Computational Game Theory
Integrated Master (BSc. + MSc.) of Computer Science and Engineering
Faculty of Sciences and Technology of New University of Lisbon of New University of Lisbon (FCT NOVA | FCT/UNL)
2018/2019 - 2nd Semester

STRATEGIES FOR THE PRISONERS' DILEMMA

By Rúben André Barreiro

INTRODUCTION

The ***Prisoners' Dilemma*** is a standard example of a *game analyzed in game theory* that shows why two completely rational individuals might not cooperate, even if it appears that it is in their best interests to do so. (*Wikipedia* - https://en.wikipedia.org/wiki/Prisoner%27s_dilemma)

Brief History

It was originally framed by ***Merrill Flood*** and ***Melvin Dresher*** while working at RAND in 1950. ***Albert W. Tucker*** formalized the game with prison sentence rewards and named it "prisoner's dilemma". (*Wikipedia* - https://en.wikipedia.org/wiki/Prisoner%27s_dilemma)

Problem

Brief Definition

Two members of a criminal gang are arrested and imprisoned. Each prisoner is in solitary confinement with no means of communicating with the other. The prosecutors lack sufficient evidence to convict the pair on the principal charge, but they have enough to convict both on a lesser charge. Simultaneously, the prosecutors offer each prisoner a bargain. Each prisoner is given the opportunity either to betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. The offer is:

- If **A** and **B** each **betray** the other, each of them serves **two years** in prison;
- If **A** betrays **B** but **B remains silent**, **A** will be set free and **B** will serve **three years** in prison (**and vice versa**);
- If **A** and **B** both **remain silent**, both of them will only serve **one year** in prison (**on the lesser charge**);

Player 1 \ Player 2	Cooperate	Defect
Cooperate	(3,3)	(4,0)
Defect	(0,4)	(1,1)

Table 1: The matrix of utilities of the original problem

(*Wikipedia* - https://en.wikipedia.org/wiki/Prisoner%27s_dilemma)

Solving the Problem

In this course, it was asked to solve a modified version of the original Prisoners' Dilemma. Where the matrix of utilities is the following one:

Player 1 \ Player 2	Cooperate	Defect
Cooperate	(3,3)	(4,0)
Defect	(0,4)	(1,1)

Table 2: The matrix of utilities of the considered problem

Strategies Implemented

1. Gradual Strategy

Initially, was implemented the known Gradual Strategy. This strategy consists, basically, in use the "Cooperate" action in the first action and then, continues to do so as long as the opponent make "Cooperate" actions too. Then, after the first "Defect" action of the opponent, it defects one time and cooperates two times. After the second "Defect" action of the opponent, it defects two times and cooperates two times... ***After the nth defection of the opponent, it reacts with n consecutive defections and then, with two cooperates (some kind of "calm down")***. ***When it's being made a defection from 1 to n, or two consecutive cooperatess, it will continuing to detect the opponent's "Defect" actions, to be made more punishments in the future, restarting all the process, but now, making a defection from 1 to (n+1), and then, two "Cooperate" actions***, and so on;

2. Hybrid Gradual Strategy

This strategy it's similar to the first one, ***but with the difference that, uses a harder concept of forgiveness***. In the normal version of the Gradual Strategy, the "calm down" it's always made by two consecutive "Cooperate" actions after nth "Defect" accordingly to the nth "Defect" of the opponent. **This version, follows the same definition of the Gradual Strategy before the first 8 initial "Defect" actions made by the opponent**. ***After that, the "calm downs" can be made with the support of just one or two consecutive "Cooperate" actions, it will depend on the current number of consecutive "Cooperate" actions made by the opponent. If the opponent made such or more than 6 consecutive "Cooperate" actions, it will be applied the "soft calm down" with half of the number of "Defect" actions in the base strategy but, with the same two "Cooperate" actions, if not, will be applied the "hard calm down" with the number of "Defect" actions like the normal version but with just one "Cooperate" action, in order, to recover the utility points loss***. In some cases, it will be verified, ***if the opponent made the "Cooperate" action such or more than 4 consecutive times and in that case, since I'm not currently making "Defect" actions neither "calming down", I will respond with "Defect" actions, i order to gain utility points over my opponent and, fast recover, in sometimes. In the last iteration, since that's known, will be made a "Defect" action***;

3. Hybrid Gradual With Cooperate Leeway Strategy

This strategy differs in comparison to the previous one, in the way, that start with a "Defect" action. And additionally, ***in the case of, I'm not currently making "Defect" actions neither "calming down", I will respond with a "Cooperate" action since the opponent have more than the double of "Defect"actions made in comparison to the number of "Cooperate" actions made, until the moment. And will respond with a "Defect" action, otherwise. It's just a way, to try do some "Cooperate" actions, in the case of, I have some kind of "secure" leeway to do some "Cooperate" actions without get some penalty related with a loss of utility points. Furthermore, will be played the "Defect" action, if the probability of continue playing or iterating be lesser than (⅓) = 33,3333%. If it's known if it's being run the last iteration, will be made a "Defect" action***.

Some experimental tests' results [1]

- *Relevant 5 experimental tests performed with 1 000 Iterations and 100% of Probability of Continue to Iterate, against other Strategies:*

Num. of Test	Num. of Iterations	Probability of Continue [0.0 - 1.0]	Strategy used by Player #1	Strategy used by Player #2	Total of Points of Player #1	Total of Points of Player #2	Winner
Test #1	1 000	1.0	Random Strategy	Hybrid Gradual Strategy	**1 110**	**4 998**	**Player #2 (Hybrid Gradual Strategy)**
Test #2	1 000	1.0	Random Strategy	Hybrid Gradual Strategy	**1 072**	**5 032**	**Player #2 (Hybrid Gradual Strategy)**
Test #3	1 000	1.0	Mimic Strategy (Modified Tit For Tat Strategy)	Hybrid Gradual Strategy	**2 057**	**2 061**	**Player #2 (Hybrid Gradual Strategy)**
Test #4	1 000	1.0	Gradual Strategy	Hybrid Gradual Strategy	**2 050**	**2 106**	**Player #2 (Hybrid Gradual Strategy)**

Table 2: Experimental results with 1 000 iterations and 100% of probability of continue to iterate

The Strategy used for the first round of the Tournament - ***Hybrid Gradual Strategy***:

- As, accordingly with the experimental tests' results, for the **first round of the tournament**, was chosen the ***Hybrid Gradual Strategy*** (*see the code in annex*);

Some experimental tests' results [2]

- *Relevant 5 experimental tests performed with 10 000 000 Iterations and 80% of Probability of Continue to Iterate (considered only Games that achieved such or more than 10 Iterations), against other Strategies:*

