U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Faculdade de Engenharia da Universidade do Porto

CROMOPARTY

Laboratório

de Computadores

2ºANO - **MIEIC**

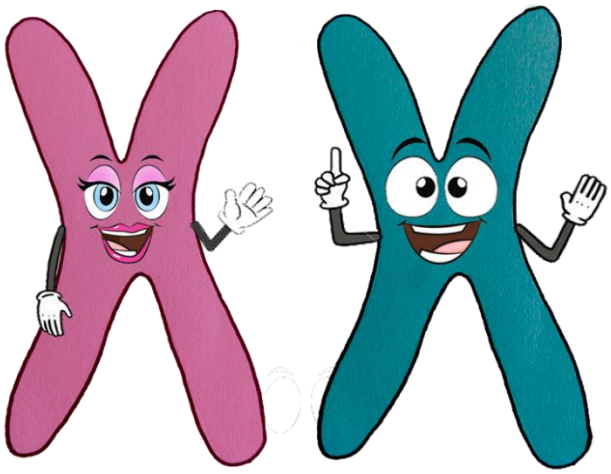
Estudantes e Autores

Diogo Silva, up201706892

up201706892@fe.up.pt

Ana Mafalda Santos, up201706791

up201706791@fe.up.pt



1 de **janeiro** de **2019**

2MIEIC**02_11**

Índice

Tabela de Figuras

Tema

O nosso jogo baseia-se na muito conhecida tipologia de jogo "Dance Dance Revolution", mas com algumas diferenças, inspiradas em **biologia**! Tentamos assim desenvolver um jogo que não só se destacasse pelo seu tema e jogabilidade, mas também pelo seu caráter didático.

A nossa personagem principal trata-se de um autossoma, **X** ou **Y** (dependendo da escolha do utilizador), que ao longo do jogo vai dançando ao ritmo da música.

O cromossoma a "dançar" pretende simbolizar o esforço para concretizar as diferentes fases da divisão celular – **meiose**.

Instruções

Menu Inicial

Antes de tudo, o jogador é confrontado com uma caixa de texto e a instrução "Enter your name!" (insira o seu nome). O jogador tem aqui a oportunidade de associar um *nickname* às pontuações que irá obter ao longo dessa mesma sessão de jogo.

Nota: Tem de ser uma palavra apenas (sem espaços) e não pode ser inserido um vazio (se o fizer, quando clica na tecla *Enter* é confrontado com o aviso "empty name!" (nome vazio). Pode utilizar no máximo 20 caracteres e este é corrigível através da tecla *Backspace*. Este menu é acedível sempre que necessário através do **Menu Principal** (explicado de seguida), sendo possível alterar o nome associado ao jogador dentro de uma mesma sessão de jogo.

Figura 1 *Menu Inicial*

Menu Principal

Após realizar a operação anterior com sucesso, é apresentado este menu ao jogador. A navegação neste é possibilitada através das teclas **W** e **S**, para percorrer o menu para cima e para baixo, respetivamente. Existem 5 opções de escolha:

- **SinglePlayer**, que como a palavra indica, inicia um jogo individual.
- **MultiPlayer**, inicia um jogo semelhante ao individual, mas que pode ser jogado em dois computadores distintos, sendo possível o acompanhar a pontuação atual do adversário.
- **High Scores**, onde é dada a possibilidade de consultar as seis melhores pontuações guardadas até ao momento.
- **Instructions**, para consultar as instruções do jogo.
- **Exit**, que, mais uma vez, como a palavra indica, sai do jogo.

Como indicado anteriormente, é permitido ao jogador alterar o *nickname* que escolheu e, caso pretenda fazê-lo, apenas necessita de clicar na tecla *Esc*, voltando ao **Menu Inicial** atrás referido.

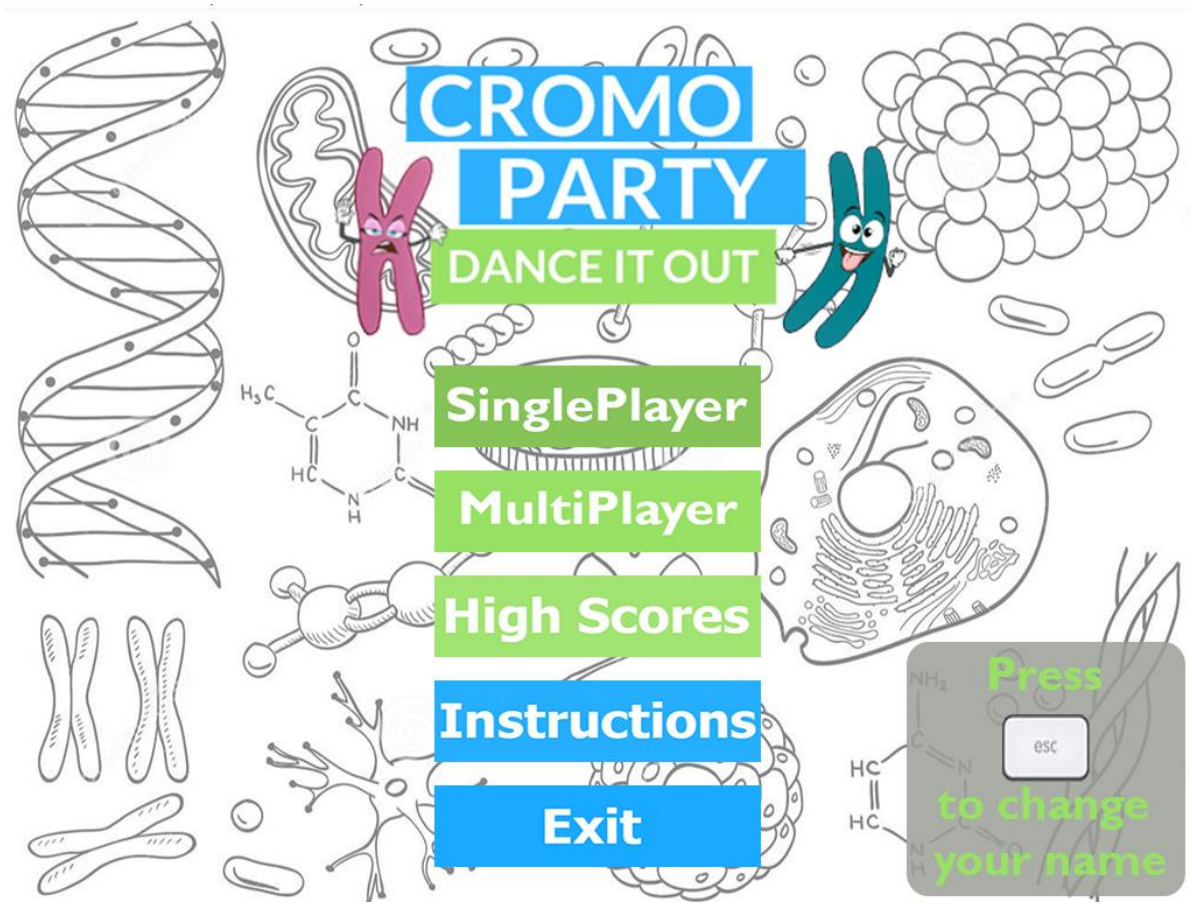


Figura 2 *Menu Principal*

Menu das Personagens

Atribuído o *username* e escolhido o modo de jogo entre os dois possíveis, o jogador tem a oportunidade de selecionar com qual dos autossomas – **X** ou **Y** – quer jogar, cada um com danças e movimentos únicos. Se pretender pode também retroceder para o **Menu Principal** clicando na tecla *Esc*.

Esta escolha vai apenas alterar a personagem que é apresentada durante o jogo.

Tanto no modo **SinglePlayer** como no modo **MultiPlayer** é dada ao jogador a escolha do protagonista que prefere, sendo que neste último é obrigatória a seleção de duas personagens distintas.

