Faculdade de Engenharia da Universidade do Porto

Tetris

Laboratório de Design e Teste de Software

Entrega Intermédia - Grupo 1505

2021/2022 – Licenciatura em Engenharia Informática e Computação

Dinis Ribeiro dos Santos Bessa de Sousa

(up202006303@edu.fe.up.pt)

Francisca Oliveira e Silva

(up202005140@edu.fe.up.pt)

Miguel Ângelo Silva Teixeira

(up202005208@edu.fe.up.pt)

Docentes:

Rui Maranhão de Abreu (rma@fe.up.pt)
Nuno Flores (nflores@fe.up.pt)

Introdução

Relatório elaborado no âmbito da Unidade Curricular: Laboratório de Desenho e Teste de software lecionada no 2º ano do ciclo de estudos de Licenciatura em Engenharia Informática e Computação na FEUP (Faculdade de Engenharia da Universidade do Porto), sob mentoria do professor Nuno Flores.

Índice

Descrição do Projeto

Este projeto é inspirado num Jogo eletrónico muito popular, o Tetris, lançado em 1984.

Tetris é caracterizado por empilhar peças formadas por blocos, com quatro por peça, que descem o ecrã de forma a completar linhas horizontais. Quando uma linha é formada com sucesso, o jogador recebe pontos, a linha é destruída e as linhas superiores a esta descem. O Jogador deve tentar obter o máximo de pontos possíveis, necessitando para isso de "encaixar" as peças de forma inteligente para estas não chegarem ao topo do ecrã. Caso isto aconteça, a partida acaba.

Lista de Funcionalidades

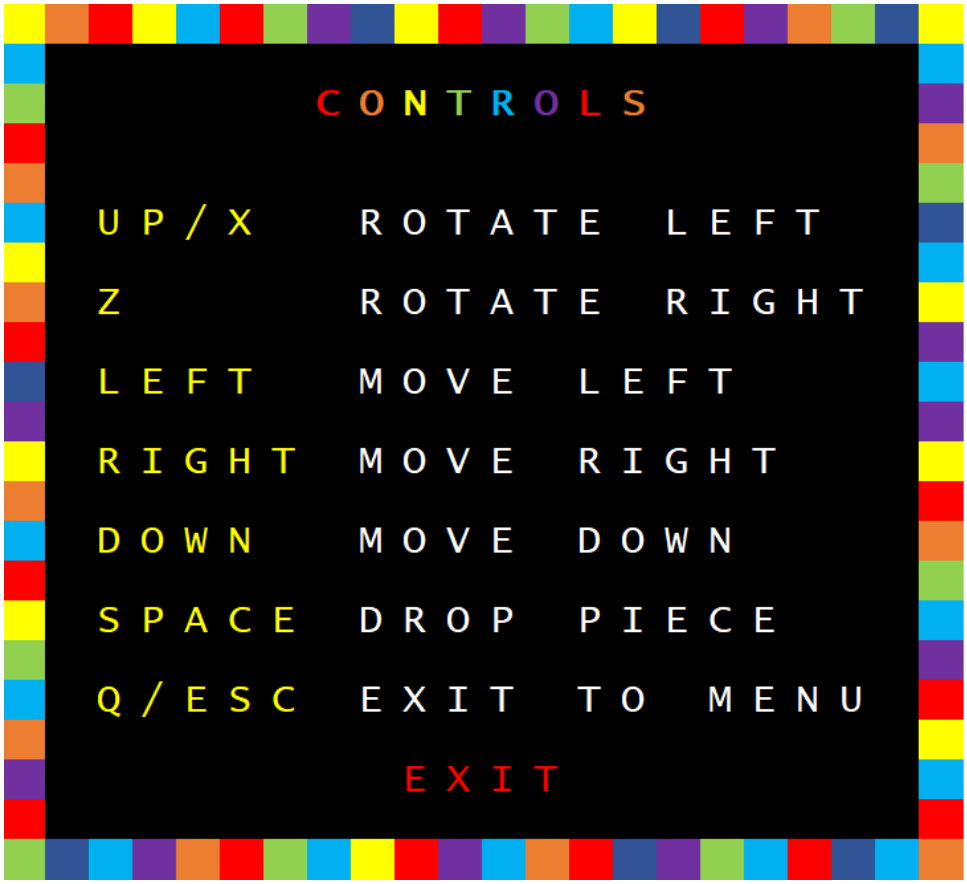
Para o utilizador, o jogo está dividido em três estados: o menu, as regras e o jogo em si.

1. Menu

O menu, depois de ser mostrado ao utilizador deve ser capaz de receber o seu input e direcioná-lo para a opção escolhida:

- Se escolher PLAY, o programa deve passar do estado de menu para o estado de jogo e começar a sua execução;
- Se a opção for CONTROLS, deve ser mostrada uma tela com as regras do jogo, e uma opção para voltar atrás;
- A opção EXIT termina a execução do programa.

