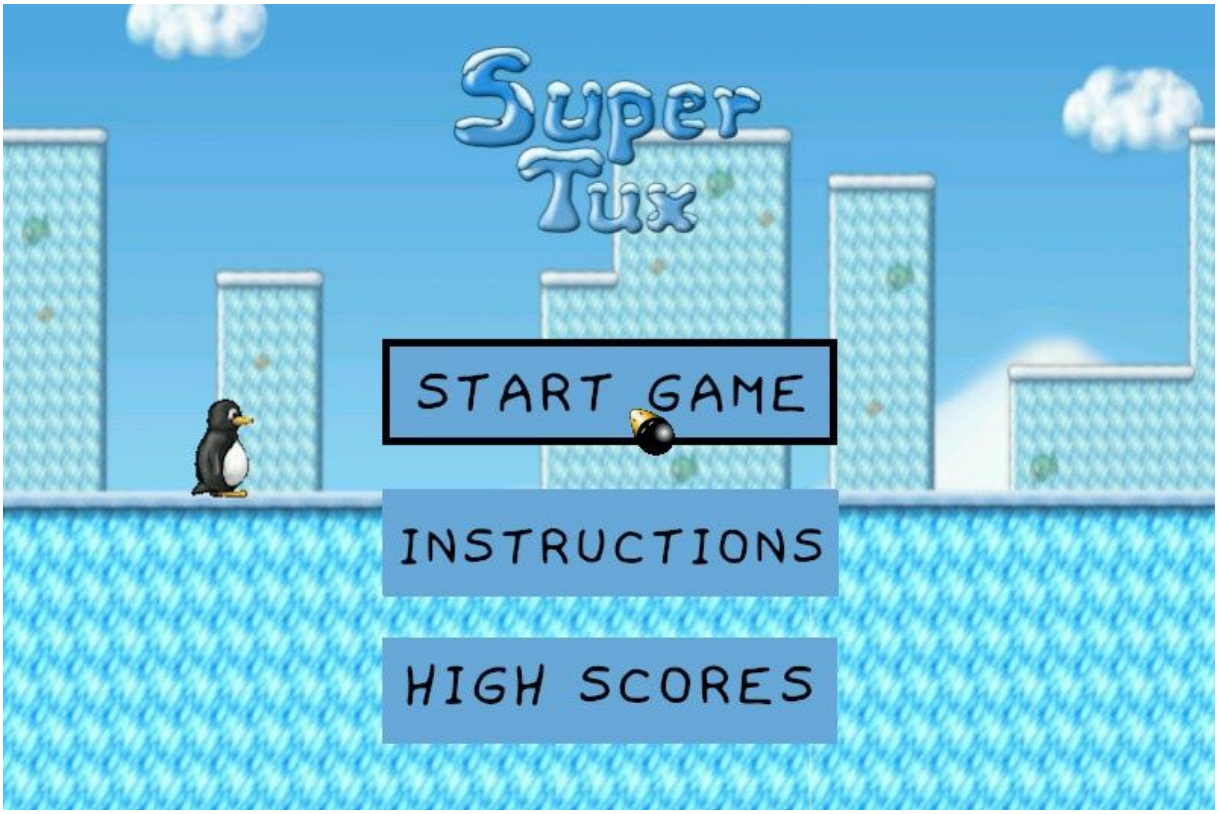
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mini Tux

Uma reinvenção do popular jogo SuperTux



Ano letivo 2019/2020
Laboratório de Computadores

Trabalho realizado pelo grupo T8G03:

Ana Clara Moreira Gadelho up201806309@fe.up.pt

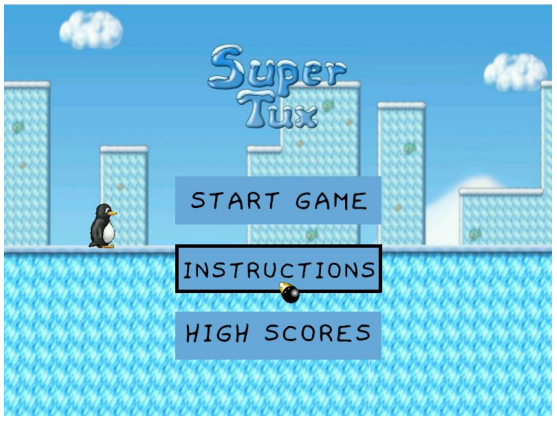
Catarina Justo dos Santos Fernandes up201806610@fe.up.pt