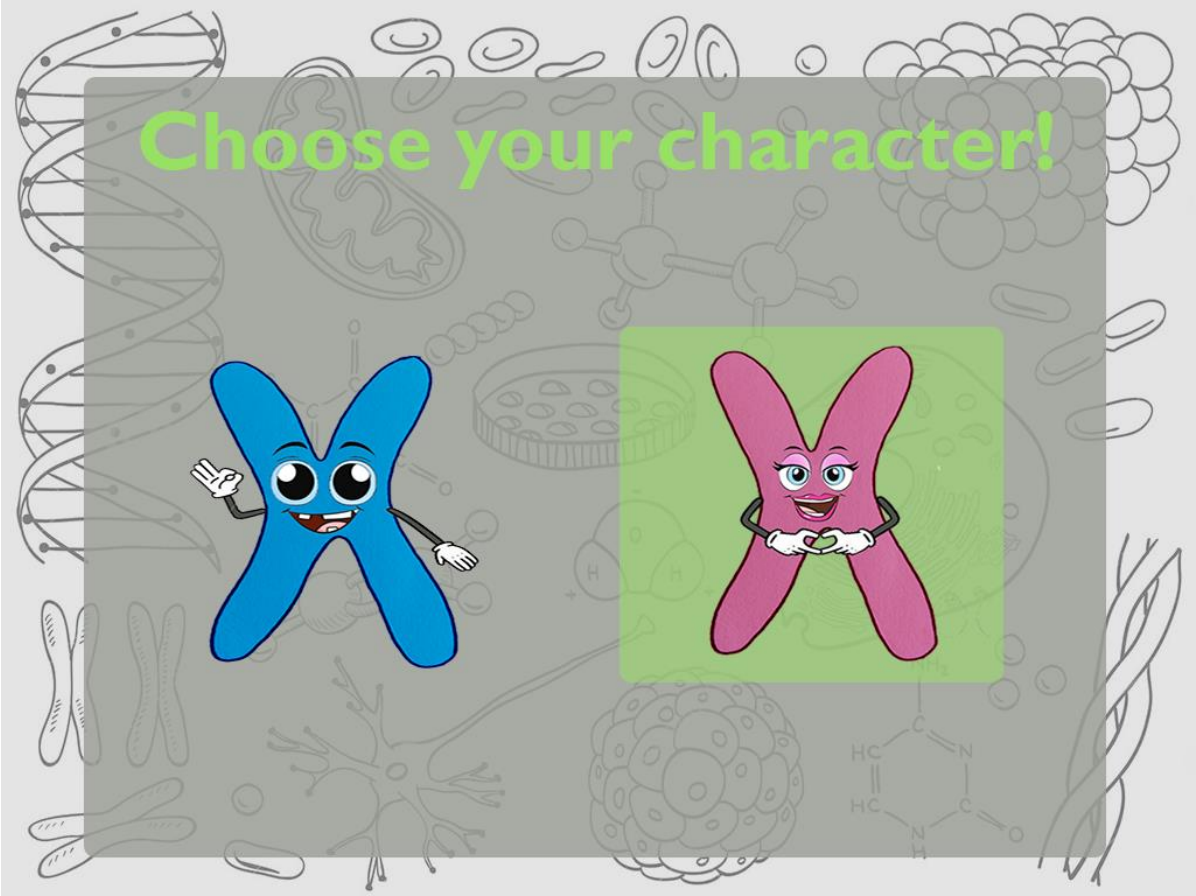


Figura 3 *Menu das Personagens*

SinglePlayer

Após fazer a sua escolha, começa o jogo. O seu objetivo é acertar na tecla com a direção que é mostrada no círculo em movimento quando esta se alinha perfeitamente com o círculo preto central e desta forma arrecadar o maior de número de pontos que consiga.

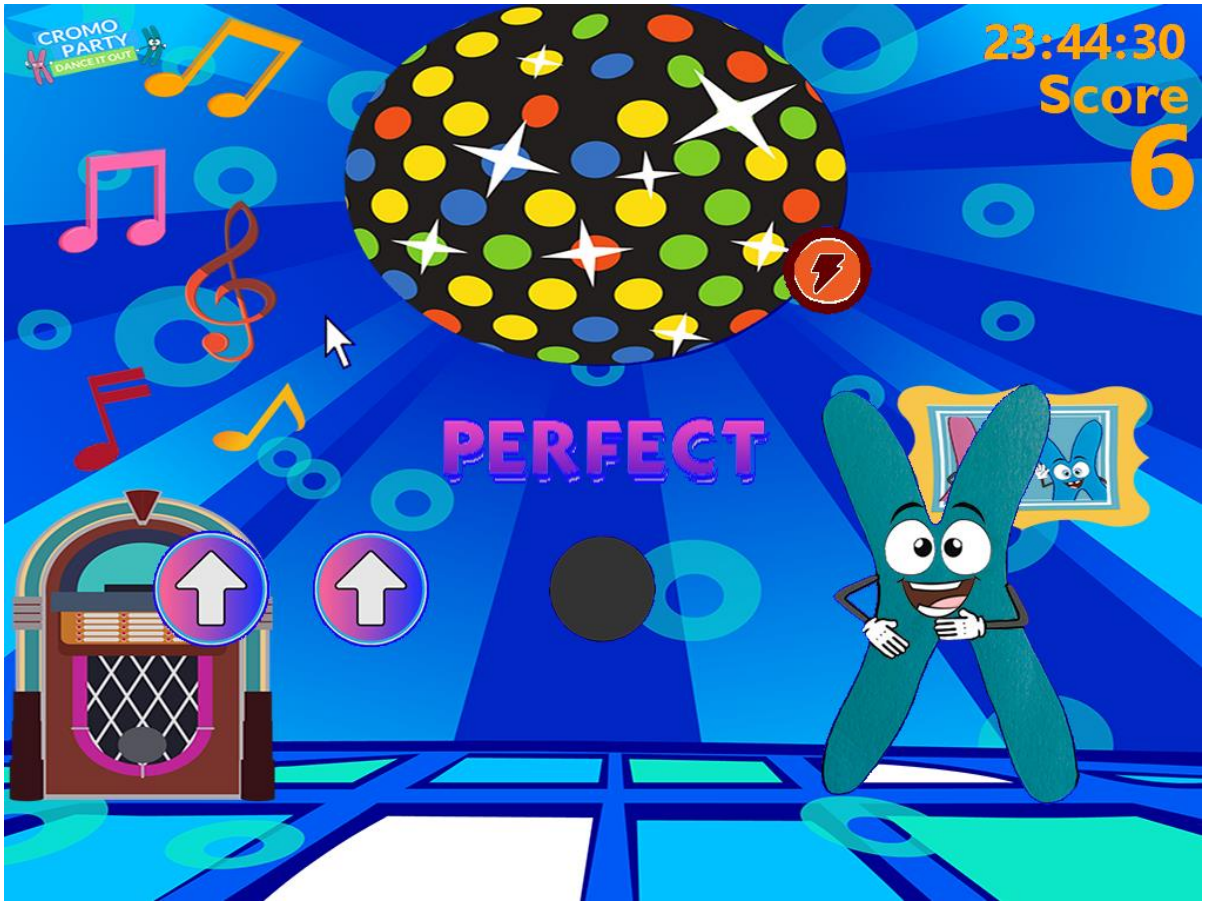


Figura 4 *Jogo a Decorrer*

Caso não falhe na tecla, passa-se à avaliação da performance. Dependendo da distância a que esta fica do círculo, terá pontuações diferentes, que podem variar entre *miss*, *okay*, *great* e *perfect*. Caso obtenha a classificação mínima de *okay*, o cromossoma executa um passo de dança de acordo com a seta gerada.

