DUCK HUNT

Projeto de Laboratório de Computadores

Autoria de:

José Diogo up202003529
Miguel Teixeira up202005208
Luís Paiva up202006094

Licenciatura em Engenharia

Informática e Computação

21 / 22

Índice

Introdução

Projeto realizado no âmbito da unidade curricular Laboratório de computadores do 2º ano de LEIC, utilizando diversas ferramentas de desenvolvimento de software como, Minix 3, Visual Studio Code, Oracle VM Virtualbox e a linguagem de programação C.

Este projeto consiste num vídeo jogo inicialmente concebido para a Nintendo NES, Duck Hunt. No jogo o jogador deve atirar em patos voadores que tentam fugir pelo topo do ecrã, com isso, ganhando pontos e aumentando a sua dificuldade ao longo do tempo. O jogador perde assim que três aves consigam fugir. Sendo assim, o jogo adquire um fim pseudo infinito, no entanto, a cada 100 pontos a velocidade aumenta, portanto, chegará um momento onde o jogador não conseguirá manter-se “vivo”, acabando por fim a sua jogada.

Secção 1: Funcionalidades

Menu Inicial

No Menu inicial temos três opções: Start, Rules, Exit. Para escolher uma das opções o utilizador deve usar as setas do teclado, assim que opção desejada esteja selecionada deve pressionar a tecla ENTER.

- **Start:** inicializa o jogo, ou seja, altera o estado do programa para GamePlaying.
- **Rules:** é apresentado ao utilizador um pequeno resumo das regras e controlos para o jogo.
- **Exit**: Altera o estado do programa para Exit, encerrando o jogo.

Rules

O jogo consiste em tentar matar o máximo número de patos possível, enquanto o jogador ainda tem vidas. O jogador começa com 3 vidas, e perde 1 por cada pato que deixa escapar pelo topo do ecrã.

Inicialmente parece uma tarefa simples, mas por cada pato que o jogador consegue matar, recebe 10 pontos, e a respetiva velocidade do jogo aumenta por uma unidade a cada 100 pontos obtidos, o que rapidamente torna a tarefa muito mais difícil. O jogo acaba quando o jogador deixa escapar 3 patos.

Para apontar e disparar, o jogador deve movimentar o rato e pressionar o botão do lado esquerdo quando pretende disparar. É também possível a qualquer momento voltar para o Menu principal carregando na tecla ESC e também colocando o jogo em pausa utilizando a tecla P.

Ciclo de jogo

Durante a execução, o ciclo de jogo passa por diversos estados: GamePlaying, GamePaused e GameOver.

Com a implementação de uma estrutura baseada em estados, aumentamos a praticidade e a compreensão geral do funcionamento.

Foram também implementados os estados: Menu e Exit, que são utilizados precisamente para abrir o menu e para sair do jogo.

GamePlaying

Quando a opção Start é selecionada, o jogo passa para o estado GamePlaying.

Aqui é onde se passa toda a ação, o jogador pode atirar sobre as figuras, tanto quanto ele conseguir. Por outro lado, pode também pressionar a tecla P para passar para o estado GamePaused, ou perder o jogo indo para o estado GameOver.

GamePaused

Neste estado o jogador não pode, essencialmente, fazer nada. Destina-se simplesmente a, logicamente, pausar o jogo. Tudo fica em “espera”, desde as vidas do jogador, pontuação, posição dos patos, etc. Após pressionar a tecla P, o jogador retornará ao estado anterior.

GameOver

Para atingir este estado, só existe uma forma. Para isso o jogador terá de não conseguir matar três patos, deixando-os fugir pelo topo do ecrã. Uma vez que esta condição é atingida, é mostrado no ecrã uma mensagem de fim de jogo (Game Over), sendo ainda possível ver a pontuação adquirida.

É ainda possível carregar na tecla R para voltar a jogar o jogo do zero, ou clicando na tecla ESC retornamos para o Menu.

Pontuação

Existe também um sistema de pontuação.

Durante o desenvolvimento do jogo, o jogador ganha 10 pontos ao abater com sucesso um pato. Esta pontuação vai aumentando a cada abate e pode ser vista no canto inferior direito do ecrã.

