Universidade do Porto

FEUP Faculdade de Engenharia

Inteligência Artificial

Pesquisa aplicada ao jogo Tetris

Relatório Final

Francisco Pinto
João Xavier
João Ribeiro

Maio 2009

Índice

1. Objectivo

O objectivo deste trabalho consiste na implementação de um jogo *Tetris* que utilize inteligência artificial de modo a cumprir as metas do jogo em questão.

A inteligência artificial tem como base, variantes de algoritmos de pesquisa, entre os quais alguns de pesquisa inteligente que através de uma função de heurística, baseada em regras e estratégias inerentes ao *Tetris*, avalia e escolhe a jogada mais apropriada de modo a atingir os objectivos do jogo, "limpar linhas" fazendo a melhor pontuação possível.

2. Motivação

Em termos académicos, a realização deste trabalho (em particular, a implementação dos vários algoritmos de pesquisa propostos, com respectivos testes e comparação) terá como principal propósito conhecer a sua aplicabilidade e diferente grau de eficácia. Além disso, poderá servir como base para outros problemas da mesma temática (videojogos) ou de áreas diferentes, onde é necessário encontrar o melhor conjunto de escolhas para optimização do resultado final.

A realização deste projecto, trouxe outros benefícios, uma vez que o *Tetris* é considerado como um bom estímulo para actividade cerebral, a implementação de um tipo de inteligência artificial para este jogo tornou-se num agradável desafio, despertou interesse e perspectivas para problemas na mesma área.

3. Descrição

3.1. Funcionalidades

3.1.1. Conceito

O jogo consiste num tabuleiro de 20 linhas por 10 colunas, portanto, 200 células, que podem estar preenchidas ou vazias.

Há sete tipos de peças, que são todas as combinações possíveis de 4 blocos ortogonalmente adjacentes. Vulgarmente atribui-se nomes de letras às peças, atendendo à sua forma, sendo estas: O, I, S, Z, L, J, T

Um exemplo da sua representação na implementação realizada é:

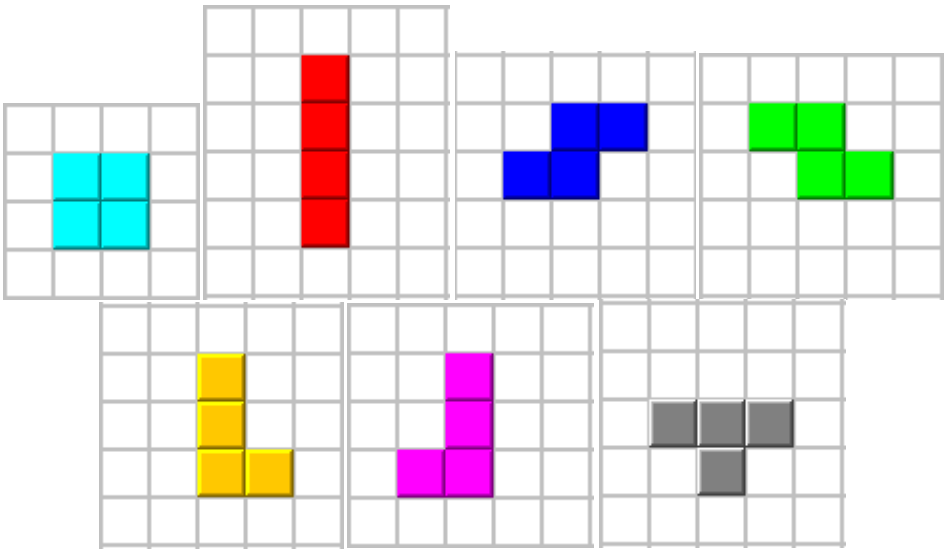

Figura 1: Peças do jogo Tetris {O, I, S, Z, L, J, T}

As peças podem efectuar 5 tipos de movimentos distintos, sendo estes:

- translação à esquerda (uma casa);

- translação à direita (uma casa);

- translação para baixo (uma casa);

- translação total para baixo (todas as casas possíveis);

- rotação no sentido dos ponteiros do relógio.

Qualquer uma destas jogadas está limitada pela existência de casas ocupadas a bloquear o caminho e pelas paredes.

Se em algum momento o tabuleiro tiver uma ou mais linhas preenchidas estas são imediatamente removidas atribuindo uma pontuação ao jogador baseada no número de linhas apagadas de uma vez em função do nível.

A função usada de cálculo da pontuação é a seguinte, baseado nas variáveis *lines*, *score* e *level*:

If (*lines* == 1) *score* := *score* + 100 + 5**level*;

If (*lines* == 2) *score* := *score* + 400 + 20**level*;

If (*lines* == 3) *score* := *score* + 900 + 45**level*;

If (*lines* == 4) *score* := *score* + 1600 + 80**level*;

As peças efectuam com o decorrer do tempo uma translação para baixo de uma unidade. O tempo que esta translação demora a acontecer diminui linearmente enquanto o nível sobe.

Há um máximo de 20 níveis, e um novo nível (antes do nível máximo) é alcançado a cada 10 linhas removidas do tabuleiro.

O objectivo do jogo é então pontuar o máximo possível antes que o tabuleiro de jogo fique obstruído de tal forma que não possibilite a colocação da peça seguinte.

3.1.2. Aplicações relevantes

O jogo data do ano de 1985, e desde então, inúmeras implementações para vários sistemas operativos e plataformas de jogos têm sido desenvolvidas, umas mais bem sucedidas e reconhecidas que outras.

A primeira versão foi programada para o sistema *Elektronika 60*.

Já em 1989, foram atribuídos direitos de autor à Nintendo, que distribuíu o jogo na sua consola da altura, a NES. Ainda nos dias de hoje esta é das versões mais conhecidas.

Em 1993, uma das versões mais conhecidas do jogo para o sistema operativo *Windows* saiu com o *Microsoft Entertainment Pack*.

3.1.3. Funcionalidades adicionais

O programa, ao ser corrido, oferece ao utilizador um menu com várias opções que o permitem decidir se é ele que vai jogar ou se o computador, assim como a heurística e o número de linhas inicial.

Dentro do jogo, é possível pausar, consultar a tabela dos maiores pontuados e reiniciar o jogo actual, quer através de um menu quer de botões na interface. Quando o jogador não tem a janela do jogo como selecção, o jogo é automaticamente pausado.

Além destas funções de controlo, é possível ao utilizador jogar em qualquer altura (mesmo intervir se for o computador a jogar) até aparecer o ecrã de fim de jogo, quando não é possível colocar mais nenhuma peça.

3.2. Estrutura do Programa

O programa, de modo a um maior aproveitamento das potencialidades das linguagens orientadas a objectos e ao polimorfismo, encontra-se subdividido em alguns módulos, sendo estes:

3.2.1 Figure

Este é o módulo mais básico de todo o projecto. É uma classe abstracta que garante capacidade aos seus filhos de realizar operações simples de movimento, como translações e rotações à esquerda e direita.

3.2.2. Figures

Contém todas as 7 peças do jogo Tetris que estendem as funcionalidades do módulo Figure. As funções de rotação e as cores são diferentes para cada peça, assim como a sua aparência. Estas informações são guardadas na classe mãe para preservar o polimorfismo modular.

3.2.3. FigureFactory

A aplicação deste módulo é muito importante na lógica do jogo, pois devolve aleatoriamente qualquer uma das 7 peças e guarda o registo das que já saíram, embora esta informação não seja disponibilizada ao utilizador.

3.2.4. TetrisGrid

Nesta classe é feita a representação do estado do jogo, assim como a leitura e escrita das melhores pontuações (*HiScore)*, validações de jogadas, eliminação de linhas, adição de linhas e avaliação do estado do tabuleiro tendo em conta as heurísticas previamente estabelecidas. Para este fim, são frequentemente manipulados objectos do tipo *Figure*.

3.2.5. HiScore

Módulo responsável pelo registo das três melhores pontuações, escritas para e lidas do ficheiro *iartris.DAT*. É chamada uma leitura a este ficheiro aquando da inicialização do programa, e nele escritas quando este mesmo acaba, com as modificações feitas no decorrer do programa.

