M2107 - Projet de programmation

Projet de programmation

Les Bâtisseurs : Moyen-Âge

Cahier d'analyse et de conception

BOCANDÉ Yvan, 1ère année – Groupe A1, Année 2020-2021

Table des matières

1) Diagramme de classe d’analyse

Voici ci-dessous le diagramme de classe et d’analyse, il permet d’avoir une vue d’ensemble du projet avec plus de légèreté que le diagramme de classes de conception.
Le diagramme est basé sur le modèle MCV, il est donc composé de 3 Packages.

- Model : contient le moteur du jeu
- View : contient les interfaces graphiques
- Contrôler : écoute les évènements des classes graphiques

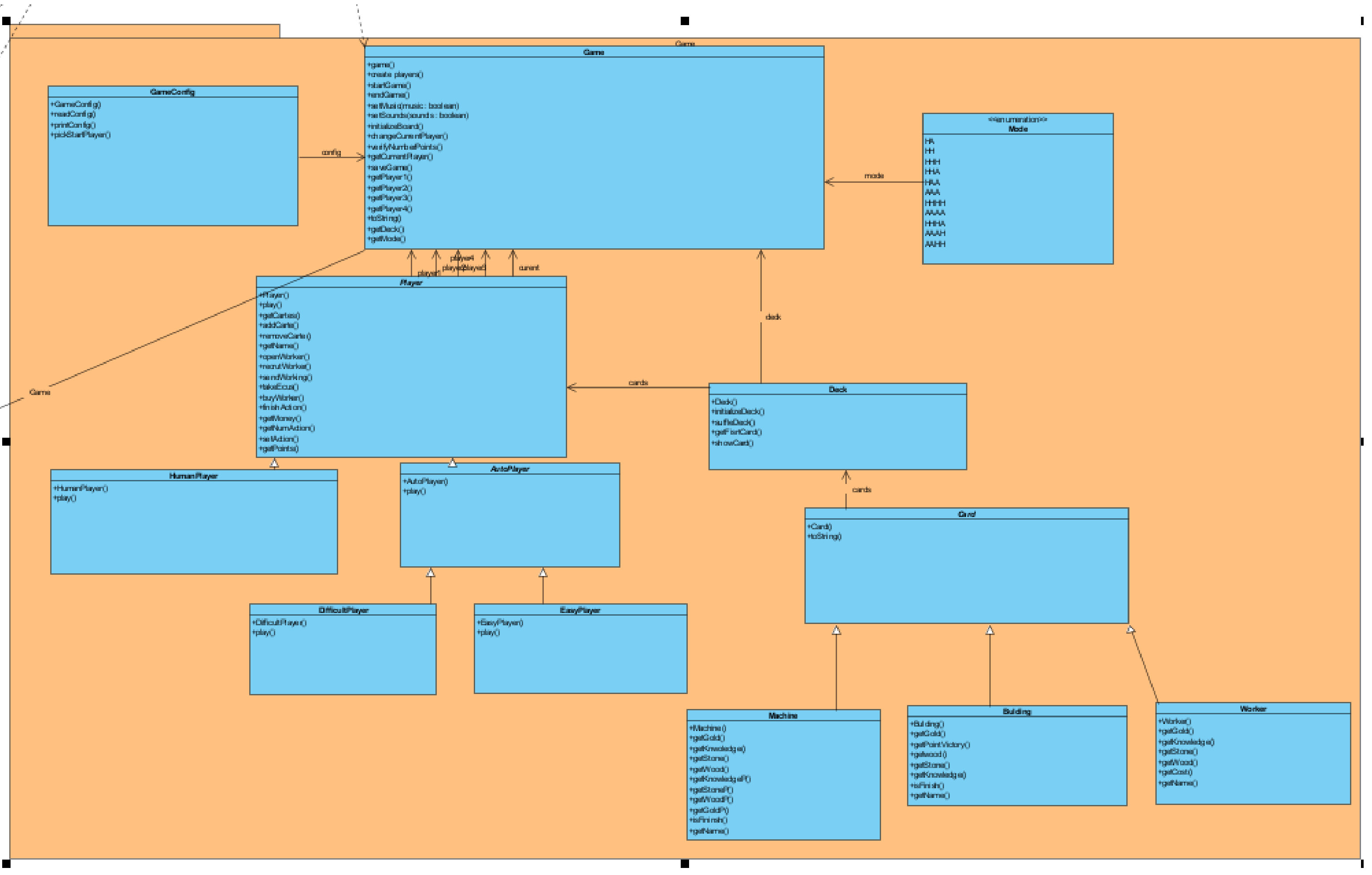


Diagramme d’analyse à retrouver en annexe (1)

Le diagramme étant petit, merci de m’adresser un mail à l’adresse suivante si vous voulez l’avoir en version Visual paradigme pour une meilleure lecture : yvanbocande@outlook.com

2) Diagramme de classe de conception

Voici le diagramme de classe de conception. L’objectif ici, n’était pas de faire la conception parfaite mais de s’approcher au maximum de la conception finale et de pouvoir rajouté des fonctionnalités dans celui-ci facilement.

