Léo DESMONTS
1B1
IUT Vannes – INFORMATIQUE - 2020

Projet de programmation : Zen l'Initié

Cahier de conception

Table des matières

1) Diagramme de classe : analyse

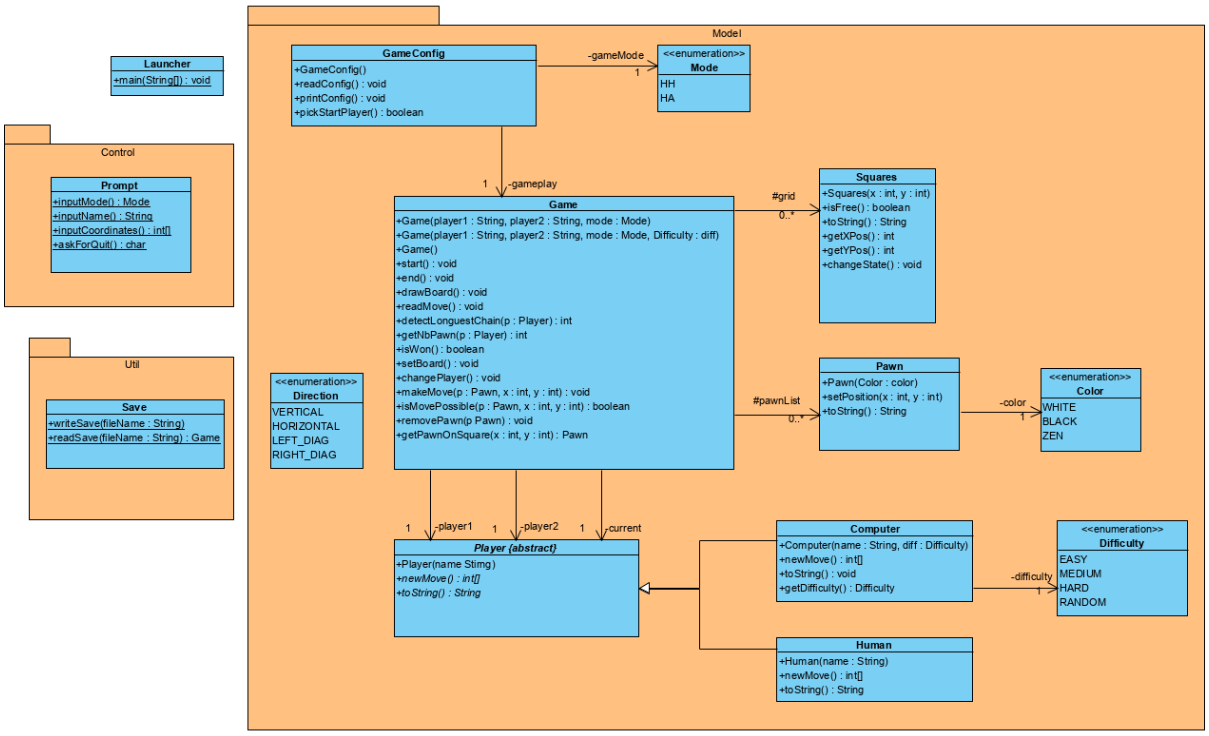
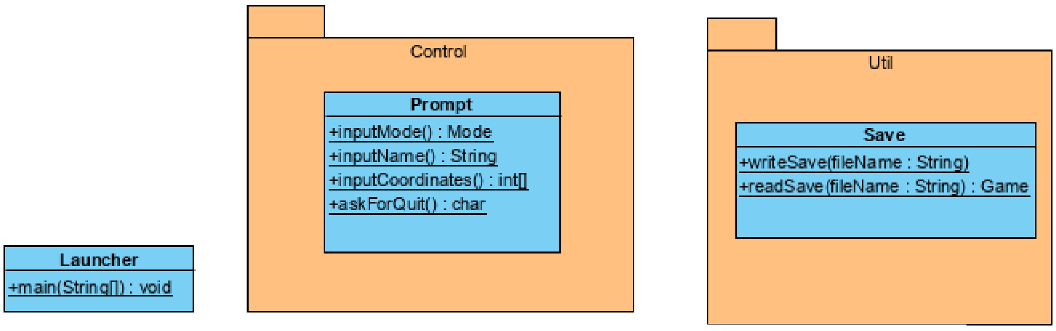
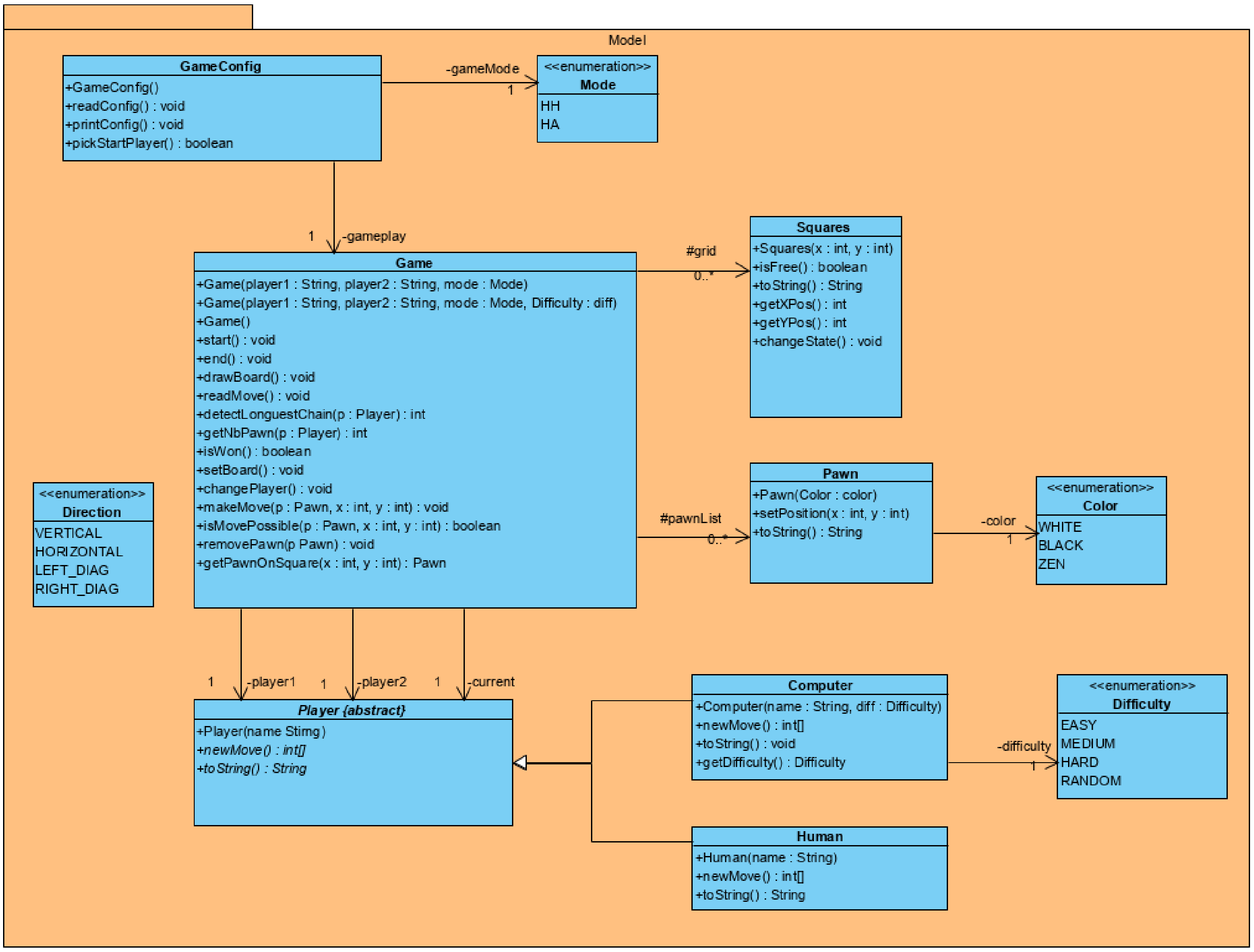


Diagramme d'analyse complet



Launcher, Package Control, Package Util



Package Model

Ce diagramme de classe, dans sa version d’analyse, permet de comprendre rapidement le fonctionnement général de l’application future.

2) Diagramme de classe : conception

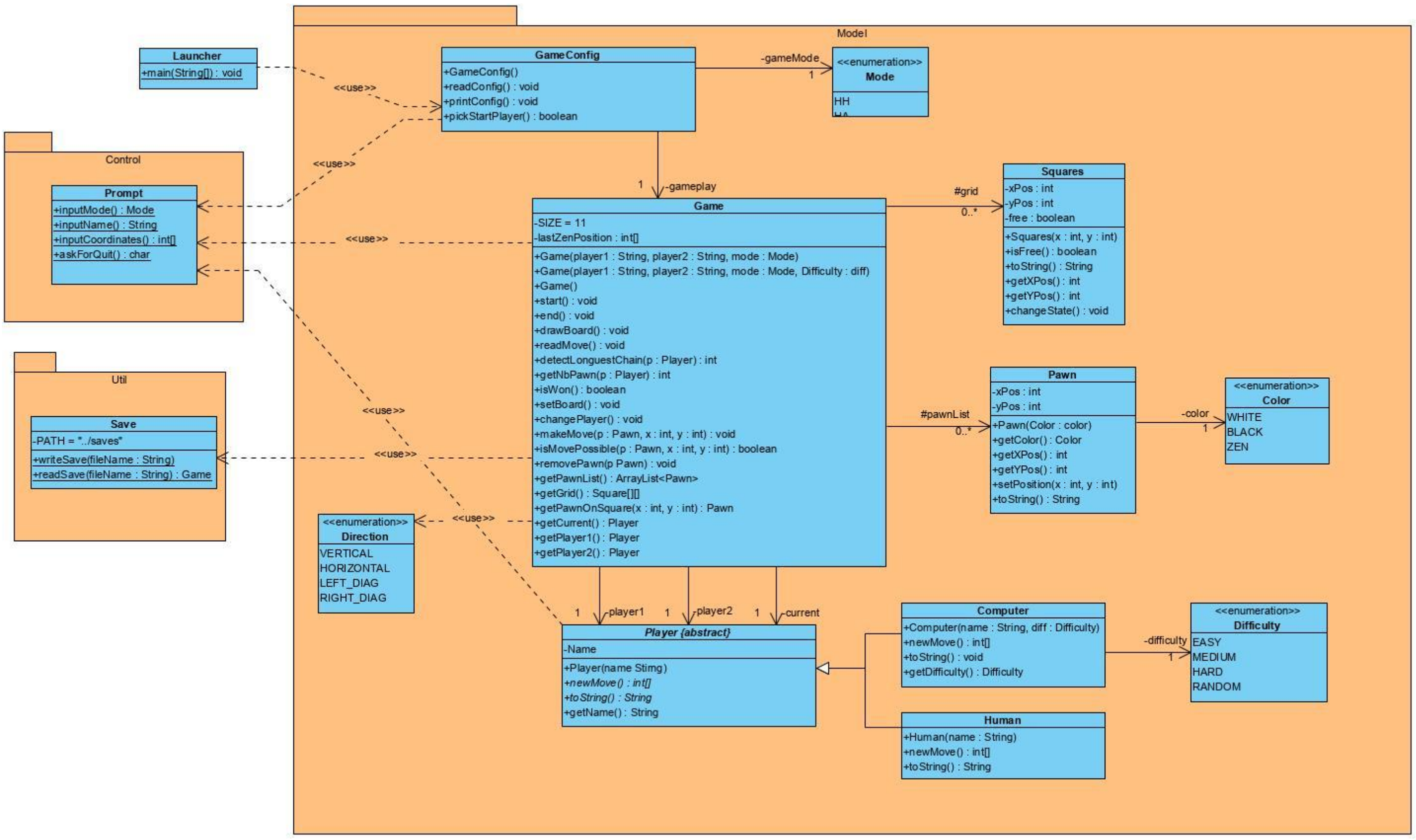
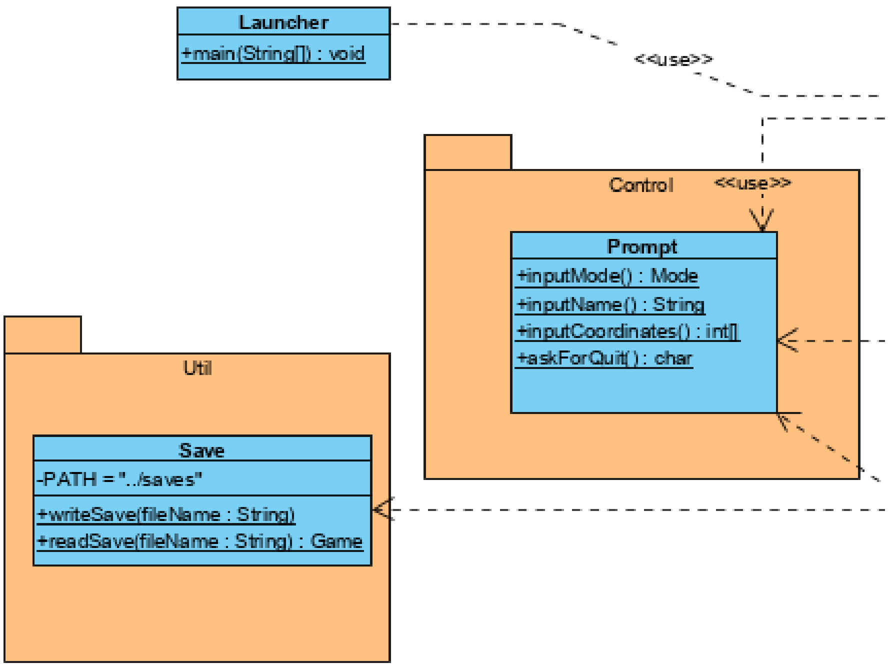
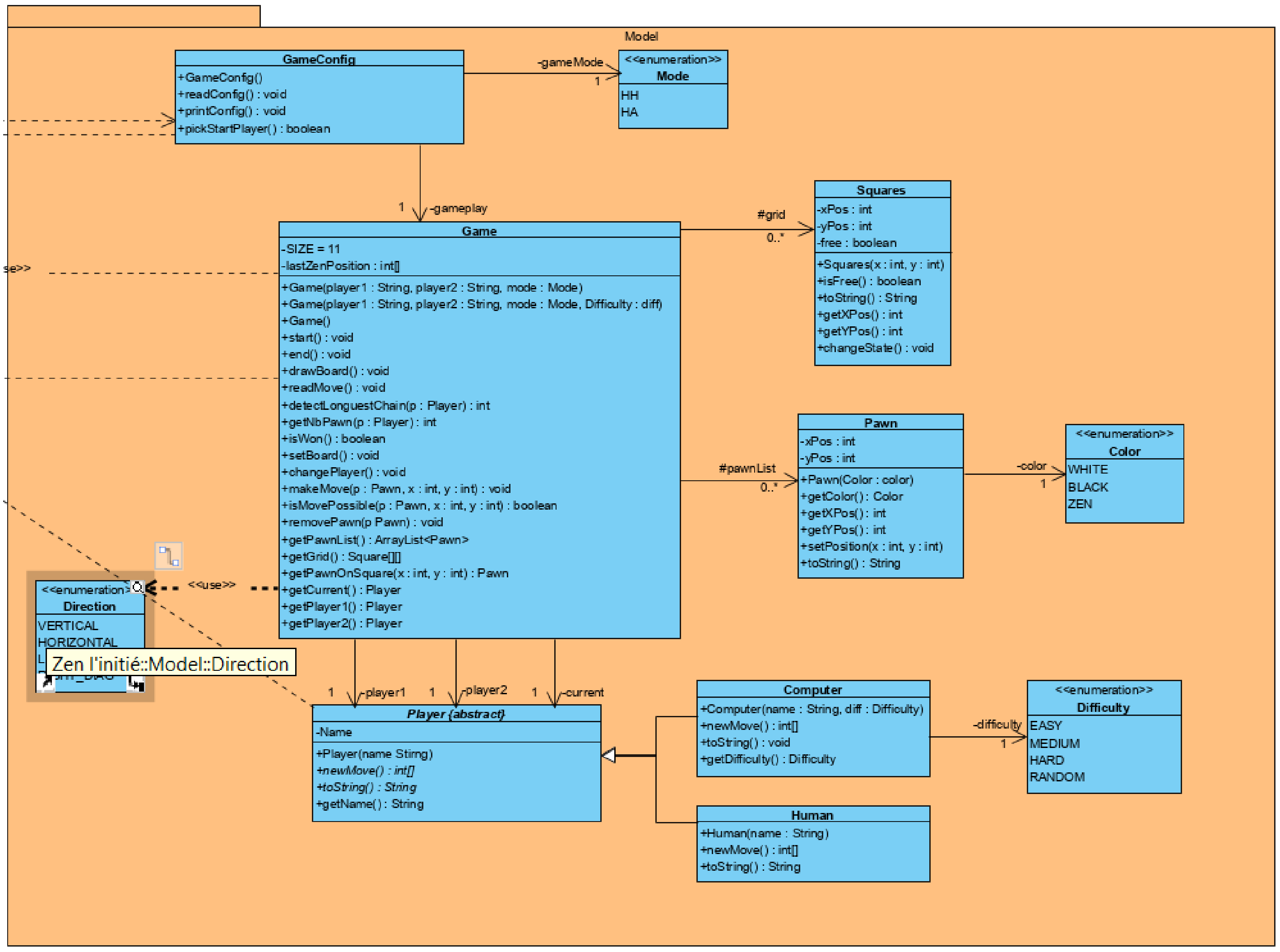


Diagramme de conception complet



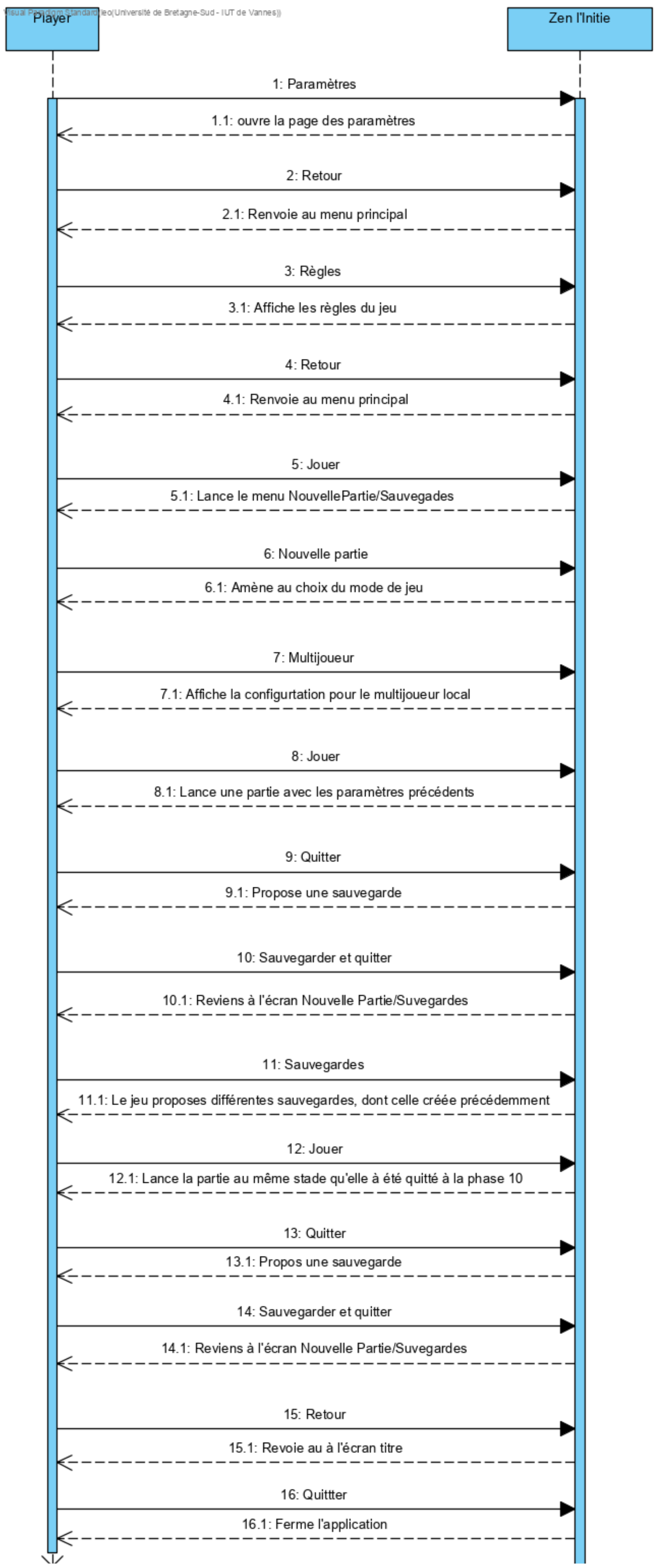
Launcher, Package Control, Package Util

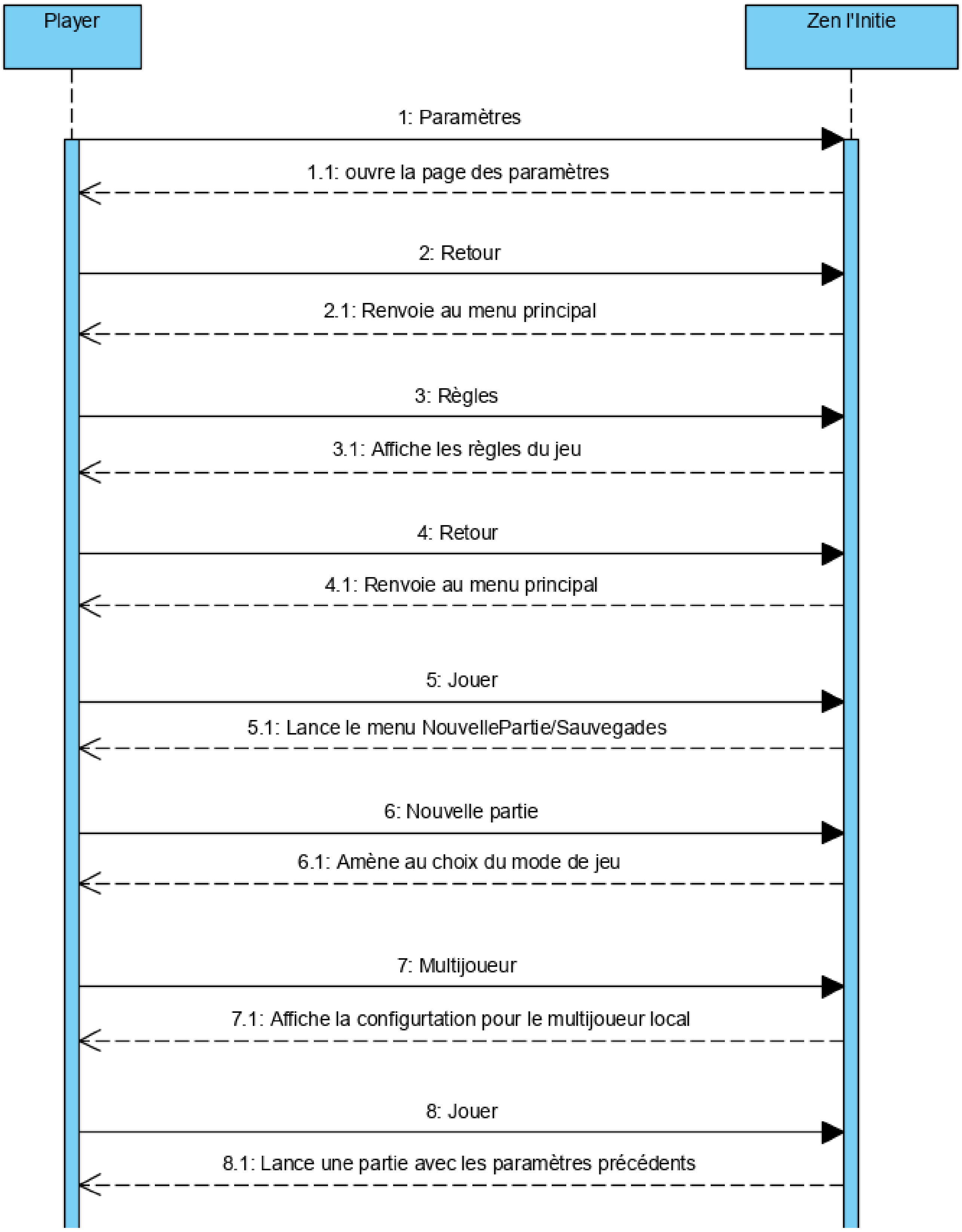


Package Model

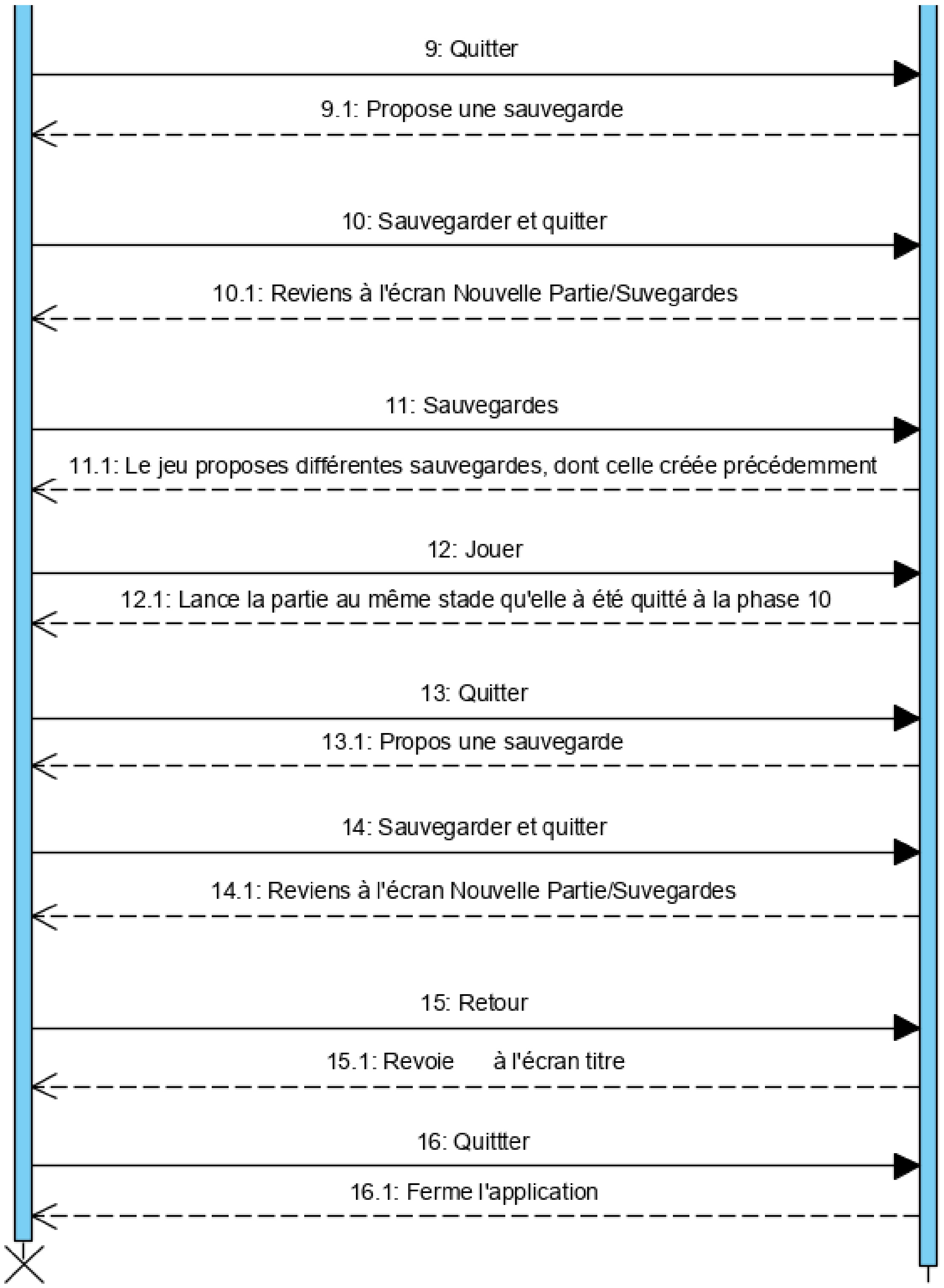
Ce diagramme de classe, dans sa version de conception, permet de comprendre le fonctionnement précis de l'application future. On y voit dessiné tous les attributs, méthodes et dépendances.