A cada 100 pontos a velocidade de movimento dos patos é incrementada, aumentando assim a dificuldade de acerto. O jogador pode obter o máximo de pontos que conseguir.

Vidas

A utilização de vidas corresponde ao único sistema que contribui para o utilizador perder o jogo. Inicialmente o jogador começa com 3 vidas, e vai perdendo uma por cada pato que deixa escapar pelo topo do ecrã.

O utilizador consegue visualizar quantas vidas possui no momento, pois um coração cheio (vermelho) demonstra que contêm essa vida, e um coração vazio demonstra que já perdeu essa vida(preto).

Secção 2: Status do Projeto

Tecnologias implementadas

Dispositivos	Implementação	Interrupções
Timer	Movimentação dos sprites e atualização do jogo	Sim
Graphic Card	Apresentar a interface do jogo	N/A
Keyboard	Seleção de opções pelo utilizador	Sim
Mouse	Mira de disparo do jogo	Sim
RTC	Alteração do background dependendo da hora do dia	Não

Graphics Card

A Graphics Card é a interface de vídeo que se encarregará de desenhar todo o nosso jogo no ecrã. Sem ela seria impossível criar qualquer tipo de imagem, portanto, é essencial no contexto deste projeto.

Neste projeto foi utilizado o modo gráfico 0x14C, com resolução de 1152x864 e Direct Color Mode, sendo que a distribuição de RGB de bits por pixel é (8) 8:8:8 (dando um total de 24 bits por pixel).

Para ativar o modo gráfico, é utilizada a função vg_init(), e no final da execução é invocada a função da LCF vg_exit() para retornar o modo da gráfica para texto.

De forma que a construção de imagem seja feita de uma forma mais fluida (de modo a não ser notório para o utilizador problemas como flickering da imagem), tiramos partido da utilização de double buffering, que consiste em desenhar as imagens num buffer secundário (double buffer) e a cada interrupção do timer copiar todo o conteúdo do double buffer para o main buffer (onde é desenhado para o utilizador), utilizando a função copyDoubleBufferToMain().

Keyboard

O keyboard é utilizado para controlar as interfaces do menu, e também para controlar alguma lógica associada ao jogo, como o sistema de pause e a possibilidade de sair do mesmo a qualquer momento.

Sempre que ocorre uma interrupção do kbc, é invocado o seu IH (interrupt handler) e se não ocorrer nenhum erro, é utilizada a informação retirada do mesmo (out byte) para processar as diferentes funcionalidades descritas acima.

Timer

O timer é um dispositivo integrado que gera interrupções a uma dada frequência. Desta forma, usamos o timer para controlar todas as estruturas do jogo, como por exemplo atualizar o estado do jogo, fazer a troca entre os buffers (double buffering), atualizar o vídeo RAM, entre outras.

A maior parte da lógica necessária para o jogo ocorre a cada interrupção do timer, como por exemplo a movimentação das diferentes sprites utilizadas e a verificação se houve colisão da mira com algum pato e controlar o intervalo pelo qual são criados novos patos.

A frequência do timer escolhida foi de 60Hz, isto é, gera 60 interrupções por segundo. Esta taxa foi escolhida para que o desenvolvimento do jogo fosse feito de forma que não seja notória a atualização da vídeo RAM e não prejudicasse a jogabilidade.

Mouse

O mouse é utilizado durante o ciclo de jogo para controlar a mira utilizada. Um deslocamento quer horizontal quer vertical resulta na atualização do sprite da crosshair. Um clique com o botão do lado esquerdo significa que o utilizador tentou disparar uma bala.