2. Regras



Controlos

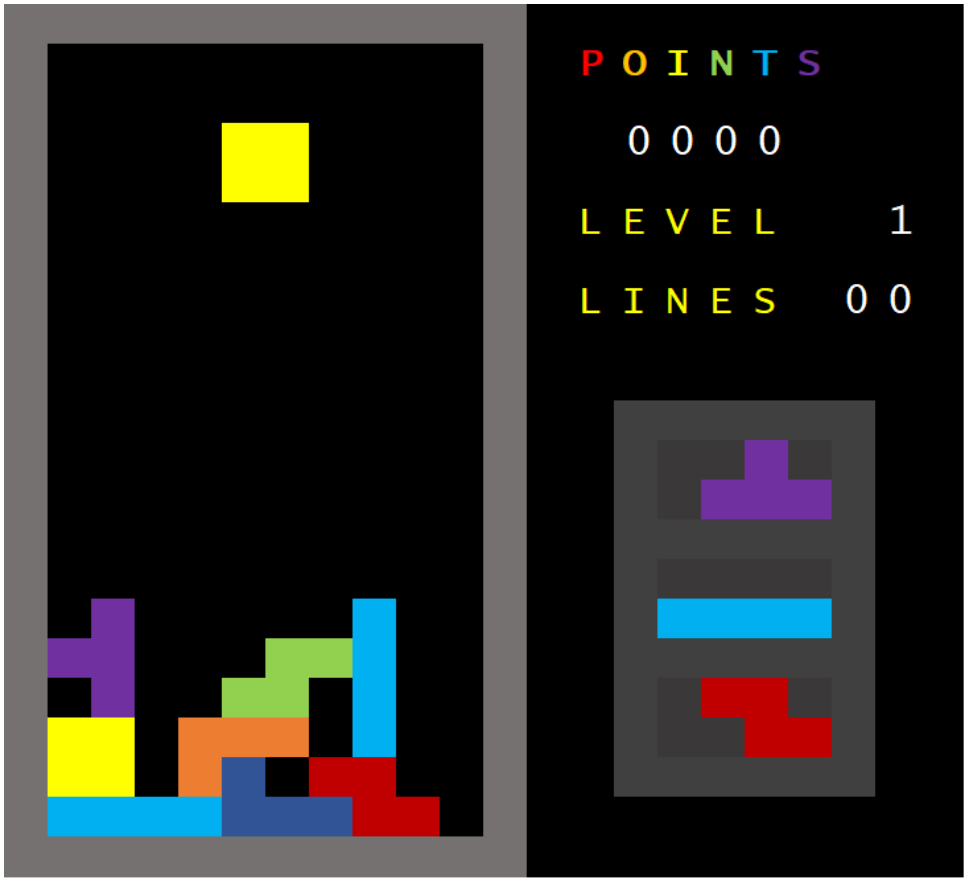
Jogo:

Up e X - Rodar no sentido dos ponteiros do relógio.
Z - Rodar no sentido contrário ao dos ponteiros do relógio
Left - Shift para a esquerda
Right - Shift para a direita
Down - Shift para baixo
Space - Deixar cair a peça
Q e ESC - Voltar ao menu

Menu:
Up - Opção em cima
Down - Opção em baixo
Enter - Selecionar opção
Q e ESC - Sair

Regras:
Q e ESC - Voltar ao menu

3. Jogo



Durante a execução do jogo há quatro grandes focos:

- A peça em movimento;
- O tabuleiro, onde estão todos os blocos caídos;
- A fila das próximas peças a cair;
- A pontuação.

3.1. Peça

A peça (tetriminó) é composta por 4 blocos posicionados entre eles de 7 formas diferentes. Ao descer no tabuleiro pode mover-se para a esquerda ou para a direita, dentro dos limites do tabuleiro, bem como rodar à volta do seu centro para encaixar nos blocos já existentes. Cada uma das 7 formas tem o seu método específico de rodar dependendo também da sua orientação.

A classe Tetrimino é superclasse de cada uma das formas. Esta classe tem funções capazes de calcular e devolver a nova posição da peça através do movimento do utilizador, seja ele uma translação ou rotação.

3.2. Tabuleiro

O tabuleiro, para além de guardar todos os blocos numa matriz, tem capacidade de saber se tem linhas completas e de as eliminar. Ao receber a nova posição da peça a cada movimento do utilizador, o tabuleiro deve ter funcionalidades que lhe permitam saber se pode mover ou rodar a peça na direção selecionada pelo utilizador. Se o movimento for uma rotação que resulte em algum bloco fora do tabuleiro ou num lugar ocupado por outra peça, este mesmo deve tentar fazer translações de forma a ser possível rodar nessas situações.