3) Diagramme de séquence boite noir


Player
Zen l'Initie
1: Paramètres
1.1: ouvre la page des paramètres
2: Retour
2.1: Renvoie au menu principal
3: Règles
3.1: Affiche les règles du jeu
4: Retour
4.1: Renvoie au menu principal
5: Jouer
5.1: Lance le menu NouvellePartie/Sauvegades
6: Nouvelle partie
6.1: Amène au choix du mode de jeu
7: Multijoueur
7.1: Affiche la configurtation pour le multijoueur local
8: Jouer
8.1: Lance une partie avec les paramètres précédents
9: Quitter
9.1: Propose une sauvegarde
10: Sauvegarder et quitter
10.1: Reviens à l'écran Nouvelle Partie/Suvegardes
11: Sauvegardes
11.1: Le jeu proposes différentes sauvegardes, dont celle créée précédemment
12: Jouer
12.1: Lance la partie au même stade qu'elle à été quitté à la phase 10
13: Quitter
13.1: Propos une sauvegarde
14: Sauvegarder et quitter
14.1: Reviens à l'écran Nouvelle Partie/Suvegardes
15: Retour
15.1: Revoie au à l'écran titre
16: Quittter
16.1: Ferme l'application
Séquence complète


Player
Zen l'Initie
1: Paramètres
1.1: ouvre la page des paramètres
2: Retour
2.1: Renvoie au menu principal
3: Règles
3.1: Affiche les règles du jeu
4: Retour
4.1: Renvoie au menu principal
5: Jouer
5.1: Lance le menu NouvellePartie/Sauvegades
6: Nouvelle partie
6.1: Amène au choix du mode de jeu
7: Multijoueur
7.1: Affiche la configurtation pour le multijoueur local
8: Jouer
8.1: Lance une partie avec les paramètres précédents

Séquence partie 1


9: Quitter
9.1: Propose une sauvegarde
10: Sauvegarder et quitter
10.1: Reviens à l'écran Nouvelle Partie/Suvegardes
11: Sauvegardes
11.1: Le jeu proposes différentes sauvegardes, dont celle créée précédemment
12: Jouer
12.1: Lance la partie au même stade qu'elle à été quitté à la phase 10
13: Quitter
13.1: Propos une sauvegarde
14: Sauvegarder et quitter
14.1: Reviens à l'écran Nouvelle Partie/Suvegardes
15: Retour
15.1: Revoie à l'écran titre
16: Quittter
16.1: Ferme l'application

Séquence partie 2

Ce diagramme de séquence présente un exemple de test effectué en boite noir. C'est-à-dire que l'utilisateur ne connait pas le fonctionnement interne de l'application. Il doit donc s'assurer qu'il comprend son utilisation, et qu'aucune fonctionnalité ne présente de bug.

4) Spécification des fichiers

Etant donné la structure complexe des objets à sauvegarder (des instances de la classe Game), on se propose d'utiliser la sérialisation. Ce procédé va nous permettre de sauvegarder l'état de l'objet à un instant T, et de le recharger à l'identique lors d'une autre partie (même après redémarrage de la machine). Il faudra pour cela que la classe Game implémente l'interface Serializable (ainsi que toutes les classes dont Game possède une instance).

On utilisera une classe externe (dans le package util), car il est fortement déconseillé d'utiliser une classe que peut effectuer une sérialisation d'elle-même. Cette classe possèdera deux méthodes, une méthode save (de sauvegarde) et une méthode load (pour charger une partie).

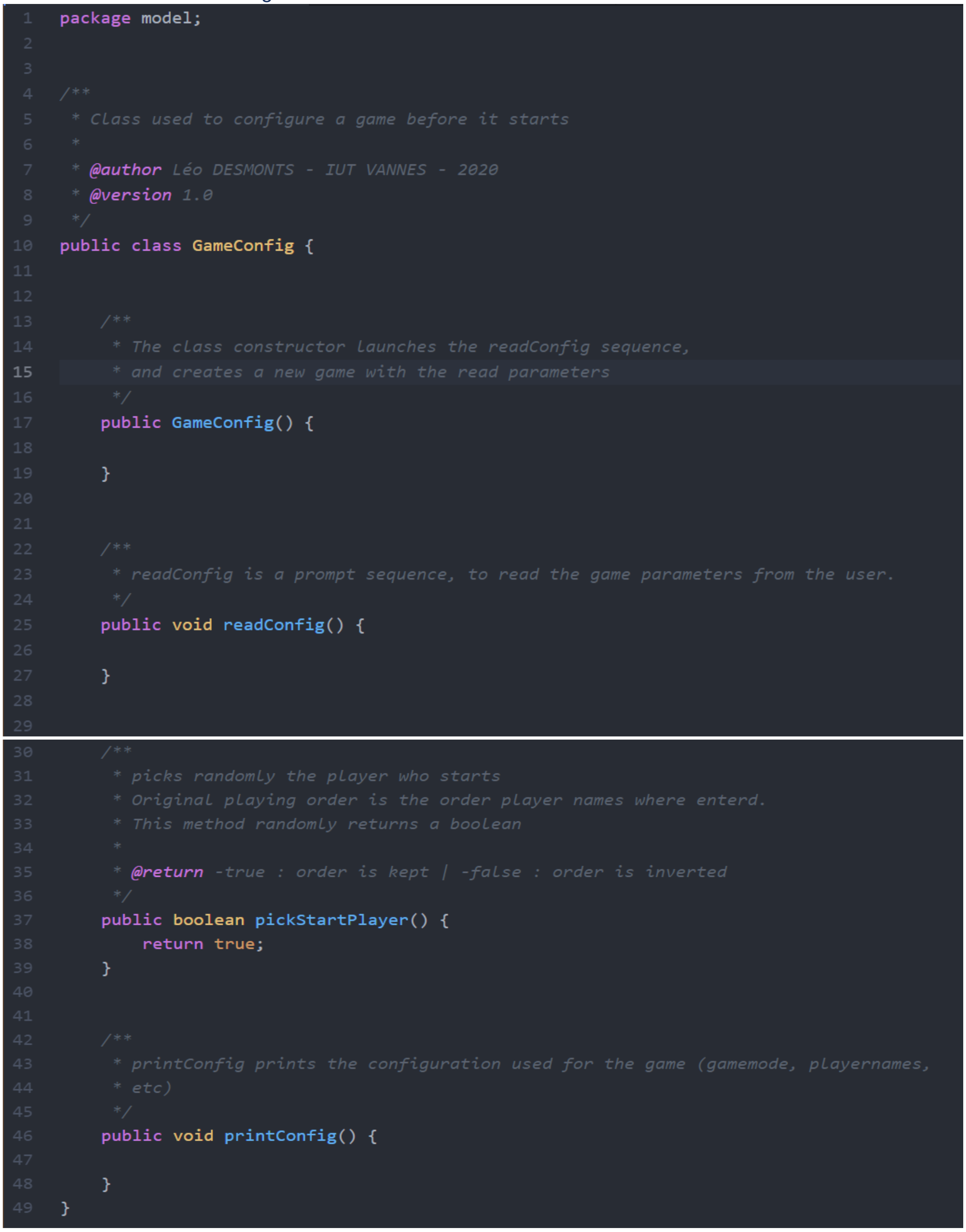
Lors de la configuration dans gameConfig, il sera demandé au joueur s'il désire créer une nouvelle partie (auquel cas il devra entrer une série de paramètres), ou bien charger une partie sauvegarder (auquel cas on lui demander de choisir le fichier). Le fichier sera ensuite utilisé pour lire l'objet, et le réinsérer dans le jeu.

5) Squelette du projet

Voici les squelettes des classes.

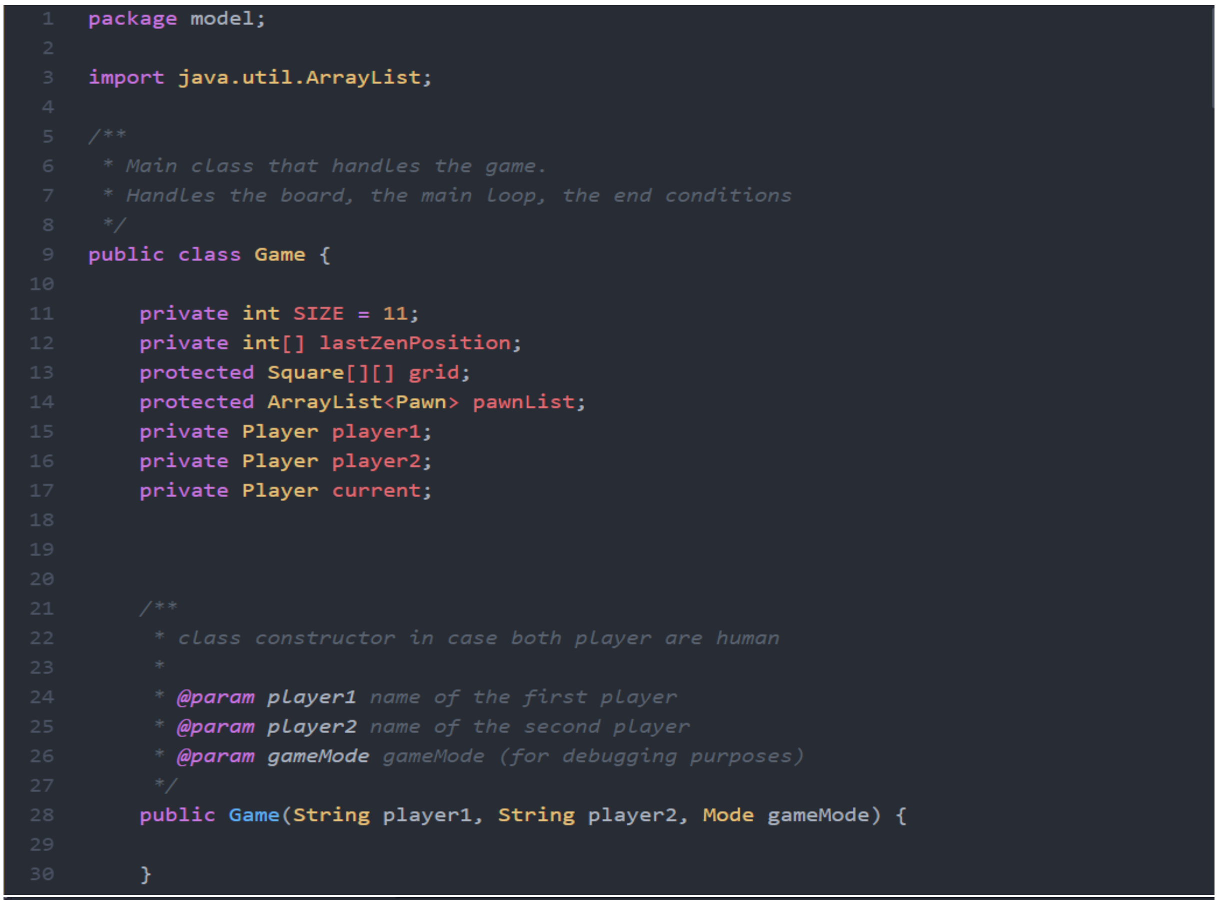
a) Package model

1. GameConfig

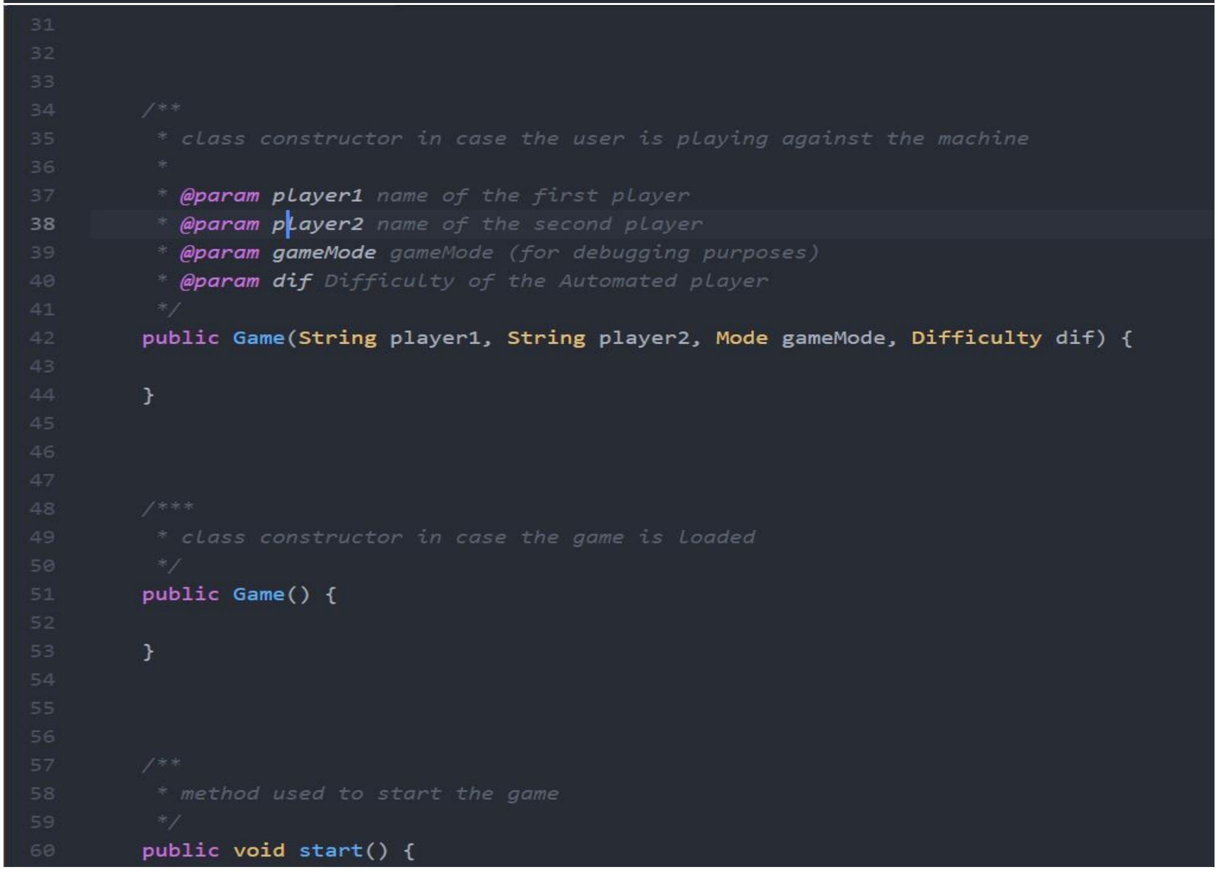


```java
package model;


/**
 * Class used to configure a game before it starts
 *
 * @author Léo DESMONTS - IUT VANNES - 2020
 * @version 1.0
 */
public class GameConfig {


    /**
     * The class constructor launches the readConfig sequence,
     * and creates a new game with the read parameters
     */
    public GameConfig() {

    }


    /**
     * readConfig is a prompt sequence, to read the game parameters from the user.
     */
    public void readConfig() {

    }


    /**
     * picks randomly the player who starts
     * Original playing order is the order player names where enterd.
     * This method randomly returns a boolean
     *
     * @return -true : order is kept | -false : order is inverted
     */
    public boolean pickStartPlayer() {
        return true;
    }


    /**
     * printConfig prints the configuration used for the game (gamemode, playernames,
     * etc)
     */
    public void printConfig() {

    }
}
```

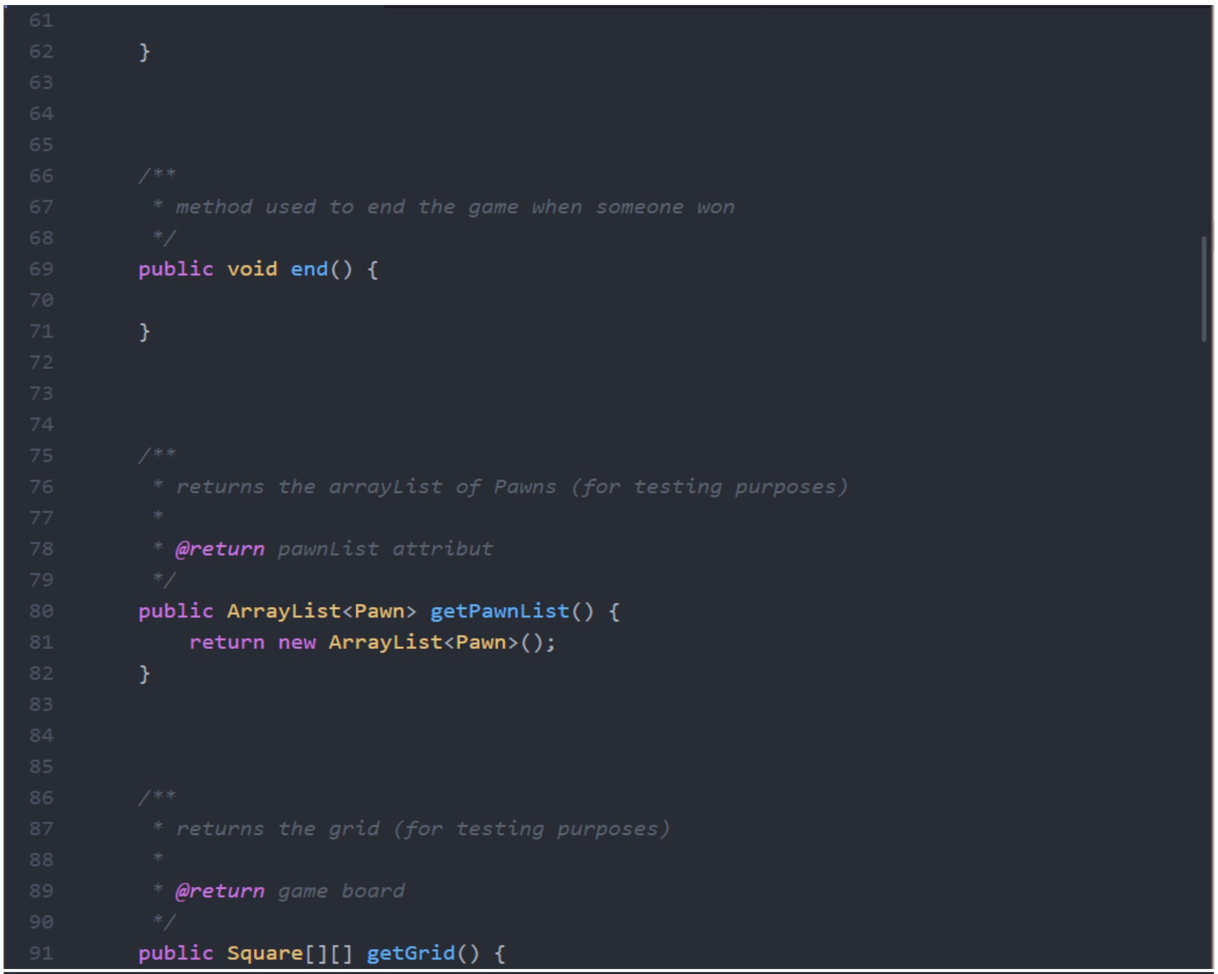
2. Game



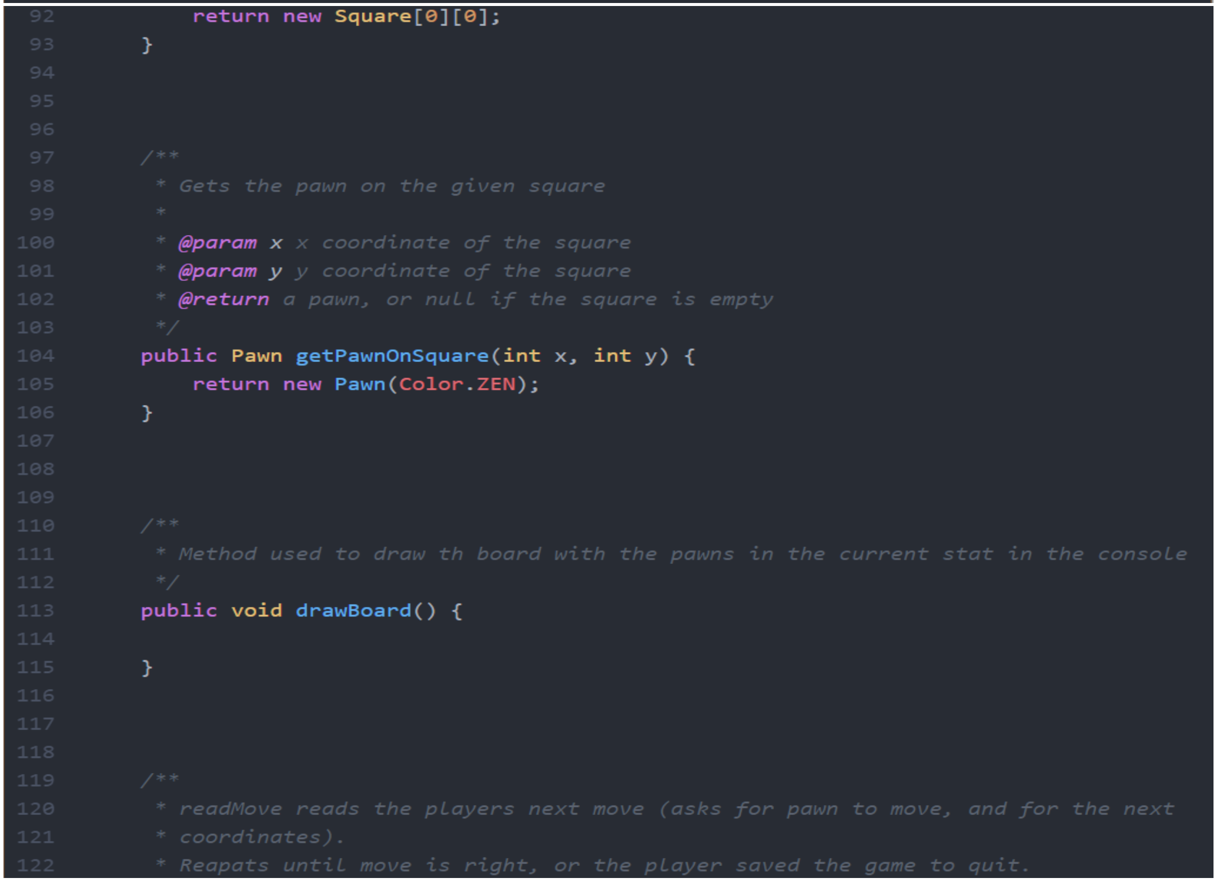
```java
package model;

import java.util.ArrayList;

/**
 * Main class that handles the game.
 * Handles the board, the main loop, the end conditions
 */
public class Game {

    private int SIZE = 11;
    private int[] lastZenPosition;
    protected Square[][] grid;
    protected ArrayList<Pawn> pawnList;
    private Player player1;
    private Player player2;
    private Player current;



    /**
     * class constructor in case both player are human
     *
     * @param player1 name of the first player
     * @param player2 name of the second player
     * @param gameMode gameMode (for debugging purposes)
     */
    public Game(String player1, String player2, Mode gameMode) {

    }
```



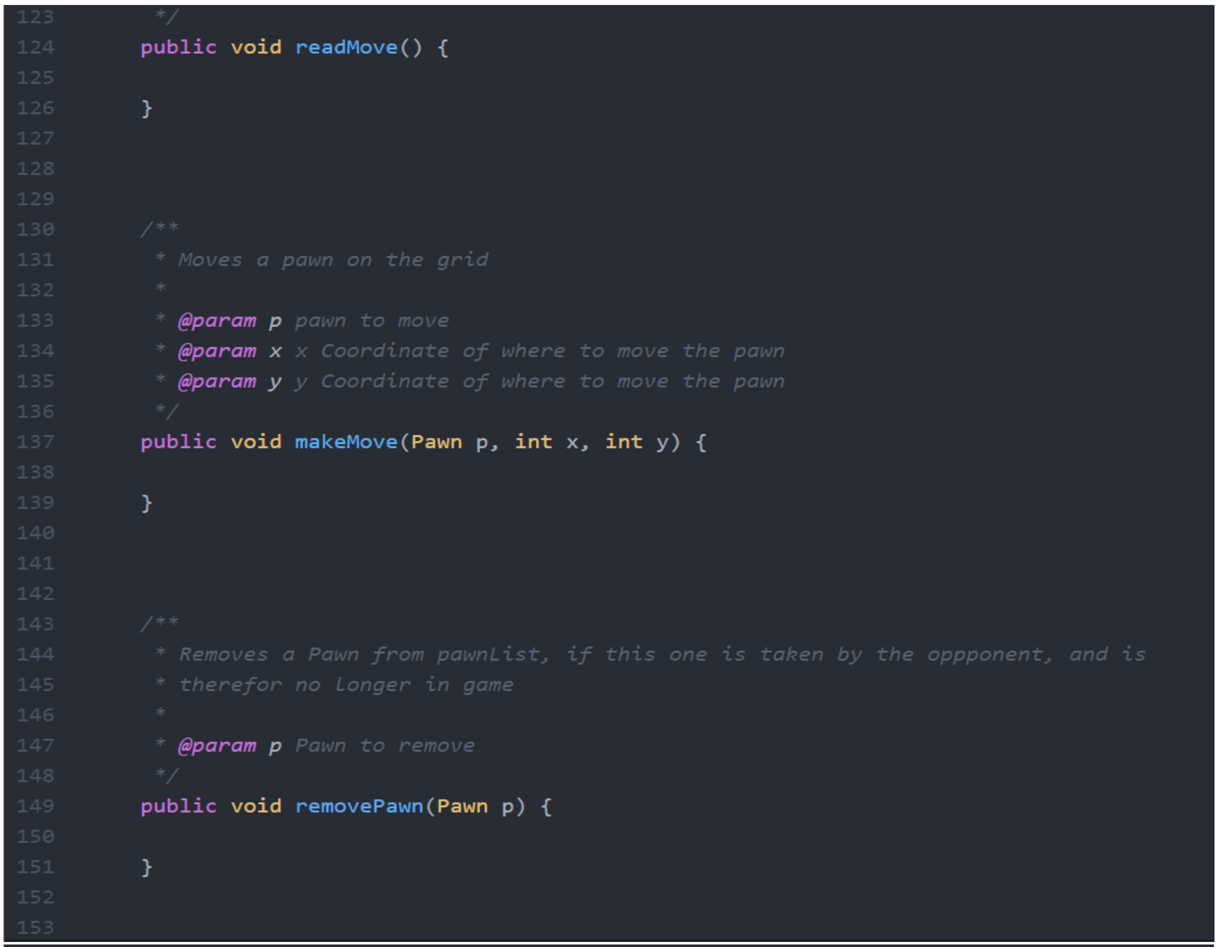
```java



    /**
     * class constructor in case the user is playing against the machine
     *
     * @param player1 name of the first player
     * @param player2 name of the second player
     * @param gameMode gameMode (for debugging purposes)
     * @param dif Difficulty of the Automated player
     */
    public Game(String player1, String player2, Mode gameMode, Difficulty dif) {

    }



    /***
     * class constructor in case the game is loaded
     */
    public Game() {

    }



    /**
     * method used to start the game
     */
    public void start() {
```