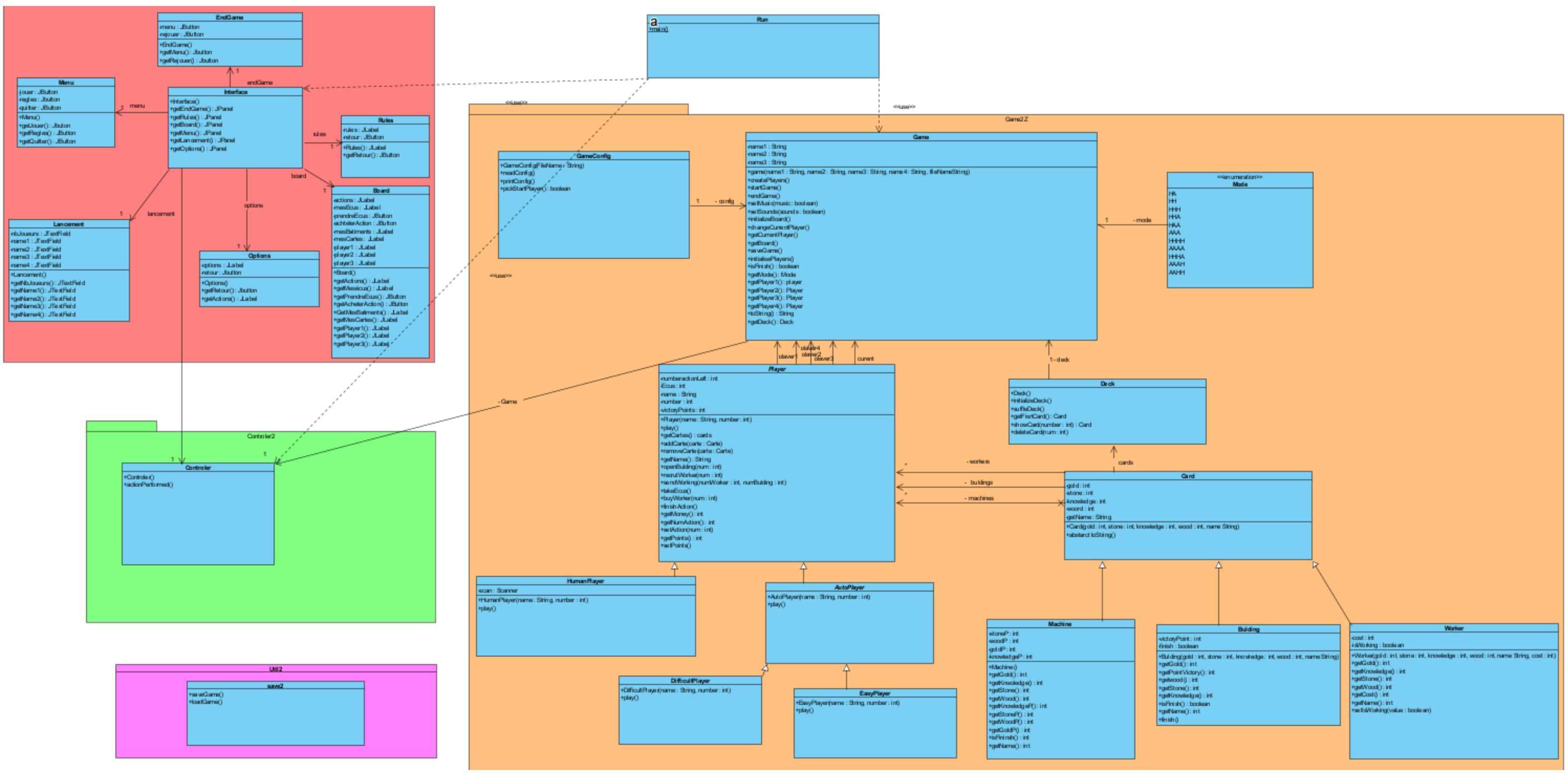
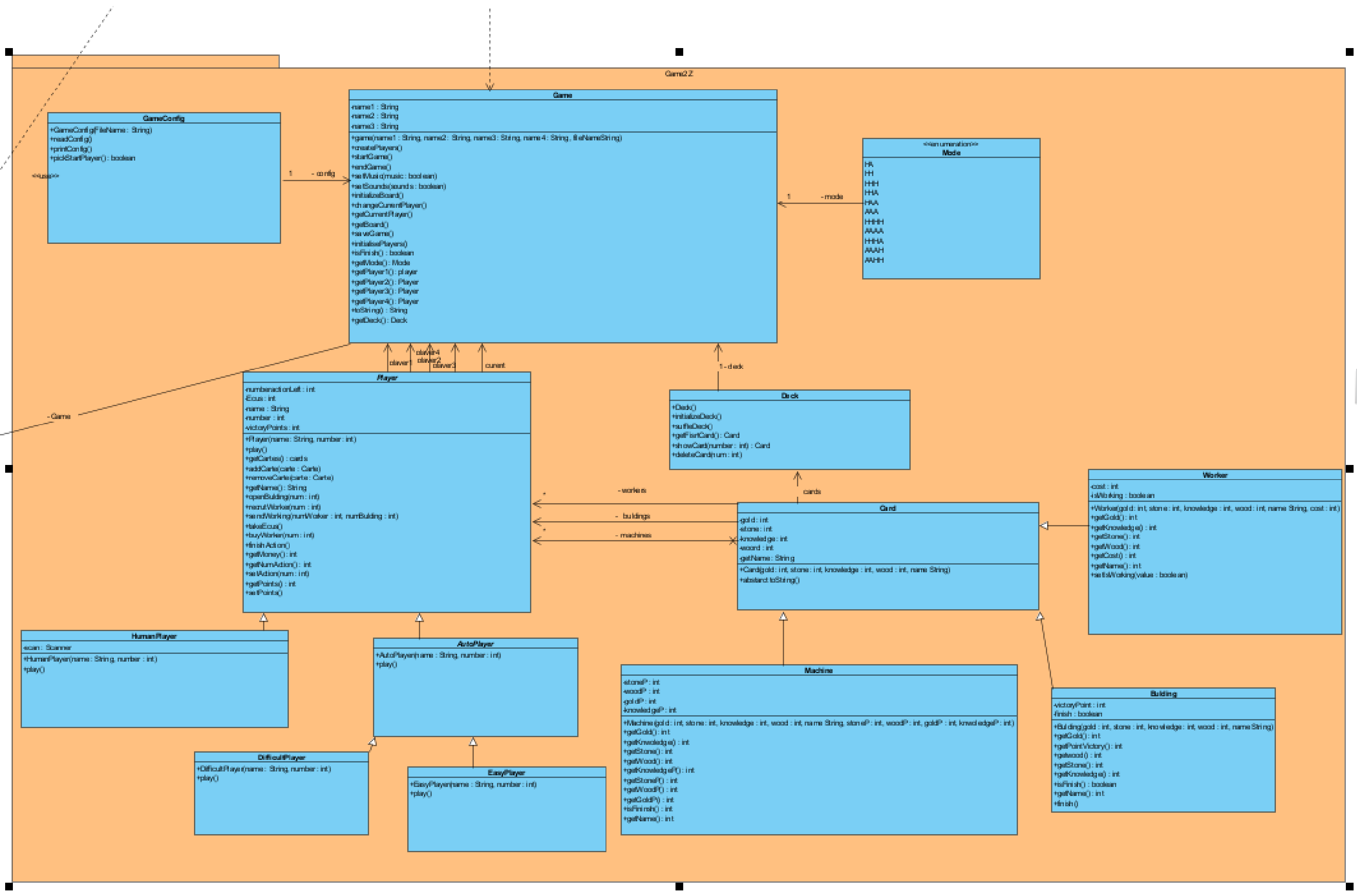
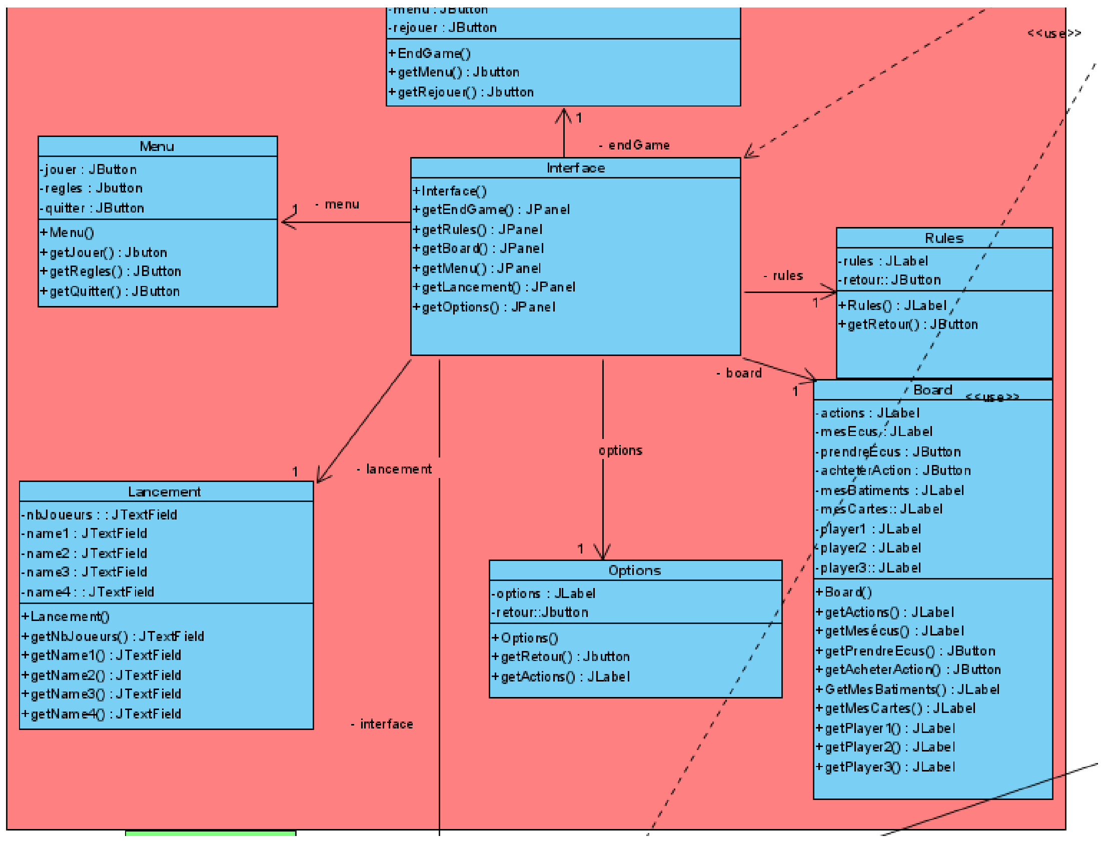
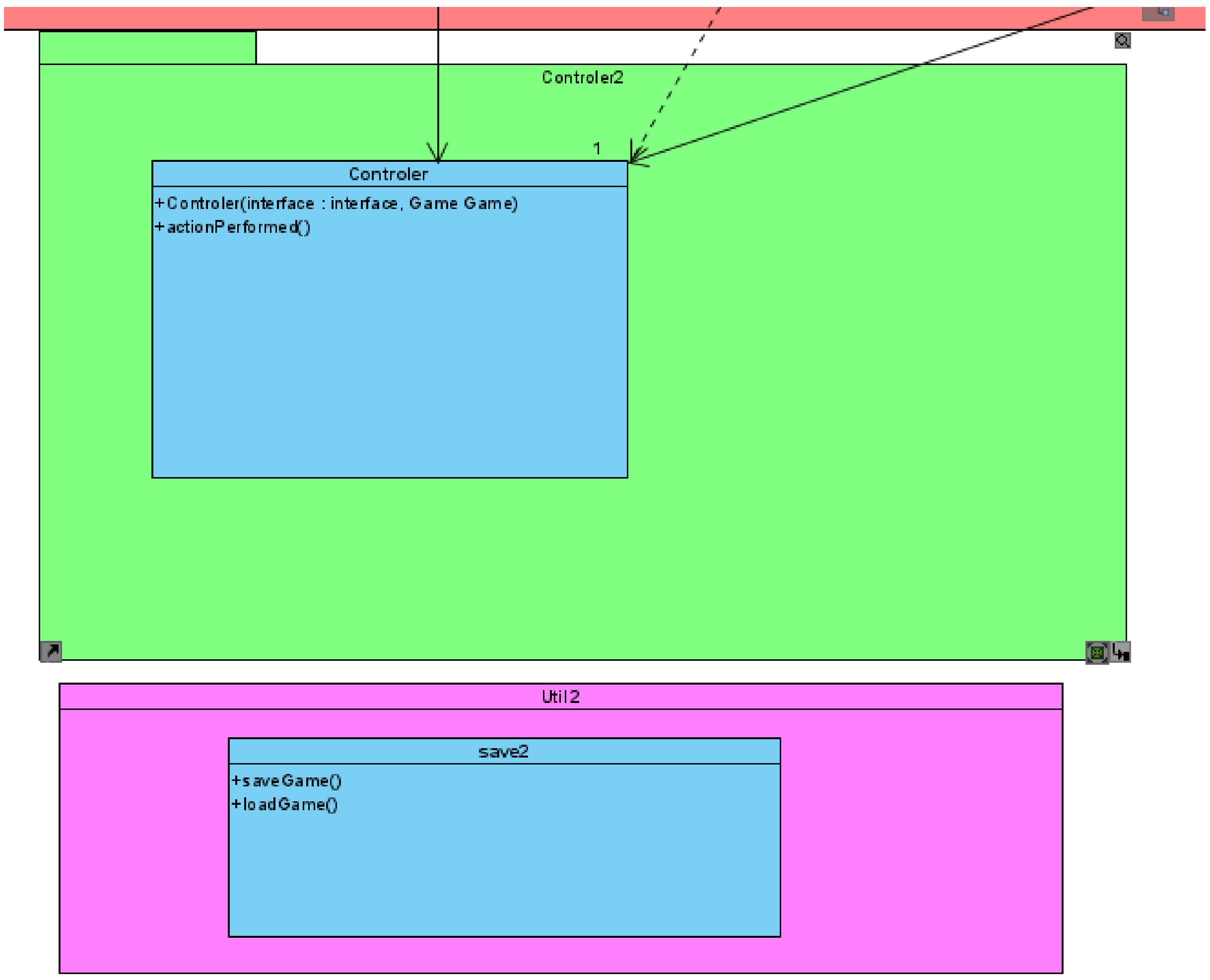