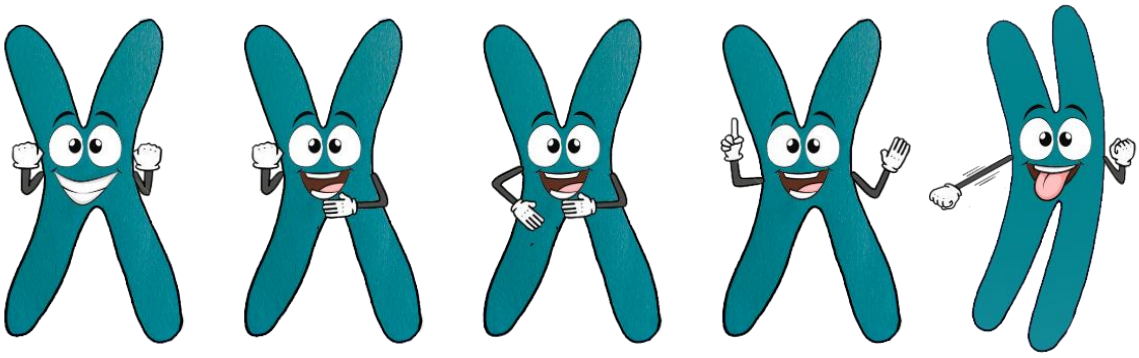


Figura 5 *Cromossoma Y*

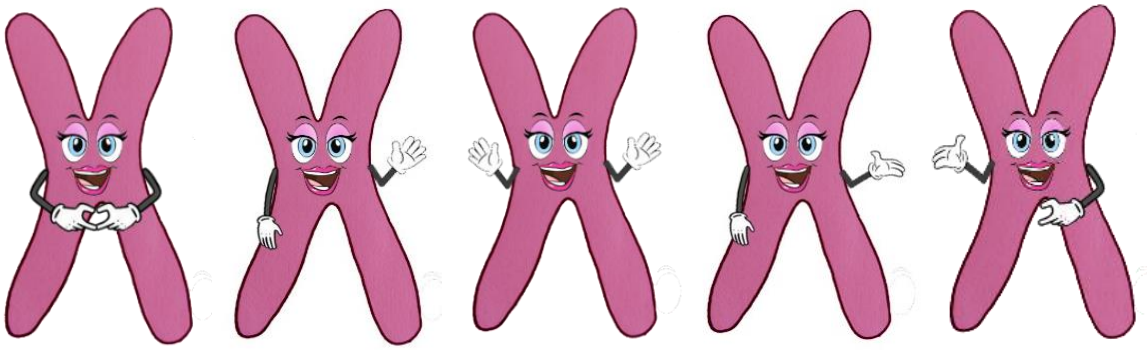


Figura 6 *Cromossoma X*

Para aumentar a dificuldade do jogo, decidimos criar um *power up* de forma a permitir que os jogadores mais ávidos alcancem pontuações superiores. Este movimenta-se na diagonal e horizontal, colidindo não só com os limites da tela, como também com diversos objetos espalhados pela mesma.

Tanto as posições como o instante em que o *power up* surge são determinadas de forma aleatória, o que lhe confere um caráter imprevisível.

Figura 7 *Power Up*

Sempre que este bónus aparece, o jogador é então desafiado a ganhar alguns pontos extra,

adquiríveis ao clicar com o botão do lado esquerdo do rato em qualquer parte da área que aquele ocupe. De salientar que esta verificação é feita através do reconhecimento da cor do pixel em que o cursor do rato se encontra, de modo a aumentar a precisão da sua deteção.

MultiPlayer

Este modo de jogo, é em tudo semelhante ao modo **SinglePlayer**, mas com o a peculiaridade de o utilizador poder jogar contra um adversário, competindo para obter a melhor pontuação.

A tela passa agora a mostrar não só a pontuação do jogador, como a do seu oponente à medida que estas vão evoluindo no decorrer da partida.

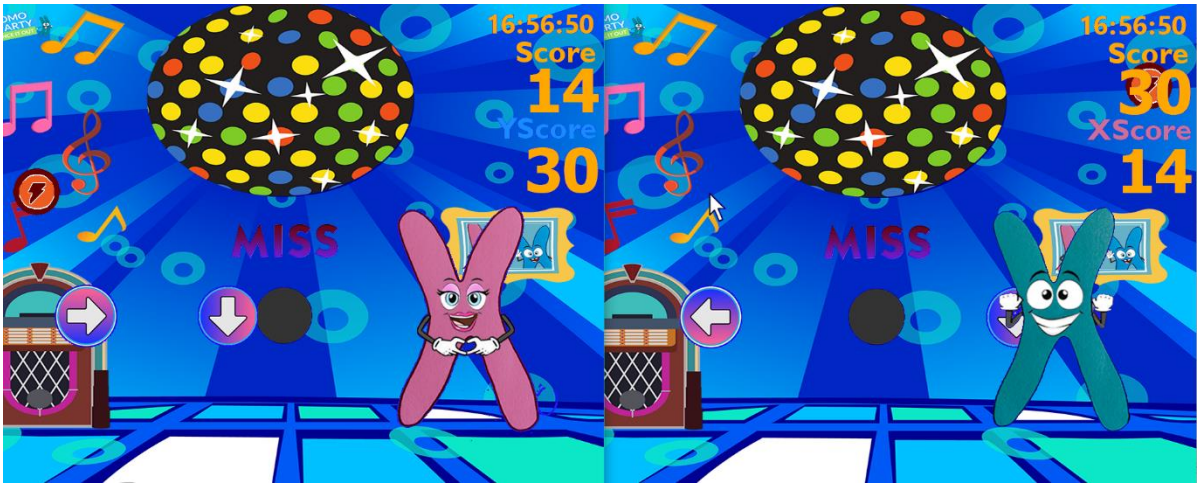


Figura 8 *Modo MultiPlayer em Ação*

High Scores

Tal como foi anteriormente referido, neste menu é dada a possibilidade de consultar as seis melhores pontuações guardadas até ao momento. Associada a cada entrada (numerada desde o 1º lugar ao 6º), temos o *nickname* que o jogador que obteve essa pontuação escolheu, a pontuação total que esse mesmo jogador atingiu e a data em que esse resultado foi alcançado.

Após terminar a consulta, o utilizador apenas tem de clicar na tecla *Esc* para ser levado de volta para o **Menu Inicial**.