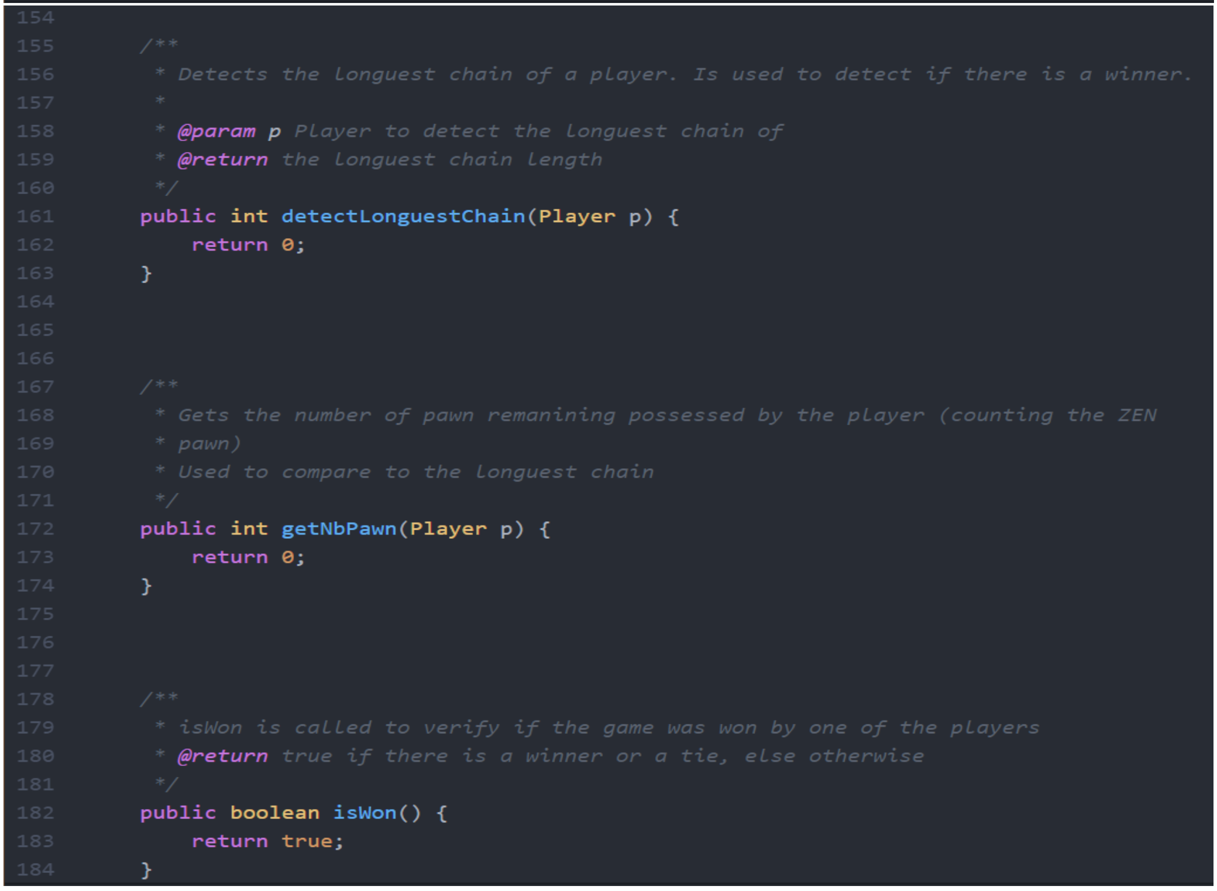
```java

    }



    /**
     * method used to end the game when someone won
     */
    public void end() {

    }



    /**
     * returns the arrayList of Pawns (for testing purposes)
     *
     * @return pawnList attribut
     */
    public ArrayList<Pawn> getPawnList() {
        return new ArrayList<Pawn>();
    }



    /**
     * returns the grid (for testing purposes)
     *
     * @return game board
     */
    public Square[][] getGrid() {
```



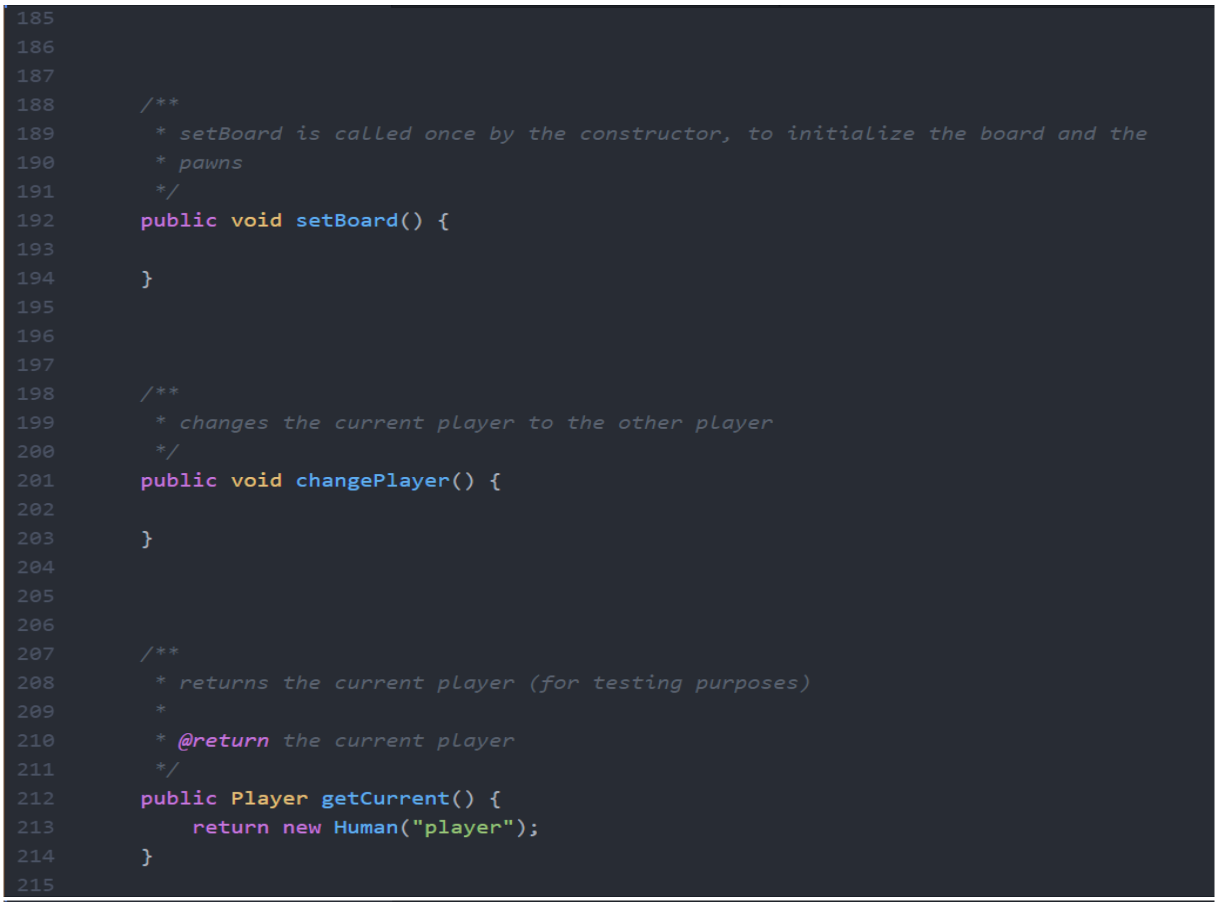
```java
        return new Square[0][0];
    }



    /**
     * Gets the pawn on the given square
     *
     * @param x x coordinate of the square
     * @param y y coordinate of the square
     * @return a pawn, or null if the square is empty
     */
    public Pawn getPawnOnSquare(int x, int y) {
        return new Pawn(Color.ZEN);
    }



    /**
     * Method used to draw th board with the pawns in the current stat in the console
     */
    public void drawBoard() {

    }



    /**
     * readMove reads the players next move (asks for pawn to move, and for the next
     * coordinates).
     * Reapats until move is right, or the player saved the game to quit.
```



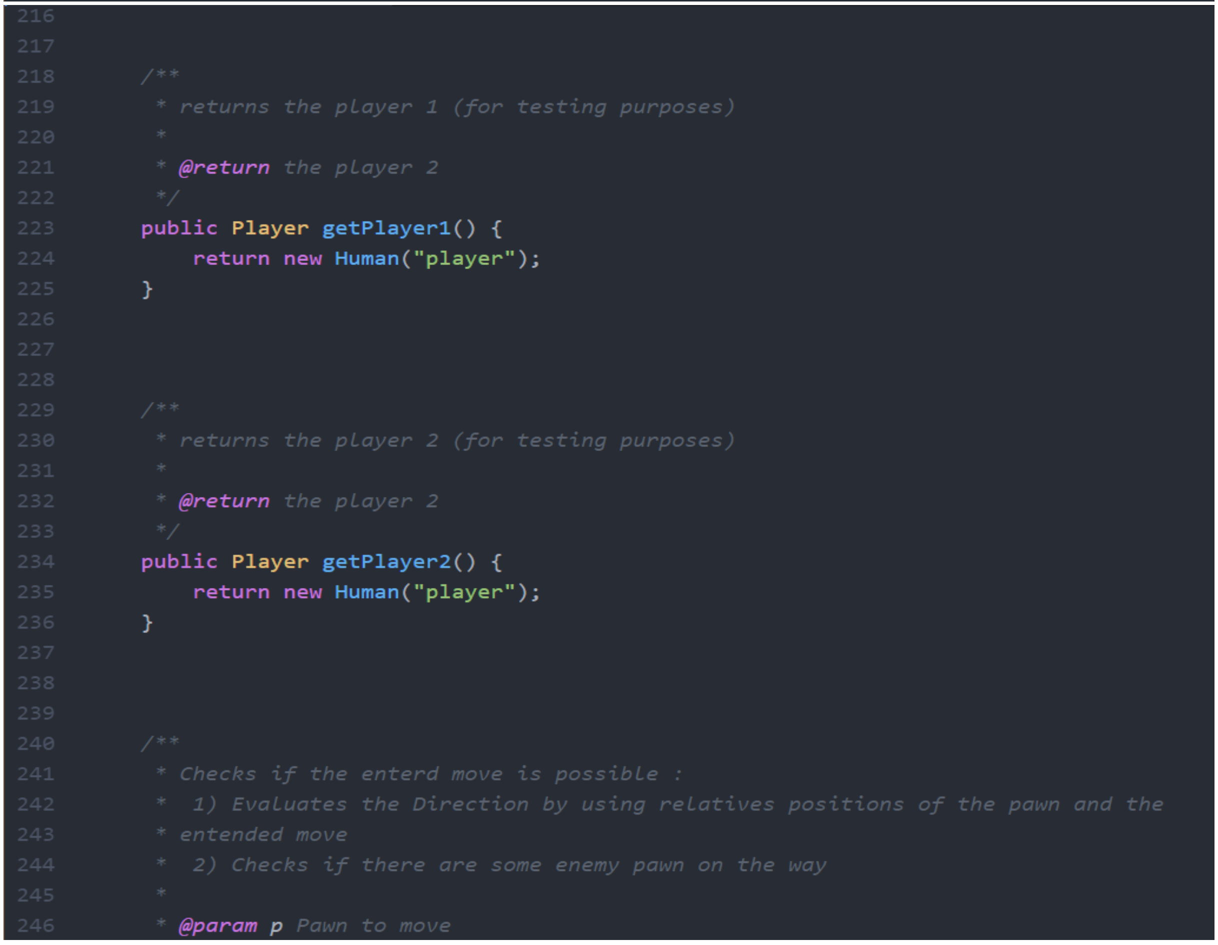
```
     */
    public void readMove() {

    }



    /**
     * Moves a pawn on the grid
     *
     * @param p pawn to move
     * @param x x Coordinate of where to move the pawn
     * @param y y Coordinate of where to move the pawn
     */
    public void makeMove(Pawn p, int x, int y) {

    }



    /**
     * Removes a Pawn from pawnList, if this one is taken by the oppponent, and is
     * therefor no longer in game
     *
     * @param p Pawn to remove
     */
    public void removePawn(Pawn p) {

    }


```



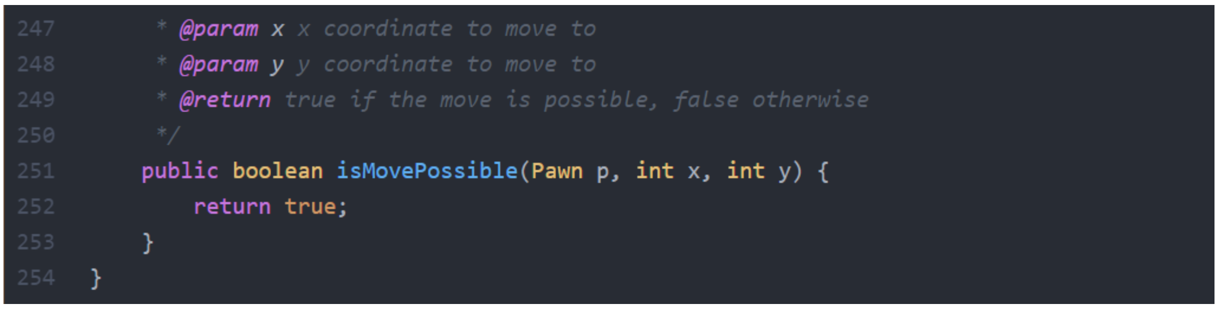
```

    /**
     * Detects the longuest chain of a player. Is used to detect if there is a winner.
     *
     * @param p Player to detect the longuest chain of
     * @return the longuest chain length
     */
    public int detectLonguestChain(Player p) {
        return 0;
    }



    /**
     * Gets the number of pawn remanining possessed by the player (counting the ZEN
     * pawn)
     * Used to compare to the longuest chain
     */
    public int getNbPawn(Player p) {
        return 0;
    }



    /**
     * isWon is called to verify if the game was won by one of the players
     * @return true if there is a winner or a tie, else otherwise
     */
    public boolean isWon() {
        return true;
    }
```



```



    /**
     * setBoard is called once by the constructor, to initialize the board and the
     * pawns
     */
    public void setBoard() {

    }



    /**
     * changes the current player to the other player
     */
    public void changePlayer() {

    }



    /**
     * returns the current player (for testing purposes)
     *
     * @return the current player
     */
    public Player getCurrent() {
        return new Human("player");
    }

```

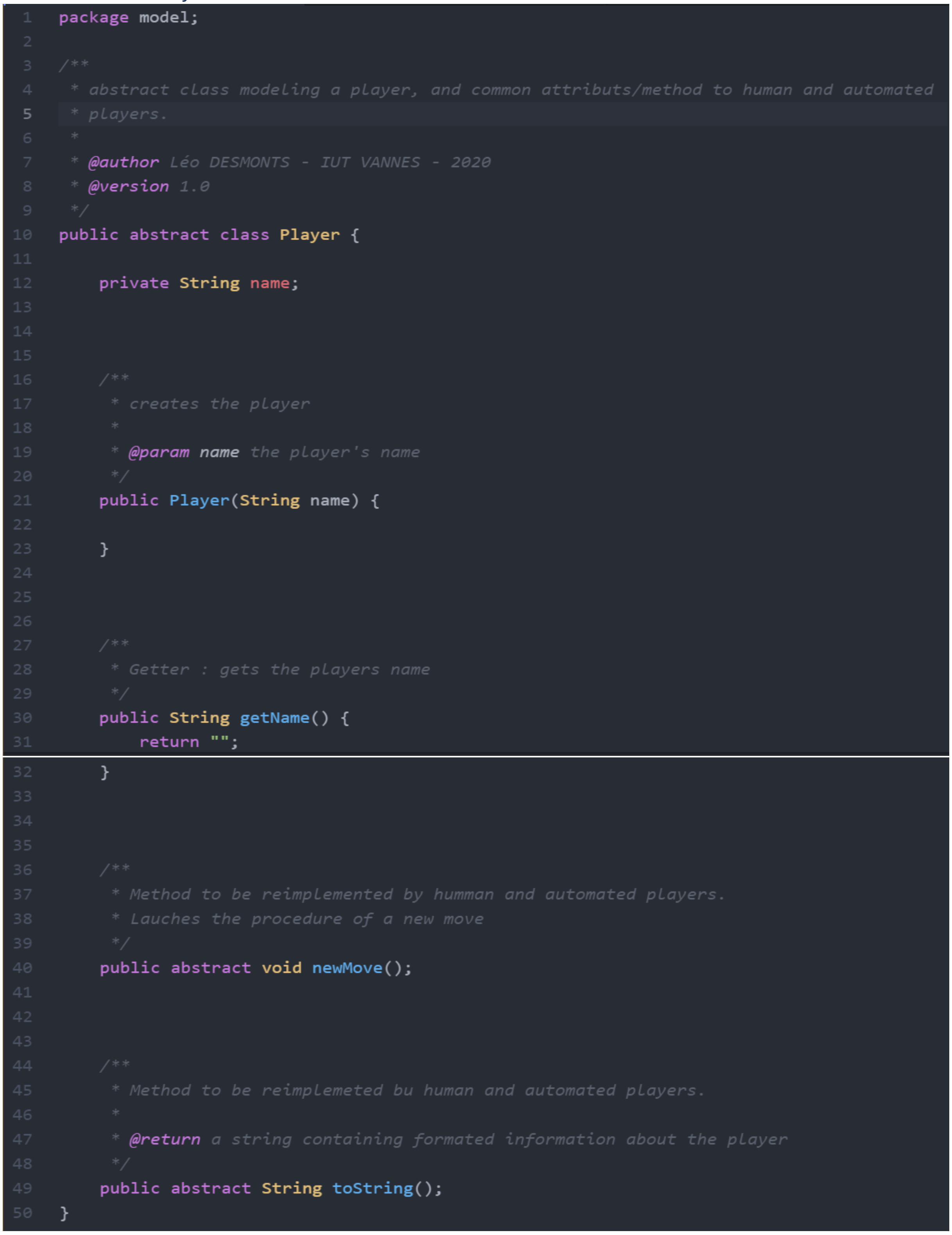


```


    /**
     * returns the player 1 (for testing purposes)
     *
     * @return the player 2
     */
    public Player getPlayer1() {
        return new Human("player");
    }



    /**
     * returns the player 2 (for testing purposes)
     *
     * @return the player 2
     */
    public Player getPlayer2() {
        return new Human("player");
    }



    /**
     * Checks if the enterd move is possible :
     *  1) Evaluates the Direction by using relatives positions of the pawn and the
     * entended move
     *  2) Checks if there are some enemy pawn on the way
     *
     * @param p Pawn to move
```



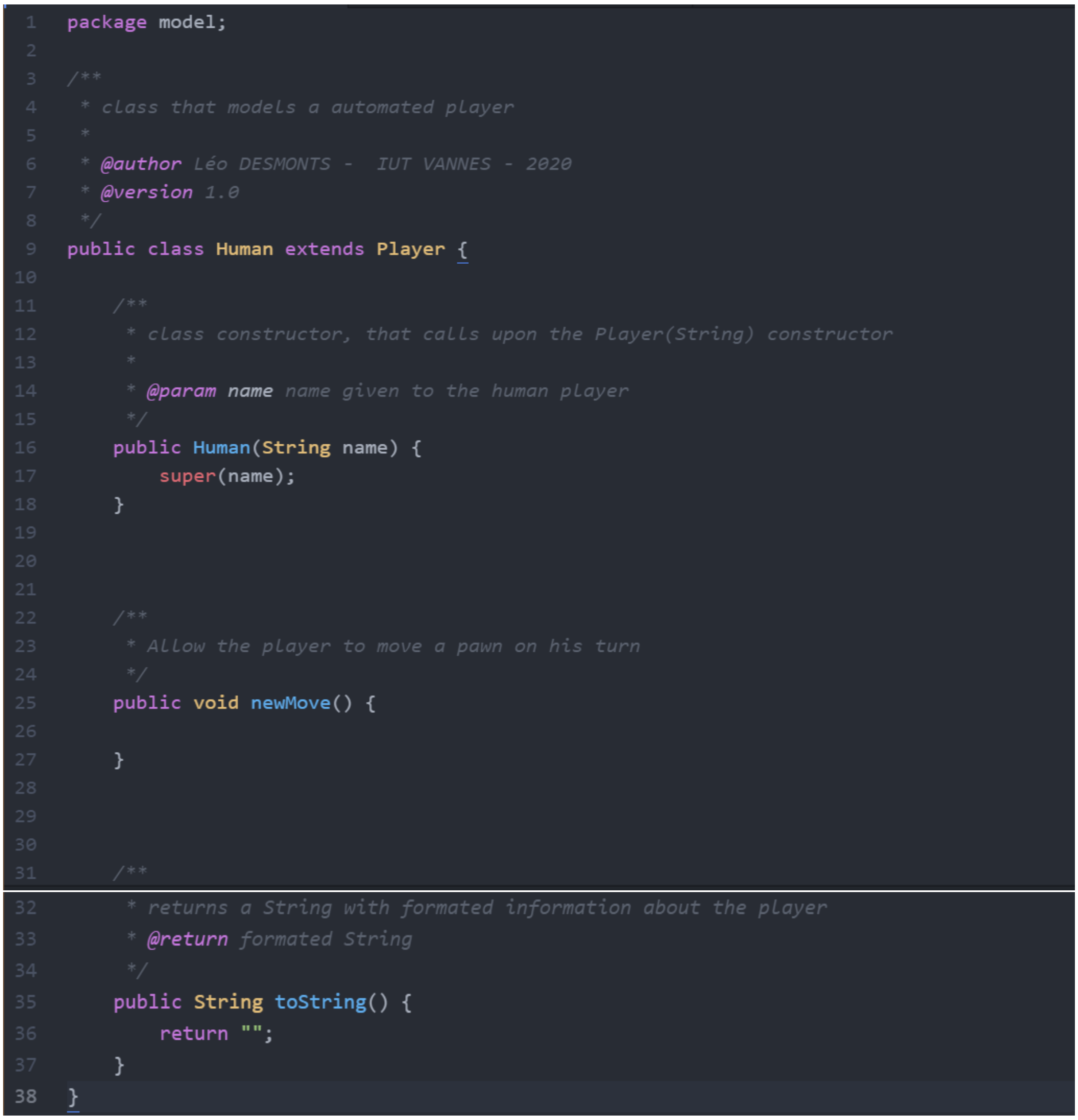
```
     * @param x x coordinate to move to
     * @param y y coordinate to move to
     * @return true if the move is possible, false otherwise
     */
    public boolean isMovePossible(Pawn p, int x, int y) {
        return true;
    }
}
```

3. Player



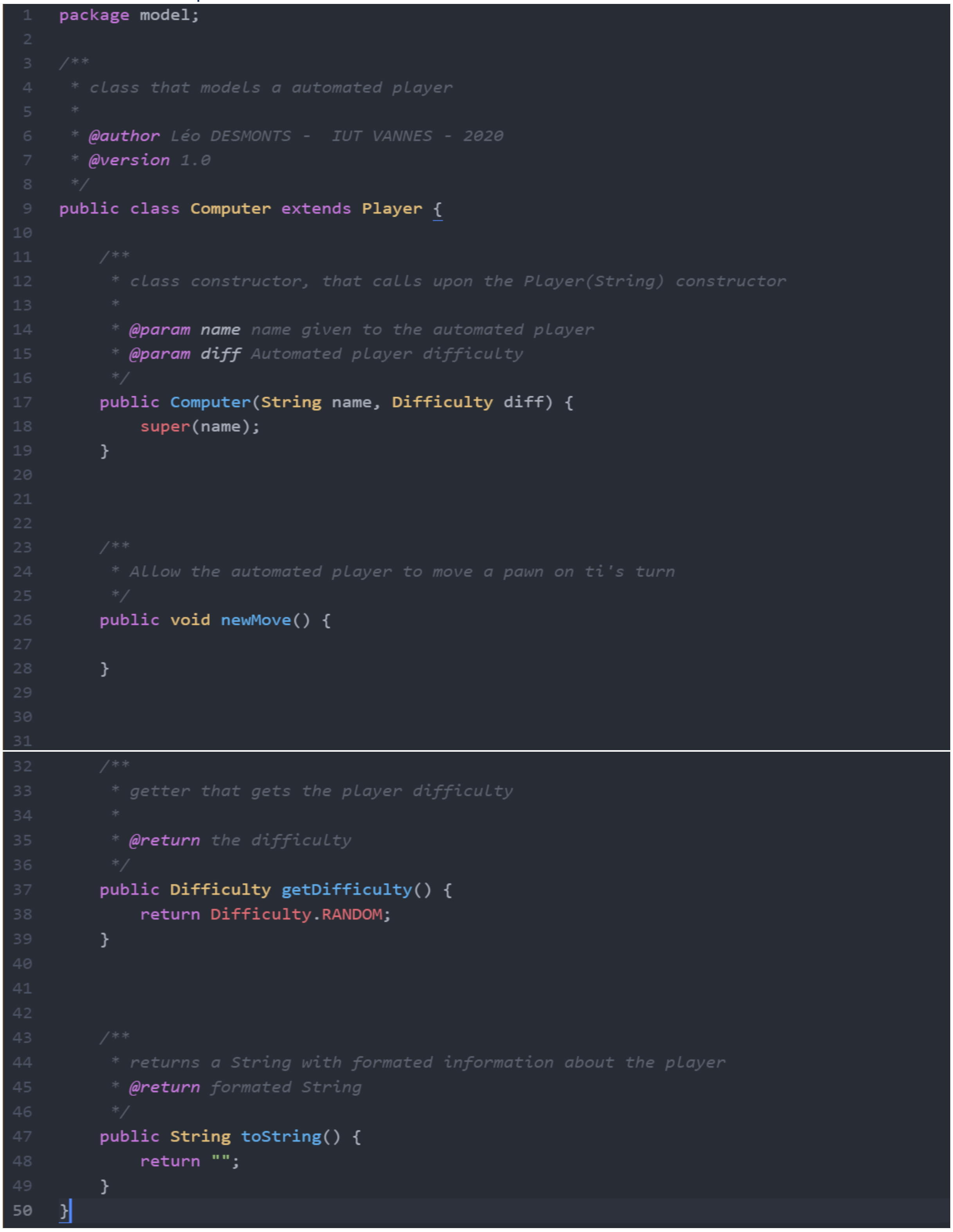
```java
package model;

/**
 * abstract class modeling a player, and common attributs/method to human and automated
 * players.
 *
 * @author Léo DESMONTS - IUT VANNES - 2020
 * @version 1.0
 */
public abstract class Player {

    private String name;



    /**
     * creates the player
     *
     * @param name the player's name
     */
    public Player(String name) {

    }



    /**
     * Getter : gets the players name
     */
    public String getName() {
        return "";
    }



    /**
     * Method to be reimplemented by humman and automated players.
     * Lauches the procedure of a new move
     */
    public abstract void newMove();



    /**
     * Method to be reimplemeted bu human and automated players.
     *
     * @return a string containing formated information about the player
     */
    public abstract String toString();
}
```

