Projet de programmation

Année 2020 - 2021

M2107 – Projet de programmation

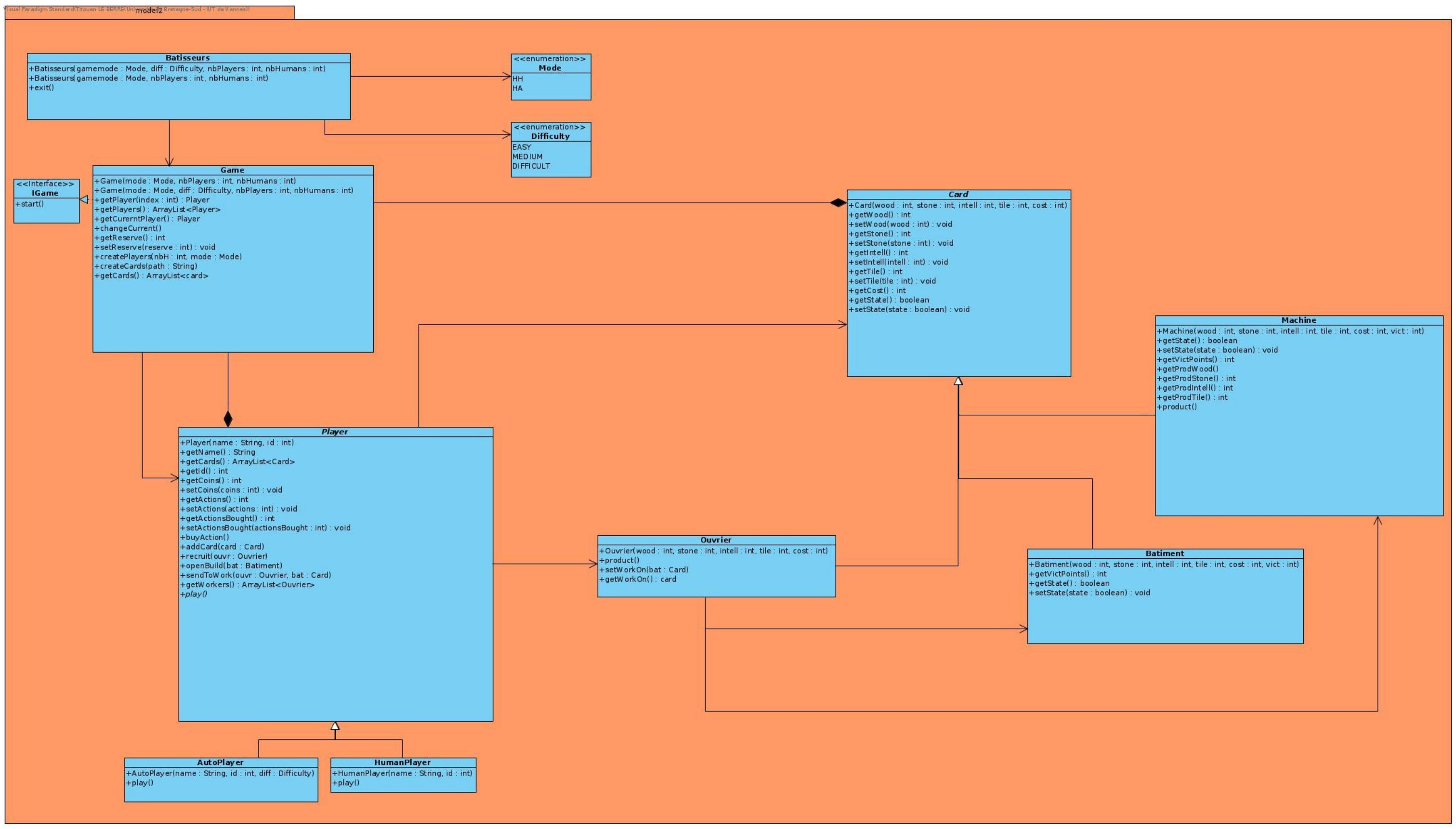
Cahier d'analyse et de conception

IUT Vannes
8 Rue Montaigne – BP 561 – 56017 VANNES Cedex
02 97 62 64 64 – www.iutvannes.fr

Table des matières

I) Diagramme de classe d’analyse....3
II) Diagrammes de classe de conception....4
III) Diagrammes de séquence boîte noire....5
IV) Spécification des formats de fichier....8
V) Squelette des classes & classes de tests....9
A) Classes principales....9
1) Batisseurs.java....9
2) Game.java....11
3) Player.java....14
4) HumanPlayer.java....18
5) AutoPlayer.java....18
6) Card.java....19
7) Ouvrier.java....22
8) Batiment.java....23
9) Machine.java....24
10) Mode.java....26
11) Igame.java....26
12) Difficulty.java....26
B) Classes de tests Junit....27
1) TestBatisseurs.java....27
2) TestPlayer.java....28
3) TestGame.java....30
4) TestOuvrier.java....31
5) TestMachine.java....33
6) TestBatiment.java....34
VI) ANT et fichier de construction....35

I) Diagramme de classe d’analyse


Visual Paradigm Standard(Titouan LE BERRE(Université de Bretagne-Sud - IUT de Vannes))
model2
Batisseurs
+Batisseurs(gamemode : Mode, diff : Difficulty, nbPlayers : int, nbHumans : int)
+Batisseurs(gamemode : Mode, nbPlayers : int, nbHumans : int)
+exit()
<<enumeration>>
Mode
HH
HA
<<enumeration>>
Difficulty
EASY
MEDIUM
DIFFICULT
<<Interface>>
IGame
+start()
Game
+Game(mode : Mode, nbPlayers : int, nbHumans : int)
+Game(mode : Mode, diff : DIfficulty, nbPlayers : int, nbHumans : int)
+getPlayer(index : int) : Player
+getPlayers() : ArrayList<Player>
+getCurerntPlayer() : Player
+changeCurrent()
+getReserve() : int
+setReserve(reserve : int) : void
+createPlayers(nbH : int, mode : Mode)
+createCards(path : String)
+getCards() : ArrayList<card>
Card
+Card(wood : int, stone : int, intell : int, tile : int, cost : int)
+getWood() : int
+setWood(wood : int) : void
+getStone() : int
+setStone(stone : int) : void
+getIntell() : int
+setIntell(intell : int) : void
+getTile() : int
+setTile(tile : int) : void
+getCost() : int
+getState() : boolean
+setState(state : boolean) : void
Machine
+Machine(wood : int, stone : int, intell : int, tile : int, cost : int, vict : int)
+getState() : boolean
+setState(state : boolean) : void
+getVictPoints() : int
+getProdWood()
+getProdStone() : int
+getProdIntell() : int
+getProdTile() : int
+product()
Player
+Player(name : String, id : int)
+getName() : String
+getCards() : ArrayList<Card>
+getId() : int
+getCoins() : int
+setCoins(coins : int) : void
+getActions() : int
+setActions(actions : int) : void
+getActionsBought() : int
+setActionsBought(actionsBought : int) : void
+buyAction()
+addCard(card : Card)
+recruit(ouvr : Ouvrier)
+openBuild(bat : Batiment)
+sendToWork(ouvr : Ouvrier, bat : Card)
+getWorkers() : ArrayList<Ouvrier>
+play()
Ouvrier
+Ouvrier(wood : int, stone : int, intell : int, tile : int, cost : int)
+product()
+setWorkOn(bat : Card)
+getWorkOn() : card
Batiment
+Batiment(wood : int, stone : int, intell : int, tile : int, cost : int, vict : int)
+getVictPoints() : int
+getState() : boolean
+setState(state : boolean) : void
AutoPlayer
+AutoPlayer(name : String, id : int, diff : Difficulty)
+play()
HumanPlayer
+HumanPlayer(name : String, id : int)
+play()

Ci-dessus le diagramme d’analyse de la partie « model » du projet (on utilisera le modèle **MVC** pour ce projet). On peut y lire toutes les classes & interfaces de ce package, avec leurs méthodes de visibilité **public**. Les liens entres les différentes classes sont indiqués, mais les multiplicités ne sont pas précisées.

II) Diagrammes de classe de conception

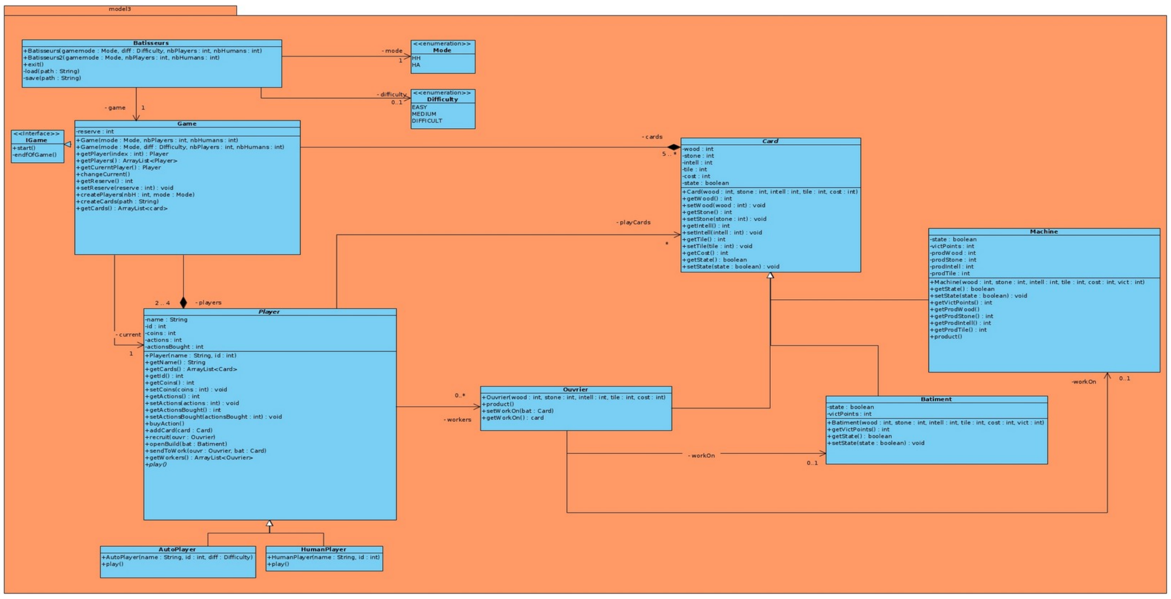

model3
Batisseurs
+Batisseurs(gamemode : Mode, diff : Difficulty, nbPlayers : int, nbHumans : int)
+Batisseurs2(gamemode : Mode, nbPlayers : int, nbHumans : int)
+exit()
-load(path : String)
-save(path : String)
- mode
1
<<enumeration>>
Mode
HH
HA
- difficulty
0..1
<<enumeration>>
Difficulty
EASY
MEDIUM
DIFFICULT
- game
1
<<Interface>>
IGame
+start()
-endfOfGame()
Game
-reserve : int
+Game(mode : Mode, nbPlayers : int, nbHumans : int)
+Game(mode : Mode, diff : Difficulty, nbPlayers : int, nbHumans : int)
+getPlayer(index : int) : Player
+getPlayers() : ArrayList<Player>
+getCurerntPlayer() : Player
+changeCurrent()
+getReserve() : int
+setReserve(reserve : int) : void
+createPlayers(nbH : int, mode : Mode)
+createCards(path : String)
+getCards() : ArrayList<card>
- cards
5 .. *
Card
-wood : int
-stone : int
-intell : int
-tile : int
-cost : int
-state : boolean
+Card(wood : int, stone : int, intell : int, tile : int, cost : int)
+getWood() : int
+setWood(wood : int) : void
+getStone() : int
+setStone(stone : int) : void
+getIntell() : int
+setIntell(intell : int) : void
+getTile() : int
+setTile(tile : int) : void
+getCost() : int
+getState() : boolean
+setState(state : boolean) : void
- playCards
*
Machine
-state : boolean
-victPoints : int
-prodWood : int
-prodStone : int
-prodIntell : int
-prodTile : int
+Machine(wood : int, stone : int, intell : int, tile : int, cost : int, vict : int)
+getState() : boolean
+setState(state : boolean) : void
+getVictPoints() : int
+getProdWood()
+getProdStone() : int
+getProdIntell() : int
+getProdTile() : int
+product()
2 .. 4
- players
- current
1
Player
-name : String
-id : int
-coins : int
-actions : int
-actionsBought : int
+Player(name : String, id : int)
+getName() : String
+getCards() : ArrayList<Card>
+getId() : int
+getCoins() : int
+setCoins(coins : int) : void
+getActions() : int
+setActions(actions : int) : void
+getActionsBought() : int
+setActionsBought(actionsBought : int) : void
+buyAction()
+addCard(card : Card)
+recruit(ouvr : Ouvrier)
+openBuild(bat : Batiment)
+sendToWork(ouvr : Ouvrier, bat : Card)
+getWorkers() : ArrayList<Ouvrier>
+play()
0..*
- workers
Ouvrier
+Ouvrier(wood : int, stone : int, intell : int, tile : int, cost : int)
+product()
+setWorkOn(bat : Card)
+getWorkOn() : card
- workOn
0..1
-workOn
0..1
Batiment
-state : boolean
-victPoints : int
+Batiment(wood : int, stone : int, intell : int, tile : int, cost : int, vict : int)
+getVictPoints() : int
+getState() : boolean
+setState(state : boolean) : void
AutoPlayer
+AutoPlayer(name : String, id : int, diff : Difficulty)
+play()
HumanPlayer
+HumanPlayer(name : String, id : int)
+play()

Diagramme de conception du package model

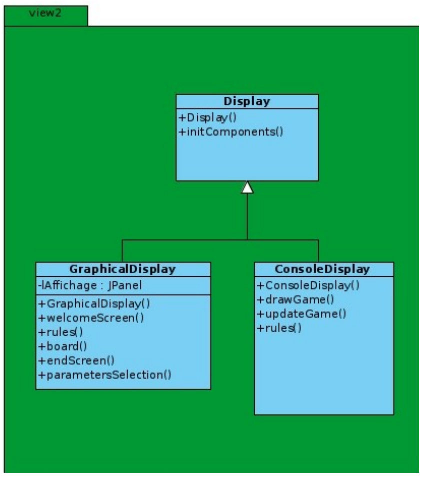
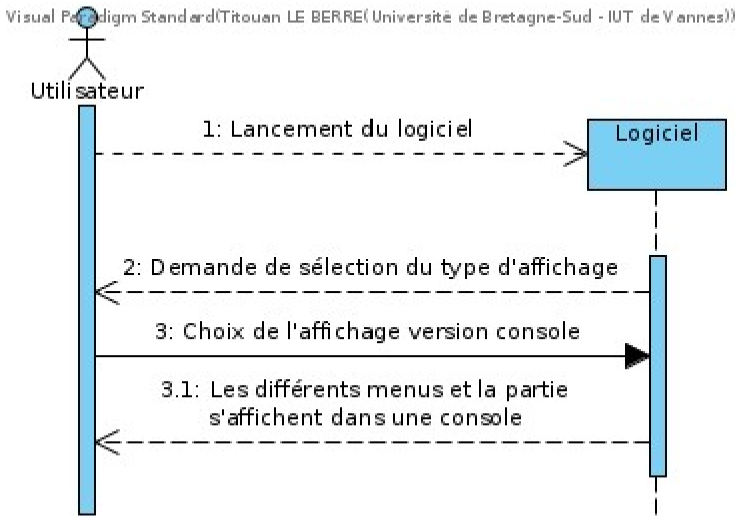


Diagramme de conception du package view

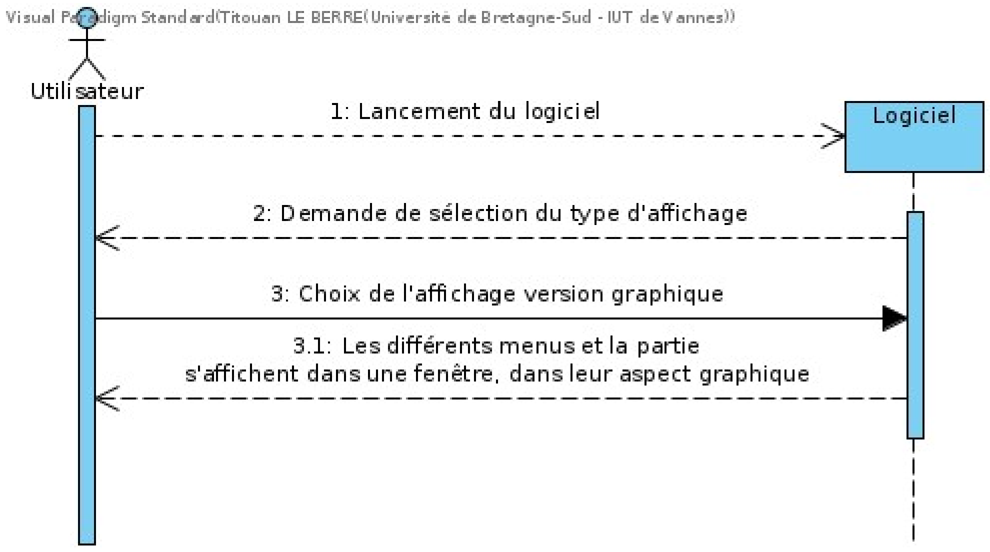
Ci-dessus les diagrammes de conception (contenant donc toutes les méthodes des différentes classes, ainsi que les dépendances fortes et les multiplicités entre ces classes) de 2 des 3 packages du modèle MVC. Le package controller servira à la gestion des écouteurs d’éventements, et servira d’intermédiaire entre le lanceur du jeu et la partie model du jeu.

III) Diagrammes de séquence boîte noire

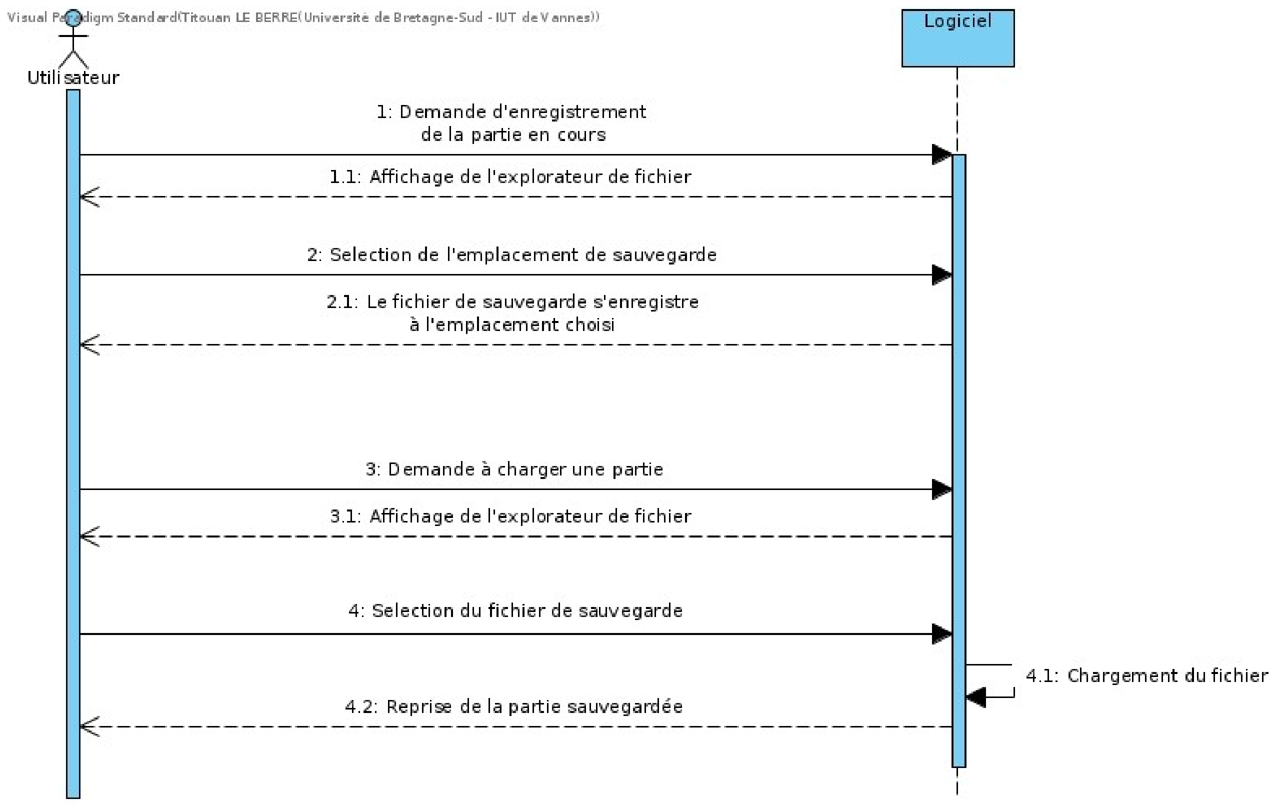
Ces diagrammes de séquence permettent de définir les différentes interactions entre l'utilisateur et le logiciel, et quelles vont être les actions que l'utilisateur va devoir effectuer en fonction du résultat voulu. Ces diagrammes étant des diagrammes de séquence boîte noire, on ne fait laisse pas apparaître les différentes interactions entres les classes et leur méthodes (à ce point du projet, certaines de ces interactions ne sont pas conçues ou imaginées).