3.2.6. Movement

Módulo que trata exclusivamente da avaliação e heurísticas. Guarda as informações da jogada efectuada, passando à parte de cálculo da pontuação dado um objecto do tipo *TetrisGrid*. A pontuação é calculada com base em diferentes pesos das funções heurísticas chamadas sobre o tabuleiro, e somada com a pontuação da próxima melhor jogada (se esta estiver a ser computada).

3.2.7 IartrisMainFrame

Módulo genérico que tem a seu cargo múltiplas funções, abrangendo a lógica do jogo, a interface com o utilizador, o controlador do jogo (humano / teclado e computador / automatizado) e a geração da árvore de pesquisa.

A nível da lógica do jogo, contém métodos que possibilitam a rotação e translação das peças, obter a próxima peça da fila (próximo movimento) e reiniciar o jogo.

A interface conta com funções que estão encarregues de lidar com todos os menus, botões e painéis, estado de jogo, tabela de pontuações e redesenho do tabuleiro.

A geração da árvore de pesquisa é efectuada por um método cuja recursão está directamente ligada ao nível de *lookup* da peça seguinte, isto é, o número de peças que o programa sabe que existem a contar com a peça actual. É auxiliado por outro método que permite retornar a melhor solução dependendo da heurística que está a ser utilizada.

3.3. Esquemas de Representação do Conhecimento

Como representação do estado de jogo e peças, optou-se por manter a implementação em LinkedList de int[], a guardar as cores das figuras (0 = célula vazia), pois fazia parte da base do trabalho.

Para guardar as informações das jogadas e suas pontuações usa-se a classe Movement, que contém três inteiros, correspondentes ao número de jogadas à esquerda, direita e rotações, um float para armazenar a pontuação e um filho da mesma classe.

De modo a percorrer todas as jogadas possíveis, optou-se por gerar uma árvore de pesquisa através de uma função recursiva simples limitada pelo nível de jogadas à frente, utilizando pesquisa em profundidade.

São usados vários métodos de cálculo de pontuação (heurísticas) que determinam o quão longe o tabuleiro está do estado ideal (vazio):

H1: Número de buracos (casas livres por baixo de peças) no tabuleiro

H2: Altura máxima do tabuleiro

H3: Altura média do tabuleiro

H4: Diferença entre a altura máxima e mínima do tabuleiro

H5: Desvio padrão das alturas do tabuleiro

H6: Número de peças por cima de buracos

H7: Distribuição pesada das alturas do tabuleiro

3.4. Implementação dos Esquemas de Representação do Conhecimento

H1 calcula todos os espaços vazios no tabuleiro que se encontram numa posição inferior a células preenchidas.

H2 percorre todas as colunas, retornando o número de blocos da maior delas, a contar do fundo do tabuleiro.

H3 soma as alturas de todas as colunas, dividindo pelo número total de colunas (10), retornando a média.

H4 devolve a diferença entre a altura máxima e a altura mínima de todas as colunas no tabuleiro.

H5 calcula a soma dos módulos da distância entre a altura de cada coluna e a média das alturas.

H6 calcula a carga de cada coluna, ou seja, conta todas as peças que se encontram acima do primeiro buraco de cada coluna.

H7 retorna a soma das alturas de todas as colunas, depois de as multiplicar um factor que vai aumentando linearmente a cada coluna. Neste caso a altura da 1ª coluna é multiplicada por 0, a 2ª por 2, a 3ª por 4 e assim sucessivamente até à última.

Embora tenham sido propostos vários algoritmos de pesquisa, como pesquisa em profundidade, largura, bidireccional ou A*, a situação foi estudada cautelosamente e chegou-se à conclusão que pesquisas que necessitassem um caso final seriam impossíveis de implementar. Tendo em conta que a pesquisa principal a incluir no trabalho (A*) necessita de ter em conta, para além de valores heurísticos, o peso de cada aresta do grafo, cujo custo é todo idêntico, considerou-se, na função f(x) = g(x) + h(x), onde f(x) é o resultado da avaliação do algoritmo A* para um peso g(x) e uma heurística h(x), g(x) = 0.

Dado que a função de pesquisa está limitada ao valor heurístico, optou-se pela implementação de uma única pesquisa (em profundidade) e a dedicar maior trabalho a nível de diferentes heurísticas, que têm inferência directa na qualidade de jogo do computador.

Desta forma, para a peça actual, são geradas todas as combinações possíveis de movimentos e rotações:

O: 10 = 10 translações * 1 rotação

I, S, Z: 20 = 10 translações * 2 rotações

T, J, L: 40 = 10 translações * 4 rotações

É simulado, para cada jogada, que a peça cai no tabuleiro e procede-se a uma avaliação deste com diferentes pesos nas heurísticas descritas acima. Para cada peça, todas as jogadas são guardadas num ArrayList<Movement>, e retirada a melhor jogada.

No caso de ser uma heurística que verifica o movimento a seguir, primeiro repete-se o processo com o tabuleiro obtido para a peça seguinte, e soma-se a pontuação da melhor segunda jogada à pontuação da primeira. Este processo repete-se recursivamente dependendo do número de peças a ver à frente.

Embora seja suportado virtualmente qualquer número de peças à frente a ter em conta, a partir da 3ª peça a geração da árvore torna-se morosa, e devido ao facto de esta geração não ser feita em concorrência com o jogo, este atrasa-se ao invés de simplesmente demorar a jogar, daí que se optou por simplesmente implementar uma heurística que vê uma só peça à frente da actual.

3.5. Análise da complexidade dos algoritmos

A nível de heurísticas, todas percorrem o tabuleiro de igual modo, portanto, para um valor **C** de comprimento (10 colunas) e **L** de largura, correr as 7 heurísticas terá uma complexidade temporal de O(**C*****L***7) = O(n), e uma complexidade espacial de O(1), pois não é gerado nenhum contentor para além do tabuleiro existente.

O algoritmo de pesquisa usado tem dois factores importantes a ter em conta, sendo estes o nível de *lookahead* (sendo que 2 é ver uma peça à frente) e os movimentos, que dependendo da peça podem ser da ordem de 10, 20 ou 40.

Em termos temporais, o algoritmo processa em média em O(((40*3 + 20*3 + 10)/ 7)^*lookahead*) e na pior das hipóteses em O(40^*lookahead)* = O(n^*lookahead*). Como guarda a cada iteração todos os movimentos, irá consumir exactamente a mesma memória.

4. Ambiente de desenvolvimento

Este projecto foi desenvolvido utilizando a linguagem Java com o suporte do IDE Eclipse 3.4.2 e com JRE 6 Update 13.

Como software extra, e que serviu como base, foi utilizado um jogo chamado Jetris, uma implementação em Java do jogo Tetris, mas que suporta apenas a componente de um jogador controlado por um humano. Desta forma é possível aproveitar a interface gráfica deste projecto para acrescentar os módulos de inteligência artificial.

Ao nível das máquinas utilizadas foram várias e apenas se verificou diferenças ao mudar de sistema operativo, sendo usado o Windows XP e Vista, denotando-se algumas limitações no último.

5. Avaliação do programa

Os factores mais importantes em inteligência artificial aplicada a jogos são a sua performance e desempenho em relação aos objectivos do jogo em questão.

No caso do *Tetris* após a realização de vários testes em condições iguais, como base de comparação e selecção seria o número médio de linhas e/ou pontos feitos por jogo, visto que o objectivo deste jogo é pontuar o máximo possível até se ultrapassar as mais famosas 20 linhas da história dos videojogos.

Nos videojogos, em particular, a interface tem sempre um papel determinante; no caso deste projecto, foi conseguida, a partir de uma base acima referida, uma interface "limpa" e de fácil uso.

6. Resultados Experimentais

De modo a aperfeiçoar as funções heurísticas utilizadas, foram realizados diversos testes ao programa, utilizando sempre a mesma *seed* de geração de peças aleatórias, para se poder comparar directamente os resultados obtidos por duas soluções diferentes.