4. Human



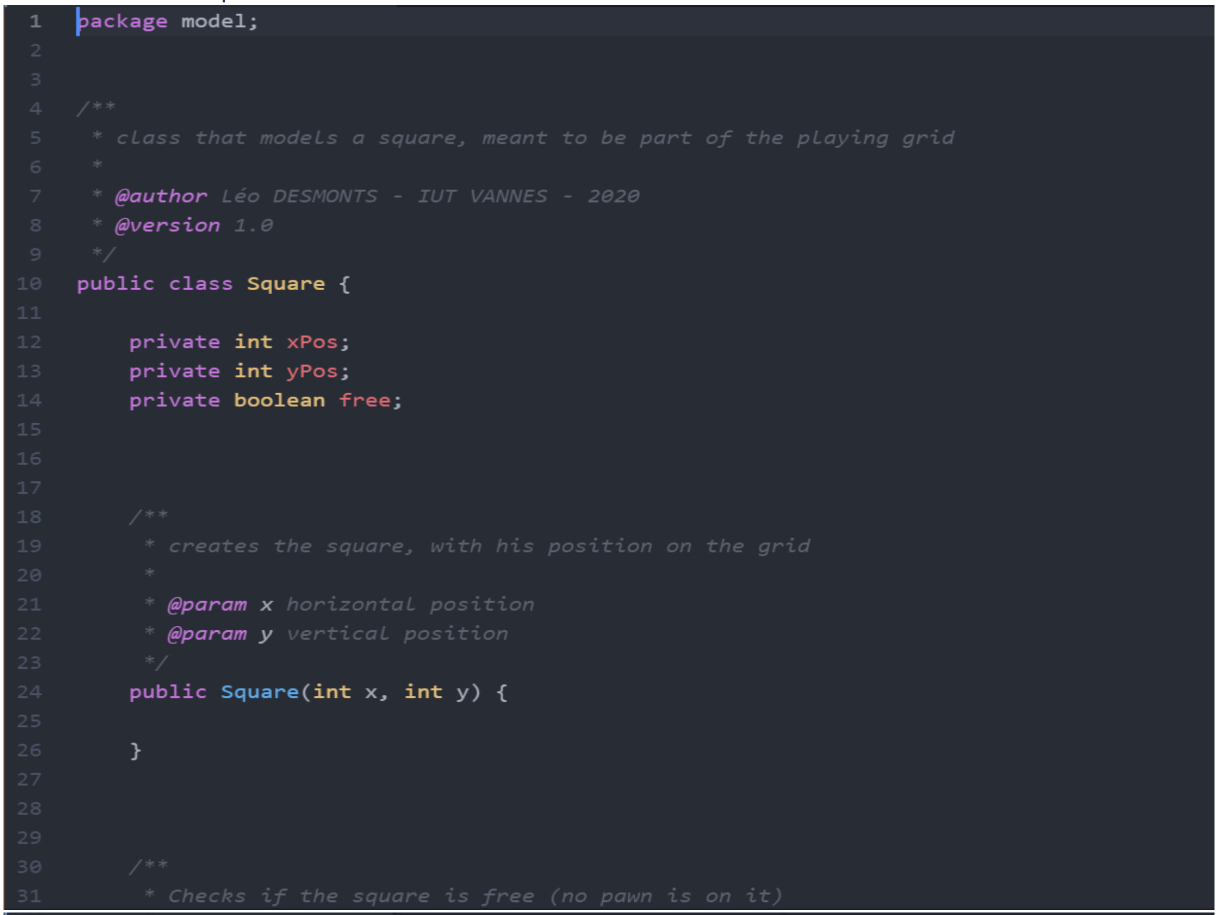
```java
package model;

/**
 * class that models a automated player
 *
 * @author Léo DESMONTS -  IUT VANNES - 2020
 * @version 1.0
 */
public class Human extends Player {

    /**
     * class constructor, that calls upon the Player(String) constructor
     *
     * @param name name given to the human player
     */
    public Human(String name) {
        super(name);
    }



    /**
     * Allow the player to move a pawn on his turn
     */
    public void newMove() {

    }



    /**
     * returns a String with formated information about the player
     * @return formated String
     */
    public String toString() {
        return "";
    }
}
```

5. Computer

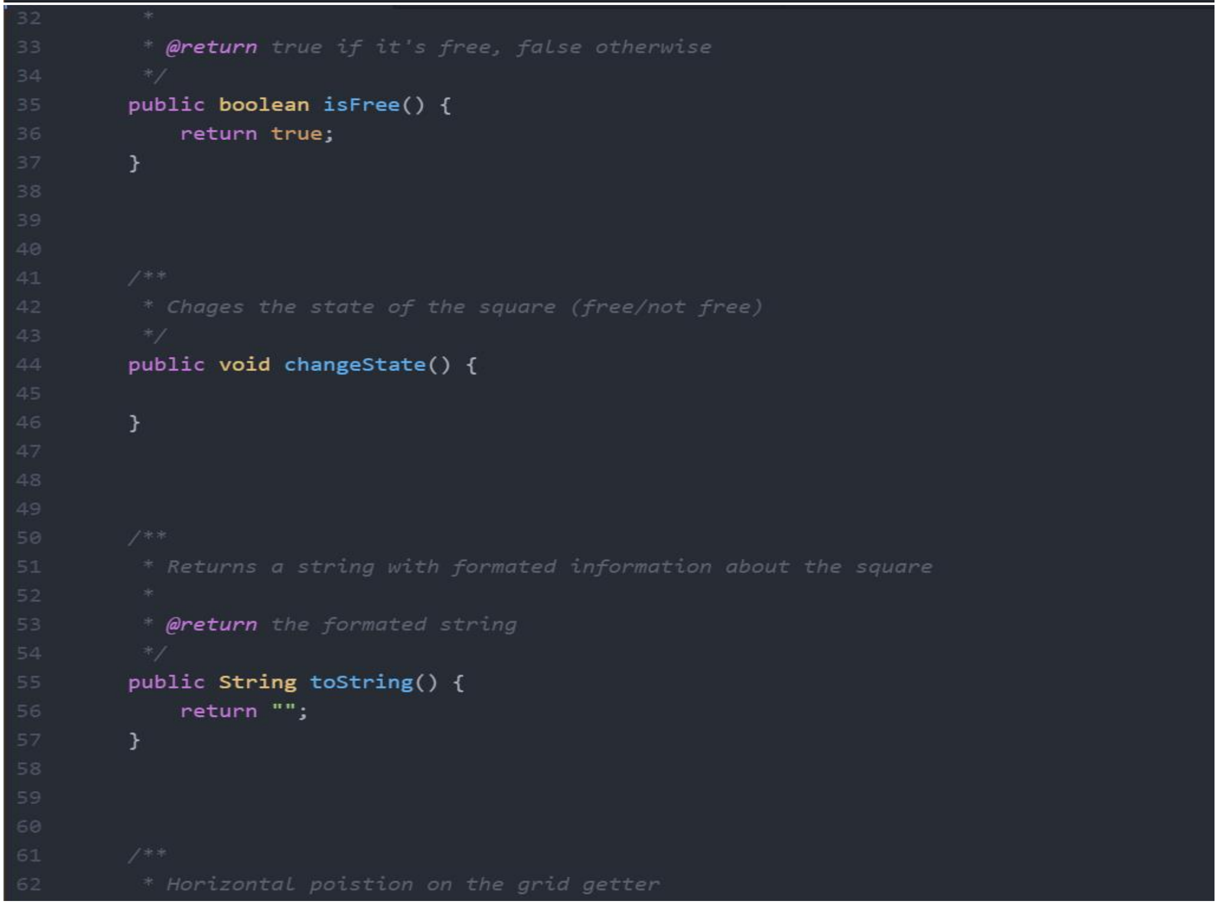


```java
package model;

/**
 * class that models a automated player
 *
 * @author Léo DESMONTS -  IUT VANNES - 2020
 * @version 1.0
 */
public class Computer extends Player {

    /**
     * class constructor, that calls upon the Player(String) constructor
     *
     * @param name name given to the automated player
     * @param diff Automated player difficulty
     */
    public Computer(String name, Difficulty diff) {
        super(name);
    }



    /**
     * Allow the automated player to move a pawn on ti's turn
     */
    public void newMove() {

    }



    /**
     * getter that gets the player difficulty
     *
     * @return the difficulty
     */
    public Difficulty getDifficulty() {
        return Difficulty.RANDOM;
    }



    /**
     * returns a String with formated information about the player
     * @return formated String
     */
    public String toString() {
        return "";
    }
}
```

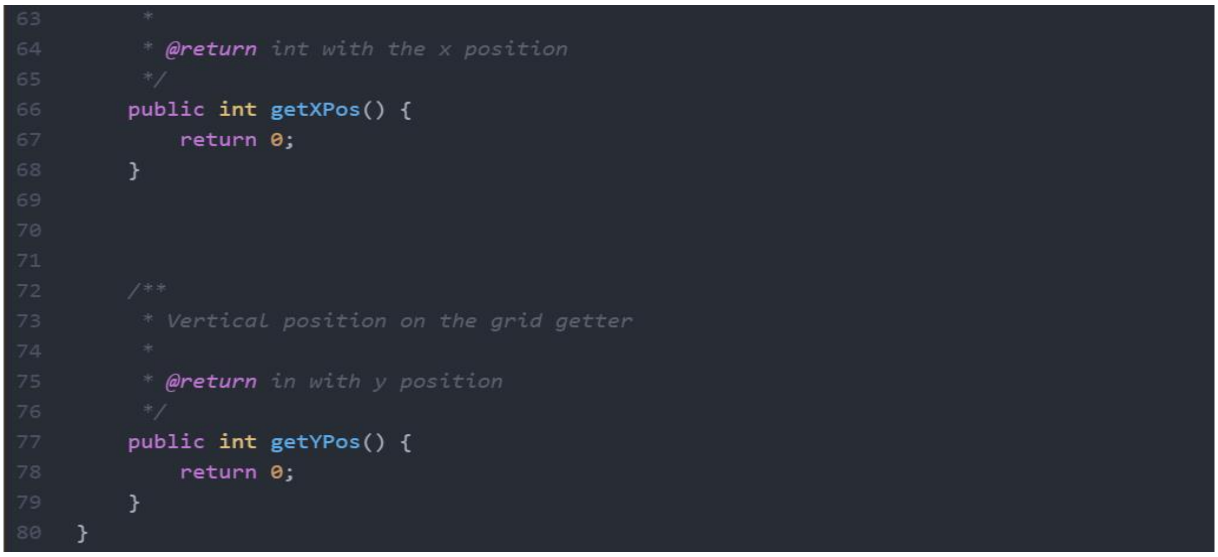
6. Square



```java
package model;


/**
 * class that models a square, meant to be part of the playing grid
 *
 * @author Léo DESMONTS - IUT VANNES - 2020
 * @version 1.0
 */
public class Square {

    private int xPos;
    private int yPos;
    private boolean free;



    /**
     * creates the square, with his position on the grid
     *
     * @param x horizontal position
     * @param y vertical position
     */
    public Square(int x, int y) {

    }



    /**
     * Checks if the square is free (no pawn is on it)
```

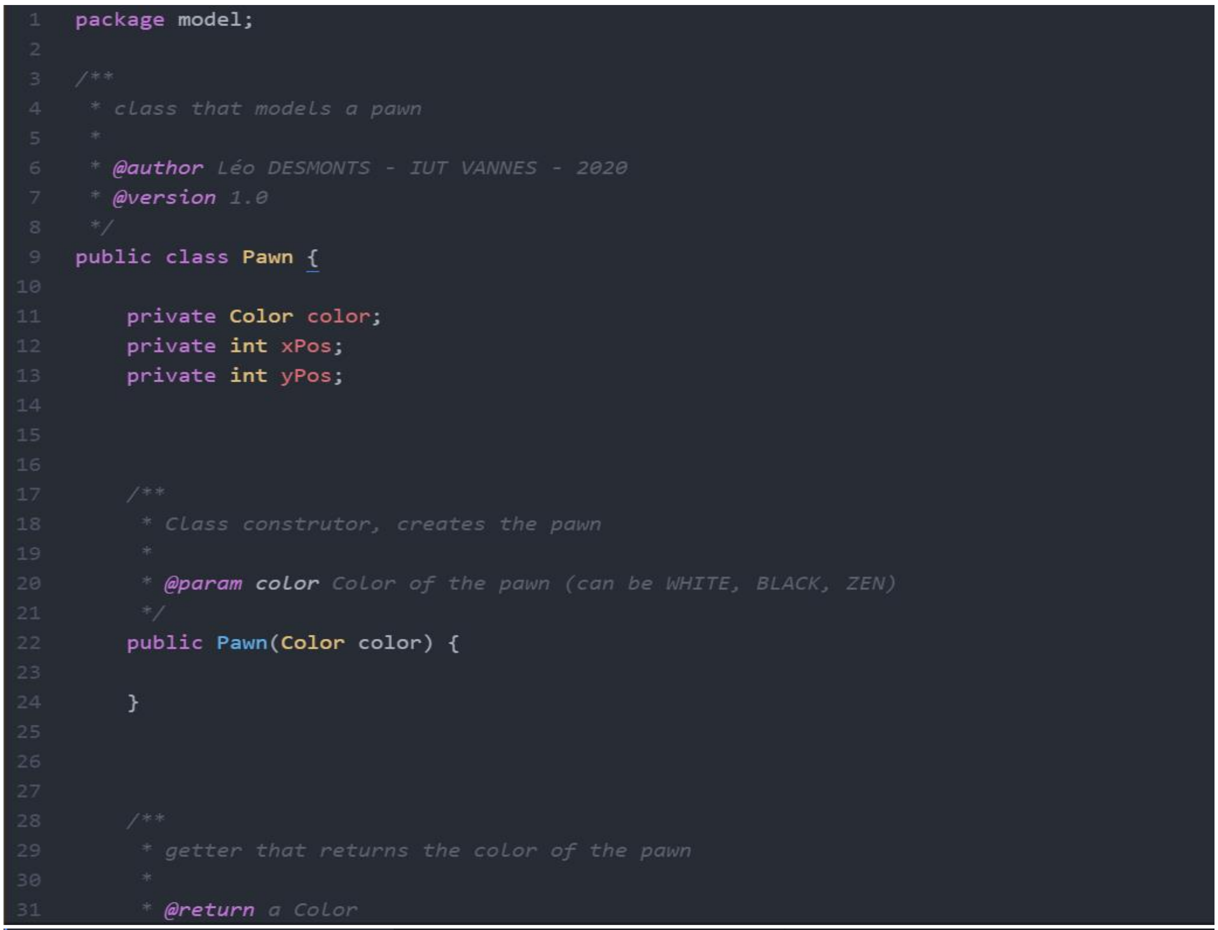


```java
     *
     * @return true if it's free, false otherwise
     */
    public boolean isFree() {
        return true;
    }



    /**
     * Chages the state of the square (free/not free)
     */
    public void changeState() {

    }



    /**
     * Returns a string with formated information about the square
     *
     * @return the formated string
     */
    public String toString() {
        return "";
    }



    /**
     * Horizontal poistion on the grid getter
```

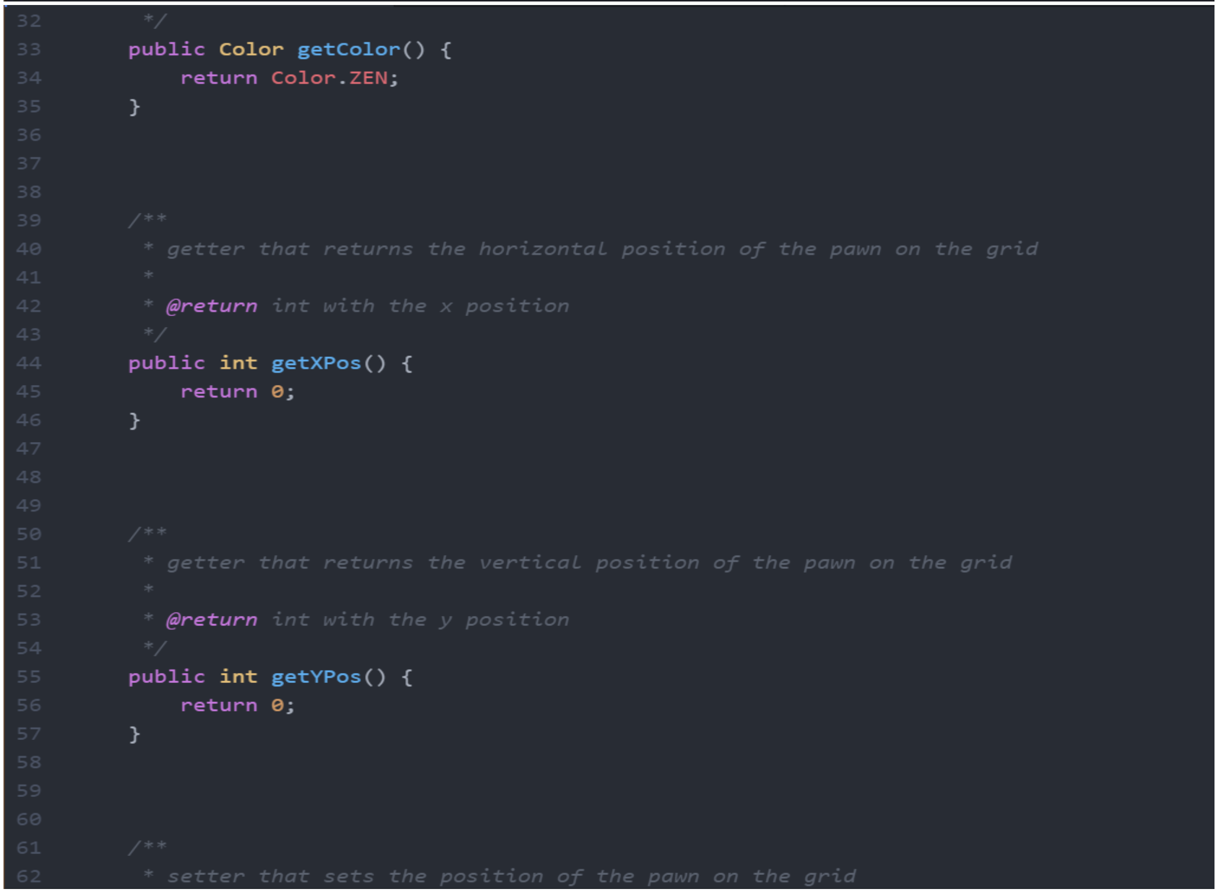


```java
     *
     * @return int with the x position
     */
    public int getXPos() {
        return 0;
    }



    /**
     * Vertical position on the grid getter
     *
     * @return in with y position
     */
    public int getYPos() {
        return 0;
    }
}
```

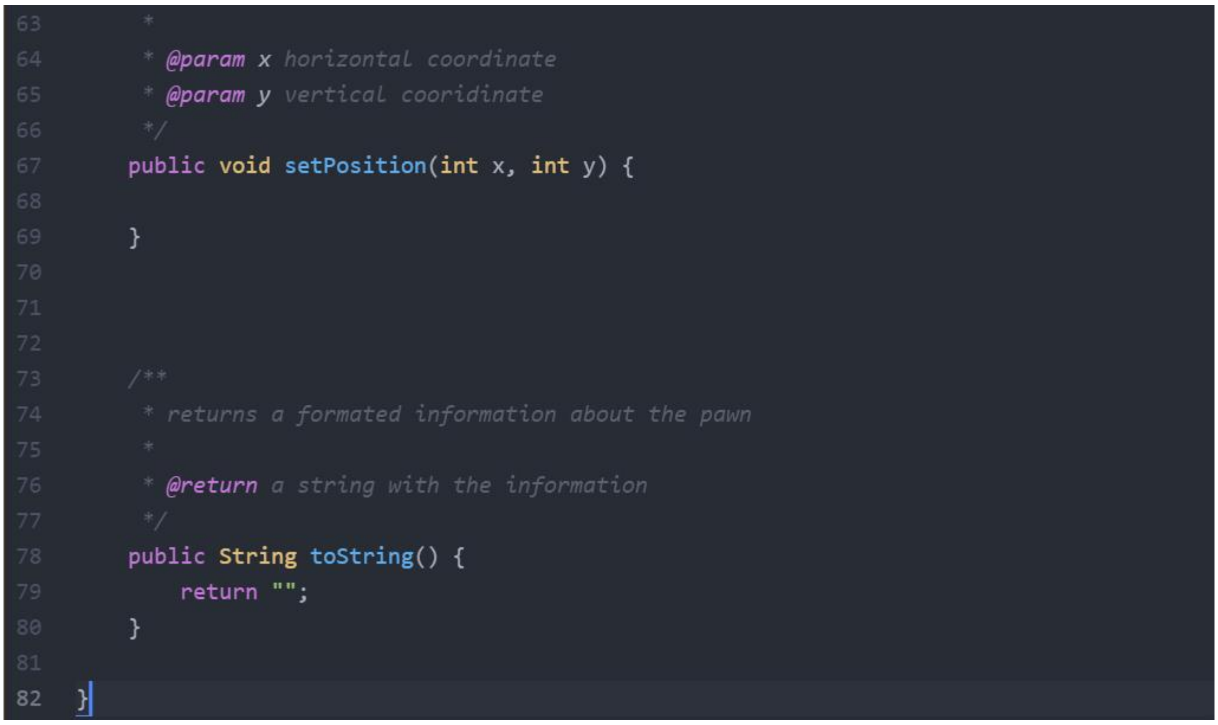
7. Pawn



```java
package model;

/**
 * class that models a pawn
 *
 * @author Léo DESMONTS - IUT VANNES - 2020
 * @version 1.0
 */
public class Pawn {

    private Color color;
    private int xPos;
    private int yPos;



    /**
     * Class construtor, creates the pawn
     *
     * @param color Color of the pawn (can be WHITE, BLACK, ZEN)
     */
    public Pawn(Color color) {

    }



    /**
     * getter that returns the color of the pawn
     *
     * @return a Color
```



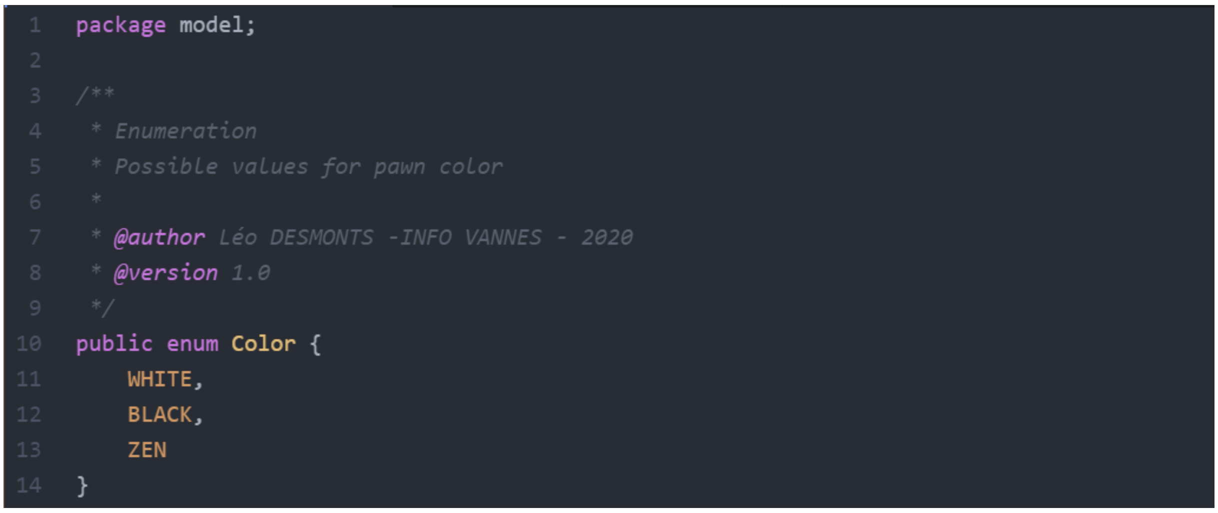
```java
     */
    public Color getColor() {
        return Color.ZEN;
    }



    /**
     * getter that returns the horizontal position of the pawn on the grid
     *
     * @return int with the x position
     */
    public int getXPos() {
        return 0;
    }



    /**
     * getter that returns the vertical position of the pawn on the grid
     *
     * @return int with the y position
     */
    public int getYPos() {
        return 0;
    }



    /**
     * setter that sets the position of the pawn on the grid
```



```
         *
         * @param x horizontal coordinate
         * @param y vertical cooridinate
         */
        public void setPosition(int x, int y) {

        }



        /**
         * returns a formated information about the pawn
         *
         * @return a string with the information
         */
        public String toString() {
            return "";
        }

}
```

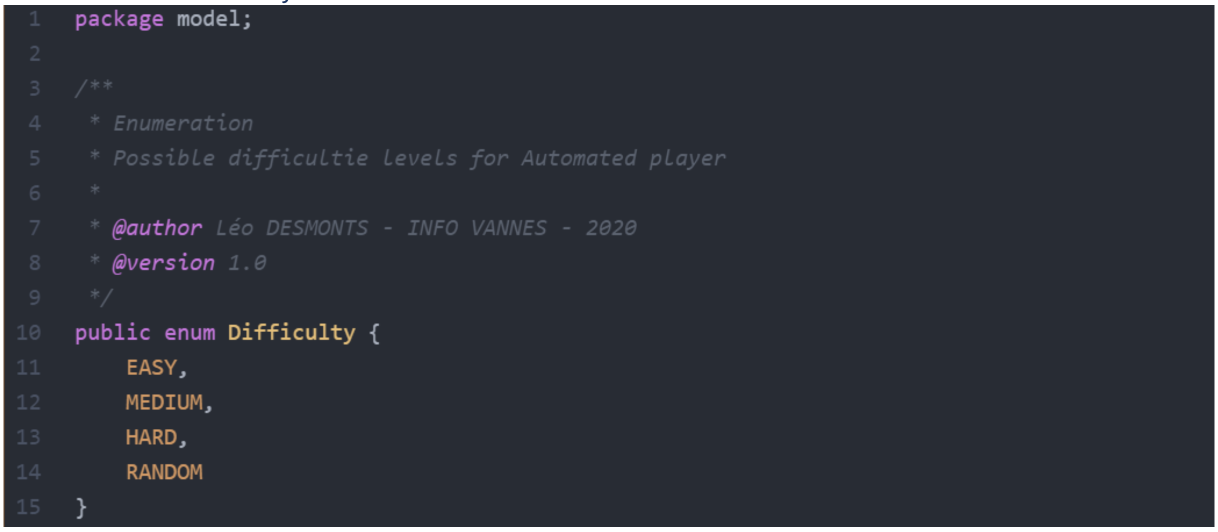
s

8. Color



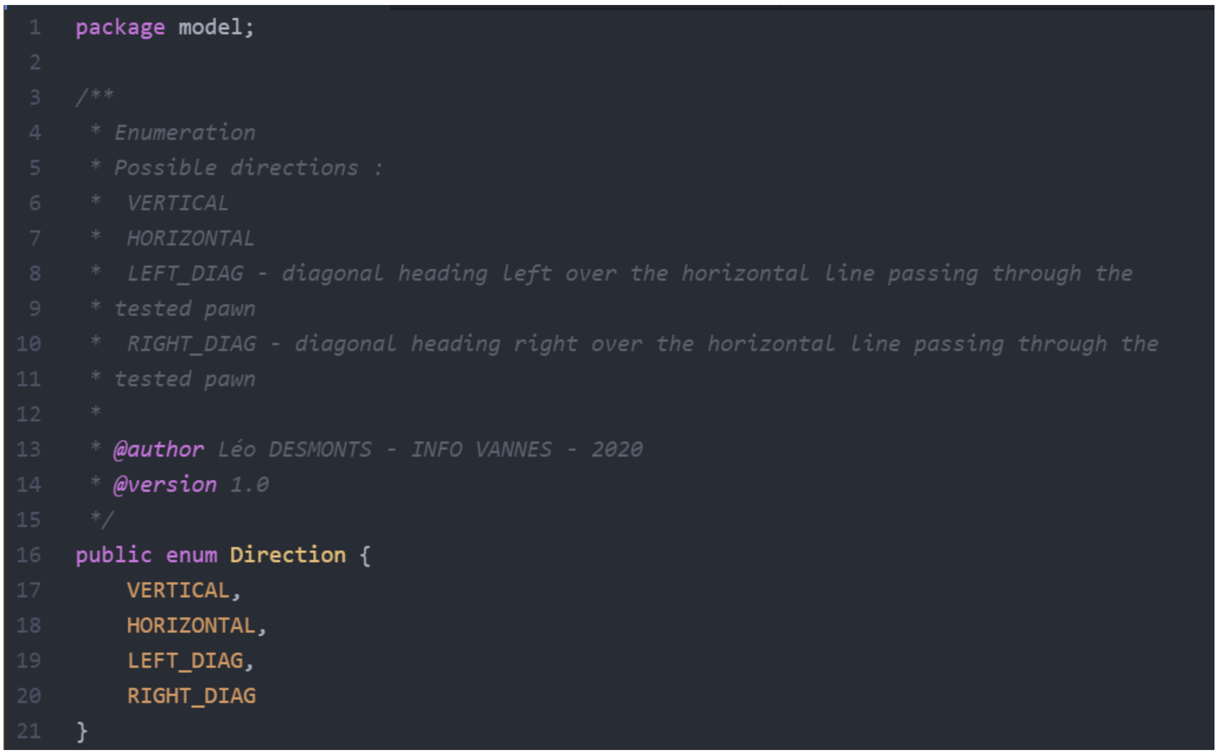
```java
package model;

/**
 * Enumeration
 * Possible values for pawn color
 *
 * @author Léo DESMONTS -INFO VANNES - 2020
 * @version 1.0
 */
public enum Color {
    WHITE,
    BLACK,
    ZEN
}
```