Visual Paradigm Standard(Titouan LE BERRE(Université de Bretagne-Sud - IUT de Vannes))
Utilisateur
Logiciel
1: Lancement du logiciel
2: Demande de sélection du type d'affichage
3: Choix de l'affichage version console
3.1: Les différents menus et la partie s'affichent dans une console

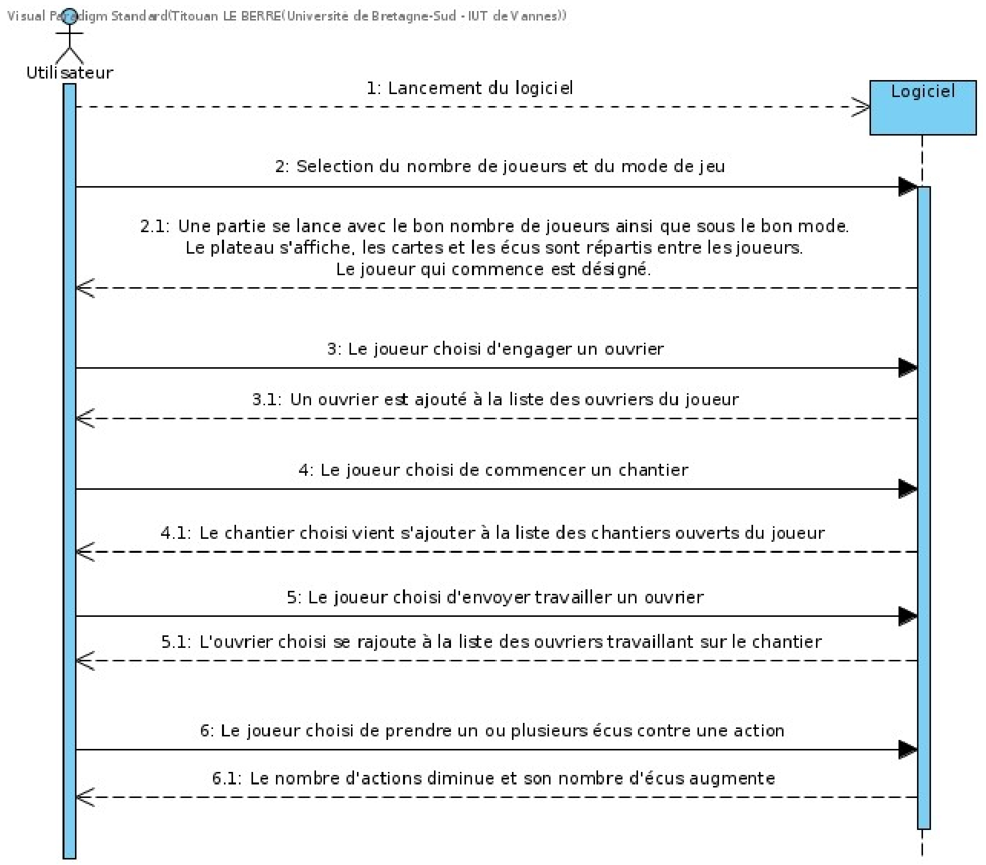
Sélection de l'affichage console


Visual Paradigm Standard(Titouan LE BERRE(Université de Bretagne-Sud - IUT de Vannes))
Utilisateur
Logiciel
1: Lancement du logiciel
2: Demande de sélection du type d'affichage
3: Choix de l'affichage version graphique
3.1: Les différents menus et la partie s'affichent dans une fenêtre, dans leur aspect graphique

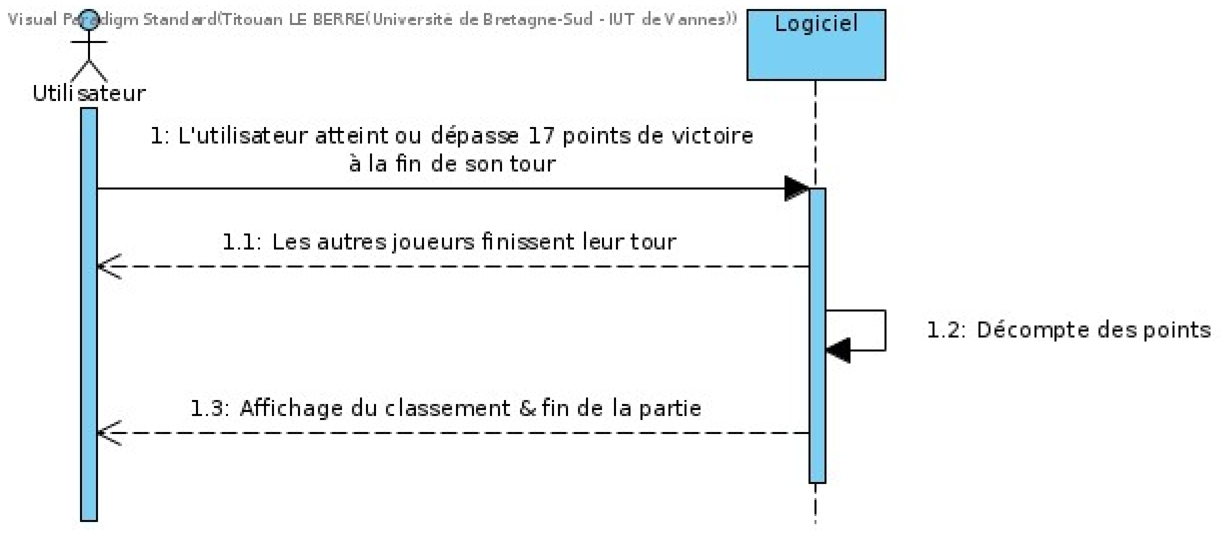
Sélection de l'affichage graphique


Visual Paradigm Standard(Titouan LE BERRE(Université de Bretagne-Sud - IUT de Vannes))
Utilisateur
Logiciel
1: Demande d'enregistrement de la partie en cours
1.1: Affichage de l'explorateur de fichier
2: Selection de l'emplacement de sauvegarde
2.1: Le fichier de sauvegarde s'enregistre à l'emplacement choisi
3: Demande à charger une partie
3.1: Affichage de l'explorateur de fichier
4: Selection du fichier de sauvegarde
4.1: Chargement du fichier
4.2: Reprise de la partie sauvegardée

Enregistrement et chargement d'une sauvegarde


Visual Paradigm Standard(Titouan LE BERRE(Université de Bretagne-Sud - IUT de Vannes))
Utilisateur
Logiciel
1: Lancement du logiciel
2: Selection du nombre de joueurs et du mode de jeu
2.1: Une partie se lance avec le bon nombre de joueurs ainsi que sous le bon mode. Le plateau s'affiche, les cartes et les écus sont répartis entre les joueurs. Le joueur qui commence est désigné.
3: Le joueur choisi d'engager un ouvrier
3.1: Un ouvrier est ajouté à la liste des ouvriers du joueur
4: Le joueur choisi de commencer un chantier
4.1: Le chantier choisi vient s'ajouter à la liste des chantiers ouverts du joueur
5: Le joueur choisi d'envoyer travailler un ouvrier
5.1: L'ouvrier choisi se rajoute à la liste des ouvriers travaillant sur le chantier
6: Le joueur choisi de prendre un ou plusieurs écus contre une action
6.1: Le nombre d'actions diminue et son nombre d'écus augmente

Lancement, mise en place & déroulement d'une partie


Visual Paradigm Standard(Titouan LE BERRE(Université de Bretagne-Sud - IUT de Vannes))
Utilisateur
Logiciel
1: L'utilisateur atteint ou dépasse 17 points de victoire à la fin de son tour
1.1: Les autres joueurs finissent leur tour
1.2: Décompte des points
1.3: Affichage du classement & fin de la partie

Fin de partie

IV) Spécification des formats de fichier

Le logiciel sera sous la forme d’une archive Java et aura donc l’extension .jar.

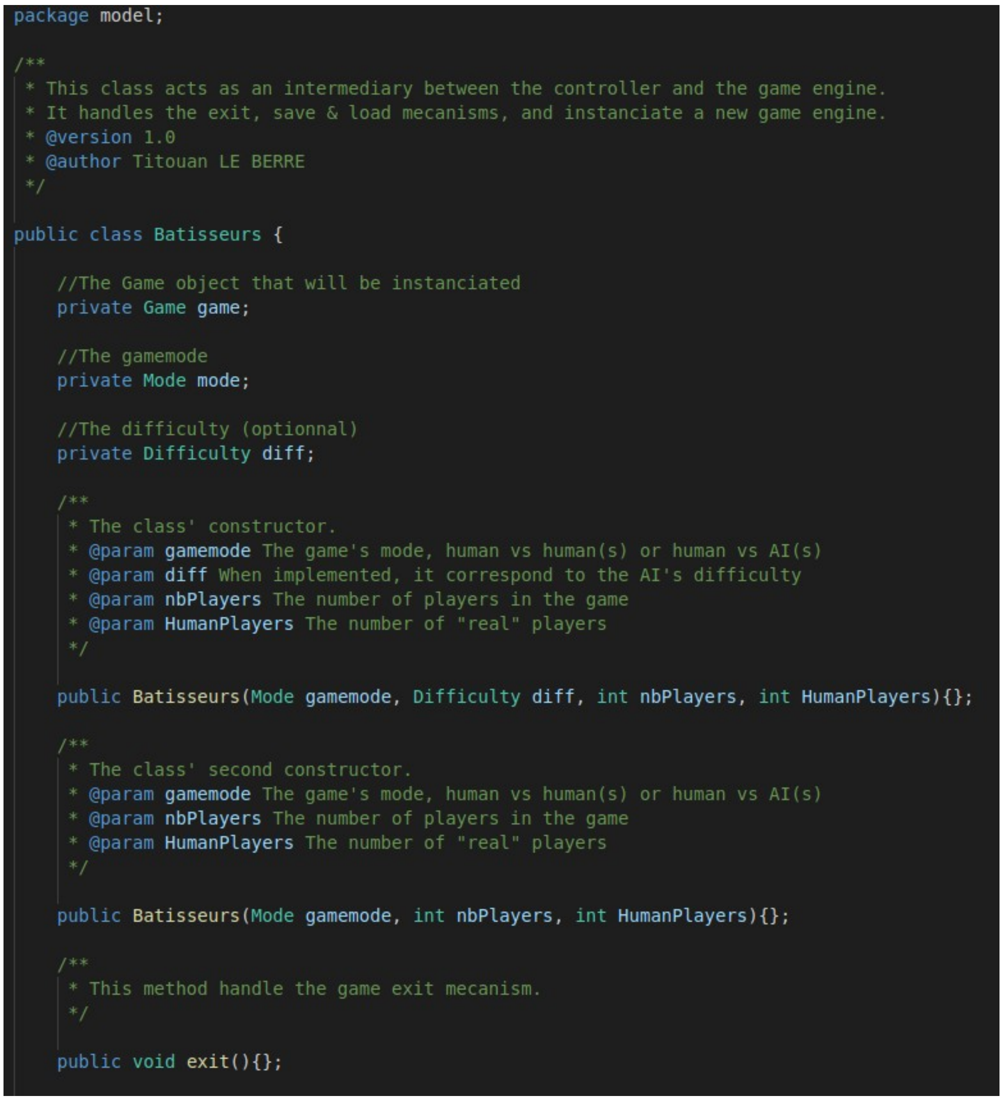
Un ou plusieurs fichiers de sauvegarde sont nécessaires pour que l’utilisateur puisse reprendre une partie.

Ces fichiers auront l’extension .txt, et seront générés grâce à l’interface Java Serializable (afin de faciliter le chargement de l’état de la partie). Leur contenu suivra donc le format classique de Serializable (nom de l’objet, attributs, etc.). On notera deux grandes catégories de fichiers : les fichiers de configuration (qui seront écrits et lus sous forme de binaire) et les fichiers de sauvegarde des objets en tant que tel.

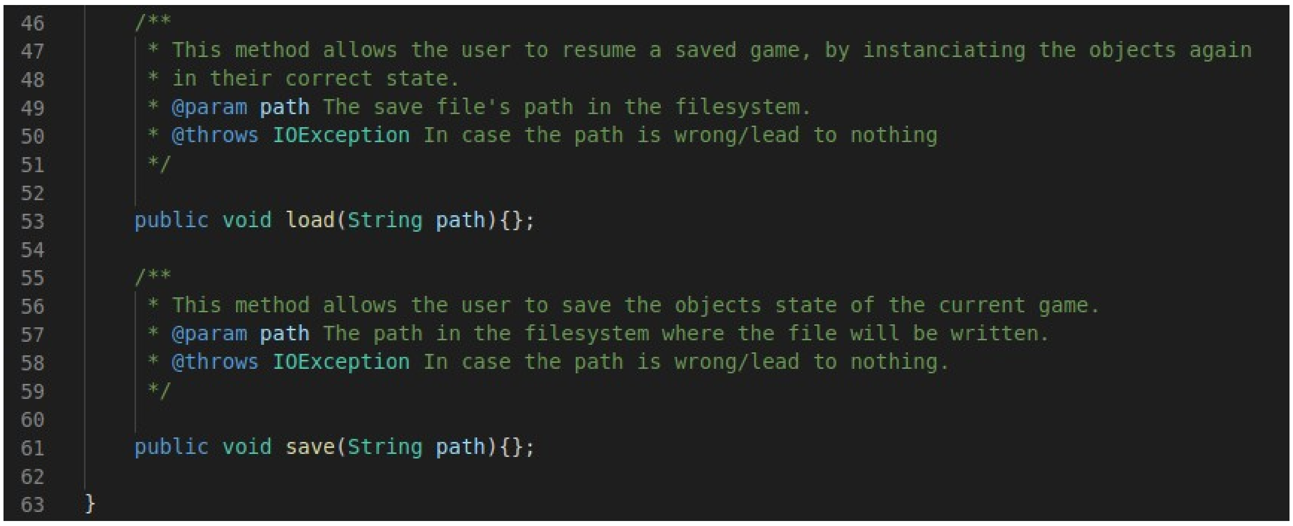
V) Squelette des classes & classes de tests

A) Classes principales

1) Batisseurs.java

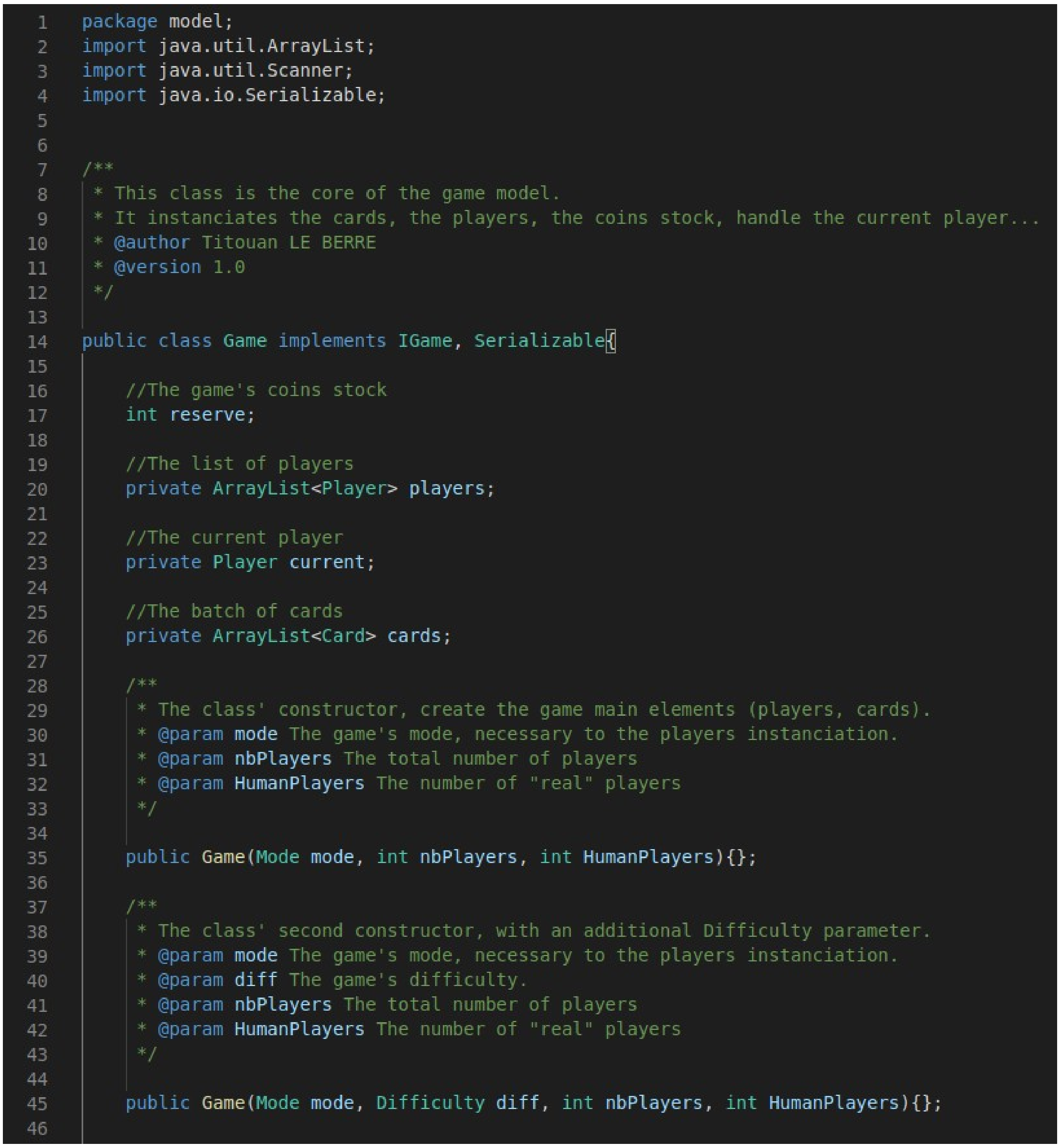


```java
package model;

/**
 * This class acts as an intermediary between the controller and the game engine.
 * It handles the exit, save & load mecanisms, and instanciate a new game engine.
 * @version 1.0
 * @author Titouan LE BERRE
 */

public class Batisseurs {

    //The Game object that will be instanciated
    private Game game;

    //The gamemode
    private Mode mode;

    //The difficulty (optionnal)
    private Difficulty diff;

    /**
     * The class' constructor.
     * @param gamemode The game's mode, human vs human(s) or human vs AI(s)
     * @param diff When implemented, it correspond to the AI's difficulty
     * @param nbPlayers The number of players in the game
     * @param HumanPlayers The number of "real" players
     */

    public Batisseurs(Mode gamemode, Difficulty diff, int nbPlayers, int HumanPlayers){};

    /**
     * The class' second constructor.
     * @param gamemode The game's mode, human vs human(s) or human vs AI(s)
     * @param nbPlayers The number of players in the game
     * @param HumanPlayers The number of "real" players
     */

    public Batisseurs(Mode gamemode, int nbPlayers, int HumanPlayers){};

    /**
     * This method handle the game exit mecanism.
     */

    public void exit(){};
```

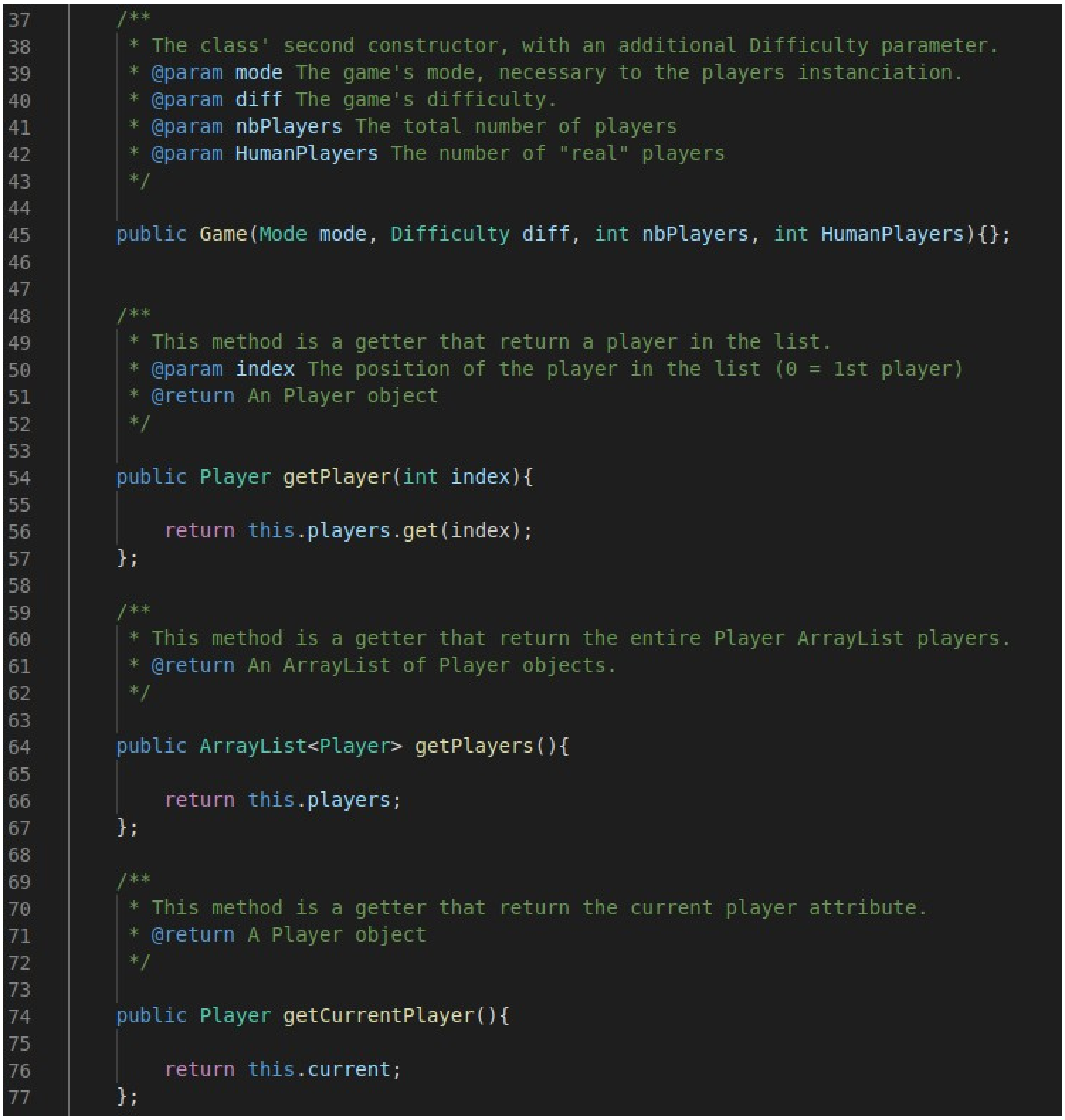


```java
    /**
     * This method allows the user to resume a saved game, by instanciating the objects again
     * in their correct state.
     * @param path The save file's path in the filesystem.
     * @throws IOException In case the path is wrong/lead to nothing
     */

    public void load(String path){};

    /**
     * This method allows the user to save the objects state of the current game.
     * @param path The path in the filesystem where the file will be written.
     * @throws IOException In case the path is wrong/lead to nothing.
     */

    public void save(String path){};

}
```