Num. of Test	Num. of Iterations	Probability of Continue [0.0 - 1.0]	Strategy used by Player #1	Strategy used by Player #2	Total of Points of Player #1	Total of Points of Player #2	Winner
Test #1	10 000 000 Possible Iterations (14 Iterations Completed)	0.8	Random Strategy	Hybrid Gradual With Cooperate Leeway Strategy	51	63	**Player #2 (Hybrid Gradual With Cooperate Leeway Strategy)**
Test #2	10 000 000 Possible Iterations (14 Iterations Completed)	0.8	Random Strategy	Hybrid Gradual With Cooperate Leeway Strategy	53	57	**Player #2 (Hybrid Gradual With Cooperate Leeway Strategy)**
Test #3	10 000 000 Possible Iterations (12 Iterations Completed)	0.8	Mimic Strategy (Modified Tit For Tat Strategy)	Hybrid Gradual With Cooperate Leeway Strategy	**56**	**56**	----------
Test #4	10 000 000 Possible Iterations (17 Iterations Completed)	0.8	Gradual Strategy	Hybrid Gradual With Cooperate Leeway Strategy	**102**	**102**	----------
Test #5	10 000 000 Possible Iterations (13 Iterations Completed)	0.8	Hybrid Gradual Strategy	Hybrid Gradual With Cooperate Leeway Strategy	**54**	**54**	----------

Table 3: Experimental results with 10 000 000 iterations and 80% of probability of continue to iterate

The Strategy used for the second and third round of the Tournament - ***Hybrid Gradual With Cooperate Leeway Strategy***:

- As, accordingly with the experimental tests' results, for the **second and third round of the tournament**, was chosen the ***Hybrid Gradual With Cooperate Leeway Strategy*** (*see the code in annex*);

Conclusions

The Prisoners' Dilemma isn't an easy computational game to solve, right?! ***The true it's that, there's no specific and "universal" strategy to solve it***. ***We can conclude that, perhaps, playing always in "defence", doing "Defect" actions always, instead of, "Cooperate" actions, can be a good strategy on games of type 1 vs. 1, because you have no round that you don't have any gain of utility, gaining always 1 or 4 utility points***.

But in, an environment where we are playing against more than one opponent in iterated games and considering the average of utility gained in every games, perhaps isn't the best strategy to use, because another two players, can be, possibly, cooperating, and gaining both, 3 utility points. And in some situations, can have more points, in average, than us.

So, maybe, isn't too bad idea, cooperate some times to try to maximise our global utility, since our opponent isn't doing "Defect" actions.

We can think that, implement a strategy that takes that in consideration, maybe should be the best approach for this situations in this computational game, since, of course, tries to avoid, the situations where we do "Cooperate" actions and our opponents do "Defect" actions.

Was proved that, the Prisoners' Dilemma it's a very hard computational game to solve in an universally way. ***This game presents a very complex paradox between individual interests and the common good***. Maybe, the only way to "win" is to change this game itself, and that's should be the larger lesson to learn from the Prisoners' Dilemma. ***A good solution, thinking in a global well-being, would be define with the people involved that, everyone should be together and agree that cooperating is the best solution, and then, everyone agrees to cooperate, and most importantly, everyone agrees that anyone who "betrays" someone will be punished by the collective group*** when they get out of prison. ***Then, that will change the payoff table so that betraying costs more and, cooperating is the better selfish choice and better choice for everyone***.

In life, nothing it's easy, like, per example, tattoo a map of the plant of a prison in the body to plan a successful escape from a prison where our partners in prison cooperates and helps each other in our plan, such as the story of Michael Scofield in the famous TV serie Prison Break... ;)

Some Bibliography and References

- https://en.wikipedia.org/wiki/Prisoner%27s_dilemma
- https://www.quora.com/What-is-the-best-strategy-for-the-Iterated-Prisoners-dilemma-game
- http://jasss.soc.surrey.ac.uk/20/4/12.html#toc-conclusion

Further Information

The Java implementations used for the strategies applied to this computational game, will be provided in the next pages and you can check it the following GitHub's repository (such as, other additional strategies used):

- https://github.com/rubenandrebarreiro/computational-game-theory-tournaments/tree/master/temp/1st%20Tournament%20-%20Prisoners'%20Dilemma

Annexes (Code of Java Implementations for the used Strategies) [1]

Java Class for the Hybrid Gradual Strategy:

```
package play;


/**

*

* ~~~ Prisoners' Dilemma ~~~

*

* Theory of Computational Games

*

* Practical Lab Work Assignment/Project #1 (for the 1st Tournament).

*

* Integrated Master of Computer Science and Engineering

* Faculty of Science and Technology of New University of Lisbon

*

* Authors:

* @author Ruben Andre Barreiro - r.barreiro@campus.fct.unl.pt

*

*/


import java.security.SecureRandom;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.Iterator;

import java.util.List;
```

```
import java.util.Set;


import gametree.GameNode;

import gametree.GameNodeDoesNotExistException;

import play.exception.InvalidStrategyException;


/**

 * Class responsible for the Hybrid Gradual Strategy, extending Strategy.

 *

 * Description:

 * - A class responsible for a modified version of the Gradual Strategy,

 *   to play the Prisoners' Dilemma Game;

 *

 * - In this version, it was implemented a version of the Gradual Strategy,

 *   that uses a hard concept of forgiveness;

 *

 * - In the normal version, the forgiveness was made by "Calming Down",

 *   always with two consecutive Cooperates;

 *

 * - In this version, the forgiveness it's more hard to get, so,

 *   it's made by a "Calming Down" with two consecutive Cooperates,

 *   since the Opponent Defected less than 8 times, initially;

 *

 * - If the Opponent Defected since the beginning, 8 or more times

 *   until the moment, it will give a harder concept of forgiveness,

 *   that will made only by a "Calming Down" of one Cooperate;
```

```
*
* - But if was detected such or more than 6 consecutive Cooperate actions,
*   will be applied a soft punishment with half of the current opponent's
*   number of Defects, accordingly to the original version of
*   the Gradual Strategy and then, a "calm down" with two "Cooperate" actions;
*
* - If was detected less than 6 consecutive Cooperate actions,
*   will be applied a hard punishment with the equal number of
*   the current opponent's number of Defects, accordingly to the
*   original version of the Gradual Strategy and then, a "calm down" with
*   just one "Cooperate" actions;
*
* - Additionally, in the case of, it wasn't currently making “Defect” actions
*   neither “calming down”, I will respond with a “Cooperate” action since
*   the opponent have more than the double of “Defect”actions made in comparison to
*   the number of “Cooperate” actions made, until the moment. And will respond with
*   a “Defect” action, otherwise. It’s just a way, to try do some “Cooperate” actions,
*   in the case of, I have some kind of “secure” leeway to do some “Cooperate” actions
*   without get some penalty related with a loss of utility points;
*
* - If the probability of keep playing it's lesser than ( 1 / 3 ) % of probability,
*   I will make a "Defect" action, to play for safe;
*
* - In the last round of the current Game, I will make a "Defect" action,
*   to play secure (if it's possible, of course);
*/
```

```
public class HybridGradualWithCooperatesLeewayStrategy extends Strategy {

// Invariants/Constraints:

/**
 * The available "Cooperate" action
 */
private static final String COOPERATE = "Cooperate";

/**
 * The available "Defect" action
 */
private static final String DEFECT = "Defect";

/**
 * The number of consecutive "Cooperate" actions I will do,
 * during a "Calm Down" process
 */
@SuppressWarnings("unused")
private static final int NUM_COOPERATES_NORMAL_CALM_DOWN = 2;

/**
 * The number of consecutive "Cooperate" actions I will do,
 * during a soft "Calm Down" process
 */
private static final int NUM_COOPERATES_SOFT_CALM_DOWN = 1;
```

