GIRAUD-RENARD Benjamin

Département INFORMATIQUE
1ère année - Groupe D

Année 2020 - 2021

# M2107 Projet de programmation 2021

## Cahier d'analyse et de conception

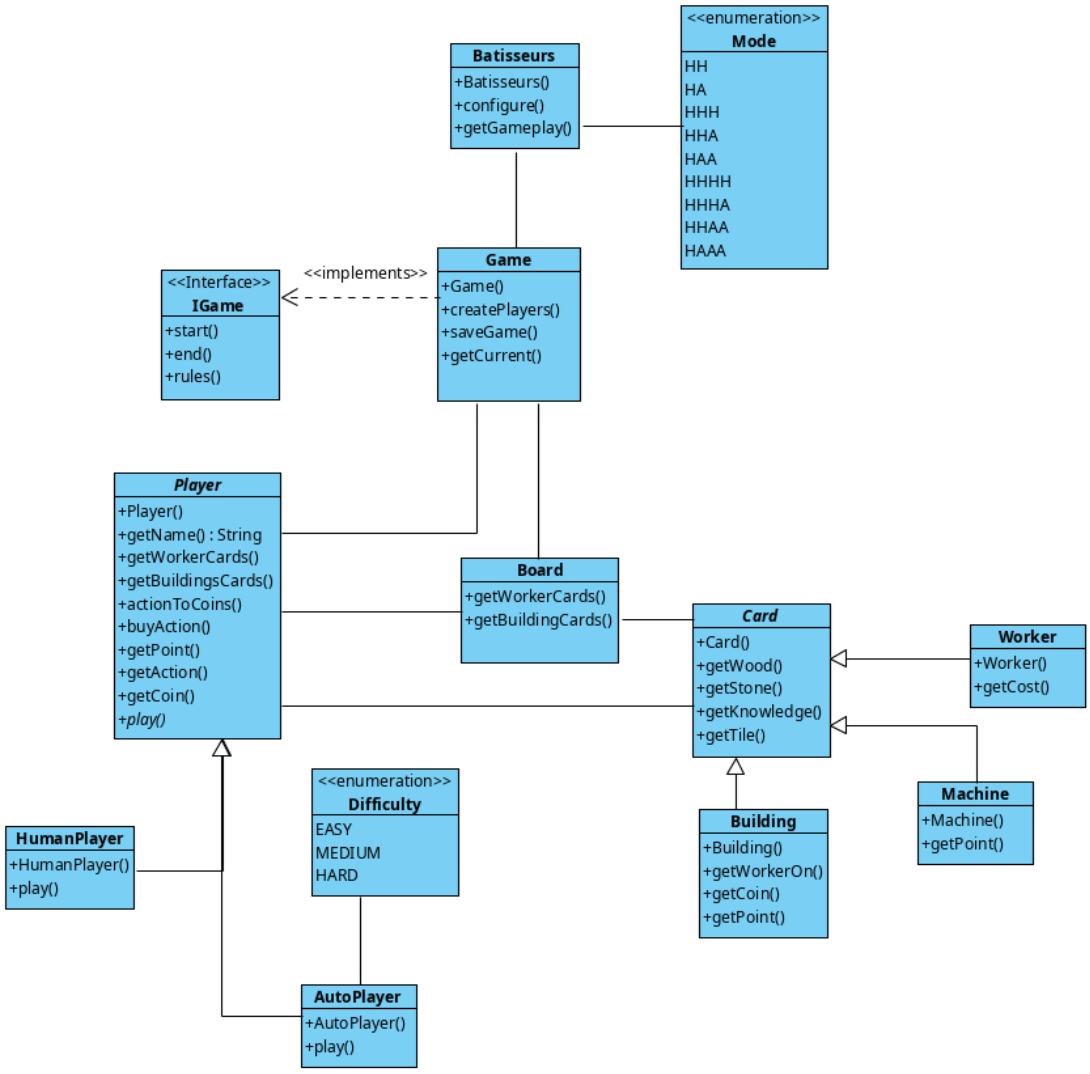
Création d'une version numérique du jeu de société "Les bâtisseurs : Moyen-Age" pour le compte de M. Sébastien Lefèvre


Destinataire
M. Sébastien Lefèvre
Enseignant-chercheur en informatique

## Cahier des charges revisité :

Grâce aux différentes remarques évoquées par différents étudiants, j'ai pu améliorer le cahier des charges en ajoutant notamment certains rendus ainsi que les coordonnées de l'IUT.

## Diagramme de classe d'analyse



Le diagramme d'analyse permet de représenter rapidement la prévision de l'architecture du projet. Sans attributs ni paramètres, il permet de simplifier la vision générale du projet avant son commencement.

La classe Bâtisseurs est la classe permettant d'appeler le rester du programme ainsi que de choisir le style visuel (interface ou non). Cette classe permettra de lire dans les fichiers de configuration les indications d'une partie sauvegardée.