Diagramme de conception à retrouver en annexe (2)

Le diagramme étant petit, merci de m’adresser un mail à l’adresse suivante si vous voulez l’avoir en version Visual paradigme pour une meilleure lecture : yvanbocande@outlook.com


Game2Z
Game
-name1 : String
-name2 : String
-name3 : String
+game(name1 : String, name2 : String, name3 : String, name4 : String, fileNameString)
+createPlayers()
+startGame()
+endGame()
+setMusic(music : boolean)
+setSounds(sounds : boolean)
+initializeBoard()
+changeCurrentPlayer()
+getCurrentPlayer()
+getBoard()
+saveGame()
+initializePlayers()
+isFinish() : boolean
+getMode() : Mode
+getPlayer1() : player
+getPlayer2() : Player
+getPlayer3() : Player
+getPlayer4() : Player
+toString() : String
+getDeck() : Deck
GameConfig
+GameConfig(FileName : String)
+readConfig()
+printConfig()
+pickStartPlayer() : boolean
<<use>>
1
- config
<<enumeration>>
Mode
HA
HH
HHH
HHA
HAA
AAA
HHHH
AAAA
HHHA
AAAH
AAHH
1
- mode
player1
player4
player2
player3
current
1 - deck
- Game
Player
-numberactionLeft : int
-Ecus : int
-name : String
-number : int
-victoryPoints : int
+Player(name : String, number : int)
+play()
+getCartes() : cards
+addCarte(carte : Carte)
+removeCarte(carte : Carte)
+getName() : String
+openBuilding(num : int)
+recrutWorker(num : int)
+sendWorking(numWorker : int, numBuilding : int)
+takeEcus()
+buyWorker(num : int)
+finishAction()
+getMoney() : int
+getNumAction() : int
+setAction(num : int)
+getPoints() : int
+setPoints()
Deck
+Deck()
+initializeDeck()
+suffleDeck()
+getFirstCard() : Card
+showCard(number : int) : Card
+deleteCard(num : int)
cards
- workers
- buildings
- machines
*
*
Card
-gold : int
-stone : int
-knowledge : int
-wood : int
-getName : String
+Card(gold : int, stone : int, knowledge : int, wood : int, name String)
+abstract toString()
Worker
-cost : int
-isWorking : boolean
+Worker(gold : int, stone : int, knowledge : int, wood : int, name String, cost : int)
+getGold() : int
+getKnowledge() : int
+getStone() : int
+getWood() : int
+getCost() : int
+getName() : int
+setIsWorking(value : boolean)
HumanPlayer
-scan : Scanner
+HumanPlayer(name : String, number : int)
+play()
AutoPlayer
+AutoPlayer(name : String, number : int)
+play()
DifficultPlayer
+DifficultPlayer(name : String, number : int)
+play()
EasyPlayer
+EasyPlayer(name : String, number : int)
+play()
Machine
-stoneP : int
-woodP : int
-goldP : int
-knowledgeP : int
+Machine(gold : int, stone : int, knowledge : int, wood : int, name String, stoneP : int, woodP : int, goldP : int, knowledgeP : int)
+getGold() : int
+getKnowledge() : int
+getStone() : int
+getWood() : int
+getKnowledgeP() : int
+getStoneP() : int
+getWoodP() : int
+getGoldP() : int
+isFinish() : int
+getName() : int
Building
-victoryPoint : int
-finish : boolean
+Building(gold : int, stone : int, knowledge : int, wood : int, name String)
+getGold() : int
+getPointVictory() : int
+getwood() : int
+getStone() : int
+getKnowledge() : int
+isFinish() : boolean
+getName() : int
+finish()

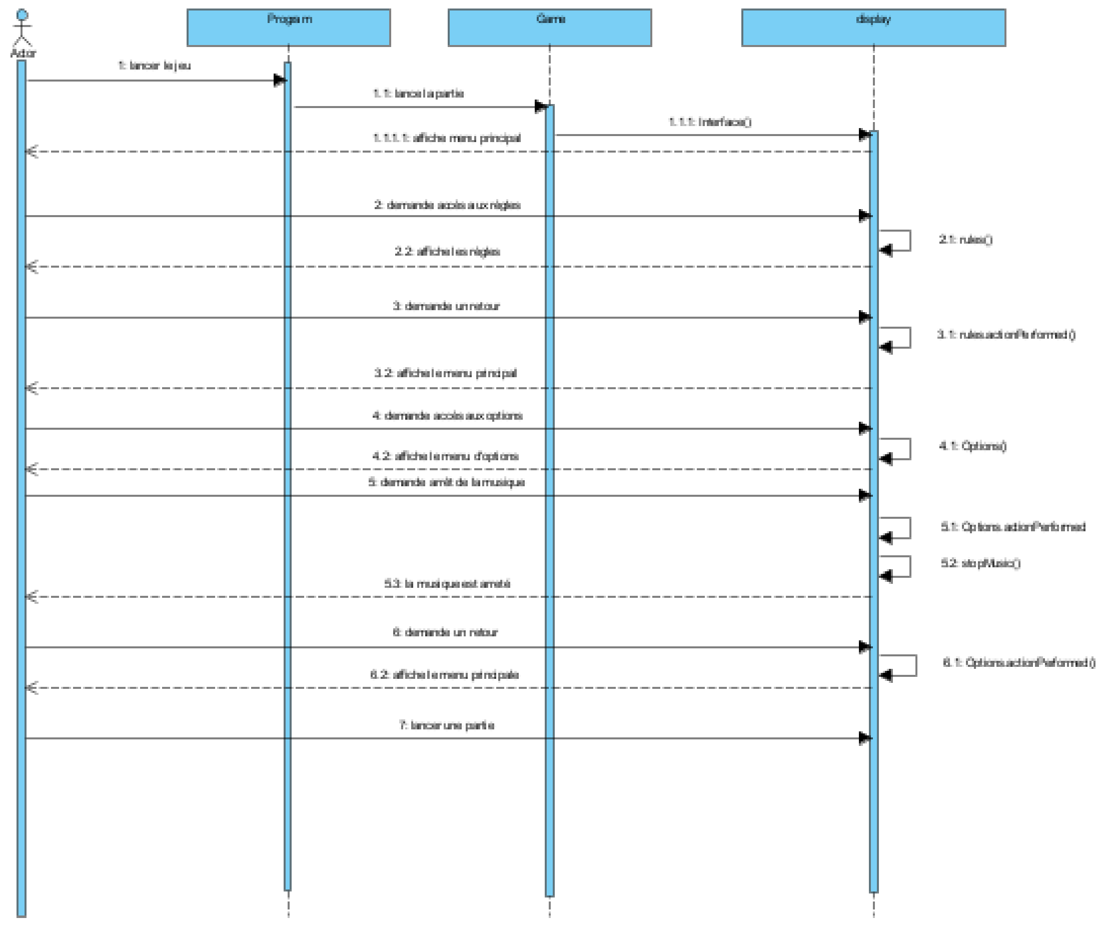

-menu : JButton
-rejouer : JButton
+EndGame()
+getMenu() : Jbutton
+getRejouer() : Jbutton
<<use>>
1
- endGame
Menu
-jouer : JButton
-regles : Jbutton
-quitter : JButton
+Menu()
+getJouer() : Jbutton
+getRegles() : JButton
+getQuitter() : JButton
1
- menu
Interface
+Interface()
+getEndGame() : JPanel
+getRules() : JPanel
+getBoard() : JPanel
+getMenu() : JPanel
+getLancement() : JPanel
+getOptions() : JPanel
Rules
-rules : JLabel
-retour:: JButton
+Rules() : JLabel
+getRetour() : JButton
- rules
1
- board
1
Board
<<use>>
-actions : JLabel
-mesEcus : JLabel
-prendreÉcus : JButton
-achteterAction : JButton
-mesBatiments : JLabel
-mesCartes:: JLabel
-player1 : JLabel
-player2 : JLabel
-player3:: JLabel
+Board()
+getActions() : JLabel
+getMesécus() : JLabel
+getPrendreEcus() : JButton
+getAcheterAction() : JButton
+GetMesBatiments() : JLabel
+getMesCartes() : JLabel
+getPlayer1() : JLabel
+getPlayer2() : JLabel
+getPlayer3() : JLabel
options
1
- lancement
1
Lancement
-nbJoueurs : : JTextField
-name1 : JTextField
-name2 : JTextField
-name3 : JTextField
-name4 : : JTextField
+Lancement()
+getNbJoueurs() : JTextField
+getName1() : JTextField
+getName2() : JTextField
+getName3() : JTextField
+getName4() : JTextField
Options
-options : JLabel
-retour:Jbutton
+Options()
+getRetour() : Jbutton
+getActions() : JLabel
- interface