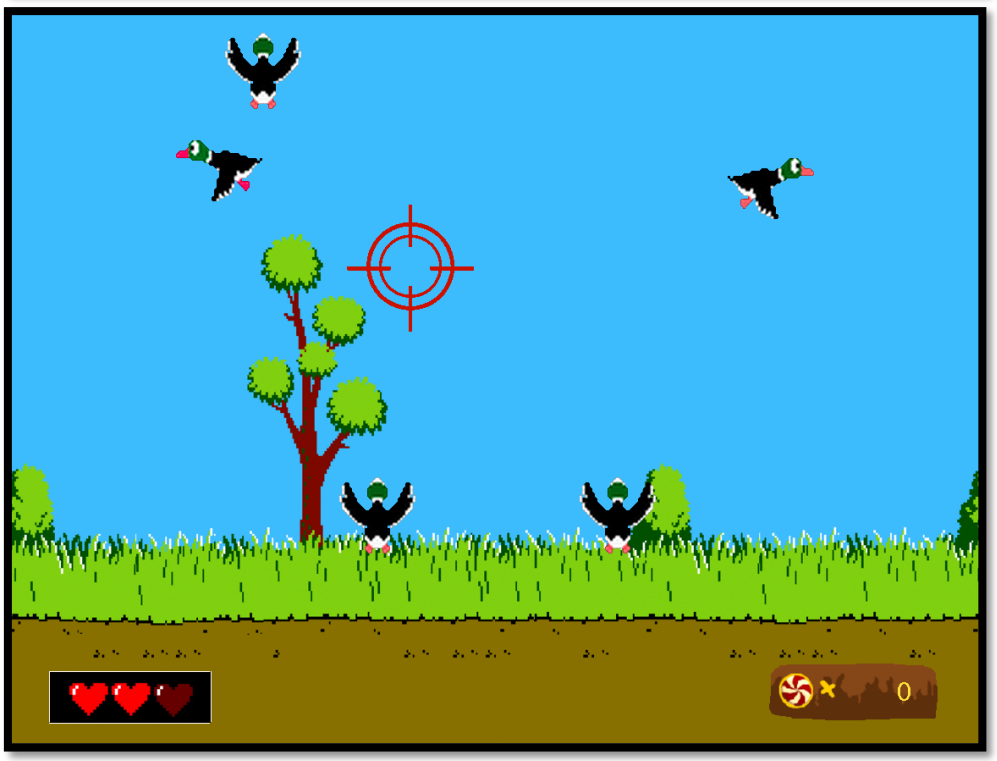
A cada interrupção do rato, é chamado o seu mouse_ih(), sendo gerado um “packet byte”. Ao fim de 3 interrupções, está concluído um packet completo. Esse packet no nosso programa é então utilizado para dar update na sprite da mira e verificar se houve um clique do botão esquerdo.

O processamento do rato só é feito caso o currentState do programa seja igual a GamePlaying, que é o único estado em que o rato é utilizado.

RTC

O Real Time Clock é um circuito integrado que contém a data e o dia atual. No nosso projeto a sua utilização baseia-se em ler qual o valor atual do tempo, utilizando a função getCurrentTime(), e com isso atualizar respetivamente o background do jogo.

Assim, é desenhado o background de noite caso a hora seja inferior a 07h30 ou superior a 22h30, e o background de dia em caso contrário.



Secção 3: Organização/Estruturação

Sprite.c / Sprite.h

Este módulo destina-se à criação de Sprites (imagens). As sprites constituem uma struct que guarda as seguintes informações: posição no eixo x, posição no eixo y, largura, altura, velocidade para o eixo x, velocidade para o eixo y, o pixel map corresponde / atual, a sua direção e estado.

É aqui que implementamos todas as funções de manipulação da struct Sprite, de forma que a todas as suas características sejam passíveis de ser alteradas facilmente e de um modo fácil e organizado.

Para isso utilizamos as seguintes funções: create_sprite(), destroy_sprite(), draw_sprite(), change_Sprite_Img().

Contribuição:

- **José Diogo: 40%**
- **Miguel Teixeira: 30%**
- **Luís Paiva: 30%**

Peso relativo no projeto: 10%

Hitboxes.c / Hitboxes.h

Este módulo é utilizado para verificar colisões de sprites e potencialmente atualizar o estado dos ducks. A função de deteção de colisões encontra-se um pouco especializada pois calcula a deteção de colisão do centro da sprite da mira com qualquer ponto no interior de uma sprite duck, pois desta forma oferece uma certa sensação de maior dificuldade, pois os tiros necessitam de ser mais precisos.

