U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

RELATÓRIO FINAL

Projeto de Laboratório de Computadores

Mestrado Integrado em Engenharia Informática e Computação

Relatório Final do Projeto desenvolvido no âmbito da U.C. Laboratório de Computadores no ano letivo 2019/2020

Gonçalo André Carneiro Teixeira 201806562

Tiago André Macedo Pinto 201808907

Índice

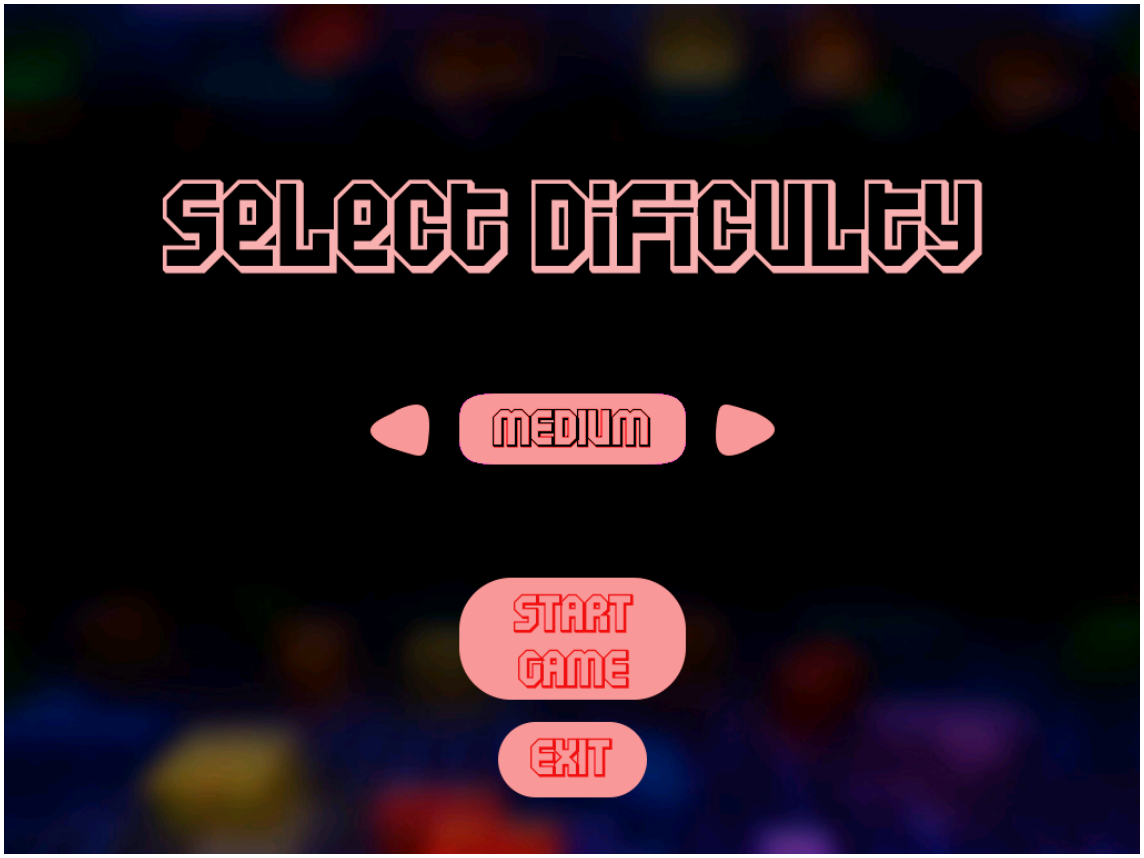
1. Instruções de Utilização

1.1. O Menu de Entrada

Ao iniciar o projeto, o utilizador é saudado com o seguinte ecrã:

Naturalmente, basta pressionar a tecla ENTER para continuar.

1.2. O Menu Principal



Neste segundo menu, o utilizador pode escolher a dificuldade do jogo, desde fácil, médio a difícil. A dificuldade do jogo está diretamente relacionada com a velocidade com que as peças caem do topo do ecrã.