```
/**
 * The number of consecutive "Cooperate" actions I will do,
 * during a soft "Calm Down" process
 */
private static final int NUM_COOPERATES_HARD_CALM_DOWN = 0;

/**
 * The number of consecutive "Cooperate" actions of Opponent,
 * I will consider, to try to attempt to maximise my gain
 */
private static final int NUM_CONSECUTIVE_COOPERATES_TO_TRY_MAXIMISE_MY_GAIN = 2;

/**
 * The number of consecutive "Cooperate" actions of my Opponent,
 * I will consider, to began to do hard "Calm Down" processes,
 * instead of, the soft ones
 */
private static final int NUM_CONSECUTIVE_COOPERATES_TO_APPLY_HARD_CALM_DOWNS = 6;

/**
 * The initial number of "Defect" actions of my Opponent,
 * I will consider, to began to do hard "Calm Down" processes,
 * instead of, the soft ones
 */
private static final int NUM_INITIAL_DEFECTS_BEFORE_HARD_CALM_DOWN = 8;
```

```
/**
 * The factor number of minimum "leeway" considered to be safe to try to do some "Cooperate" actions
 */
private static final int FACTOR_OF_MINIMUM_LEEWAY_OF_COOPERATES_DIFFERENCE = 2;

/**
 * The beta-value for the minimum probability value to play safe
 */
private static final double SAFE_BETA_PROBABILITY_TO_CONTINUE = ( 1 / 3 );



// Global Instance Variables:

/**
 * The numbers of Defects of the both, my Players
 */
private int[] currentNumMyselfDefects = {0, 0};

/**
 * The numbers of Cooperates of the both, my Players
 */
private int[] currentNumMyselfCooperates = {0, 0};

/**
```

```
 * The numbers of Defects of the both, Opponent's Players
 */
private int[] currentNumOpponentDefects = {0, 0};

/**
 * The numbers of Cooperates of the both, Opponent's Players
 */
private int[] currentNumOpponentCooperates = {0, 0};

/**
 * The numbers of Cooperates of the both, Players' Opponents
 */
private int[] currentNumOpponentConsecutiveCooperates = {0, 0};

/**
 * The numbers of Defects remaining, in a Defecting process,
 * for the both Players
 */
private int[] numDefectsRemaining = {0, 0};

/**
 * The numbers of Cooperates remaining, in a "Calm Down" process,
 * for the both Players
 */
private int[] numCooperatesRemaining = {0, 0};
```

```
/**
 * The boolean values, too keep the information about if,
 * there's some Punishments currently pending or not,
 * for the both Players
 */
private boolean[] pendingPunishments = {false, false};



// Methods/Functions:


/**
 * Returns true if, the Player related to a given number,
 * it's currently Defecting and false, otherwise
 *
 * @param numPlayer the number of the Player,
 *        that it's pretending to be verified
 *        if it's currently Defecting or not
 *
 * @return true if, the Player related to a given number,
 *         it's currently Defecting and false, otherwise
 */
private boolean currentlyDefecting(int numPlayer) {
                return (numDefectsRemaining[ (numPlayer - 1) ] > 0);
}
```

```
/**
 * Returns true if, the Player related to a given number,
 * it's currently "Calming Down" and false, otherwise.
 *
 * @param numPlayer the number of the Player,
 *        that it's pretending to be verified
 *        if it's currently "Calming Down" or not
 *
 * @return true if, the Player related to a given number,
 *         it's currently "Calming Down" and false, otherwise
 */
private boolean currentlyCalmingDown(int numPlayer) {
                return (numCooperatesRemaining[ (numPlayer - 1) ] > 0);
}


/**
 * Returns true if, the Player related to a given number,
 * it's currently Defecting or "Calming Down" and false, otherwise.
 *
 * @param numPlayer the number of the Player,
 *        that it's pretending to be verified
 *        if it's currently Defecting or "Calming Down", or not
 *
 * @return true if, the Player related to a given number,
 *         it's currently Defecting or "Calming Down" and false, otherwise
 */
```

```
private boolean currentlyDefectingOrCalmingDown(int numPlayer) {
		return ( this.currentlyDefecting(numPlayer) || this.currentlyCalmingDown(numPlayer) );
}

/**
 * Starts a set of punishments, by doing, a given number of Defects and then,
 * "Calm Down" (2 Consecutive Cooperates).
 *
 * @param numPlayer the number of the Player, that it's pretended
 *                              to be started a set of punishments
 *
 * @param numDefects the number of Defects to be
 *       associated to this set of punishments
 */
private void startDefectAndCalmDownAsPunishment(int numPlayer, int numDefects) {
		this.numDefectsRemaining[ (numPlayer - 1) ] = numDefects;

		// If the Opponent, until the moment, made less than 10 Defects
		if(numDefects < NUM_INITIAL_DEFECTS_BEFORE_HARD_CALM_DOWN) {
			this.numCooperatesRemaining[ (numPlayer - 1) ] = NUM_COOPERATES_SOFT_CALM_DOWN;
		}

		// If the Opponent, until the moment, made 10 or more Defects
		else if(numDefects >= NUM_INITIAL_DEFECTS_BEFORE_HARD_CALM_DOWN) {

			// Every time that, my Opponent made 10 or more consecutive Cooperates,
```

```
                        // I will reconsider my concept of "Calming Down" by soft it again
                        if( (this.currentNumOpponentConsecutiveCooperates[ (numPlayer - 1) ] >=
NUM_CONSECUTIVE_COOPERATES_TO_APPLY_HARD_CALM_DOWNS) ) {

                                this.numCooperatesRemaining[ (numPlayer - 1) ] = NUM_COOPERATES_SOFT_CALM_DOWN;
                        }

                        // Since that, the Opponent aren't doing 10 or more consecutive Cooperates, at the moment,
                        // I will reconsider my concept of "Calming Down" by the hard way
                        else {
                                this.numCooperatesRemaining[ (numPlayer - 1) ] = NUM_COOPERATES_HARD_CALM_DOWN;
                        }
                }
}

/**
 * Performs a punishment, associated to a given Player.
 *
 * @param numPlayer the number of the Player,
 *        to who be applied this punishment
 */
@SuppressWarnings("unused")
private void defectAndCalmDownAsPunishment(int numPlayer) {

                // I still have some previous consecutive Defects to do
                if(numDefectsRemaining[ (numPlayer - 1) ] > 0) {
```

```
                // I will Defect
                numDefectsRemaining[ (numPlayer - 1) ]--;
        }

        // I'm not currently consecutively Defecting,
        // but probably, I'm currently "Calming Down"
        // (2 consecutive Cooperates)
        else if((numDefectsRemaining[( numPlayer - 1) ] == 0) &&
                        (numCooperatesRemaining[ (numPlayer - 1) ] > 0)) {

                // I'm "Calming Down", so, I will Cooperate
                numCooperatesRemaining[ (numPlayer - 1) ]--;
        }
}

/**
 * Performs a Hard Punishment, associated to a given Player.
 *
 * @param numPlayer the number of the Player,
 *       to who be applied this punishment
 */
private void defectAndCalmDownAsPunishmentHard(int numPlayer) {