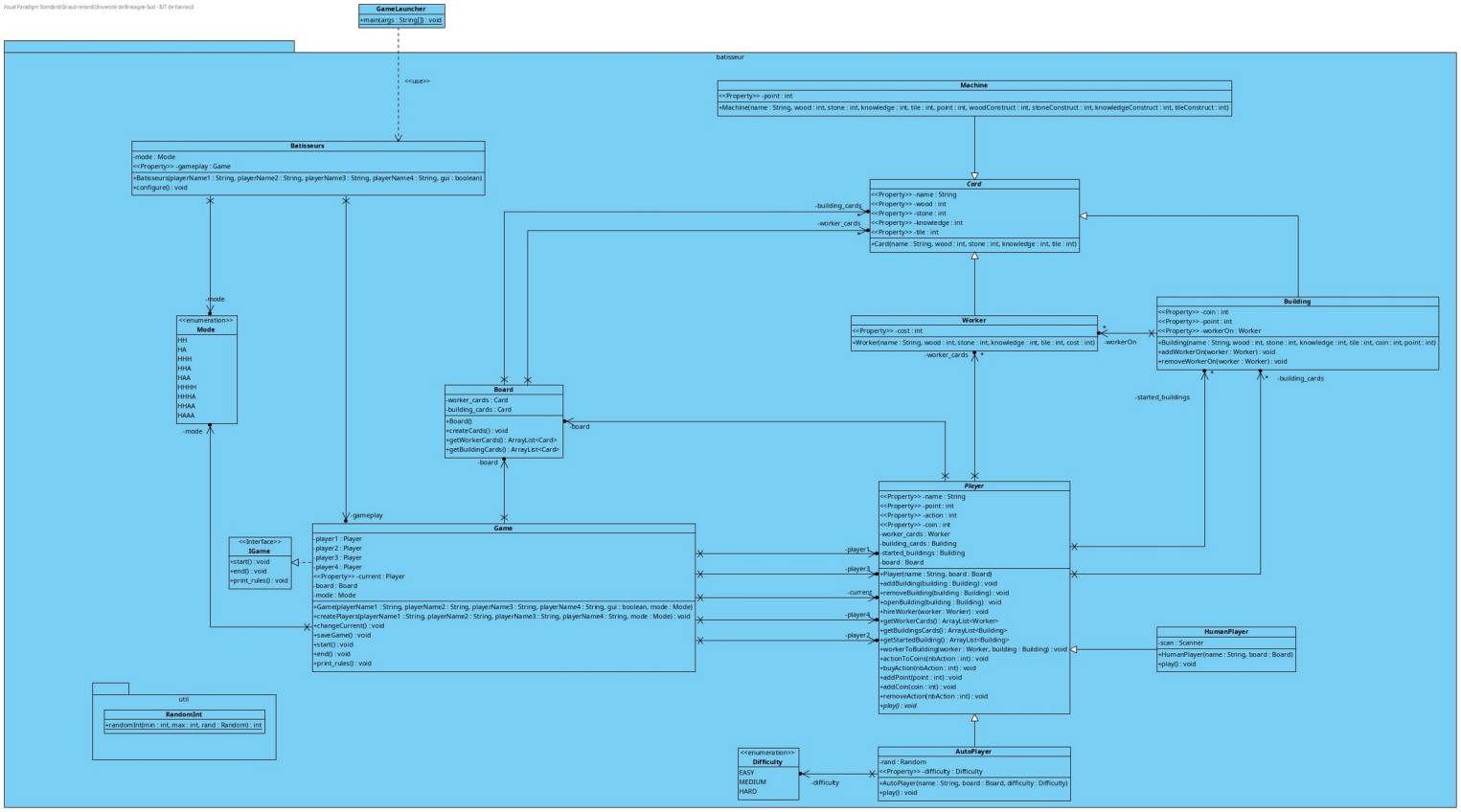
La classe Game permet, elle d'initialiser une partie et de la commencer / terminer. Il s'agit de la classe possédant la boucle principale du jeu, permettant de lancer une partie sauvegardée ou une nouvelle.

La classe abstraite Player possède un ensemble de méthodes permettant de convenir aux règles du jeu. Chacune de ses méthodes seront appelées durant la méthode play des sous-classes « HumanPlayer » et « AutoPlayer ».

Les différentes cartes seront tout d'abord initialisées sur le plateau à l'aide de la classe « Board » et sont regroupées par catégories correspondantes aux sous-classes de « Card ».

Toutes ses interactions seront réunies dans un package et pourront être lancé par un Launcher extérieur ou directement depuis le jar prévu à la fin du projet. Sa représentation plus précise est disponible dans le diagramme de conception

## Diagramme de conception



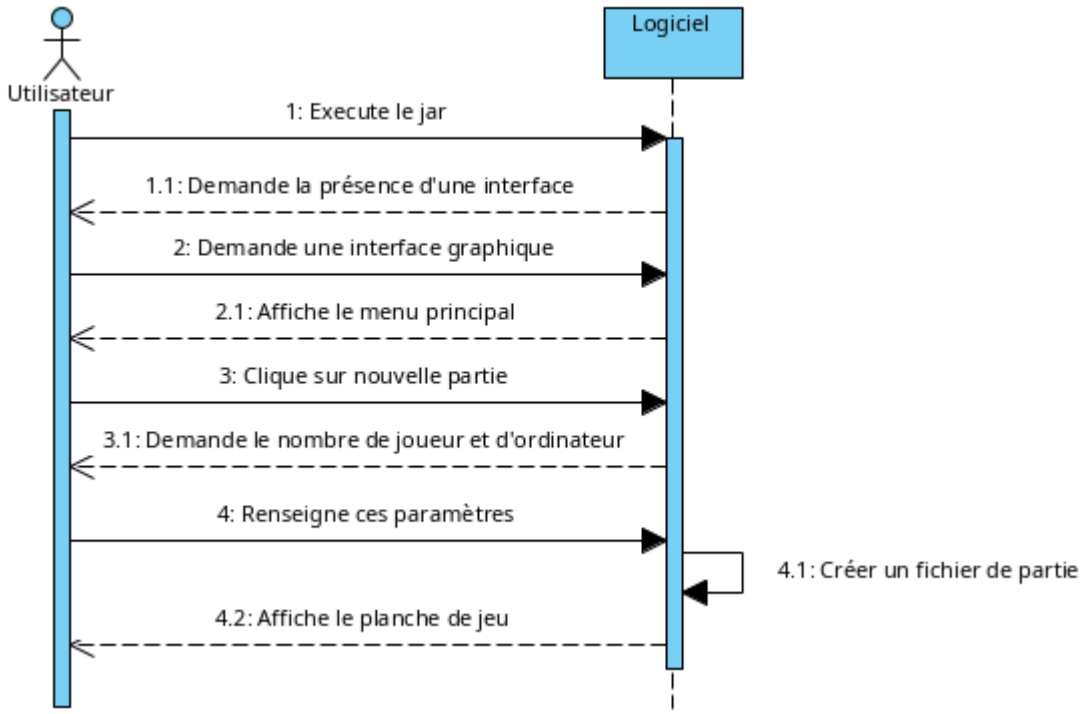

Le cahier d'analyse correspond à une version plus détaillée du cahier d'analyse. Nous pouvons y apprendre davantage comment le jeu va fonctionner et les différentes méthodes que nous allons utiliser. Nous pouvons également y apprendre la nature des différentes liaisons ainsi que le type de retour des méthodes.