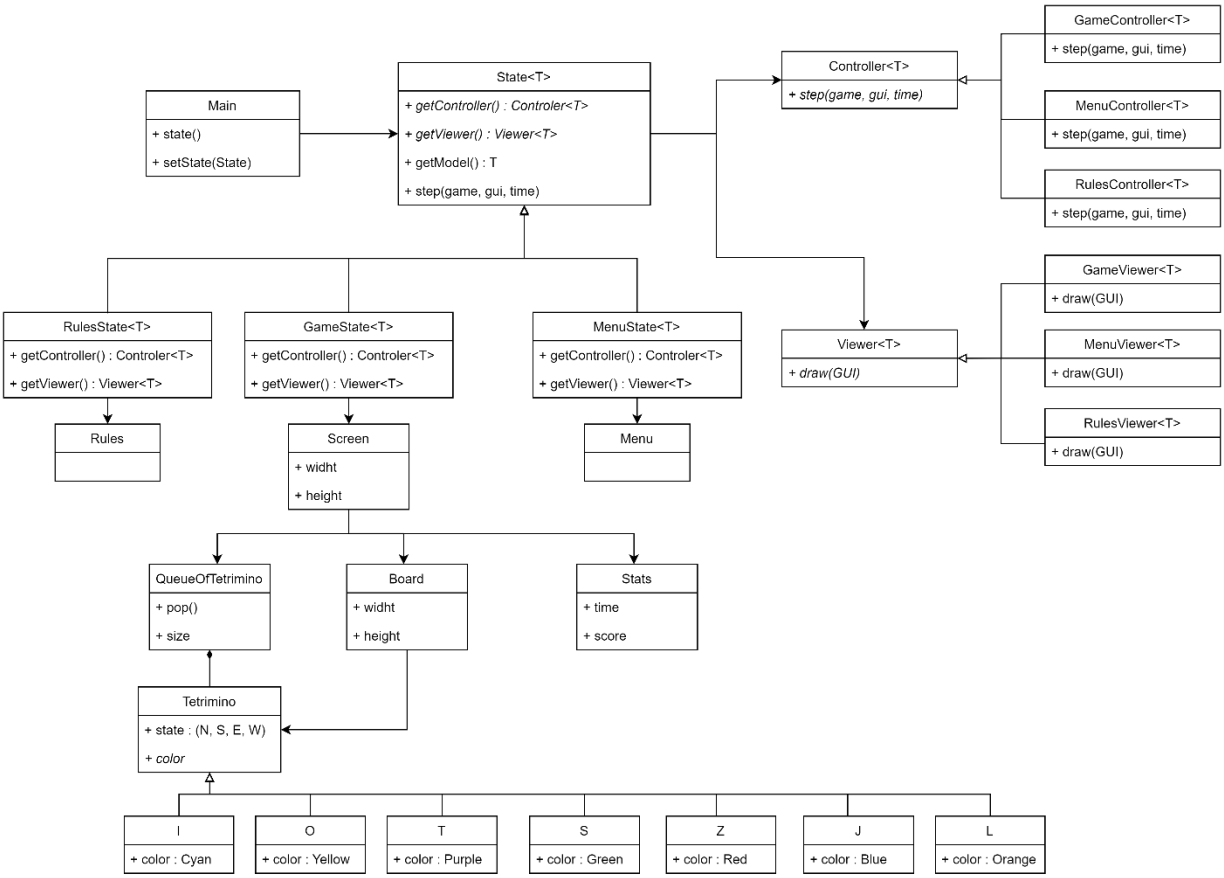
3.3. Fila

A fila, composta por três peças, é preenchida aleatoriamente com um dos 7 tipos de peças disponíveis. À medida que o utilizador coloca peças no tabuleiro, caso ainda não tenha atingido o topo, é lhe dada a primeira peça na fila e aparece uma nova no seu fim.

3.4. Pontuação

A cada fila removida ou a cada bloco colocado na grelha, o jogador recebe o respetivo número de pontos. Após um determinado número de linhas completadas, o jogador passa para o nível seguinte, onde a peça desce mais depressa.