        // I still have some previous consecutive Defects to do
        if(numDefectsRemaining[ (numPlayer - 1) ] > 0) {
```

```
                // I will Defect
                numDefectsRemaining[ (numPlayer - 1) ]--;
        }

        // I'm not currently consecutively Defecting,
        // but probably, I'm currently "Calming Down"
        // (2 consecutive Cooperates)
        else if((numDefectsRemaining[( numPlayer - 1) ] == 0) &&
                        (numCooperatesRemaining[ (numPlayer - 1) ] > 0)) {

                // I'm "Calming Down", so, I will Cooperate
                numCooperatesRemaining[ (numPlayer - 1) ]--;
        }
}

/**
 * Performs a Soft Punishment, associated to a given Player.
 *
 * @param numPlayer the number of the Player,
 *      to who be applied this punishment
 */
private void defectAndCalmDownAsPunishmentSoft(int numPlayer) {

        // I still have some previous consecutive Defects to do
        if( (numDefectsRemaining[ (numPlayer - 1) ] > 0 ) &&
```

```
            ((numDefectsRemaining[ (numPlayer - 1) ] % 2) == 0) ) {


                    // I will Defect

                    numDefectsRemaining[ (numPlayer - 1) ] -= 2;

            }


            // I'm not currently consecutively Defecting,

            // but probably, I'm currently "Calming Down"

            // (2 consecutive Cooperates)

            else if((numDefectsRemaining[( numPlayer - 1) ] == 0) &&

                            (numCooperatesRemaining[ (numPlayer - 1) ] > 0)) {


                    // I'm "Calming Down", so, I will Cooperate

                    numCooperatesRemaining[ (numPlayer - 1) ]--;

            }

}


/**

 * The method to perform a possible Cooperate action, knowing that my Opponent Cooperate in the previous round.

 *

 * @param myStrategy the Strategy's object, that's currently being used

 *

 * @param myselfPlayerNum the number of the Myself's Player, that's being analysed the possible move

 *

 * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move

 *
```

```
 * @param possibleMove the possible Move, that's being analysed

 */

private void possibleCooperateActionKnowingThatMyOpponentCooperateInPreviousRound(PlayStrategy myStrategy, int
myselfPlayerNum,

                                                    int opponentPlayerNum, String possibleMove) {


        // The probability to continue playing to the next iteration

        double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();



        // I'm deciding if I Cooperate,

        // knowing that my Opponent Cooperate in the last round


        // I detect a Cooperate action made by my Opponent in the last round

        this.currentNumOpponentCooperates[ (opponentPlayerNum - 1) ]++;


        // I detect one or more than one consecutive Cooperate actions

        this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ]++;



        // The maximum number of iterations remaining for the current Game

        int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();


        // If I'm playing the last round, I will make always a "Defect" action

        if( numMaxIterationsRemaining == 1 ) {


                // I'm playing the last round of the current Game
```

```
                System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
                System.out.println();

                // I'm Cooperating,
                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(0.0));
                System.out.println("Setting " + possibleMove + " with probability of 0.0");
        }

        // Otherwise, I will consider other current aspects of the current Game
        else {

                // If the probability of continue in the next round,
                // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
                if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {

                        // But, I'm not currently consecutively Defecting neither
                        // currently "Calming Down" (2 consecutive Cooperates)
                        if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {

                                // My Opponent was revealing some Cooperating "patterns"
                                // (I will consider, 4 consecutive "Cooperate" actions),
                                // so, maybe, it's a good opportunity to be a little severe
                                // and try to maximise my gain since now on
                                if( (this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] ) >=
NUM_CONSECUTIVE_COOPERATES_TO_TRY_MAXIMISE_MY_GAIN) {
```

```
                                        System.out.println("I'm not currently consecutively Defecting neither currently
\"Calming Down\", \n"
                                                                + "and my Opponent was being
Cooperating, recently!!!");
                                        System.out.println();

                                        // If my Opponent Cooperated such or more than the times that he Defected,
                                        // I will Defect because I have some "leeway" to do Defect actions
                                        if( (this.currentNumOpponentCooperates[ (opponentPlayerNum - 1) ]) >=
                                                                ( (this.currentNumOpponentDefects[
(opponentPlayerNum - 1) ]) /
FACTOR_OF_MINIMUM_LEEWAY_OF_COOPERATES_DIFFERENCE ) ) {

                                                // I'm Cooperating,
                                                // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 1.0]
                                                myStrategy.put(possibleMove, new Double(1.0));
                                                System.out.println("Setting " + possibleMove + " with probability of
1.0");
                                        }

                                        // If my Opponent Cooperated less than the times that he Defected,
                                        // I will Defect because I have some "leeway" to do Defect actions
                                        else {
                                                // I'm Cooperating,
                                                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                                                myStrategy.put(possibleMove, new Double(0.0));
```

```
                                                System.out.println("Setting " + possibleMove + " with probability of
0.0");

                                        }

                                }


                                // My Opponent wasn't being Cooperating recently,

                                // so, may it's a good idea to punish him, just if, perhaps, he have some pending
Punishments

                                else {


                                        // I have some pending Punishments,

                                        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]

                                        if(pendingPunishments[ (opponentPlayerNum - 1) ]) {

                                                this.startDefectAndCalmDownAsPunishment(opponentPlayerNum,
this.currentNumOpponentDefects[ (opponentPlayerNum - 1) ]);

                                                pendingPunishments[ (opponentPlayerNum - 1) ] = false;


                                                System.out.println("I'm not currently consecutively Defecting neither
currently \"Calming Down\", \n"

                                                                + "but I have some pending
Punishments!!!");

                                                System.out.println();


                                                // I'm Defecting,

                                                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]

                                                myStrategy.put(possibleMove, new Double(0.0));

                                                System.out.println("Setting " + possibleMove + " with probability of
0.0");

                                        }
```

```
                                        // I'm not currently consecutively Defecting neither
                                        // currently "Calming Down" (2 consecutive Cooperates),

                                        // So, I will do the same of my opponent in the previous round
                                        // by mimic (Cooperate)
                                        else {
                                                System.out.println("I'm not currently consecutively Defecting neither
currently \"Calming Down\", \n"
                                                                + "so I will mimic and
Cooperate!!!");

                                                // I'm Cooperating,
                                                // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 0.0]
                                                myStrategy.put(possibleMove, new Double(1.0));
                                                this.currentNumMyselfCooperates[ ( myselfPlayerNum - 1) ]++;
                                                System.out.println("Setting " + possibleMove + " with probability of
1.0");
                                        }
                                }
                        }

                        // Possibly, currently consecutively Defecting or
                        // currently "Calming Down" (2 consecutive Cooperates)
                        else {

                                // I still have some previous consecutive Defects to do
                                if(this.currentlyDefecting(opponentPlayerNum)) {
```

```
                        // I'm Cooperating,
                        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(0.0));
                        System.out.println("Setting " + possibleMove + " with probability of 0.0");
                    }

                    // I'm not currently consecutively Defecting,
                    // but probably, I'm currently "Calming Down"
                    // (2 consecutive Cooperates)
                    else if(this.currentlyCalmingDown(opponentPlayerNum)) {

                        // I'm "Calming Down",
                        // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfCooperates[ ( myselfPlayerNum - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                    }
                }
            }