Restant tout de même un outil préparatif, le projet final ne ressemblera certainement pas à cela. Des modifications peuvent avoir lieu entre ce diagramme prévisionnel et celui de fin de projet

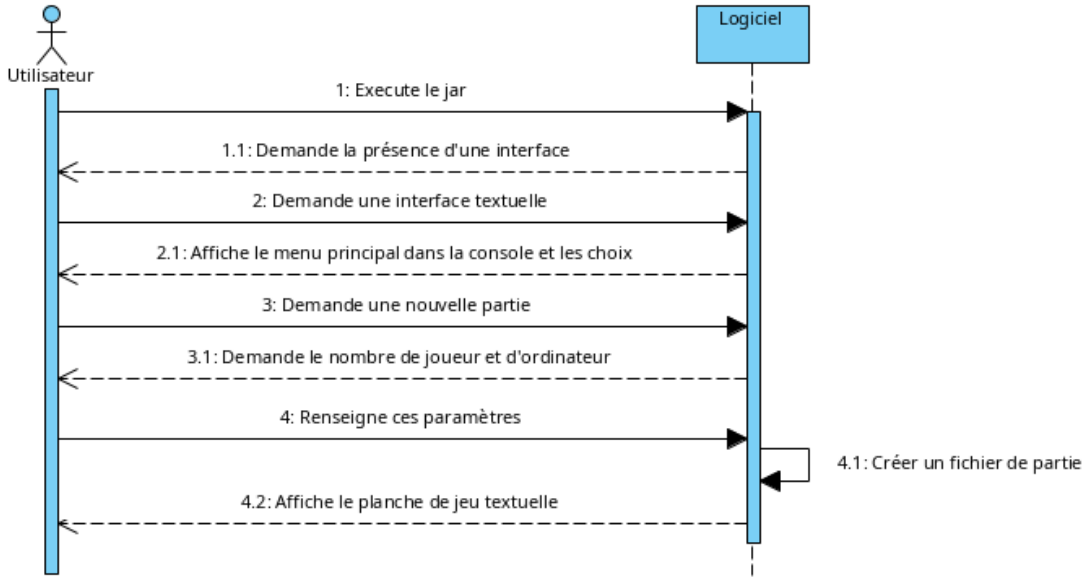
# Diagramme de séquence boite noire

Dans cette partie, sauf indication contraire, le mode de jeu est supposé Graphique.

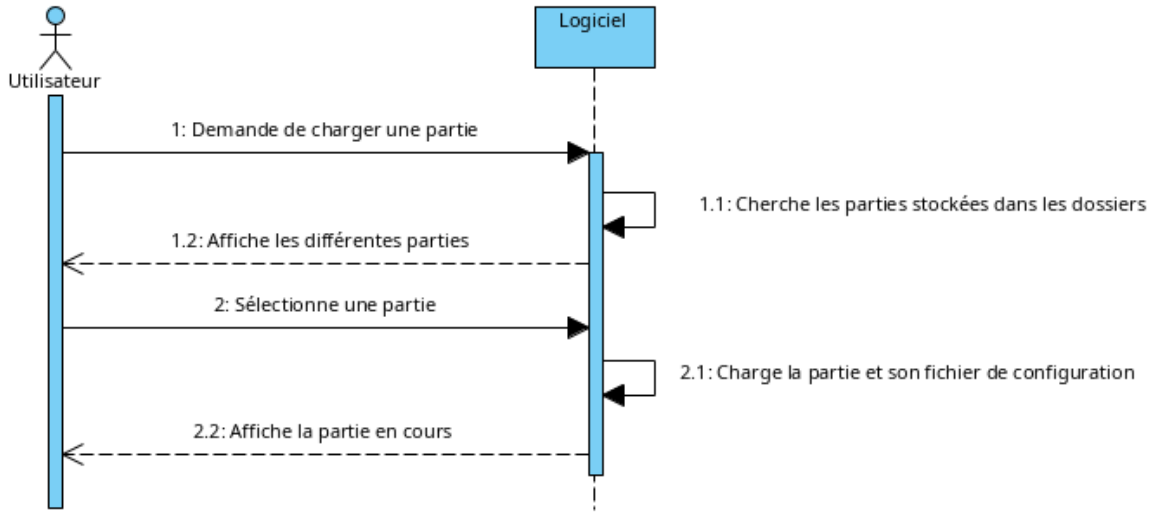
## Lancer partie GUI :



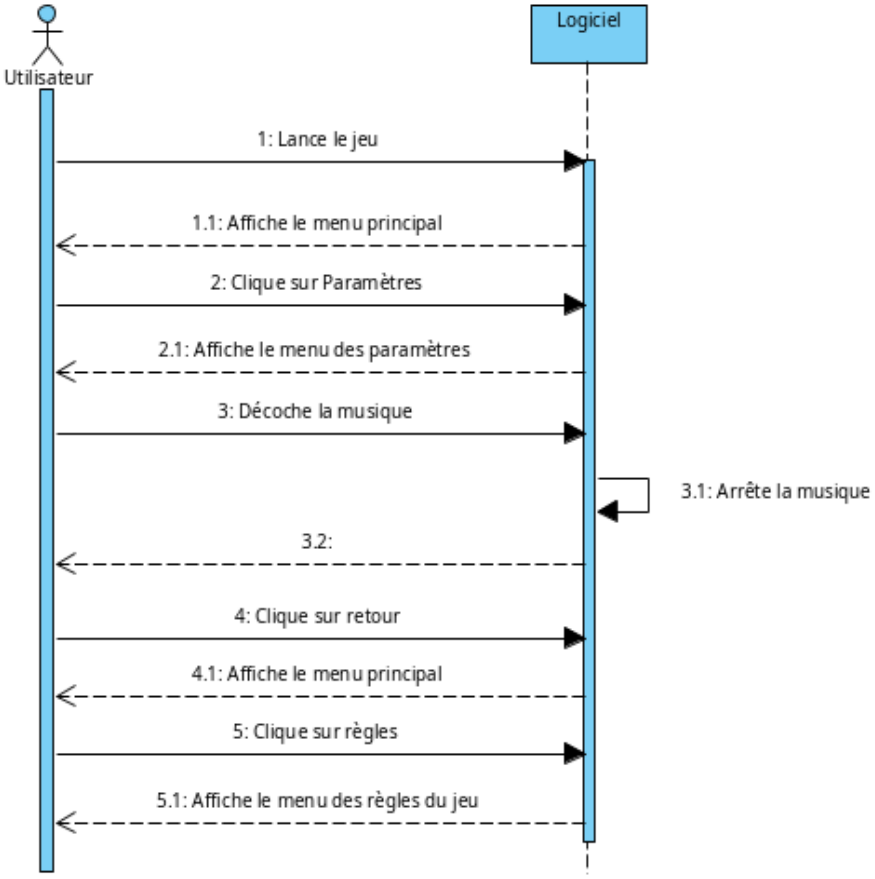
## Lancer partie Console :