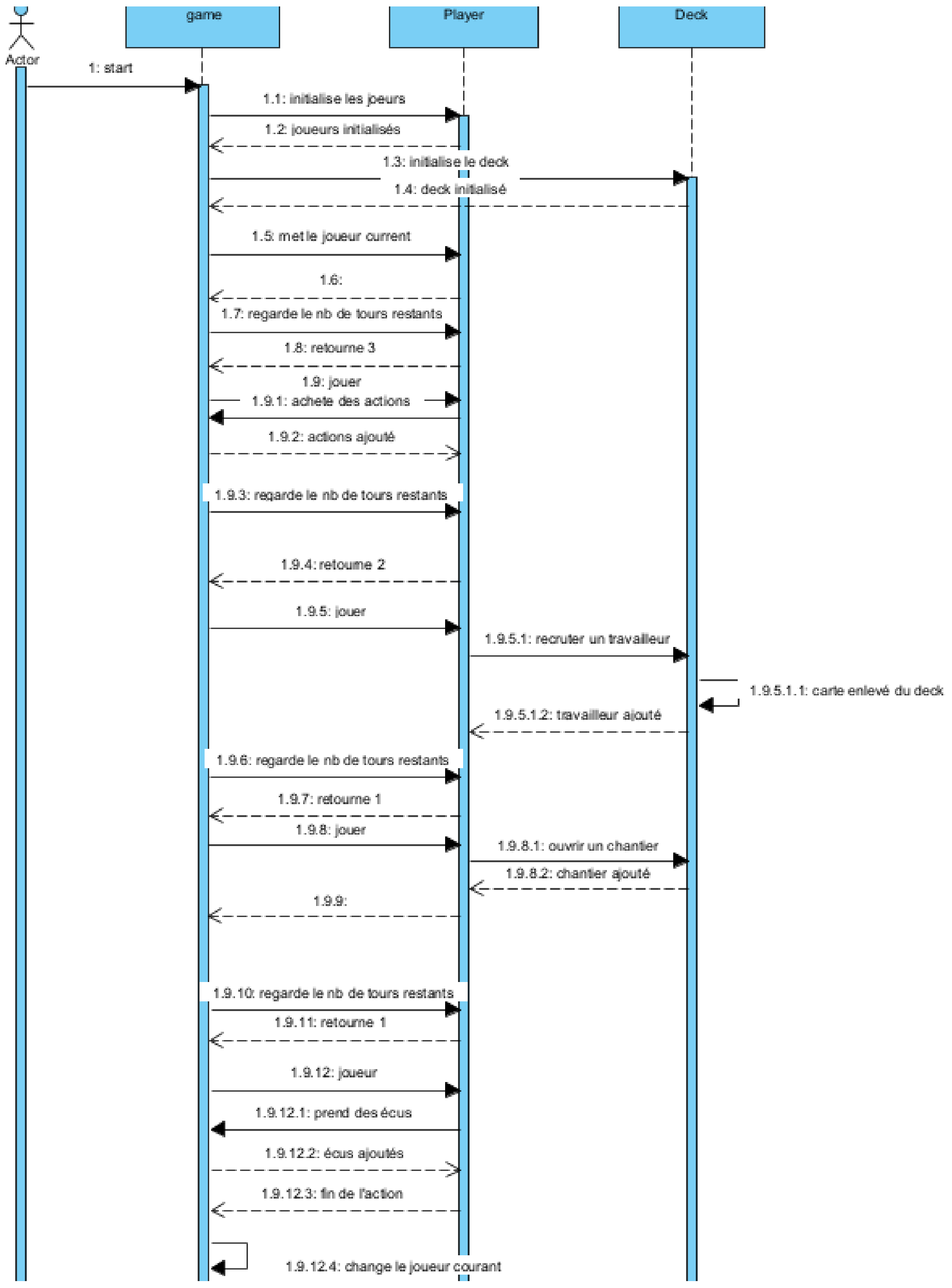
3) Diagramme de séquences

J'ai décider de produire deux diagrammes de séquence afin de différencier le fonctionnement général du logiciel du fonctionnement d'une partie et des choix des joueurs.

Ce premier diagramme les interactions entre l'utilisateur avec le menu comportant la partie graphique. On y retrouve les différentes parties du menu (options, règles, etc…).


Actor
Program
Game
display
1: lancer le jeu
1.1: lance la partie
1.1.1: Interface()
1.1.1.1: affiche menu principal
2: demande accès aux règles
2.1: rules()
2.2: affiche les règles
3: demande un retour
3.1: rules.actionPerformed()
3.2: affiche le menu principal
4: demande accès aux options
4.1: Options()
4.2: affiche le menu d'options
5: demande arrêt de la musique
5.1: Options.actionPerformed
5.2: stopMusic()
5.3: la musique est arrêté
6: demande un retour
6.1: Options.actionPerformed()
6.2: affiche le menu principale
7: lancer une partie

Le second diagramme ci-dessous décrit les interactions entre les joueurs et le jeu en cours de partie :


Actor
game
Player
Deck
1: start
1.1: initialise les joeurs
1.2: joueurs initialisés
1.3: initialise le deck
1.4: deck initialisé
1.5: met le joueur current
1.6:
1.7: regarde le nb de tours restants
1.8: retourne 3
1.9: jouer
1.9.1: achete des actions
1.9.2: actions ajouté
1.9.3: regarde le nb de tours restants
1.9.4: retourne 2
1.9.5: jouer
1.9.5.1: recruter un travailleur
1.9.5.1.1: carte enlevé du deck
1.9.5.1.2: travailleur ajouté
1.9.6: regarde le nb de tours restants
1.9.7: retourne 1
1.9.8: jouer
1.9.8.1: ouvrir un chantier
1.9.8.2: chantier ajouté
1.9.9:
1.9.10: regarde le nb de tours restants
1.9.11: retourne 1
1.9.12: joueur
1.9.12.1: prend des écus
1.9.12.2: écus ajoutés
1.9.12.3: fin de l'action
1.9.12.4: change le joueur courant

4) Spécification des formats de fichiers

Afin de sauvegarder des données d'une partie en cours nous allons utiliser la sérialisation d'objet.
L'extension du fichier une extension spécialement conçu pour ce jeu et ainsi pouvoir retrouver et reconnaît nos dossiers de sauvegarde. Nous avons donc choisi d'utilisateur l'extension .bat. (Exemple : save.bat).

Le fichier ne sera pas lisible à l'œil cependant il sera ouvert par le programme qui lira les objets nécessaires à reprendre une partie en cours. Seulement des objets du package model ont besoin d'êtres sauvegardés. Ce fichier sera écrit lorsqu'une partie est quitter brusquement ou lorsque que l'utilisateur cliquera sur le bouton correspondant à une sauvegarde de la partie.

5) Fichier build.xml pour ANT

Pour pouvoir automatiser les différentes taches lors de la phase de codage (compilation, génération de la javadoc, création du jar..) nous utiliserons un fichier xml qui sera utilisé avec ANT. Ce logiciel est spécialisé dans l'automation de tâches répétitives liées au développement logiciel.

```xml
<project name="Les batisseurs" default="jar" basedir=".">
    <description>
 Fichier build.xml (ANT) pour le projet de programmation
 </description>
    <!--Les dossiers-->
    <property name="build" location="build" />
    <property name="src" location="src" />
    <property name="class" location="${build}/class" />
    <property name="jar" location="${build}/jar" />
    <property name="javadoc" location="${build}/javadoc" />
    <property name="report" location="${build}/report" />
    <property name="main-class" value="src/Lanceur.java" />

    <target name="init">
        <!--Timestamp to now when the compilation append-->
        <tstamp />
        <!-- Directory used to store the compiled files-->
        <mkdir dir="${build}" />
    </target>
    <target name="all" depends="compiler, jar,javadoc" description="Execute compiler, jar et
javadoc"></target>
```

```
    <target name="compiler" depends="init" description="Compiler les classes">
        <!-- Créer un répertoire class-->
        <mkdir dir="${class}" />
        <!-- Compile le code java de ${src} dans ${class} -->
        <javac srcdir="${src}" destdir="${class}" includeantruntime="false" />
    </target>

    <target name="javadoc" depends="init" description="Création des fichier javadoc">
        <!--Créer le répertoire javadoc -->
        <mkdir dir="${javadoc}" />
        <!--Créer la javadoc -->
        <javadoc destdir="${javadoc}" author="true" version="true">
            <fileset dir="${src}" defaultexcludes="yes">
                <include name="*/*.java" />
                <exclude name="*/*Test.java" />
            </fileset>
        </javadoc>
    </target>

    <target name="jar" depends="compiler" description="Créer le jar">
        <!-- Créer le dossier jar -->
        <mkdir dir="${jar}" />
        <!-- Créer le jar -->
        <jar jarfile="${jar}/${DSTAMP}_Zen-L'initié.jar" basedir="${class}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}" />
            </manifest>
            32
        </jar>
    </target>
    <target name="junit" description="Execute tous les tests">
        <!-- Créer le dossier class et compile -->
        <mkdir dir="${class}" />
        <javac srcdir="${src}" destdir="${class}" includeantruntime="false" />
        <!-- Créer le dossier report -->
        <mkdir dir="${report}" />
        <junit printsummary="yes" haltonfailure="no">
            <formatter type="xml" />
            <classpath>
                <pathelement location="${class}" />
                <pathelement location="./lib/junit-4.12.jar" />
                <pathelement location="./lib/hamcrest-core-1.3.jar" />
            </classpath>
            <batchtest todir="${report}">
```

```
                <fileset dir="${src}">
                    <include name="test/*Test.java" />
                </fileset>
            </batchtest>
        </junit>
        <junitreport todir="${report}">
            <fileset dir="${report}">
                <include name="TEST-*.xml" />
            </fileset>
            <report format="frames" todir="${report}/html" />
        </junitreport>
    </target>
    <target name="clean" description="Suppresion des anciens fichiers">
        <!-- Supprime les dossiers ${javadoc} et ${jar} -->
        <delete dir="${javadoc}" />
        <delete dir="${jar}" />
    </target>
</project>
```