9. Difficulty



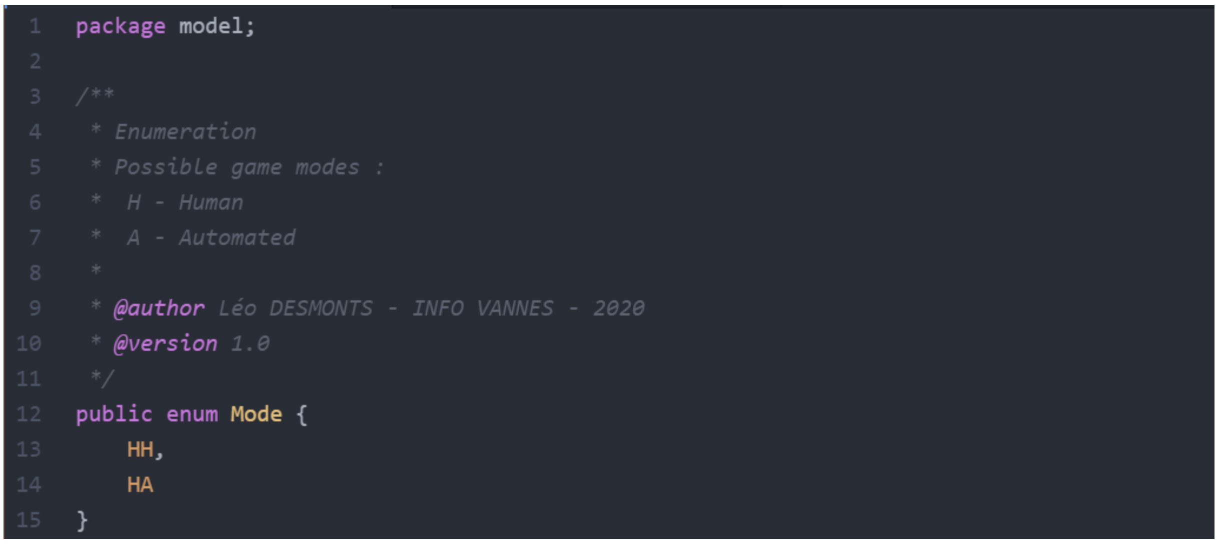
```java
package model;

/**
 * Enumeration
 * Possible difficultie levels for Automated player
 *
 * @author Léo DESMONTS - INFO VANNES - 2020
 * @version 1.0
 */
public enum Difficulty {
    EASY,
    MEDIUM,
    HARD,
    RANDOM
}
```

10. Direction



```java
package model;

/**
 * Enumeration
 * Possible directions :
 *  VERTICAL
 *  HORIZONTAL
 *  LEFT_DIAG - diagonal heading left over the horizontal line passing through the
 * tested pawn
 *  RIGHT_DIAG - diagonal heading right over the horizontal line passing through the
 * tested pawn
 *
 * @author Léo DESMONTS - INFO VANNES - 2020
 * @version 1.0
 */
public enum Direction {
    VERTICAL,
    HORIZONTAL,
    LEFT_DIAG,
    RIGHT_DIAG
}
```

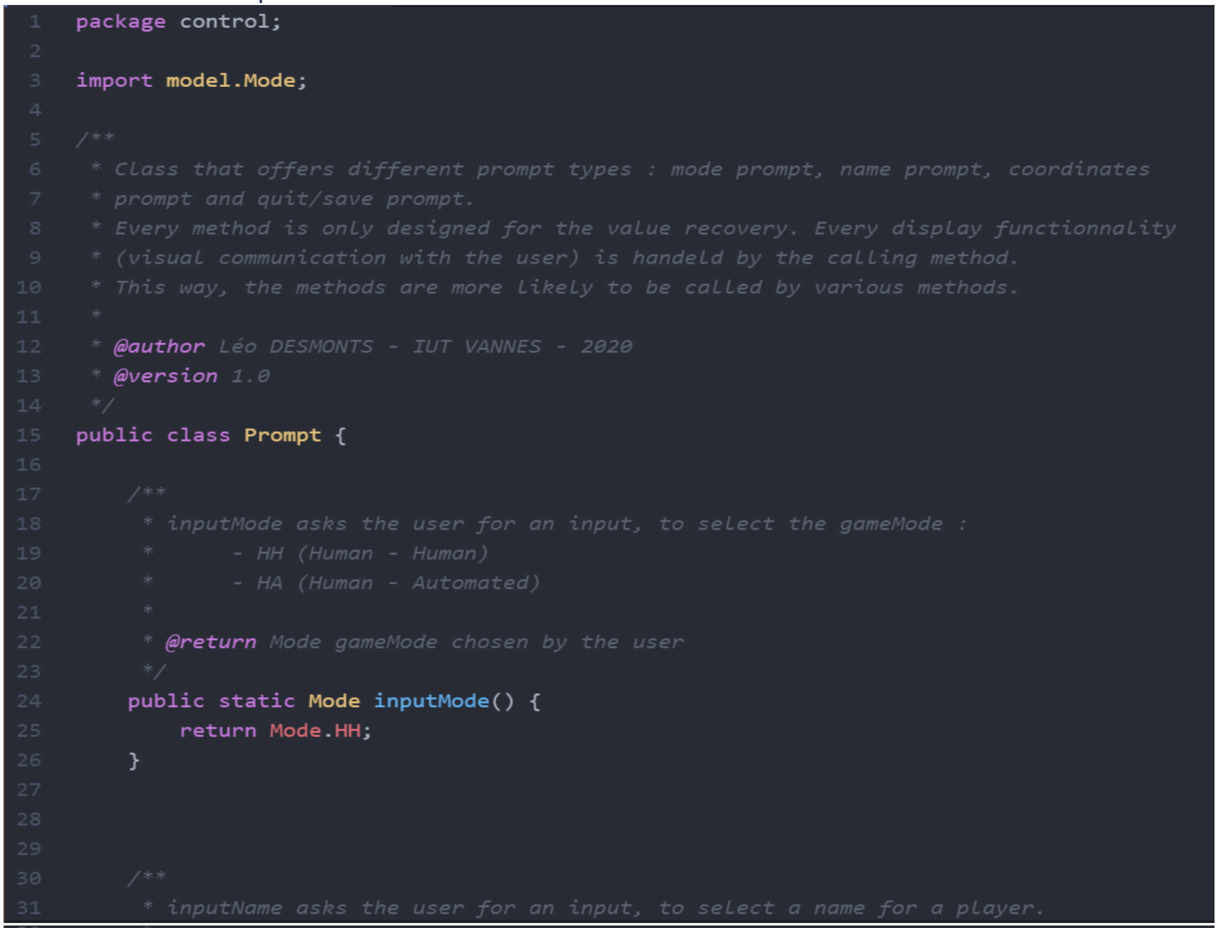
11. Mode



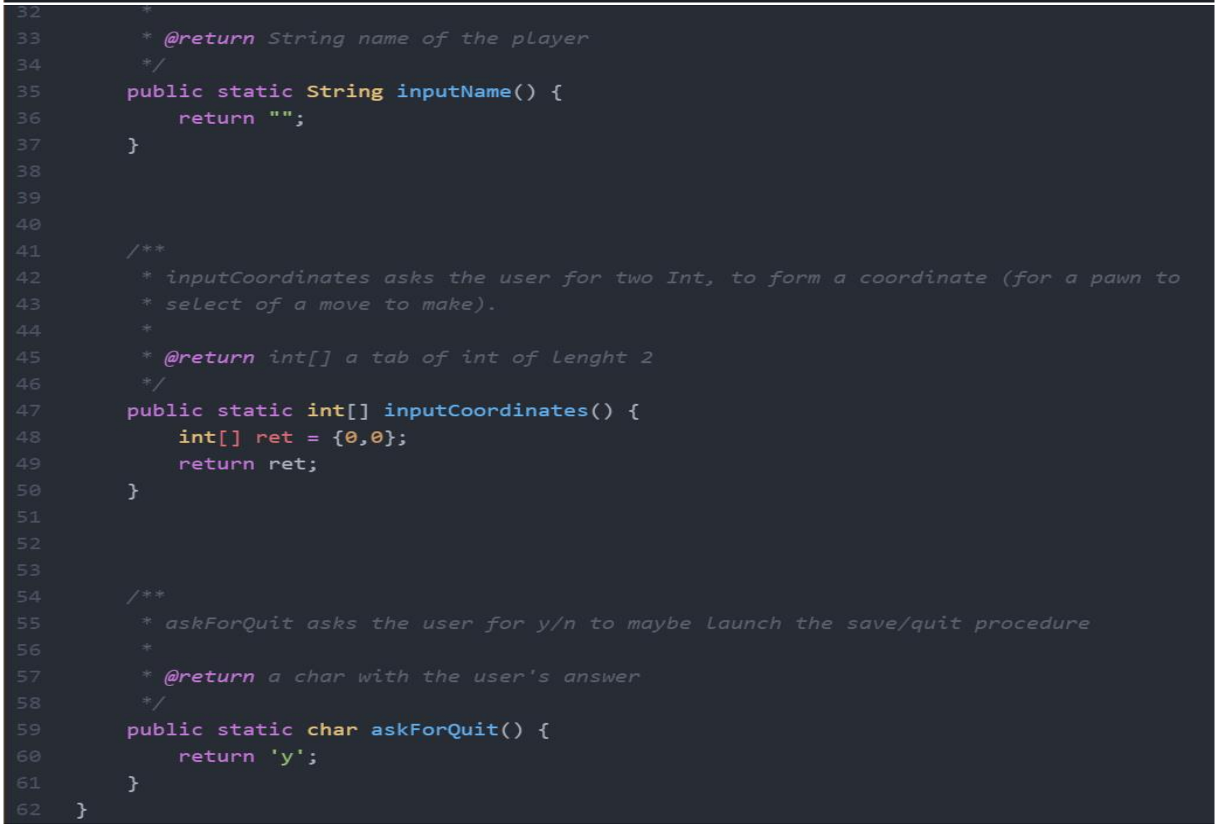
```java
package model;

/**
 * Enumeration
 * Possible game modes :
 *  H - Human
 *  A - Automated
 *
 * @author Léo DESMONTS - INFO VANNES - 2020
 * @version 1.0
 */
public enum Mode {
    HH,
    HA
}
```

b) Package control

1. Prompt



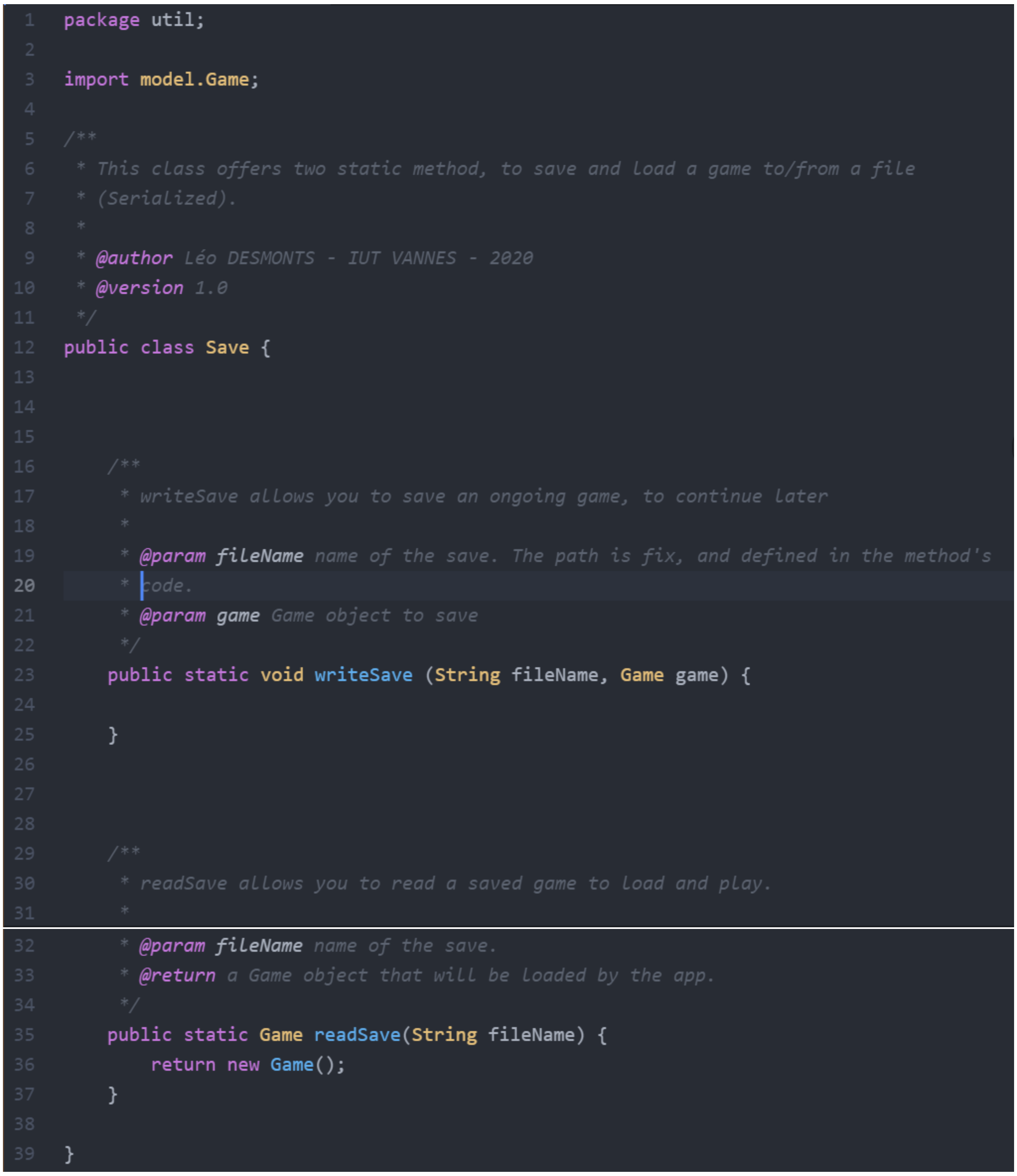
```java
package control;

import model.Mode;

/**
 * Class that offers different prompt types : mode prompt, name prompt, coordinates
 * prompt and quit/save prompt.
 * Every method is only designed for the value recovery. Every display functionnality
 * (visual communication with the user) is handeld by the calling method.
 * This way, the methods are more likely to be called by various methods.
 *
 * @author Léo DESMONTS - IUT VANNES - 2020
 * @version 1.0
 */
public class Prompt {

    /**
     * inputMode asks the user for an input, to select the gameMode :
     *       - HH (Human - Human)
     *       - HA (Human - Automated)
     *
     * @return Mode gameMode chosen by the user
     */
    public static Mode inputMode() {
        return Mode.HH;
    }



    /**
     * inputName asks the user for an input, to select a name for a player.
```



```java
     *
     * @return String name of the player
     */
    public static String inputName() {
        return "";
    }



    /**
     * inputCoordinates asks the user for two Int, to form a coordinate (for a pawn to
     * select of a move to make).
     *
     * @return int[] a tab of int of lenght 2
     */
    public static int[] inputCoordinates() {
        int[] ret = {0,0};
        return ret;
    }



    /**
     * askForQuit asks the user for y/n to maybe launch the save/quit procedure
     *
     * @return a char with the user's answer
     */
    public static char askForQuit() {
        return 'y';
    }
}
```

c) Package util

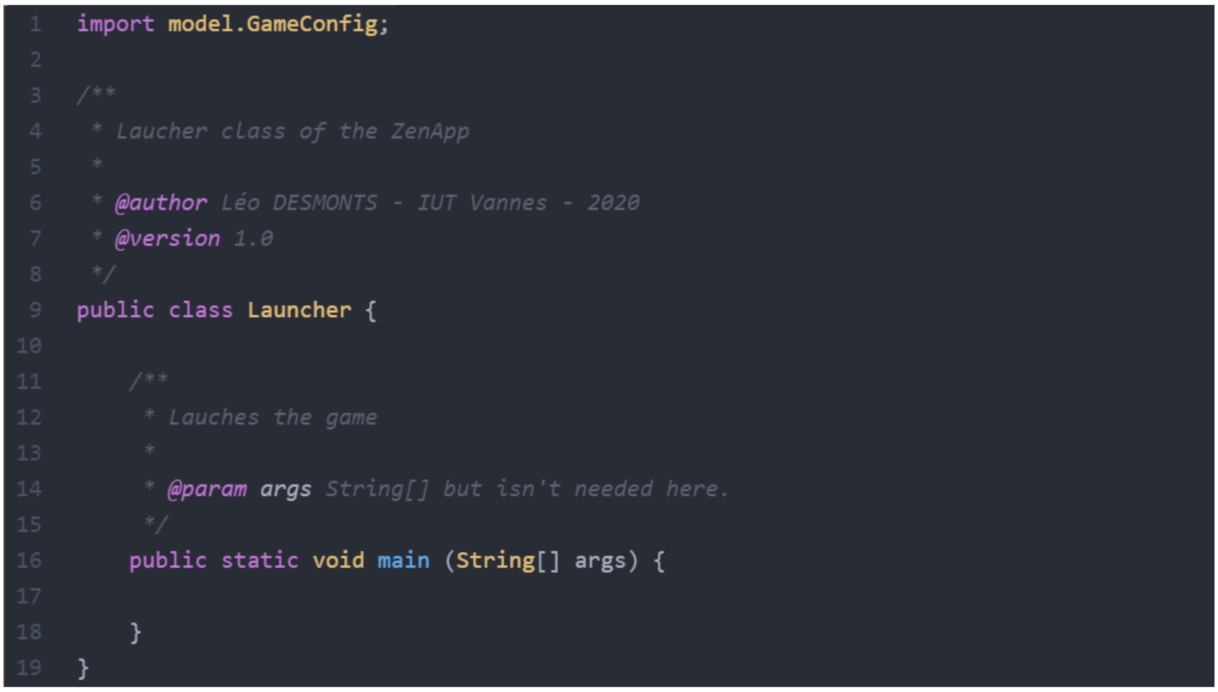
1. Save



```java
package util;

import model.Game;

/**
 * This class offers two static method, to save and load a game to/from a file
 * (Serialized).
 *
 * @author Léo DESMONTS - IUT VANNES - 2020
 * @version 1.0
 */
public class Save {



    /**
     * writeSave allows you to save an ongoing game, to continue later
     *
     * @param fileName name of the save. The path is fix, and defined in the method's
     * code.
     * @param game Game object to save
     */
    public static void writeSave (String fileName, Game game) {

    }



    /**
     * readSave allows you to read a saved game to load and play.
     *
     * @param fileName name of the save.
     * @return a Game object that will be loaded by the app.
     */
    public static Game readSave(String fileName) {
        return new Game();
    }

}
```

Les classes présentées ci-dessus sont les classes qui seront complété durant la phase de codage.

d) Launcher



```java
import model.GameConfig;

/**
 * Laucher class of the ZenApp
 *
 * @author Léo DESMONTS - IUT Vannes - 2020
 * @version 1.0
 */
public class Launcher {

    /**
     * Lauches the game
     *
     * @param args String[] but isn't needed here.
     */
    public static void main (String[] args) {

    }
}
```

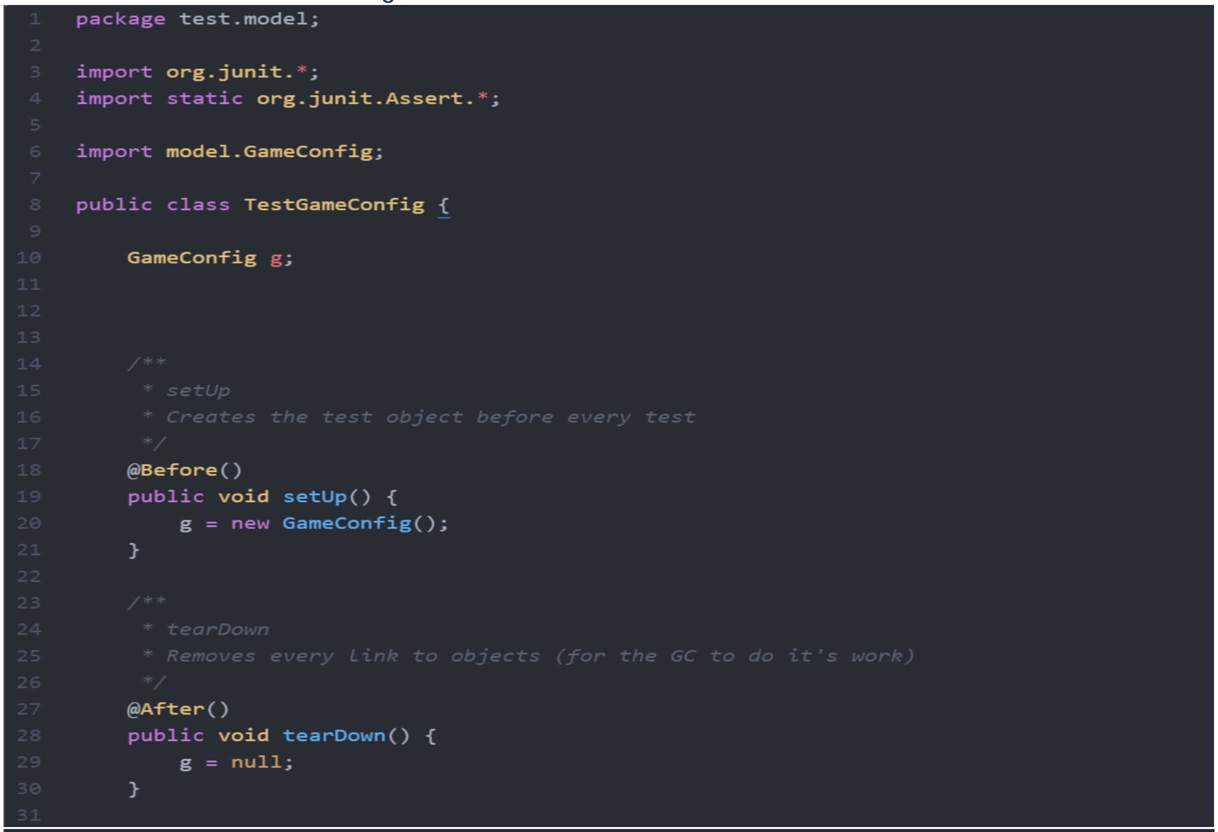
6) Test unitaires (JUnit)

Toute méthode, toute classe doit être testée. On suit ici un modèle de TDD (Test Driven development), développement piloté par les tests en français ; on écrit les tests avant de commencer la phase de code, afin de voir au fur et à mesure si nos méthodes passent tous les tests auxquels on les soumet. Bien sur il est difficile d'être exhaustif, il est donc tout à fait possible de rajouter des test durant/après la phase de développement.

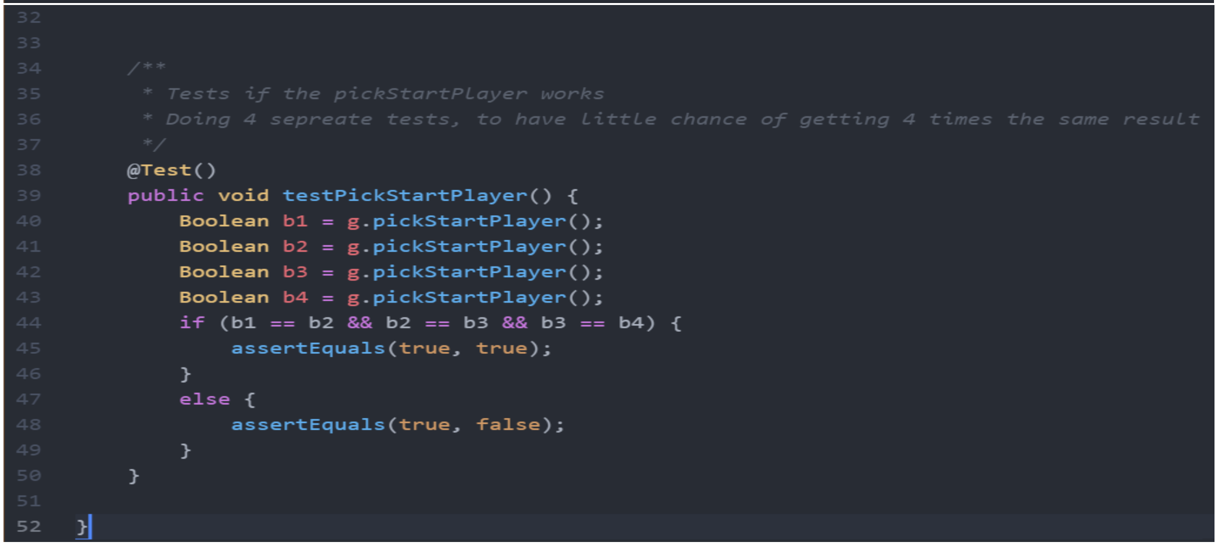
On tente, dans la mesure du possible de faire une classe de test, par classe de l'application. Les enumerations, ainsi que les méthodes demandant une entrée de donnée de la part de l'utilisateur ne seront pas testées.

a) Test du package model

1. TestGameConfig

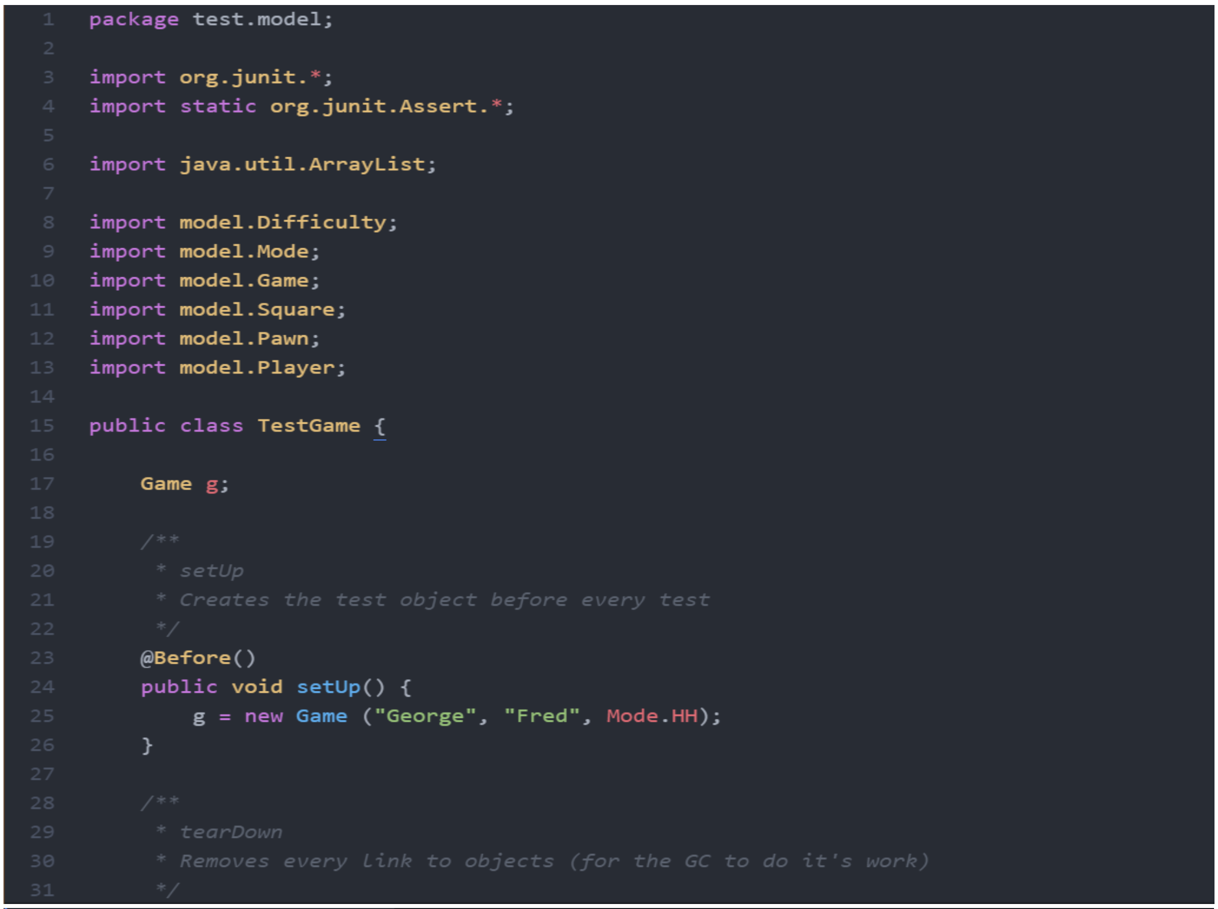


```java
package test.model;

import org.junit.*;
import static org.junit.Assert.*;

import model.GameConfig;

public class TestGameConfig {

    GameConfig g;



    /**
     * setUp
     * Creates the test object before every test
     */
    @Before()
    public void setUp() {
        g = new GameConfig();
    }

    /**
     * tearDown
     * Removes every link to objects (for the GC to do it's work)
     */
    @After()
    public void tearDown() {
        g = null;
    }