Para facilitar a comparação das soluções utilizadas, isto é, dos pesos das diferentes heurísticas, a sua representação será a seguinte:

(H1, H2, H3, H4, H5, H6, H7)

Neste esquema podem ver-se os pesos dos valores das heurísticas H1 a H7 (consultar secção 3.3).

Durante a implementação do algoritmo *greedy*, após várias modificações, os valores ficaram (60, 40, 0, 0, 0, 0), conseguindo assim limpar 42 linhas. Este foi o caso base de comparação para as modificações seguintes.

Inicialmente não foi utilizada a possibilidade de ter em conta a peça seguinte, para melhorar tanto quanto possível a jogada simples. Começou-se então pela solução (30, 15, 0, 0, 0, 0, 1), obtendo-se 33 linhas limpas. Com o último valor a 1, tentou-se que o computador encostasse as peças ao lado esquerdo. Seguidamente, e para começar a utilizar as outras funções disponíveis, foi utilizada a solução (9, 6, 3, 8, 10, 8, 1), com a qual se completaram 34 linhas, ligeiramente melhor, mas com muitos buracos pelo meio. Por esse motivo, a solução seguinte, (12, 6, 4, 2, 3, 5, 1), penalizava mais os buracos criados. Com esta, conseguiram-se 44 linhas. Ainda assim, o espaço do tabuleiro não era muito bem gerido.

Por outro lado, verificou-se que seria proveitoso que a estratégia utilizada variasse para diferentes estados do tabuleiro. Foi então criada uma nova verificação: assim que a coluna mais alta do tabuleiro passasse as 10 unidades de altura, os pesos das funções heurísticas seriam diferentes, dando-se prioridade a diminuir rapidamente a altura do tabuleiro, em detrimento dos pontos efectuados e de possíveis buracos criados (*despair mode*).

Então, e utilizando agora um esquema do género

(H1, H2, H3, H4, H5, H6, H7 | D1, D2, D3, D4, D5, D6, D7)

com a solução (12, 6, 5, 2, 3, 7, 1 | 6, 12, 5, 2, 3, 7, 1) foi possível atingir as 132 linhas. Depois de algumas mudanças sucessivas dos valores, (50, 25, 3, 5, 5, 1, 1 | 35, 70, 2, 10, 5, 1, 1) foi a solução com que se obteve o melhor resultado, 218 linhas.

Passou-se, então, a considerar a peça seguinte no cálculo da melhor jogada, utilizando a solução obtida anteriormente, conseguindo-se completar 286 linhas.

Até ao momento, o melhor resultado conseguido foi 589 linhas.

7. Conclusão

Após a realização deste trabalho é possível concluir que a implementação de uma inteligência artificial capaz atingir os objectivos de um jogo pode-se tornar complexa e necessitar de um profundo conhecimento das regras e estratégias a ele associadas.

A fase de testes foi essencial para determinar quais os aspectos a ter conta no tabuleiro, alguns destes menos óbvios e que antes eram dados como irrelevantes na estratégia de jogo, e a sua melhor combinação de importância (pesos) para aperfeiçoar a função de heurística.

Ao longo da fase de implementação foi gratificante, ao melhorar os valores e parâmetros da função de heurística, verificar um aumento na capacidade de decisão e respectivamente a obtenção de melhores resultados.

Conclui-se após a realização deste trabalho que o desenvolvimento de agentes é uma tarefa que se pode tornar bastante árdua e complexa principalmente quando confrontada com a dificuldade de obtenção de dados exactos em ambientes com alguma complexidade ou, como neste caso, não conhecidos *a priori*.

Desde que se estabeleça uma forma de atacar o problema, um agente pode proporcionar bons resultados, o que é bastante gratificante, dando a sensação que se conferiu "inteligência" a uma máquina.

8. Melhoramentos

Em perspectiva futura, este trabalho poderá evoluir para um jogo onde existirão dois adversários competindo directamente por uma melhor pontuação. Estes adversários poderão ser humano contra um adversário artificialmente inteligente, com diferentes graus de dificuldade consoante a complexidade da heurística e algoritmo de pesquisa utilizado, ou então o computador controlar ambos os adversários em competição onde é possível observar em tempo real as diferenças, vantagens e desvantagens de um algoritmo em relação a outro neste tipo de problema.

Uma boa ideia para melhorar as penalizações nas heurísticas, seria desenvolver um algoritmo genético que gerasse populações de soluções, e fosse evoluindo para encontrar uma solução óptima ou próxima dela.

Em termos de eficiência do programa, esta poderia ser melhorada com a criação de um thread para geração da árvore de pesquisa e cálculo da função de heurística e posteriormente reduzir o número de nós da pesquisa através de uma geração de casos específica para cada tipo de peça.

9. Bibliografia

• Site do projecto *Jetris*; consultado em Abril de 2009

http://jetris.sourceforge.net/

• Técnicas de *Tetris*; consultado em Abril de 2009

http://www.angelfire.com/pa5/b_g_stuhan/Tetris.html

• Páginas da Wikipedia referentes aos algoritmos a utilizar; consultadas em Abril de 2009

*http://en.wikipedia.org/wiki/A**

http://en.wikipedia.org/wiki/Breadth-first_search

http://en.wikipedia.org/wiki/Depth-first_search

http://en.wikipedia.org/wiki/Greedy_algorithm

http://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search

http://en.wikipedia.org/wiki/Bidirectional_search

• Site com visualização da aplicação de alguns dos algoritmos; consultado em Abril de 2009

http://el-tramo.be/software/jsearchdemo; consultado em Abril de 2009

• Relatório de uma implementação de inteligência artificial para o jogo Tetris; consultado em Abril de 2009

http://www.cs.cmu.edu/afs/cs/project/ACRL/www/TetrisReports/RodriguezLindzeyTeschHW1.pdf

• Fórum sobre inteligência artificial aplicada ao jogo Tetris; consultado em Abril de 2009

http://www.gamedev.net/community/forums/topic.asp?topic_id=391139

• Artigo que descreve como um computador pode jogar o jogo Tetris; consultado em Abril de 2009

http://www.colinfahey.com/tetris/tetris_en.html

• Visão geral de uma possível arquitectura de Tetris

http://cslibrary.stanford.edu/112/Tetris-Architecture.html

10. Apêndices

I. Manual do utilizador

I.1. Instruções de compilação

O ficheiro *make.bat* contém a instrução necessária à compilação do jogo, sendo esta "*jar cvfm iarTris.jar manifest.mf IartrisMain.* iartris.DAT .\main\iartris .\res\ .\score\iartris*" supondo que se está na pasta de origem do projecto.

I.2. Instruções de execução

Correr o ficheiro executável *iarTris.jar* gerado pela execução do passo anterior.

I.3. Instruções de jogo

Irá surgir um ecrã que permite escolher se é o computador a jogar e sua heurística, ou um humano. É necessário escolher também o número de linhas com que o tabuleiro começa. Este ecrã irá surgir de cada vez que se reiniciar um jogo.

Para além dos botões e da interface em menu, as teclas fazem as seguintes operações:

W, ↑ : rodar a peça no sentido dos ponteiros do relógio

A, ←: deslocar a peça uma casa para a esquerda

D, →: deslocar a peça uma casa para a direita

S, ↓: deslocar a peça uma casa para baixo

ESPAÇO: deixar cair a peça no tabuleiro

R: reiniciar o jogo

P: pausar / recomeçar o jogo

ESC: sair do jogo

II. Exemplo de uma execução

Iniciando o programa, é-nos mostrado o menu de configuração do jogo (Figura 2), onde podemos escolher se é o computador que joga e que módulo de inteligência artificial utilizar, ou se é antes o utilizador a jogar. É também possível começar em modo *Challenge*, seleccionando linhas extra que começam já no tabuleiro, completamente ocupadas à excepção de um espaço (Figura. 3).

