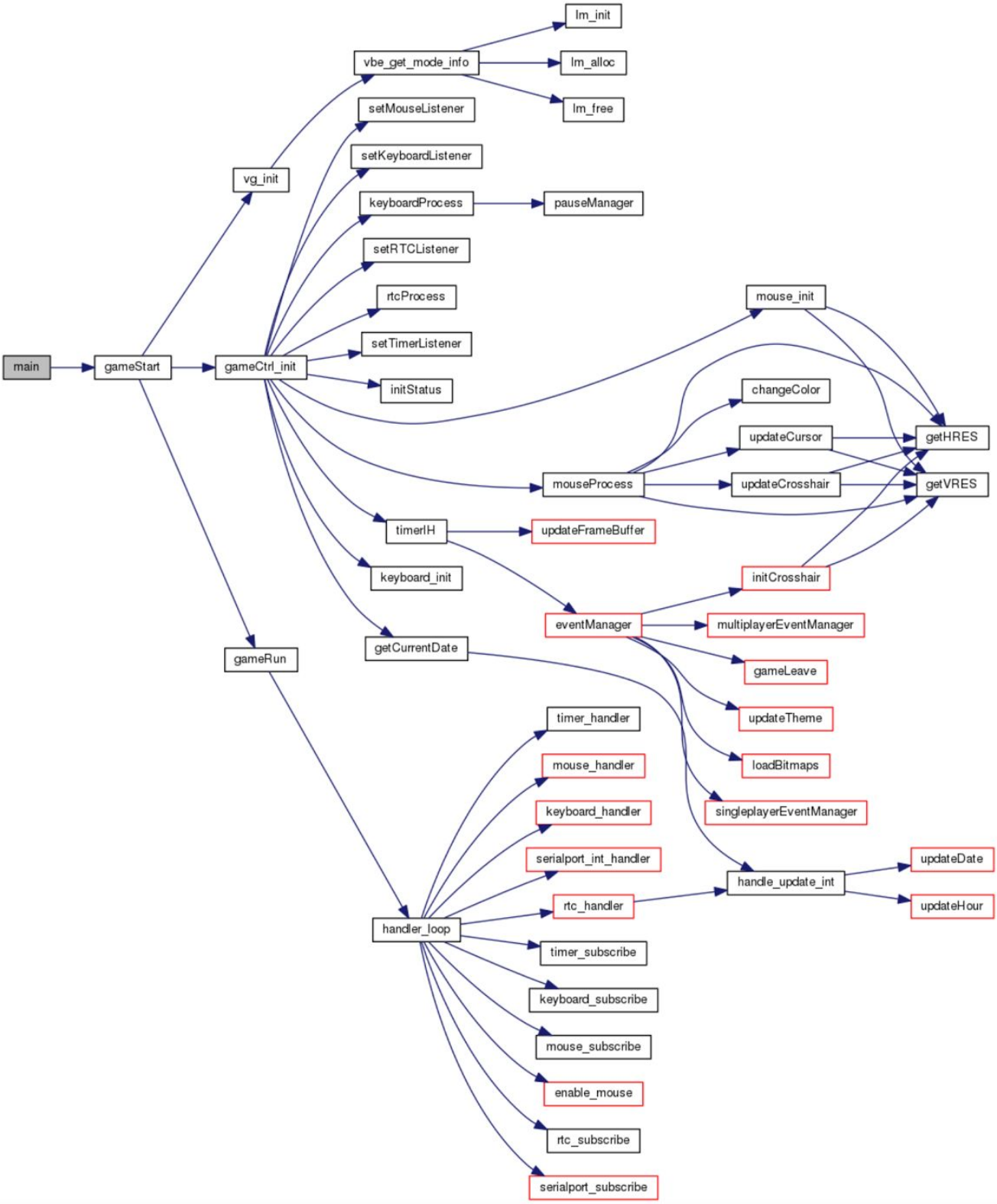
Este módulo foi importado do código do lab 3.

3.2 Peso dos módulos

A seguinte tabela representa a importância de cada módulo para o projeto.