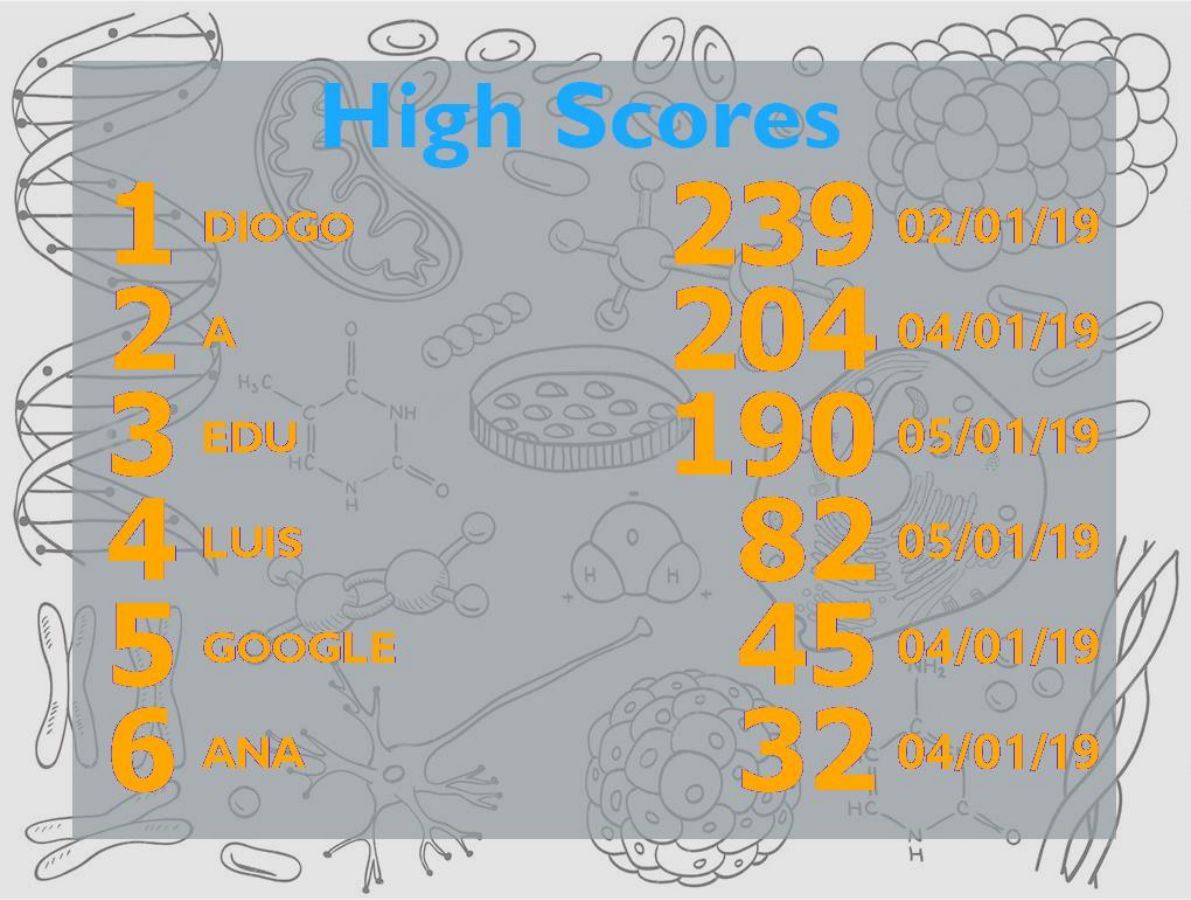


Figura 9 *Menu de Pontuações*

Instructions

Aqui, como se pode verificar na ***Figura 10***, são explicados os conceitos básicos do jogo para que ninguém fique com dúvidas antes de iniciar uma partida.

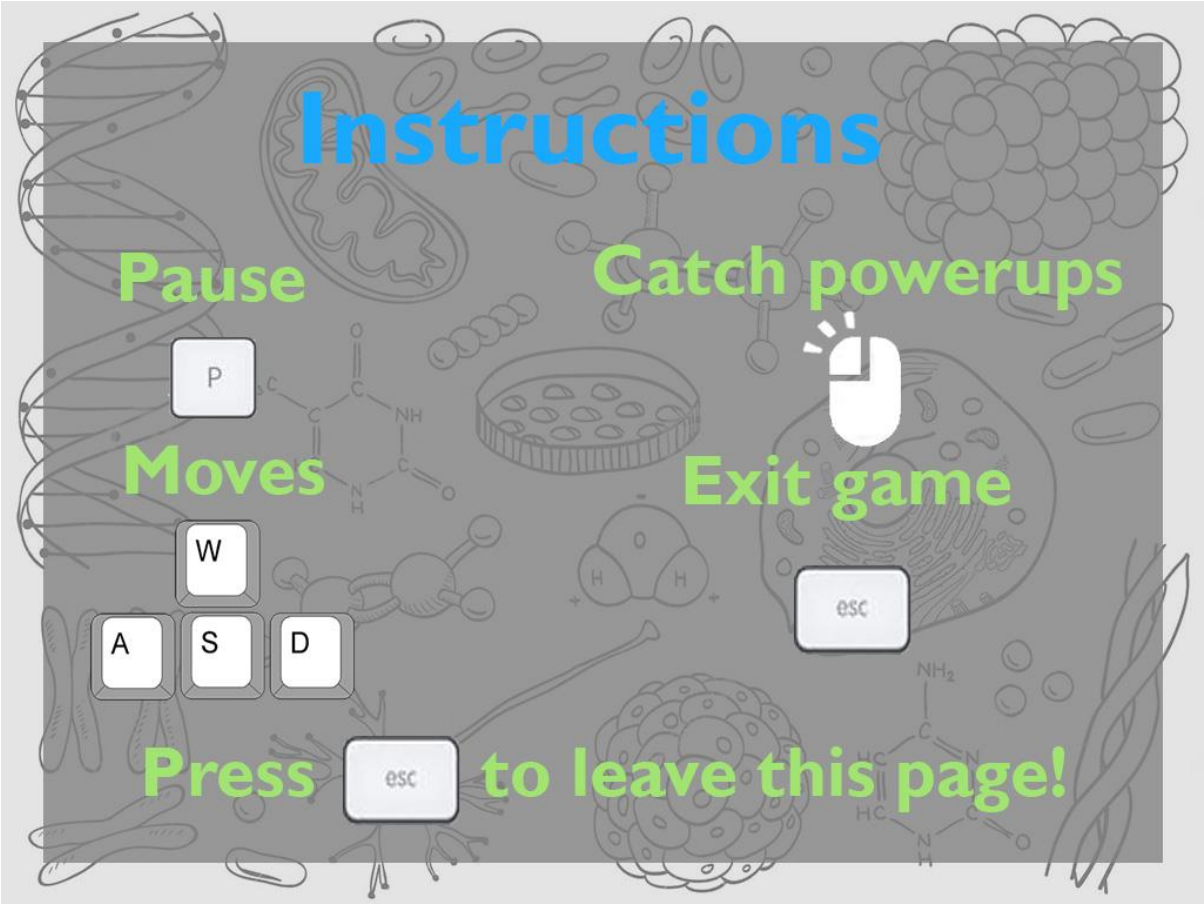


Figura 10 *Instruções*

Game Over

Sempre que o utilizador estiver a jogar, pode terminar o jogo a qualquer altura clicando na tecla *Esc*. Se pretender jogar até terminar o tempo de jogo estabelecido – **45** segundos – este é automaticamente concluído ao fim desse período. Em ambas as situações, é apresentado um painel de final de jogo a felicitar o jogador pelo seu desempenho e onde é apresentada a pontuação que atingiu. Toda esta informação está presente no ecrã durante 3 segundos.