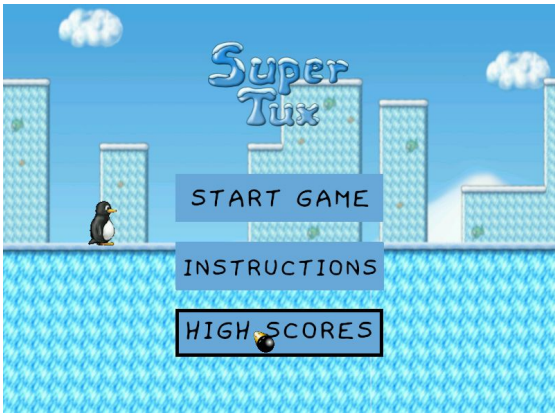
Índice

User Instructions

Menu Inicial

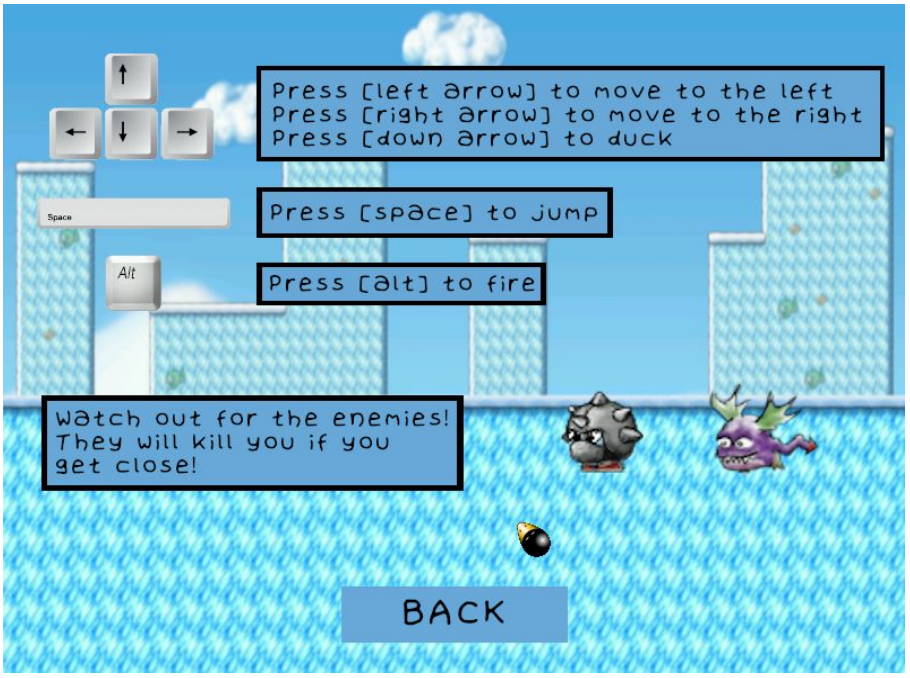
O programa inicia com um menu, onde são apresentadas três opções: Start Game, Instructions e Highscores. Para o utilizador selecionar a opção pretendida deve utilizar o botão esquerdo do rato e manter o cursor por cima do botão pretendido.



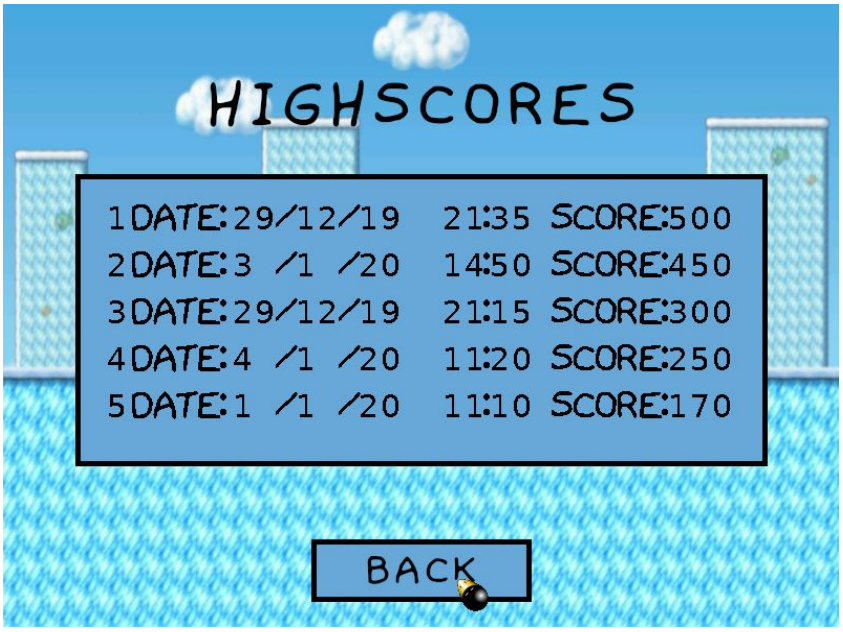


Instructions

Ao carregar com o rato na opção Instructions, o utilizador irá deparar-se com o seguinte menu que mostra as instruções para como jogar o jogo:



Se voltar para o menu inicial (através da opção Back) o utilizador poderá também selecionar a opção Highscores, que mostra o top 5 dos melhores jogadores, juntamente com a data e hora em que o jogador conseguiu a sua pontuação.



Start Game

Quando é selecionada a opção “Start Game”, o jogo é iniciado.
O jogo que criámos consiste numa versão simplificada de Super Tux. O objetivo do jogo é a personagem principal do jogo, o Tux, conseguir recolher o máximo de moedas e matar os inimigos que vão aparecendo, evitando que estes o atinjam. À medida que o tempo passa, começam a aparecer inimigos mais difíceis de derrotar e com maior frequência.

Personagens / Elementos do jogo

Tux - É a personagem principal do jogo, morre se for atingido por um inimigo. Possui fireballs para matar os inimigos.

Spiky - Inimigo mais simples de derrotar, desloca-se sempre para a esquerda e morre quando é atingido por uma fireball.

Zeekling - Surge sempre a voar do lado direito do ecrã, se verificar que o Tux se situa debaixo dele, voa até ao chão para o tentar apanhar. Morre quando é atingido por uma fireball.

Spike - Surge no topo do ecrã e cai até atingir o chão ou o Tux.