Instruções para este menu:

Teclado:

- Tecla de seta para esquerda ou direita seleciona dificuldade;

- Tecla ENTER inicia o jogo com a dificuldade selecionada;

- Tecla ESC faz o jogo acabar.

Rato:

- Tocar com o botão esquerdo nas setas escolhe a dificuldade, à semelhança do teclado;

- Tocar nos botões “START GAME” e “EXIT” inicia ou sai do jogo, respetivamente.

1.3. O Jogo

O jogo “Tetris” é um jogo de pontuação, sem vitória. As peças, tetraminós, caem do topo do ecrã e o jogador perde quando já não houver espaço para fazer cair uma peça nova. Para que haja sempre espaço o jogador tem como objetivo completar linhas horizontais para que estas se “desintegrem” dando espaço para mais peças poderem encaixar.

A cada linha que é desintegrada, 10 pontos são adicionados.

As peças são as seguintes:

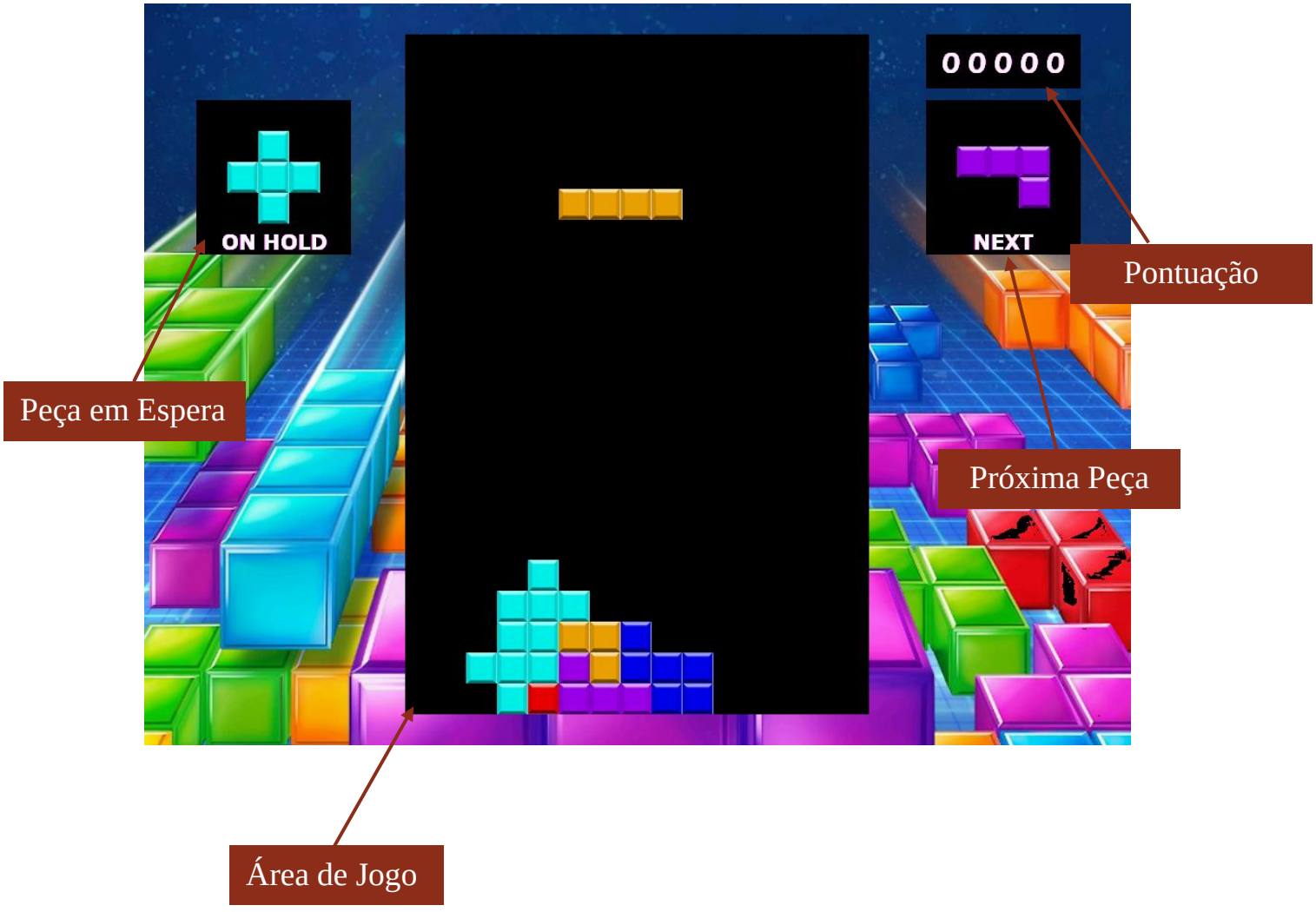
Nota – As cores destas peças não correspondem às cores que usamos no projeto.