Contribuição:

- **José Diogo: 40%**
- **Miguel Teixeira: 30%**
- **Luís Paiva: 30%**

Peso relativo no projeto: 5%

Crosshair.c / Crosshair.h

Aqui encontramos uma função denominada update_mouse() que nos determina a posição exata do mouse no ecrã, isto é, a mira, e verifica se o botão da esquerda do mouse está a ser pressionado, uma vez pressionado significa que o utilizador deu um tiro, sendo essa informação necessária para o sistema de atualização de pontuação, como também para a atualização do estado do pato atingido.

Contribuição:

- **José Diogo: 33.3(3) %**
- **Miguel Teixeira:33.3(3) %**
- **Luís Paiva: 33.3(3) %**

Peso relativo no projeto: 5%

Duck.c / Duck.h

Esta parte é fundamental ao nosso jogo, uma vez que é nela onde escolhemos o local onde o pato deve nascer, a direção para onde deve prosseguir, usando as funções update_direction() e generate_random_dir(), e por fim onde atualizamos todas características dos patos, como por exemplo: a sua velocidade, o seu estado(vivo/morto) etc, recorrendo à função update_Duck().

Contribuição:

- **José Diogo: 35%**
- **Miguel Teixeira: 35%**
- **Luís Paiva: 30%**

Peso relativo no projeto: 15%

i8042.h

Código proveniente dos labs 3 e 4, contendo macros utilizadas no tratamento do mouse e também do keyboard.

Peso relativo no projeto: 0.5%

i8254.h

Código proveniente do lab 2, contendo macros utilizadas no tratamento do timer.

Peso relativo no projeto: 0.5%

timer.c / timer.h

Código proveniente do lab 2, contendo as diversas funções utilizadas no tratamento do timer.

Peso relativo no projeto: 5%

kbc.c / kbc.h

Código proveniente do lab 3, contém as diversas funções necessárias para utilizar interrupções do kbc, e consequentemente utilizar o rato e teclado. Contêm também macros para os breakcodes de teclas utilizadas no nosso projeto.

Peso relativo no projeto: 2%

mouse.c / mouse.h

Código proveniente do lab 4, contém as diversas funções necessárias para utilizar o rato, bem como algumas macros para facilitar o processamento de mouse packets.

Peso relativo no projeto: 5%

video_gr.c / video_gr.h

Contém código proveniente do Lab 5, contendo diversas funções relacionadas com a utilização da placa gráfica e manipulação da VRAM, essenciais para o bom funcionamento da parte do projeto visível para o utilizador.

Para além das funções provenientes dos labs, foram acrescentadas alguns getters para facilitar a utilização de variáveis como horizontal / vertical resolution, mas também toda a implementação do mecanismo de double buffering, que foi um aspeto crucial para manter um bom funcionamento visual do programa.

Peso relativo no projeto: 15%

rtc.c / rtc.h

Contém código utilizado para interagir diretamente com o Real Time Clock. Possui funções para fazer a correta extração dos valores das horas e dos minutos, uma função para verificar se os valores extraídos se encontram em binário ou em complemento para dois, e uma função de conversão de modo binário para BCD.

Peso relativo no projeto: 1%

utils.c / utils.h

Contém algumas funções auxiliares maioritariamente utilizadas pelo código do lab 2 e 3, não utilizadas “diretamente” no projeto.

Peso relativo no projeto: 1%

Pasta Images

Contém todas as xpms utilizadas pela nossa aplicação, desde sprites dos patos e da mira, botões de menu, os dígitos utilizados, por exemplo para o score, os diferentes backgrounds, etc.

Peso relativo no projeto: 5%

database.c / database.h

Aqui encontram-se todas as funções relativas à manipulação da base de dados, sendo esta, uma estrutura que serve como um armazenamento central para todos os dados importantes do programa. A base de dados guarda toda a informação utilizada pelo programa, tais como todas as sprites, imagens utilizadas, número de vidas que o jogador possui, pontuação, estado atual no qual o programa se encontra, qual a velocidade à qual o jogo se encontra (diretamente relacionado com a dificuldade), etc.