Figura 2 – ecrã de configuração

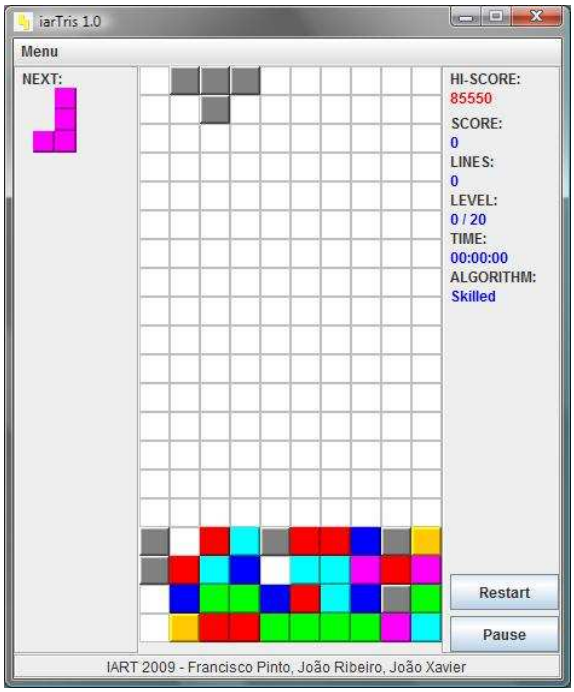

Figura 3 – início do jogo

Após algum tempo de execução, o tabuleiro poderá ter um aspecto semelhante ao observável na Figura 4.

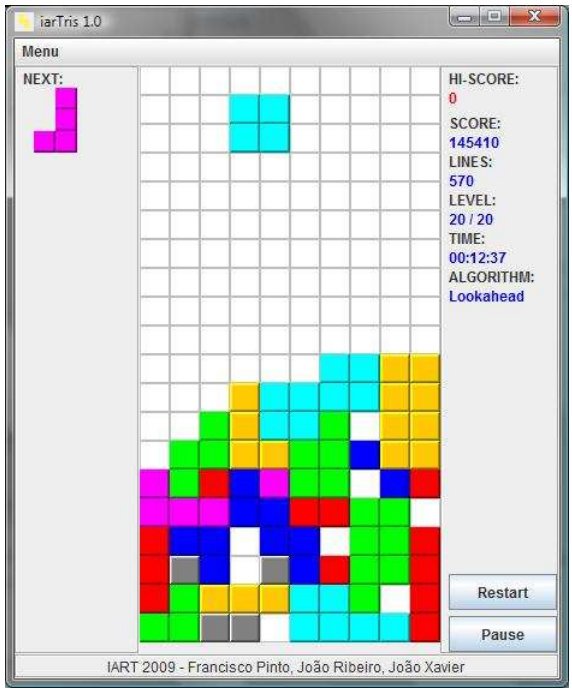

Figura 4 – jogo em execução

Terminando o jogo, é mostrada a mensagem *GAME OVER* no ecrã (Figura 5).

A qualquer momento da execução pode-se pressionar R para recomeçar o jogo (sendo mostrado o menu de configurações novamente), H para mostrar o ecrã de *highscores* (Figura 6), P para fazer *pause*, ou ESC para encerrar a aplicação. Todas estas funções podem também ser acedidas pelo menu da janela (Figura 7).