Para o utilizador empilhar as peças da melhor forma possível pode:

- Rodar a peça:
    - Sentido horário com a tecla “D”;
    - Sentido anti-horário com a tecla “A”;
- Deslocar a peça horizontalmente:
    - Para a esquerda com a tecla de seta para a esquerda;
    - Para a direita com a tecla de seta para a direita.
- Acelerar o movimento da peça:
    - Às vezes torna-se aborrecido esperar que a peça caia quando temos a certeza que é naquele lugar que queremos que caia, então adicionamos a funcionalidade para a peça cair com o dobro da velocidade com a tecla de seta para baixo.
- Fazer a peça cair instantaneamente:
    - A peça pode cair instantaneamente clicando na tecla seta para cima.

- Colocar uma peça em espera:
    - No decorrer do jogo vão existir situações que a peça que está a cair simplesmente não é o que queremos, então para melhorar essa situação implementamos um sistema de peças em espera: Ao tocar no ESPAÇO o jogador troca a peça que está a cair, por uma peça já esteja em espera (ON HOLD) ou pela peça seguinte, caso não exista ainda nenhuma peça em espera.

1.4. O layout do Jogo



1.5. O Menu de Pausa

O jogador pode colocar o jogo em “Pausa” durante o tempo que achar necessário. Para continuar o jogo atual basta pressionar ENTER, para terminar o jogo atual e regressar ao Menu Principal pressiona ESC.

1.6. O Menu Game Over

Quando não é possível gerar nenhuma peça sem que esta colida com uma peça já existente (i.e. já não há mais espaço na tela) o jogo termina com o ecrã abaixo, apresentando os pontos com os quais terminou o jogo. É pedido ao jogador que pressione a tecla ENTER para regressar ao Menu Principal.

2. Estado do Projeto

2.1. Dispositivos utilizados

Dispositivo	Utilização	Interrupções?
Timer	Atualização do jogo, movimento das peças.	Sim
Teclado	Movimentar as peças, navegar nos menus.	Sim
Rato	Navegar nos menus.	Sim
Placa de Vídeo	Interface do jogo e menus	Não

Timer

O timer é responsável pela atualização do estado do jogo, atualização de gráficos, mas mais precisamente para atualizar a posição da peça que está a cair. Também utilizamos o timer para fazer aparecer a mensagem “Press ENTER to continue…” no menu inicial de segundo em segundo.

O timer é utilizado nos seguintes módulos e respetivas funções:

- tetris.c
    - dropPiece
- menus.c
    - mainTittle

Teclado

O teclado é usado na totalidade do projeto, mas não necessariamente, uma vez que no menu de escolha da dificuldade podemos fazer tudo com o rato.

O teclado é talvez o dispositivo mais importante, uma vez que é com ele que controlamos o movimento das peças, e, portanto, todo o curso do jogo.