            // Otherwise, if the probability of continue in the next round,
            // it's lesser than 0.3333%, I will play a "Defect" action
            else {

                // I'm Cooperating,
```

```
                                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                                myStrategy.put(possibleMove, new Double(0.0));
                                System.out.println("Setting " + possibleMove + " with probability of 0.0");
                        }
                }
        }

/**
 * The method to perform a possible Cooperate action, knowing that my Opponent Defects in the previous round.
 *
 * @param myStrategy the Strategy's object, that's currently being used
 *
 * @param opponentPlayerNum the number of Myself's Player, that's being analysed the possible move
 *
 * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move
 *
 * @param possibleMove the possible Move, that's being analysed
 */
private void possibleCooperateActionKnowingThatMyOpponentDefectInPreviousRound(PlayStrategy myStrategy, int
myselfPlayerNum,
                                                int opponentPlayerNum, String possibleMove) {

        // The probability to continue playing to the next iteration
        double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();
```

```
// The maximum number of iterations remaining for the current Game
int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();

// If I'm playing the last round, I will make always a "Defect" action
if( numMaxIterationsRemaining == 1 ) {

        // I'm playing the last round of the current Game
        System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
        System.out.println();

        // I'm Cooperating,
        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
        myStrategy.put(possibleMove, new Double(0.0));
        System.out.println("Setting " + possibleMove + " with probability of 0.0");
}

// Otherwise, I will consider other current aspects of the current Game
else {

        // I'm deciding if I Cooperate,
        // knowing that my Opponent Defect in the last round


        // If the probability of continue in the next round,
        // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
        if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {
```

```
// But, I'm not currently consecutively Defecting neither
// currently "Calming Down" (2 consecutive Cooperates)
if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {

        // I will make so many Defects as my Opponent,
        // and after that, I will "Calm Down"
        // (2 consecutive Cooperates)

        // I'm Cooperating,
        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
        myStrategy.put(possibleMove, new Double(0.0));
        System.out.println("Setting " + possibleMove + " with probability of 0.0");
}

// Possibly, currently consecutively Defecting or
// currently "Calming Down" (2 consecutive Cooperates)
else {

        // I still have some previous consecutive Defects to do
        if(this.currentlyDefecting(opponentPlayerNum)) {

                // I'm Cooperating,
                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(0.0));
                System.out.println("Setting " + possibleMove + " with probability of 0.0");
```

```
                    }


                    // I'm not currently consecutively Defecting,
                    // but probably, I'm currently "Calming Down"
                    // (2 consecutive Cooperates)
                    else if(this.currentlyCalmingDown(opponentPlayerNum)) {


                        // I'm "Calming Down",
                        // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfCooperates[ ( myselfPlayerNum - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                    }
                }
            }


            // Otherwise, if the probability of continue in the next round,
            // it's lesser than 0.3333%, I will play a "Defect" action
            else {


                // I'm Cooperating,
                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(0.0));
                System.out.println("Setting " + possibleMove + " with probability of 0.0");
            }
        }
```

```
}

/**
 * The method to perform a possible Defect action, knowing that my Opponent Cooperate in the previous round.
 *
 * @param myStrategy the Strategy's object, that's currently being used
 *
 * @param myselfPlayerNum the number of Myself's Player, that's being analysed the possible move
 *
 * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move
 *
 * @param possibleMove the possible Move, that's being analysed
 */
private void possibleDefectActionKnowingThatMyOpponentCooperateInPreviousRound(PlayStrategy myStrategy, int
myselfPlayerNum,
                                                int opponentPlayerNum, String possibleMove) {

        // The probability to continue playing to the next iteration
        double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();

        // The maximum number of iterations remaining for the current Game
        int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();

        // If I'm playing the last round, I will make always a "Defect" action
        if( numMaxIterationsRemaining == 1 ) {
```

```
                // I'm playing the last round of the current Game
                System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
                System.out.println();

                // I'm Defecting,
                // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(1.0));
                System.out.println("Setting " + possibleMove + " with probability of 1.0");
        }

        // Otherwise, I will consider other current aspects of the current Game
        else {

                // I'm deciding if I Defect,
                // knowing that my Opponent Cooperate in the last round


                // If the probability of continue in the next round,
                // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
                if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {

                        // But, I'm not currently consecutively Defecting neither
                        // currently "Calming Down" (2 consecutive Cooperates)
                        if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {

                                // My Opponent was revealing some Cooperating "patterns"
```

```
                                        // (I will consider, 4 consecutive "Cooperate" actions),
                                        // so, maybe, it's a good opportunity to be a little severe
                                        // and try to maximise my gain since now on
                                        if( (this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] ) >=
NUM_CONSECUTIVE_COOPERATES_TO_TRY_MAXIMISE_MY_GAIN) {

                                            System.out.println("I'm not currently consecutively Defecting neither currently
\"Calming Down\", \n "
                                                    + "and my Opponent was being
Cooperating, recently!!!");
                                            System.out.println();

                                            // If my Opponent Cooperated such or more than the times that he Defected,
                                            // I will Defect because I have some "leeway" to do Defect actions
                                            if( (this.currentNumOpponentCooperates[ (opponentPlayerNum - 1) ]) >=
                                                    ( (this.currentNumOpponentDefects[
(opponentPlayerNum - 1) ]) /
FACTOR_OF_MINIMUM_LEEWAY_OF_COOPERATES_DIFFERENCE ) ) {

                                                // I'm Defecting,
                                                // so, I will Cooperate, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                                                myStrategy.put(possibleMove, new Double(0.0));
                                                System.out.println("Setting " + possibleMove + " with probability of
0.0");
                                            }

                                            // If my Opponent Cooperated less than the times that he Defected,
```

```
                                        // I will Defect because I have some "leeway" to do Defect actions
                                        else {
                                                // I'm Defecting,
                                                // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                                                myStrategy.put(possibleMove, new Double(1.0));
                                                System.out.println("Setting " + possibleMove + " with probability of
1.0");
                                        }
                                }

                                // My Opponent wasn't being Cooperating recently,
                                // so, may it's a good idea to punish him, just if, perhaps, he have some pending
Punishments
                                else {

                                        // I have some pending Punishments
                                        // so, I will Defect, D = 0.0, accordingly to [C = 0.0; D = 1.0]
                                        if(pendingPunishments[ (opponentPlayerNum - 1) ]) {
                                                this.startDefectAndCalmDownAsPunishment(opponentPlayerNum,
this.currentNumOpponentDefects[ (opponentPlayerNum - 1) ]);
                                                pendingPunishments[ (opponentPlayerNum - 1) ] = false;

                                                // I'm Defecting,
                                                // So, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                                                myStrategy.put(possibleMove, new Double(1.0));
                                                this.currentNumMyselfDefects[ ( myselfPlayerNum - 1) ]++;
                                                System.out.println("Setting " + possibleMove + " with probability of
1.0");
```

```
                                                    // Attempts to make a Defect and Calm Down punishment,
                                                    // accordingly to the Gradual strategy
                                                    System.out.println();
                                                    System.out.println("My opponent made " +
this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ]
                                                                                    + " consecutive
Cooperates!!!");

                                                    System.out.println();