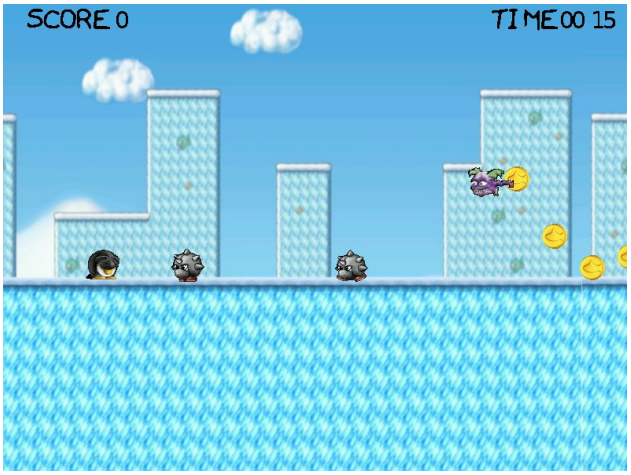
Kamikaze - Surge do lado esquerdo do ecrã a alta velocidade. Morre quando é atingido por uma fireball.

Coin - Surge do lado direito do ecrã, é apanhada pelo Tux quando este lhe toca

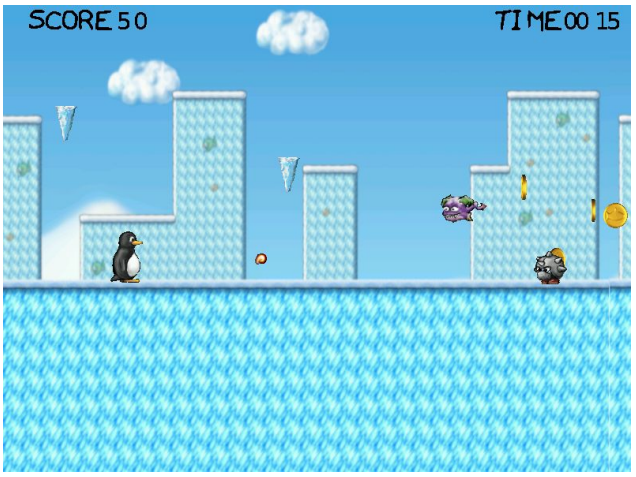
Nota: As imagens utilizadas no projeto são provenientes do jogo original Super Tux, não possuímos direitos de autor sobre elas.

O movimento e ações do Tux são controlados pelo teclado:

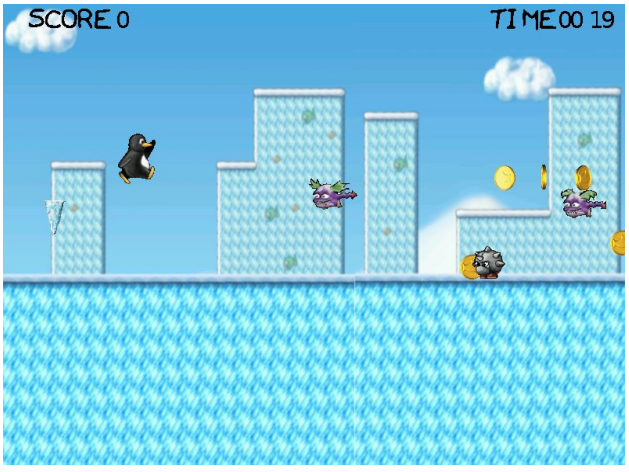
- Left arrow - Tux movimenta-se para a esquerda;
- Right arrow - Tux movimenta-se para a direita;
- Down arrow - Tux baixa-se;
- Spacebar - Tux salta;
- Alt - Tux dispara uma fireball;