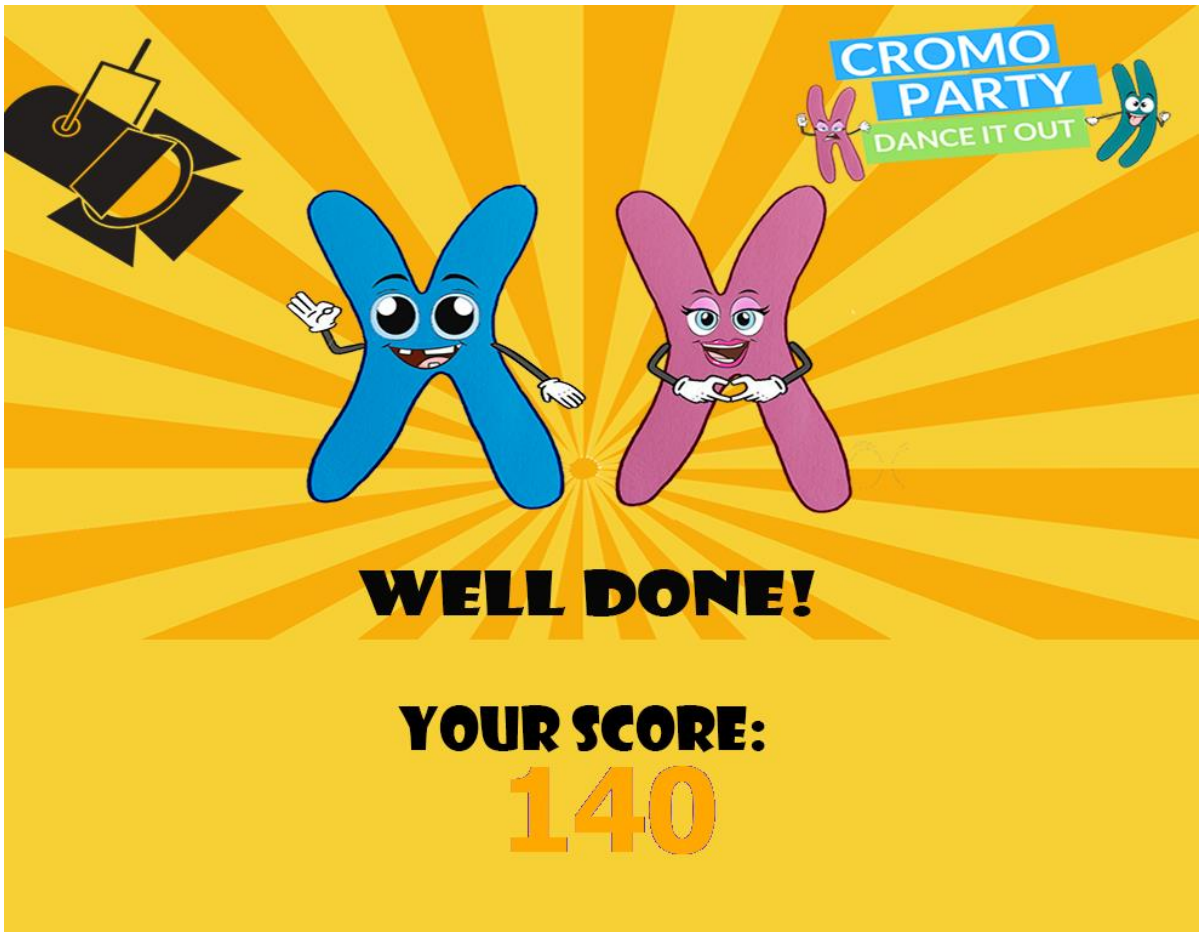


Figura 11 *Game Over*

Pause

A qualquer instante, durante uma partida, o jogo por ser posto em pausa clicando na tecla *P*. Quando o jogador estiver pronto a retomá-la, apenas é necessário clicar nessa tecla novamente.

Figura 12 *Pausa*

Estado do Projeto

Dispositivos Utilizados

Dispositivo	*Utilização*	*Interrupções*
Timer	Controlar *frames, animações e* duração do jogo.	Sim
Keyboard	Escrever o *username*, navegar nos menus e ações específicas durante o jogo.	Sim
Mouse	Ações específicas durante o jogo.	Sim
Video Card	Apresentação dos menus, cenários e objetos.	Não
Real Time Clock	Mostrar a hora, guardar pontuações com a data.	Não
Serial Port	Modo MultiPlayer.	Sim

Figura 13 *Tabela de Utilização de Dispositivos*

Timer

Usado para a medição de intervalos de tempo e para atualizar periodicamente o estado das várias componentes do jogo, nomeadamente a hora, pontuação e posição dos objetos em movimento (setas, *power up*, e cromossoma). O timer, embora seja agora mais simples de implementar, revelou-se essencial para este projeto dado que desempenha um papel determinante na situação do jogo ao longo da partida.

As funções relativas a este periférico encontram-se no timer.c – *void (timer_int_handler)()* – e são usadas no cromoparty.c – *int game(…)*.

Keyboard

O teclado, teve diversos usos no decorrer do trabalho. Inicialmente é usado para escrever o nome do utilizador. Depois, no Menu Principal, para navegar pelas suas opções. No jogo em si, para saber se a tecla clicada corresponde à que aparece nos círculos e saber se o jogador pretende colocar o jogo em pause (tecla *P*) ou sair do jogo (tecla *ESC*).

O teclado é sem dúvida, o periférico que demonstra mais versatilidade neste projeto, visto que é utilizado em todas as fases do programa.

As suas principais funções estão no kbc_asm_ih.S e no keyboard.c. Estas são chamadas no cromoparty.c – *int game(…), void keyboardArrows()* – e menu.c – *int menu()* .

Mouse

É usado no jogo para apanhar o *power up* (conceito do jogo anteriormente explicado) e que essencialmente permite ao utilizador conseguir mais pontos. A sua deteção é feita com colisão a nível de pixel. Uma vez que a posição em que o rato se encontra é conhecida, se o utilizador clicar no botão esquerdo, passa-se à análise desse pixel para perceber se este contém alguma das cores do *power up*, que, por sua vez, são também conhecidas à priori.

Este periférico encontra-se no módulo PS2Mouse.c – *void (mouse_ih)(...)*, *void packet_create()*, *void currentMousePosition()* – e as funções são executadas no cromoparty.c – *int game()*.

Video Card

Usado para criar a interface gráfica do projeto. Habilita o *graphic mode* e permite que imagens sejam desenhadas, o que possibilita que em conjunto com o Timer surjam animações. Temos ainda tanto imagens de letras como de números que viabilizam a impressão de informações essenciais no ecrã durante o jogo à medida que as estas são modificadas, como, por exemplo, frases e pontuações, algo que não seria doutra forma exequível.

O modo escolhido foi o 0x144 cuja resolução é 1024 por 768 pixéis com uma palete de cores de 32 bit.

À medida que é recebido *input* do teclado ocorre a animação do cromossoma, e é com as interrupções do timer que se dá o movimento das setas e do *power up*. No movimento deste último foi implementada colisão através da cor dos pixéis, que ocorre com o cromossoma e as setas.
Foi também implementada a técnica de *Double Buffering*, com vista a tornar a atualização dos *frames* mais consistente e suave.

Conseguimos encontrar as suas funções mais relevantes divididas entre o cromoparty.c – *int pix_map_move(...)*, *void printDance()*, *void powerUps(...)* – a score.c – void *printScore()*, *void print_sentence(...)*, *void print_high_scores()* – e a interface.c – *void double_buffer_to_video_mem()*, *void loadImages()*, *void deleteImages()*, *Bitmap* loadBitmap(...)*, *void drawBitmap(...)*.

Real Time Clock

Usado para ler e mostrar a hora atual durante o jogo e gravar a data de cada partida na tabela de pontuações.

Todas as funções referentes ao periférico estão no módulo RTC.c e todas lá presentes são fulcrais para o seu correto funcionamento.

Serial Port

Tornar o jogo *multiplayer*, no sentido em que pode ser jogado por 2 pessoas em computadores diferentes ao mesmo tempo, sendo possível consultar a pontuação atual do oponente ao longo do jogo. Optamos por recorrer ao uso de interrupções na implementação do perférico.

O módulo que está encarregado de lidar com o mesmo é o serialPort.c sendo todas as suas funções relevantes.

Estrutura e Organização do Código

Proj

Módulo fornecido pelos docentes. Inicializa o modo gráfico pretendido através da função do lab5, *void *(vg_init)(uint16_t mode)*. De seguida chama a função *int menu()* que nos dirige para o módulo seguinte que tem por base a interação com os menus pré-jogo.

Por fim, trata de repor o *text mode* com a mesma função das aulas, *void vg_exit()*.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Menu

Neste módulo são feitas as subscrições e respetivos cancelamentos de subscrição (*unsubscribe*) de interrupções de três periféricos: timer, keyboard e mouse. É neste módulo que praticamente tudo é inicializado para que esteja pronto a ser utilizado. Faz-se carregamento das imagens – *void loadImages()*, dos *high scores* guardados – *void load_score_from_file()* e, no final, o respetivo *delete* – *void deleteImages()* e respetiva gravação dos resultados – *void save_score_to_file()*. É também aqui que se faz a análise das teclas em que o utilizador está a clicar e que irão ser usadas para escrever o seu *nickname* no Menu Inicial (*void convert_key() e void print_sentence(char string[], int x, int y)*).

Como dito anteriormente, a navegação no Menu Principal aqui implementado é possível através das teclas **W** e **S**, de modo a percorrê-lo para cima e para baixo, respetivamente. Já a navegação no Menu das Personagens, mais uma vez aqui implementado, faz-se com recurso às teclas **A** e **D**, para o percorrer para a esquerda e para a direita, respetivamente. Em ambos é possível executar corretamente a opção pretendida através da *State Machine* implementada maioritariamente pela função *void change_menu_state(...),* que atualiza o estado em que o programa se encontra e o seu comportamento. Nesta, as funções pretendidas são chamadas de acordo com o estado atual do programa.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

State Machine

De forma a simplificar a estruturação do jogo, este foi dividido em fases através dum conjunto de estados em que o este se pode encontrar.