                                                    // If my Opponent made 10 or more consecutive Cooperates,
                                                    // I will Defect less and apply a softest Punishment
                                                    if(this.currentNumOpponentConsecutiveCooperates[
(opponentPlayerNum - 1) ] >=
NUM_CONSECUTIVE_COOPERATES_TO_APPLY_HARD_CALM_DOWNS) {

this.defectAndCalmDownAsPunishmentSoft(opponentPlayerNum);
                                                    }

                                                    // If my Opponent made less than 10 consecutive Cooperates,
                                                    // I will Defect less and apply a hardest Punishment
                                                    else {
this.defectAndCalmDownAsPunishmentHard(opponentPlayerNum);
                                                    }
                                            }
```

```
                        // So, I will do the same of my opponent in the previous round
                        // by mimic (Cooperate)
                        else {

                                // So, I will Cooperate, D = 0.0, accordingly to [C = 1.0; D = 0.0]
                                myStrategy.put(possibleMove, new Double(0.0));
                                System.out.println("Setting " + possibleMove + " with probability of
0.0");
                        }
                }
        }

        // Possibly, currently consecutively Defecting or
        // currently "Calming Down" (2 consecutive Cooperates)
        else {

                // I still have some previous consecutive Defects to do
                if(this.currentlyDefecting(opponentPlayerNum)) {

                        // I'm Defecting,
                        // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfDefects[ ( myselfPlayerNum - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                }
```

```
                // I'm not currently consecutively Defecting,
                // but probably, I'm currently "Calming Down"
                // (2 consecutive Cooperates)
                else if(this.currentlyCalmingDown(opponentPlayerNum)) {

                        // I'm "Calming Down",
                        // so, I will Cooperate, D = 0.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(0.0));
                        System.out.println("Setting " + possibleMove + " with probability of 0.0");
                }

                // Attempts to make a Defect and Calm Down punishment,
                // accordingly to the Gradual strategy

                // If my Opponent made 10 or more consecutive Cooperates,
                // I will Defect less and apply a softest Punishment
                if(this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] >= 10) {
                        this.defectAndCalmDownAsPunishmentSoft(opponentPlayerNum);
                }

                // If my Opponent made less than 10 consecutive Cooperates,
                // I will Defect less and apply a hardest Punishment
                else {
                        this.defectAndCalmDownAsPunishmentHard(opponentPlayerNum);
                }
        }
```

```
                }

                // Otherwise, if the probability of continue in the next round,
                // it's lesser than 0.3333%, I will play a "Defect" action
                else {

                        // I'm Defecting,
                        // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                }
        }
}

/**
 * The method to perform a possible Defect action, knowing that my Opponent Defect in the previous round.
 *
 * @param myStrategy the Strategy's object, that's currently being used
 *
 * @param myselfNumPlayer the number of the Myself's Player, that's being analysed the possible move
 *
 * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move
 *
 * @param possibleMove the possible Move, that's being analysed
 */
private void possibleDefectActionKnowingThatMyOpponentDefectInPreviousRound(PlayStrategy myStrategy, int
myselfNumPlayer,
```

```
                                        int opponentPlayerNum, String possibleMove) {

// The probability to continue playing to the next iteration
double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();



// I'm deciding if I Defect,
// knowing that my Opponent Defect in the last round

// I detect a Defect action made by my Opponent in the last round
this.currentNumOpponentDefects[opponentPlayerNum - 1]++;

// I will reset the counter for the consecutive Cooperates made by my Opponent, until the moment
this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] = 0;

// I will pass to have a new pending Punishments
this.pendingPunishments[opponentPlayerNum - 1] = true;

// The maximum number of iterations remaining for the current Game
int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();

// If I'm playing the last round, I will make always a "Defect" action
if( numMaxIterationsRemaining == 1 ) {

        // I'm playing the last round of the current Game
        System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
```

```
                System.out.println();


                // I'm Defecting,
                // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(1.0));
                System.out.println("Setting " + possibleMove + " with probability of 1.0");
        }


        // Otherwise, I will consider other current aspects of the current Game
        else {


                // If the probability of continue in the next round,
                // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
                if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {


                        // So, I will Defect and continue to do it, so,
                        // until I done so many Defects as my Opponent at the moment,
                        // and after, I will "Calm Down" (2 consecutive Cooperates)
                        if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {


                                // I will make so many Defects as my Opponent,
                                // and after that, I will "Calm Down"
                                // (2 consecutive Cooperates)
                                this.startDefectAndCalmDownAsPunishment(opponentPlayerNum,
this.currentNumOpponentDefects[opponentPlayerNum - 1]);


                                // I'm Defecting,
```

```
                // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(1.0));
                this.currentNumMyselfDefects[ ( myselfNumPlayer - 1) ]++;
                System.out.println("Setting " + possibleMove + " with probability of 1.0");
        }

        // Possibly, currently consecutively Defecting or
        // currently "Calming Down" (2 consecutive Cooperates)
        else {

                // I still have some previous consecutive Defects to do
                if(this.currentlyDefecting(opponentPlayerNum)) {

                        // I'm Defecting,
                        // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfDefects[ ( myselfNumPlayer - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                }

                // I'm not currently consecutively Defecting,
                // but probably, I'm currently "Calming Down"
                // (2 consecutive Cooperates)
                else if(this.currentlyCalmingDown(opponentPlayerNum)) {

                        // I'm "Calming Down",
```

```
                        // so, I will Cooperate, D = 0.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(0.0));
                        System.out.println("Setting " + possibleMove + " with probability of 0.0");
                }
        }

        // Attempts to make a Defect and Calm Down punishment,
        // accordingly to the Gradual strategy

        // If my Opponent made 10 or more consecutive Cooperates,
        // I will Defect less and apply a softest Punishment
        if(this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] >= 10) {
                this.defectAndCalmDownAsPunishmentSoft(opponentPlayerNum);
        }

        // If my Opponent made less than 10 consecutive Cooperates,
        // I will Defect less and apply a hardest Punishment
        else {
                this.defectAndCalmDownAsPunishmentHard(opponentPlayerNum);
        }
}

// Otherwise, if the probability of continue in the next round,
// it's lesser than 0.3333%, I will play a "Defect" action
else {
```

```
                                // I'm Defecting,
                                // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                                myStrategy.put(possibleMove, new Double(1.0));
                                System.out.println("Setting " + possibleMove + " with probability of 1.0");
                        }
                }
}

/**
 * Returns the reverse path, by backward, from a given current Game Node.
 *
 * @param current the current Game Node,
 *        from it's being calculated the reverse path, by backward
 *
 * @return the reverse path, by backward, from a given current Game Node
 */
private List<GameNode> getReversePath(GameNode current) {
        try {
                GameNode n = current.getAncestor();

                List<GameNode> l =  getReversePath(n);
                l.add(current);

                return l;
        }
        catch (GameNodeDoesNotExistException e) {
```

```
                List<GameNode> l = new ArrayList<GameNode>();
                l.add(current);

                return l;
        }
}

/**
 * Computes the Game Strategy, that I defined previously. It's here where will be applied all the computation for my strategy.
 *
 * @param listP1 the list of Game Nodes of my Game Tree, as Player no. 1
 *
 * @param listP2 the list of Game Nodes of my Game Tree, as Player no. 2
 *
 * @param myStrategy the computational strategy, that I defined previously and that will be used by me for the current Game
 *
 * @param random a Secure Random object, to calculate random numbers' operations
 *
 * @throws GameNodeDoesNotExistException a GameNodeDoesNotExist to be thrown if
 *         the a certain Game Node don't exist in the current Game
 */
private void cumputeStrategy(List<GameNode> listP1, List<GameNode> listP2,