Tux a baixar-se

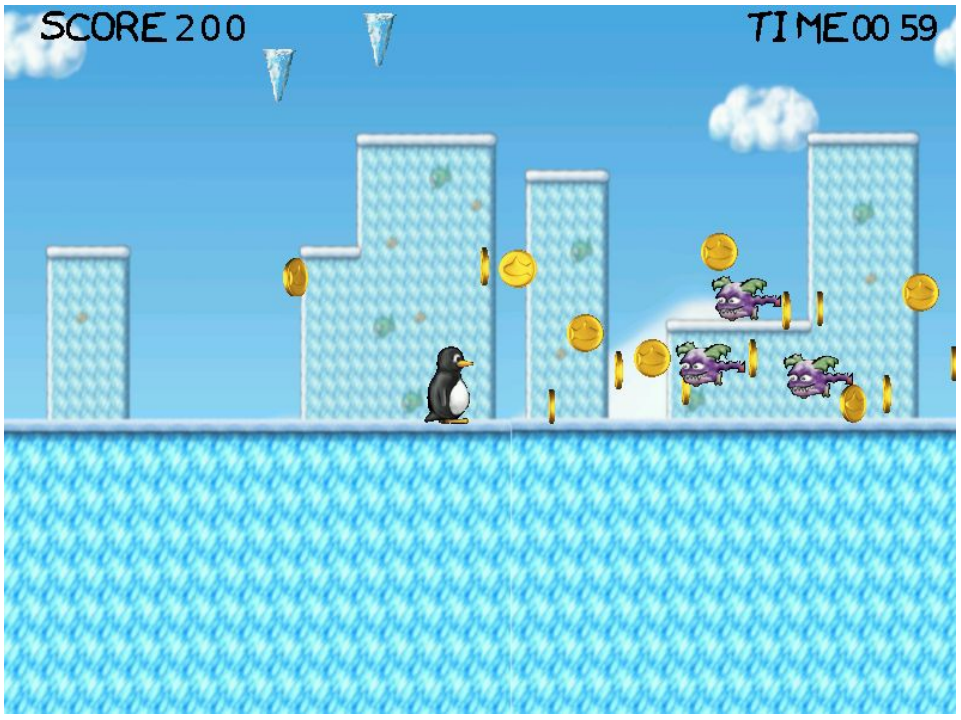


Tux a disparar

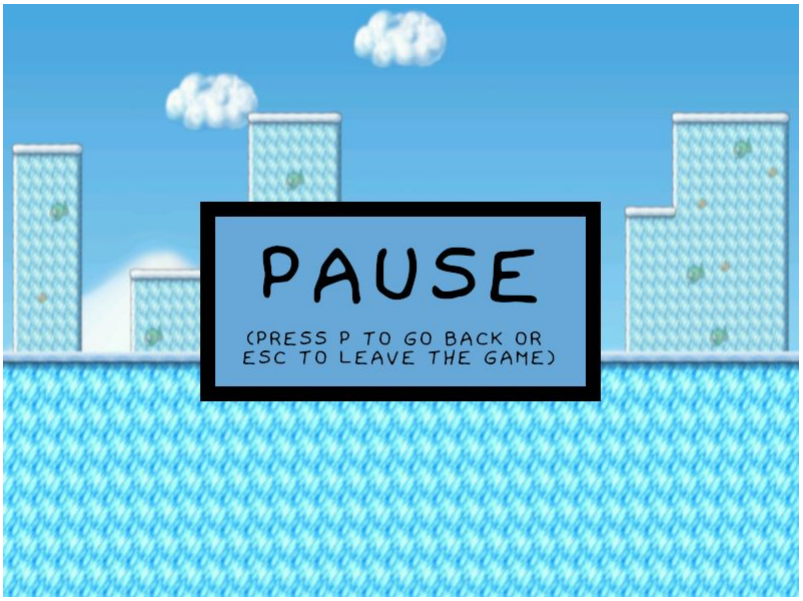


Tux a saltar

Durante o jogo, no canto superior esquerdo do ecrã é apresentado o score do jogador, que é incrementado sempre que o Tux apanha uma moeda ou mata um inimigo, sendo que cada tipo de inimigo vale um determinado número de pontos.
No canto superior direito é apresentado um cronómetro de minutos e segundos, para o utilizador saber há quanto tempo está a jogar.

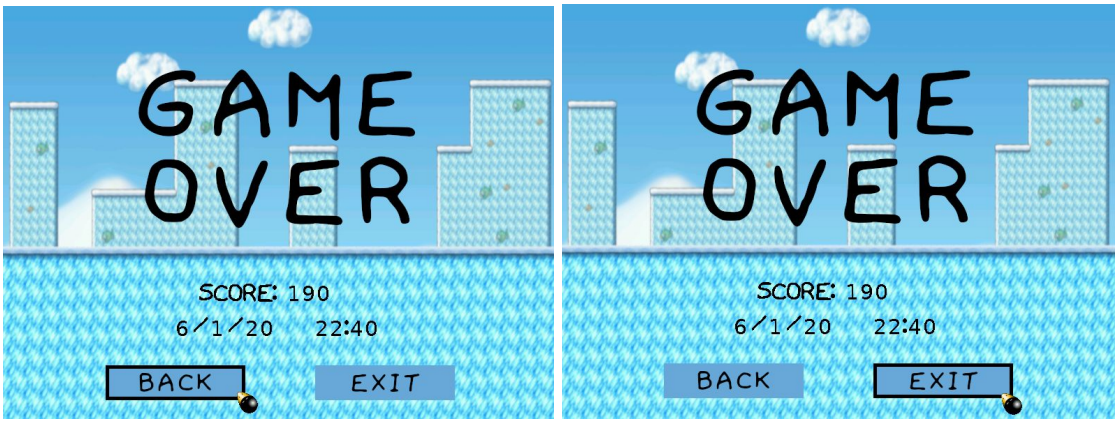


Se durante o jogo o utilizador quiser fazer uma pausa, deve premir a tecla “P” do teclado e aparecerá no ecrã o menu de pausa.



Caso o utilizador queira deixar de estar em modo Pausa deve carregar outra vez na tecla P. Para além disso, caso o utilizador queira voltar ao menu inicial, pode premir a tecla ESC.

Quando o Tux é atingido por um inimigo, o jogo acaba e aparece o menu de Game Over, que mostra a pontuação atingida pelo utilizador no jogo e também a data e a hora daquele momento.
Esse menu tem a opção de Back e Exit. Ao premir Back, o utilizador voltar ao menu inicial, onde pode voltar a jogar o jogo novamente. Contrariamente, ao premir Exit, o utilizador sai do jogo.



Project Status

Dispositivos Usados

Dispositivo I/O	Utilização	Interrupções
Timer	Controlo do Frame rate Atualização do estado do jogo Cronómetro do jogo	Sim
Keyboard	Movimentos e ações do Tux Navegação em alguns menus	Sim
Video Card	Desenho do jogo	Não
Mouse	Escolher as opções nos menus	Sim
RTC	Registo da data nos highscores	Não

Timer

As principais funcionalidades do timer são a atualização da memória gráfica e do estado do jogo, incluindo as posições das personagens, animação de sprites, o movimento do fundo, entre outros.Trata-se por isso do dispositivo base para a implementação do jogo, pois todo o tipo de informação é manuseado pelas interrupções do timer.Para estas funcionalidades são usadas as funções timer_subscribe_int(), timer_unsubscribe_int() e timer_handler(), do ficheiro timer.c criado no Lab2.

Keyboard

O teclado é maioritariamente utilizado para controlar as ações do Tux durante o jogo, na função scancode_handler(Game *g,Tux*t) do ficheiro jogo_funcs.h. Para esse efeito são subscritas as interrupções deste periférico, sendo utilizadas as funções keyboard_subscribe_int(), keyboard_unsubscribe_int(), e kbc_ih() criadas no ficheiro keyboard.h durante o Lab3.