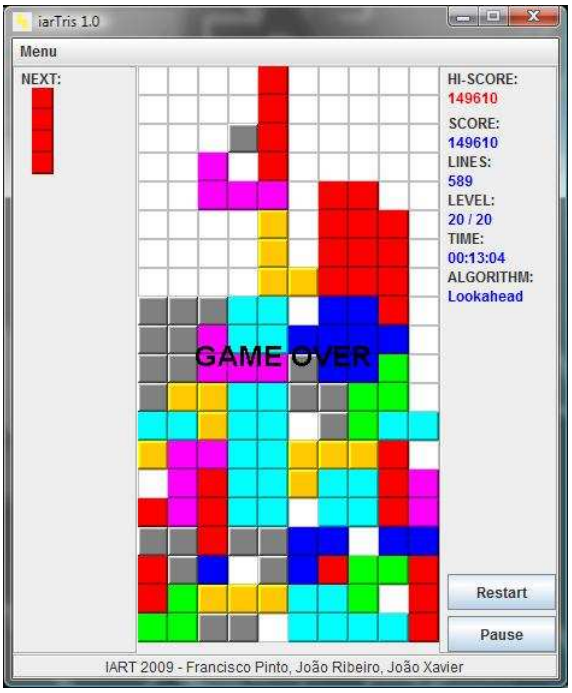

Figura 5 – jogo terminado

Figura 6 – ecrã de *highscores*

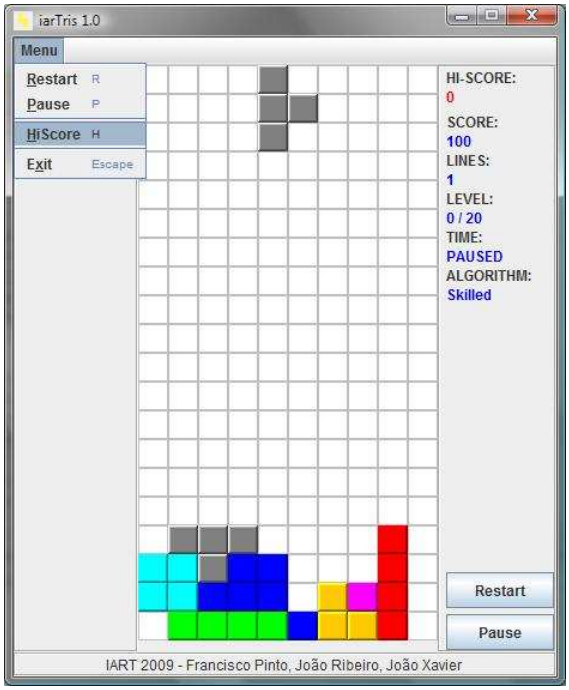

Figura 7 – menu da janela

III. Listagem do código

Ficheiro 1 – *IartrisMainFrame.java* (ainda que este ficheiro não seja da nossa autoria, foi alvo de modificações profundas, pelo que é incluído nesta secção)

```
package main.iartris;

import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.border.BevelBorder;
import javax.swing.border.EtchedBorder;

import res.ResClass;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.AdjustmentEvent;
import java.awt.event.AdjustmentListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowFocusListener;
import java.io.BufferedInputStream;
import java.util.ArrayList;
import java.util.LinkedList;

public class IartrisMainFrame extends JFrame
{
    private static final long serialVersionUID = 1L;
    private static final String NAME = "iarTris 1.0";
    private static final int CELL_H = 24;

    private Font font;
    private JPanel playPanel;
    private JLabel score;
    private JLabel lines;
    private JLabel time;
    private JLabel cpuNameLabel;
    private JLabel scrollerLabel;
    private JLabel levelLabel;
    private JLabel hiScoreLabel;

    private String cpuName;

    private Scrollbar scroller;

    private Movement movement;
    private JPanel[][] cells;
    private TetrisGrid tg;
    private JPanel[][] next;
    private int nextX;
    private int nextY;
    private int elapsedTime;
    private Figure f;
    private LinkedList<Figure> fNext;

    private int nSteps;
    private int lookup;
    private int challengeLines;

    private FigureFactory ff;
    private boolean isNewFigureDroped;
    private boolean isGameOver;
```

```
private boolean isPause;
private Color nextBg;
private TimeThread tt;
private KeyListener keyHandler;

// menu
private JMenuItem iartrisRestart;
private JMenuItem iartrisPause;
private JMenuItem iartrisHiScore;
private JMenuItem iartrisExit;

private JPanel hiScorePanel;
private JPanel gameSettingsMenu;

private class GridThread extends Thread
{
    private int count = 0;

    public void run() {
        try {
            while (true) {
                if (isGameOver || isPause) {
                    Thread.sleep(50);
                } else {
                      if(isNewFigureDroped) {
                                          isNewFigureDroped = false;
                                      count = 0;
                                      nextMove();
                                      continue;
                      }
                        else
                          Thread.sleep(50);        // drop speed

                      count += 50;

                      if (count + 50*tg.getLevel() >= 1100)
                      {
                          count = 0;
                          nextY++;
                          nextMove();
                      }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

private class TimeThread extends Thread {

    private int hours;
    private int min;
    private int sec;

    private int count;

    private void incSec() {
        elapsedTime++;
         sec++;
        if(sec == 60) {
            sec = 0;
            min++;
        }
        if(min == 60) {
            min = 0;
            hours++;
        }
```

```
        }

        private void resetTime() {
            hours = min = sec = 0;
        }

        public void run() {
            try {
                while (true) {
                    Thread.sleep(50);
                    if (isGameOver) {
                        Graphics g = playPanel.getGraphics();
                        Font font = new Font(g.getFont().getFontName(),
Font.BOLD, 24);
                        g.setFont(font);
                        g.drawString("GAME OVER", 47, 250);

                    } else if(isPause) {
                        time.setText("PAUSED");
                    } else if(count >= 1000) {
                        count = 0;
                        incSec();
                        time.setText(this.toString());
                    } else {
                        count+=50;
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        public String toString() {
            StringBuffer sb = new StringBuffer();
            if(hours < 10) {
                sb.append('0');
            }
            sb.append(hours);

            sb.append(':');

            if(min < 10) {
                sb.append('0');
            }
            sb.append(min);

            sb.append(':');

            if(sec < 10) {
                sb.append('0');
            }
            sb.append(sec);

            return sb.toString();
        }
    }

    public IartrisMainFrame() {
        super(NAME);

        nSteps = 3;
        fNext = new LinkedList<Figure>();

        setIconImage(loadImage("icon.jpg"));

        lookup = 1;
        challengeLines = 0;
        showAiSelector();
```

```
        keyHandler = new KeyAdapter(){

            public void keyPressed(KeyEvent e) {
                int code = e.getKeyCode();
                if(code == KeyEvent.VK_A || code == KeyEvent.VK_LEFT) {
                    moveLeft();
                } else if(code == KeyEvent.VK_D || code == KeyEvent.VK_RIGHT)
{
                    moveRight();
                } else if(code == KeyEvent.VK_S || code == KeyEvent.VK_DOWN) {
                    moveDown();
                } else if(code == KeyEvent.VK_W || code == KeyEvent.VK_UP) {
                    rotation();
                } else if(code == KeyEvent.VK_SPACE) {
                    moveDrop();
                }
            }
        };
        addKeyListener(keyHandler);

        font = new Font("Dialog", Font.PLAIN, 12);
        tg = new TetrisGrid();

        tg.addChallenge(challengeLines);


        ff = new FigureFactory();
        nextBg = new Color(238,238,238);

        initMenu();

        JPanel all = new JPanel(new BorderLayout());
        all.add(getLeftPanel(), BorderLayout.WEST);         // next piece
        all.add(getPlayPanel(), BorderLayout.CENTER);
        all.add(getMenuPanel(), BorderLayout.EAST);         // score, level,
time
        all.add(getCopyrightPanel(), BorderLayout.SOUTH);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.getContentPane().add(all, BorderLayout.CENTER);
        pack();
        this.setResizable(false);


        for (int i = 0; i != nSteps; i++)
             fNext.add(ff.getRandomFigure());

        dropNext();


        ActionListener taskPerformer = new ActionListener() {
            public void actionPerformed(ActionEvent evt) {

             if (movement == null)
                    return;

                int mov = movement.getInstruction();

                switch (mov)
                {
                       case Constants.LEFT:   moveLeft();  break;
                       case Constants.RIGHT:  moveRight(); break;
                       case Constants.ROTATE: rotation();  break;
                       case Constants.DROP:   moveDrop();  break;
                }
            }
        };
```

```
        new Timer(Constants.PC_PLAYING_TIME_MS, taskPerformer).start();




        GridThread gt = new GridThread();
        tt = new TimeThread();
        gt.start();
        tt.start();

        paintTG();

        addWindowFocusListener(new WindowFocusListener(){

            public void windowGainedFocus(WindowEvent arg0) {}

            public void windowLostFocus(WindowEvent arg0) {
                isPause = true;
            }
        });


        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation(screenSize.width / 2 - getWidth() / 2, screenSize.height /
2 - getHeight() / 2);
        setVisible(true);
    }

    private void initMenu() {

        MenuHandler mH = new MenuHandler();

        JMenuBar menu = new JMenuBar();
        setJMenuBar(menu);

        JMenu miartris = new JMenu();
        menu.add(miartris);
        miartris.setText("Menu");
        miartris.setMnemonic('J');
        {
            iartrisRestart = new JMenuItem("Restart");
            miartris.add(iartrisRestart);
            setKeyAcceleratorMenu(iartrisRestart, 'R',0);
            iartrisRestart.addActionListener(mH);
            iartrisRestart.setMnemonic('R');

            iartrisPause = new JMenuItem("Pause");
            miartris.add(iartrisPause);
            setKeyAcceleratorMenu(iartrisPause, 'P',0);
            iartrisPause.addActionListener(mH);
            iartrisPause.setMnemonic('P');

            miartris.addSeparator();

            iartrisHiScore = new JMenuItem("HiScore");
            miartris.add(iartrisHiScore);
            setKeyAcceleratorMenu(iartrisHiScore, 'H',0);
            iartrisHiScore.addActionListener(mH);
            iartrisHiScore.setMnemonic('H');

            miartris.addSeparator();

            iartrisExit = new JMenuItem("Exit");
            miartris.add(iartrisExit);
            setKeyAcceleratorMenu(iartrisExit, KeyEvent.VK_ESCAPE, 0);
            iartrisExit.addActionListener(mH);
            iartrisExit.setMnemonic('X');
```

```
        }
    }

    private void setKeyAcceleratorMenu(JMenuItem mi, int keyCode, int mask) {
        KeyStroke ks = KeyStroke.getKeyStroke(keyCode, mask);
        mi.setAccelerator(ks);
    }

    private JPanel getPlayPanel() {
        playPanel = new JPanel();
        playPanel.setLayout(new GridLayout(20,10));
        playPanel.setPreferredSize(new Dimension(10*CELL_H, 20*CELL_H));

        cells = new JPanel[20][10];
        for (int i = 0; i < 20; i++) {
            for (int j = 0; j < 10; j++) {
                cells[i][j] = new JPanel();
                cells[i][j].setBackground(Color.WHITE);

cells[i][j].setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
                playPanel.add(cells[i][j]);
            }
        }
        return playPanel;
    }

    private JPanel getLeftPanel(){
       JPanel r = new JPanel();
        BoxLayout rL = new BoxLayout(r,BoxLayout.Y_AXIS);
        r.setLayout(rL);
        r.setBorder(new EtchedBorder());
        Dimension ra = new Dimension(5, 0);
        next = new JPanel[4][4];
        JPanel nextP = new JPanel();
        nextP.setLayout(new GridLayout(4,4));
        Dimension d = new Dimension(4*18, 4*18);
        nextP.setMinimumSize(d);
        nextP.setPreferredSize(d);
        nextP.setMaximumSize(d);
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                next[i][j] = new JPanel();
                nextP.add(next[i][j]);
            }
        }

        JPanel jp = new JPanel();
        jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
        jp.add(Box.createRigidArea(ra));
        jp.add(new JLabel("NEXT:"));
        jp.add(Box.createHorizontalGlue());
        r.add(jp);
        r.add(nextP);

        r.add(Box.createRigidArea(new Dimension(100, 10)));

        return r;
    }



    private JPanel getMenuPanel() {
        JPanel r = new JPanel();
        BoxLayout rL = new BoxLayout(r,BoxLayout.Y_AXIS);
        r.setLayout(rL);
        r.setBorder(new EtchedBorder());
        Dimension ra = new Dimension(5, 0);
```

```
JPanel jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(new JLabel("HI-SCORE:"));
jp.add(Box.createHorizontalGlue());
r.add(jp);

hiScoreLabel = new JLabel(""+tg.hiScore[0].score);
hiScoreLabel.setForeground(Color.RED);


jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(hiScoreLabel);
jp.add(Box.createHorizontalGlue());
r.add(jp);

r.add(Box.createVerticalStrut(5));

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(new JLabel("SCORE:"));
jp.add(Box.createHorizontalGlue());
r.add(jp);

score = new JLabel("0");
score.setForeground(Color.BLUE);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(score);
jp.add(Box.createHorizontalGlue());
r.add(jp);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(new JLabel("LINES:"));
jp.add(Box.createHorizontalGlue());
r.add(jp);

lines = new JLabel("0");
lines.setForeground(Color.BLUE);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(lines);
jp.add(Box.createHorizontalGlue());
r.add(jp);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(new JLabel("LEVEL:"));
jp.add(Box.createHorizontalGlue());
r.add(jp);

levelLabel = new JLabel("1");
levelLabel.setForeground(Color.BLUE);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(levelLabel);
```

```
jp.add(Box.createHorizontalGlue());
r.add(jp);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(new JLabel("TIME:"));
jp.add(Box.createHorizontalGlue());
r.add(jp);

time = new JLabel("00:00:00");
time.setForeground(Color.BLUE);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(time);
jp.add(Box.createHorizontalGlue());
r.add(jp);

JLabel cpuLabel = new JLabel("\n0");
cpuLabel.setForeground(Color.GREEN);
jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(new JLabel("ALGORITHM:"));
jp.add(Box.createHorizontalGlue());
r.add(jp);

cpuNameLabel = new JLabel(cpuName);
cpuNameLabel.setForeground(Color.BLUE);

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
jp.add(cpuNameLabel);
jp.add(Box.createHorizontalGlue());
r.add(jp);

r.add(Box.createVerticalGlue());

//BUTTONS
r.add(Box.createRigidArea(new Dimension(0, 10)));

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
jp.add(Box.createRigidArea(ra));
JButton restartBut = new JButton("Restart");
restartBut.setToolTipText("Press 'R'");
restartBut.setFocusable(false);
restartBut.addKeyListener(keyHandler);
restartBut.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        restart();
    }
});
Dimension d = new Dimension(90, 30);
restartBut.setMinimumSize(d);
restartBut.setPreferredSize(d);
restartBut.setMaximumSize(d);
jp.add(restartBut);
jp.add(Box.createHorizontalGlue());
r.add(jp);

r.add(Box.createRigidArea(new Dimension(0, 5)));

jp = new JPanel();
jp.setLayout(new BoxLayout(jp, BoxLayout.LINE_AXIS));
```

```
        jp.add(Box.createRigidArea(ra));
        JButton pauseBut = new JButton("Pause");
        pauseBut.setToolTipText("Press 'P'");
        pauseBut.setFocusable(false);
        pauseBut.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                pause();
            }
        });
        pauseBut.setMinimumSize(d);
        pauseBut.setPreferredSize(d);
        pauseBut.setMaximumSize(d);
        jp.add(pauseBut);
        jp.add(Box.createHorizontalGlue());
        r.add(jp);

        return r;
    }


    private JPanel getCopyrightPanel() {
        JPanel r = new JPanel(new BorderLayout());
        BoxLayout rL = new BoxLayout(r,BoxLayout.LINE_AXIS);
        r.setLayout(rL);
        r.setBorder(new EtchedBorder());
        r.add(Box.createRigidArea(new Dimension(75,0)));

        JLabel jL = new JLabel("IART 2009 - Francisco Pinto, João Ribeiro,
João Xavier");
        jL.setFont(font);

        r.add(jL);
        r.setAlignmentX(CENTER_ALIGNMENT);

        return r;
    }

    static Image loadImage(String imageName) {
        try {
            Image im = ImageIO.read(new BufferedInputStream(
                    new
ResClass().getClass().getResourceAsStream(imageName)));
            return im;
        } catch (Exception e) {
            e.printStackTrace(System.out);
            return null;
        }
    }

    private synchronized void nextMove() {
        f.setOffset(nextX, nextY);

        if(tg.addFigure(f)) {
            dropNext();
            f.setOffset(nextX, nextY);