PlayStrategy myStrategy, SecureRandom random)

                                        throws GameNodeDoesNotExistException {
```

```
Set<String> opponentMoves = new HashSet<String>();

// When I played as Player no. 1, I'm going to check
// what were the moves of my opponent as Player no. 2
for(GameNode n: listP1) {
        if(n.isNature() || n.isRoot()) continue;

        if(n.getAncestor().isPlayer2()) {
                opponentMoves.add(n.getLabel());
        }
}

// When I played as Player no. 2, I'm going to check
// what were the moves of my opponent as Player no. 1
for(GameNode n: listP2) {
        if(n.isNature() || n.isRoot()) continue;

        if(n.getAncestor().isPlayer1()) {
                opponentMoves.add(n.getLabel());
        }
}

System.out.println();

System.out.println("My Opponent's Plays:");
for(String opponentMove : opponentMoves) {
```

```
                System.out.println("- " + opponentMove);
}

System.out.println();

Iterator<String> moves = myStrategy.keyIterator();

// I will analyse all the possible moves
while(moves.hasNext()) {

                // The current possible move
                String currentMove = moves.next();

                System.out.println();
                System.out.println();

                System.err.println("Analysing " + currentMove + " ...");

                System.err.println();

                String[] playStructure = currentMove.split(":");

                int currentOpponentPlayer = Integer.parseInt(playStructure[0]);

                int currentMyselfPlayer = (currentOpponentPlayer == 1) ? 2 : 1;
```

```
                String currentAction = playStructure[2];


                // Currently, analysing a possible Cooperate action,
                // before I decide
                if(currentAction.equalsIgnoreCase(COOPERATE)) {


                        // In this case, my opponent Cooperates in the previous round
                        if(opponentMoves.contains(currentMove)) {
                                System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Cooperates
in the last round!!!");
                                System.out.println();

this.possibleCooperateActionKnowingThatMyOpponentCooperateInPreviousRound(myStrategy, currentMyselfPlayer,
currentOpponentPlayer, currentMove);
                        }


                        // In this case, my opponent Defect in the previous round
                        else {
                                System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Defects in
the last round!!!");

this.possibleCooperateActionKnowingThatMyOpponentDefectInPreviousRound(myStrategy, currentMyselfPlayer,
currentOpponentPlayer, currentMove);
                        }
                }


                // Currently, analysing a possible Defect action,
                // before I decide
                if(currentAction.equalsIgnoreCase(DEFECT)) {
```

```
                        // In this case, my opponent Defect in the previous round

                        if(opponentMoves.contains(currentMove)) {

                                System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Defects in
the last round!!!");

                                System.out.println();

                                this.possibleDefectActionKnowingThatMyOpponentDefectInPreviousRound(myStrategy,
currentMyselfPlayer, currentOpponentPlayer, currentMove);

                        }


                        // In this case, my opponent Cooperates in the previous round

                        else {

                                System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Cooperates
in the last round!!!");

                                System.out.println();


this.possibleDefectActionKnowingThatMyOpponentCooperateInPreviousRound(myStrategy, currentMyselfPlayer,
currentOpponentPlayer, currentMove);

                        }

                }

        }




        System.out.println();




        // Print my Player's statistics, related to the number of "Cooperate" and "Defect" actions

        System.out.println();
```

```
// Print the current number of Cooperates of Myself
System.out.println("Number of Cooperates of Myself as Player no. 1: " + this.currentNumMyselfCooperates[0]);
System.out.println("Number of Cooperates of Myself as Player no. 2: " + this.currentNumMyselfCooperates[1]);

System.out.println();

// Print the current number of Defect of Myself
System.out.println("Number of Defects of Myself as Player no. 1: " + this.currentNumMyselfDefects[0]);
System.out.println("Number of Defects of Myself as Player no. 2: " + this.currentNumMyselfDefects[1]);

System.out.println();



// Print Opponent's Player's statistics, related to the number of "Cooperate" and "Defect" actions
System.out.println();

// Print the current number of Cooperates of the Opponent
System.out.println("Number of Cooperates of the Opponent as Player no. 1: " + this.currentNumOpponentCooperates[0]);
System.out.println("Number of Cooperates of the Opponent as Player no. 2: " + this.currentNumOpponentCooperates[1]);

System.out.println();

// Print the current number of Defect of the Opponent
System.out.println("Number of Defects of the Opponent as Player no. 1: " + this.currentNumOpponentDefects[0]);
System.out.println("Number of Defects of the Opponent as Player no. 2: " + this.currentNumOpponentDefects[1]);
```

```
System.out.println();



// The following piece of code has the goal of checking if there was a portion
// of the game for which we could not infer the moves of the adversary
// (because none of the current Game's plays in the previous round pass through those paths)
Iterator<Integer> validationSetIte = tree.getValidationSet().iterator();
moves = myStrategy.keyIterator();

while(validationSetIte.hasNext()) {
        int possibleMoves = validationSetIte.next().intValue();
        String[] labels = new String[possibleMoves];
        double[] values = new double[possibleMoves];
        double sum = 0;

        for(int i = 0; i < possibleMoves; i++) {
                labels[i] = moves.next();
                values[i] = ((Double) myStrategy.get(labels[i])).doubleValue();
                sum += values[i];
        }

        if(sum != 1) {

                // In the previous current Game's play,
                // I couldn't infer what the adversary played here
```

```
                        // Will be applied a random move on this validation set
                        sum = 0;

                        for(int i = 0; i < values.length - 1; i++) {
                                values[i] = random.nextDouble();
                                while(sum + values[i] >= 1) values[i] = random.nextDouble();
                                sum = sum + values[i];
                        }

                        values[values.length - 1] = ((double) 1) - sum;

                        for(int i = 0; i < possibleMoves; i++) {
                                myStrategy.put(labels[i], values[i]);
                                System.err.println("Unexplored path: Setting " + labels[i] + " with probability of " +
values[i]);
                        }
                }
        }
}


@Override
public void execute() throws InterruptedException {

        SecureRandom random = new SecureRandom();

        while(!this.isTreeKnown()) {
```

```
                System.err.println("Waiting for the Game Tree to become available...");
                Thread.sleep(1000);
        }

        GameNode finalP1 = null;
        GameNode finalP2 = null;

        while(true) {

                PlayStrategy myStrategy = this.getStrategyRequest();



                // The current Game was terminated by an outside event
                if(myStrategy == null) {
                        break;
                }

                boolean playComplete = false;

                while(!playComplete) {
                        if(myStrategy.getFinalP1Node() != -1) {
                                finalP1 = this.tree.getNodeByIndex(myStrategy.getFinalP1Node());
                                if(finalP1 != null)
                                        System.out.println("Final/Terminal node in last round as P1: " + finalP1);
                        }
```