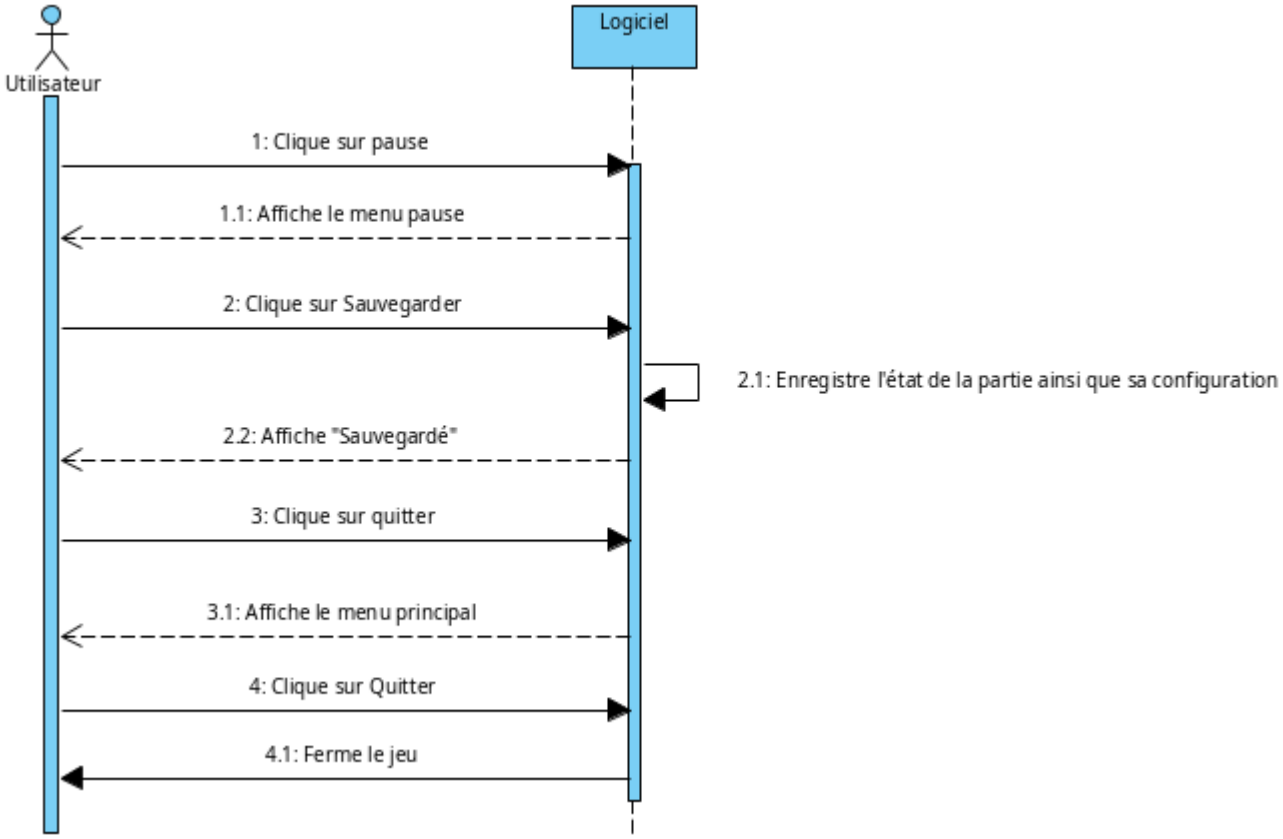

## Charger une partie :



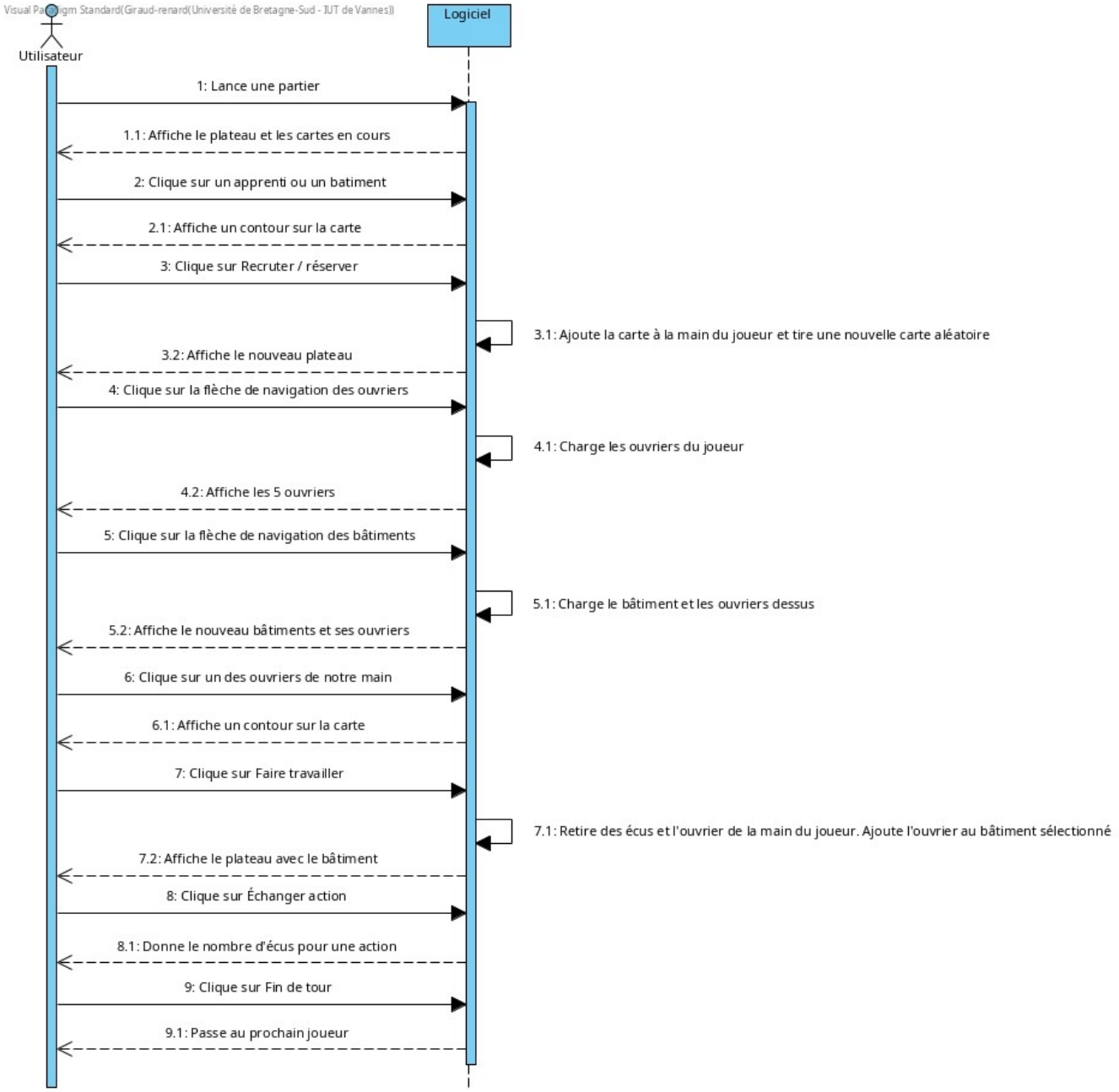
## Naviguer dans le menu principal :