Mouse

O mouse é utilizado nos menus e utiliza tanto os botões como o
movimento para selecionar as opções.
Funciona por interrupções, logo, sempre que há mudança de posição ou um clique de

botão é gerada uma interrupção que é devidamente tratada pelo seu handler.

Video Card

Tal como o timer, a placa gráfica é um dos dispositivos mais importantes do jogo, visto que é responsável pelo desenho na consola de todo o jogo, permitindo a sua visualização por parte do utilizador.
As imagens que utilizamos no nosso projeto são todas no formato xpm.
No nosso projeto escolhemos usar o modo gráfico **0x114**, que possui uma resolução de **800x600** com 16 bits per pixel (RGB 5-6-5).
Para inicializar o modo gráfico são utilizadas as funções get_mode_info() e set_vbe_mode() do ficheiro grafics.c elaborado no Lab5.
Para tornar os gráficos do jogo mais fluidos, foi implementada a técnica do double buffering, sendo que todos os sprites são desenhados num buffer auxiliar que posteriormente é copiado para memória gráfica utilizando a função refresh_graphics() do ficheiro grafics.c.

```
int refresh_graphics(){
  memcpy(v_mem, aux_buffer , x_res * y_res * (bppixel/8) );
  return 0;
}
```

Como o background do jogo é uma imagem muito grande, para não causar atrasos no desenho da mesma, criamos uma função draw_background() que torna o processo mais eficiente.

```
int draw_background(Background *b){

  xpm_image_t img=b->sprite;
  uint8_t *temp=(img.bytes);
  const int img_pixel = img.width* bytes_per_pixel;
  b->x=(b->x)%img.width;
  temp+=b->x*bytes_per_pixel;
  for(int row=0;row<b->sprite.height;++row){
    memcpy(aux_buffer+row*img_pixel,temp+row*img_pixel,img_pixel);

  }
  return 0;
}
```

RTC

O RTC é utilizado para ler a data e hora do sistema. Quando o utilizador atinge um score no jogo que lhe permita ficar na lista dos Highscores, esta informação é lida do RTC para ficar registada na lista. A função que permite efetuar essa leitura é a get_date() do ficheiro rtc.c, sendo que todas as funções deste ficheiro são necessárias para a complementar.

Code organization / Structure

coins.c

No módulo coins é feita toda a gestão das moedas do jogo. Contém funções para a sua criação, eliminação, desenho no ecrã e atualização do estado.

Desenvolvido por: Clara Gadelho
Peso no projeto: 5%

collisions.c

É no módulo collisions que são tratadas as colisões entre os vários sprites do jogo. Contém as funções que permitem gerar e apagar uma área de colisão e detetar a sobreposição entre duas áreas de colisão.

Desenvolvido por: Clara Gadelho
Peso no projeto: 4%

enemies.c

Neste módulo foram desenvolvidas as funções que gerem os inimigos do jogo. Permite a criação, eliminação, desenho no ecrã e atualização do estado dos mesmos.
Desenvolvido por: Clara Gadelho
Peso no projeto: 8%

game_macros.h

Neste módulo estão definidas as diversas macros que são utilizadas ao longo do projeto.
Desenvolvido por Clara Gadelho e Catarina Fernandes (50%/50%)
Peso no projeto: 1%

game_structs.h

Neste módulo estão definidas as diversas *structs* e *enums* que que foram criadas para organizar e facilitar o armazenamento de informação do jogo e que são utilizadas ao longo do projeto.
Desenvolvido por Clara Gadelho e Catarina Fernandes (50%/50%)
Peso no projeto: 3%

grafics.c

Contém as funções necessárias para a utilização da placa gráfica no projeto, como a inicialização do modo gráfico, saída do modo gráfico, desenho de sprites e implementação do double buffering.
Este módulo foi criado durante a elaboração do Lab5, tendo sido adicionadas algumas novas funções para o projeto, como a draw_sprite() e refresh_grapgics().

Desenvolvido por Clara Gadelho e Catarina Fernandes (60%/40%)
Peso no projeto: 8%

I8042.h

Importado do código do Lab3 e Lab4.
Desenvolvido por Clara Gadelho e Catarina Fernandes (50%/50%)
Peso no projeto: 1%

jogo_funcs.c

Este ficheiro contém as funções mais abrangentes do jogo, como a que gere os estados da máquina de estados do jogo, a que atualiza o jogo em si, entre muitas outras.
Desenvolvido por Clara Gadelho e Catarina Fernandes (35%/65%)
Peso no projeto: 35%

jogo.c

Neste módulo encontra-se a função que contém o *loop* das interrupções e onde é feita a inicialização das structs que contém a informação do jogo.
Desenvolvido por Clara Gadelho e Catarina Fernandes (30%/70%)
Peso no projeto: 15%

I8254.h

Importado do código do Lab2.
Peso no projeto: 1%

keyboard.h

Importado do código do Lab3
Desenvolvido por Clara Gadelho e Catarina Fernandes (50%/50%)
Peso no projeto: 5%

mouse.h

Importado do código do Lab4.
Desenvolvido por Clara Gadelho e Catarina Fernandes (50%/50%)
Peso no projeto: 5%

numbers_letters.c

Este módulo contém funções que permitem escrever números e letras no ecrã a partir de inteiros e *strings* passados como argumentos.
Desenvolvido por Clara Gadelho e Catarina Fernandes (20%/80%)
Peso no projeto: 6%