6) Squelette et documentation de la classe modèle

Class Game :

```
package Game;

/**
 * this class is used to start a game and to manage the players during a game
 * @author Yvan Bocandé
 */
public class Game {

    private Player player1;
    private Player player2;
    private Player player3;
    private Player player4;
    private Player curent;
    private GameConfig config;
    private String name1;
    private String name2;
    private String name3;
    private String name4;
    private Mode mode;
    private Deck deck;

    /**
     * this constructor instantiates an object of type Game and starts the gam
e with the configuration file. It can load a game in progress.
     * @param name1 a String wich is the name of the first player
     * @param name2 a String wich is the name of the second player
     * @param name3 a String wich is the name of the third player
     * @param name4 a String wich is the name of the fourth player
     * @param fileName a String wich is the name of the file to found
     */
    public void game(String name1, String name2, String name3, String name4, i
nt fileNameString) {
        // TODO - implement Game.game
    }

    /**
     * this method instantiates the players according to the number of players
 and their type
     */
    public void createPlayers() {
        // TODO - implement Game.createPlayers
    }

    /**
     * this method starts the game
     */
    public void startGame() {
        // TODO - implement Game.startGame
    }

    /**
     * this method is used to end the game and announce the winner
     */
    public void endGame() {
        // TODO - implement Game.endGame
    }

    /**
     * this method is used to set the music during the game
     * @param music if it's true: it put the music on
     */
    public void setMusic(boolean music) {
        // TODO - implement Game.setMusic
    }

    /**
     * this method is used to set all the differents sound during the game
     * @param sounds if it's true: it put the sounds on
     */
    public void setSounds(boolean sounds) {
        // TODO - implement Game.setSounds
    }

    /**
```

```
     * this method is used to initialize the board, it also initialize the dec
k of the player
     */
    public void initializeBoard() {
        // TODO - implement Game.initializeBoard
    }

    /**
     * change the player to play
     */
    public void changeCurrentPlayer() {
    }

    /**
     * return the attribute current
     * @return the Player wich is in the attribute current
     */
    public Player getCurrentPlayer() {
        // TODO - implement Game.getCurrentPlayer
        throw new UnsupportedOperationException();

    }

    /**
     * use the methods of the package util to save the game
     */
    public void saveGame() {
        // TODO - implement Game.saveGame
    }

    /**
     * this method verify if a player have win the game using a method of the
class Player
     * @return true if a player have win the game
    */
    public boolean isFinish() {
        // TODO - implement Game.isFinish
        throw new UnsupportedOperationException();
    }

    /**
     * return the value of the attribute Game
     * @return the Mode of the object Game
     */
    public Mode getMode() {
        return this.mode;
    }

    /**
```

```java
     * return the value of the attribute Player1
     * @return the Player1 of the object game
     */
    public Player getPlayer1() {
        return this.player1;
    }

    /**
     * return the value of the attribute Player2
     * @return the Player2 of the object game
     */
    public Player getPlayer2() {
        return this.player2;
    }

    /**
     * return the value of the attribute Player3
     * @return the Player3 of the object game
     */
    public Player getPlayer3() {
        return this.player3;
    }

    /**
     * return the value of the attribute Player4
     * @return the Player4 of the object game
     */
    public Player getPlayer4() {
        return this.player4;
    }

    /**
     * print the board of the game to play
     */
    public String toString() {
        // TODO - implement Game.toString
        throw new UnsupportedOperationException();
    }

    /**
     * return the deck of cards of the game
     * @return the attribute deck of type Deck of the object
     */
    public Deck getDeck() {
        return this.deck;
    }

}
```

Class GameConfig :

```
package Game;

/**
 * this method is used to create a configuration for a game using a file to co
nfigure
 */
public class GameConfig {

    /**
     * Read the file enter in parameter and exctract informations using readFi
le()
     * @param FileName
     */
    public GameConfig(String FileName) {
        // TODO - implement GameConfig.GameConfig
        throw new UnsupportedOperationException();
    }

    /**
     * use a scanner to exctract informations for the file of configuration
     */
    public void readConfig() {
        // TODO - implement GameConfig.readConfig
        throw new UnsupportedOperationException();
    }

    /**
     * print all the informations of the configuration
     */
    public void printConfig() {
        // TODO - implement GameConfig.printConfig
        throw new UnsupportedOperationException();
    }

    /**
    * pick the list of the number of player to play in random way
    * @return the list of player to play
    */
    public int[] pickStartPlayer() {
        // TODO - implement GameConfig.pickStartPlayer
        throw new UnsupportedOperationException();
    }

}
```

Enum mode :

```
package Game;

public enum Mode {
    HA,
    HH,
    HHH,
    HHA,
    HAA,
    AAA,
    HHHH,
    AAAA,
    HHHA,
    AAAH,
    AAHH
}
```

Class Player :

```
package Game;

import java.util.ArrayList;

public abstract class Player {

    private ArrayList<Machine> myMachines;
    private ArrayList<Bulding> myBuldings;
    private ArrayList<Worker> myWorkers;
    private int numberActionLeft;
    private int ecus;
    private String name;
    private int number;
    private int victoryPoints;

    /**
     * initialize all the attribute of the player. initialize all the arrayLis
t with nothing inside.
     * @param name a String wich is the name of the player
     * @param number an int wich is the number of the player
     */
    public Player(String name, int number) {
        // TODO - implement Player.Player
        throw new UnsupportedOperationException();
    }

    /**
```

```
     * ask to the player the action he want to do and do the action with an ot
her method
     */
    public abstract void play();

    /**
     * this method add a card to the Arraylist choice in parameters (myMachine
s,myWorkers,myBuldings)
     * @param carte the card to add to the ArrayList
     */
    public void addCarte(Card carte, String deck){

    }

    /**
     * this method remove a card to the Arraylist choice in parameters (myMach
ines,myWorkers,myBuldings)
     * @param carte the card to remove of the ArrayList
     */
    public void removeCarte(Card carte, String deck){

    }

    /**
     * return the name of the player
     * @return a String wich is the attribute name
     */
    public String getName() {
        return this.name;
    }

    /**
     * this method is used ti start a bulding in taking a new card (type buldi
ng) on the deck
     * @param num the num of the card to take on the dek (beetween 1 and 5)
     */
    public void openBulding(int num) {
        // TODO - implement Player.openBulding
        throw new UnsupportedOperationException();
    }

    /**
     * this method is used to add a new worker to our myWorkers. The cost of t
he worker will be remove of the ecus of the player
     * @param num the num of the card to take on the dek (beetween 1 and 5)
     */
    public void recrutWorker(int num) {
        // TODO - implement Player.recrutWorker
        throw new UnsupportedOperationException();
    }

    /**
     * This method is used to send a worker to a bulding
     * @param num the num of the worker to send in your hand
     */
    public void sendWorking(int num) {
        // TODO - implement Player.sendWorking
        throw new UnsupportedOperationException();
    }

    /**
    * add 5 ecus of the attribute ecus
    */
    public void takeEcus() {
        // TODO - implement Player.takeEcus
        throw new UnsupportedOperationException();
    }

    /**
     * this method finish a bulding and give the points of victory of it
     */
    public void finishAction() {
        // TODO - implement Player.finishAction
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of ecus of the player
     * @return an int wicch is the attributes ecus
     */
    public int getMoney() {
        // TODO - implement Player.getMoney
        throw new UnsupportedOperationException();
    }

    /**
     * give the number of actions lefts of the player
     * @return an int wich corresponds to the attribute numberActionsLeft
     */
    public int getNumAction() {
        // TODO - implement Player.getNumAction
        throw new UnsupportedOperationException();
    }

    /**
     * set the number of the numberActionLeft
     * @param num the new int of action left
     */
    public void setAction(int num) {
        // TODO - implement Player.setAction
        throw new UnsupportedOperationException();
    }

    /**
     * give the number of points of victory of the player
     * @return an int wich corresponds to the attribute victoryPoints
     */
    public int getPoints() {
        // TODO - implement Player.getPoints
        throw new UnsupportedOperationException();
    }

    /**
     * set the number of the victoryPoints
     * @param num the number to add to the attribute pointsVictory
     */
    public void setPoints(int num) {
        // TODO - implement Player.setAction
        throw new UnsupportedOperationException();
    }

}
```

Class HumanPlayer :

```
package Game;

import java.util.Scanner;

/**
 * this class is used for a player to play. It ask what he want to do and how
he want to do.
 */
public class HumanPlayer extends Player {

    private Scanner scan;

    /**
     * the constructor initializes the auto player using the player constructo
r.
     * @param name a String wich is the name of the player
     * @param number an int wich is the number of player.
     */
    public HumanPlayer(String name, int number) {
        super(name, number);
        // TODO - implement HumanPlayer.HumanPlayer
        throw new UnsupportedOperationException();
    }

    /**
     * this methods makes play the human player using the scanner.
     */
    public void play() {
        // TODO - implement HumanPlayer.play
        throw new UnsupportedOperationException();
    }

}
```