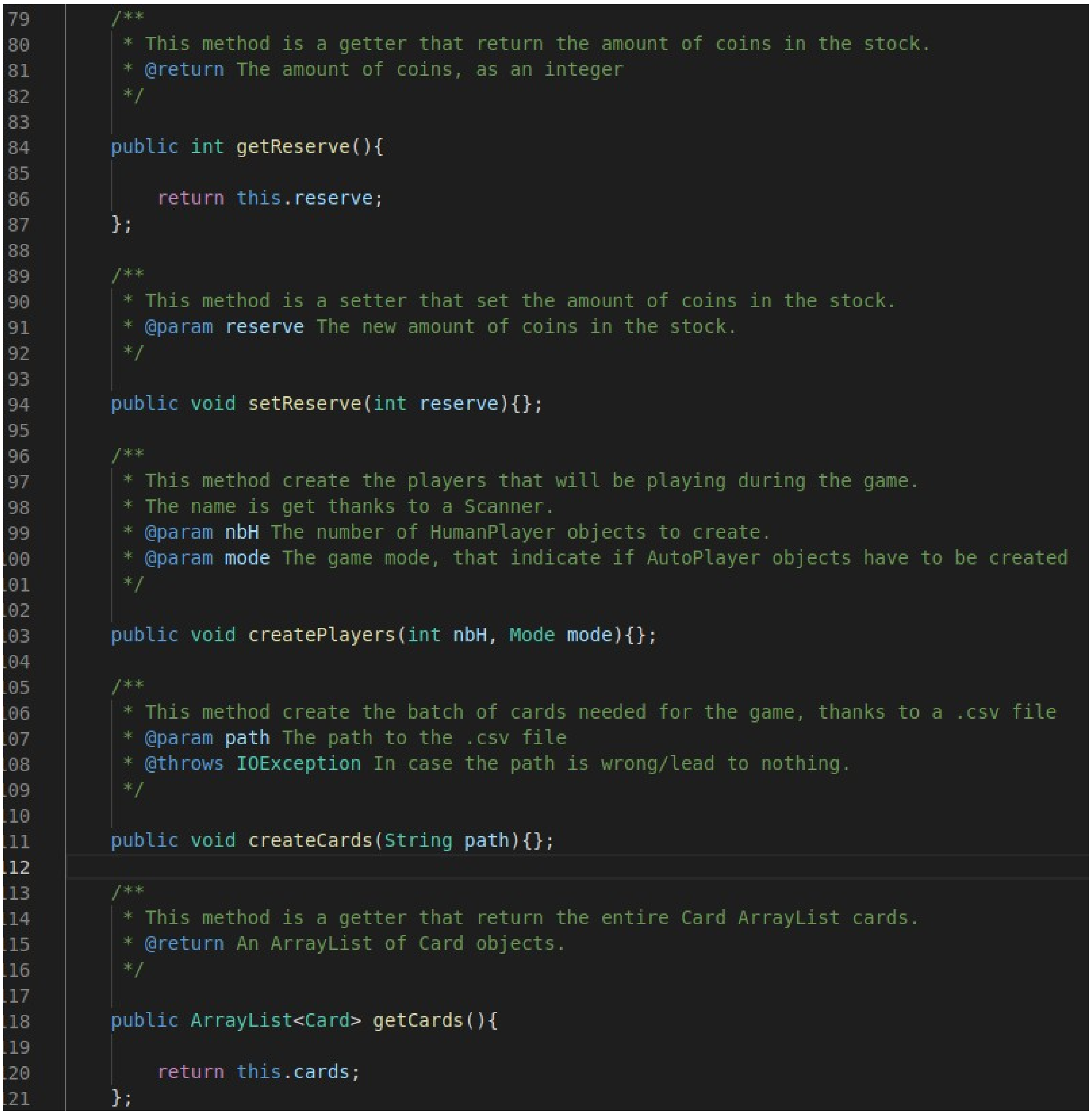
2) Game.java



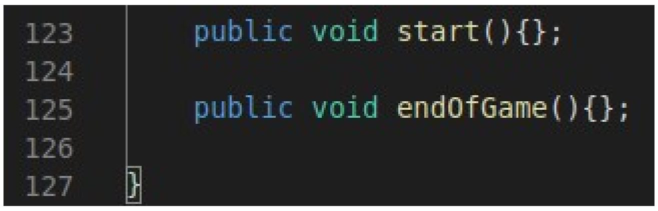
```java
package model;
import java.util.ArrayList;
import java.util.Scanner;
import java.io.Serializable;


/**
 * This class is the core of the game model.
 * It instanciates the cards, the players, the coins stock, handle the current player...
 * @author Titouan LE BERRE
 * @version 1.0
 */

public class Game implements IGame, Serializable{

    //The game's coins stock
    int reserve;

    //The list of players
    private ArrayList<Player> players;

    //The current player
    private Player current;

    //The batch of cards
    private ArrayList<Card> cards;

    /**
     * The class' constructor, create the game main elements (players, cards).
     * @param mode The game's mode, necessary to the players instanciation.
     * @param nbPlayers The total number of players
     * @param HumanPlayers The number of "real" players
     */

    public Game(Mode mode, int nbPlayers, int HumanPlayers){};

    /**
     * The class' second constructor, with an additional Difficulty parameter.
     * @param mode The game's mode, necessary to the players instanciation.
     * @param diff The game's difficulty.
     * @param nbPlayers The total number of players
     * @param HumanPlayers The number of "real" players
     */

    public Game(Mode mode, Difficulty diff, int nbPlayers, int HumanPlayers){};

```



```java
    /**
     * The class' second constructor, with an additional Difficulty parameter.
     * @param mode The game's mode, necessary to the players instanciation.
     * @param diff The game's difficulty.
     * @param nbPlayers The total number of players
     * @param HumanPlayers The number of "real" players
     */

    public Game(Mode mode, Difficulty diff, int nbPlayers, int HumanPlayers){};


    /**
     * This method is a getter that return a player in the list.
     * @param index The position of the player in the list (0 = 1st player)
     * @return An Player object
     */

    public Player getPlayer(int index){

        return this.players.get(index);
    };

    /**
     * This method is a getter that return the entire Player ArrayList players.
     * @return An ArrayList of Player objects.
     */

    public ArrayList<Player> getPlayers(){

        return this.players;
    };

    /**
     * This method is a getter that return the current player attribute.
     * @return A Player object
     */

    public Player getCurrentPlayer(){

        return this.current;
    };
```

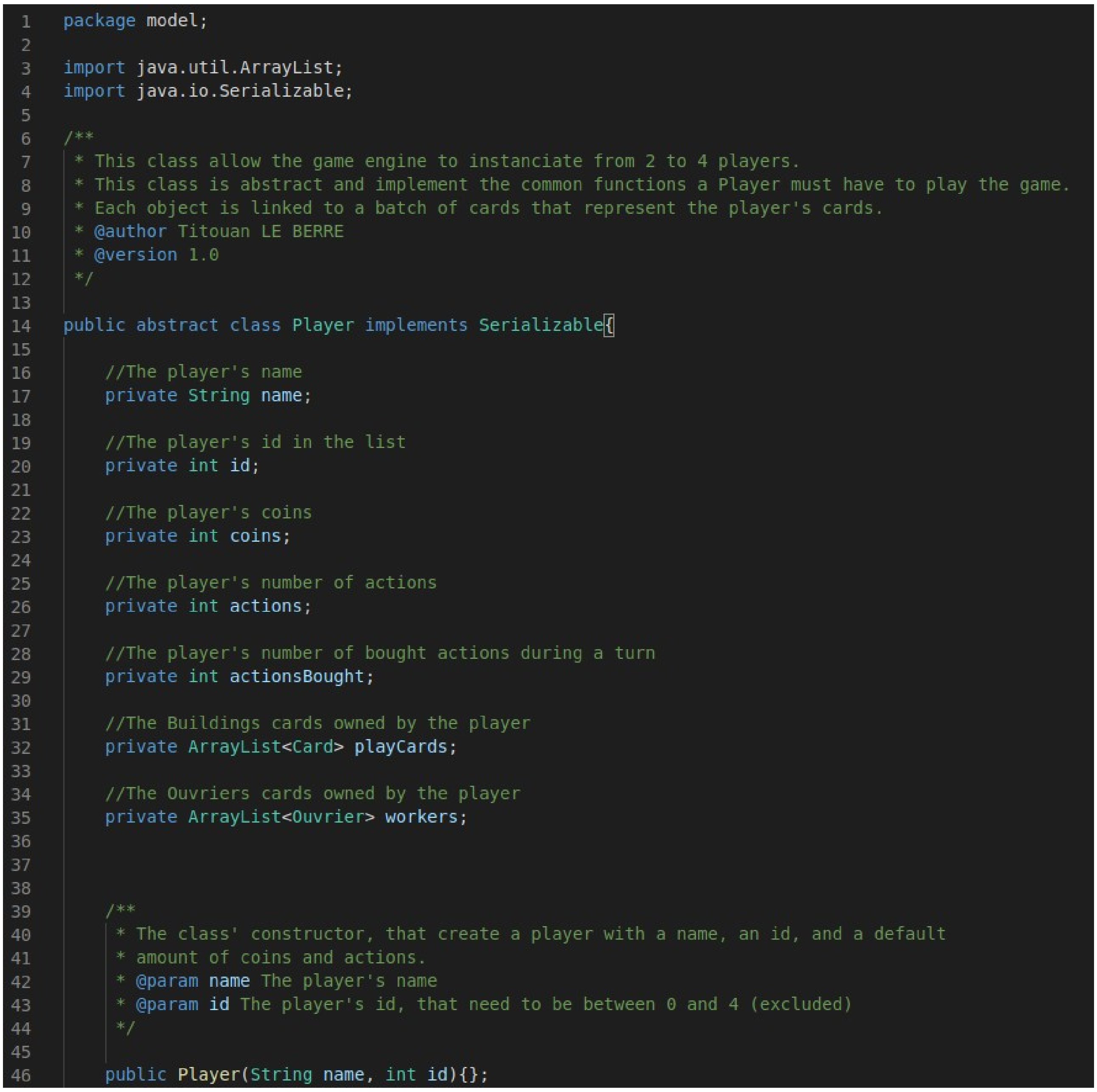


```java
    /**
     * This method is a getter that return the amount of coins in the stock.
     * @return The amount of coins, as an integer
     */

    public int getReserve(){

        return this.reserve;
    };

    /**
     * This method is a setter that set the amount of coins in the stock.
     * @param reserve The new amount of coins in the stock.
     */

    public void setReserve(int reserve){};

    /**
     * This method create the players that will be playing during the game.
     * The name is get thanks to a Scanner.
     * @param nbH The number of HumanPlayer objects to create.
     * @param mode The game mode, that indicate if AutoPlayer objects have to be created
     */

    public void createPlayers(int nbH, Mode mode){};

    /**
     * This method create the batch of cards needed for the game, thanks to a .csv file
     * @param path The path to the .csv file
     * @throws IOException In case the path is wrong/lead to nothing.
     */

    public void createCards(String path){};

    /**
     * This method is a getter that return the entire Card ArrayList cards.
     * @return An ArrayList of Card objects.
     */

    public ArrayList<Card> getCards(){

        return this.cards;
    };
```

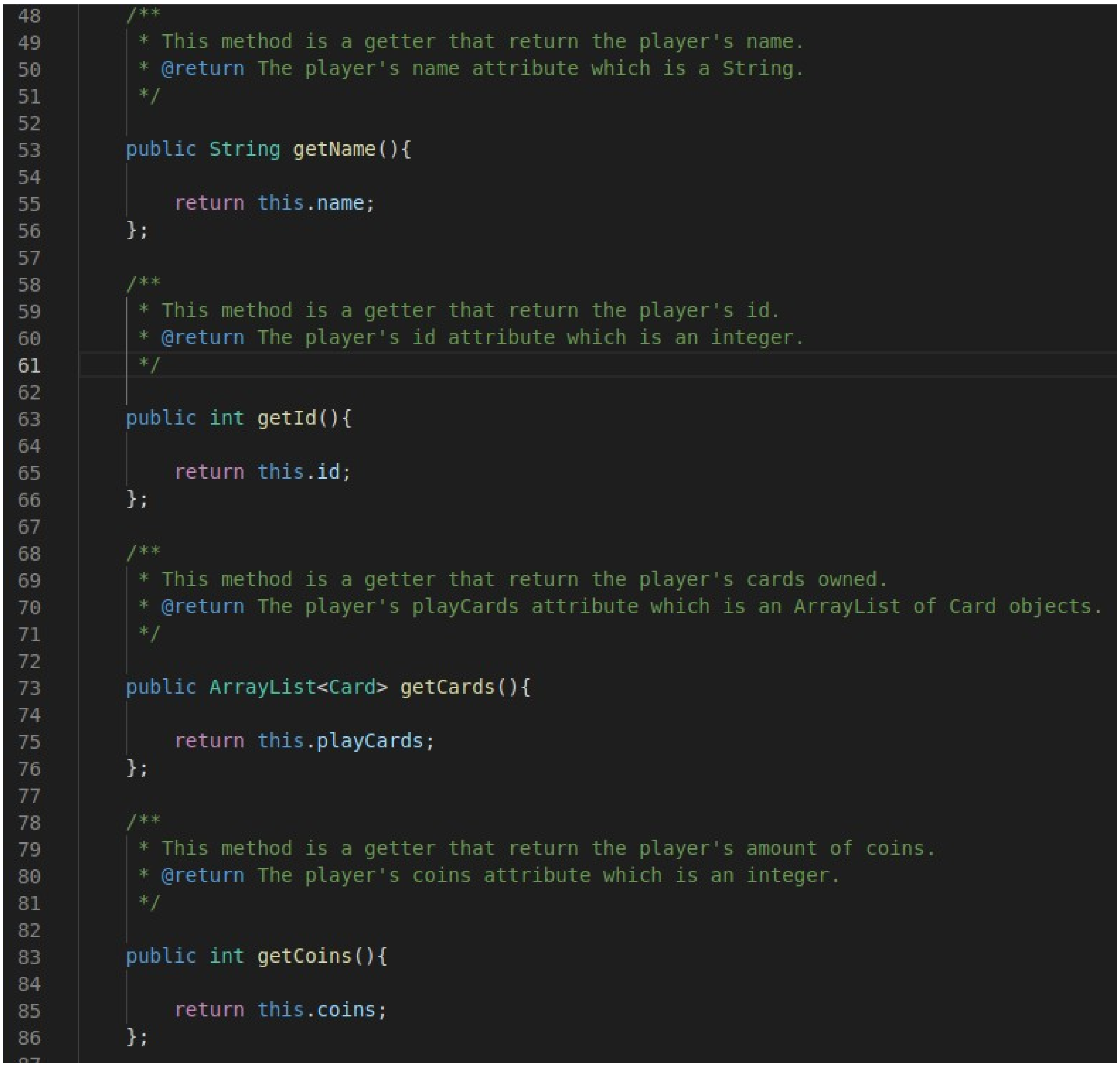


```java
    public void start(){};

    public void endOfGame(){};

}
```

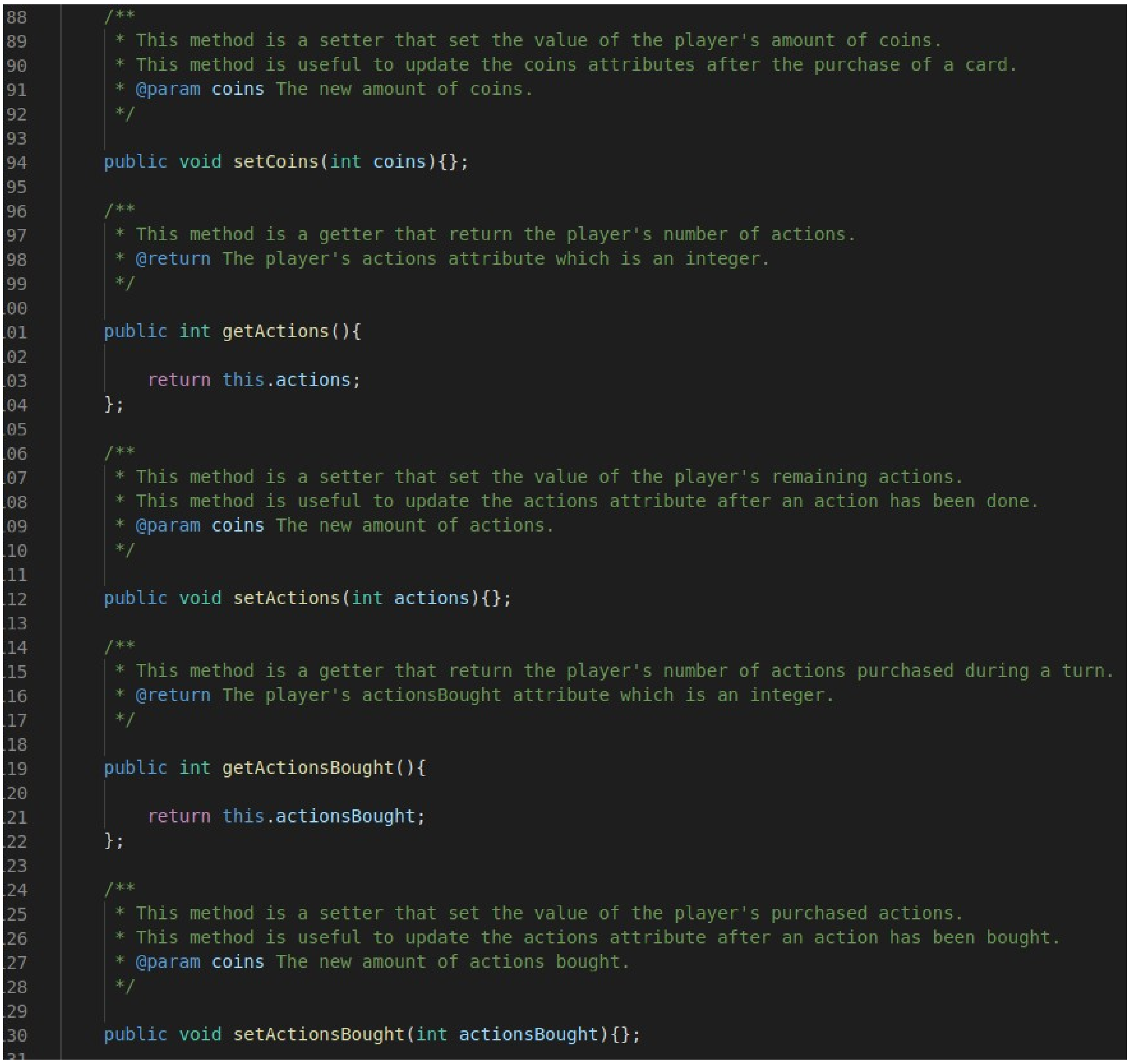
3) Player.java



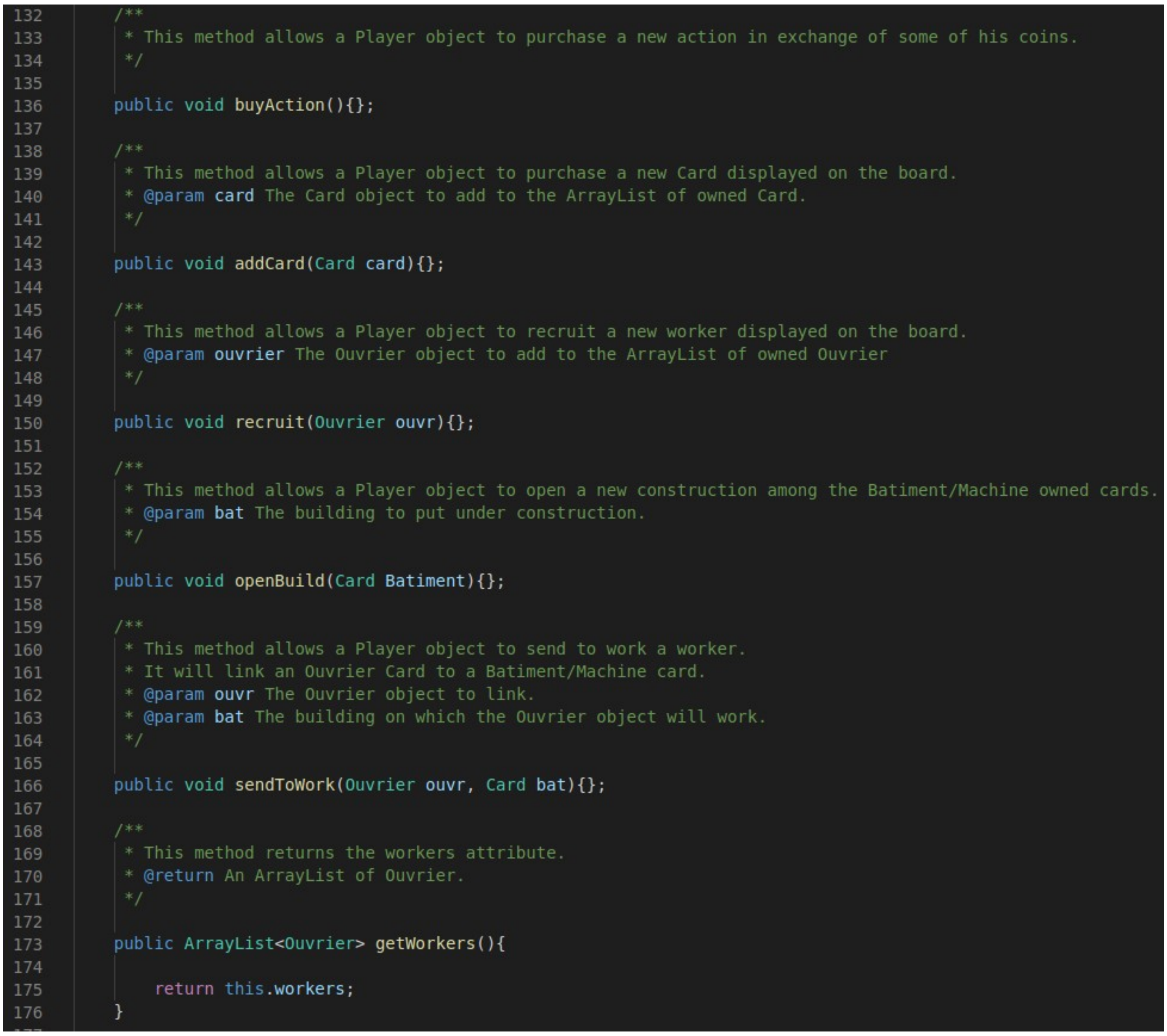
```java
package model;

import java.util.ArrayList;
import java.io.Serializable;

/**
 * This class allow the game engine to instanciate from 2 to 4 players.
 * This class is abstract and implement the common functions a Player must have to play the game.
 * Each object is linked to a batch of cards that represent the player's cards.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public abstract class Player implements Serializable{

    //The player's name
    private String name;

    //The player's id in the list
    private int id;

    //The player's coins
    private int coins;

    //The player's number of actions
    private int actions;

    //The player's number of bought actions during a turn
    private int actionsBought;

    //The Buildings cards owned by the player
    private ArrayList<Card> playCards;

    //The Ouvriers cards owned by the player
    private ArrayList<Ouvrier> workers;



    /**
     * The class' constructor, that create a player with a name, an id, and a default
     * amount of coins and actions.
     * @param name The player's name
     * @param id The player's id, that need to be between 0 and 4 (excluded)
     */

    public Player(String name, int id){};
```