Na origem desta máquina de estados estão três conjuntos de que se interligam:

- typedef enum {START, MENU, CHARACTER, GAME} st_t;
- typedef enum {SINGLE, MULTI, HIGHSCORES, INSTRUCTIONS, EXIT} state_t;
- typedef enum {C_Y, C_X} state2_t;

Estando no **Menu Inicial** pode aceder-se ao **Menu Principal**, daí até ao **SinglePlayer**, **MultiPlayer**, **High Scores**, **Instructions** e **Exit**. De qualquer um destes estados, podemos voltar para o **Menu Inicial**. Tanto do **SinglePlayer** como do **MultiPlayer** passa-se ao **Menu de Personagem**, e daqui novamente para o **Menu Principal** ou então para uma das personagens – **X** ou **Y**. Destas podemos aceder ao jogo. Do jogo é permitido voltar ao **Menu Principal**.

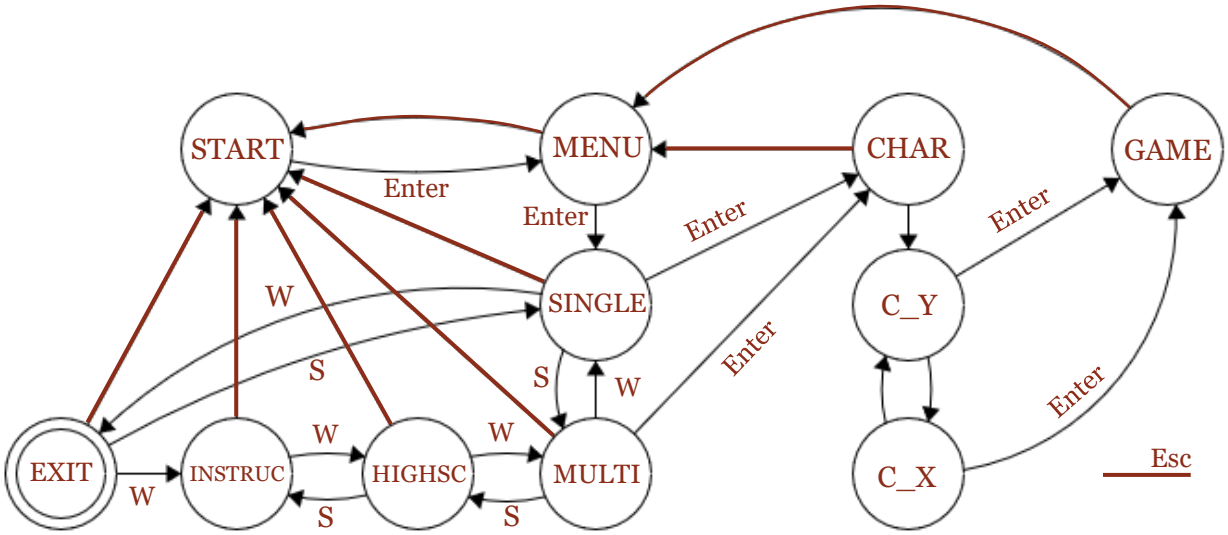


Figura 14 *Diagrama da Máquina de Estados*

Cromoparty

Este módulo é o mais importante porque é aqui que se encontra a base do nosso jogo, sendo então complementado pelos outros módulos. *int game(...)* é a função principal, onde é logo feita a inicialização das setas que irão percorrer o ecrã pela função *void init_arrows()*. Após vários testes, determinamos que idealmente apenas **duas** setas deveriam estar ao mesmo tempo visíveis (de forma a garantir uma boa jogabilidade), sendo que, da maneira como o código foi estruturado, facilmente se consegue variar este número.

Para mais eficazmente gerir o comportamento de uma seta, criamos uma *struct Arrow*. Assim, aspetos como a **posição** em que se encontra, a **velocidade** com que percorre o ecrã, a **direção** para que aponta e o seu **estado** (se está ativa no ecrã, ou não), tornam-se únicos para cada uma.

De tomar nota que tanto a direção que cada seta toma, como a velocidade com que percorre o jogo, são, para cada, gerados aleatoriamente pela função *int arrowRate(int i)*.

```
typedef struct {
int currentX;
int speed;
int direction;
bool active;
} Arrow;
```

Aqui são atendidas as notificações de três periféricos (*timer*, *keyboard* e *mouse*), sendo que a cada *tick* do relógio, para além do handler, é chamado *void arrowProcessing(int x)*, que entre outras coisas de menor relevo, chama a função *int pix_map_move_pos(int x)* responsável for refrescar todos os conteúdos que estão presentes no ecrã. Nesta atualização são verificados aspetos como a prestação do jogador e, a partir daí, modificados fatores a imprimir como, por exemplo, a pontuação (*void show_score(int score, int x, int y)*) e a dança do cromossoma (*void printDance()*). A cada interrupção do teclado, se a tecla pressionada for a *P*, tudo fica em pausa até que esta seja clicada novamente. Caso seja a tecla *Esc*, o jogo é terminado da mesma forma que teria sido automaticamente ao fim de 45 segundos: o *output buffer* é limpo e *end_game()* garante basicamente que tudo esteja pronto para um novo jogo ao restaurar valores de variáveis usadas, guardar os resultados no ficheiro **highscores.txt**, entre outros. Se antes de isto acontecer houver alguma notificação do rato, a sua posicação é atualizada com *currentMousePosition()* .

Trata-se também aqui do movimento do *power up*, mais especificamente na função *void powerUps(…)*. Quando este colide com o cromossoma ou alguma das setas desaparece e volta a aparecer na posição nesse mesmo local, mas com a abcissa x = 0. Caso chegue ao final do ecrã o seu movimento é invertido. Em ambos os casos a sua altura final é gerada aleatoriamente na função *void changeDirect()*.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Interface

A função *void *(vg_init)(uint16_t mode)* referida inicialmente no módulo **Proj** é aqui implementada, utilizando *int get_mode_info(uint16_t mode, vbe_mode_info_t * vmi_p),* também proveniente do lab5 e que fundamentalmente retorna as informações sobre o modo. Para evitar *flickering* nas animações usadas foi usado o método de *double buffering* na escrita dos conteúdos para a memória gráfica, em que estes são primeiramente colocados num outro buffer e só quando estiverem totalmente carregados é que são copiados para a memória pela função *void*

double_buffer_to_video_mem().

Para um fácil e eficiente acesso às imagens utilizadas, utilizamos a *struct Images*. Assim, estas são carregadas – usando *void loadImages()* – e apagadas (libertando a memória que ocupavam) – usando void deleteImages() – apenas uma vez no início e no final do módulo **Menu** (como dito).

As funções[1] que utilizamos tanto para **carregar**, **desenhar** e **apagar** o formato de imagem escolhido (*Bitmap*) foram retiradas da página[2] do ex-aluno do MIEIC Henrique Ferrolho, tendo a primeira sido ligeiramente modificada para melhor servir o propósito do nosso jogo. A segunda acabou por ser bastante reestruturada e alterada para tanto suportar o modo de vídeo usado como a utilização de um segundo *buffer*. Todas as imagens têm de fundo a cor #1F 0F F8, tendo posteriormente sido elaborado um algoritmo na função que as desenha de modo a ignorar esta em específico e, desta forma, surgir transparência.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Score

Este é um dos módulos mais importantes no sentido em que é responsável por uma grande parte da "jogabilidade". Aqui, com a função *void score(int distance, int *cdance)* é dada uma pontuação à prestação do jogador tendo em conta se a seta clicada é a pedida e a sua sincronização com o *pad* central. Essa pontuação conseguida é então mostrada ao jogador de duas formas:

- *void printScore()*, imprime no ecrã um de cinco possíveis *prompts* – "miss", "okay", "great", "perfect", "extra points".
- *void show_score(int score, int x, int y)*, mostra no local indicado, neste caso, canto superior direito, a pontuação numérica atual.