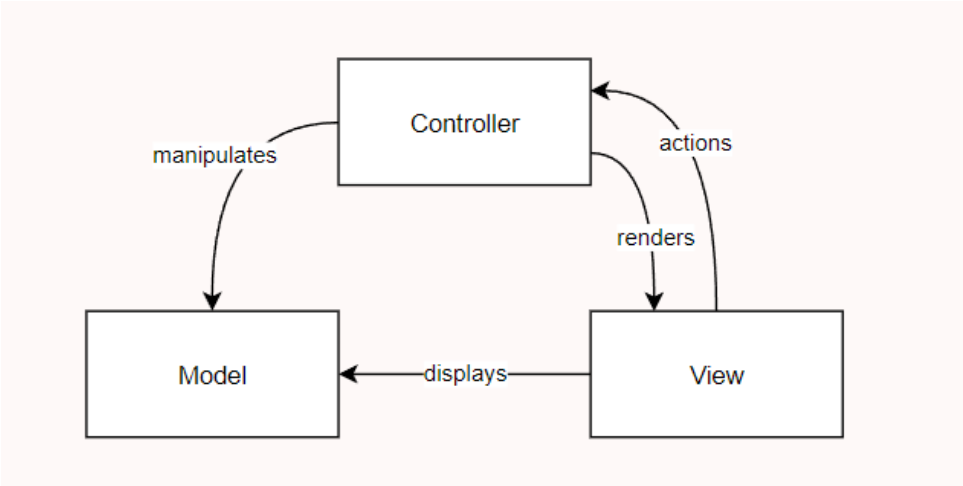
UML


Main
+ state()
+ setState(State)
State<T>
+ getController() : Controler<T>
+ getViewer() : Viewer<T>
+ getModel() : T
+ step(game, gui, time)
Controller<T>
+ step(game, gui, time)
GameController<T>
+ step(game, gui, time)
MenuController<T>
+ step(game, gui, time)
RulesController<T>
+ step(game, gui, time)
RulesState<T>
+ getController() : Controler<T>
+ getViewer() : Viewer<T>
GameState<T>
+ getController() : Controler<T>
+ getViewer() : Viewer<T>
MenuState<T>
+ getController() : Controler<T>
+ getViewer() : Viewer<T>
Viewer<T>
+ draw(GUI)
GameViewer<T>
+ draw(GUI)
MenuViewer<T>
+ draw(GUI)
RulesViewer<T>
+ draw(GUI)
Rules
Screen
+ widht
+ height
Menu
QueueOfTetrimino
+ pop()
+ size
Board
+ widht
+ height
Stats
+ time
+ score
Tetrimino
+ state : (N, S, E, W)
+ color
I
+ color : Cyan
O
+ color : Yellow
T
+ color : Purple
S
+ color : Green
Z
+ color : Red
J
+ color : Blue
L
+ color : Orange

Padrões de Desenho

MVC Architecture Pattern

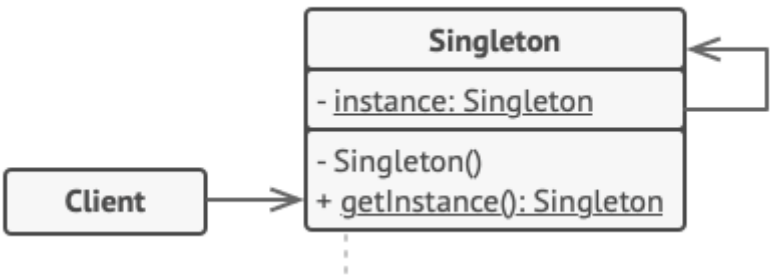
Toda a aplicação está divida em três partes: 'Model', 'Controller' e 'View'.



- O modelo guarda apenas a informação
- A vista é onde a informação do modelo é mostrada ao utilizador. Os inputs do utilizador são recebidos pela vista e enviados para o controlador.
- O controlador "transforma" ações do user em mudanças ao modelo ou ao estado do programa e envia informações sobre o model para a vista.

Singleton Pattern

Implementado na classe 'Main', de forma a garantir que apenas existe um objeto desta classe, visto ser esta classe a classe "mãe" de todo o jogo.



State Pattern

Usado para permitir mudar do estado entre ‘Game’, ‘Menu’ e ‘Rules’. A classe ‘Main’ guarda o estado atual, e esse estado permite à ‘Main’ usar o ‘Controller’ e ‘View’ próprios de cada estado. Isto permite que o código seja fácil de manter e evita o uso de *switch clauses* demasiado longas.

Factory Method Pattern

É implementado no jogo no seguimento do uso do State Pattern. Os diferentes estados usam a função de ‘getController()’ e ‘getViewer’ de uma classe estado abstrtata que permite que ambos os estados usem controladores e vistas diferentes, criando objetos diferentes.

Game Loop Pattern

O loop do jogo corre continuamente enquanto o jogo está a decorrer. A cada loop, o input é processado sem parar o jogo, o modelo é atualizado e o jogo renderizado. A frequência com que isto acontece é determinada pela ‘framerate’ do jogo.

Testes

No decorrer do trabalho serão implementados testes unitários, usando JUnit, de forma a garantir a correção do código. Os testes serão elaborados, segundo uma abordagem do tipo top-down, *a priori* da implementação de qualquer parte do código, para evitar que os testes sejam feitos "à medida do código". Idealmente serão testados não só o modelo e os controlos, mas também a interface.

Para criar mutações de forma a garantir a qualidade dos testes e fazer um relatório sobre a *coverage* dos mesmos será usada a ferramenta PIT.