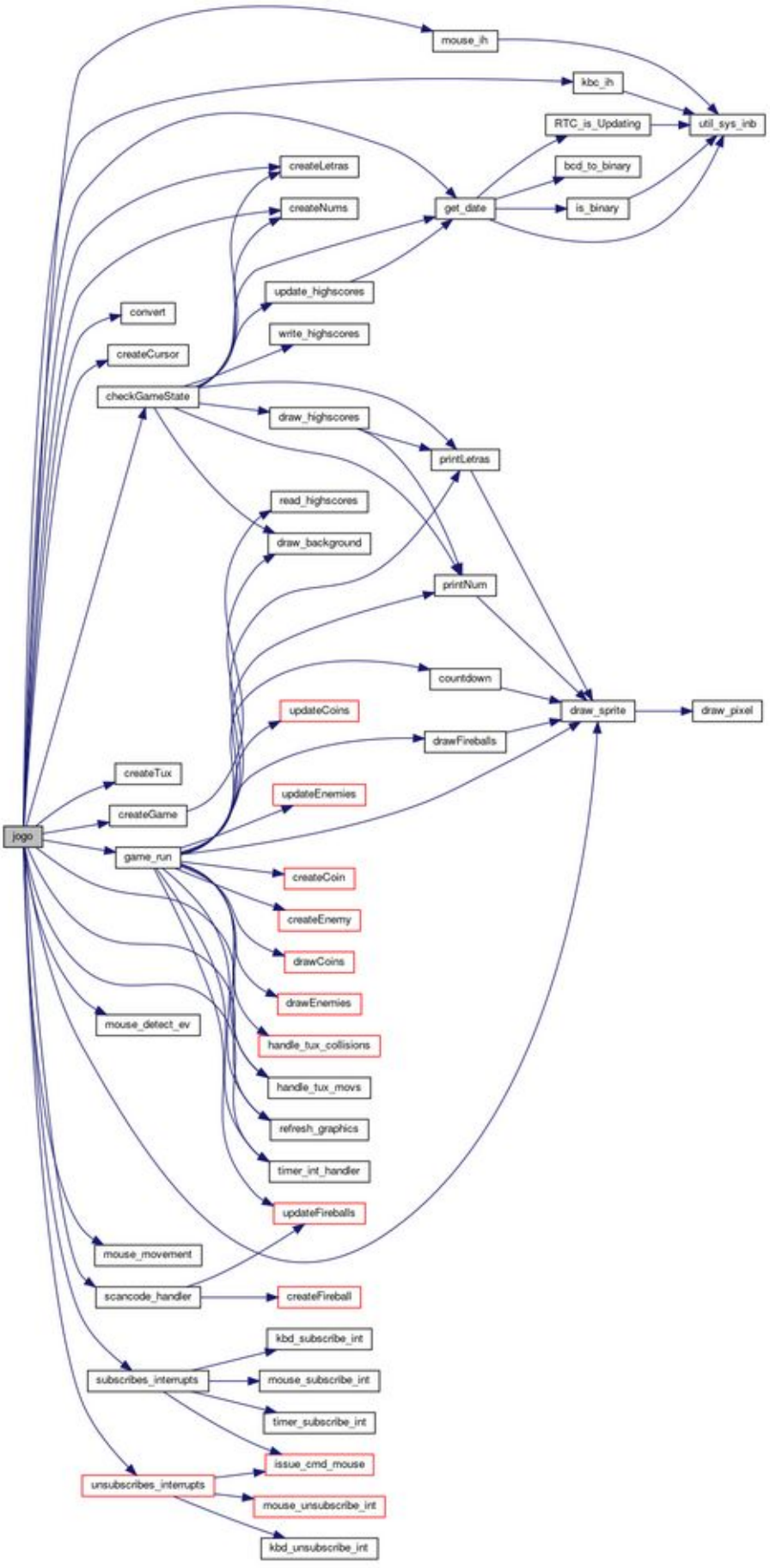
rtc.c

Este módulo permite ler a data atual proveniente do RTC
Desenvolvido por: Clara Gadelho
Peso no projeto: 2%

utils.c

Importado do código do Lab2.
Peso no projeto: 1%

Call
graph


mouse_ih
kbc_ih
util_sys_inb
RTC_is_Updating
bcd_to_binary
is_binary
get_date
createLetras
createNums
update_highscores
convert
write_highscores
createCursor
checkGameState
draw_highscores
printLetras
read_highscores
draw_background
printNum
countdown
draw_sprite
draw_pixel
updateCoins
drawFireballs
createTux
updateEnemies
createGame
jogo
game_run
createCoin
createEnemy
drawCoins
drawEnemies
mouse_detect_ev
handle_tux_collisions
handle_tux_movs
refresh_graphics
timer_int_handler
updateFireballs
mouse_movement
scancode_handler
createFireball
kbd_subscribe_int
subscribes_interrupts
mouse_subscribe_int
timer_subscribe_int
issue_cmd_mouse
unsubscribes_interrupts
mouse_unsubscribe_int
kbd_unsubscribe_int

Detalhes de Implementação

Criação de *arrays* dinâmicos:

Visto que queríamos implementar no nosso jogo vários elementos (inimigos e moedas) que são criados ao longo do tempo e vão sendo destruídos, precisávamos de ter uma estrutura de dados que nos permitisse essa dinâmica. Como na linguagem de programação C, a utilizada nesta cadeira, não existe o conceito de vetores, tivemos que criar nós próprias uma solução para o problema. De forma simples, o que fizemos foi alocar memória para um array com tamanho suficiente para que em nenhum momento do jogo não fosse possível adicionar mais um elemento e criamos uma variável para registar qual o índice do último elemento presente no array em cada momento, sendo que sempre que algum elemento já não fosse necessário, o espaço de memória por ele ocupado passava a conter o último elemento do array e o índice do último elemento era decrementado.

```
deleteCoin(g->coins[i]);
g->coins[i]=g->coins[g->coins_size-1];
g->coins_size--;
```