Como também, é responsável pela criação de objetos, desenho de todos os componentes utilizados pelo jogo, desde o background, patos, scoreboard, o container das vidas (etc), mas também a atualização e manipulação das diversas variáveis de jogo.

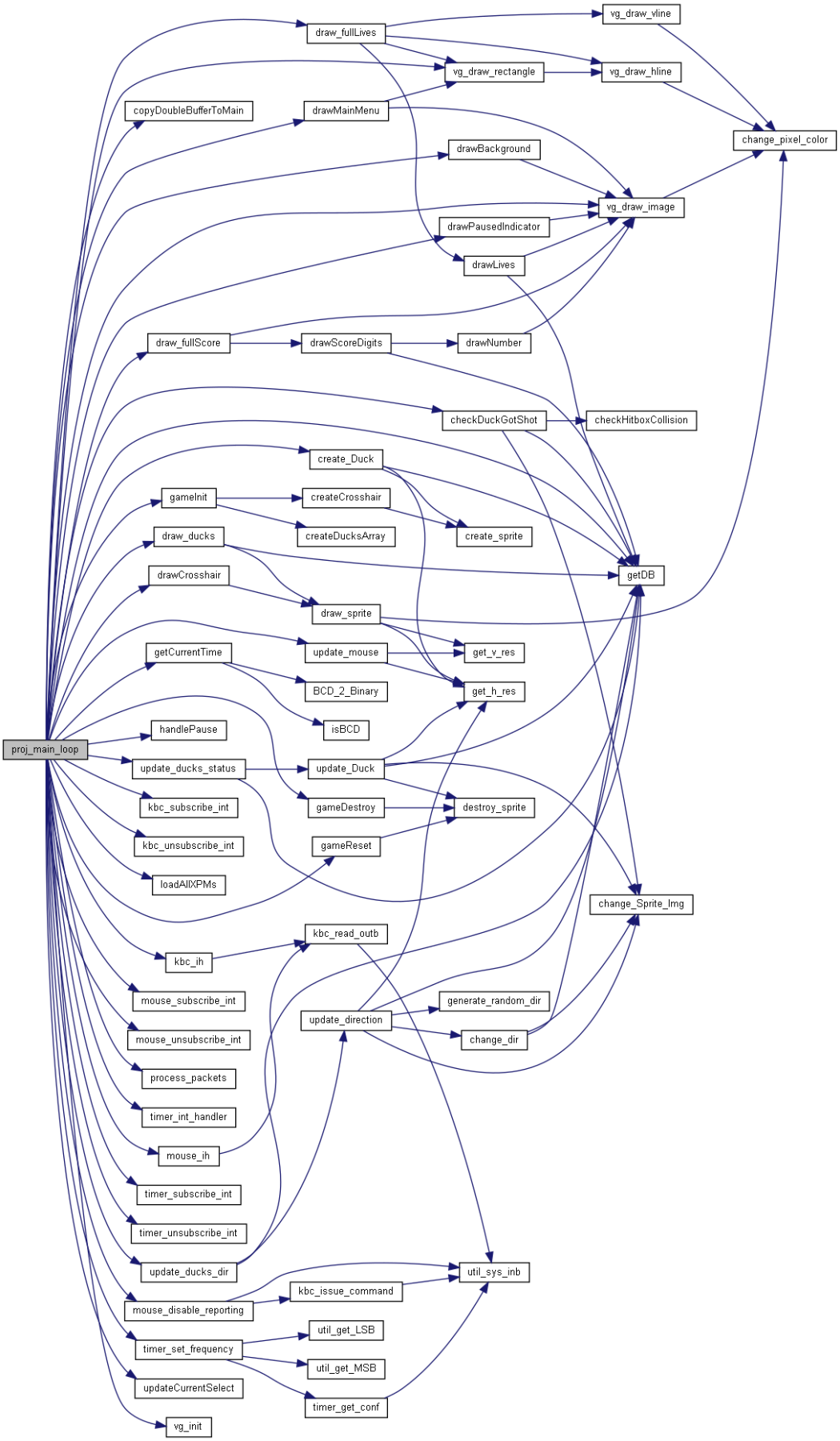
Suporta para além disso operações como gameInit() (função utilizada para inicializar todas as variáveis do jogo para os valores default desejados, mas também para alocar a memória necessária para o array de ducks), gameReset() (para após elas terem sido utilizados pelo modo jogo, poderem voltar ao seu estado inicial), e gameDestroy() (que liberta toda a memória que foi alocada para as variáveis e structs utilizadas pelo jogo).