Class AutoPlayer :

```
package Game;
/**
 * this class is a subclass of the player class. It defines a player who plays
 alone in a random way.
 * @author Yvan Bocande
 */
public abstract class AutoPlayer extends Player {

    /**
     * the constructor initializes the auto player using the player constructo
r.
     * @param name a String wich is the name of the player
     * @param number an int wich is the number of player.
     */
    public AutoPlayer(String name, int number) {
        super(name,number);
        // TODO - implement AutoPlayer.AutoPlayer
        throw new UnsupportedOperationException();
    }

    /**
     * this abstract class will be redefined according to the difficulty of th
e player to play
     */
    public abstract  void play();
}
```

Class EasyPlayer :

```
package Game;
/**
 * this class redifine the method play to play in a random way
 */
public class EasyPlayer extends AutoPlayer {

    /**
     * the constructor initializes the player using the player constructor.
     * @param name a String wich is the name of the player
     * @param number an int wich is the number of player.
     */
    public EasyPlayer(String name, int number) {
        super(name, number);
        throw new UnsupportedOperationException();
    }

    /**
     * this class will play in a random way
     */
    public void play() {
        // TODO - implement EasyPlayer.play
        throw new UnsupportedOperationException();
    }

}
```

Class DifficultPlayer :

```
package Game;

public class DifficultPlayer extends AutoPlayer {

    /**
     * the constructor initializes the player using the player constructor.
     * @param name a String wich is the name of the player
     * @param number an int wich is the number of player.
     */
    public DifficultPlayer(String name, int number) {
        super(name, number);
        throw new UnsupportedOperationException();
    }

    /**
     * this method will play automatically in trying to win against the other
player
     */
    public void play() {
        // TODO - implement DifficultPlayer.play
        throw new UnsupportedOperationException();
    }

}
```

<u>Class Deck :</u>

```
package Game;

import java.util.ArrayList;

public class Deck {

    private ArrayList<Card> cards;

    /**
     * initialize the attribute cards using the methods initializeDeck and suf
fleDeck
     */
    public Deck() {
        // TODO - implement Deck.Deck
        throw new UnsupportedOperationException();
    }

    /**
     * create all the cards using the constructor of card of the deck and put
them in the attribute card
     */
    public void initializeDeck() {
        // TODO - implement Deck.initializeDeck
        throw new UnsupportedOperationException();
    }

    /**
     * change the order of the cards in the arrayList in attributes
     */
    public void suffleDeck() {
        // TODO - implement Deck.suffleDeck
        throw new UnsupportedOperationException();
    }

    /**
     * return the first card of the deck
     * @return the cart wich is in the first postion of the arraylist
    */
    public Card getFisrtCard() {
        // TODO - implement Deck.getFisrtCard
        throw new UnsupportedOperationException();
    }

    /**
     * show the card wich is at the number passed in parameters in the ArrayLi
st
     * @param number the number of the card to return
     * @return a Card wich is at the number in the arraylist
     */
    public Card showCard(int number) {
        // TODO - implement Deck.showCard
        throw new UnsupportedOperationException();
    }

    /**
     * delete the card wich is at the number passed in parameters in the Array
List
     * @param number the number of the card to delete
     */
    public void deleteCard(int num){

    }

}
```

Class Card :

```
package Game;

/**
 * this abstract class is used to define cards with their attributes. Other at
tributes will be added depending on the type of player.
 */
public abstract class Card {

    private int gold;
    private int stone;
    private int knowledge;
    private int wood;
    private String name;

    /**
     * the constructor initializes the card with the attributes.
     *  For machines and workers this corresponds to the materials produced an
d for buldings it corresponds to the necessary materials.
     * @param gold an int wich is the number of gold on the card
     * @param stone an int wich is the number of stone on the card
     * @param knowledge an int wich is the number of knowledge on the card
     * @param wood an int wich is the number of wood on the card
     * @param name a String wich is the name on the card
     */
    public Card(int gold,int stone,int knowledge, int wood, String name) {
        // TODO - implement Card.Card
        throw new UnsupportedOperationException();
    }

    public abstract String toString();
}
```

Class machine :

```
package Game;

/**
 * this class represent a Machine with its skills and cost to build
 */
public class Machine extends Card {

    private int stoneP;
    private int woodP;
    private int goldP;
    private int knowledgeP;

    /**
     * this constructor initialize all the attibute of the Machine object
     * @param gold an int wich is the number of gold on the card
     * @param stone an int wich is the number of stone on the card
     * @param knowledge an int wich is the number of knowledge on the card
     * @param wood an int wich is the number of wood on the card
     * @param name a String wich is the name on the card
     * @param goldP an int wich is the gold produce vby the card
     * @param stoneP an int wich is the stone produce vby the card
     * @param knowledgeP an int wich is the knowledge produce by the card
     * @param woodP an int wich is the wood produce by the card
     */
    public Machine(int gold,int stone,int knowledge, int wood, String name, in
t stoneP, int woodP, int knowledgeP, int goldP) {
        super(gold, stone, knowledge, wood, name);
        throw new UnsupportedOperationException();
    }



    /**
     * return the number of gold of the card
     * @return an int wich is the attribute gold
     */
    public int getGold() {
        // TODO - implement Bulding.getGold
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of wood of the card
     * @return an int wich is the attribute wood
     */
    public int getwood() {
        // TODO - implement Bulding.getwood
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of stone of the card
     * @return an int wich is the attribute stone
     */
    public int getStone() {
        // TODO - implement Bulding.getStone
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of knowledge of the card
     * @return an int wich is the attribute knowledge
     */
    public int getKnowledge() {
        // TODO - implement Bulding.getKnowledge
        throw new UnsupportedOperationException();
    }


    /**
     * return the name of the card
     * @return a String wich correspond to the attribute name
     */
    public String getName() {
        // TODO - implement Bulding.getName
        throw new UnsupportedOperationException();
    }
```

```java
    /**
     * print the card with all the informations and attributes
     */
    public String toString(){
        throw new UnsupportedOperationException();
    }

        /**
     * return the number of gold produce of the card
     * @return an int wich is the attribute gold
     */
    public int getGoldP() {
        // TODO - implement Bulding.getGold
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of wood produce of the card
     * @return an int wich is the attribute wood
     */
    public int getwoodP() {
        // TODO - implement Bulding.getwood
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of stone produce of the card
     * @return an int wich is the attribute stone
     */
    public int getStoneP() {
        // TODO - implement Bulding.getStone
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of knowledge produce of the card
     * @return an int wich is the attribute knowledge
     */
    public int getKnowledgeP() {
        // TODO - implement Bulding.getKnowledge
        throw new UnsupportedOperationException();
    }


}
```

Classe Worker

```java
package Game;

/**
 * this class represent a card of a worker with the competences and cost
 */
public class Worker extends Card {

    private int cost;
    private boolean isWorking;

    /**
     * this constructor initialize all the attibute of the Worker object
     * @param gold an int wich is the number of gold on the card
     * @param stone an int wich is the number of stone on the card
     * @param knowledge an int wich is the number of knowledge on the card
     * @param wood an int wich is the number of wood on the card
     * @param name a String wich is the name on the card
     * @param cost the cost to send the worker work
     */
    public Worker(int gold,int stone,int knowledge, int wood, String name, int
 cost) {
        super(gold, stone, knowledge, wood, name);
        // TODO - implement Worker.Worker
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of gold of the card
     * @return an int wich is the attribute gold
     */
    public int getGold() {
        // TODO - implement Bulding.getGold
        throw new UnsupportedOperationException();
    }

    /**
     * return the cost of the card
     * @return an int wich is the attribute cost
     */
    public int getCOst() {
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of wood of the card
     * @return an int wich is the attribute wood
     */
    public int getwood() {
        // TODO - implement Bulding.getwood
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of stone of the card
     * @return an int wich is the attribute stone
     */
    public int getStone() {
        // TODO - implement Bulding.getStone
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of knowledge of the card
     * @return an int wich is the attribute knowledge
     */
    public int getKnowledge() {
        // TODO - implement Bulding.getKnowledge
        throw new UnsupportedOperationException();
    }


    /**
     * return the name of the card
     * @return a String wich correspond to the attribute name
     */
    public String getName() {
        // TODO - implement Bulding.getName
        throw new UnsupportedOperationException();
    }

    /**
     * return true if the worker is working
     * @return the attribute isWorking
     */
    public boolean isWorking(){
        throw new UnsupportedOperationException();
    }

    /**
     * change the value of the attribute isWorking with the param
     * @param value the new value of the attribute isWorking
     */
    public boolean setIsWorking(boolean value){
        throw new UnsupportedOperationException();
    }

    /**
     * print the card with all the informations and attributes
     */
    public String toString(){
        throw new UnsupportedOperationException();
    }
}
```