Com as teclas “A” e “D” podemos rodar a peça, com as teclas de seta para a esquerda ou direita podemos movimentar a peça na direção horizontal, com a tecla de seta para cima podemos fazer a peça cair instantaneamente, com a tecla de seta para baixo podemos acelerar o movimento de queda da peça e com a tecla ESPAÇO podemos colocar a peça atual em espera.

O teclado é utilizado nos seguintes módulos e respetivas funções:

- menus.h
    - mainTittle
    - diffMenu
- tetris.h
    - dropPiece
    - pauseMenu
    - gameOverMenu

Rato

Tivemos dificuldades a conciliar as interrupções do rato com a queda das peças, uma vez que o objetivo inicial seria rodar a peça e possivelmente deslocar a peça com o rato. No entanto, a implementação do rato foi mais simples do que queríamos, o rato pode ser usado no menu de escolha da dificuldade. Tanto o movimento como os botões funcionam neste e só neste menu.

O teclado é utilizado no módulo menus.c na função diffMenu.

Placa de Vídeo

A placa de vídeo é o dispositivo que "faz o jogo", uma vez que é responsável por apresentar os gráficos no ecrã. Naturalmente, utilizamos a placa de vídeo no modo gráfico, com o modo 0x117, de resolução de 1024x768px, 64K cores, no formato RGB565. As imagens foram feitas no formato bmp utilizando o GIMP e o Adobe Photoshop no formato RGB565.

Para não haver nenhum tipo de *flickering* utilizamos a técnica de *double buffering* para dar fluidez ao nosso jogo: depois de estarem todos os componentes "montados" no buffer auxiliar, a função doubleBuffering copia a memória correspondente ao buffer auxiliar para o buffer principal. Esta técnica faz com que tudo o que é suposto aparecer no mesmo frame apareça ao mesmo tempo, sem delay nem *screen flickering*.

Ao atualizar o frame, com a função updateFrame (tetris.c):

1. Resetar o buffer auxiliar;
2. Desenhar o background no buffer auxiliar;
3. Desenhar os pontos atuais;
4. Desenhar a próxima peça no local conveniente;
5. Desenhar a peça em espera (se existir) no local conveniente;

Depois de chamar esta função, é normalmente chamada a função printBoard (tetris.c) para desenhar no ecrã os blocos, e por fim é chamada a função doubleBuffering (vídeo_gr.c) para efetuar o processo de double buffering.

Quanto a deteção de colisões, optamos por usar um array com as posições dos blocos, que iremos explicar no módulo tetris, mais adiante.

2.2. Organização e Estrutura do Código

Bitmap

Este módulo é responsável por carregar as imagens em formato bmp RGB565, da autoria de Henrique Ferrolho, que publicou o dito código no seu blog - http://difusal.blogspot.com/2014/09/minixtutorial-8-loading-bmp-images.html

Fizemos algumas alterações neste código para conseguirmos aplicar o efeito de transparência nas imagens, para isso, utilizamos o magenta, com o código 0xff00ff nos softwares GIMP e Adobe Photostop, de seguida fizemos um algoritmo para ignorar a cor magenta ao carregar o bitmap.

Gonçalo – XX%

Tiago – XX%

Timer

Módulo importado das aulas práticas (lab2).

Menus

Este módulo contém o código relativo à implementação de menus no projeto.

Existem 4 menus, o Inicial, o Principal, o de Pausa e o Game Over.

Neste módulo existe uma estrutura de dados (struct) que contem pointers para bitmaps com os fundos dos menus, cursor, mas também a dificuldade escolhida e ainda a posição do cursor.

Tem 3 funçoes:

- startMenus()
    - Esta função inicia e devolve a struct do tipo menus_t com os vários campos preenchidos.
- mainTittle()
    - Esta função é responsável por apresentar o menu inicial no ecrã, alternando a mensagem “Press ENTER to continue…” de segundo em segundo, a função termina quando o utilizador pressiona ENTER.
- diffMenu()
    - Esta função é responsável por apresentar o menu principal (escolha da dificuldade) no ecrã. A função lida com interrupções tanto do mouse como do teclado, o jogador pode utilizar ambos os dispositivos. A função atualiza a dificuldade sempre que o utilizador seleciona uma dificuldade diferente. A função termina quando o utilizador escolhe começar o jogo ou então sair.