É também neste módulo que as pontuações recorde tomam vida. *void print_high_scores()* permite a visualização dos melhores resultados, previamente carregados do ficheiro **highscores.txt** pela função *void load_score_from_file()*, caso o utilizador aceda à opção ***High Scores***, como explicado anteriormente. A cada jogador é associada uma *struct* que permite mais facilmente gerir alguns atributos como o *name* (que já agora é atribuído usando *void set_current_player_name(char name[25])* que contém o nome indicado pelo jogador inicialmente, *rank* (posição atual), *score* (pontuação) e a data atual. Caso haja espaço ou a pontuação deste jogador seja superior ao dos resultados guardados, *void save_score()* substitui o

[1] As funções em causa são, por ordem: *Bitmap* loadBitmap(const char* filename), void drawBitmap(Bitmap* bmp, int x, int y, Alignment alignment)* e *void deleteBitmap(Bitmap* bmp)*.

[2] https://difusal.blogspot.com/2014/09/minixtutorial-8-loading-bmp-images.html

último classificado no *array* ***players*** pelo novo jogador e reordena a posição atual de cada jogador guardado (*int rank()*). Por fim, e como visto antes, quando o jogo é fechado, a tabela é novamente gravada no ficheiro supramencionado pela função *void save_score_to_file()*, de forma a garantir que estes estão sempre atualizados após cada sessão de jogo.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Timer

Concebido durante o lab2 e importado para o projeto com pequenas modificações visto conter funções desnecessárias e ter sido o primeiro trabalho da disciplina. Faz-se a subscrição (*int (timer_subscribe_int)(uint8_t *bit_no)*) e respetivo cancelamento de subscrição (*int (timer_unsubscribe_int)()*) das interrupções deste periférico. A cada *tick* do relógio é gerada uma interrupção e o *handler* (*void (timer_int_handler)()*) limita-se a incrementar um contador. Estando o sistema na frequência padrão de 60Hz, a cada segundo são geradas 60 interrupções. Este contador pode depois ser acedido por qualquer outro módulo, abrindo imensas possibilidades para a sua utilização na contagem de tempo.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Keyboard

Este módulo foi desenvolvido para o lab3. Uma vez que nem todas as funcionalidades foram necessárias, procedemos à sua simplificação e restruturação. Aqui, e tal como no **Timer**, é feita a subscrição (*int kbd_subscribe_int(uint8_t *bit_no)*) e respetivo cancelamento de subscrição (*int kbd_unsubscribe_int()*) das interrupções do keyboard.

A sua função é atender às notificações do teclado que se dão a cada interrupção (visto ter sido este o modo escolhido), que neste caso são geridas pelo *handler* em *Assembly void kbc_asm_ih(),* explicado no tópico seguinte. Este lê então os respetivos Scan e Break Codes gerados, para que possam ser interpretados e assim usados por outros módulos.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

kbc_asm_ih

Tal como o **Keyboard**, este módulo foi desenvolvido inicialmente durante o lab3, tendo sido revisto e ligeiramente otimizado para o projeto. Como é possível retirar do nome, foi aqui implementado o *interrupt handler* do *keyboard controller* (KBC) em linguagem *Assembly* (*Intel*

Syntax). Sendo esta uma operação intensamente usada durante todo o projeto, é assim possível aumentar um pouco a sua eficiência e ter um maior controlo sobre esta, sem influência do compilador.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

PS2Mouse

Tal como os anteriores, foi importado de um dos labs, neste caso o 4. É preciso ter em atenção que neste caso, para ser utilizado, é necessário primeiro habilitar o rato e mudar para *stream mode* (*int mouse_write_int()*) e, no final, desativá-lo (*int disable_int()*). É então aqui possível fazer a subscrição (*int (mouse_subscribe_int)(uint8_t *bit_no)*) e respetivo cancelamento de subscrição (*int (mouse_unsubscribe_int)()*) das interrupções deste periférico. O rato gera *packets* obtidos pelo seu *handler void (mouse_ih)(...)* que após a sua análise permite, por exemplo, à função *void currentMousePosition()* saber em que coordenadas este se encontra a qualquer momento.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Real Time Clock

Neste módulo são lidos os valores dos registos do Real Time Clock (RTC), neste caso, o dia (*int get_day()*), o mês (int *get_month()*), o ano (*int get_year()*), a hora (*int get_hour()*), os minutos (*int get_min()*) e por último os segundos (*int get_sec()*). Em todas as funções mencionadas anteriormente verificamos se a informação obtida está em BCD (Binary Coded Decimal) e, se isso se verificar, convertemos para binário. Estas funções só são chamadas quando as informações do RTC são válidas (*int read_rtc(uint32_t reg)*). É ainda aqui que através da função *void print_time(int x, int y)* se imprime a hora no ecrã durante o jogo.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Serial Port

É nesta fase do trabalho que se trata da interação entre as duas máquinas virtuais na modalidade MultiPlayer.

Temos aqui 4 ciclos de interrupções distintos, dois tratam da sincronização entre ambos os jogadores (*int playerY_sync(...)* e *int playerX_sync(...)*), ou seja, o jogo só começa quando jogador **X** receber um *char* do jogador **Y** e, vice-versa. Os outros dois tratam do jogo (*int gameMultiX(...)* e *int gameMultiY(...)*) em si com as devidas diferenças do jogo MultiPlayer, sendo

que a “escolha" entre os ciclos está dependente da decisão do jogador entre o cromossoma X ou Y.

O caracter entre as duas máquinas virtuais é passado na função *int write_to_THR(...)* e é lido pelo *handler* do Serial Port (*int serialPort_handler(...)*).

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

i8042

Originário do lab3 e lab4, foi ligeiramente modificado e passou a conter macros pontuais de outros módulos, possibilitando também uma melhor organização e legibilidade do código.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

i8254

Proveniente do lab2, contém *macros* importantes para o bom funcionamento do timer e, tal como o anterior, possibilita uma melhor organização e legibilidade do código.

Desenvolvimento: Diogo Silva (50%) e Mafalda Santos (50%).