## Sauvegarder et fermer une partie :



## Déroulement supposé d'un tour de jeu :



## Spécification des formats de fichier

Chaque partie créera son fichier de configuration sous une forme simple

nombreD'humain:NombreD'ordis:ModeDeJeu

Grâce à l'interface Serializable, nous allons écrire l'état de chaque élément de la partie dans un dossier réservé à celle-ci. Ainsi lors de la relecture, nous pouvons séparer correctement les différents éléments.

## Squelette des classes principales

### GameLauncher

```
import batisseur.*;

public class GameLauncher {

	/**
	 * run the game
	 * @param args
	 */
	public static void main(String[] args) {

	}

}
```

### Mode

```
package batisseur;

public enum Mode {
	HH,
	HA,
	HHH,
	HHA,
	HAA,
```

```
                HHHH,
        HHHA,
        HHAA,
        HAAA
}
```

## Bâtisseurs

```
package batisseur;

public class Batisseurs {

        private Mode mode;
        private Game gameplay;

        /**
         * create and launch the game
         * @param playerName1 the first player name
         * @param playerName2 the second player name
         * @param playerName3 the third player name
         * @param playerName4 the fourth player name
         * @param gui true if you want to play with the interface on
         */
        public Batisseurs(String playerName1, String playerName2, String
playerName3, String playerName4, boolean gui) {
                // TODO - implement Batisseurs.Batisseurs
        }

        /*
        * read the configuration file
        */
        public void configure() {
                // TODO - implement Batisseurs.configure
        }

        /**
         * get the current gameplay
         * @return the game
         **/
        public Game getGameplay() {
                return this.gameplay;
```

```
                }

}
```

## IGame

```
package batisseur;

public interface IGame {

	public void start();
	public void end();
	public void print_rules();

}
```

## Game

```
package batisseur;

public class Game implements IGame {

	private Player player1;
	private Player player2;
	private Player player3;
	private Player player4;
	private Player current;
	private Board board;
	private Mode mode;

	/**
	 * create the main game with all the player
	 * @param playerName1 the first player name
	 * @param playerName2 the second player name
	 * @param playerName3 the third player name
	 * @param playerName4 the fourth player name
	 * @param gui true if you want to play with the interface on
	 * @param mode the current mode
	 */
	public Game(String playerName1, String playerName2, String player-
Name3, String playerName4, boolean gui, Mode mode) {
		// TODO - implement Game.Game
	}

	/**
```

```
             * create all the player depends of the mode
         * @param playerName1 the first player name
         * @param playerName2 the second player name
         * @param playerName3 the third player name
         * @param playerName4 the fourth player name
         * @param mode the current mode
         */

        public void createPlayers(String playerName1, String playerName2,
String playerName3, String playerName4, Mode mode) {
                // TODO - implement Game.createPlayers
        }

        /**
         * change the current player
         **/
        public void changeCurrent() {
                // TODO - implement Game.changeCurrent
        }

        /**
         * save the game
         **/
        public void saveGame() {
                // TODO - implement Game.saveGame
        }

        /**
         * get the current player
         * @return the current Player
         **/
        public Player getCurrent() {
                return this.current;
        }

        /**
         * contains the main loop
         **/
        public void start() {

        }

        /**
```

```
                * when a player reached 13 points
         **/
        public void end() {

        }

        /**
         * print the rules
         **/
        public void print_rules() {

        }

}
```