```

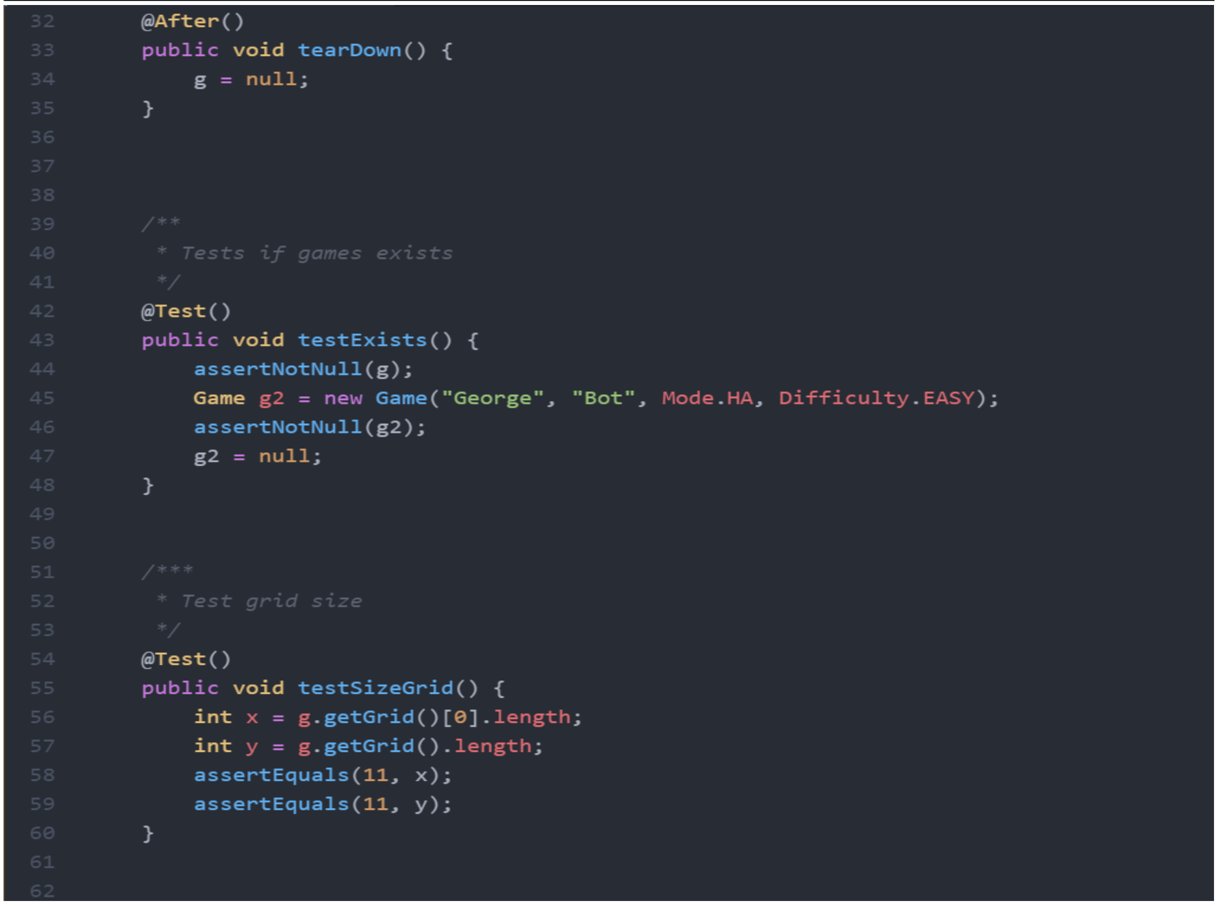


```java


    /**
     * Tests if the pickStartPlayer works
     * Doing 4 sepreate tests, to have little chance of getting 4 times the same result
     */
    @Test()
    public void testPickStartPlayer() {
        Boolean b1 = g.pickStartPlayer();
        Boolean b2 = g.pickStartPlayer();
        Boolean b3 = g.pickStartPlayer();
        Boolean b4 = g.pickStartPlayer();
        if (b1 == b2 && b2 == b3 && b3 == b4) {
            assertEquals(true, true);
        }
        else {
            assertEquals(true, false);
        }
    }

}
```

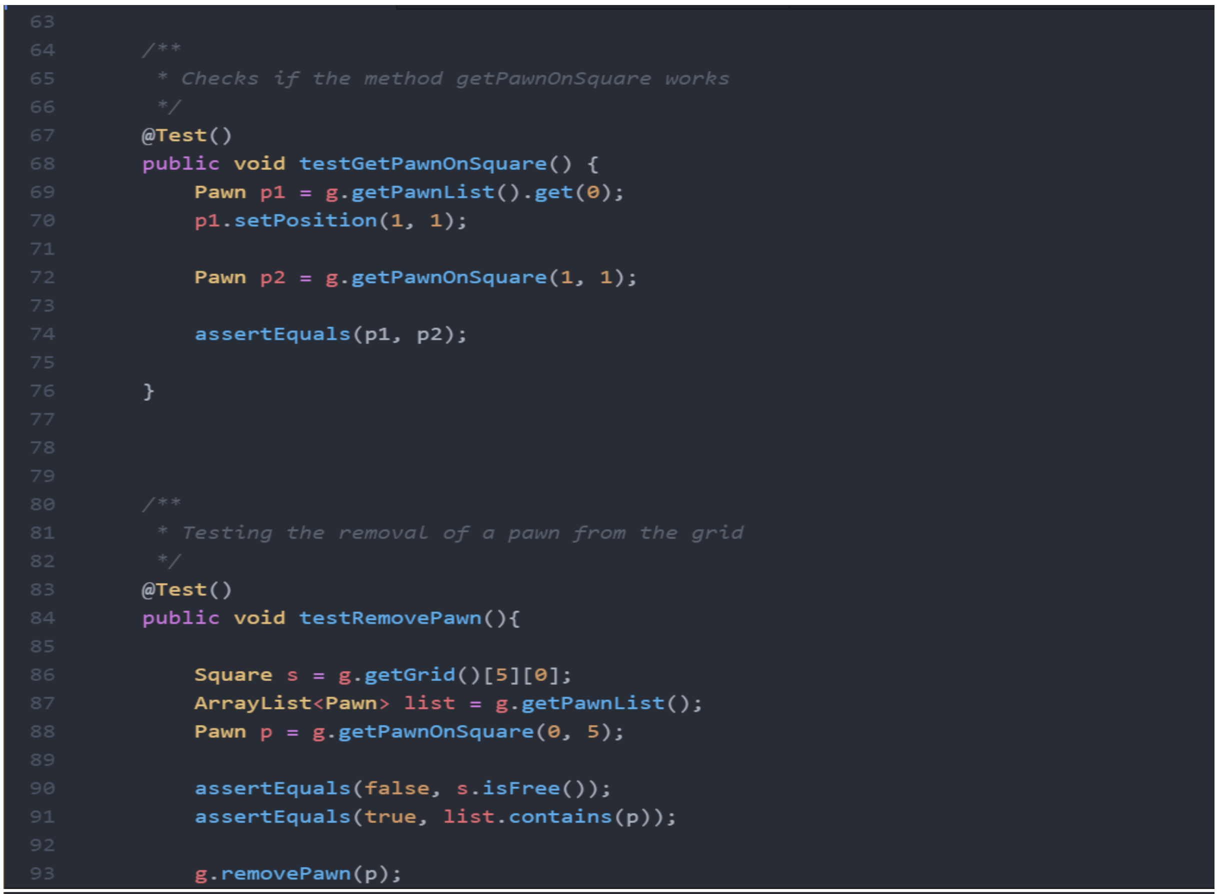
2. TestGame



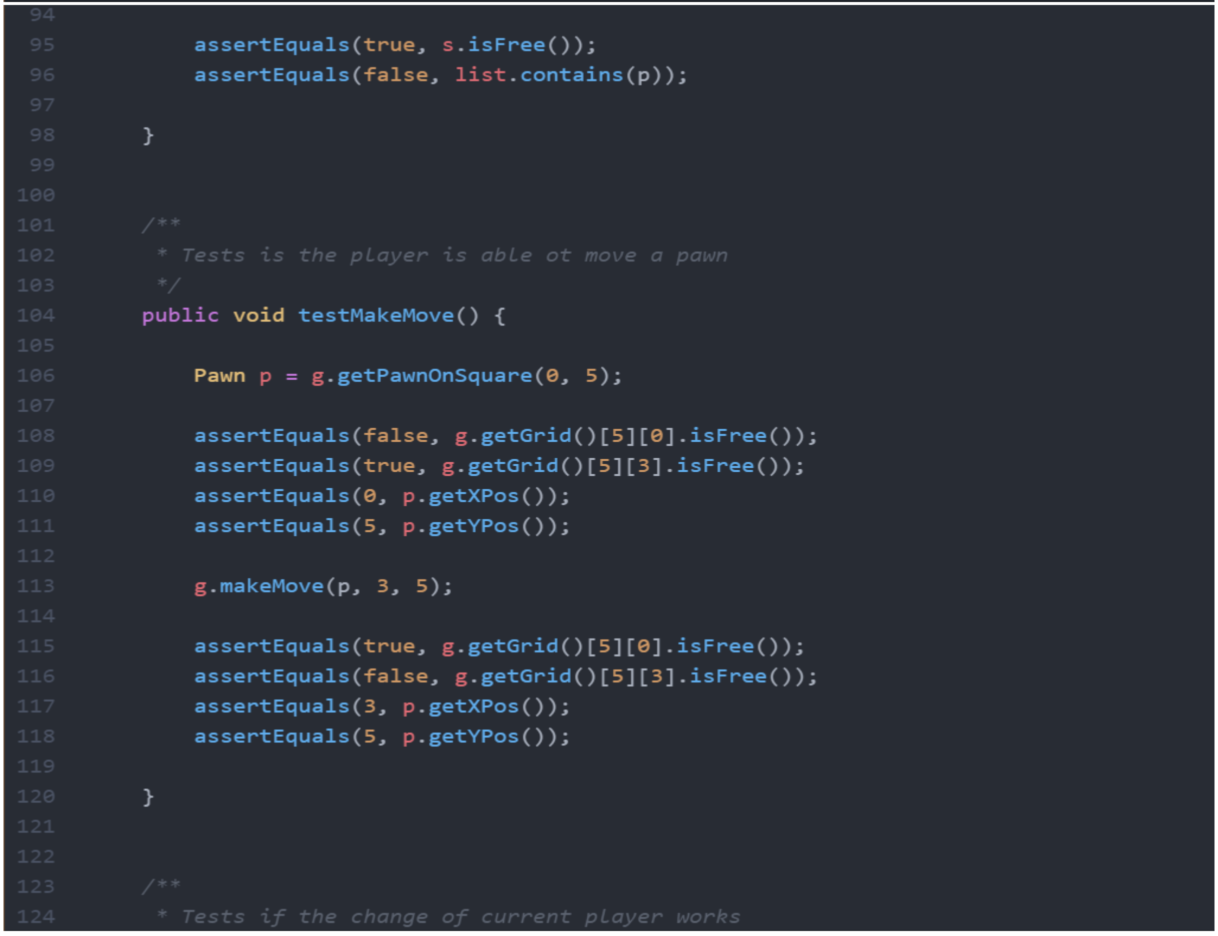
```java
package test.model;

import org.junit.*;
import static org.junit.Assert.*;

import java.util.ArrayList;

import model.Difficulty;
import model.Mode;
import model.Game;
import model.Square;
import model.Pawn;
import model.Player;

public class TestGame {

    Game g;

    /**
     * setUp
     * Creates the test object before every test
     */
    @Before()
    public void setUp() {
        g = new Game ("George", "Fred", Mode.HH);
    }

    /**
     * tearDown
     * Removes every link to objects (for the GC to do it's work)
     */
```



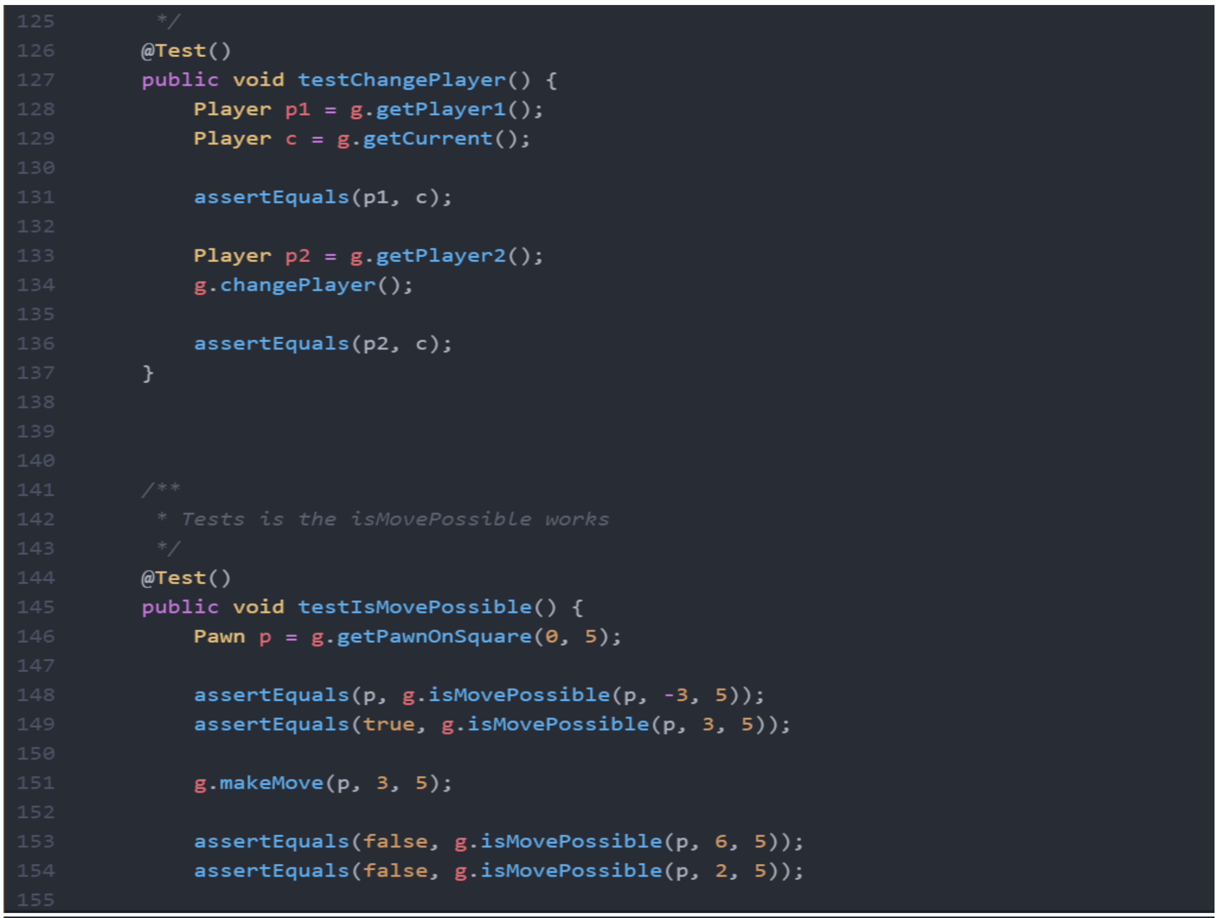
```java
    @After()
    public void tearDown() {
        g = null;
    }



    /**
     * Tests if games exists
     */
    @Test()
    public void testExists() {
        assertNotNull(g);
        Game g2 = new Game("George", "Bot", Mode.HA, Difficulty.EASY);
        assertNotNull(g2);
        g2 = null;
    }


    /***
     * Test grid size
     */
    @Test()
    public void testSizeGrid() {
        int x = g.getGrid()[0].length;
        int y = g.getGrid().length;
        assertEquals(11, x);
        assertEquals(11, y);
    }


```



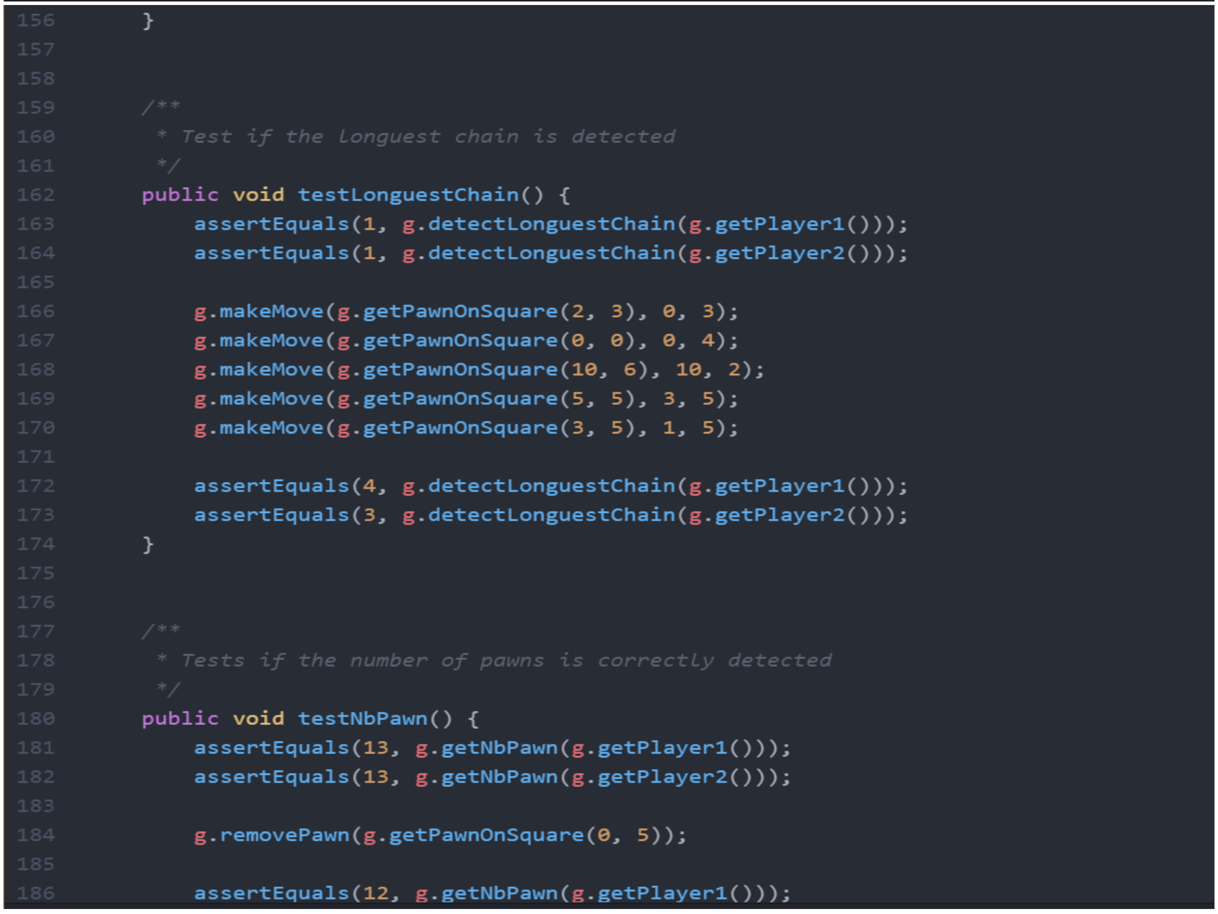
```java

    /**
     * Checks if the method getPawnOnSquare works
     */
    @Test()
    public void testGetPawnOnSquare() {
        Pawn p1 = g.getPawnList().get(0);
        p1.setPosition(1, 1);

        Pawn p2 = g.getPawnOnSquare(1, 1);

        assertEquals(p1, p2);

    }



    /**
     * Testing the removal of a pawn from the grid
     */
    @Test()
    public void testRemovePawn(){

        Square s = g.getGrid()[5][0];
        ArrayList<Pawn> list = g.getPawnList();
        Pawn p = g.getPawnOnSquare(0, 5);

        assertEquals(false, s.isFree());
        assertEquals(true, list.contains(p));

        g.removePawn(p);
```



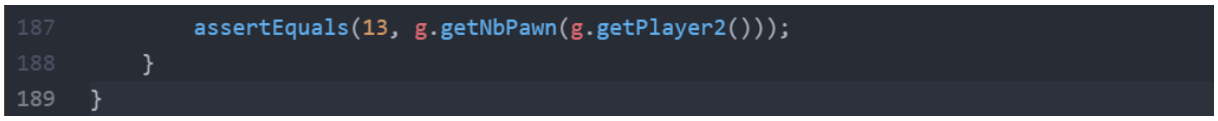
```java

        assertEquals(true, s.isFree());
        assertEquals(false, list.contains(p));

    }


    /**
     * Tests is the player is able ot move a pawn
     */
    public void testMakeMove() {

        Pawn p = g.getPawnOnSquare(0, 5);

        assertEquals(false, g.getGrid()[5][0].isFree());
        assertEquals(true, g.getGrid()[5][3].isFree());
        assertEquals(0, p.getXPos());
        assertEquals(5, p.getYPos());

        g.makeMove(p, 3, 5);

        assertEquals(true, g.getGrid()[5][0].isFree());
        assertEquals(false, g.getGrid()[5][3].isFree());
        assertEquals(3, p.getXPos());
        assertEquals(5, p.getYPos());

    }


    /**
     * Tests if the change of current player works
```



```
     */
    @Test()
    public void testChangePlayer() {
        Player p1 = g.getPlayer1();
        Player c = g.getCurrent();

        assertEquals(p1, c);

        Player p2 = g.getPlayer2();
        g.changePlayer();

        assertEquals(p2, c);
    }



    /**
     * Tests is the isMovePossible works
     */
    @Test()
    public void testIsMovePossible() {
        Pawn p = g.getPawnOnSquare(0, 5);

        assertEquals(p, g.isMovePossible(p, -3, 5));
        assertEquals(true, g.isMovePossible(p, 3, 5));

        g.makeMove(p, 3, 5);

        assertEquals(false, g.isMovePossible(p, 6, 5));
        assertEquals(false, g.isMovePossible(p, 2, 5));

```

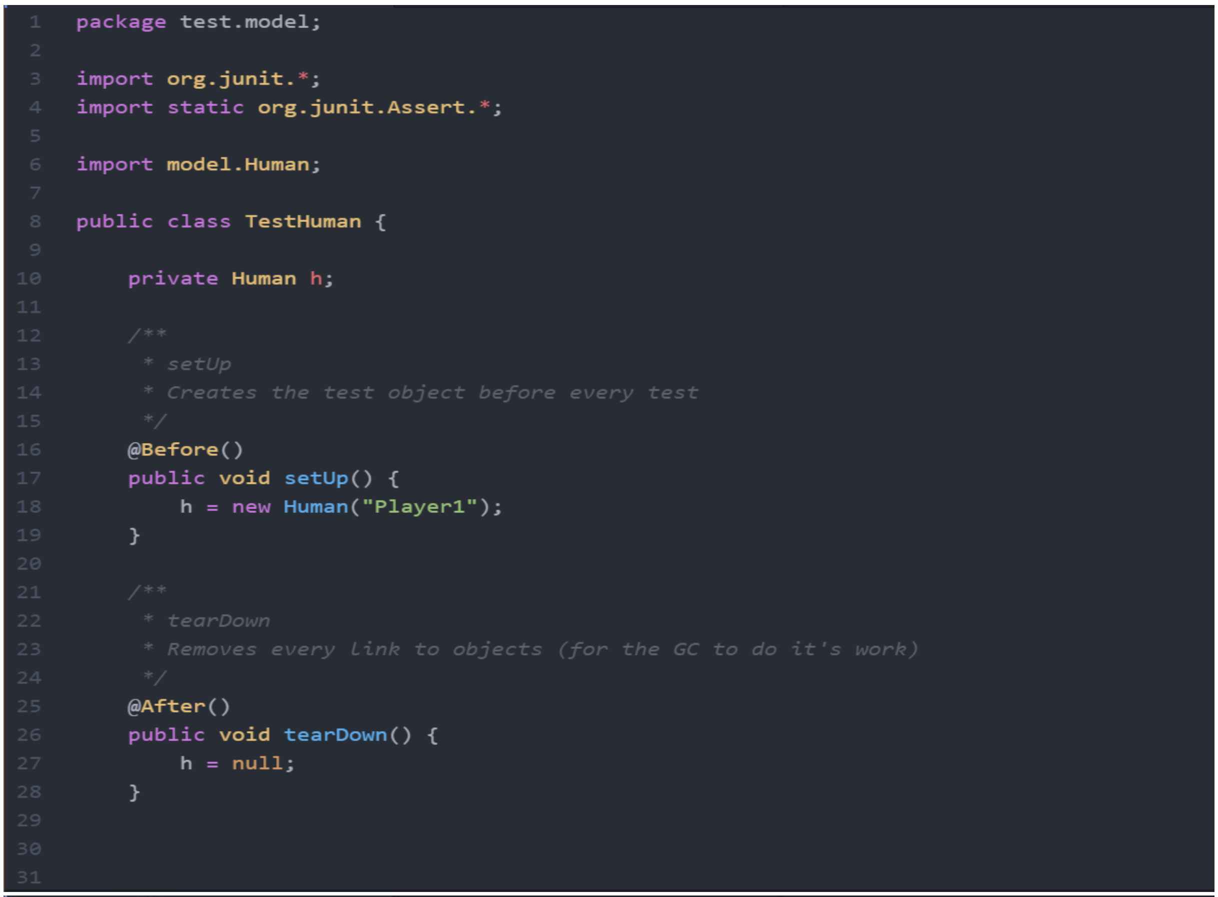


```
    }


    /**
     * Test if the longuest chain is detected
     */
    public void testLonguestChain() {
        assertEquals(1, g.detectLonguestChain(g.getPlayer1()));
        assertEquals(1, g.detectLonguestChain(g.getPlayer2()));

        g.makeMove(g.getPawnOnSquare(2, 3), 0, 3);
        g.makeMove(g.getPawnOnSquare(0, 0), 0, 4);
        g.makeMove(g.getPawnOnSquare(10, 6), 10, 2);
        g.makeMove(g.getPawnOnSquare(5, 5), 3, 5);
        g.makeMove(g.getPawnOnSquare(3, 5), 1, 5);

        assertEquals(4, g.detectLonguestChain(g.getPlayer1()));
        assertEquals(3, g.detectLonguestChain(g.getPlayer2()));
    }


    /**
     * Tests if the number of pawns is correctly detected
     */
    public void testNbPawn() {
        assertEquals(13, g.getNbPawn(g.getPlayer1()));
        assertEquals(13, g.getNbPawn(g.getPlayer2()));

        g.removePawn(g.getPawnOnSquare(0, 5));

        assertEquals(12, g.getNbPawn(g.getPlayer1()));
```