Class bulding :

```
package Game;

public class Bulding extends Card {

    private int victoryPoint;
    private boolean finish;

    /**
     * this constructor initialize all the attibute of the Bulding object
     * @param gold an int wich is the number of gold on the card
     * @param stone an int wich is the number of stone on the card
     * @param knowledge an int wich is the number of knowledge on the card
     * @param wood an int wich is the number of wood on the card
     * @param name a String wich is the name on the card
     * @param victoryPoint the number of point of victory
     */
    public Bulding(int gold,int stone,int knowledge, int wood, String name, in
t victoryPoint) {
        super(gold, stone, knowledge, wood, name);
        // TODO - implement Bulding.Bulding
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of gold of the card
     * @return an int wich is the attribute gold
     */
    public int getGold() {
        // TODO - implement Bulding.getGold
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of point of victory of the card
     * @return an int wich is the attribute victoryPoint
     */
    public int getPointVictory() {
        // TODO - implement Bulding.getPointVictory
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of wood of the card
     * @return an int wich is the attribute wood
     */
    public int getwood() {
        // TODO - implement Bulding.getwood
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of stone of the card
     * @return an int wich is the attribute stone
     */
    public int getStone() {
        // TODO - implement Bulding.getStone
        throw new UnsupportedOperationException();
    }

    /**
     * return the number of knowledge of the card
     * @return an int wich is the attribute knowledge
     */
    public int getKnowledge() {
        // TODO - implement Bulding.getKnowledge
        throw new UnsupportedOperationException();
    }

    /**
     * return true if the bulding is finish to build (enough materials)
     * @return a boolean wich is the attribute finish
     */
    public boolean isFinish() {
        // TODO - implement Bulding.isFinish
        throw new UnsupportedOperationException();
    }

    /**
     * return the name of the card
     * @return a String wich correspond to the attribute name
     */
    public String getName() {
        // TODO - implement Bulding.getName
        throw new UnsupportedOperationException();
    }

    /**
     * print the card with all the informations and attributes
     */
    public String toString(){
        throw new UnsupportedOperationException();
    }

}
```

7) Test unitaires avec JUnit

Test de game :

```
import Game.*;

import org.junit.After;
import org.junit.Before;
import java.util.ArrayList;
import static org.junit.Assert.*;

/**
* La classe GameTest
* @author Yvan
* @version 1.0
*/
public class GameTest {
    private Player player1;
    private Player player2;
    private Player player3;
    private Player player4;
    private Player curent;
    private GameConfig config;
    private String name1;
    private String name2;
    private String name3;
    private String name4;
    private Mode mode;
    private Deck deck;

    @test
    public final void testGame(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
```

```java
        assertNotNull("Instance non creee", game);
    }

    @Test
    public final void testCreatePlayers(){
        // TODO - Modifier le String en fonction
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        assertSame("yvan",game.getPlyaer1.getName());
        assertSame("arthur",game.getPlyaer2.getName());
        assertSame("matteo",game.getPlyaer3.getName());
        assertSame("eva",game.getPlayer4.getName());
    }

    @Test
    public final void testChangeCurrent(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        game.changeCurrentPlayer();
        assertSame(game.getPlayer2(), Game.getCurrent());
    }

    @Test
    public final void testSaveGame(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        assertSame("fileName", game.save());
    }

    @after
    public final void testIsFinish(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        assertSame(false, game.isFinish());
    }

    @Test
    public final void testToString(){
        // TODO - Modifier le toString en fonction
        Game game = new Game("yvan","arthur", HH, table);
        assertEquals("à définir", game.toString());
    }

    @Test
    public final void testGetPlayer1(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        assertEquals(this.player1, game.GetPlayer1());
    }

    @Test
    public final void testGetPlayer2(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        assertEquals(this.player1, game.GetPlayer2());
    }

    @Test
    public final void testGetPlayer3(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        assertEquals(this.player1, game.GetPlayer3());
    }

    @Test
    public final void testGetPlayer4(){
        Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
        assertEquals(this.player1, game.GetPlayer4());
    }

        @Test
        public final void testGetDeck(){
            Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
            assertEquals(this.deck, game.GetDeck());
        }

        @Test
        public final void testGetMode(){
            Game game = new Game("yvan","arthur","mattéo","eva", "../data/test");
            assertEquals(this.mode, game.GetMode());
        }

    }
```

Test de Deck :

```
package Game;

import org.junit.After;
import org.junit.Before;
import java.util.ArrayList;
import static org.junit.Assert.*;

/**
* La classe Game DECK
* @author Yvan
* @version 1.0
*/
public class DeckTest{
    private ArrayList<Card> cards;

    @test
    public final void testDeck(){
        Deck deck = new Deck());
        assertNotNull("Instance non creee", deck);
    }

    @test
    public final void testInitializeDeck(){
        Deck deck = new Deck());
        assertNotNull("ArrayList non créé", this.cards);
    }

    @test
    public final void GetFisrtCard(){
        Game game = new Deck());
        assertEquals("à définir", this.cards.get(0));
    }

    @test
    public final void deleteCard(0){
        Game game = new Deck());
        assertEquals("à définir", this.cards.get(0));
    }

    @test
    public final void ShowCard(0){
        Game game = new Deck());
        assertEquals("à définir", this.cards.get(0));
    }
}
```

Test de Worker :

```
package Game;

import org.junit.After;
import org.junit.Before;
import java.util.ArrayList;
import static org.junit.Assert.*;
```

```
/**
* La classe Worker
* @author Yvan
* @version 1.0
*/
public class WorkerTest{


    @test
    public final void testWorker(){
        Worker worker = new Worker(1,1,1,1,"test",1);
        assertNotNull("Instance non creee", worker);
    }

    @Test
    public final void testGetGold(){
        Worker worker = new Worker(1,2,3,4,"test",5);
        assertEquals(1, worker.getGold());
    }

    @Test
    public final void testGetCost(){
        Worker worker = new Worker(1,2,3,4,"test",5);
        assertEquals(5, worker.getCost());
    }

    @Test
    public final void testGetStone(){
        Worker worker = new Worker(1,2,3,4,"test",5);
        assertEquals(2, worker.getStone());
    }

    @Test
    public final void testGetKnowledge(){
        Worker worker = new Worker(1,2,3,4,"test",5);
        assertEquals(3, worker.getKnowledge());
    }

    @Test
    public final void testGetWood(){
        Worker worker = new Worker(1,2,3,4,"test",5);
        assertEquals(4, worker.getWood());
    }

    @Test
    public final void testGetName(){
        Worker worker = new Worker(1,2,3,4,"test",5);
        assertEquals("test", worker.getName());
    }
```

```java
    @Test
    public final void testWorking(){
        Worker worker = new Worker(1,2,3,4,"test",5);
        assertEquals(false, worker.isWorking());
        worker.setIsWorking(true);
        assertEquals(true, worker.isWorking());
    }

}
```

Test de Bulding :

```java
package Game;

import org.junit.After;
import org.junit.Before;
import java.util.ArrayList;
import static org.junit.Assert.*;

/**
* La classe bulding
* @author Yvan
* @version 1.0
*/
public class BuldingTest{


    @test
    public final void testBulding(){
        Bulding bulding = new Bulding(1,1,1,1,"test",1);
        assertNotNull("Instance non creee", bulding);
    }

    @Test
    public final void testGetGold(){
        Bulding bulding = new Bulding(1,2,3,4,"test",5);
        assertEquals(1, bulding.getGold());
    }

    @Test
    public final void testGetPointVictory(){
        Bulding bulding = new Bulding(1,2,3,4,"test",5);
        assertEquals(5, bulding.getPointVictory());
    }

    @Test
    public final void testGetStone(){
        Bulding bulding = new Bulding(1,2,3,4,"test",5);
        assertEquals(2, bulding.getStone());
    }

    @Test
    public final void testGetKnowledge(){
        Bulding bulding = new Bulding(1,2,3,4,"test",5);
        assertEquals(3, bulding.getKnowledge());
    }

    @Test
    public final void testGetWood(){
        Bulding bulding = new Bulding(1,2,3,4,"test",5);
        assertEquals(4, bulding.getWood());
    }

    @Test
    public final void testGetName(){
        Bulding bulding = new Bulding(1,2,3,4,"test",5);
        assertEquals("test", bulding.getName());
    }

    @Test
    public final void testFinish){
        Bulding bulding = new Bulding(1,2,3,4,"test",5);
        assertEquals(false, bulding.isFinish());
    }

}
```

<u>Test de Machine :</u>

```
package Game;

import org.junit.After;
import org.junit.Before;
import java.util.ArrayList;
import static org.junit.Assert.*;

/**
* La classe machine
* @author Yvan
* @version 1.0
*/
public class MachineTest{
```