Peso Relativo no Projeto

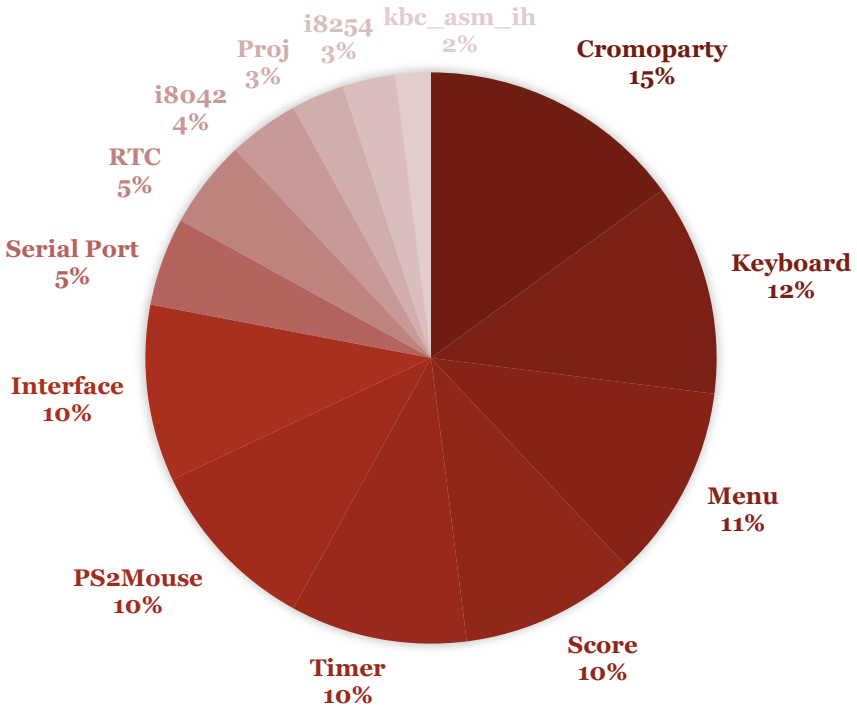


Figura 15 *Peso Relativo dos Módulos no Projeto*

Gráfico de Chamada das Funções

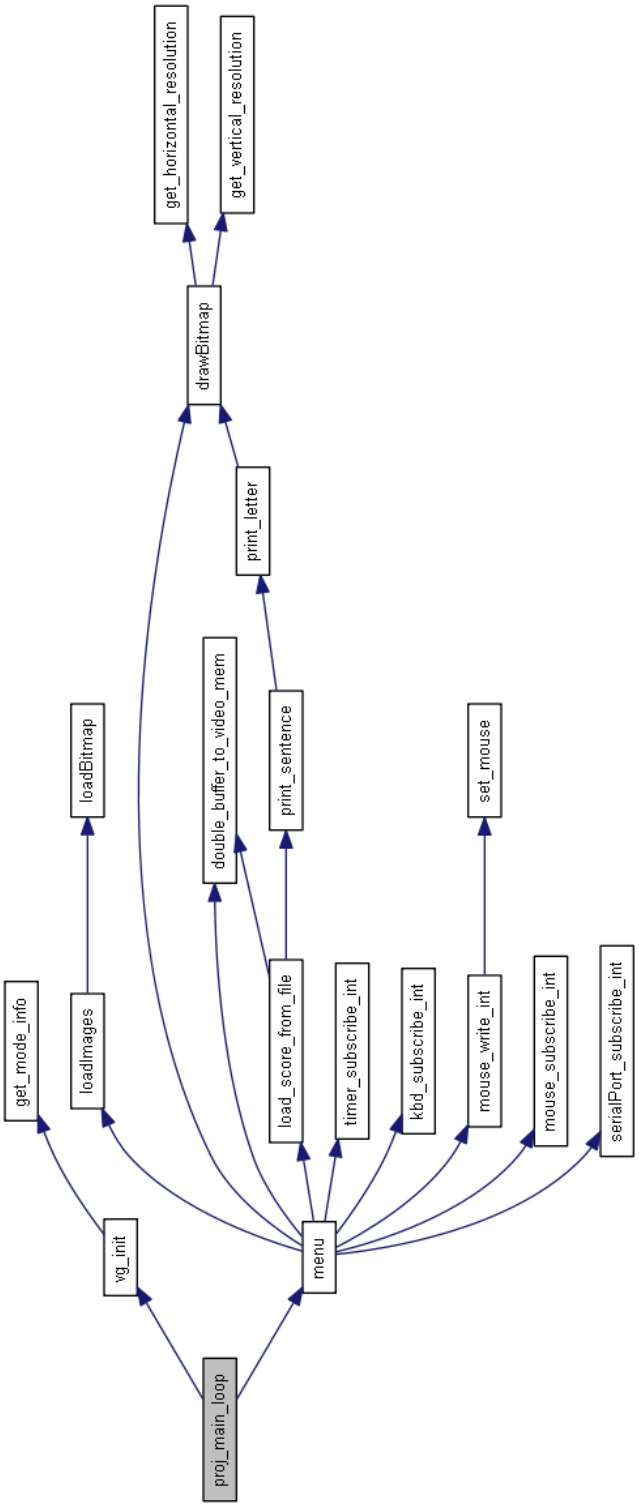


Figura 16 *Call Graph de proj_main_loop*

Detalhes da Implementação

Achamos importante referir a implementação de código em linguagem *Assembly* no desenvolvimento da função que gere as interrupções do teclado (módulo **kbc_asm_ih**) que se tratou de uma oportunidade de usar em nossa vantagem as potencialidades da mistura desta com a linguagem C.

Por outro lado, a elaboração das funções relativas ao Serial Port constituiu um desafio: o facto de não ter sido trabalhado em nenhuma aula prática tornou a sua compreensão/execução bastante mais difícil, nomeadamente nos mecanismos de receção e transmissão de caracteres.

Gostaríamos ainda de mencionar, a construção da máquina de estado já referida através de três conjuntos de estados, o que conduziu a uma estrutura de código mais organizada e compacta, facilitando a introdução de novas funcionalidades ao longo do desenvolvimento do trabalho.

```
typedef enum {START, MENU, CHARACTER, GAME} st_t;
typedef enum {SINGLE, MULTI, HIGHSCORES, INSTRUCTIONS, EXIT} state_t;
typedef enum {C_Y, C_X} state2_t;
```

No que diz respeito ao registo das pontuações, optamos por criar uma *struct Player* que contém todas as informações acerca do jogador que está a jogar naquele preciso momento. No final de cada jogo, a pontuação do jogador atual é comparada com as pontuações dos jogadores na tabela de pontuações (*players*, *array* de *structs Player*). Se esta for superior à menor pontuação, o jogador ultrapassado perde o lugar (é retirado do *array*) e a sua posição é tomada pelo jogador atual. Este processo repete-se até que seja encontrada a sua posição no *ranking*. Desta forma o *array players* pode estar organizado de qualquer maneira, sendo o correto posicionamento de cada jogador na tabela mantido apenas pelo atributo *rank*.

```
typedef struct {
char name[25];
int score;
int rank;
int day;
int month;
int year;
} Player;
```

Por fim, resta apenas apontar o facto de grande parte das imagens utilizadas serem da nossa autoria, inclusive a base das personagens (cromossomas) que foram desenhadas à mão. O restante foi produzido recorrendo ao Adobe Photoshop®.

Conclusão

Com este trabalho pudemos pôr em prática os conhecimentos adquiridos ao longo do semestre de forma muito mais prática e autónoma, o que ajudou bastante a desenvolver as nossas capacidades de programação.

Apesar de a complexidade dos trabalhos ter sido sempre elevada e crescente, ao longo de semestre fomos apanhando o ritmo.

No entanto, ambos consideramos que o que é pedido no lab2 (que apesar de ser o mais simples), com os recursos disponíveis é demasiado complexo para quem está a iniciar a disciplina. Nestes recursos (*handouts*, *slides*, etc) as informações muitas das vezes estão espalhadas e não seguem uma ordem lógica de desenvolvimento.

É ainda de especial importância destacar que não foi dado nenhum resultado dos trabalhos realizados nas aulas práticas ao longo do semestre. Como os labs seguintes (e, inclusivamente, o projeto) muitas vezes implicavam a reutilização de código, é muito provável que ocorra propagação de erros. No entanto, com a adição neste ano letivo da *framework* LCF, obtivemos bastante mais *feedback*, no sentido em que sabíamos se o que estávamos a fazer estava correto ou errado durante o próprio desenvolvimento.

Ao longo do semestre começa também a sentir-se o desfasamento entre as aulas teóricas e as práticas, o que é de certa forma prejudicial. Por outro lado, as aulas práticas são bastante úteis para troca de ideias e resolução de problemas que doutra forma poderiam não ser resolvidos.

O projeto final é bastante desafiante e interessante e as *deadlines* são muito amigáveis. Ainda que, para que o projeto ficasse exatamente como idealizávamos, fosse necessário mais tempo, conseguimos implementar praticamente tudo a que nos propusemos e ainda acrescentamos algumas melhorias.

Ambos dividimos o seu desenvolvimento equitativamente e a cooperação foi a ferramenta mais valiosa para o terminar.

Apêndice

De seguida encontram-se as instruções para corretamente correr o nosso programa a partir da *root* do projeto.

Uma vez que o programa invoca funções que requerem privilégios, caso ainda não tenham sido dadas as permissões necessárias para que este possa correr, deve ser feito:

```
minix$ lcom_conf add conf/proj
```

Sendo a instrução anterior desnecessária, deve-se de seguida aceder ao diretório "src" onde se encontra o código fonte do projeto:

```
minix$ cd src
```

A partir daqui pode ser feita a compilação deste código:

```
minix$ make
```

Por fim, de forma a evitar caminhos absolutos no carregamento de imagens, implementamos o algoritmo *char* appendString(char* s)* na interface.c para que as imagens possam estar em qualquer local do projeto. Assim, apenas é necessário especificar o diretório das imagens dentro do projeto passando-o como argumento pela linha de comandos.

Para tal, para correr basta:

```
minix$ lcom_run proj "diretório da pasta das imagens"
```

Portanto, por exemplo, caso a pasta das imagens *bitmap* se encontre na *root* do projeto:

```
minix$ lcom_run proj "/home/lcom/labs/proj"
```