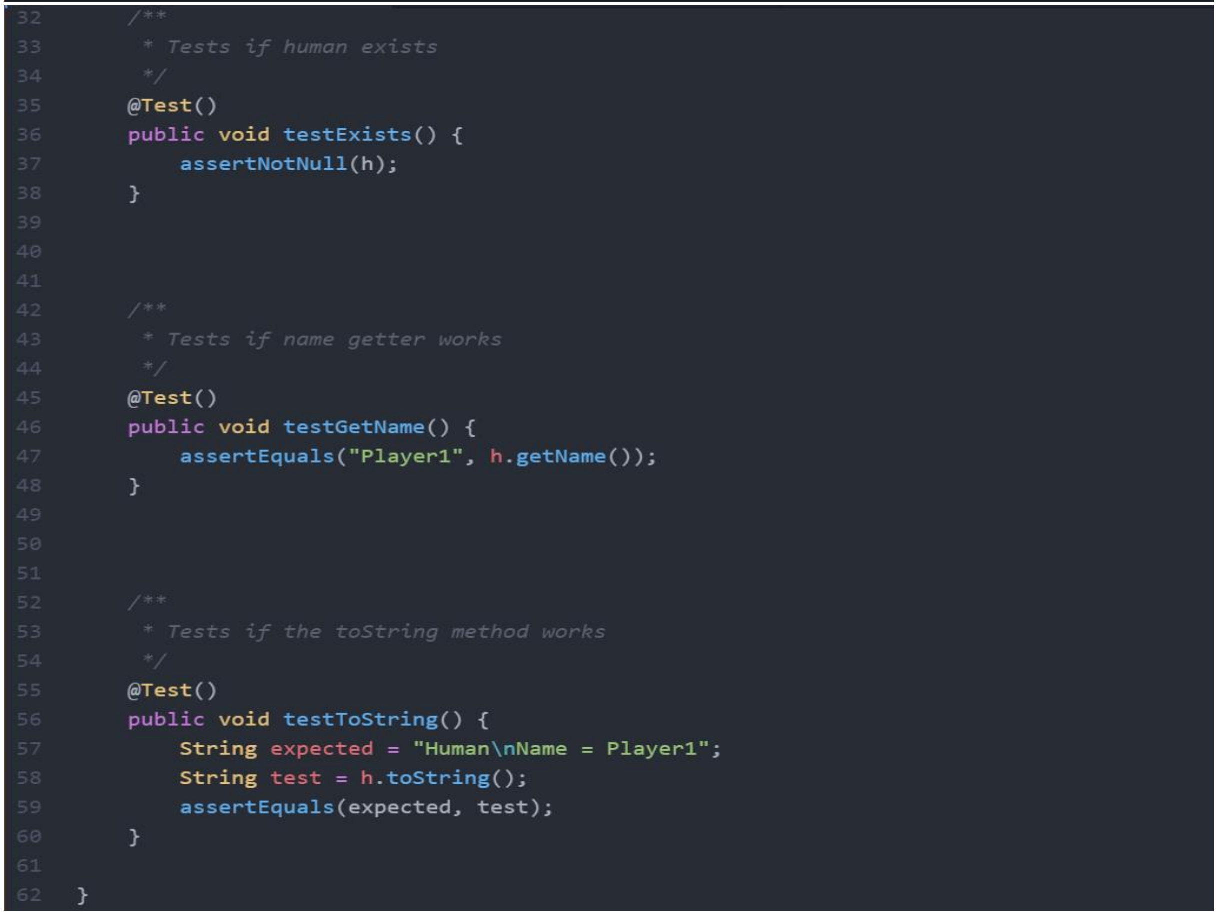


```
        assertEquals(13, g.getNbPawn(g.getPlayer2()));
    }
}
```

3. TestHuman

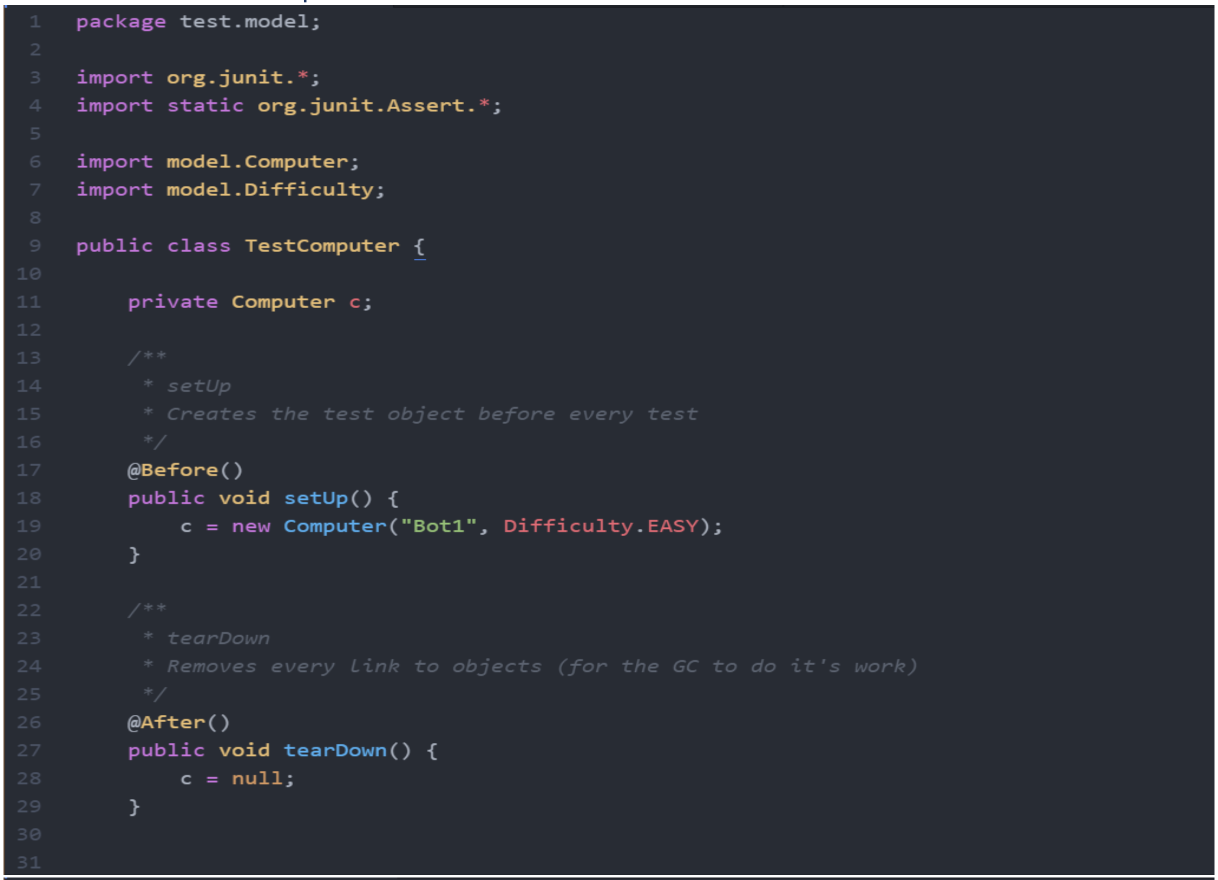


```java
package test.model;

import org.junit.*;
import static org.junit.Assert.*;

import model.Human;

public class TestHuman {

    private Human h;

    /**
     * setUp
     * Creates the test object before every test
     */
    @Before()
    public void setUp() {
        h = new Human("Player1");
    }

    /**
     * tearDown
     * Removes every link to objects (for the GC to do it's work)
     */
    @After()
    public void tearDown() {
        h = null;
    }



```

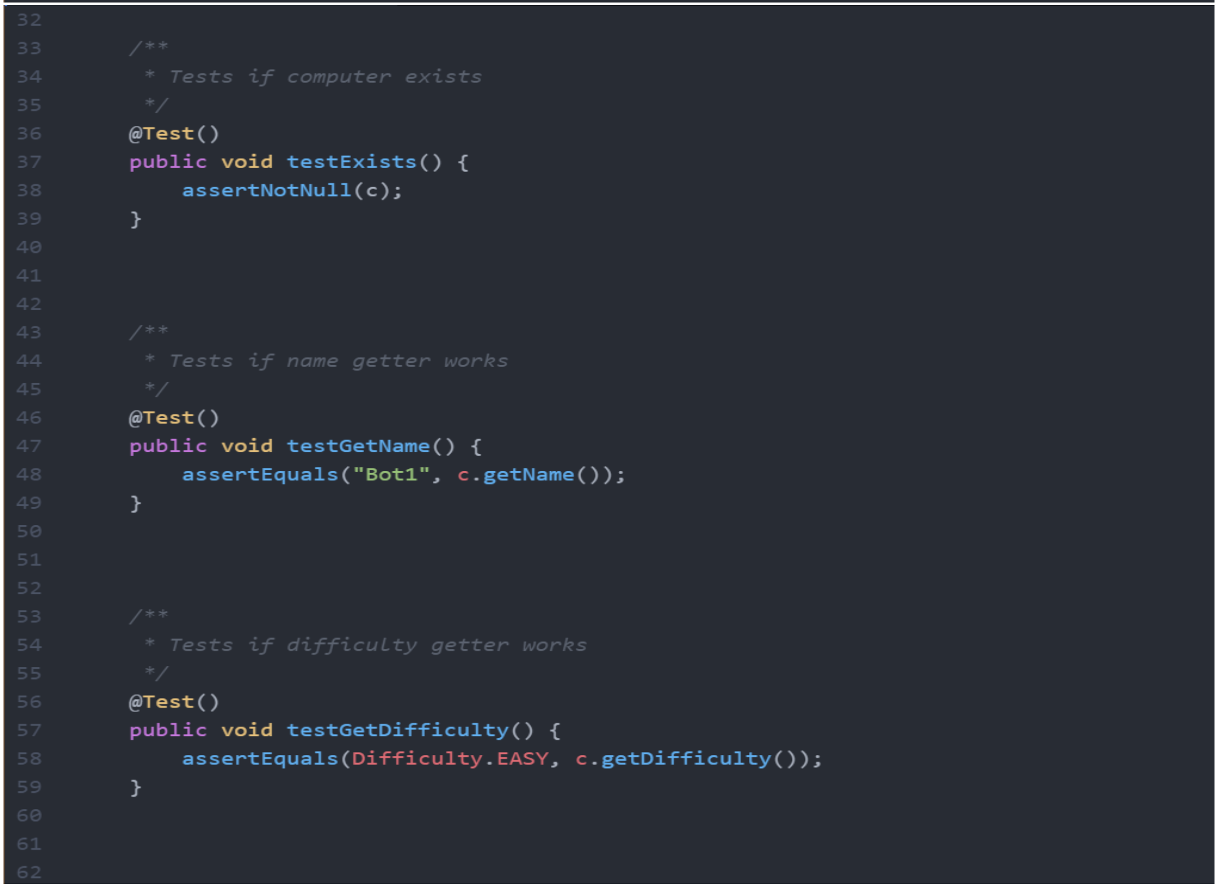


```java
    /**
     * Tests if human exists
     */
    @Test()
    public void testExists() {
        assertNotNull(h);
    }



    /**
     * Tests if name getter works
     */
    @Test()
    public void testGetName() {
        assertEquals("Player1", h.getName());
    }



    /**
     * Tests if the toString method works
     */
    @Test()
    public void testToString() {
        String expected = "Human\nName = Player1";
        String test = h.toString();
        assertEquals(expected, test);
    }

}
```

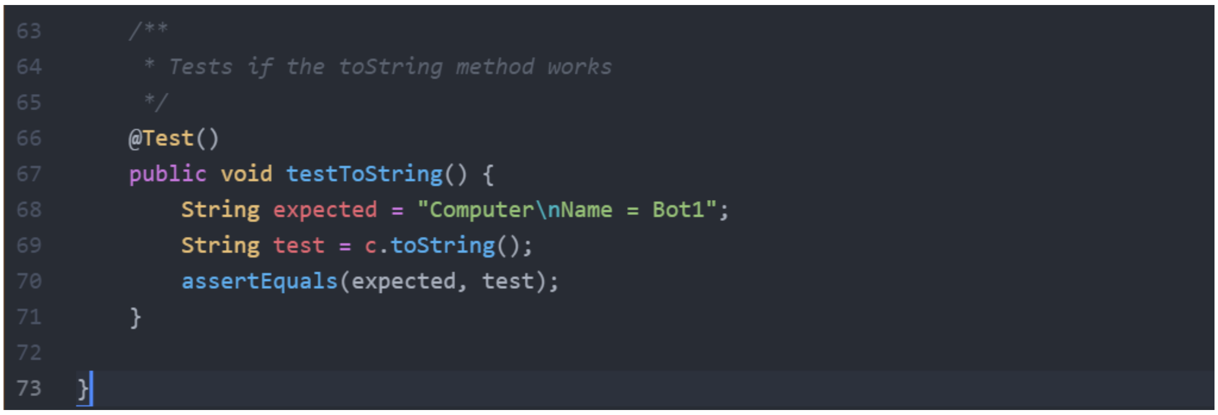
4. TestComputer



```java
package test.model;

import org.junit.*;
import static org.junit.Assert.*;

import model.Computer;
import model.Difficulty;

public class TestComputer {

    private Computer c;

    /**
     * setUp
     * Creates the test object before every test
     */
    @Before()
    public void setUp() {
        c = new Computer("Bot1", Difficulty.EASY);
    }

    /**
     * tearDown
     * Removes every link to objects (for the GC to do it's work)
     */
    @After()
    public void tearDown() {
        c = null;
    }


```

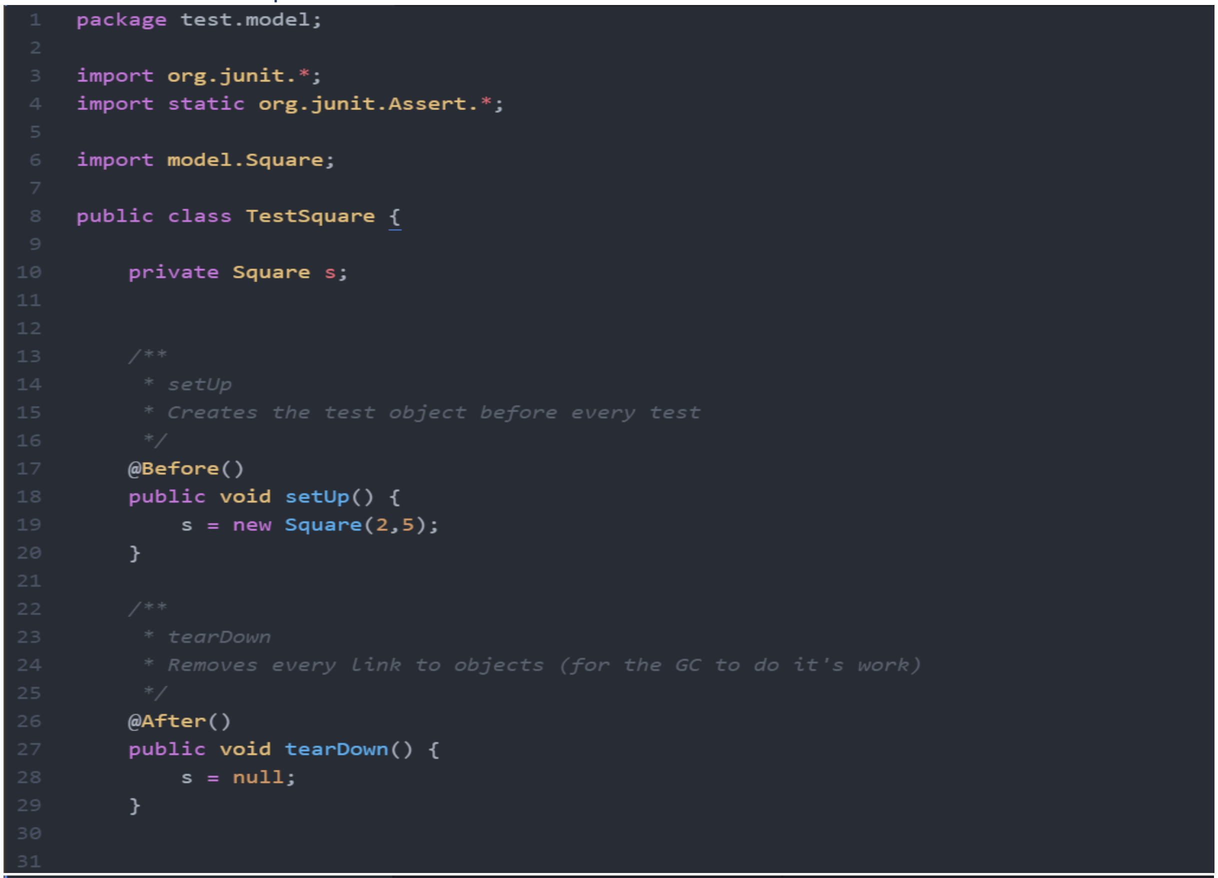


```java

    /**
     * Tests if computer exists
     */
    @Test()
    public void testExists() {
        assertNotNull(c);
    }



    /**
     * Tests if name getter works
     */
    @Test()
    public void testGetName() {
        assertEquals("Bot1", c.getName());
    }



    /**
     * Tests if difficulty getter works
     */
    @Test()
    public void testGetDifficulty() {
        assertEquals(Difficulty.EASY, c.getDifficulty());
    }



```

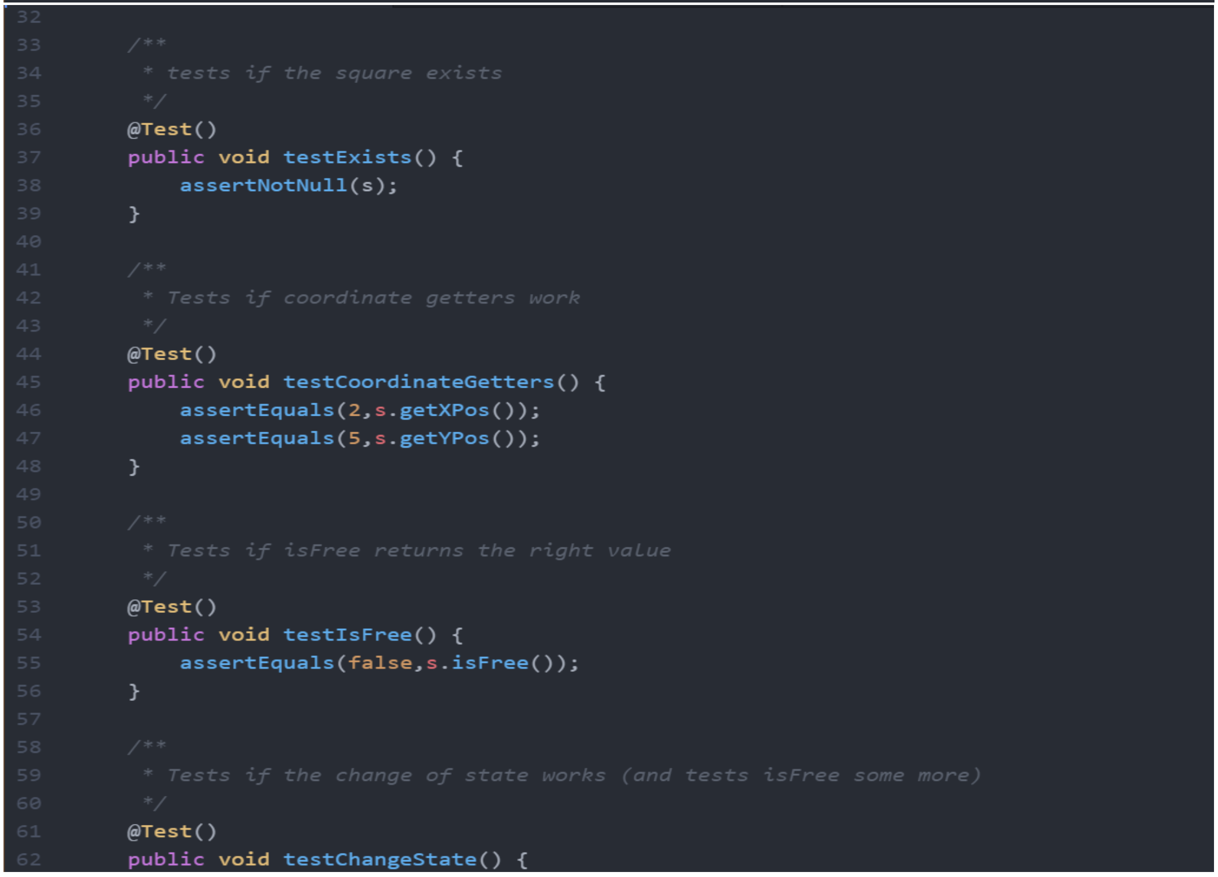


```
    /**
     * Tests if the toString method works
     */
    @Test()
    public void testToString() {
        String expected = "Computer\nName = Bot1";
        String test = c.toString();
        assertEquals(expected, test);
    }

}
```

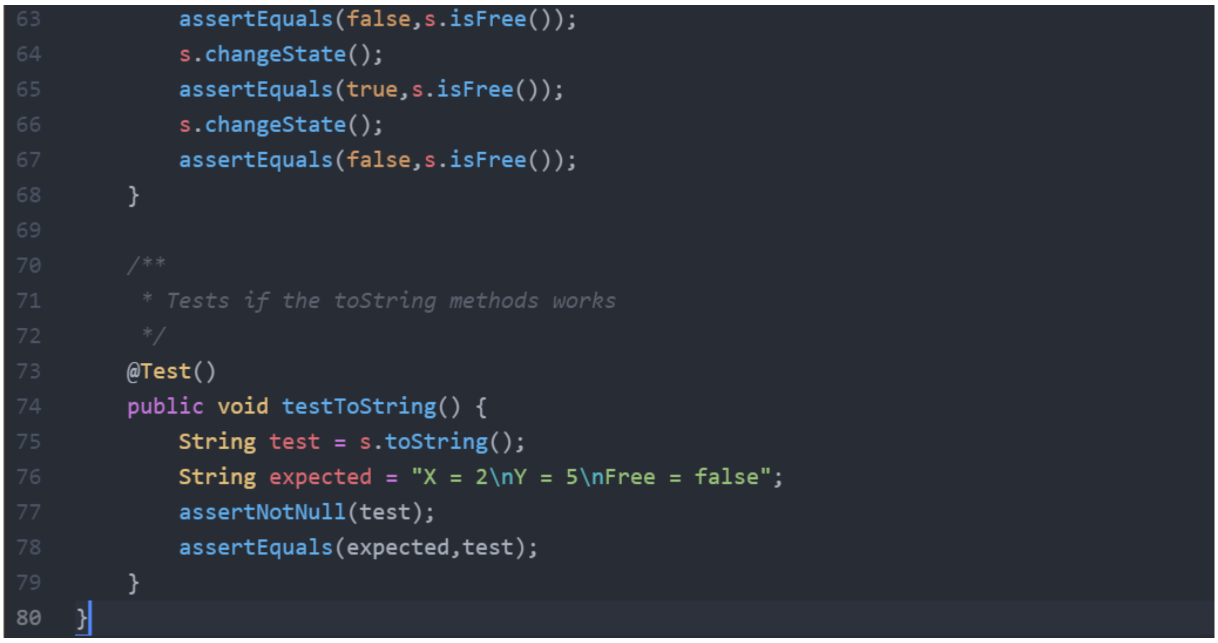
5. TestSquare



```java
package test.model;

import org.junit.*;
import static org.junit.Assert.*;

import model.Square;

public class TestSquare {

    private Square s;


    /**
     * setUp
     * Creates the test object before every test
     */
    @Before()
    public void setUp() {
        s = new Square(2,5);
    }

    /**
     * tearDown
     * Removes every link to objects (for the GC to do it's work)
     */
    @After()
    public void tearDown() {
        s = null;
    }


```

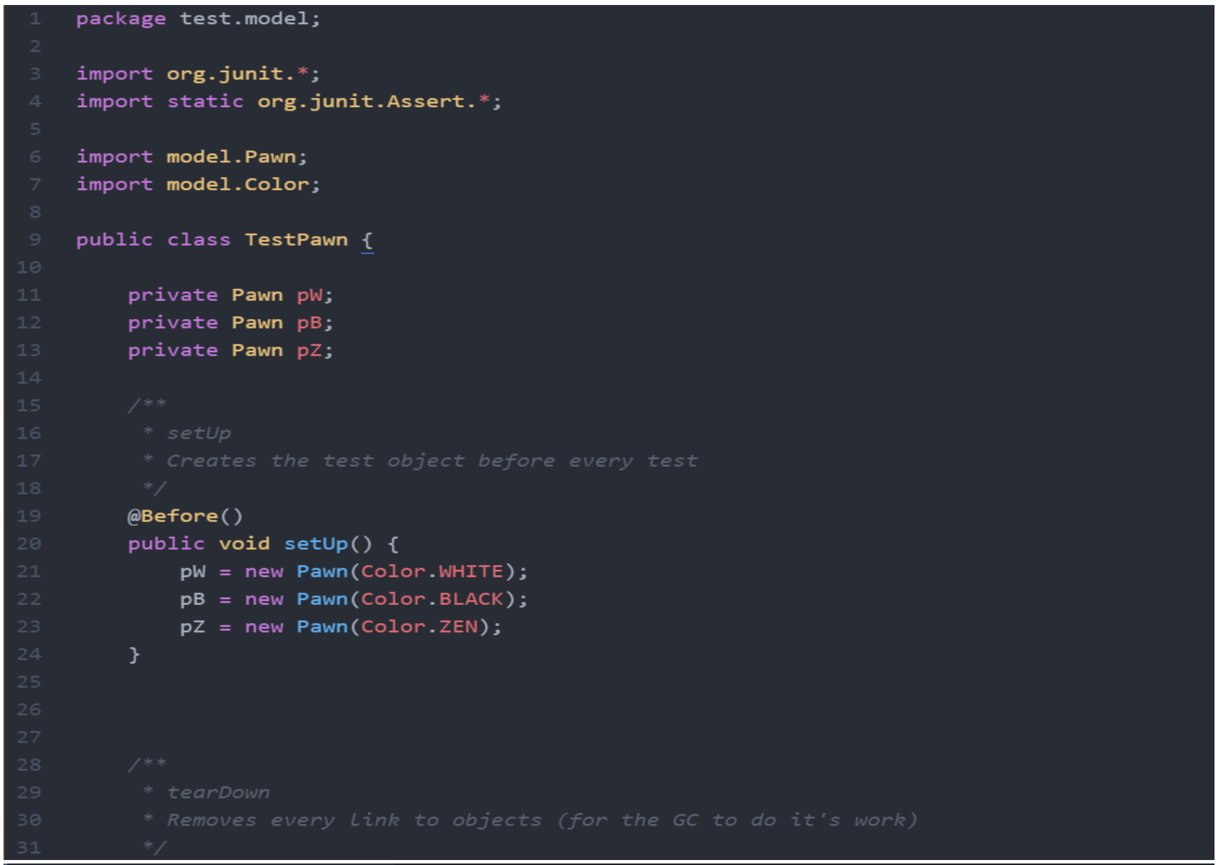


```java

    /**
     * tests if the square exists
     */
    @Test()
    public void testExists() {
        assertNotNull(s);
    }

    /**
     * Tests if coordinate getters work
     */
    @Test()
    public void testCoordinateGetters() {
        assertEquals(2,s.getXPos());
        assertEquals(5,s.getYPos());
    }

    /**
     * Tests if isFree returns the right value
     */
    @Test()
    public void testIsFree() {
        assertEquals(false,s.isFree());
    }

    /**
     * Tests if the change of state works (and tests isFree some more)
     */
    @Test()
    public void testChangeState() {
```