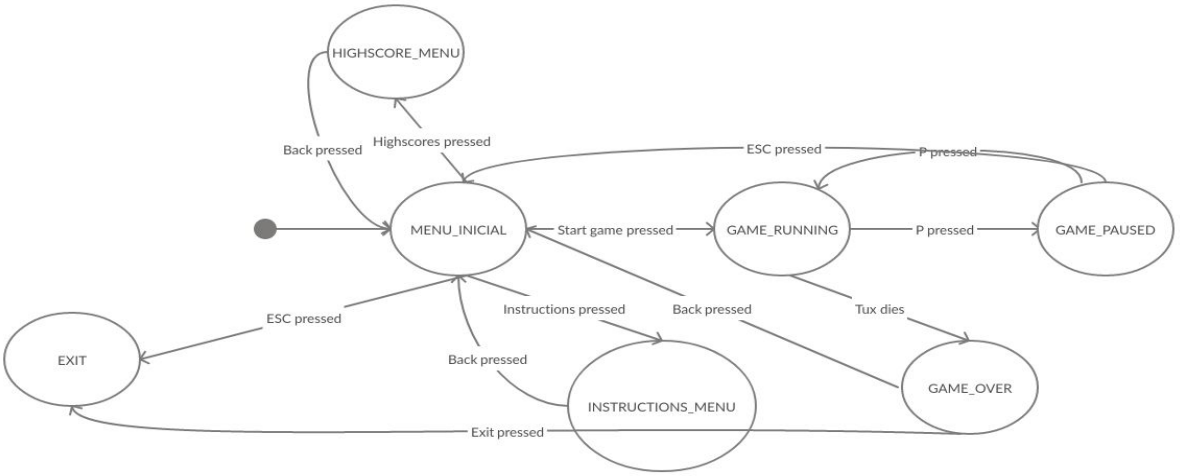
Implementação de máquinas de estados:

- Estados de jogo:

  Como queríamos ter um jogo com vários menus e estados de jogo interligados entre si decidimos que a maneira mais fácil de os implementar era utilizando uma máquina de estados. Os estados são constituídos por:

  - INITIAL_MENU
  - HIGHSCORE_MENU
  - INSTRUCTIONS_MENU
  - GAME_RUNNING
  - GAME_PAUSED
  - GAME_OVER
  - EXIT

  As transições entre os vários estados estão representadas no seguinte diagrama de estados:



- Estados de movimento do Tux:

  São os movimentos que o tux faz e que, para serem movimentos realistas têm as suas limitações (por exemplo, o tux não pode estar baixado virado para a esquerda e logo a seguir saltar virado para a frente, pois precisa de se virar para a frente). Para demonstrarmos estes movimentos usamos as funções handle_tux_movs e scancode_handler do módulo jogo_funcs.

```
int scancode_handler(Game *g,Tux*t){
  if(scancode_bytes[0]==ALT_BRK)t->active_fireball=false;
    switch (t->currentmove)
  {
  case STANDING_R:
    if (scancode_bytes[1]==RIGHT_ARROW){t->currentmove=WALKING_R;t->moving_right=true;}
    if (scancode_bytes[1]==LEFT_ARROW){t->currentmove=WALKING_L;t->moving_left=true;}
    if (scancode_bytes[1]==DOWN_ARROW)t->currentmove=DUCKING_R;
    if (scancode_bytes[0]==SPACEBAR)t->currentmove=JUMPING_R;
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,1);updateFireballs(g,t);}
    break;
  case STANDING_L:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,0);updateFireballs(g,t);}
    if (scancode_bytes[1]==RIGHT_ARROW){t->currentmove=WALKING_R;t->moving_right=true;}
    if (scancode_bytes[1]==LEFT_ARROW){t->currentmove=WALKING_L;t->moving_left=true;}
    if (scancode_bytes[1]==DOWN_ARROW)t->currentmove=DUCKING_L;
    if (scancode_bytes[0]==SPACEBAR)t->currentmove=JUMPING_L;
    break;
  case WALKING_R:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,1);updateFireballs(g,t);}
    if (scancode_bytes[1]==RIGHT_ARROW_BRK){t->currentmove=STANDING_R;t->moving_right=false;}
    if (scancode_bytes[1]==LEFT_ARROW){t->currentmove=STANDING_L;t->moving_right=false;}
    if (scancode_bytes[1]==DOWN_ARROW){t->currentmove=DUCKING_R;t->moving_right=false;}
    if (scancode_bytes[0]==SPACEBAR){t->currentmove=JUMPING_R;t->moving_right=true;}
    break;
  case WALKING_L:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,0);updateFireballs(g,t);}
    if (scancode_bytes[1]==RIGHT_ARROW){t->currentmove=STANDING_R;t->moving_left=false;}
    if (scancode_bytes[1]==LEFT_ARROW_BRK){t->currentmove=STANDING_L;t->moving_left=false;}
    if (scancode_bytes[0]==SPACEBAR){t->currentmove=JUMPING_L;t->moving_left=true;}
    if (scancode_bytes[1]==DOWN_ARROW){t->currentmove=DUCKING_L;t->moving_left=false;}
    break;
  case JUMPING_R:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,1);updateFireballs(g,t);}
    if (scancode_bytes[0]==SPACEBAR_BRK)t->currentmove=AFTERJUMPING_R;
    if (scancode_bytes[1]==RIGHT_ARROW && !t->moving_left){t->moving_right=true;}
    if (scancode_bytes[1]==LEFT_ARROW_BRK){t->moving_left=false;}
    if (scancode_bytes[1]==RIGHT_ARROW_BRK){t->moving_right=false;}
    if (scancode_bytes[1]==LEFT_ARROW && !t->moving_right){t->moving_left=false;}
    break;
  case JUMPING_L:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,0);updateFireballs(g,t);}
    if (scancode_bytes[0]==SPACEBAR_BRK)t->currentmove=AFTERJUMPING_L;
    if (scancode_bytes[1]==RIGHT_ARROW && !t->moving_right){t->moving_right=false;}
    if (scancode_bytes[1]==LEFT_ARROW_BRK){t->moving_left=false;}
    if (scancode_bytes[1]==RIGHT_ARROW_BRK){t->moving_right=false;}
    if (scancode_bytes[1]==LEFT_ARROW && !t->moving_right){t->moving_left=true;}
    break;
  case DUCKING_R:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,1);updateFireballs(g,t);}
    if (scancode_bytes[1]==DOWN_ARROW_BRK)t->currentmove=STANDING_R;
    break;
  case DUCKING_L:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,0);updateFireballs(g,t);}
    if (scancode_bytes[1]==DOWN_ARROW_BRK)t->currentmove=STANDING_L;
    break;
  case AFTERJUMPING_R:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,1);updateFireballs(g,t);}
    if (scancode_bytes[1]==RIGHT_ARROW && !t->moving_left){t->moving_right=true;}
    if (scancode_bytes[1]==LEFT_ARROW_BRK){t->moving_left=false;}
    if (scancode_bytes[1]==RIGHT_ARROW_BRK){t->moving_right=false;}
  case AFTERJUMPING_L:
    if( scancode_bytes[0]==ALT&&!t->active_fireball){t->active_fireball=true ;createFireball(t,0);updateFireballs(g,t);}
    if (scancode_bytes[1]==LEFT_ARROW_BRK){t->moving_left=false;}
    if (scancode_bytes[1]==RIGHT_ARROW_BRK){t->moving_right=false;}
    if (scancode_bytes[1]==LEFT_ARROW && !t->moving_right){t->moving_left=true;}
    break;
  default:
    break;
  }
  return 0;
}
```

```
int handle_tux_movs(Tux *tux){
    switch (tux->currentmove)
    {
    case WALKING_R:
        if(tux->xpos>=800-tux->sprites[tux->currentsprite].width)
            tux->currentmove=STANDING_R;
        else{
          tux->xpos+=tux->speed;
          tux->currentsprite=(tux->currentsprite+1)%4;
        }
        break;
    case WALKING_L:
        if(tux->xpos <= 0)
            tux->currentmove = STANDING_L;
        else{
            tux->xpos-=tux->speed;
            tux->currentsprite=(tux->currentsprite + 1)%4 + 6;
        }
        break;
    case DUCKING_R:
        tux->currentsprite=5;
        break;
    case DUCKING_L:
        tux->currentsprite=11;
        break;
    case STANDING_R:
        tux->ypos=FLOOR-tux->sprites[tux->currentsprite].height;
        tux->currentsprite=0;
        break;
    case STANDING_L:
        tux->ypos=FLOOR-tux->sprites[tux->currentsprite].height;
        tux->currentsprite=6;
        break;
    case JUMPING_R:
        if(tux->ypos<=MAX_TUX_JUMP)tux->currentmove=AFTERJUMPING_R;
        else{
            tux->currentsprite=4;
            tux->ypos-=15;
            if(tux->moving_right && tux->xpos<800-tux->sprites[tux->currentsprite].width)
                tux->xpos+=tux->speed;
        }
        break;
    case JUMPING_L:
        if(tux->ypos<=MAX_TUX_JUMP)tux->currentmove=AFTERJUMPING_L;
        else{
            tux->currentsprite=10;
            tux->ypos-=15;
            if(tux->moving_left && tux->xpos > 0)
                tux->xpos-=tux->speed;
        }
        break;
    case AFTERJUMPING_R:
        if(tux->ypos>=FLOOR-tux->sprites[tux->currentsprite].height){
            if(tux->moving_right)tux->currentmove=WALKING_R;
            else tux->currentmove=STANDING_R;
        }
        else
        {
            tux->currentsprite=4;
            tux->ypos+=15;
            if(tux->moving_right && tux->xpos<800-tux->sprites[tux->currentsprite].width)
                tux->xpos+=tux->speed;
        }
        break;
    case AFTERJUMPING_L:
        if(tux->ypos>=FLOOR-tux->sprites[tux->currentsprite].height){
            if(tux->moving_left)tux->currentmove=WALKING_L;
            else tux->currentmove=STANDING_L;
        }
        else
        {
            tux->currentsprite=10;
            tux->ypos+=15;
            if (tux->moving_left &&tux->xpos>0)
                tux->xpos-=tux->speed;
        }
        break;
    default:
        break;
    }
    return 0;
}
```

Conclusions

De forma geral, ambas concordamos que esta unidade curricular representou um grande desafio e permitiu-nos aprender imenso e desenvolver competências que nos ajudarão ao longo do curso.

Embora a estrutura da cadeira esteja bem desenvolvida, um aspeto que achamos que poderia ter algum impacto no nosso aproveitamento seria a publicação dos resultados dos *Labs* antes de termos que desenvolver o projeto, de forma a termos alguma noção se estamos ou não a fazer um bom trabalho.
Realçamos também que os labs poderiam estar melhor explicados, pois ás vezes sentimos dificuldade em entender o assunto tratado num certo lab.

No entanto achamos que no geral esta unidade curricular foi benéfica para o nosso percurso enquanto estudantes de engenharia informática.