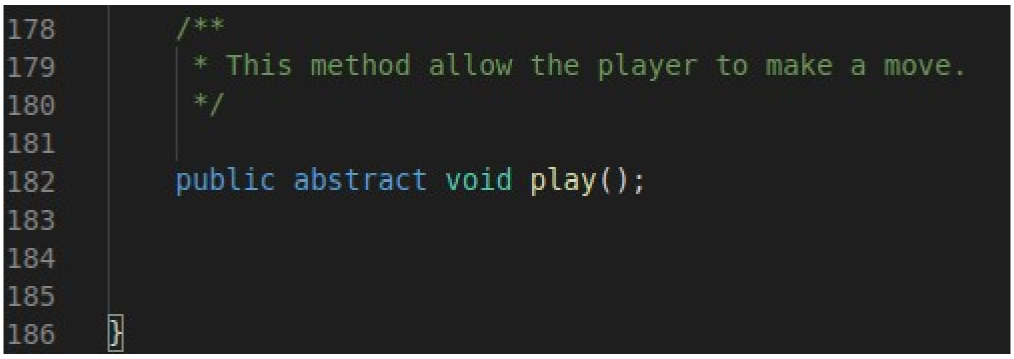
```
    /**
     * This method is a getter that return the player's name.
     * @return The player's name attribute which is a String.
     */

    public String getName(){

        return this.name;
    };

    /**
     * This method is a getter that return the player's id.
     * @return The player's id attribute which is an integer.
     */

    public int getId(){

        return this.id;
    };

    /**
     * This method is a getter that return the player's cards owned.
     * @return The player's playCards attribute which is an ArrayList of Card objects.
     */

    public ArrayList<Card> getCards(){

        return this.playCards;
    };

    /**
     * This method is a getter that return the player's amount of coins.
     * @return The player's coins attribute which is an integer.
     */

    public int getCoins(){

        return this.coins;
    };
```



```
    /**
     * This method is a setter that set the value of the player's amount of coins.
     * This method is useful to update the coins attributes after the purchase of a card.
     * @param coins The new amount of coins.
     */

    public void setCoins(int coins){};

    /**
     * This method is a getter that return the player's number of actions.
     * @return The player's actions attribute which is an integer.
     */

    public int getActions(){

        return this.actions;
    };

    /**
     * This method is a setter that set the value of the player's remaining actions.
     * This method is useful to update the actions attribute after an action has been done.
     * @param coins The new amount of actions.
     */

    public void setActions(int actions){};

    /**
     * This method is a getter that return the player's number of actions purchased during a turn.
     * @return The player's actionsBought attribute which is an integer.
     */

    public int getActionsBought(){

        return this.actionsBought;
    };

    /**
     * This method is a setter that set the value of the player's purchased actions.
     * This method is useful to update the actions attribute after an action has been bought.
     * @param coins The new amount of actions bought.
     */

    public void setActionsBought(int actionsBought){};
```

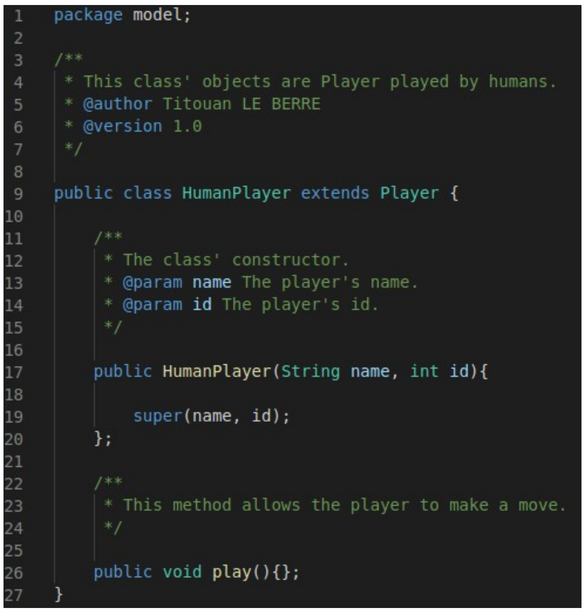


```java
    /**
     * This method allows a Player object to purchase a new action in exchange of some of his coins.
     */

    public void buyAction(){};

    /**
     * This method allows a Player object to purchase a new Card displayed on the board.
     * @param card The Card object to add to the ArrayList of owned Card.
     */

    public void addCard(Card card){};

    /**
     * This method allows a Player object to recruit a new worker displayed on the board.
     * @param ouvrier The Ouvrier object to add to the ArrayList of owned Ouvrier
     */

    public void recruit(Ouvrier ouvr){};

    /**
     * This method allows a Player object to open a new construction among the Batiment/Machine owned cards.
     * @param bat The building to put under construction.
     */

    public void openBuild(Card Batiment){};

    /**
     * This method allows a Player object to send to work a worker.
     * It will link an Ouvrier Card to a Batiment/Machine card.
     * @param ouvr The Ouvrier object to link.
     * @param bat The building on which the Ouvrier object will work.
     */

    public void sendToWork(Ouvrier ouvr, Card bat){};

    /**
     * This method returns the workers attribute.
     * @return An ArrayList of Ouvrier.
     */

    public ArrayList<Ouvrier> getWorkers(){

        return this.workers;
    }
```



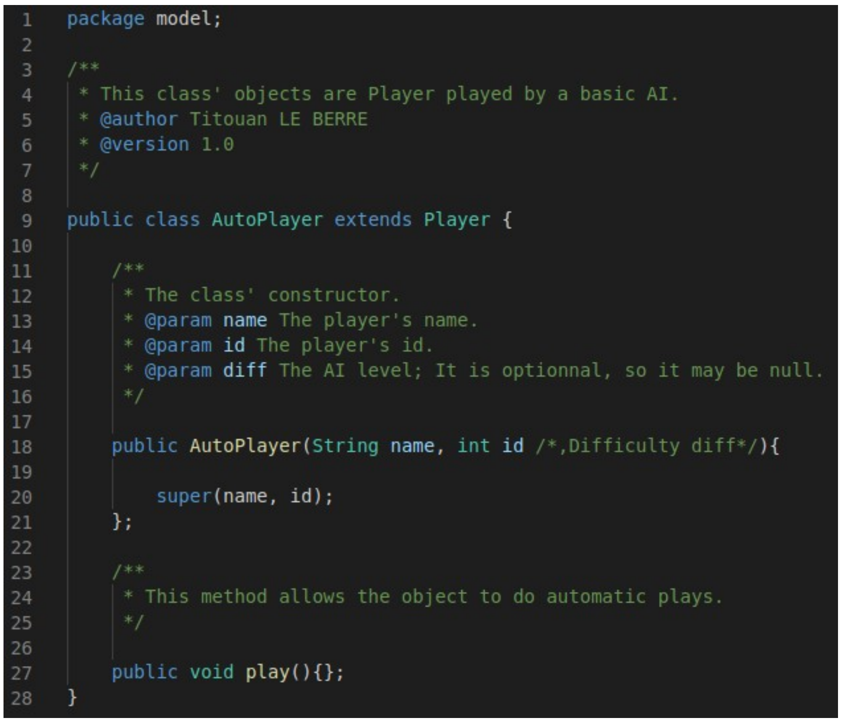
```java
    /**
     * This method allow the player to make a move.
     */

    public abstract void play();



}
```

4) HumanPlayer.java



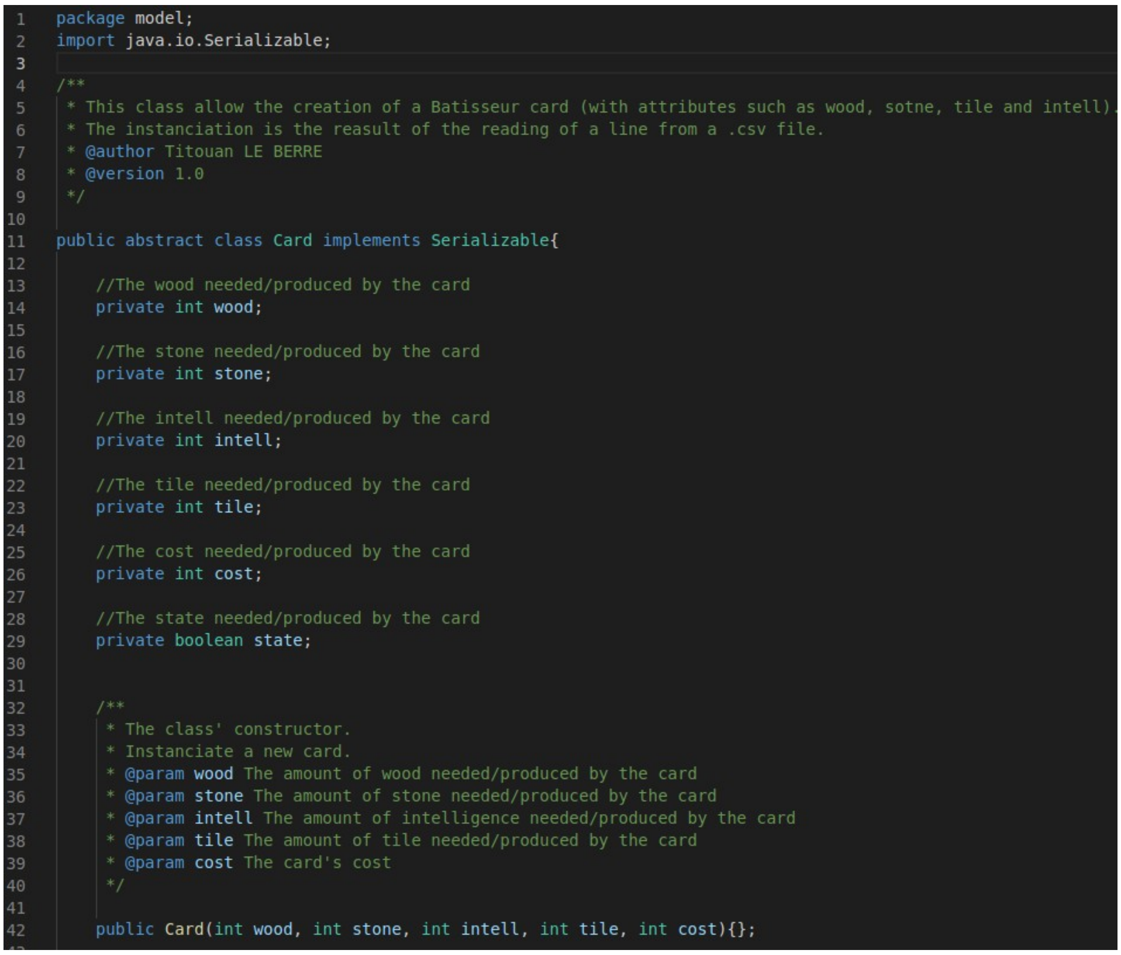
```java
package model;

/**
 * This class' objects are Player played by humans.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public class HumanPlayer extends Player {

    /**
     * The class' constructor.
     * @param name The player's name.
     * @param id The player's id.
     */

    public HumanPlayer(String name, int id){

        super(name, id);
    };

    /**
     * This method allows the player to make a move.
     */

    public void play(){};
}
```

5) AutoPlayer.java

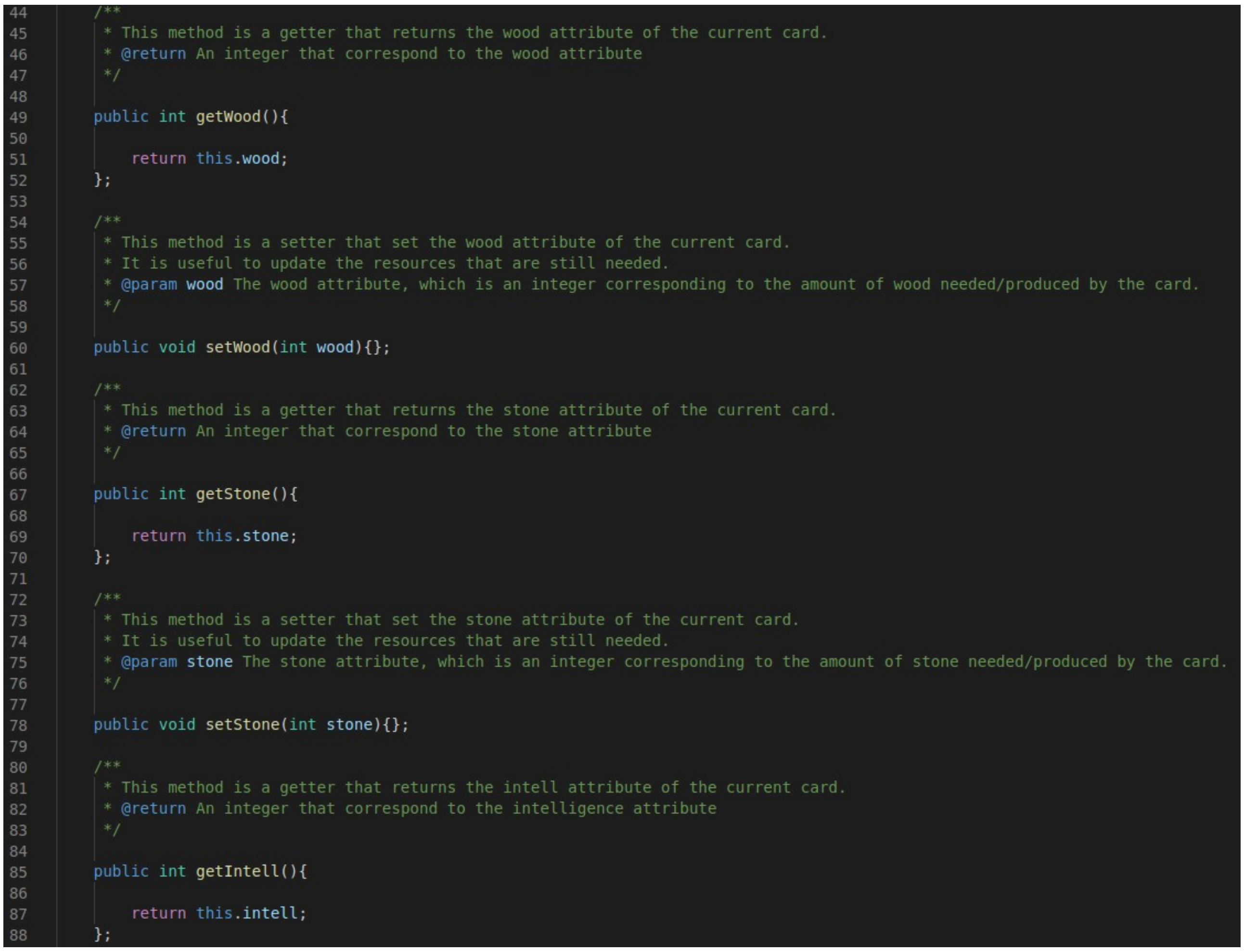


```java
package model;

/**
 * This class' objects are Player played by a basic AI.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public class AutoPlayer extends Player {

    /**
     * The class' constructor.
     * @param name The player's name.
     * @param id The player's id.
     * @param diff The AI level; It is optionnal, so it may be null.
     */

    public AutoPlayer(String name, int id /*,Difficulty diff*/){

        super(name, id);
    };

    /**
     * This method allows the object to do automatic plays.
     */

    public void play(){};
}
```

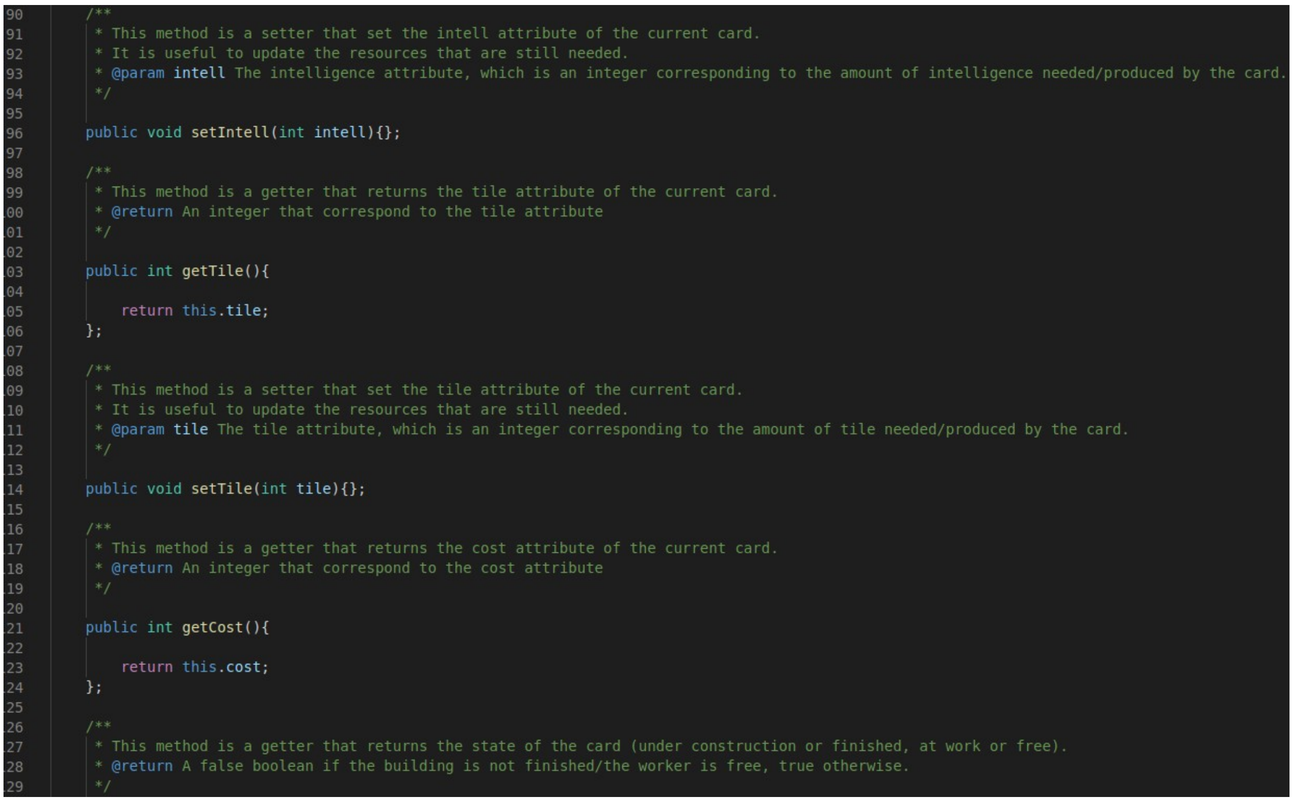
6) Card.java



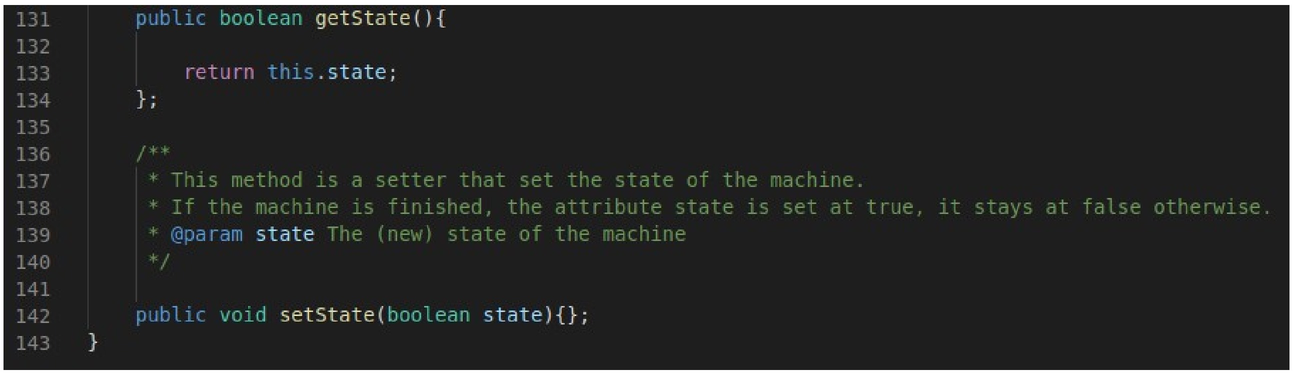
```java
package model;
import java.io.Serializable;

/**
 * This class allow the creation of a Batisseur card (with attributes such as wood, sotne, tile and intell).
 * The instanciation is the reasult of the reading of a line from a .csv file.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public abstract class Card implements Serializable{

    //The wood needed/produced by the card
    private int wood;

    //The stone needed/produced by the card
    private int stone;

    //The intell needed/produced by the card
    private int intell;

    //The tile needed/produced by the card
    private int tile;

    //The cost needed/produced by the card
    private int cost;

    //The state needed/produced by the card
    private boolean state;


    /**
     * The class' constructor.
     * Instanciate a new card.
     * @param wood The amount of wood needed/produced by the card
     * @param stone The amount of stone needed/produced by the card
     * @param intell The amount of intelligence needed/produced by the card
     * @param tile The amount of tile needed/produced by the card
     * @param cost The card's cost
     */

    public Card(int wood, int stone, int intell, int tile, int cost){};
```



```java
    /**
     * This method is a getter that returns the wood attribute of the current card.
     * @return An integer that correspond to the wood attribute
     */

    public int getWood(){

        return this.wood;
    };

    /**
     * This method is a setter that set the wood attribute of the current card.
     * It is useful to update the resources that are still needed.
     * @param wood The wood attribute, which is an integer corresponding to the amount of wood needed/produced by the card.
     */

    public void setWood(int wood){};

    /**
     * This method is a getter that returns the stone attribute of the current card.
     * @return An integer that correspond to the stone attribute
     */

    public int getStone(){

        return this.stone;
    };

    /**
     * This method is a setter that set the stone attribute of the current card.
     * It is useful to update the resources that are still needed.
     * @param stone The stone attribute, which is an integer corresponding to the amount of stone needed/produced by the card.
     */

    public void setStone(int stone){};

    /**
     * This method is a getter that returns the intell attribute of the current card.
     * @return An integer that correspond to the intelligence attribute
     */

    public int getIntell(){

        return this.intell;
    };
```