        } else {
            clearOldPosition();
        }
        paintNewPosition();

        if(isGameOver) {
            int tmp = tg.updateHiScore();
            if(tmp >= 0) {

                tg.saveHiScore(cpuName, tmp);

                if(tmp == 0)
```

```
                    hiScoreLabel.setText(""+tg.hiScore[0].score);
                }
            }
        }

        private void clearOldPosition() {
            for (int j = 0; j < 4; j++) {

cells[f.arrY[j]+f.offsetYLast][f.arrX[j]+f.offsetXLast].setBackground(Color.WH
ITE);

cells[f.arrY[j]+f.offsetYLast][f.arrX[j]+f.offsetXLast].setBorder(BorderFactor
y.createLineBorder(Color.LIGHT_GRAY));
            }
        }

        private void paintNewPosition() {
            for (int j = 0; j < 4; j++) {

cells[f.arrY[j]+f.offsetY][f.arrX[j]+f.offsetX].setBackground(f.getGolor());

cells[f.arrY[j]+f.offsetY][f.arrX[j]+f.offsetX].setBorder(BorderFactory.create
BevelBorder(BevelBorder.RAISED));
            }
        }

        private void paintTG() {
            int i = 0;
            Color c;
            for (int[] arr : tg.gLines) {
                for (int j = 0; j < arr.length; j++) {
                    if(arr[j]!= 0) {
                        switch (arr[j]) {
                        case Figure.I: c = Figure.COL_I; break;
                        case Figure.T: c = Figure.COL_T; break;
                        case Figure.O: c = Figure.COL_O; break;
                        case Figure.J: c = Figure.COL_J; break;
                        case Figure.L: c = Figure.COL_L; break;
                        case Figure.S: c = Figure.COL_S; break;
                        default: c = Figure.COL_Z; break;
                        }
                        cells[i][j].setBackground(c);

cells[i][j].setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
                    } else {
                        cells[i][j].setBackground(Color.WHITE);

cells[i][j].setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
                    }
                }
                i++;
            }
        }

        private void showNextFigures()
        {
           //for(int i=0; i!= nSteps; i++)
                 showNext(fNext.get(1));
        }


        private void showNext(Figure f) {
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    next[i][j].setBackground(nextBg);
                    next[i][j].setBorder(BorderFactory.createEmptyBorder());
                }
            }
```

```
        for (int j = 0; j < f.arrX.length; j++) {
            next[f.arrY[j]][f.arrX[j]].setBackground(f.getGolor());

next[f.arrY[j]][f.arrX[j]].setBorder(BorderFactory.createBevelBorder(BevelBord
er.RAISED));
        }
    }

    private void dropNext() {
        if(isGameOver) return;
        nextX = 4;
        nextY = 0;

        score.setText(""+tg.getScore());
        lines.setText(""+tg.getLines());
        levelLabel.setText(tg.getLevel()+" / 20");

        f = fNext.getFirst();

        fNext.addLast(ff.getRandomFigure());
        showNextFigures();


        movement = search(tg, 0);


        fNext.removeFirst();

        isGameOver = tg.isGameOver(f);
        isNewFigureDroped = true;
    }

    private void moveLeft() {
        if(isGameOver || isPause) return;
        if(nextX-1 >= 0) {
            if (tg.isNextMoveValid(f,f.offsetX-1,f.offsetY)) {
                nextX--;
                nextMove();
            }
        }
    }

    private void moveLeftDummy(Figure f, TetrisGrid temp)
    {
        if(isGameOver || isPause) return;

        if(nextX-1 >= 0)
        {
            if (temp.isNextMoveValid(f,f.offsetX-1,f.offsetY))
            {
                nextX--;
                f.setOffset(nextX, nextY);
            }
        }
    }

    private void moveRight() {
        if(isGameOver || isPause) return;
        if(f.getMaxRightOffset()+1 < 10) {
            if (tg.isNextMoveValid(f,f.offsetX+1,f.offsetY)) {
                nextX++;
                nextMove();
            }
        }
    }

    private void moveRightDummy(Figure f, TetrisGrid temp)
```

```
    {
        if(isGameOver || isPause) return;
        if(f.getMaxRightOffset()+1 < 10)
        {
            if (temp.isNextMoveValid(f,f.offsetX+1,f.offsetY))
            {
                nextX++;
                f.setOffset(nextX, nextY);
            }
        }
    }

    private synchronized void moveDown() {
        if(isGameOver || isPause) return;
        nextY++;
        nextMove();
    }

    private synchronized void moveDrop() {
        if(isGameOver || isPause) return;

        f.offsetYLast = f.offsetY;
        f.offsetXLast = f.offsetX;
        clearOldPosition();

        while(tg.isNextMoveValid(f, f.offsetX, f.offsetY)) {
            f.setOffset(f.offsetX, f.offsetY+1);
        }

        tg.addFigure(f);
        paintTG();
        dropNext();
        nextMove();
    }

    private synchronized void moveDropDummy(Figure f, TetrisGrid temp)
    {
        if(isGameOver || isPause) return;

        f.offsetYLast = f.offsetY;
        f.offsetXLast = f.offsetX;

        while(temp.isNextMoveValid(f, f.offsetX, f.offsetY))
            f.setOffset(f.offsetX, f.offsetY+1);

        temp.addFigure(f);
    }

    private synchronized void rotation() {
        if(isGameOver || isPause) return;
        for (int j = 0; j < f.arrX.length; j++) {

cells[f.arrY[j]+f.offsetY][f.arrX[j]+f.offsetX].setBackground(Color.WHITE);

cells[f.arrY[j]+f.offsetY][f.arrX[j]+f.offsetX].setBorder(BorderFactory.create
LineBorder(Color.LIGHT_GRAY));
        }
        f.rotationRight();
        if(!tg.isNextMoveValid(f,f.offsetX,f.offsetY)) {
            f.rotationLeft();
        }
        nextMove();
    }

    private synchronized void rotationDummy(Figure f, TetrisGrid temp) {
        if(isGameOver || isPause) return;

        f.rotationRight();
```

```
            if(!temp.isNextMoveValid(f,f.offsetX,f.offsetY)) {
                f.rotationLeft();
            }
        }

        private synchronized void pause() {
            isPause = !isPause;
        }

        private void restart() {
            for (int i = 0; i < 20; i++) {
                for (int j = 0; j < 10; j++) {
                    tg.gLines.get(i)[j] = 0;
                    cells[i][j].setBackground(Color.WHITE);

cells[i][j].setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
                }
            }

            paintTG();

            ff.resetCounts();
            isGameOver = false;

            fNext.clear();
            for (int i = 0; i != nSteps; i++)
                 fNext.add(ff.getRandomFigure());

            tt.resetTime();
            time.setText("00:00:00");
            tg.resetStats();

            lookup = 1;

            challengeLines = 0;
            showAiSelector();

            tg.addChallenge(challengeLines);
            paintTG();

            isPause = false;
            dropNext();
            nextMove();
        }

        private void showAiSelector()
        {
           cpuName = "Skilled";
           lookup = 1;

           if (cpuNameLabel == null)
                  cpuNameLabel = new JLabel(cpuName);
           else
                  cpuNameLabel.setText(cpuName);

           gameSettingsMenu = new JPanel(null);
           gameSettingsMenu.setLayout(new GridLayout(3, 0));


           JRadioButton greedyButton = new JRadioButton("Greedy", false);
           greedyButton.addActionListener(new RadioListener() );
           greedyButton.setActionCommand("Greedy");

           JRadioButton h0Button = new JRadioButton("Skilled", true);
           h0Button.addActionListener(new RadioListener() );
           h0Button.setActionCommand("Skilled");

           JRadioButton h1Button = new JRadioButton("Lookahead", false);
```

```
        h1Button.addActionListener(new RadioListener() );
        h1Button.setActionCommand("Lookahead");

        JRadioButton humanButton = new JRadioButton("Human", false);
        humanButton.addActionListener(new RadioListener() );
        humanButton.setActionCommand("Human");

        ButtonGroup bgroup = new ButtonGroup();
        bgroup.add(greedyButton);
        bgroup.add(h0Button);
        bgroup.add(h1Button);
        bgroup.add(humanButton);

        JPanel radioPanel = new JPanel();
        radioPanel.setLayout(new GridLayout(2, 1));
        radioPanel.add(h0Button);
        radioPanel.add(greedyButton);
        radioPanel.add(h1Button);
        radioPanel.add(humanButton);

        int min=0, max=11;
        scroller = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, min, max);
        scroller.addAdjustmentListener(new ScrollerListener() );

        scrollerLabel = new JLabel("Challenge: \n starting with 0 extra
line(s)");


        gameSettingsMenu.add(radioPanel);
        gameSettingsMenu.add(scrollerLabel);
        gameSettingsMenu.add(scroller);


        JOptionPane.showMessageDialog(this,gameSettingsMenu,"Game Settings",
                  JOptionPane.PLAIN_MESSAGE );
    }

    private class ScrollerListener implements AdjustmentListener
    {
             public void adjustmentValueChanged(AdjustmentEvent arg0) {
                    challengeLines = arg0.getValue();
                    scrollerLabel.setText("Challenge: \n starting with "
                                 + arg0.getValue() + " extra line(s)"  );
             }
    }

    private class RadioListener implements ActionListener
    {
       public void actionPerformed(ActionEvent ae) {
             cpuName = ae.getActionCommand();
             cpuNameLabel.setText(cpuName);
             if(cpuName == "Human")
              lookup = 0;
             else if (cpuName == "Skilled" || cpuName == "Greedy")
              lookup = 1;
             else if (cpuName == "Lookahead")
              lookup = 2;
        }
    }

    private void showHiScore() {
        setHiScorePanel();

        JOptionPane.showMessageDialog(this,hiScorePanel,"HI SCORE",
                JOptionPane.PLAIN_MESSAGE,
                new ImageIcon(loadImage("splash32.png")));

        hiScorePanel = null;
```

```
}

private void setHiScorePanel() {
    hiScorePanel = new JPanel(new BorderLayout());

    String[] colNames = {"Place", "Points", "Lines", "Name"};
    String[][] data = new String[tg.hiScore.length+1][colNames.length];
    data[0] = colNames;
    for (int i = 0; i < tg.hiScore.length; i++) {
        data[i+1] = new String[colNames.length];
        data[i+1][0] = (i+1)+".";
        data[i+1][1] = (""+tg.hiScore[i].score);
        data[i+1][2] = (""+tg.hiScore[i].lines);
        data[i+1][3] = (""+tg.hiScore[i].name);
    }

    JTable table = new JTable(data, colNames);
    table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    table.setBackground(new Color(230,255,255));
    table.setEnabled(false);

    hiScorePanel.add(table,BorderLayout.CENTER);
}

private class MenuHandler implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        try {
            JMenuItem tmp = (JMenuItem) e.getSource();
            if (tmp == iartrisRestart) {
                restart();
            } else if (tmp == iartrisPause) {
                pause();
            } else if (tmp == iartrisHiScore) {
                showHiScore();
            } else if (tmp == iartrisExit) {
                System.exit(0);
            }
        } catch (Exception exc) {
            exc.printStackTrace(System.out);
        }
    }
}


Movement search(TetrisGrid old, int level)
{
   if (level == lookup)
         return null;

   Figure f = fNext.get(level);
   TetrisGrid temp = new TetrisGrid(old);

   ArrayList<Movement> movementTemp = new ArrayList<Movement>();

   Movement mov;

   int maxRotation;

   switch (f.getGridVal())
   {
         case Figure.I:
         case Figure.S:
         case Figure.Z:
               maxRotation = 2;
               break;

         case Figure.O:
```

```
                        maxRotation = 1;

                default:
                        maxRotation = 4;
        }

        for (int xM = -4; xM <= 5; xM++)
        {
                if (xM <= 0)
                {
                        for (int i = 0; i != maxRotation; i++)
                        {
                                for (int h = 0; h != xM; h--)
                                        moveLeftDummy(f, temp);

                                for (int j = 0; j != i; j++)
                                        rotationDummy(f, temp);

                                moveDropDummy(f, temp);

                                mov = new Movement(-xM, 0, i, search(temp, level +
1));

                                mov.setScore(cpuName, temp.countHoles(),
temp.getHeight(), temp.averageHeight(),
                                                temp.deltaHeight(), temp.variance(),
temp.numPiecesOverload(),
                                                temp.weightedHeight());

                                movementTemp.add(mov);

                                temp = new TetrisGrid(old);
                                nextX = 4;
                          nextY = 0;
                                f.resetRotation();
                                f.setOffset(4, 0);
                        }
                }
                        else
                        {
                        for (int i = 0; i != maxRotation; i++)
                        {
                                for (int h = 1; h <= xM; h++)
                                        moveRightDummy(f, temp);

                                for (int j = 0; j != i; j++)
                                        rotationDummy(f, temp);

                                mov = new Movement(0, xM, i, search(temp, level +
1));

                                moveDropDummy(f, temp);
                                mov.setScore(cpuName, temp.countHoles(),
temp.getHeight(), temp.averageHeight(),
                                                temp.deltaHeight(), temp.variance(),
temp.numPiecesOverload(),
                                                temp.weightedHeight());

                                movementTemp.add(mov);

                                temp = new TetrisGrid(old);
                                nextX = 4;
                          nextY = 0;
                                f.resetRotation();
                                f.setOffset(4, 0);
                        }
                }
                }
```

```
		for (int i = 1; i != maxRotation; i++)
		{
			for (int h = 1; h <= 5; h++)
			{
				for (int j = 0; j != i; j++)
					rotationDummy(f, temp);

				for (int k = 0; k != h; k++)
					moveRightDummy(f, temp);

				mov = new Movement(0, h, i, search(temp, level + 1));

					moveDropDummy(f, temp);
					mov.setScore(cpuName, temp.countHoles(),
temp.getHeight(), temp.averageHeight(),
						temp.deltaHeight(), temp.variance(),
temp.numPiecesOverload(),
						temp.weightedHeight());

				movementTemp.add(mov);

				temp = new TetrisGrid(old);
				nextX = 4;
			 nextY = 0;
					f.resetRotation();
					f.setOffset(4, 0);
			}
		}

		return getBest(movementTemp);
	}

		Movement getBest(ArrayList<Movement> movementTemp)
		{
			Movement best = null;
			float bestScore = Float.MAX_VALUE;

			for (Movement m: movementTemp)
				if (m.getScore() < bestScore)
				{
					bestScore = m.getScore();
					best = m;
				}

			return best;
		}
}
```