## Player

```
package batisseur;

import java.util.ArrayList;

public abstract class Player {

        private String name;
        private int point;
        private int action;
        private int coin;

        private ArrayList<Worker> worker_cards;
        private ArrayList<Building> building_cards;
        private ArrayList<Building> started_buildings;
        private Board board;

        /**
         * Generate a new player
         * @param name
         * @param board
         */
        public Player(String name, Board board) {
                if(name != null && board != null) {
                        this.name = name;
```

```
                        this.board = board;

                this.action = 3;

                this.worker_cards = new ArrayList<Worker>();
                this.building_cards = new ArrayList<Building>();
                this.started_buildings = new ArrayList<Building>();
        }
    }

    /**
     * get the name of the player
     * @return the name
     **/
    public String getName() {
            return this.name;
    }

    /**
     * add a building to your cards
     * @param building the building to add

     **/
    public void addBuilding(Building building) {

            if(building != null) {
                    this.building_cards.add(building);
            }
    }

    /**
     * remove a building from your cards
     * @param building the building to remove
     **/
    public void removeBuilding(Building building) {
            if(building != null) {
                    this.building_cards.remove(building);
            }
    }

    /**
     * open a building and add it to the arraylist
     * @param building
```

```
         */
        public void openBuilding(Building building) {
                this.started_buildings.add(building);
                this.building_cards.remove(building);
        }

        /**
         * hire a new worker
         * @param worker
         */
        public void hireWorker(Worker worker) {
                if(this.coin-worker.getCost() >= 0) {
                        this.worker_cards.add(worker);
                        this.coin -= worker.getCost();
                }
        }

        /**
         * get all the worker cards of the player
         * @return all the worker cards of the player
         **/
        public ArrayList<Worker> getWorkerCards() {
                return this.worker_cards;
        }

        /**
         * get all the building cards of the player
         * @return all the building cards of the player
         **/

        public ArrayList<Building> getBuildingsCards() {
                return this.building_cards;
        }

        /**
         * get all the started building cards of the player
         * @return all the building cards of the player
         **/
        public ArrayList<Building> getStartedBuilding() {
                return this.started_buildings;
        }

        /**
```

```
         * make a worker work on a certain building
     * @param building
     * @param worker
     */
    public void workerToBuilding(Worker worker, Building building) {
        if(this.started_buildings.contains(building)) {
            building.addWorkerOn(worker);
        }
    }

    /**
     * change action to some coins
     * @param nbAction
     */
    public void actionToCoins(int nbAction) {
        if(nbAction == 1) {
            this.coin += 1;
        } else if(nbAction == 2) {
            this.coin += 3;
        } else if(nbAction == 3) {
            this.coin += 6;
        }
        this.action -= nbAction;
    }

    /**
     * buy action from coin
     * @param nbAction
     */
    public void buyAction(int nbAction) {
        if(this.coin >= nbAction * 5) {
            this.action += nbAction;
            this.coin -= nbAction*5;
        }
    }



    /**
     * get the player's point
     * @return the number of point of the player
     **/
    public int getPoint() {
```

```
        return this.point;
    }

    /**
     * get the player's number of action
     * @return the number of action of the player
     **/
    public int getAction() {
        return this.action;
    }

    /**
     * get the player's coin
     * @return the number of coin of the player
     **/
    public int getCoin() {
        return this.coin;
    }

    /**
     * set a new name
     * @param name the new name
     **/
    public void setName(String name) {
        this.name = name;
    }

    /**
     * add some point to the player
     * @param point the number of point you need to add
     **/
    public void addPoint(int point) {
        this.point += point;
    }

    /**
     * add some coin to the player
     * @param coin the number of coin you need to add
     **/
    public void addCoin(int coin) {
        this.coin += coin;
    }
```

```
	/**
	 * set a new number action
	 * @param nbAction the new number of action
	 **/
	public void setAction(int nbAction) {
		this.action = nbAction;
	}

	/**
	 * set a new number coin
	 * @param nbCoin the new number of coin
	 **/
	public void setCoin(int nbCoin) {
		this.coin = nbCoin;
	}

	/**
	 * remove action
	 * @param nbAction the number of action you want to remove
	 **/
	public void removeAction(int nbAction) {
		if(this.action - nbAction >= 0) {
			this.action -= nbAction;
		} else {
			System.err.println("removeAction : too many action to
remove");
		}
	}

	//play
	public abstract void play();
}
```

## Difficulty

```
package batisseur;

/**
```

```
             * create an enum for the difficulty of the bot
 **/
public enum Difficulty {
        EASY,
        MEDIUM,
        HARD
}
```

## AutoPlayer

```
package batisseur;

import java.util.Random;
import util.RandomInt;

public class AutoPlayer extends Player {

	private Random rand;
	private Difficulty difficulty;

	/**
	 * Create an autoplayer using the name and the difficulty you want
	 * @param name
	 * @param board
	 * @param difficulty
	 */
	public AutoPlayer(String name, Board board, Difficulty difficulty) {
		super(name, board);
		this.difficulty = difficulty;
		this.rand = new Random();
	}

	/**
	 * do the player play
	 **/
	public void play() {
		// TODO - implement AutoPlayer.play
	}

	/**
	 * get the difficulty of the bot
	 * @return the difficulty
	 **/
	public Difficulty getDifficulty() {
		return this.difficulty;
	}

	/**
	 * set the difficulty of the bot
```

```
                * @param difficulty the new difficulty
         **/
        public void setDifficulty(Difficulty difficulty) {
                this.difficulty = difficulty;
        }
}
```

## HumanPlayer

```
package batisseur;

import java.util.Scanner;

public class HumanPlayer extends Player {

        private Scanner scan;

        /**
         * Create a new Human player
         * @param name the name of the player
         * @param board the current board
         */
        public HumanPlayer(String name, Board board) {
                super(name, board);
        }

        /**
         * play
         **/
        public void play() {
                // TODO - implement HumanPlayer.play
        }

}
```

## Board

```
package batisseur;

import java.util.ArrayList;
```

```
import java.util.Random;
import util.RandomInt;

public class Board {

	private ArrayList<Card> worker_cards;
	private ArrayList<Card> building_cards;

	public Board() {

	}

	/**
	 * create all the cards using the file
	 **/

	public void createCards() {
		// TODO - implement Board.createCards
	}

	/**
	 * get the worker cards on the board
	 * @return the arraylist containing the worker cards on the board
	 **/
	public ArrayList<Card> getWorkerCards() {
		return this.worker_cards;
	}

	/**
	 * get the building cards on the board
	 * @return the arraylist containing the building cards on the board
	 **/
	public ArrayList<Card> getBuildingCards() {
		return this.building_cards;
	}
}
```

## Card

```
package batisseur;

public abstract class Card {
```

```
	private String name;
	private int wood;
	private int stone;
	private int knowledge;
	private int tile;

	/**
	 * Create a new card
	 * @param name the name
	 * @param wood the number of wood
	 * @param stone the number of stone
	 * @param knowledge the number of knowledge
	 * @param tile the number of tile
	 */
	public Card(String name, int wood, int stone, int knowledge, int
tile) {
		if(name != null) {
			this.name = name;
			this.wood = wood;
			this.stone = stone;
			this.knowledge = knowledge;
			this.tile = tile;
		} else {

			System.err.println("Card : name null");
		}
	}

	/**
	 * get the wood value of the card
	 * @return the wood
	 **/
	public int getWood() {
		return this.wood;
	}

	/**
	 * get the stone value of the card
	 * @return the stone
	 **/
	public int getStone() {
		return this.stone;
```

```
		}

	/**
	 * get the knowledge value of the card
	 * @return the knowledge
	 **/
	public int getKnowledge() {
		return this.knowledge;
	}

	/**
	 * get the tile value of the card
	 * @return the tile
	 **/
	public int getTile() {
		return this.tile;
	}

	/**
	 * get the name of the card
	 * @return the name
	 **/
	public String getName() {
		return this.name;
	}

}
```