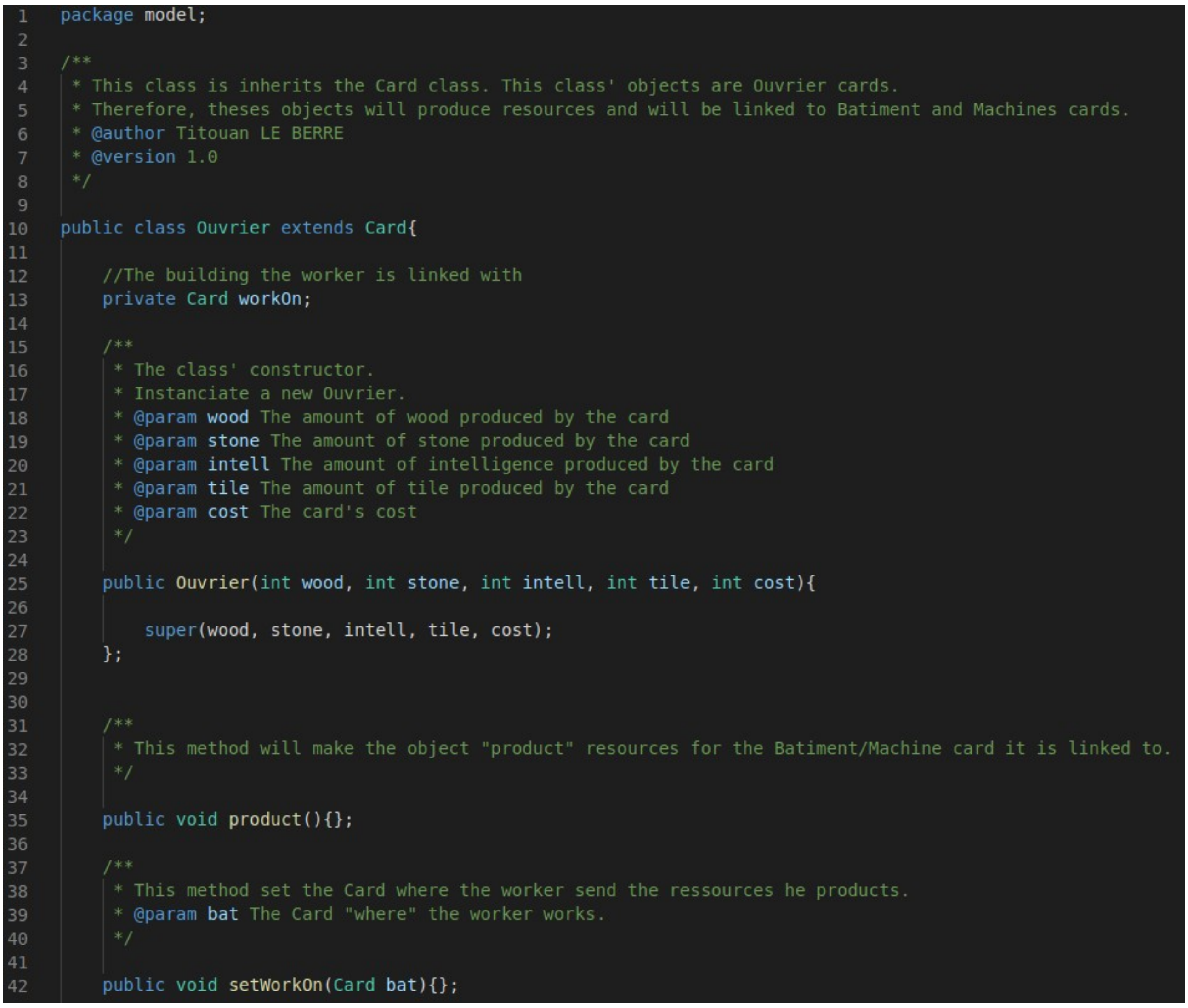


```java
    /**
     * This method is a setter that set the intell attribute of the current card.
     * It is useful to update the resources that are still needed.
     * @param intell The intelligence attribute, which is an integer corresponding to the amount of intelligence needed/produced by the card.
     */

    public void setIntell(int intell){};

    /**
     * This method is a getter that returns the tile attribute of the current card.
     * @return An integer that correspond to the tile attribute
     */

    public int getTile(){

        return this.tile;
    };

    /**
     * This method is a setter that set the tile attribute of the current card.
     * It is useful to update the resources that are still needed.
     * @param tile The tile attribute, which is an integer corresponding to the amount of tile needed/produced by the card.
     */

    public void setTile(int tile){};

    /**
     * This method is a getter that returns the cost attribute of the current card.
     * @return An integer that correspond to the cost attribute
     */

    public int getCost(){

        return this.cost;
    };

    /**
     * This method is a getter that returns the state of the card (under construction or finished, at work or free).
     * @return A false boolean if the building is not finished/the worker is free, true otherwise.
     */
```

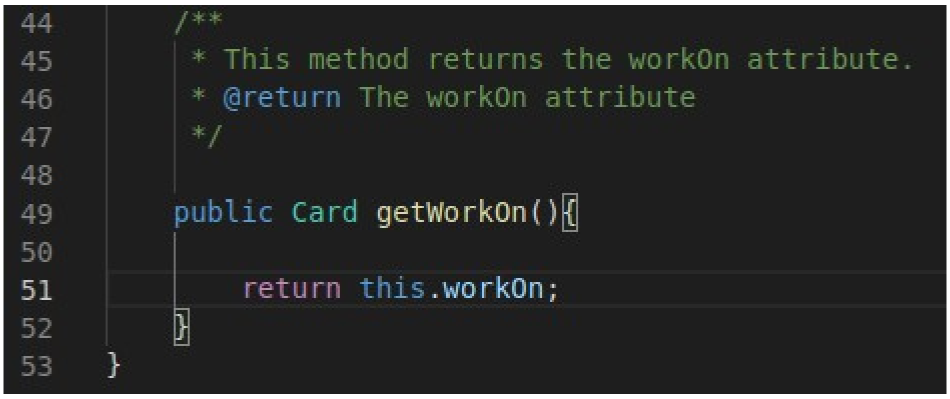


```java
    public boolean getState(){

        return this.state;
    };

    /**
     * This method is a setter that set the state of the machine.
     * If the machine is finished, the attribute state is set at true, it stays at false otherwise.
     * @param state The (new) state of the machine
     */

    public void setState(boolean state){};
}
```

7) Ouvrier.java

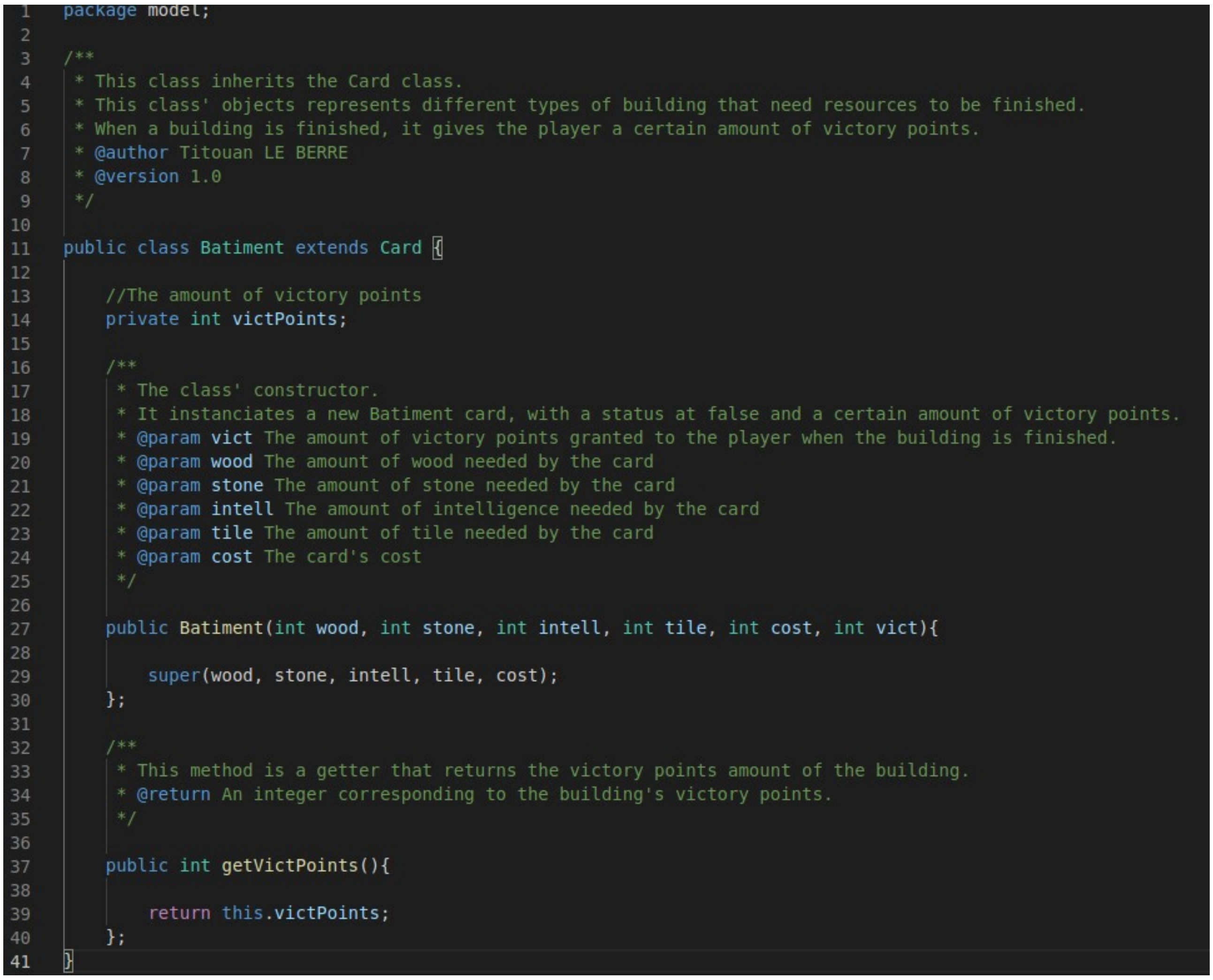


```java
package model;

/**
 * This class is inherits the Card class. This class' objects are Ouvrier cards.
 * Therefore, theses objects will produce resources and will be linked to Batiment and Machines cards.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public class Ouvrier extends Card{

    //The building the worker is linked with
    private Card workOn;

    /**
     * The class' constructor.
     * Instanciate a new Ouvrier.
     * @param wood The amount of wood produced by the card
     * @param stone The amount of stone produced by the card
     * @param intell The amount of intelligence produced by the card
     * @param tile The amount of tile produced by the card
     * @param cost The card's cost
     */

    public Ouvrier(int wood, int stone, int intell, int tile, int cost){

        super(wood, stone, intell, tile, cost);
    };


    /**
     * This method will make the object "product" resources for the Batiment/Machine card it is linked to.
     */

    public void product(){};

    /**
     * This method set the Card where the worker send the ressources he products.
     * @param bat The Card "where" the worker works.
     */

    public void setWorkOn(Card bat){};
```



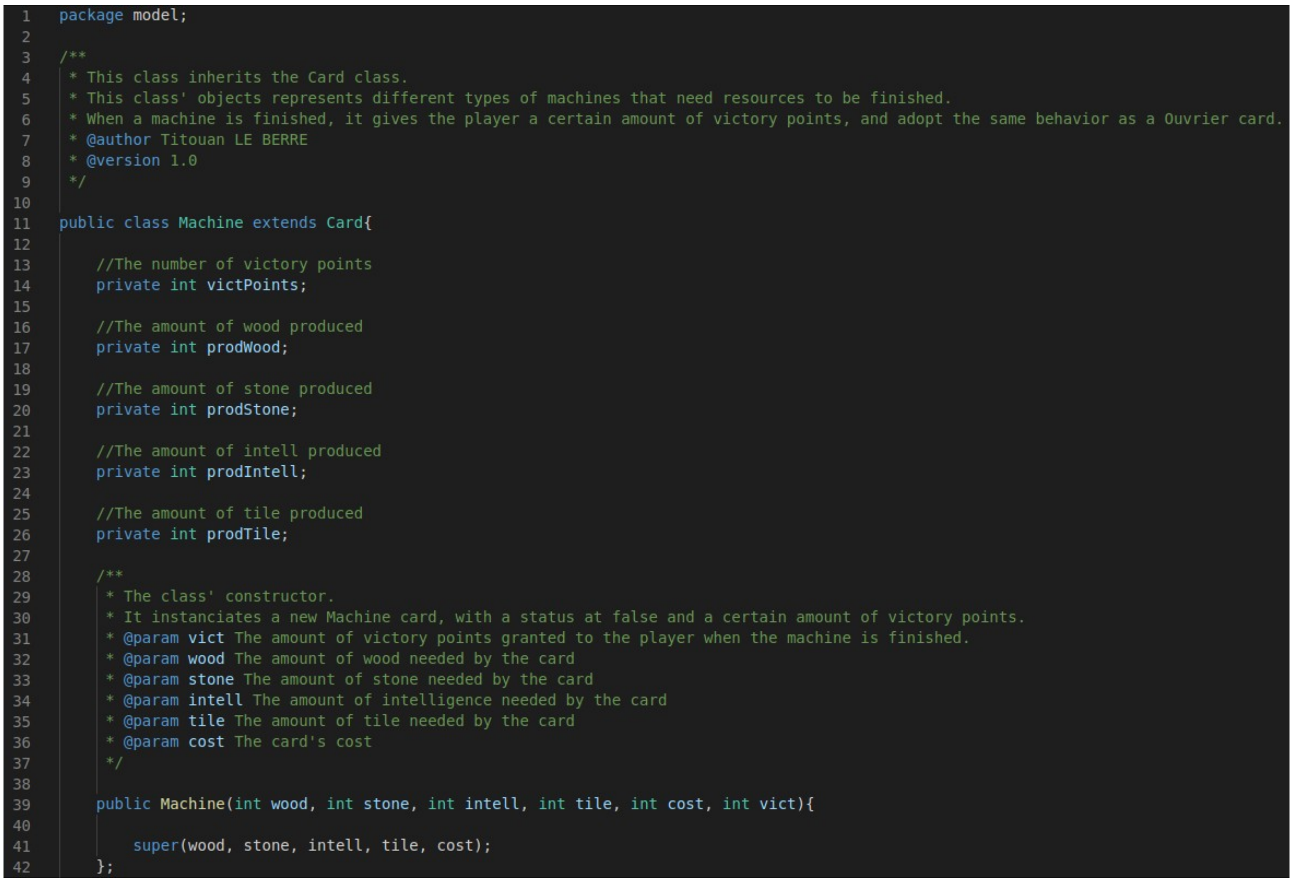
```java
    /**
     * This method returns the workOn attribute.
     * @return The workOn attribute
     */

    public Card getWorkOn(){

        return this.workOn;
    }
}
```

8) Batiment.java

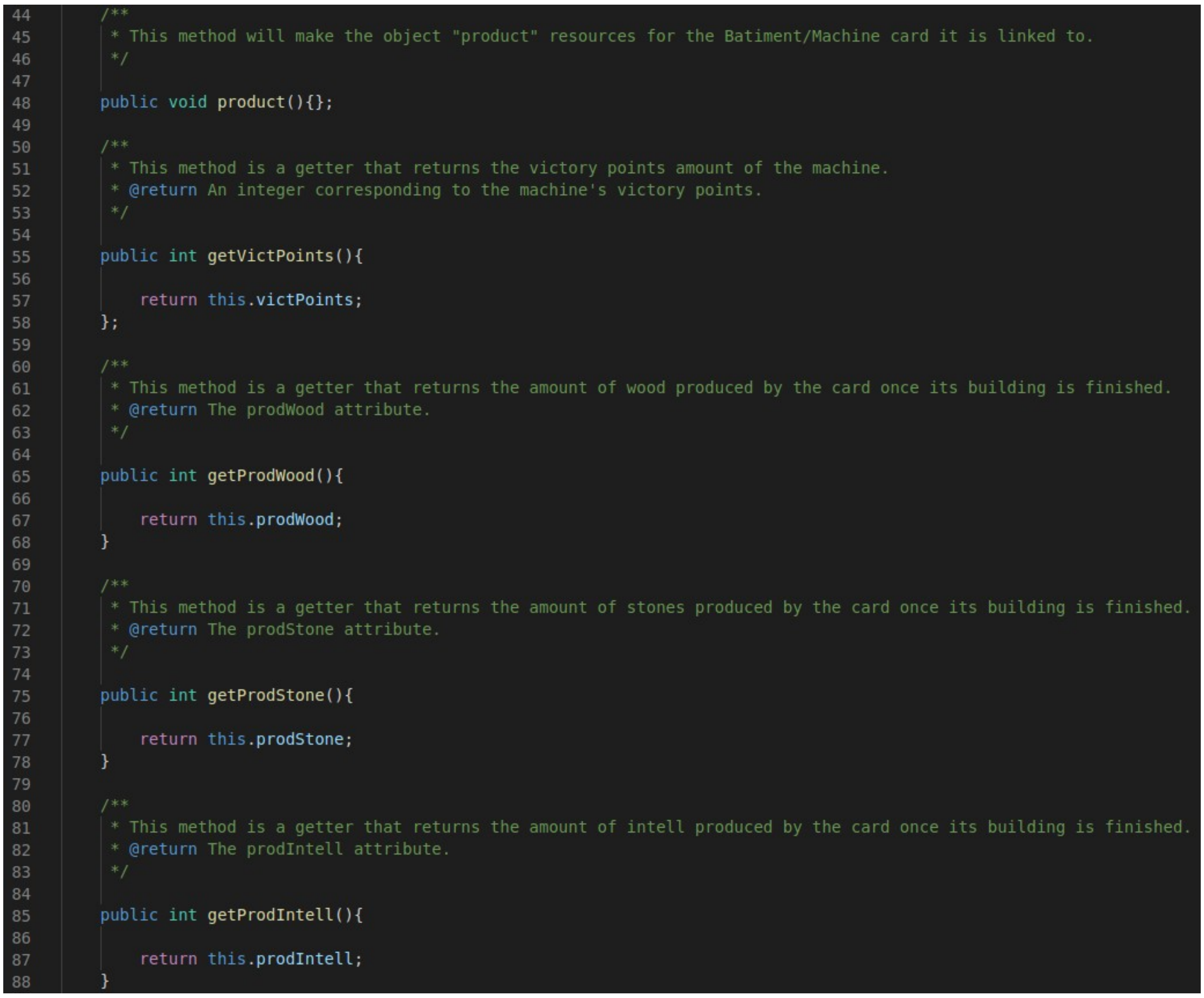


```java
package model;

/**
 * This class inherits the Card class.
 * This class' objects represents different types of building that need resources to be finished.
 * When a building is finished, it gives the player a certain amount of victory points.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public class Batiment extends Card {

    //The amount of victory points
    private int victPoints;

    /**
     * The class' constructor.
     * It instanciates a new Batiment card, with a status at false and a certain amount of victory points.
     * @param vict The amount of victory points granted to the player when the building is finished.
     * @param wood The amount of wood needed by the card
     * @param stone The amount of stone needed by the card
     * @param intell The amount of intelligence needed by the card
     * @param tile The amount of tile needed by the card
     * @param cost The card's cost
     */

    public Batiment(int wood, int stone, int intell, int tile, int cost, int vict){

        super(wood, stone, intell, tile, cost);
    };

    /**
     * This method is a getter that returns the victory points amount of the building.
     * @return An integer corresponding to the building's victory points.
     */

    public int getVictPoints(){

        return this.victPoints;
    };
}
```

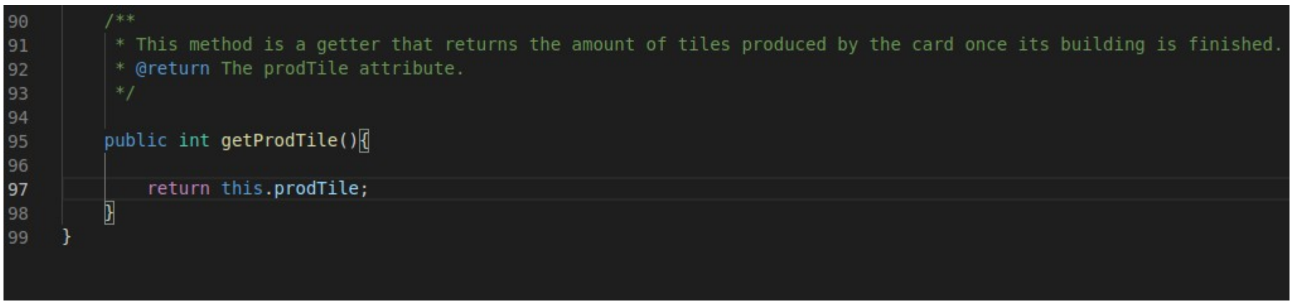
9) Machine.java



```java
package model;

/**
 * This class inherits the Card class.
 * This class' objects represents different types of machines that need resources to be finished.
 * When a machine is finished, it gives the player a certain amount of victory points, and adopt the same behavior as a Ouvrier card.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public class Machine extends Card{

    //The number of victory points
    private int victPoints;

    //The amount of wood produced
    private int prodWood;

    //The amount of stone produced
    private int prodStone;

    //The amount of intell produced
    private int prodIntell;

    //The amount of tile produced
    private int prodTile;

    /**
     * The class' constructor.
     * It instanciates a new Machine card, with a status at false and a certain amount of victory points.
     * @param vict The amount of victory points granted to the player when the machine is finished.
     * @param wood The amount of wood needed by the card
     * @param stone The amount of stone needed by the card
     * @param intell The amount of intelligence needed by the card
     * @param tile The amount of tile needed by the card
     * @param cost The card's cost
     */

    public Machine(int wood, int stone, int intell, int tile, int cost, int vict){

        super(wood, stone, intell, tile, cost);
    };
```