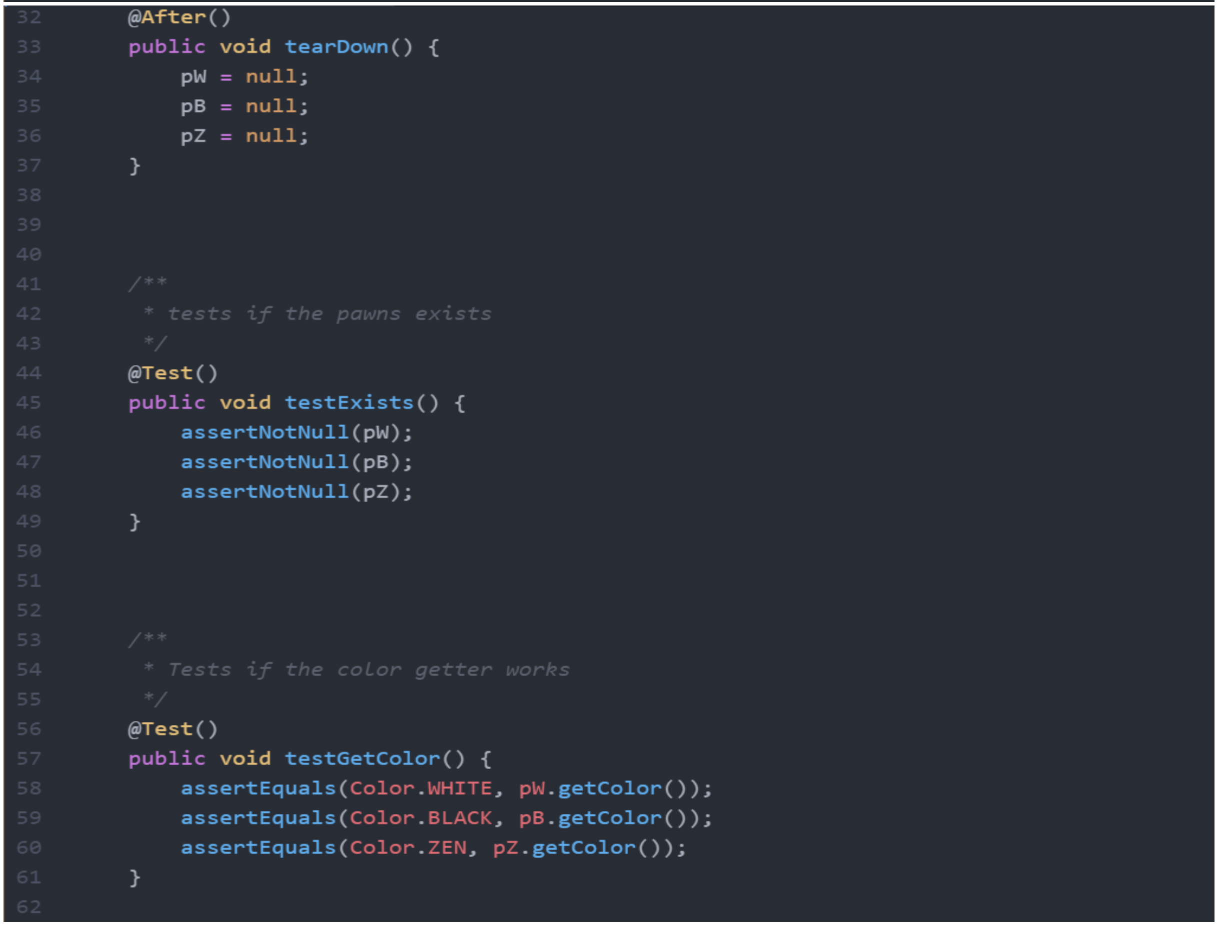


```
        assertEquals(false,s.isFree());
        s.changeState();
        assertEquals(true,s.isFree());
        s.changeState();
        assertEquals(false,s.isFree());
    }

    /**
     * Tests if the toString methods works
     */
    @Test()
    public void testToString() {
        String test = s.toString();
        String expected = "X = 2\nY = 5\nFree = false";
        assertNotNull(test);
        assertEquals(expected,test);
    }
}
```

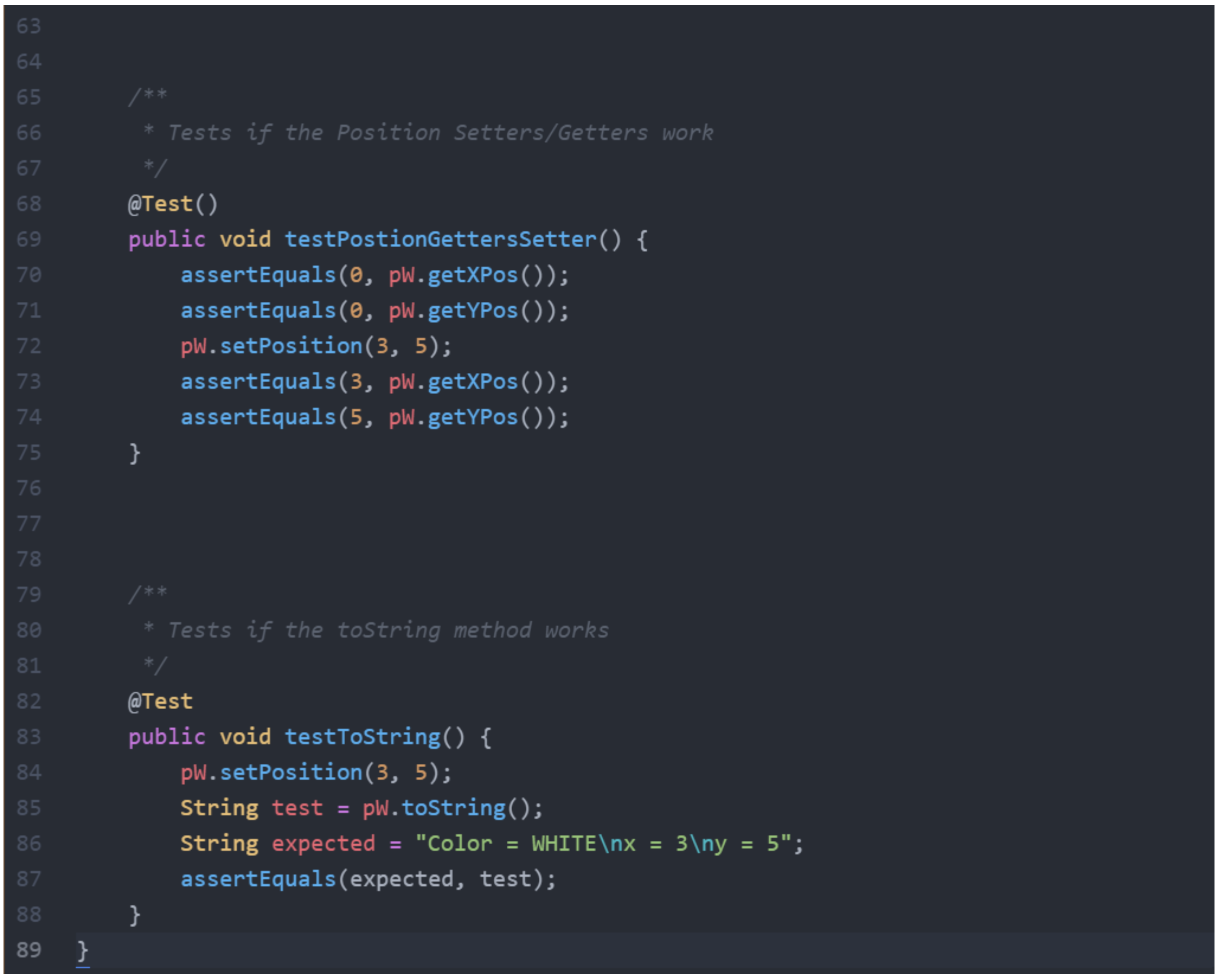
6. TestPawn



```java
package test.model;

import org.junit.*;
import static org.junit.Assert.*;

import model.Pawn;
import model.Color;

public class TestPawn {

    private Pawn pW;
    private Pawn pB;
    private Pawn pZ;

    /**
     * setUp
     * Creates the test object before every test
     */
    @Before()
    public void setUp() {
        pW = new Pawn(Color.WHITE);
        pB = new Pawn(Color.BLACK);
        pZ = new Pawn(Color.ZEN);
    }



    /**
     * tearDown
     * Removes every link to objects (for the GC to do it's work)
     */
```



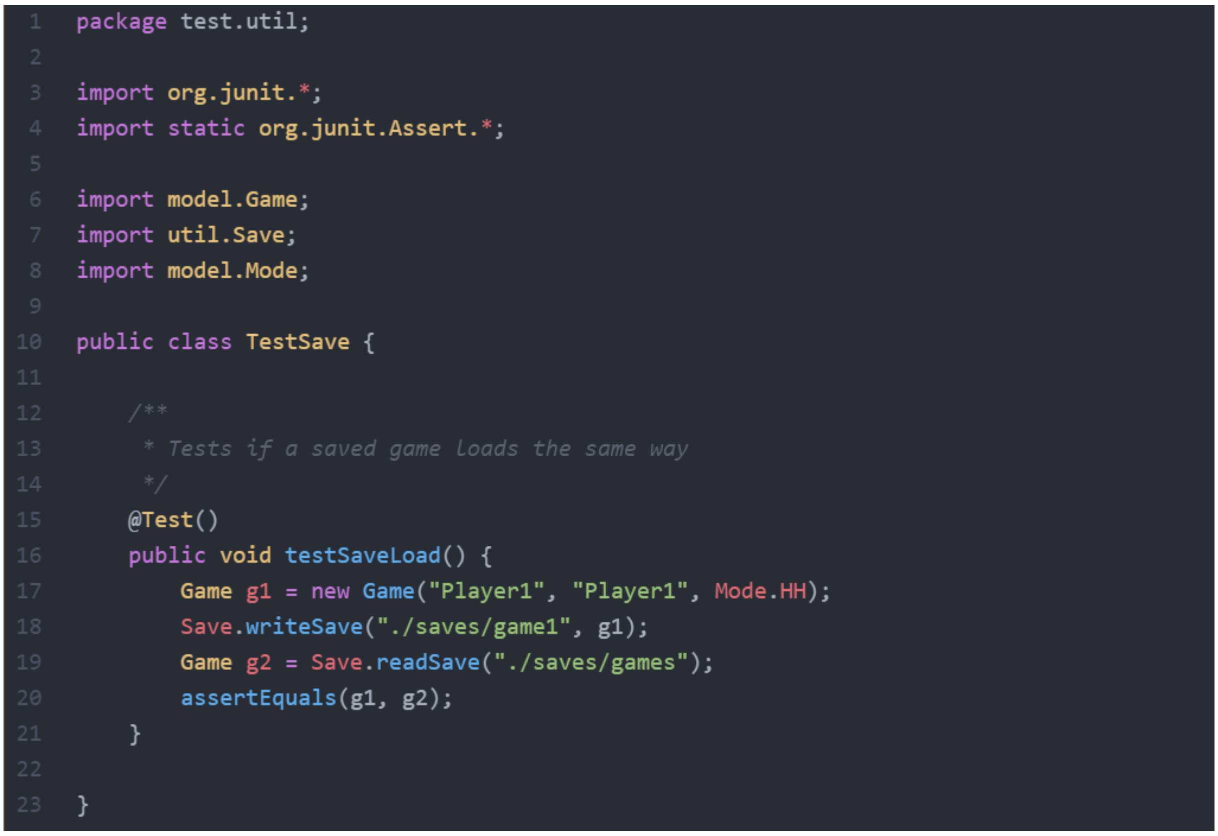
```java
    @After()
    public void tearDown() {
        pW = null;
        pB = null;
        pZ = null;
    }



    /**
     * tests if the pawns exists
     */
    @Test()
    public void testExists() {
        assertNotNull(pW);
        assertNotNull(pB);
        assertNotNull(pZ);
    }



    /**
     * Tests if the color getter works
     */
    @Test()
    public void testGetColor() {
        assertEquals(Color.WHITE, pW.getColor());
        assertEquals(Color.BLACK, pB.getColor());
        assertEquals(Color.ZEN, pZ.getColor());
    }

```



```


    /**
     * Tests if the Position Setters/Getters work
     */
    @Test()
    public void testPostionGettersSetter() {
        assertEquals(0, pW.getXPos());
        assertEquals(0, pW.getYPos());
        pW.setPosition(3, 5);
        assertEquals(3, pW.getXPos());
        assertEquals(5, pW.getYPos());
    }



    /**
     * Tests if the toString method works
     */
    @Test
    public void testToString() {
        pW.setPosition(3, 5);
        String test = pW.toString();
        String expected = "Color = WHITE\nx = 3\ny = 5";
        assertEquals(expected, test);
    }
}
```

b) Test du package util

1. TestSave

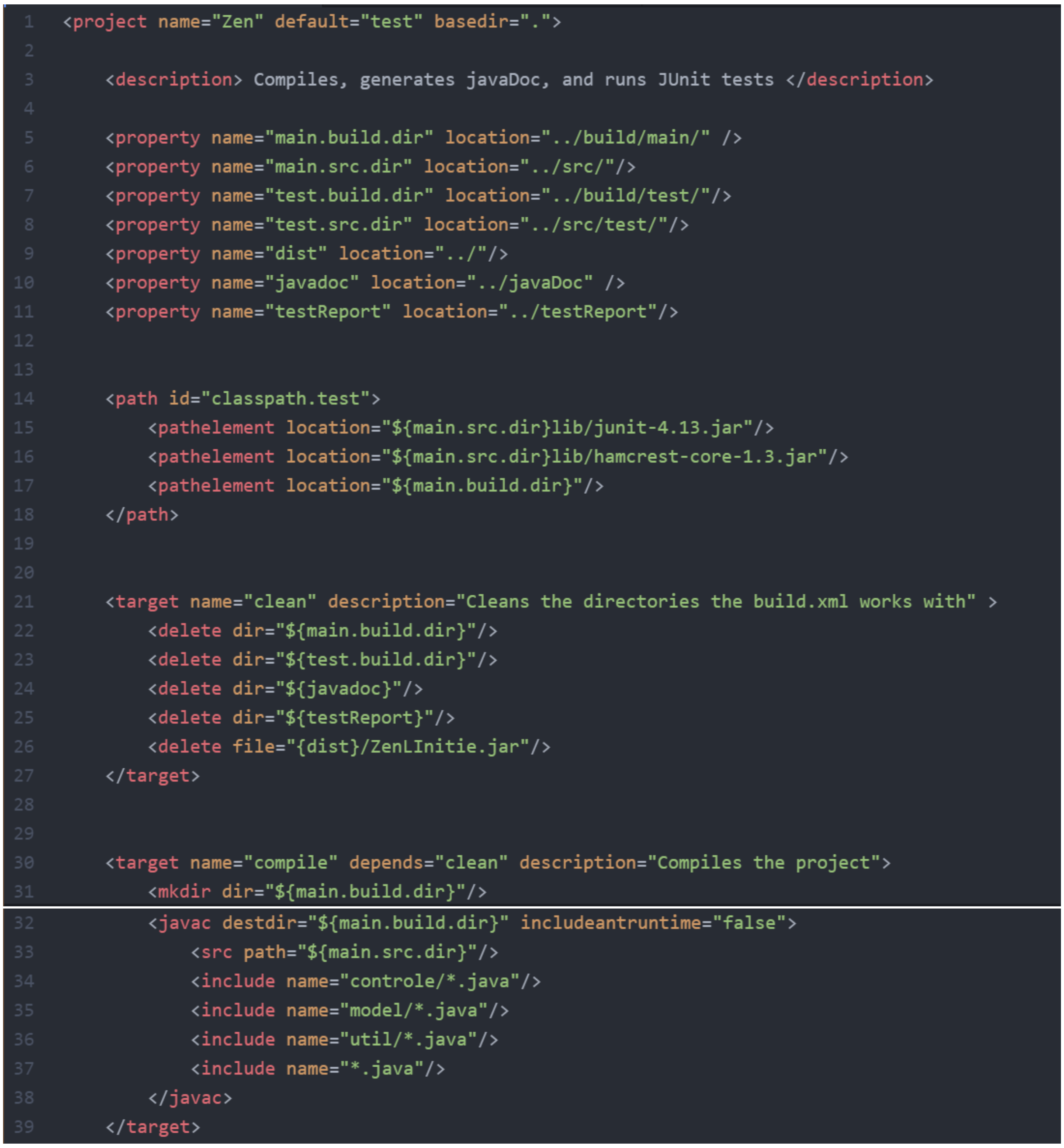


```java
package test.util;

import org.junit.*;
import static org.junit.Assert.*;

import model.Game;
import util.Save;
import model.Mode;

public class TestSave {

    /**
     * Tests if a saved game loads the same way
     */
    @Test()
    public void testSaveLoad() {
        Game g1 = new Game("Player1", "Player1", Mode.HH);
        Save.writeSave("./saves/game1", g1);
        Game g2 = Save.readSave("./saves/games");
        assertEquals(g1, g2);
    }

}
```

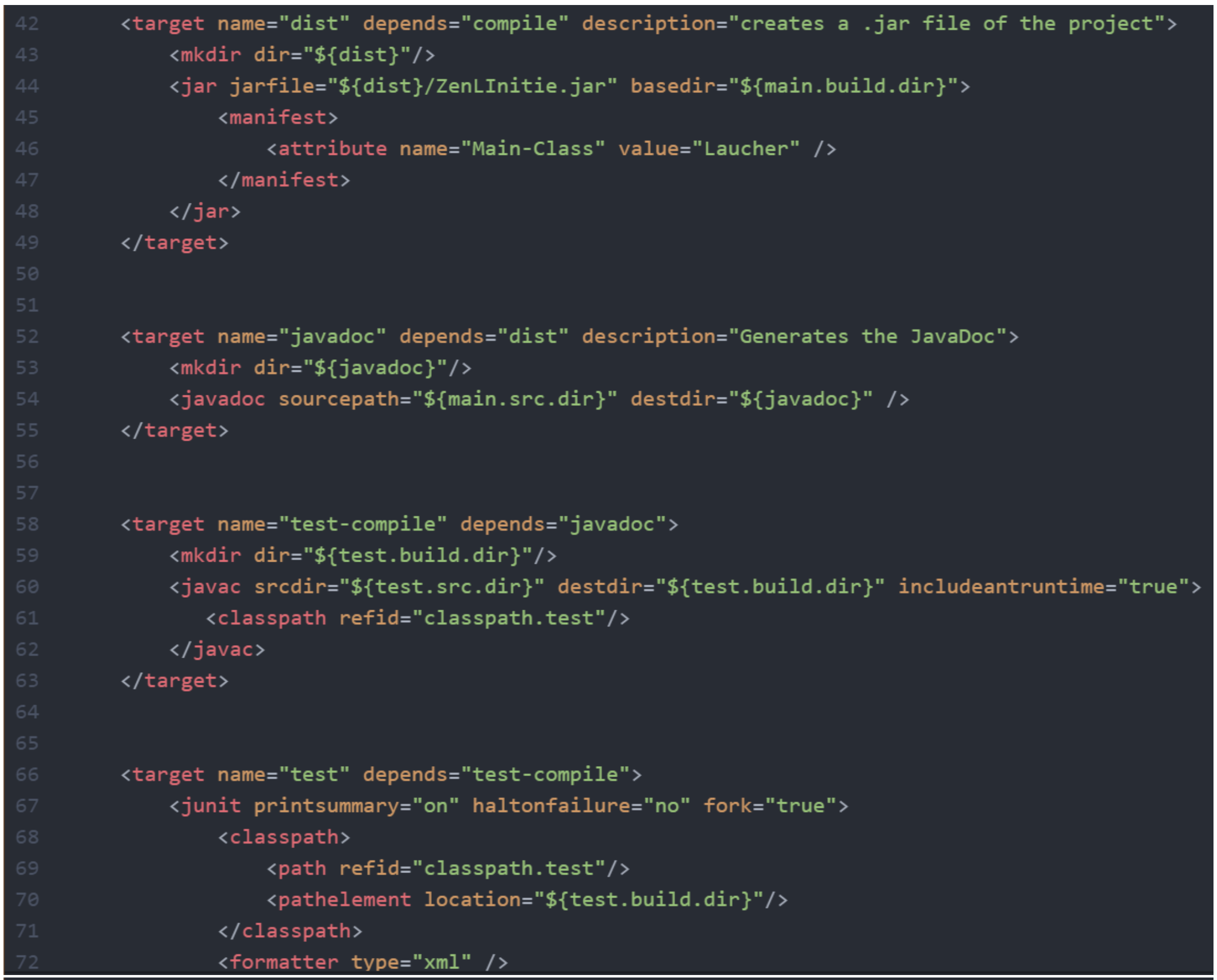
7) ANT

Il est possible grâce à l'outil ANT, d'automatiser une partie des tâches liées au développement. Le fichier build.xml permet de décrire à ANT les actions à effectuer. Il fait, dans l'ordre :

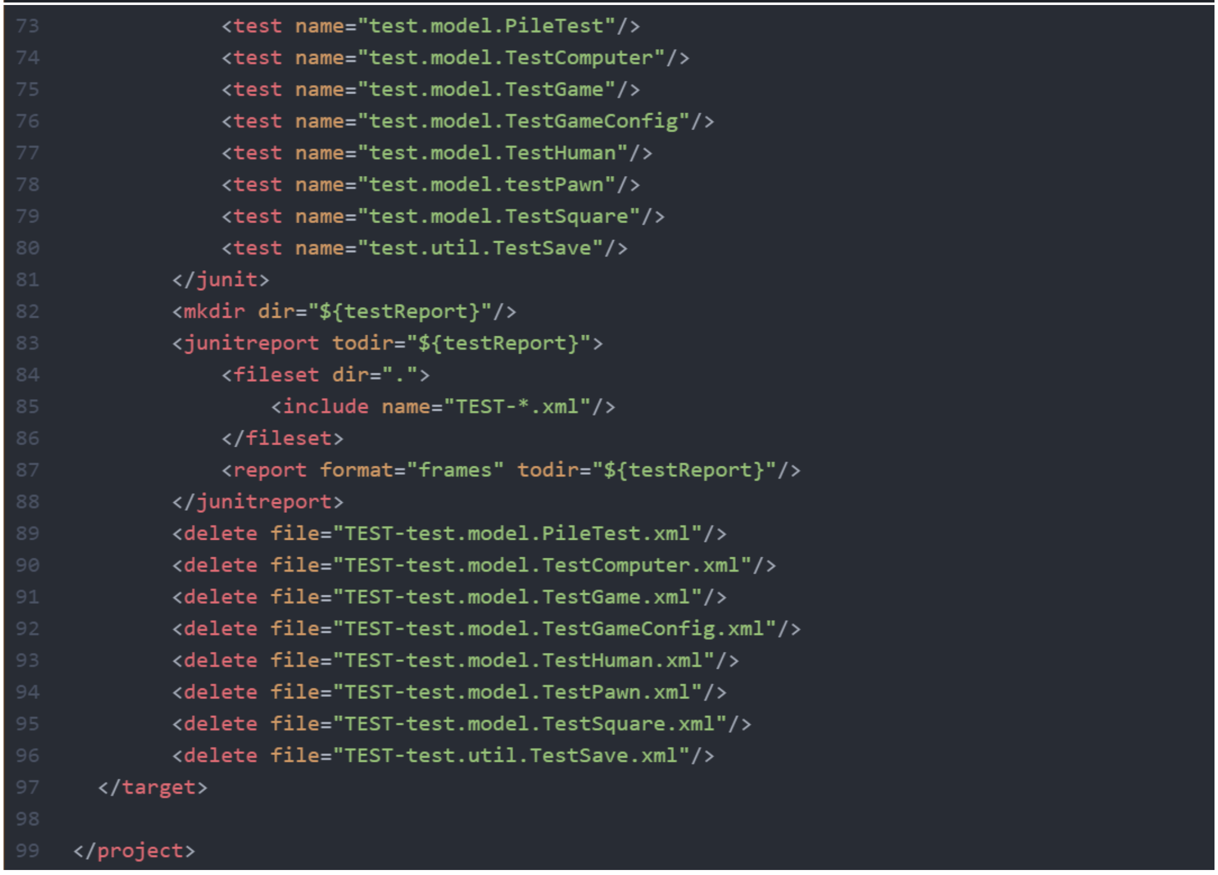
- De nettoyer les dossiers du projet (qui seront regénérés après)
- De compiler les classes de l'application
- A générer un exécutable (.jar)
- A générer la JavaDoc
- A compiler les classes de tests
- A lancer les tests, et produire un rapport de ces derniers



```
<project name="Zen" default="test" basedir=".">

    <description> Compiles, generates javaDoc, and runs JUnit tests </description>

    <property name="main.build.dir" location="../build/main/" />
    <property name="main.src.dir" location="../src/"/>
    <property name="test.build.dir" location="../build/test/"/>
    <property name="test.src.dir" location="../src/test/"/>
    <property name="dist" location="../"/>
    <property name="javadoc" location="../javaDoc" />
    <property name="testReport" location="../testReport"/>


    <path id="classpath.test">
        <pathelement location="${main.src.dir}lib/junit-4.13.jar"/>
        <pathelement location="${main.src.dir}lib/hamcrest-core-1.3.jar"/>
        <pathelement location="${main.build.dir}"/>
    </path>


    <target name="clean" description="Cleans the directories the build.xml works with" >
        <delete dir="${main.build.dir}"/>
        <delete dir="${test.build.dir}"/>
        <delete dir="${javadoc}"/>
        <delete dir="${testReport}"/>
        <delete file="{dist}/ZenLInitie.jar"/>
    </target>


    <target name="compile" depends="clean" description="Compiles the project">
        <mkdir dir="${main.build.dir}"/>
        <javac destdir="${main.build.dir}" includeantruntime="false">
            <src path="${main.src.dir}"/>
            <include name="controle/*.java"/>
            <include name="model/*.java"/>
            <include name="util/*.java"/>
            <include name="*.java"/>
        </javac>
    </target>
```



```
    <target name="dist" depends="compile" description="creates a .jar file of the project">
        <mkdir dir="${dist}"/>
        <jar jarfile="${dist}/ZenLInitie.jar" basedir="${main.build.dir}">
            <manifest>
                <attribute name="Main-Class" value="Laucher" />
            </manifest>
        </jar>
    </target>


    <target name="javadoc" depends="dist" description="Generates the JavaDoc">
        <mkdir dir="${javadoc}"/>
        <javadoc sourcepath="${main.src.dir}" destdir="${javadoc}" />
    </target>


    <target name="test-compile" depends="javadoc">
        <mkdir dir="${test.build.dir}"/>
        <javac srcdir="${test.src.dir}" destdir="${test.build.dir}" includeantruntime="true">
           <classpath refid="classpath.test"/>
        </javac>
    </target>


    <target name="test" depends="test-compile">
        <junit printsummary="on" haltonfailure="no" fork="true">
            <classpath>
                <path refid="classpath.test"/>
                <pathelement location="${test.build.dir}"/>
            </classpath>
            <formatter type="xml" />
```



```
            <test name="test.model.PileTest"/>
            <test name="test.model.TestComputer"/>
            <test name="test.model.TestGame"/>
            <test name="test.model.TestGameConfig"/>
            <test name="test.model.TestHuman"/>
            <test name="test.model.testPawn"/>
            <test name="test.model.TestSquare"/>
            <test name="test.util.TestSave"/>
        </junit>
        <mkdir dir="${testReport}"/>
        <junitreport todir="${testReport}">
            <fileset dir=".">
                <include name="TEST-*.xml"/>
            </fileset>
            <report format="frames" todir="${testReport}"/>
        </junitreport>
        <delete file="TEST-test.model.PileTest.xml"/>
        <delete file="TEST-test.model.TestComputer.xml"/>
        <delete file="TEST-test.model.TestGame.xml"/>
        <delete file="TEST-test.model.TestGameConfig.xml"/>
        <delete file="TEST-test.model.TestHuman.xml"/>
        <delete file="TEST-test.model.TestPawn.xml"/>
        <delete file="TEST-test.model.TestSquare.xml"/>
        <delete file="TEST-test.util.TestSave.xml"/>
  </target>

</project>
```