## Building

```
package batisseur;

import java.util.ArrayList;

public class Building extends Card {

	private int coin;
	private int point;
	private ArrayList<Worker> workerOn;

	/**
	 * Create a new building card
	 * @param name the name of the card
	 * @param wood the number of wood to build
	 * @param stone the number of stone to build
	 * @param knowledge the number of knowledge to build
	 * @param tile the number of tile to build
	 * @param coin the number of coin you earn
	 * @param point the number of point you earn
	 */
	public Building(String name, int wood, int stone, int knowledge, int
tile, int coin, int point) {
		super(name, wood, stone, knowledge, tile);
		if(coin >=0 && point >=0) {
			this.coin = coin;
			this.point = point;
			this.workerOn = new ArrayList<Worker>();
		} else {
			System.err.println("Building : coin or point inva-
lid");
		}
	}

	/**
	 * get the worker on
	 * @return the arrayList containing all the worker assigned to the
building
	 **/
	public ArrayList<Worker> getWorkerOn() {
```

```
                return this.workerOn;
        }

        /**
         * add a worker on the building
         * @param worker the worker you want to add on
         **/
        public void addWorkerOn(Worker worker) {
                this.workerOn.add(worker);

        }

        /**
         * remove a worker on the building
         * @param worker the worker you want to remove from
         **/
        public void removeWorkerOn(Worker worker) {
                this.workerOn.remove(worker);
        }

        /**
         * get the coin you get when you finished the building
         * @return the number of coin
         **/
        public int getCoin() {
                return this.coin;
        }

        /**
         * get the point you get when you finished the building
         * @return the number of point
         **/
        public int getPoint() {
                return this.point;
        }

        /**
         * set a number of coin
         * @param coin the new number of coin
         **/
        public void setCoin(int coin) {
                this.coin = coin;
        }
```

```
        /**
         * set a number of point
         * @param point the new number of point
         **/
        public void setPoint(int point) {
                this.point = point;
        }

}
```

## Worker

```
package batisseur;

public class Worker extends Card {

	private int cost;

	/**
	 * Generate a new Worker
	 * @param name the name of the worker
	 * @param wood the number of wood it product
	 * @param stone the number of stone it product
	 * @param knowledge the number of knowledge it product
	 * @param tile the number of tile it product
	 * @param cost the salary
	 */
	public Worker(String name, int wood, int stone, int knowledge, int
tile, int cost) {
		super(name, wood, stone, knowledge, tile);
		if(cost >= 0) {
			this.cost = cost;
		} else {
			System.err.println("Worker : cost invalid");
		}
	}

	/**
	 * get the cost value of the worker
	 * @return the cost of the worker
	 **/
	public int getCost() {
		return this.cost;
	}

	public void setCost(int cost) {
		this.cost = cost;
	}

}
```

## Machine

```
package batisseur;

public class Machine extends Card {

	private int point;

	/**
	 * Create a new card machine
	 * @param name the name of the card
	 * @param wood the wood value the machine require to be created
	 * @param stone the stone value the machine require to be created
	 * @param knowledge the knowledge value the machine require to be
created
	 * @param tile the tile value the machine require to be created
	 * @param point the number of point the machine will product
	 * @param woodConstruct the wood value the machine will product
	 * @param stoneConstruct the stone value the machine will product
	 * @param knowledgeConstruct the knowledge value the machine will
product
	 * @param tileConstruct the tile value the machine will product
	 */
	public Machine(String name, int wood, int stone, int knowledge, int
tile, int point, int woodConstruct, int stoneConstruct, int knowledgeCons-
truct, int tileConstruct) {
		super(name, wood, stone, knowledge, tile);
	}

	/**
	 * get the number of point the machine product
	 * @return the number of point
	 **/
	public int getPoint() {
		return this.point;
	}

}
```

## Tests unitaires Junit

Pour tester, nous utilisons le framework JUnit, permettant de réaliser des tests complet et rapidement.

Voici l'exemple de trois tests effectués sur les classes AutoPlayer, Building et Worker

### AutoPlayerTest

```
package test;

import org.junit.*;
import static org.junit.Assert.*;
import batisseur.AutoPlayer;
import batisseur.Player;
import batisseur.Difficulty;
import batisseur.Worker;
import batisseur.Building;
import batisseur.Board;

import java.util.ArrayList;

public class AutoPlayerTest {

    AutoPlayer p;

    @Before()
    public void setUp() {
        p = new AutoPlayer("name",new Board(),Difficulty.EASY);
    }

    @After()
    public void tearDown() {
        p = null;
    }

    @Test()
    public void testAutoPlayer() {
        assertNotNull(p);
```

```
            }

    @Test()
    public void getDifficulty() {
        assertTrue(p.getDifficulty() == Difficulty.EASY);
        p.setDifficulty(Difficulty.HARD);
        assertTrue(p.getDifficulty() == Difficulty.HARD);
        assertFalse(p.getDifficulty() == Difficulty.EASY);
    }

    @Test()
    public void getName() {
        assertSame(p.getName(), "name");
    }

    @Test()
    public void getPoint() {

        assertTrue(p.getPoint() == 0);
        p.addPoint(3);
        assertTrue(p.getPoint() == 3);
        p.addPoint(3);
        assertTrue(p.getPoint() == 6);
    }

    @Test()
    public void getCoin() {
        assertTrue(p.getCoin() == 0);
        p.addCoin(3);
        assertTrue(p.getCoin() == 3);
        p.addCoin(3);
        assertTrue(p.getCoin() == 6);
    }

    @Test()
    public void getAction() {
        p.setAction(3);
        assertTrue(p.getAction() == 3);
        p.removeAction(2);
        assertTrue(p.getAction() == 1);
        p.removeAction(2);
        assertTrue(p.getAction() == 1);
    }
```

```java
@Test()
public void buyAction() {
    p.setAction(0);
    p.buyAction(2);
    assertFalse(p.getAction() == 2);
    p.addCoin(10);
    p.buyAction(2);
    assertTrue(p.getAction() == 2);
    assertTrue(p.getCoin() == 0);
    p.addCoin(3);
    p.buyAction(1);
    assertFalse(p.getPoint() == 3);
    assertTrue(p.getCoin() == 3);
}

@Test()
public void actionToCoins() {
    p.setAction(3);
    p.setCoin(0);
    p.actionToCoins(3);
    assertTrue(p.getCoin() == 6);
    p.setAction(3);
    p.setCoin(0);
    p.actionToCoins(2);

    assertTrue(p.getCoin() == 3);
    assertTrue(p.getAction() == 1);
    p.setCoin(0);
    p.actionToCoins(1);
    assertTrue(p.getCoin() == 1);
    assertTrue(p.getAction() == 0);
}

@Test()
public void workerToBuilding() {
    Worker w1 = new Worker("Test",0,1,2,3,4);
    Building b1 = new Building("name",1,2,3,0,3,5);
    p.addBuilding(b1);
    p.workerToBuilding(w1,b1);
    assertFalse(p.getStartedBuilding().contains(b1));
    assertTrue(p.getBuildingsCards().contains(b1));
    p.openBuilding(b1);
```

```
                    p.workerToBuilding(w1,b1);
        assertTrue(p.getStartedBuilding().contains(b1));
        assertFalse(p.getBuildingsCards().contains(b1));
        ArrayList<Worker> worker = b1.getWorkerOn();
        assertTrue(b1.getWorkerOn().contains(w1));
    }

    @Test()
    public void getBuildingsCards() {
        Building b2 = new Building("name",1,2,3,0,3,5);
        ArrayList<Building> arr = new ArrayList<Building>();
        arr.add(b2);
        p.addBuilding(b2);
        assertEquals(arr,p.getBuildingsCards());
    }

    @Test()
    public void getWorkerCards() {
        Worker w1 = new Worker("Test",0,1,2,3,4);
        p.setCoin(30);
        p.hireWorker(w1);
        assertTrue(p.getWorkerCards().contains(w1));
        assertTrue(p.getCoin() == 26);
    }
}
```