Ficheiro 2 – *Constants.java*

```
package main.iartris;

public final class Constants
{
		// despair line
		public static final int DESPAIR_LINE = 10;

		// skilled (height <= DESPAIR_LINE)
		public static final int HOLES_VALUE = 50;
		public static final int MAX_HEIGHT_VALUE = 25;
		public static final int AVERAGE_HEIGHT_VALUE = 3;
		public static final int DELTA_HEIGHT_VALUE = 5;
		public static final int VARIANCE_HEIGHT_VALUE = 5;
		public static final int PIECE_OVERLOAD_VALUE = 1;
```

```
	public static final int WEIGHTED_HEIGHT_VALUE = 1;

	// greedy (height <= DESPAIR_LINE)
	public static final int GREEDY_HOLES_VALUE = 60;
	public static final int GREEDY_MAX_HEIGHT_VALUE = 30;

	// skilled (height > DESPAIR_LINE)
	public static final int HOLES_DESPAIR_VALUE = 35;
	public static final int MAX_HEIGHT_DESPAIR_VALUE = 70;
	public static final int AVERAGE_HEIGHT_DESPAIR_VALUE = 2;
	public static final int DELTA_HEIGHT_DESPAIR_VALUE = 10;
	public static final int VARIANCE_HEIGHT_DESPAIR_VALUE = 5;
	public static final int PIECE_OVERLOAD_DESPAIR_VALUE = 1;
	public static final int WEIGHTED_HEIGHT_DESPAIR_VALUE = 1;

	// greedy (height > DESPAIR_LINE)
	public static final int GREEDY_HOLES_DESPAIR_VALUE = 20;
	public static final int GREEDY_MAX_HEIGHT_DESPAIR_VALUE = 80;

	// instructions
	public static final int LEFT = 0;
	public static final int RIGHT = 1;
	public static final int ROTATE = 2;
	public static final int DROP = 3;

	// pc playing time in miliseconds
	public static final int PC_PLAYING_TIME_MS = 100;

	// max number of highscores
	public static final int HIGHSCORE_N = 5;


}
```