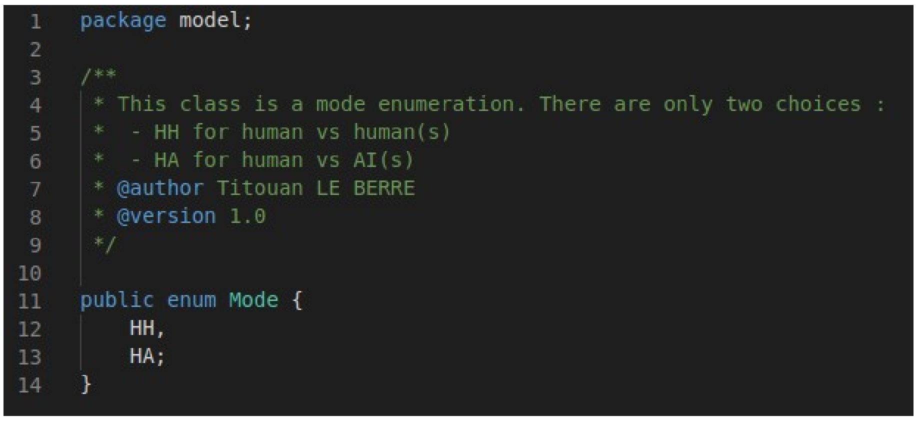


```java
    /**
     * This method will make the object "product" resources for the Batiment/Machine card it is linked to.
     */

    public void product(){};

    /**
     * This method is a getter that returns the victory points amount of the machine.
     * @return An integer corresponding to the machine's victory points.
     */

    public int getVictPoints(){

        return this.victPoints;
    };

    /**
     * This method is a getter that returns the amount of wood produced by the card once its building is finished.
     * @return The prodWood attribute.
     */

    public int getProdWood(){

        return this.prodWood;
    }

    /**
     * This method is a getter that returns the amount of stones produced by the card once its building is finished.
     * @return The prodStone attribute.
     */

    public int getProdStone(){

        return this.prodStone;
    }

    /**
     * This method is a getter that returns the amount of intell produced by the card once its building is finished.
     * @return The prodIntell attribute.
     */

    public int getProdIntell(){

        return this.prodIntell;
    }
```



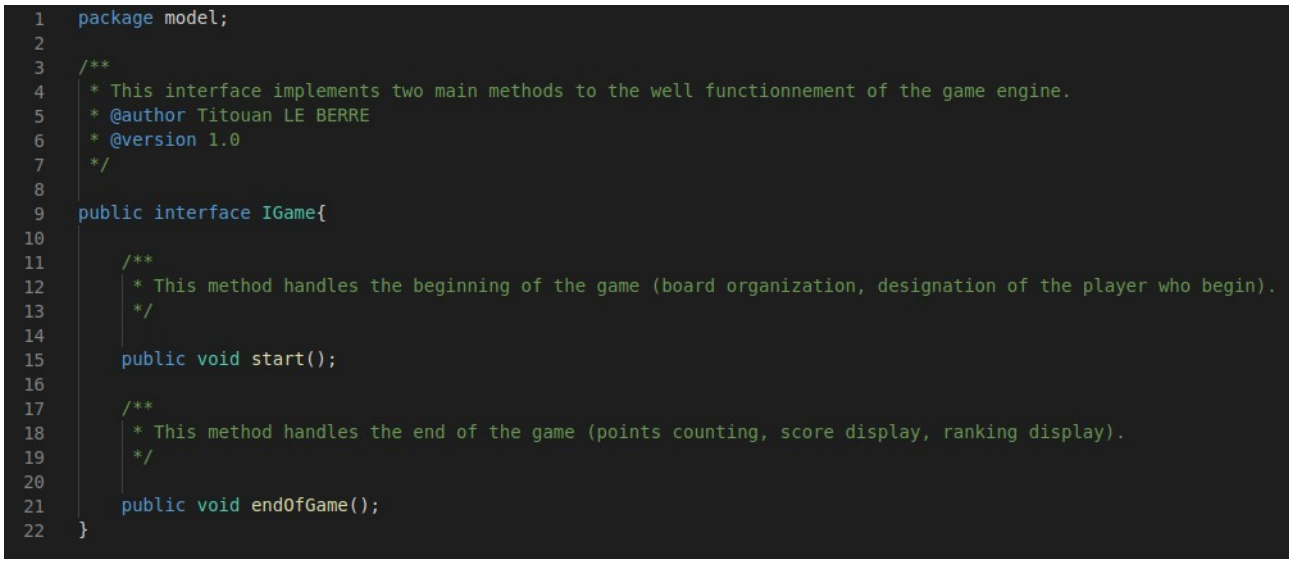
```java
    /**
     * This method is a getter that returns the amount of tiles produced by the card once its building is finished.
     * @return The prodTile attribute.
     */

    public int getProdTile(){

        return this.prodTile;
    }
}
```

10) Mode.java



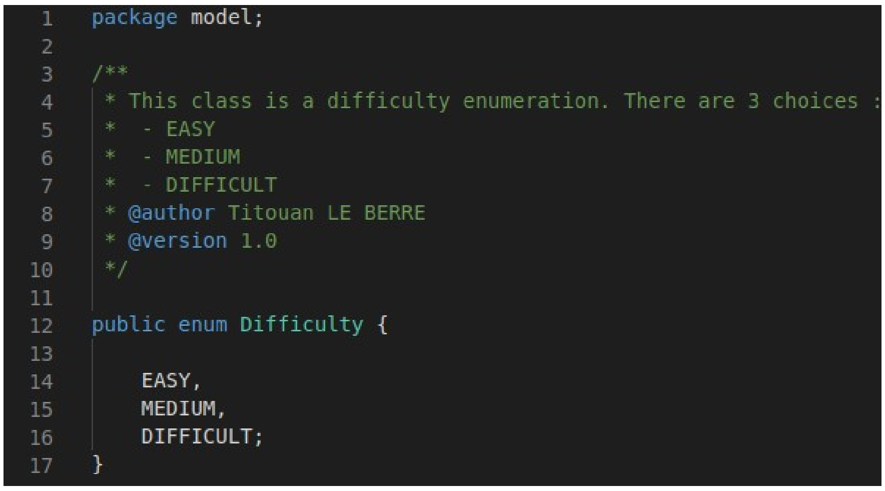
```java
package model;

/**
 * This class is a mode enumeration. There are only two choices :
 *  - HH for human vs human(s)
 *  - HA for human vs AI(s)
 * @author Titouan LE BERRE
 * @version 1.0
 */

public enum Mode {
    HH,
    HA;
}
```

11) Igame.java



```java
package model;

/**
 * This interface implements two main methods to the well functionnement of the game engine.
 * @author Titouan LE BERRE
 * @version 1.0
 */

public interface IGame{

    /**
     * This method handles the beginning of the game (board organization, designation of the player who begin).
     */

    public void start();

    /**
     * This method handles the end of the game (points counting, score display, ranking display).
     */

    public void endOfGame();
}
```

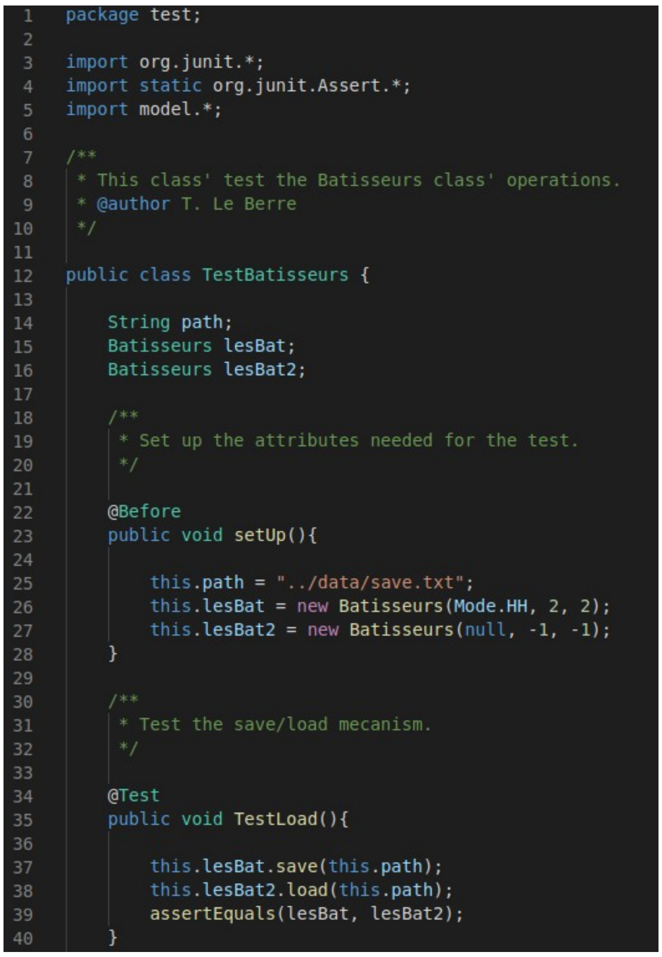
12) Difficulty.java



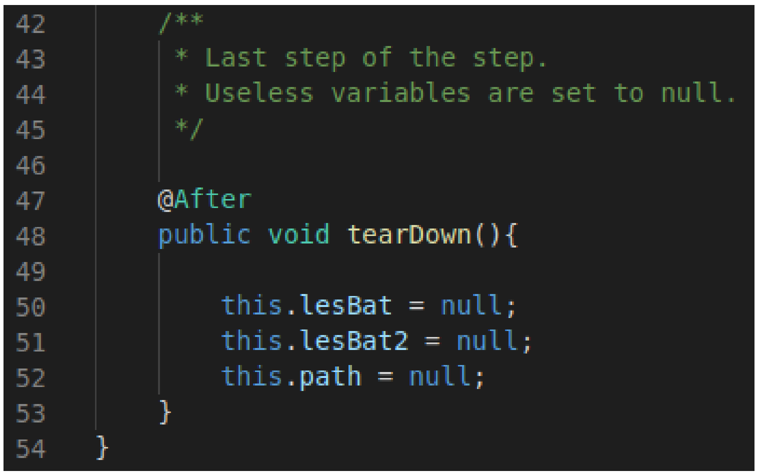
```java
package model;

/**
 * This class is a difficulty enumeration. There are 3 choices :
 *  - EASY
 *  - MEDIUM
 *  - DIFFICULT
 * @author Titouan LE BERRE
 * @version 1.0
 */

public enum Difficulty {

    EASY,
    MEDIUM,
    DIFFICULT;
}
```

B) Classes de tests Junit

1) TestBatisseurs.java

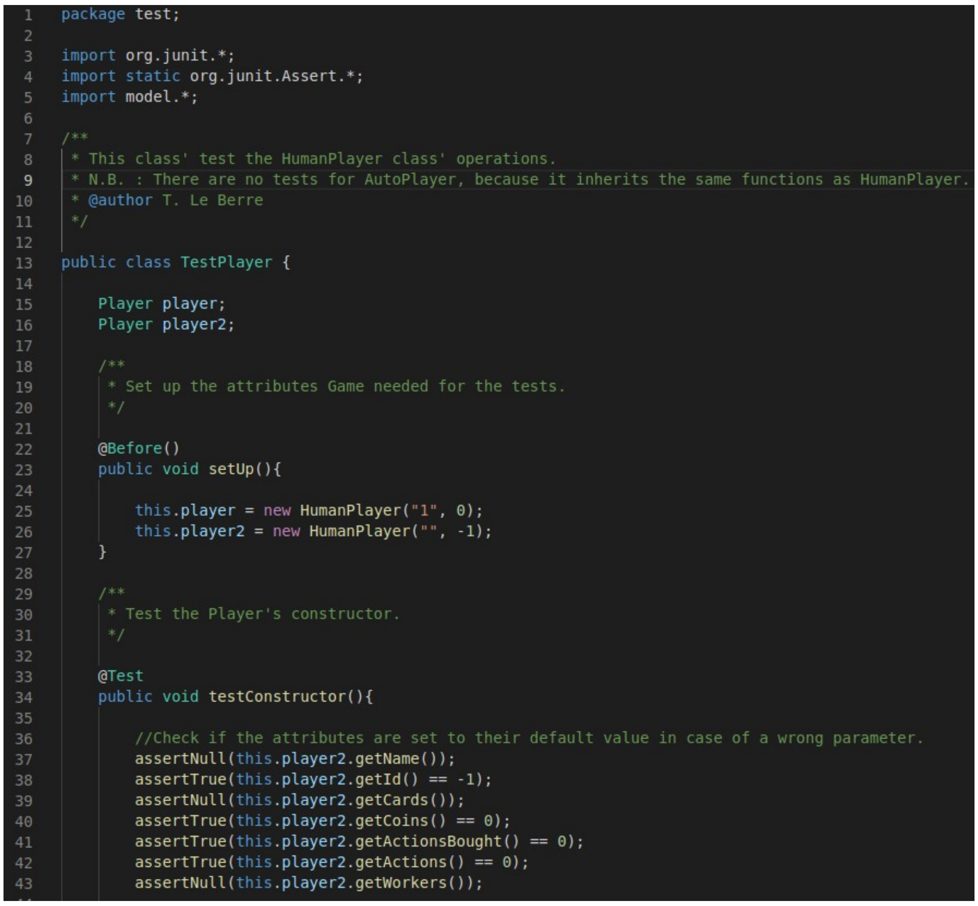


```java
package test;

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

/**
 * This class' test the Batisseurs class' operations.
 * @author T. Le Berre
 */

public class TestBatisseurs {

    String path;
    Batisseurs lesBat;
    Batisseurs lesBat2;

    /**
     * Set up the attributes needed for the test.
     */

    @Before
    public void setUp(){

        this.path = "../data/save.txt";
        this.lesBat = new Batisseurs(Mode.HH, 2, 2);
        this.lesBat2 = new Batisseurs(null, -1, -1);
    }

    /**
     * Test the save/load mecanism.
     */

    @Test
    public void TestLoad(){

        this.lesBat.save(this.path);
        this.lesBat2.load(this.path);
        assertEquals(lesBat, lesBat2);
    }
```

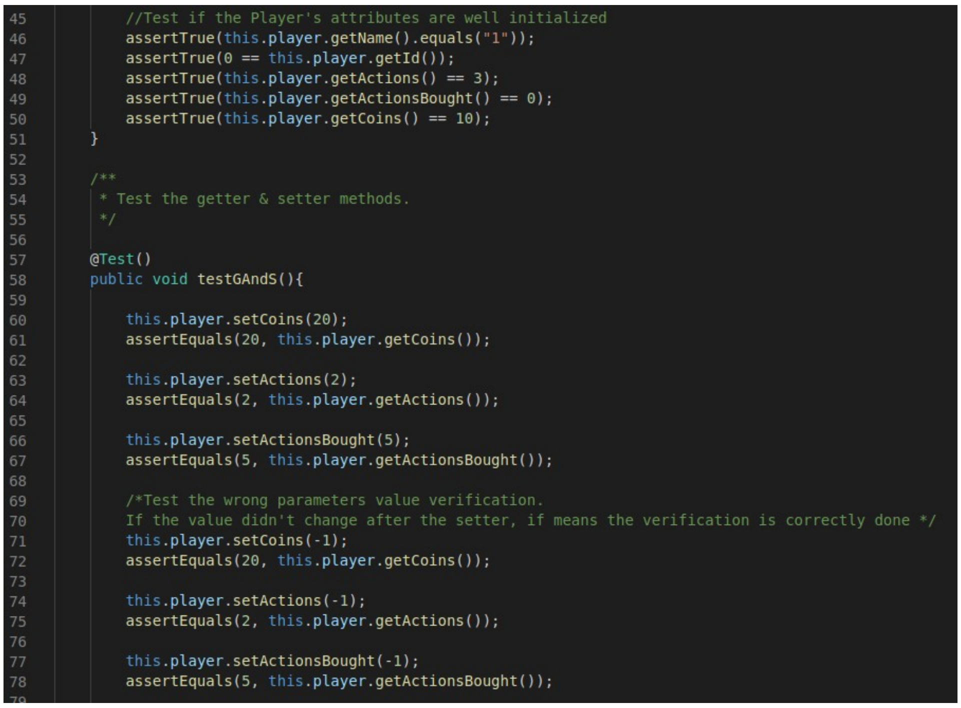


```java
    /**
     * Last step of the step.
     * Useless variables are set to null.
     */

    @After
    public void tearDown(){

        this.lesBat = null;
        this.lesBat2 = null;
        this.path = null;
    }
}
```

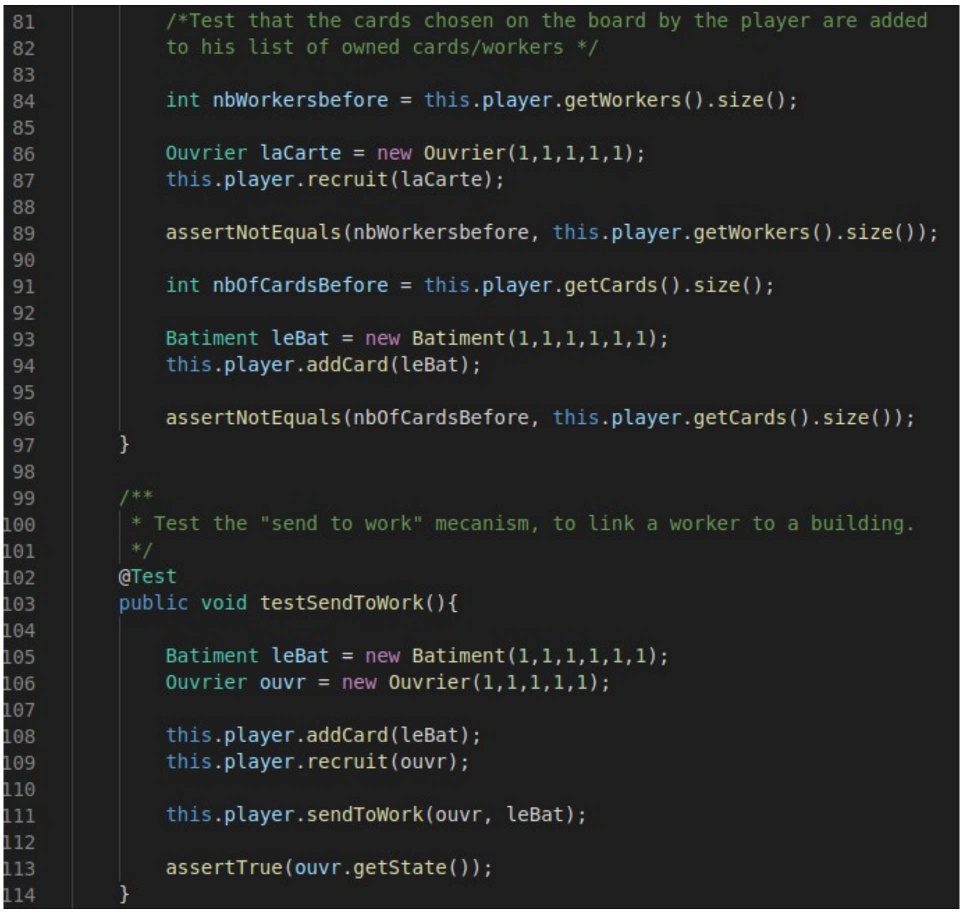
2) TestPlayer.java



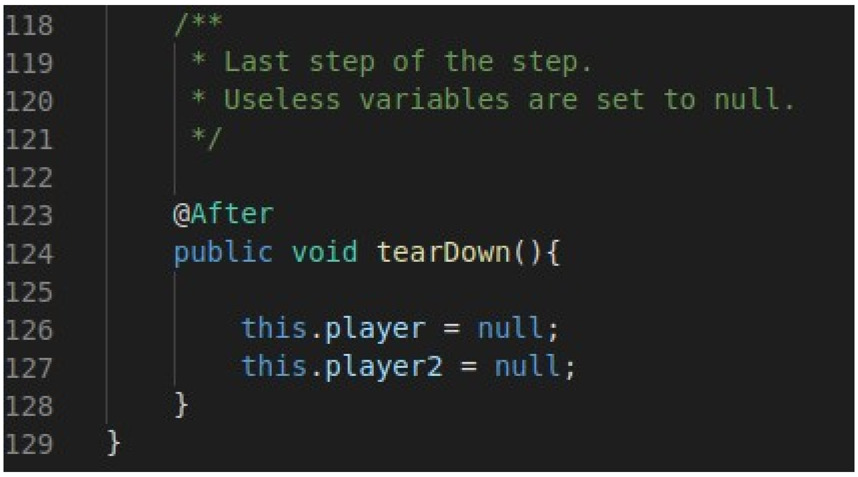
```java
package test;

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

/**
 * This class' test the HumanPlayer class' operations.
 * N.B. : There are no tests for AutoPlayer, because it inherits the same functions as HumanPlayer.
 * @author T. Le Berre
 */

public class TestPlayer {

    Player player;
    Player player2;

    /**
     * Set up the attributes Game needed for the tests.
     */

    @Before()
    public void setUp(){

        this.player = new HumanPlayer("1", 0);
        this.player2 = new HumanPlayer("", -1);
    }

    /**
     * Test the Player's constructor.
     */

    @Test
    public void testConstructor(){

        //Check if the attributes are set to their default value in case of a wrong parameter.
        assertNull(this.player2.getName());
        assertTrue(this.player2.getId() == -1);
        assertNull(this.player2.getCards());
        assertTrue(this.player2.getCoins() == 0);
        assertTrue(this.player2.getActionsBought() == 0);
        assertTrue(this.player2.getActions() == 0);
        assertNull(this.player2.getWorkers());
```



```java
        //Test if the Player's attributes are well initialized
        assertTrue(this.player.getName().equals("1"));
        assertTrue(0 == this.player.getId());
        assertTrue(this.player.getActions() == 3);
        assertTrue(this.player.getActionsBought() == 0);
        assertTrue(this.player.getCoins() == 10);
    }

    /**
     * Test the getter & setter methods.
     */

    @Test()
    public void testGAndS(){

        this.player.setCoins(20);
        assertEquals(20, this.player.getCoins());

        this.player.setActions(2);
        assertEquals(2, this.player.getActions());

        this.player.setActionsBought(5);
        assertEquals(5, this.player.getActionsBought());

        /*Test the wrong parameters value verification.
        If the value didn't change after the setter, if means the verification is correctly done */
        this.player.setCoins(-1);
        assertEquals(20, this.player.getCoins());

        this.player.setActions(-1);
        assertEquals(2, this.player.getActions());

        this.player.setActionsBought(-1);
        assertEquals(5, this.player.getActionsBought());
```