Gonçalo – XX%

Tiago – XX%

Tetris

Este é o módulo principal do código, neste módulo é inicializada a struct “tetris”, já referida acima.

Na função tetris_start, é criado um loop para as peças caírem enquanto o jogo poder prosseguir. Quando o jogo é iniciado em startGame, é atribuído aleatóriamente um ID entre 1 e 12 para a peça atual e para a próxima. No fim de cair uma peça, o valor da próxima peça é passado para a atual, e a próxima peça é recalculada utilizado o método rand().

A área de jogo é um retângulo com capacidade para 22 blocos na vertical por 15 na horizontal, portanto criamos um array de números para representar as cores dos blocos a desenhar, e outro array de valores booleanos para saber se na posição YX existe um bloco.

As funções mais importantes deste módulo são:

- startGame – inicia a struct tetris;
- printBoard – apresenta no ecrã os blocos, de acordo com as cores no array;
- printPoints – apresenta no ecrã os pontos atuais;
- printNext – apresnta no ecrã a próxima peça a cair;
- printOnHold – apresenta no ecrã a peça que está em espera (se existir);
- dropPiece – função com loop para fazer cair a peça, lida com as interrupções e movimentos da peça.
- tetris_start – função para começar o jogo em si.
- checkColision – função que verifica se existe uma colisão.
- pauseMenu – função que apresenta o menu de pausa
- gameOverMenu – função que apresenta o menu de fim de jogo

Gonçalo – XX%

Tiago – XX%

Keyboard

Este módulo foi importado das aulas práticas (lab3), com modificações mínimas.

O módulo é responsável por subscrição de interrupções e fim de subscrição de interrupções do teclado, assim como a gestão da leitura de *scancodes*.

Mouse

Este módulo foi importado das aulas práticas (lab4).

O módulo é responsável por subscrição de interrupções e fim de subscrição de interrupções do mouse, bem como a gestão e leitura de packets.

Video_gr

Importado das aulas práticas (lab5), com algumas alterações.

O módulo é responsável pela inicialização da placa de vídeo em modo gráfico no modo desejado e por regressar ao modo de texto no fim da execução.

i8042

Importado das aulas práticas, com pequenos acréscimos.

i8254

Importado das aulas práticas (lab2)

Proj

Módulo base do projeto, inicializa os menus, inicializa o jogo em si, subscreve e remove a subscrição dos dispositivos, inicializa a placa de vídeo no modo gráfico e regressa ao modo de texto no fim. Contém um loop para o programa correr até que o utilizador selecione “EXIT” no menu principal.