Ficheiro 3 – *Movement.java*

```
package main.iartris;

public class Movement implements Comparable<Movement>
{
	private int leftM;
	private int rightM;
	private int rotations;
	private float score;

	private Movement child;

	public Movement(int leftM, int rightM, int rotations, Movement
movement)
	{
		this.leftM = leftM;
		this.rightM = rightM;
		this.rotations = rotations;
		this.child = movement;
	}

	public float getScore()
	{
		return score;
	}
```

```
	public void setScore(String cpuName, int holes, int height,
float average,
			float delta, float variance, int overload, float
weight)
	{
		if (cpuName == "Greedy")
		{
			if (height > Constants.DESPAIR_LINE)
				score =
					height  *
Constants.GREEDY_MAX_HEIGHT_DESPAIR_VALUE +
					holes *
Constants.GREEDY_HOLES_DESPAIR_VALUE;
			else
				score =
					height  *
Constants.GREEDY_MAX_HEIGHT_VALUE +
					holes * Constants.GREEDY_HOLES_VALUE;

		}
		else
		{
			if (height > Constants.DESPAIR_LINE)
				score =
					height   *
Constants.MAX_HEIGHT_DESPAIR_VALUE +
					holes    * Constants.HOLES_DESPAIR_VALUE
+
					average  *
Constants.AVERAGE_HEIGHT_DESPAIR_VALUE +
					delta    *
Constants.DELTA_HEIGHT_DESPAIR_VALUE +
					variance *
Constants.VARIANCE_HEIGHT_DESPAIR_VALUE +
					overload *
Constants.PIECE_OVERLOAD_DESPAIR_VALUE +
					weight   *
Constants.WEIGHTED_HEIGHT_DESPAIR_VALUE;
			else
				score =
					height   * Constants.MAX_HEIGHT_VALUE +
					holes    * Constants.HOLES_VALUE +
					average  * Constants.AVERAGE_HEIGHT_VALUE
+
					delta    * Constants.DELTA_HEIGHT_VALUE +
					variance *
Constants.VARIANCE_HEIGHT_VALUE +
					overload * Constants.PIECE_OVERLOAD_VALUE
+
					weight   *
Constants.WEIGHTED_HEIGHT_VALUE;
		}

		if (child != null)
			score += child.score;
	}

	public int getInstruction()
	{
		if (rotations != 0)
		{
```

```
                rotations--;
                return Constants.ROTATE;
            }

            if (leftM != 0)
            {
                leftM--;
                return Constants.LEFT;
            }

            if (rightM != 0)
            {
                rightM--;
                return Constants.RIGHT;
            }

            return Constants.DROP;
        }

        public String toString()
        {
            String retval = "\n";
            retval += "Left: " + leftM + "\n";
            retval += "Right: " + rightM + "\n";
            retval += "Rotation: " + rotations + "\n";
            retval += "Score: " + score + "\n";

            return retval;
        }

        public int compareTo(Movement m)
        {
            if (this.score < m.score)
                return -1;
            else if (this.score == m.score)
                return 0;
            else
                return 1;
        }
}
```

IV. Nome dos elementos do grupo

Francisco Hernani Teixeira Pinto - 060509055

João Cristóvão Afonso Sampaio Xavier - 060509116

João Pedro Matos Ribeiro - 060509019