## BuildingTest

```
package test;

import org.junit.*;
import static org.junit.Assert.*;
import batisseur.Building;
import batisseur.Worker;

import java.util.ArrayList;

public class BuildingTest {

    Building b;

    @Before()
    public void setUp() {
        b = new Building("name",1,2,3,0,3,5);
    }

    @After()
    public void tearDown() {
        b = null;
    }

    @Test()
    public void testBuilding() {
        assertNotNull(b);
    }

    @Test()
    public void getWorkerOn() {
        ArrayList<Worker> workerOn = new ArrayList<Worker>();
        Worker w1 = new Worker("Test",0,1,2,3,4);
        b.addWorkerOn(w1);
        workerOn.add(w1);
        assertEquals(b.getWorkerOn(),workerOn);
        Worker w2 = new Worker("Test",0,1,2,3,4);
        b.addWorkerOn(w2);
        workerOn.add(w2);
        assertEquals(b.getWorkerOn(),workerOn);
```

```
                }

    @Test()
    public void addWorkerOn() {
        Worker w2 = new Worker("Test",0,1,2,3,4);
        b.addWorkerOn(w2);
        assertTrue(b.getWorkerOn().contains(w2));
    }

    @Test()
    public void removeWorkerOn() {
        Worker w3 = new Worker("Test",0,1,2,3,4);
        b.addWorkerOn(w3);
        b.removeWorkerOn(w3);
        assertFalse(b.getWorkerOn().contains(w3));
    }

    @Test()
    public void getCoin() {
        assertTrue(b.getCoin() == 3);
        b.setCoin(6);
        assertTrue(b.getCoin() == 6);
    }

    @Test()
    public void getPoint() {
        assertTrue(b.getPoint() == 5);
        b.setPoint(2);
        assertTrue(b.getPoint() == 2);
    }
}
```

## WorkerTest

```
package test;

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

import java.util.ArrayList;
```

```
public class WorkerTest {

    Worker w;

    @Before()
    public void setUp() {
        w = new Worker("name",1,2,3,0,3);
    }

    @After()
    public void tearDown() {
        w = null;
    }

    @Test()
    public void testWorker() {

        assertNotNull(w);
    }

    @Test()
    public void getCost() {
        assertTrue(w.getCost() == 3);
        w.setCost(6);
        assertTrue(w.getCost() == 6);
    }

    @Test()
    public void getWood() {
        assertTrue(w.getWood() == 1);
    }

    @Test()
    public void getStone() {
        assertTrue(w.getStone() == 2);
    }

    @Test()
    public void getKnowledge() {
        assertTrue(w.getKnowledge() == 3);
    }

    @Test()
```

```
                public void getTile() {
        assertTrue(w.getTile() == 0);
    }

    @Test()
    public void getName() {
        assertSame(w.getName(), "name");
    }
}
```

## Fichier build.xml pour ANT

Pour faciliter la compilation et l'exécution, nous utilisons ANT. Et pour cela nous avons besoin d'un fichier build.xml spécifiant les différentes actions possibles.

```
<project name="LesBatisseurs" default="run" basedir=".">
	<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="${build}/javadoc"/>
	<property name="test" value="${build}/test"/>
	<property name="mainClass" value="Batisseurs"/>
	<property name="jarName" value="${mainClass}"/>

	<target name="init">
		<mkdir dir="${build}"/>
		<mkdir dir="${jar}"/>
		<mkdir dir="${class}"/>
		<mkdir dir="${test}"/>
	</target>

	<target name="clean">
		<delete dir="${build}"/>
	</target>

	<target name="compile" depends="init">
		<javac srcdir="${src}" destdir="${class}" includeantrun-
time="false">
			<exclude name="test/**"/>
		</javac>
	</target>

	<target name="jar" depends="compile">
	 <jar jarfile="${jar}/${jarName}.jar" basedir="${class}">
	    <manifest>
	       <attribute name="Main-Class" value="${mainClass}"/>
	    </manifest>
```

```
                </jar>
    </target>

       <target name="run" depends="jar">
              <java jar="${jar}/${jarName}.jar" fork="true"/>
       </target>


       <target name="javadoc">
              <delete dir="${javadoc}"/>
              <javadoc author="true"
                           destdir="${javadoc}">
                     <fileset dir="${src}">
                           <include name="**"/>
                     </fileset>
              </javadoc>
       </target>

       <target name="compile-test" depends="compile">
              <javac srcdir="${src}/test" destdir="${test}" includeantrun-
time="true">
                     <classpath>
                           <pathelement path="${class}"/>
                     </classpath>
              </javac>
       </target>

       <target name="test" depends="compile-test">
              <junit printsummary="on" haltonfailure="off" fork="true" in-
cludeantruntime="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>
```