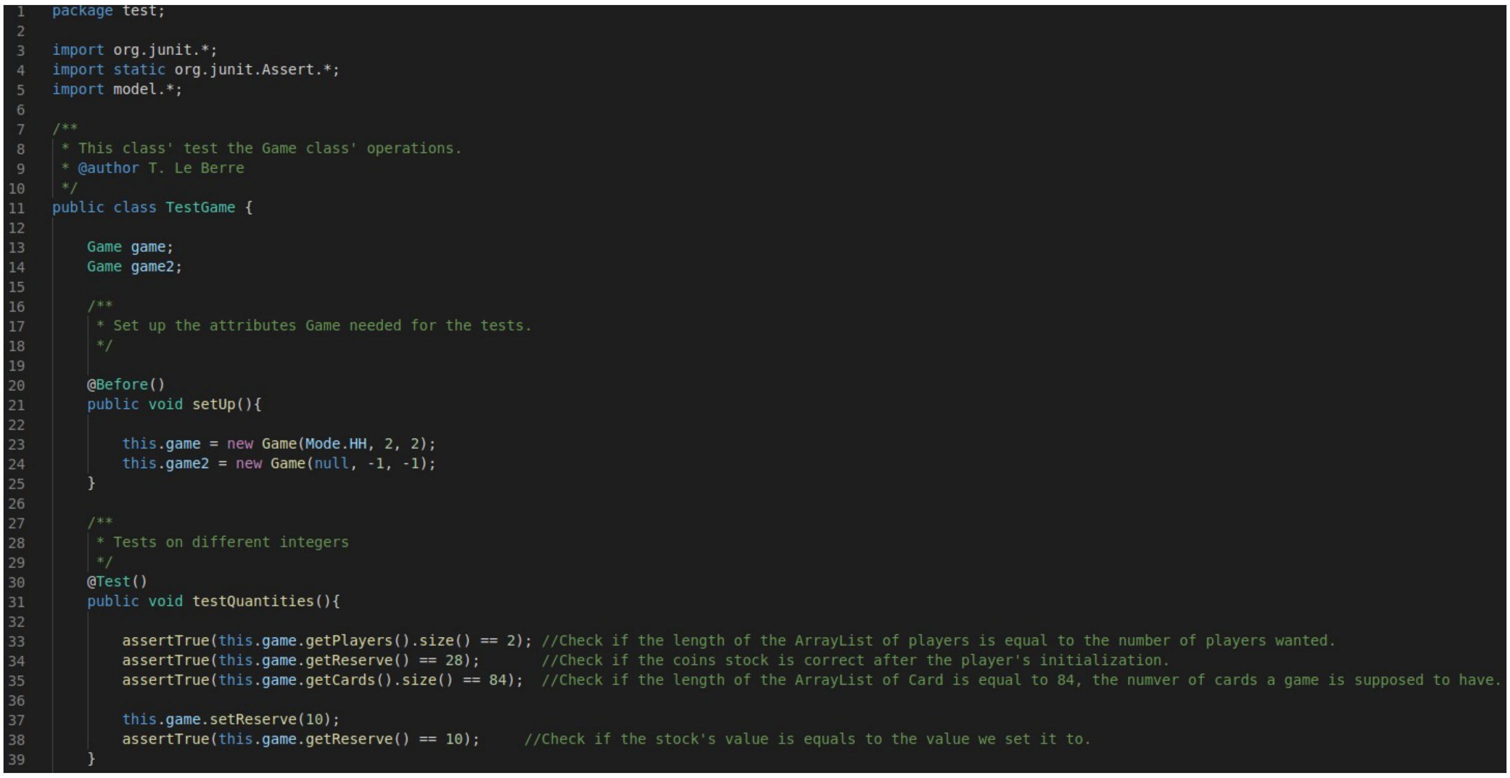


```java
        /*Test that the cards chosen on the board by the player are added
        to his list of owned cards/workers */

        int nbWorkersbefore = this.player.getWorkers().size();

        Ouvrier laCarte = new Ouvrier(1,1,1,1,1);
        this.player.recruit(laCarte);

        assertNotEquals(nbWorkersbefore, this.player.getWorkers().size());

        int nbOfCardsBefore = this.player.getCards().size();

        Batiment leBat = new Batiment(1,1,1,1,1,1);
        this.player.addCard(leBat);

        assertNotEquals(nbOfCardsBefore, this.player.getCards().size());
    }

    /**
     * Test the "send to work" mecanism, to link a worker to a building.
     */
    @Test
    public void testSendToWork(){

        Batiment leBat = new Batiment(1,1,1,1,1,1);
        Ouvrier ouvr = new Ouvrier(1,1,1,1,1);

        this.player.addCard(leBat);
        this.player.recruit(ouvr);

        this.player.sendToWork(ouvr, leBat);

        assertTrue(ouvr.getState());
    }
```

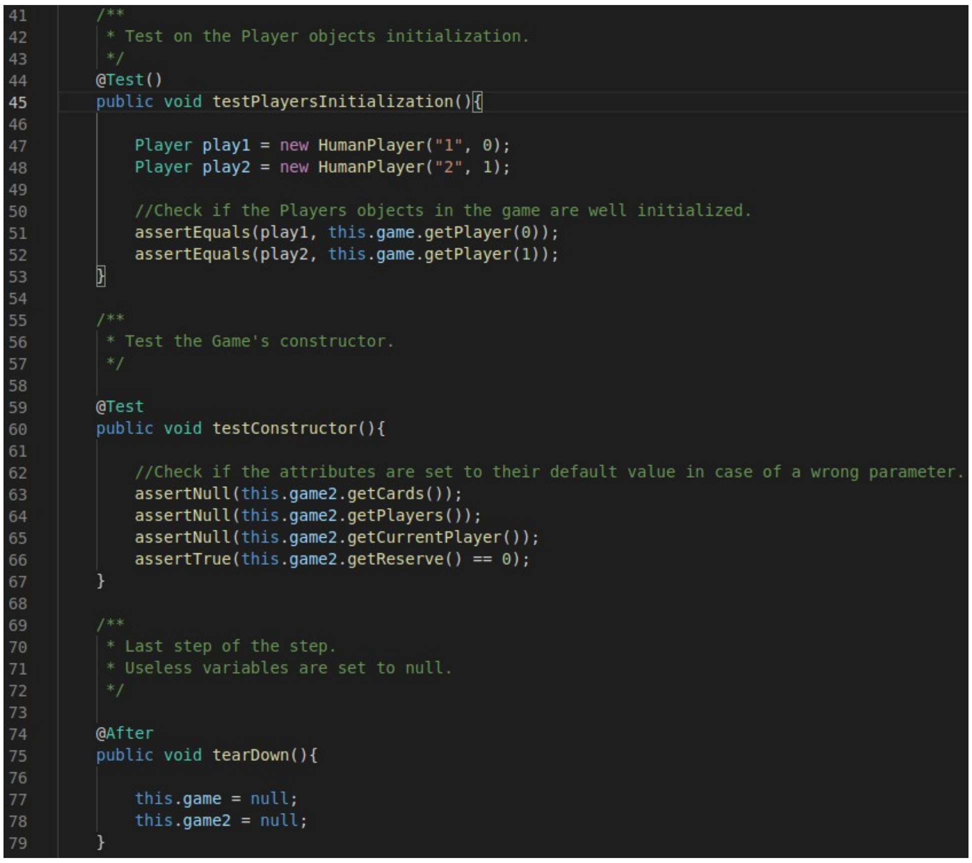


```java
    /**
     * Last step of the step.
     * Useless variables are set to null.
     */

    @After
    public void tearDown(){

        this.player = null;
        this.player2 = null;
    }
}
```

3) TestGame.java

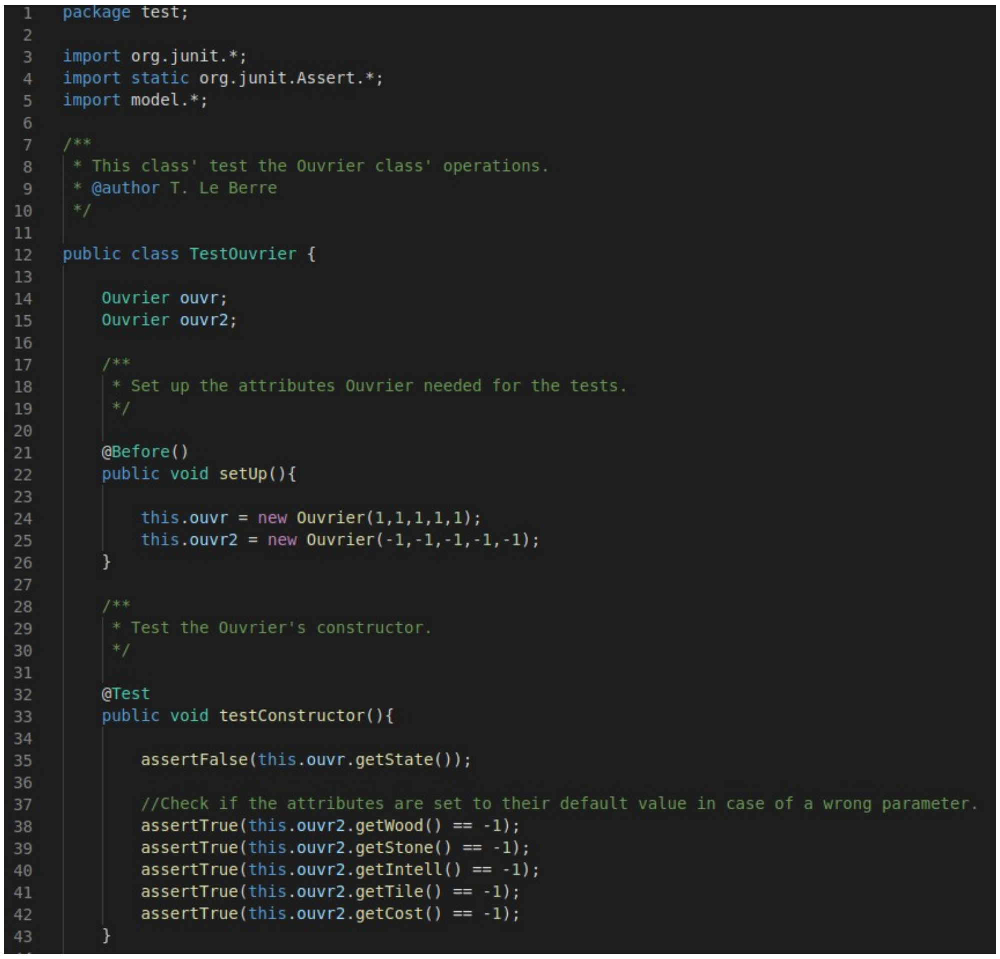


```java
package test;

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

/**
 * This class' test the Game class' operations.
 * @author T. Le Berre
 */
public class TestGame {

    Game game;
    Game game2;

    /**
     * Set up the attributes Game needed for the tests.
     */

    @Before()
    public void setUp(){

        this.game = new Game(Mode.HH, 2, 2);
        this.game2 = new Game(null, -1, -1);
    }

    /**
     * Tests on different integers
     */
    @Test()
    public void testQuantities(){

        assertTrue(this.game.getPlayers().size() == 2); //Check if the length of the ArrayList of players is equal to the number of players wanted.
        assertTrue(this.game.getReserve() == 28);       //Check if the coins stock is correct after the player's initialization.
        assertTrue(this.game.getCards().size() == 84);  //Check if the length of the ArrayList of Card is equal to 84, the numver of cards a game is supposed to have.

        this.game.setReserve(10);
        assertTrue(this.game.getReserve() == 10);     //Check if the stock's value is equals to the value we set it to.
    }
```

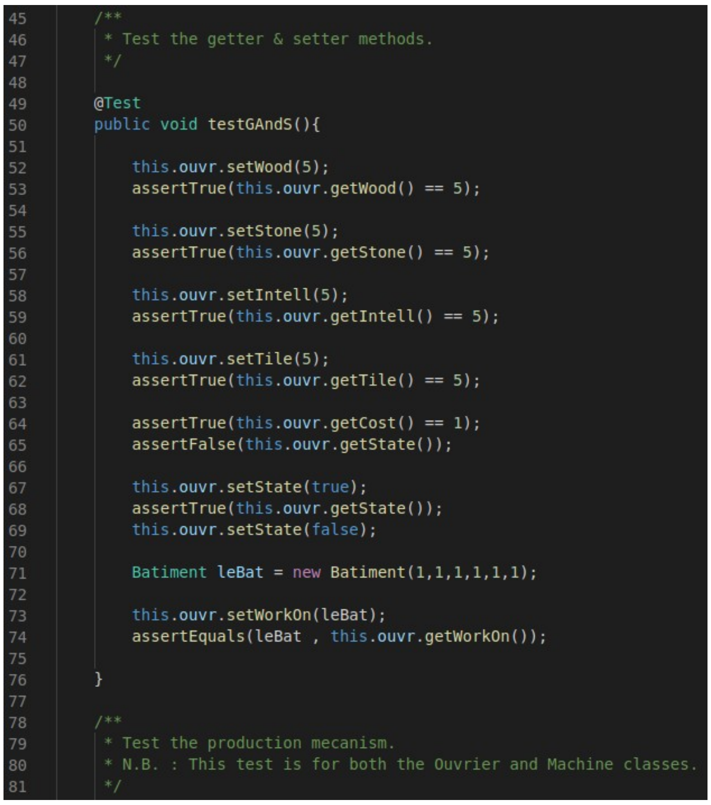


```java
    /**
     * Test on the Player objects initialization.
     */
    @Test()
    public void testPlayersInitialization(){

        Player play1 = new HumanPlayer("1", 0);
        Player play2 = new HumanPlayer("2", 1);

        //Check if the Players objects in the game are well initialized.
        assertEquals(play1, this.game.getPlayer(0));
        assertEquals(play2, this.game.getPlayer(1));
    }

    /**
     * Test the Game's constructor.
     */

    @Test
    public void testConstructor(){

        //Check if the attributes are set to their default value in case of a wrong parameter.
        assertNull(this.game2.getCards());
        assertNull(this.game2.getPlayers());
        assertNull(this.game2.getCurrentPlayer());
        assertTrue(this.game2.getReserve() == 0);
    }

    /**
     * Last step of the step.
     * Useless variables are set to null.
     */

    @After
    public void tearDown(){

        this.game = null;
        this.game2 = null;
    }
```

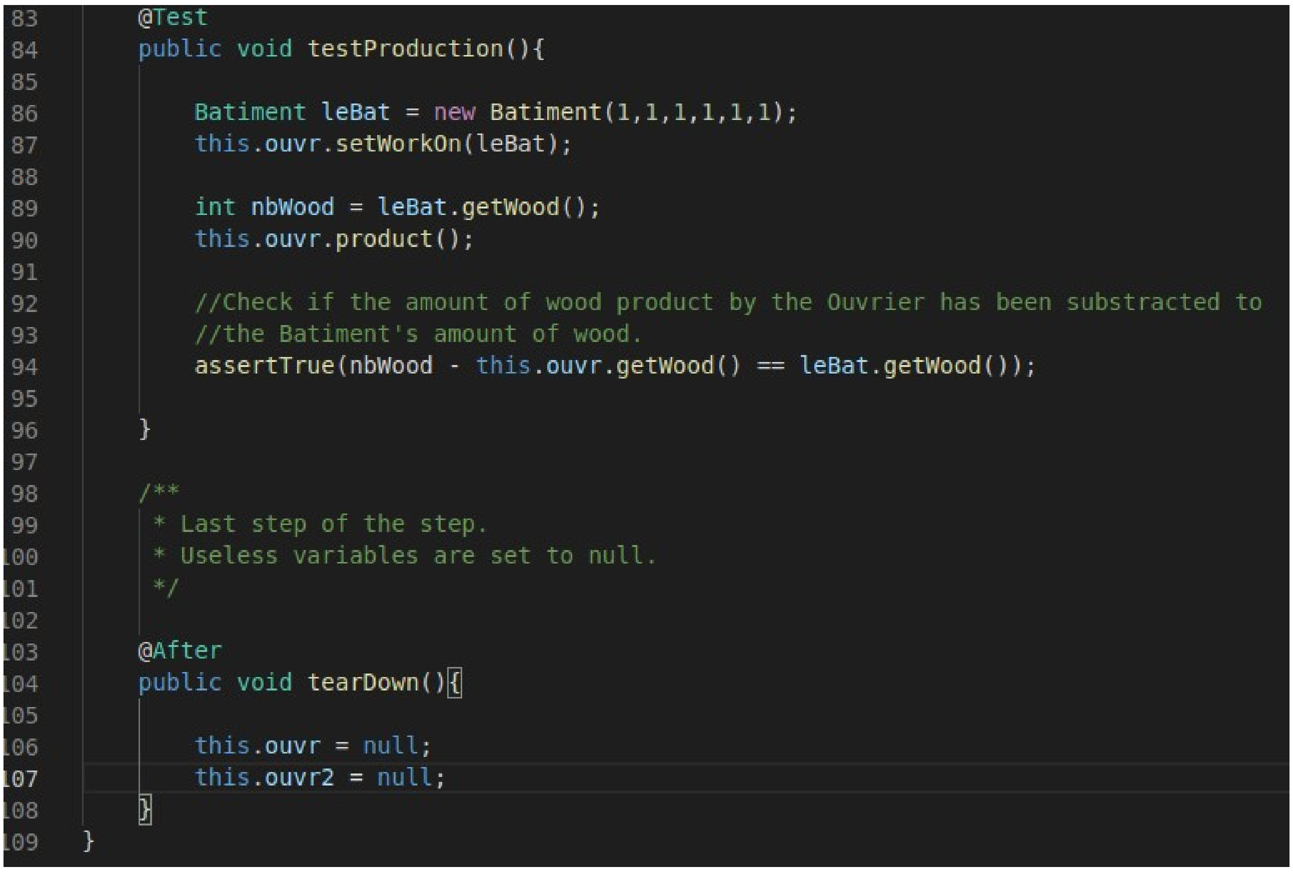
4) TestOuvrier.java



```java
package test;

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

/**
 * This class' test the Ouvrier class' operations.
 * @author T. Le Berre
 */

public class TestOuvrier {

    Ouvrier ouvr;
    Ouvrier ouvr2;

    /**
     * Set up the attributes Ouvrier needed for the tests.
     */

    @Before()
    public void setUp(){

        this.ouvr = new Ouvrier(1,1,1,1,1);
        this.ouvr2 = new Ouvrier(-1,-1,-1,-1,-1);
    }

    /**
     * Test the Ouvrier's constructor.
     */

    @Test
    public void testConstructor(){

        assertFalse(this.ouvr.getState());

        //Check if the attributes are set to their default value in case of a wrong parameter.
        assertTrue(this.ouvr2.getWood() == -1);
        assertTrue(this.ouvr2.getStone() == -1);
        assertTrue(this.ouvr2.getIntell() == -1);
        assertTrue(this.ouvr2.getTile() == -1);
        assertTrue(this.ouvr2.getCost() == -1);
    }
```

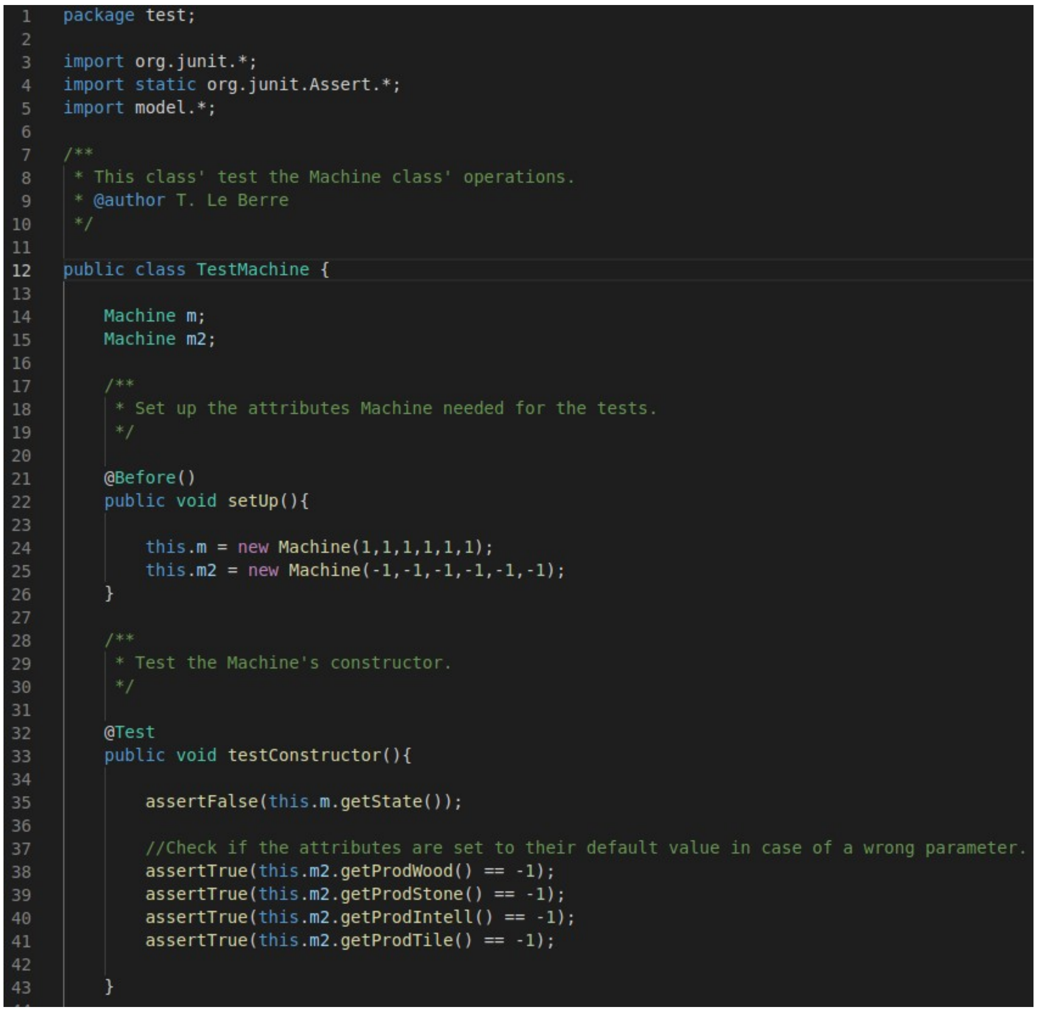


```java
    /**
     * Test the getter & setter methods.
     */

    @Test
    public void testGAndS(){

        this.ouvr.setWood(5);
        assertTrue(this.ouvr.getWood() == 5);

        this.ouvr.setStone(5);
        assertTrue(this.ouvr.getStone() == 5);

        this.ouvr.setIntell(5);
        assertTrue(this.ouvr.getIntell() == 5);

        this.ouvr.setTile(5);
        assertTrue(this.ouvr.getTile() == 5);

        assertTrue(this.ouvr.getCost() == 1);
        assertFalse(this.ouvr.getState());

        this.ouvr.setState(true);
        assertTrue(this.ouvr.getState());
        this.ouvr.setState(false);

        Batiment leBat = new Batiment(1,1,1,1,1,1);

        this.ouvr.setWorkOn(leBat);
        assertEquals(leBat , this.ouvr.getWorkOn());

    }

    /**
     * Test the production mecanism.
     * N.B. : This test is for both the Ouvrier and Machine classes.
     */
```