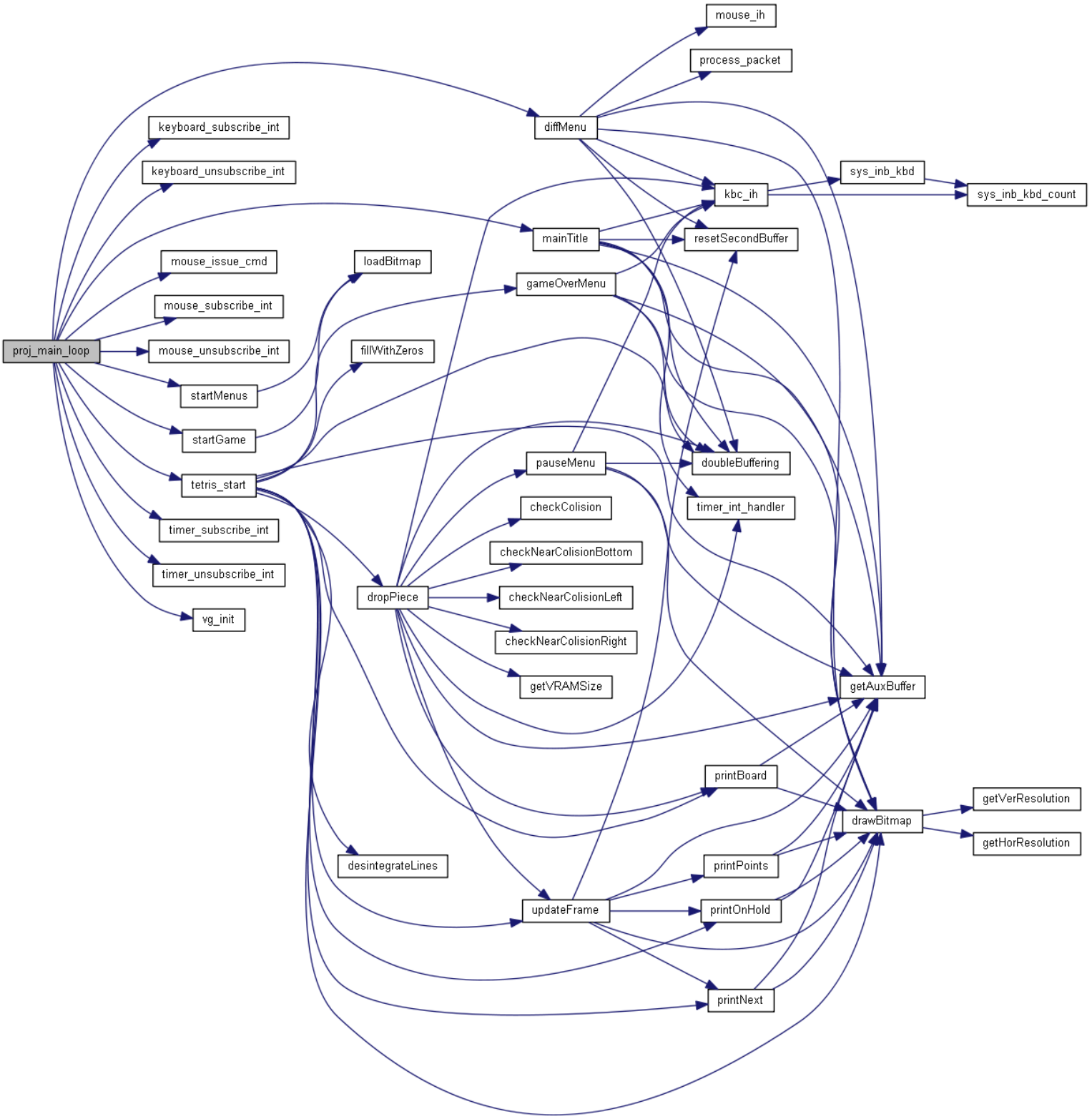
Gonçalo – XX%

Tiago – XX%

2.3. Peso dos módulos

Módulo	Peso Relativo %
Bitmap	10
i8042	5
i8254	5
Keyboard	15
Menus	13
Mouse	5
Tetris	17
Timer	13
Vídeo_gr	9
Proj	8

2.4. Gráfico de chamadas de funções



3. Implementação

Gostaríamos de começar por explicar melhor como está feita a parte principal do projeto, o módulo tetris. Os blocos que usamos para construir as peças tem de dimensão 32x32px, portanto na área de jogo cabem 15x22 blocos. Tivemos a ideia de criar um array 2D (board) que seria codificado com números correspondentes à cor do bloco que estaria naquela posição do array (0 para posição não ocupada). Desta forma seria mais simples lidar com colisões e gestão de posições de blocos.

A melhor estratégia que encontramos para lidar com colisões foi criar um novo array, com o mesmo tamanho de board, mas desta vez com valores booleanos, (true – ocupado), desta forma saberíamos se o bloco que estava a cair estaria a ocupar uma posição previamente preenchida ou não. Esta array (status) é atualizado no fim de cada queda.