Módulo	Peso no projeto (%)
GameCtrl	15
GameModel	13
Video_GR	12
Bitmap	9
Timer	9
Mouse	9
Serialport	7
MultiplayerCtrl	6
Handler	5
Keyboard	4
RTC	3
Game	2
Vbe	1
rtcIntHandler	1
Proj	1
I8042	1
I8254	1
Debug	1

3.3 Call graph



4. Detalhes de implementação

Achamos relevante mencionar um aspeto curioso do jogo que nos levou a utilizar uma implementação multiplayer bastante mais simples que muitos outros jogos. A forma como o jogo é desenhado baseia-se na aleatoriedade ou, pseudo-aleatoriedade para ser mais correto, isto é, os números aleatórios da função *rand()* são gerados baseados na seed utilizada na função *srand().* Nós optamos por utilizar o RTC para gerar essa seed, quer para single como multiplayer. Utilizando uma soma de todos os valores da data, podemos gerar uma seed, e sempre que se começa um novo jogo essa seed é recalculada. Como toda a variabilidade do jogo se baseia nos números gerados pela função *rand(),* utilizando a mesma seed, o jogo será exatamente o mesmo. Tirando partido disto, em modo multiplayer, os 2 jogadores, têm um jogo, teoricamente, igual. No entanto, como o jogo evolui em função do score, um dos jogadores pode aparentar ter um jogo diferente, nomeadamente, o jogador com menor score aparenta ter um jogo mais fácil, porque a *spawn rate* não é tão elevada. Isto revela-se também interessante já que torna possível a reviravolta dentro do modo multiplayer, visto que um começo fraco de um jogador leva a que o seu jogo evolua mais lentamente, podendo assim ainda sair vitorioso.

Achamos também importante realçar a importância das 4 máquinas de estado implementadas (estado do programa, estado do jogo singleplayer, estado do jogo multiplayer e multiplayer lobby), que permitem contornar o facto de o programa ser *single threaded*, podendo a cada iteração realizar diferentes operações, dependendo do estado do programa.

5. Conclusão

Ambos sentimos que o aproveitamento de todas as aulas foi positivo, principalmente das aulas laboratoriais pois foram estas que nos permitiram estar suficientemente preparados para o desenvolvimento deste projeto.

Apesar de considerarmos esta Unidade Curricular exigente e desafiante, ambos a achamos muito interessante, pois podemos ver de perto e perceber como é que é realizado o processamento de por exemplo o rato e o teclado, que usamos no dia a dia sem pensar como funcionam.

No entanto, pensamos que a dificuldade e exigência da Unidade Curricular Laboratório De Computadores não corresponde ao número de créditos que lhe estão atribuídos, usando como comparação outras cadeiras com igual número de créditos deste semestre, visto que estes se devem refletir maioritariamente com o tempo gasto na unidade curricular.

Também gostaríamos de mencionar que foi muito útil poder consultar o blog do Henrique Ferrolho, pois agilizou muito o desenvolvimento da parte da interface gráfica do jogo.

Por último, achamos por bem referir que ambos os membros do grupo se empenharam de igual forma para o desenvolvimento do deste projeto e que ambos dedicaram muitas horas para que todos os objetivos fossem atingidos e que todos os desafios e problemas fossem ultrapassados, tendo sido o empenho dos dois igual.

Apêndice

Instruções de instalação

Para facilitar a instalação do projeto, fornecemos alguns scripts dentro da pasta proj/scripts, que fazem a instalação dos *assets* do projeto, nomeadamente bitmaps para uma localização neutra, que será utilizada com path absoluto no código do programa.

Assim, na primeira instalação do programa deve ser corrido o script "**install.sh**" em *root*, já que este copia os *assets*, mas também o ficheiro de configuração do programa para /etc/system.conf.d.

Depois, basta correr os scripts "**compile.sh**" e "**run.sh**", que efetuam a compilação e execuação do programa, respetivamente, relembrando que é necessário estar em *root* para realizar a execução.

Resumindo:

Dentro da pasta proj/conf, correr:

1. **# sh run.sh** (1ª instalção, obrigatório realizar em root)
2. **$ sh compile.sh**
3. **# sh run.sh** (obrigatório realizar em root)