Contribuição:

- **José Diogo: 40 %**
- **Miguel Teixeira: 30 %**
- **Luís Paiva: 30 %**

Peso relativo no projeto: 30 %

Gráfico de Chamada de Funções



Detalhes de Implementação

Estruturação do código

A estruturação geral do código foi essencial para manter uma boa qualidade de código e robustez. Foi feita uma construção do código *layered*, isto significa que separamos as diversas funções e estruturas por vários ficheiros, separando mais ou menos assim o código pelas suas funções, o que resultou num código mais fácil de compreender, interpretar e modificar.

A separação das diferentes componentes do código, como os ficheiros .h na pasta Include, os diferentes ficheiros .c dispositivos de entrada e saída na pasta Devices também contribuiu para a organização do código.

Movimentação e *update* dos patos

Apesar de ser um jogo simples, a movimentação dos patos não poderia ser "irrealista", ou seja, se um pato está a movimentar-se para a esquerda, não faria sentido no movimento a seguir estar a movimentar-se para a direita, isso causaria um movimento brusco, o que não seria, tanto visualmente como mecanicamente, interessante para o jogador.

Devido a esta questão, decidimos implementar um sistema de geração de novas direções, restringindo cada direção aos seus possíveis movimentos. Como por exemplo, se a direção atual é direita então, a próxima só poderá ser direita na diagonal para cima (visto que não desejamos movimentos para baixo).

Para implementar esta ideia, foram criados vários arrays que guardam as possíveis direções tendo em conta a direção anterior. De seguida, é gerado um número aleatório que escolherá a direção seguinte, dentro do array de possibilidades.

```
enum Direction up[] = {Right, Left, Up_Right, Up_Left, Up};
enum Direction right[] = {Right, Up_Right};
enum Direction left[] = {Left, Up_Left};
enum Direction upright[] = {Up_Right, Right, Up};
enum Direction upleft[] = {Left, Up_Left, Up};
```