Desta forma, o pensamento é o seguinte:

1- Cai uma nova peça, é determinado se houve colisão, se houver termina o jogo, pois não é possível gerar mais peças;

2- As funções checkNearColisions ajudam a saber se, antes de deslocar a peça para um dos lados ou estando apenas a cair, vai colidir com as margens do jogo ou outras peças. Desta forma, o algoritmo decide se deixa ou não a peça movimentar.

3- Quando a função checkNearColisionBottom retornar true, significa que depois de uma atualização de posição e a peça vai estar em cima de uma outra ou depois da margem, e, portanto, interrompe ali o movimento da peça.

4- Ao rodar a peça tivemos dificuldades a saber como é que a peça iria reagir de forma a não colidir com outras peças já colocadas:

Implementamos um algoritmo complexo que avalia como será a melhor forma da peça se adaptar para que não colida com peças já colocadas:

a. Se a peça estiver bloqueada à esquerda e à direita e ao subir uma posição ainda colidir o movimento de rotação é bloqueado, caso contrário a peça sobe uma posição e roda.

b. Se a peça estiver bloqueada à esquerda ou direita, será movimentada para a direita ou esquerda, respetivamente, se possível, caso contrário o movimento de rotação é bloqueado.

Uma vez que a parte mais importante do nosso projeto seria algo bastante geométrico e regular, e num formato de grelha, achamos melhor utilizar este método de arrays.

Gostaríamos ainda de realçar o método que utilizamos para rodar as peças. Depois de muitas horas de pesquisa, em vão, à procura de um algoritmo que fosse capaz de rodar elementos específicos de um array sem alterar os restantes, decidimos utilizar matemática e geometria para criarmos o nosso próprio algoritmo para a rotação das peças.

Um facto curioso, apesar de cada peça ter o seu algoritmo de rotação, esta implementação levou menos tempo a ser elaborada do que a procura por um algoritmo universal e mais eficiente.

Finalmente gostaríamos de elaborar como mapeamos os blocos individuais de cada peça. Sabemos quantos blocos tem cada peça, portanto criamos um novo array de posições “pos” que guarda uma posição Y e uma posição X para um bloco individual, então por exemplo, digamos que a peça 2, com 2 blocos, tenha cor 3, o array pos[2][2] seria algo do género:

{ {0,0} ,{1,0} }

Significa que o primeiro bloco estaria na posição Y=0, X=0 e o segundo na posição Y=1, X=0. A área de jogo está orientada para a direita no eixo dos xx e para baixo no eixo dos yy. Desta forma conseguimos sempre atualizar os arrays board e status utilizando a seguinte formula:

board[pos[i][0]][pos[i][1]] = [color] - cor de 1 a 7 ou 0 para blank

status[pos[i][0]][pos[i][1]] = [true | false]

No fim para escrever o board no ecrã com a função printBoard seria apenas um “for” com as dimensões que queremos.

4. Conclusão

No geral o conteúdo das aulas de Laboratório de Computadores é interessante, no entanto é um assunto que não nos era nada familiar, e, portanto, foi uma espécie de choque no início do semestre. Consideramos que as aulas práticas deveriam ser um pouco mais didáticas.

O teste de programação é um teste razoável, mas basta um pequeno deslize ou um esquecimento, e somos sujeitos a prova de recurso para salvarmos a U.C. por uma componente que representa 10% da classificação.

As principais dificuldades foram juntar tudo o que fizemos nos laboratórios, implementar os algoritmos relacionados com as colisões e implementar um algoritmo eficiente na rotação das peças.

Deixamos aqui o nosso agradecimento ao Henrique Ferrolho pelo módulo de carregamento e apresentação de bitmaps que postou no seu blogue, cujo link deixamos acima, na primeira vez que lhe fazemos referência.

O projeto foi sem dúvida um desafio, mas foi a melhor parte do decorrer da U.C.

Nota: para correr o projeto basta executar os seguintes comandos na pasta ~/proj:

```
make depend && make

lcom_run proj
```