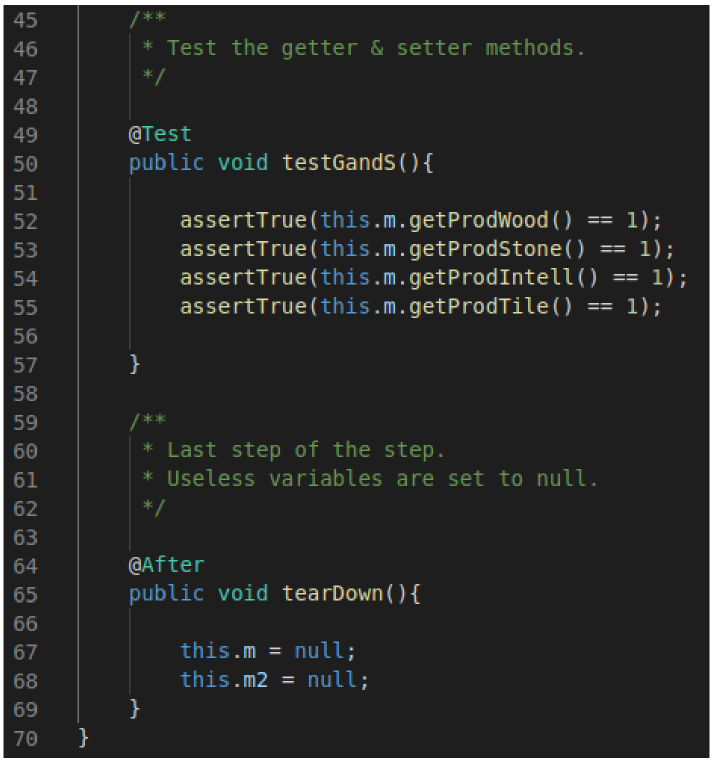


```java
    @Test
    public void testProduction(){

        Batiment leBat = new Batiment(1,1,1,1,1,1);
        this.ouvr.setWorkOn(leBat);

        int nbWood = leBat.getWood();
        this.ouvr.product();

        //Check if the amount of wood product by the Ouvrier has been substracted to
        //the Batiment's amount of wood.
        assertTrue(nbWood - this.ouvr.getWood() == leBat.getWood());

    }

    /**
     * Last step of the step.
     * Useless variables are set to null.
     */

    @After
    public void tearDown(){

        this.ouvr = null;
        this.ouvr2 = null;
    }
}
```

5) TestMachine.java

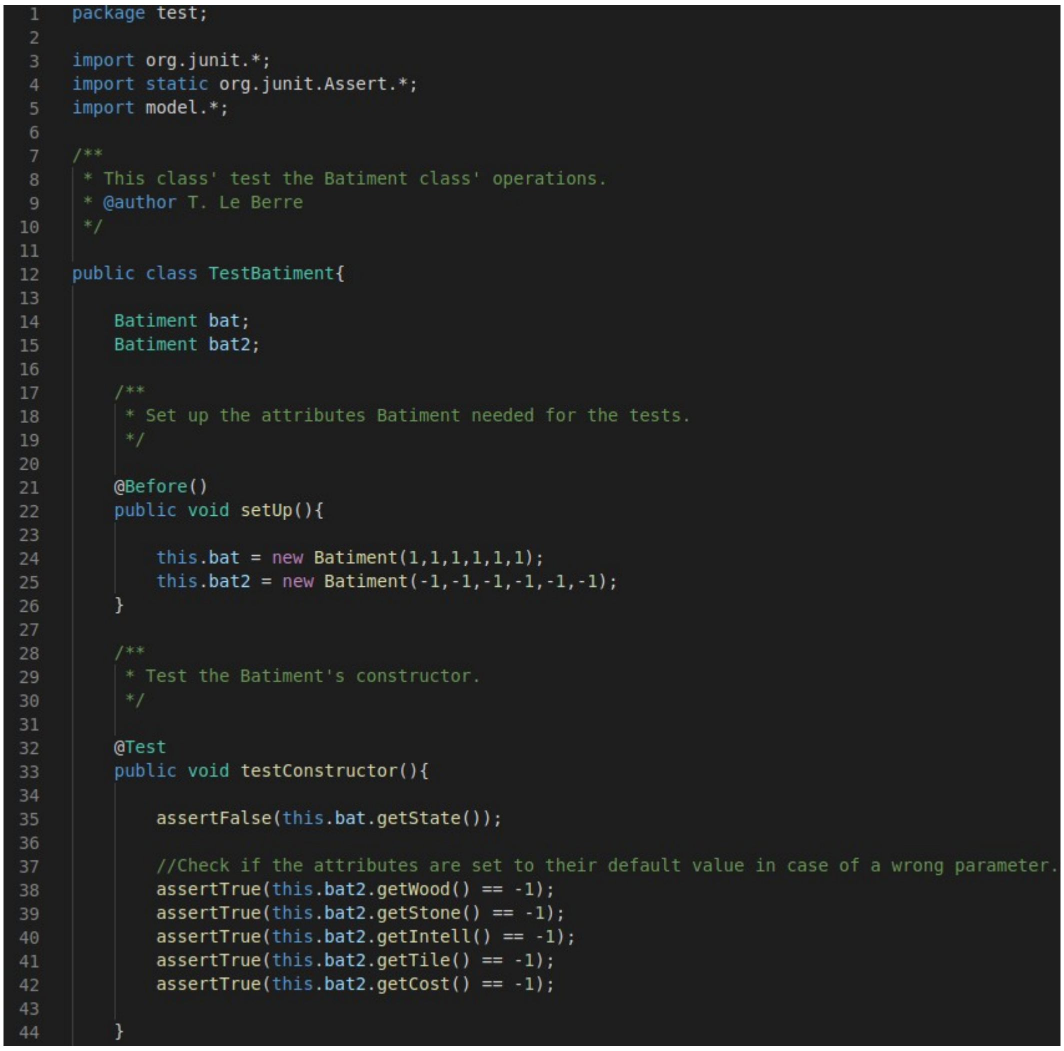


```java
package test;

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

/**
 * This class' test the Machine class' operations.
 * @author T. Le Berre
 */

public class TestMachine {

    Machine m;
    Machine m2;

    /**
     * Set up the attributes Machine needed for the tests.
     */

    @Before()
    public void setUp(){

        this.m = new Machine(1,1,1,1,1,1);
        this.m2 = new Machine(-1,-1,-1,-1,-1,-1);
    }

    /**
     * Test the Machine's constructor.
     */

    @Test
    public void testConstructor(){

        assertFalse(this.m.getState());

        //Check if the attributes are set to their default value in case of a wrong parameter.
        assertTrue(this.m2.getProdWood() == -1);
        assertTrue(this.m2.getProdStone() == -1);
        assertTrue(this.m2.getProdIntell() == -1);
        assertTrue(this.m2.getProdTile() == -1);

    }
```

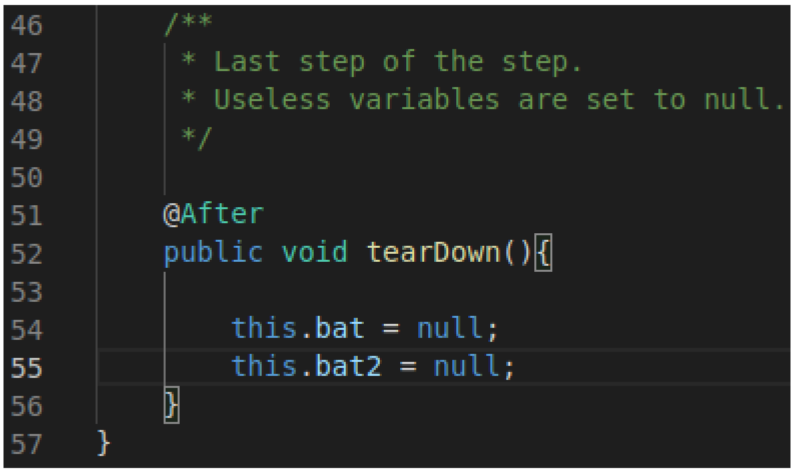


```java
    /**
     * Test the getter & setter methods.
     */

    @Test
    public void testGandS(){

        assertTrue(this.m.getProdWood() == 1);
        assertTrue(this.m.getProdStone() == 1);
        assertTrue(this.m.getProdIntell() == 1);
        assertTrue(this.m.getProdTile() == 1);

    }

    /**
     * Last step of the step.
     * Useless variables are set to null.
     */

    @After
    public void tearDown(){

        this.m = null;
        this.m2 = null;
    }
}
```

6) TestBatiment.java



```java
package test;

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

/**
 * This class' test the Batiment class' operations.
 * @author T. Le Berre
 */

public class TestBatiment{

    Batiment bat;
    Batiment bat2;

    /**
     * Set up the attributes Batiment needed for the tests.
     */

    @Before()
    public void setUp(){

        this.bat = new Batiment(1,1,1,1,1,1);
        this.bat2 = new Batiment(-1,-1,-1,-1,-1,-1);
    }

    /**
     * Test the Batiment's constructor.
     */

    @Test
    public void testConstructor(){

        assertFalse(this.bat.getState());

        //Check if the attributes are set to their default value in case of a wrong parameter.
        assertTrue(this.bat2.getWood() == -1);
        assertTrue(this.bat2.getStone() == -1);
        assertTrue(this.bat2.getIntell() == -1);
        assertTrue(this.bat2.getTile() == -1);
        assertTrue(this.bat2.getCost() == -1);

    }
```

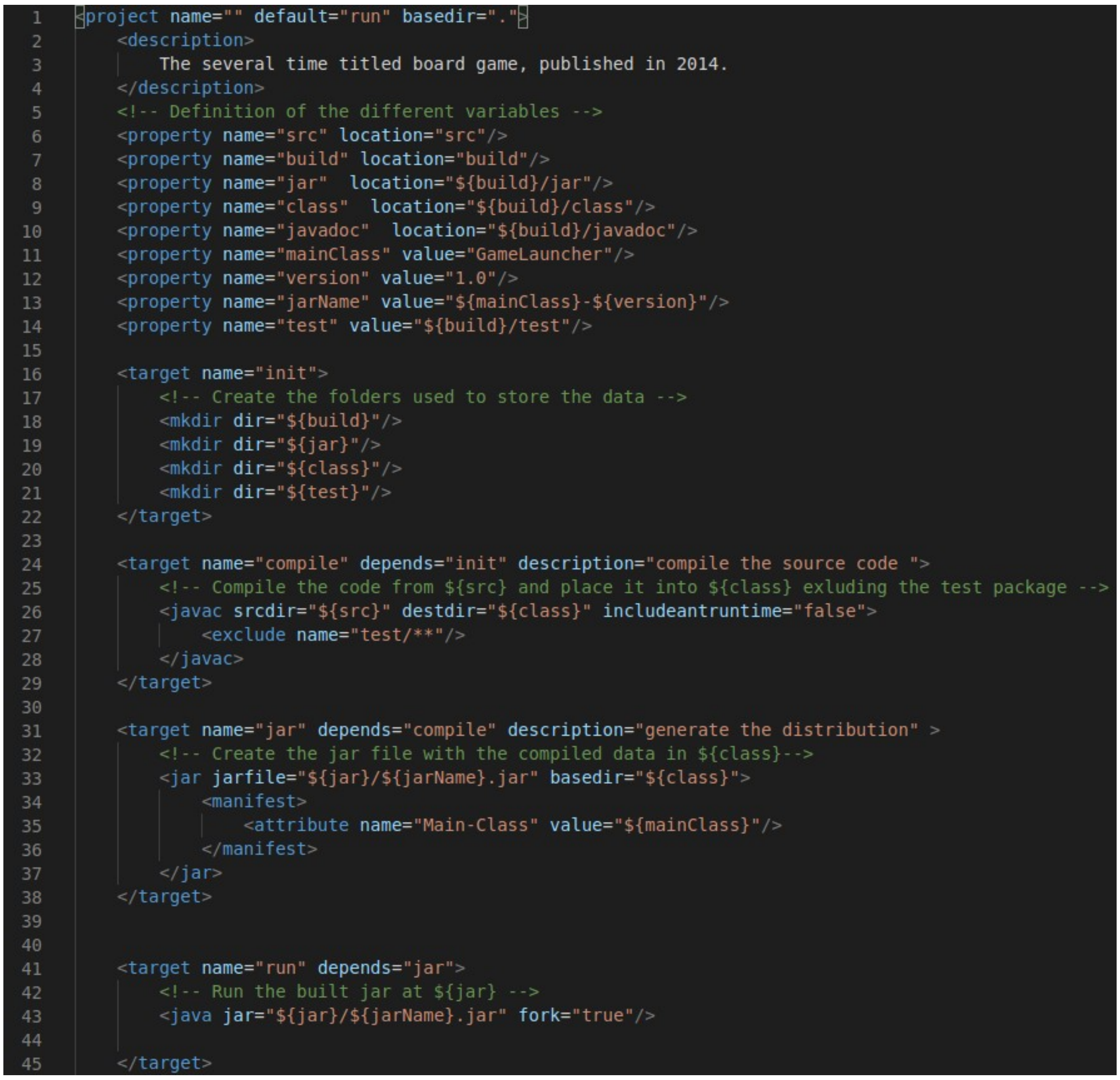


```java
    /**
     * Last step of the step.
     * Useless variables are set to null.
     */

    @After
    public void tearDown(){

        this.bat = null;
        this.bat2 = null;
    }
}
```

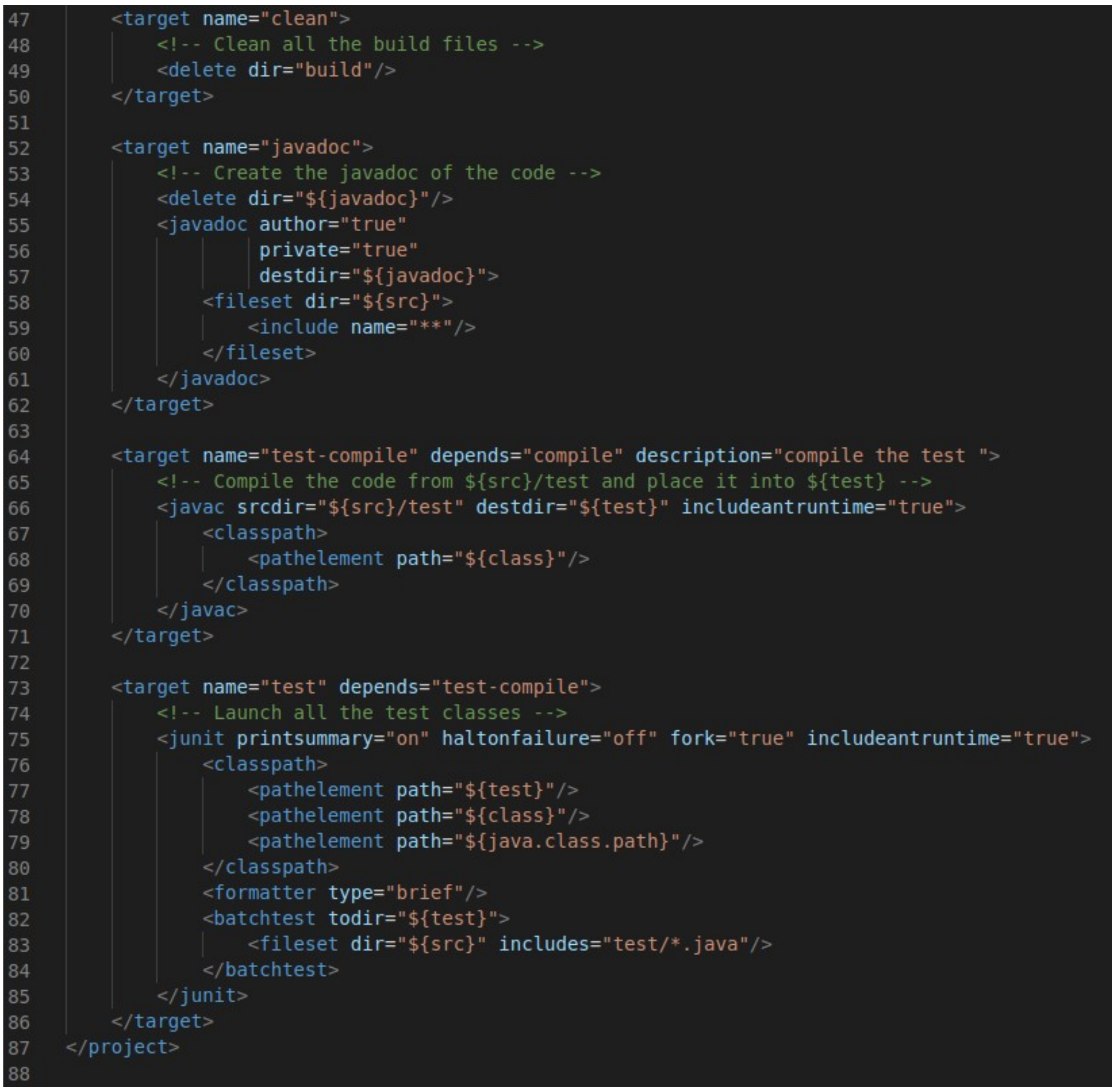
VI) ANT et fichier de construction

Le projet open source ANT permet l’automatisation des tâches de compilation, de documentation, de tests et d’archivage sous forme distribuable.

Un fichier de construction *build.xml* est nécessaire pour cette automatisation, celui utilisé pour ce projet figure ci-dessous :



```xml
<project name="" default="run" basedir=".">
    <description>
        The several time titled board game, published in 2014.
    </description>
    <!-- Definition of the different variables -->
    <property name="src" location="src"/>
    <property name="build" location="build"/>
    <property name="jar"  location="${build}/jar"/>
    <property name="class"  location="${build}/class"/>
    <property name="javadoc"  location="${build}/javadoc"/>
    <property name="mainClass" value="GameLauncher"/>
    <property name="version" value="1.0"/>
    <property name="jarName" value="${mainClass}-${version}"/>
    <property name="test" value="${build}/test"/>

    <target name="init">
        <!-- Create the folders used to store the data -->
        <mkdir dir="${build}"/>
        <mkdir dir="${jar}"/>
        <mkdir dir="${class}"/>
        <mkdir dir="${test}"/>
    </target>

    <target name="compile" depends="init" description="compile the source code ">
        <!-- Compile the code from ${src} and place it into ${class} exluding the test package -->
        <javac srcdir="${src}" destdir="${class}" includeantruntime="false">
            <exclude name="test/**"/>
        </javac>
    </target>

    <target name="jar" depends="compile" description="generate the distribution" >
        <!-- Create the jar file with the compiled data in ${class}-->
        <jar jarfile="${jar}/${jarName}.jar" basedir="${class}">
            <manifest>
                <attribute name="Main-Class" value="${mainClass}"/>
            </manifest>
        </jar>
    </target>


    <target name="run" depends="jar">
        <!-- Run the built jar at ${jar} -->
        <java jar="${jar}/${jarName}.jar" fork="true"/>

    </target>
```



```xml
    <target name="clean">
        <!-- Clean all the build files -->
        <delete dir="build"/>
    </target>

    <target name="javadoc">
        <!-- Create the javadoc of the code -->
        <delete dir="${javadoc}"/>
        <javadoc author="true"
                 private="true"
                 destdir="${javadoc}">
            <fileset dir="${src}">
                <include name="**"/>
            </fileset>
        </javadoc>
    </target>

    <target name="test-compile" depends="compile" description="compile the test ">
        <!-- Compile the code from ${src}/test and place it into ${test} -->
        <javac srcdir="${src}/test" destdir="${test}" includeantruntime="true">
            <classpath>
                <pathelement path="${class}"/>
            </classpath>
        </javac>
    </target>

    <target name="test" depends="test-compile">
        <!-- Launch all the test classes -->
        <junit printsummary="on" haltonfailure="off" fork="true" includeantruntime="true">
            <classpath>
                <pathelement path="${test}"/>
                <pathelement path="${class}"/>
                <pathelement path="${java.class.path}"/>
            </classpath>
            <formatter type="brief"/>
            <batchtest todir="${test}">
                <fileset dir="${src}" includes="test/*.java"/>
            </batchtest>
        </junit>
    </target>
</project>
```