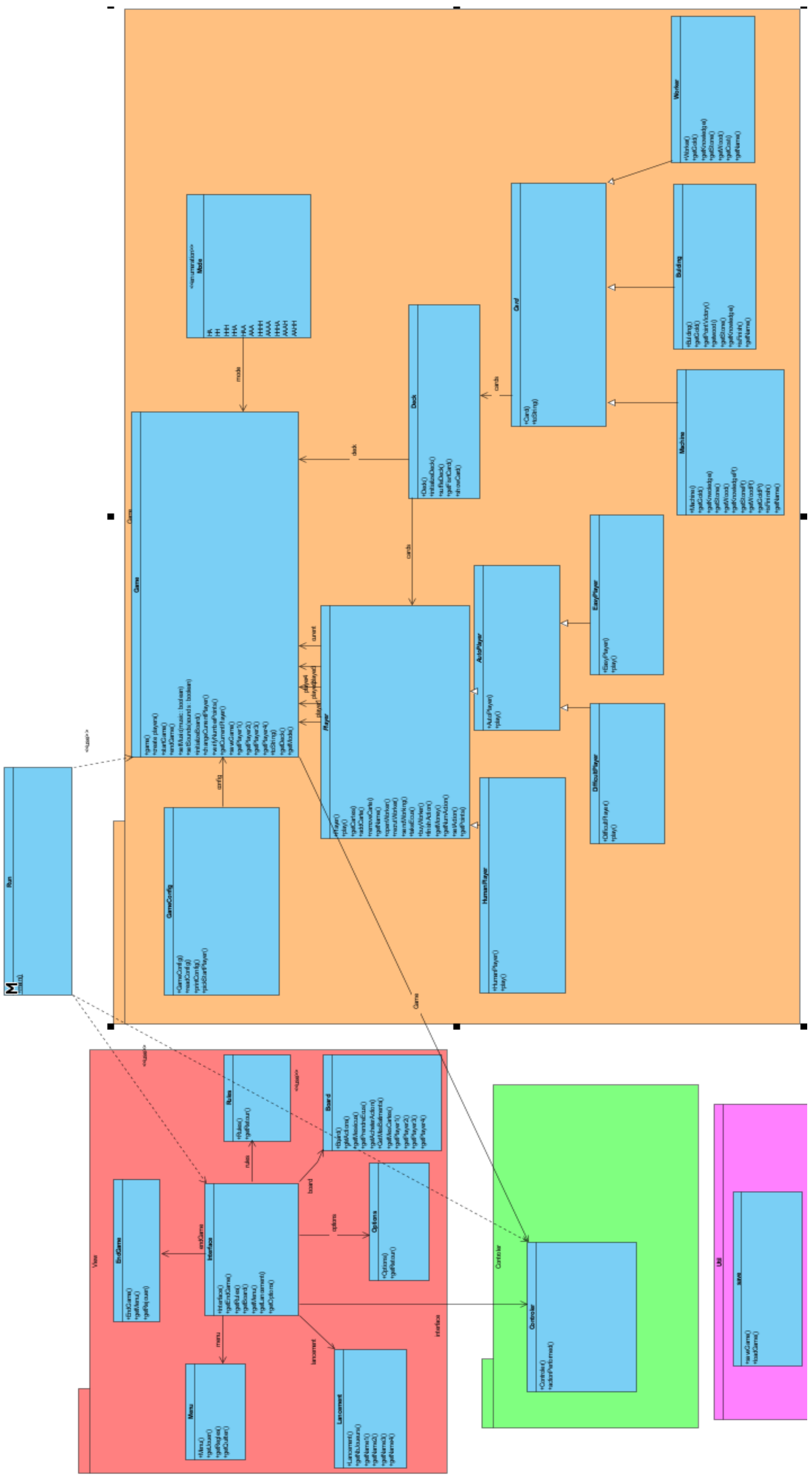
```java
    @test
    public final void testMachine(){
        Machine machine = new Machine(1,1,1,1,"test",1,1,1,1);
        assertNotNull("Instance non creee", machine);
    }

    @Test
    public final void testGetGold(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(1, machine.getGold());
    }

    @Test
    public final void testGetStone(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(2, machine.getStone());
    }

    @Test
    public final void testGetKnowledge(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(3, machine.getKnowledge());
    }

    @Test
    public final void testGetWood(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(4, machine.getWood());
    }

    @Test
    public final void testGetName(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals("test", machine.getName());
    }

    @Test
    public final void testGetGoldP(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(5, machine.getGoldP());
    }

    @Test
    public final void testGetStoneP(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(6, machine.getStoneP());
    }

    @Test
    public final void testGetKnowledgeP(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(7, machine.getKnowledgeP());
    }

    @Test
    public final void testGetWoodP(){
        Machine machine = new Machine(1,2,3,4,"test",5,6,7,8);
        assertEquals(8, machine.getWoodP());
    }
}
```

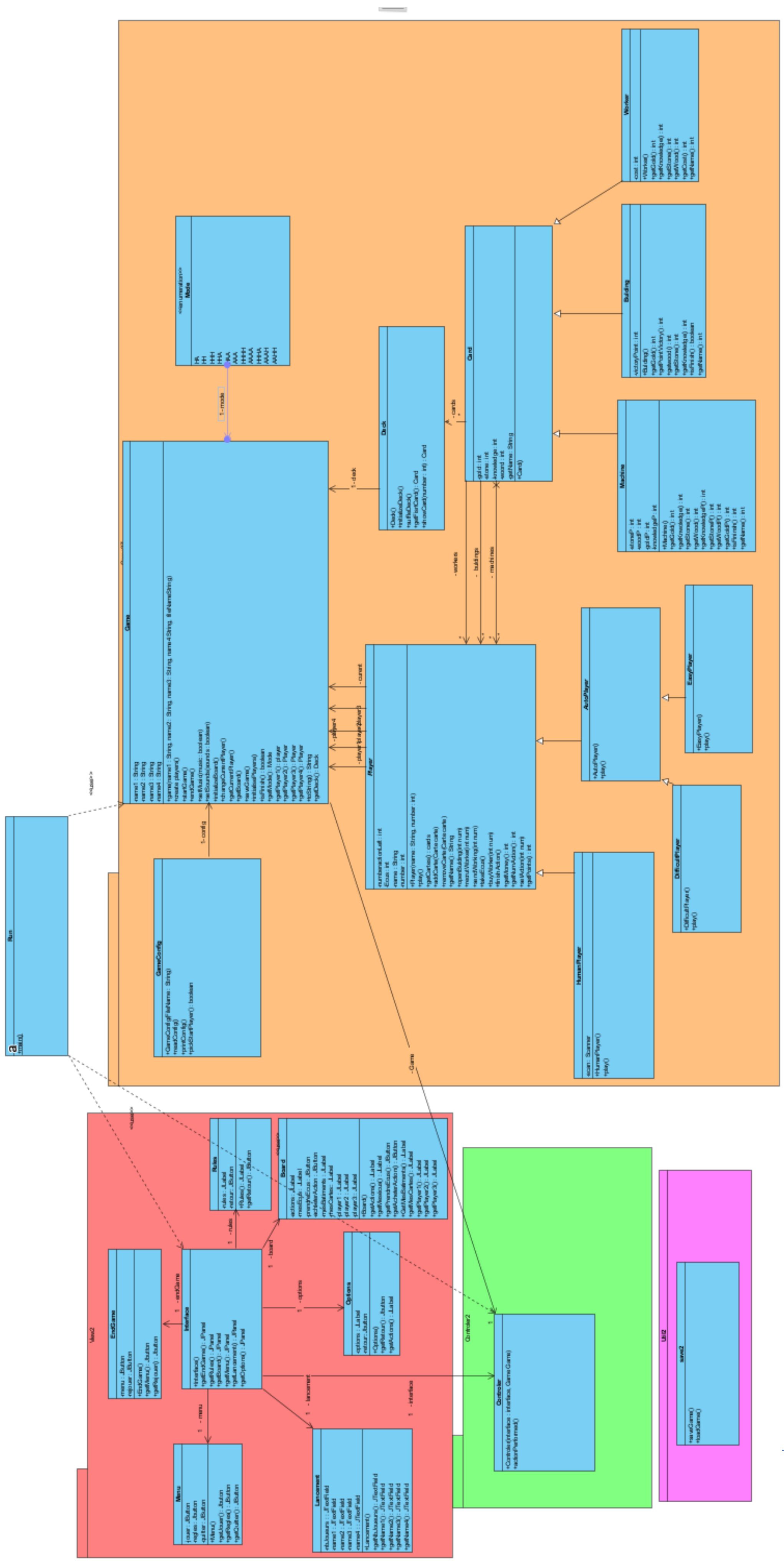
Test de HumanPlayer :

```
package Game;

import org.junit.After;
import org.junit.Before;
import java.util.ArrayList;
import static org.junit.Assert.*;

/**
* La classe machine
* @author Yvan
* @version 1.0
*/
public class PlayerTest{


    @test
    public final void PlayerTest(){
        HumanPlayer player = new HumanPlayer("test",1);
        assertNotNull("Instance non creee", player);

        EasyPlayer auto = new EasyPlayer("test",1);
        assertNotNull("Instance non creee", auto);

        DifficultPlayer auto = new DifficultPlayer("test",1);
        assertNotNull("Instance non creee", auto);
    }

    @test
    public final void getNameTest(){
        HumanPlayer player = new HumanPlayer("test",1);
        assertEquals("test", player.getName());

        AutoPlayer auto = new EasyPlayer("test",1);
        assertEquals("test", auto.getName());
```

```java
    }

    @test
    public final void actionsTest(){
        HumanPlayer player = new HumanPlayer("test",1);
        player.openBulding(1);
        assertEquals(1, player.getMyBuldings().size());

        player.recrutWorker(2);
        assertEquals(1, player.getMyWorkers().size());

        player.sendWorking(1);
        assertEquals(0, player.getMyWorkers().size());

        player.takeEcus();
        assertEquals(5, player.getMoney());

        player.setAction(6);
        assertEquals(6, player.getNumAction());

        player.setPoints(3);
        assertEquals(3, player.getNumAction());

    }
}
```

Annexes


Worker
Building
Card
Machine
Deck
Mode
Game
Player
AutoPlayer
EasyPlayer
DifficultPlayer
HumanPlayer
GameConfig
Run
Rules
Board
Options
EndGame
Interface
Menu
Lancement
Controller
Save
View
Util


Worker
Building
Machine
Card
Deck
Mode
Game
Player
AutoPlayer
EasyPlayer
DifficultPlayer
HumanPlayer
GameConfig
Run
Rules
Board
Options
Interface
EndGame
Menu
Lancement
Controller
save2
View2
Controller2
Util2