Double Buffering

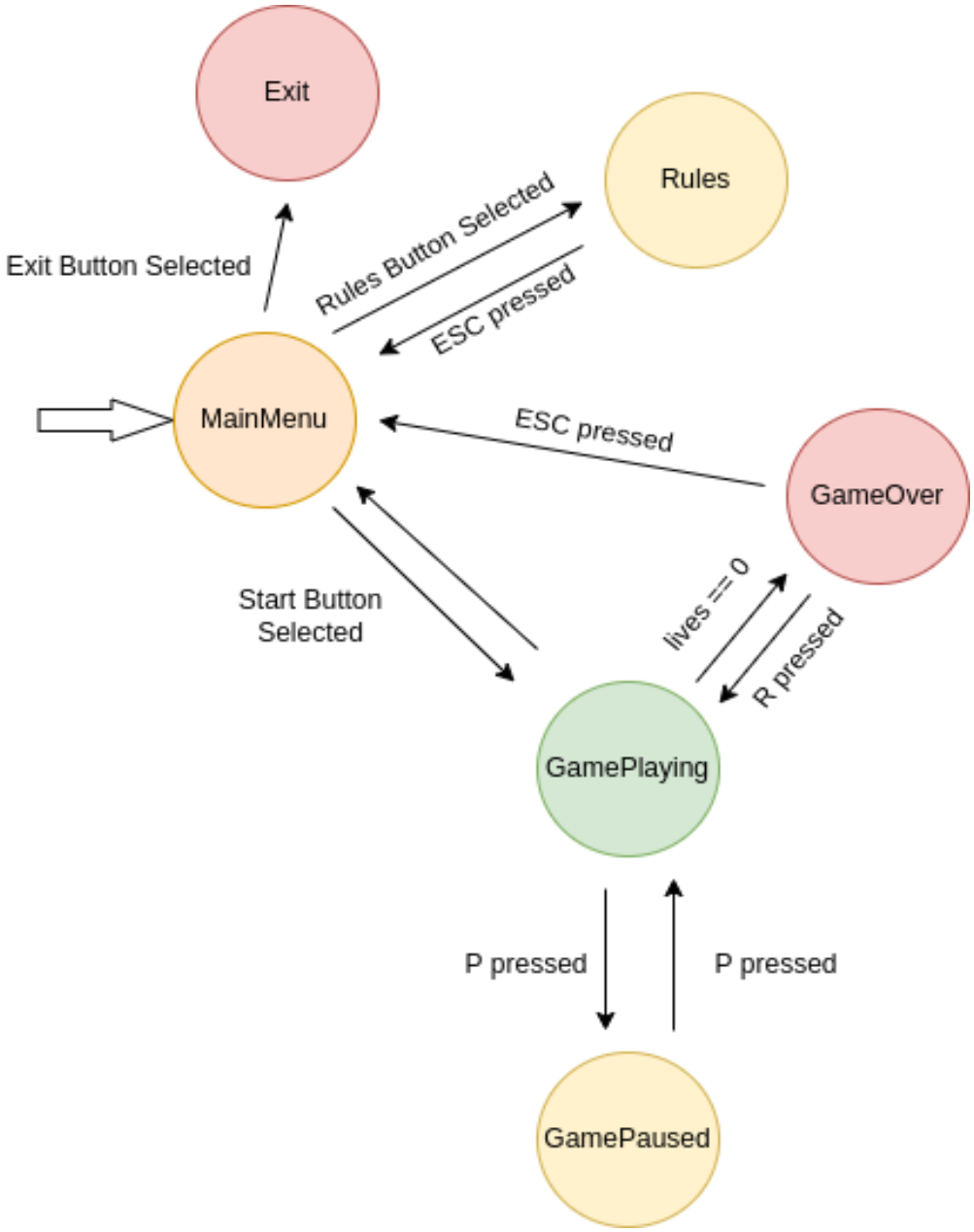
O double buffering consiste na criação de dois buffers. Um é o buffer atual, que possui a imagem desenhada, o outro recebe a informação da próxima imagem a ser desenhada. Desta forma, aumentamos a eficiência do código e evitamos problemas de flickering.

Deteção de colisões

Embora não tenha sido um aspeto abordado nas aulas, o facto de conseguirmos fazer uma boa organização das informações obtidas através das mouse packets, resultou num bom mecanismo de deteção de colisões da mira (rato) com os diferentes ducks, e transmite uma boa sensação que quando acertamos num pato, acertamos verdadeiramente nele.

State Machines

Foi criada uma state machine com os seguintes estados: Menu, Menu Rules, Game Playing, GamePaused, GameOver, Exit. Assim, tornou-se mais simples e prática a implementação de cada estado e a utilização correta das funções implementadas.



Conclusões

Para concluir, consideramos que este projeto foi essencial para consolidar os conhecimentos adquiridos nesta UC, pois foi uma experiência engraçada e desafiante utilizar todos os mecanismos de interrupções e não só dos diferentes dispositivos de entrada e saída para a construção de algo mais aplicado ao mundo real, em vez de estar apenas a programar os labs que temos vindo a fazer, que tinham um objetivo bem definido e guiões para nos ajudar.

Gostaríamos de acrescentar que conseguimos implementar tudo aquilo que tínhamos planeado inicialmente como obrigatórias, no entanto, existe sempre espaço para melhorar.

Algumas mudanças que tínhamos ponderado como opcionais, pensamos que poderiam ajudar o jogo a adquirir uma experiência ainda mais completa, tais como: a implementação de movimento nas asas, ou um sistema de limitação da munição.

Sentimo-nos realizados com este projeto e acreditamos que demos o nosso melhor para a sua concretização.

Contribuição total de cada elemento

José Diogo: 40%

Miguel Teixeira: 30%

Luís Paiva: 30%