```
if(myStrategy.getFinalP2Node() != -1) {
        finalP2 = this.tree.getNodeByIndex(myStrategy.getFinalP2Node());
        if(finalP2 != null)
            System.out.println("Final/Terminal node in last round as P2: " + finalP2);
}

Iterator<Integer> iterator = tree.getValidationSet().iterator();
Iterator<String> keys = myStrategy.keyIterator();

if(finalP1 == null || finalP2 == null) {

        // This is the first round of the current Game
        while(iterator.hasNext()) {
            double[] moves = new double[iterator.next()];

            double probabilityToContinueInFirstRound =
myStrategy.probabilityForNextIteration();

            // If it's will be played more than one round
            if(myStrategy.getMaximumNumberOfIterations() > 1) {

                // If the probability of continue in the next round,
                // it's greater or equal than 0.3333%, I will play a "Cooperate" action
                if( probabilityToContinueInFirstRound >=
SAFE_BETA_PROBABILITY_TO_CONTINUE ) {
```

```
                    // Here, I will start to Cooperate, as both, Player no. 1 and
Player no. 2
                    moves[0] = 1.0;
                    moves[1] = 0.0;
                }

                // Otherwise, if the probability of continue in the next round,
                // it's lesser than 0.3333%, I will play a "Defect" action
                else {

                    // Here, I will start to Defect, as both, Player no. 1 and Player
no. 2
                    moves[0] = 0.0;
                    moves[1] = 1.0;
                }
            }

            // If it's will be played only one round
            else {

                // Here, I will start to Defect, as both, Player no. 1 and Player no. 2
                moves[0] = 0.0;
                moves[1] = 1.0;
            }

            for(int i = 0; i < moves.length; i++) {
```

```
                                                if(!keys.hasNext()) {
                                                        System.err.println("PANIC: Strategy structure doesn't match
the current Game!!!");
                                                        return;
                                                }

                                                String firstPlay = keys.next();

                                                System.out.println();
                                                System.out.println("My First Play - " + firstPlay + " with probability of "
+ moves[i]);

                                                myStrategy.put(firstPlay, moves[i]);
                                        }
                                }
                        }
                        else {

                                // Let's, now, play the Gradual Strategy (at least what we can infer)
                                List<GameNode> listP1 = getReversePath(finalP1);
                                List<GameNode> listP2 = getReversePath(finalP2);

                                try {
                                        cumputeStrategy(listP1, listP2, myStrategy, random);
                                }
                                catch (GameNodeDoesNotExistException gameNodeDoesNotExistException) {
                                        System.err.println("PANIC: Strategy structure doesn't match the current
Game!!!");
```

```
                        }
                    }

                    try {
                        this.provideStrategy(myStrategy);
                        playComplete = true;
                    }
                    catch(InvalidStrategyException invalidStrategyException) {
                        System.err.println("Invalid Strategy: " + invalidStrategyException.getMessage());;
                        invalidStrategyException.printStackTrace(System.err);
                    }
                }
            }
        }
    }
```

Annexes (Code of Java Implementations for the used Strategies) [2]

Java Class for the Hybrid Gradual With Cooperates Leeway Strategy:

```
package play;


/**
*
* ~~~ Prisoners' Dilemma ~~~
*
* Theory of Computational Games
*
* Practical Lab Work Assignment/Project #1 (for the 1st Tournament).
*
* Integrated Master of Computer Science and Engineering
* Faculty of Science and Technology of New University of Lisbon
*
* Authors:
* @author Ruben Andre Barreiro - r.barreiro@campus.fct.unl.pt
*
*/


import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
```

```
import java.util.Set;


import gametree.GameNode;

import gametree.GameNodeDoesNotExistException;

import play.exception.InvalidStrategyException;


/**

 * Class responsible for the Hybrid Gradual Strategy, extending Strategy.

 *

 * Description:

 * - A class responsible for a modified version of the Gradual Strategy,

 *   to play the Prisoners' Dilemma Game;

 *

 * - In this version, it was implemented a version of the Gradual Strategy,

 *   that uses a hard concept of forgiveness;

 *

 * - In the normal version, the forgiveness was made by "Calming Down",

 *   always with two consecutive Cooperates;

 *

 * - In this version, the forgiveness it's more hard to get, so,

 *   it's made by a "Calming Down" with two consecutive Cooperates,

 *   since the Opponent Defected less than 8 times, initially;

 *

 * - If the Opponent Defected since the beginning, 8 or more times

 *   until the moment, it will give a harder concept of forgiveness,

 *   that will made only by a "Calming Down" of one Cooperate;
```

```
*
* - But if was detected such or more than 6 consecutive Cooperate actions,
*   will be applied a soft punishment with half of the current opponent's
*   number of Defects, accordingly to the original version of
*   the Gradual Strategy and then, a "calm down" with two "Cooperate" actions;
*
* - If was detected less than 6 consecutive Cooperate actions,
*   will be applied a hard punishment with the equal number of
*   the current opponent's number of Defects, accordingly to the
*   original version of the Gradual Strategy and then, a "calm down" with
*   just one "Cooperate" actions;
*
* - Additionally, in the case of, it wasn't currently making “Defect” actions
*   neither “calming down”, I will respond with a “Cooperate” action since
*   the opponent have more than the double of “Defect”actions made in comparison to
*   the number of “Cooperate” actions made, until the moment. And will respond with
*   a “Defect” action, otherwise. It’s just a way, to try do some “Cooperate” actions,
*   in the case of, I have some kind of “secure” leeway to do some “Cooperate” actions
*   without get some penalty related with a loss of utility points;
*
* - If the probability of keep playing it's lesser than ( 1 / 3 ) % of probability,
*   I will make a "Defect" action, to play for safe;
*
* - In the last round of the current Game, I will make a "Defect" action,
*   to play secure (if it's possible, of course);
*/
```

```
public class HybridGradualWithCooperatesLeewayStrategy extends Strategy {

// Invariants/Constraints:

/**
 * The available "Cooperate" action
 */
private static final String COOPERATE = "Cooperate";

/**
 * The available "Defect" action
 */
private static final String DEFECT = "Defect";

/**
 * The number of consecutive "Cooperate" actions I will do,
 * during a "Calm Down" process
 */
@SuppressWarnings("unused")
private static final int NUM_COOPERATES_NORMAL_CALM_DOWN = 2;

/**
 * The number of consecutive "Cooperate" actions I will do,
 * during a soft "Calm Down" process
 */
private static final int NUM_COOPERATES_SOFT_CALM_DOWN = 1;
```

```
/**
 * The number of consecutive "Cooperate" actions I will do,
 * during a soft "Calm Down" process
 */
private static final int NUM_COOPERATES_HARD_CALM_DOWN = 0;

/**
 * The number of consecutive "Cooperate" actions of Opponent,
 * I will consider, to try to attempt to maximise my gain
 */
private static final int NUM_CONSECUTIVE_COOPERATES_TO_TRY_MAXIMISE_MY_GAIN = 2;

/**
 * The number of consecutive "Cooperate" actions of my Opponent,
 * I will consider, to began to do hard "Calm Down" processes,
 * instead of, the soft ones
 */
private static final int NUM_CONSECUTIVE_COOPERATES_TO_APPLY_HARD_CALM_DOWNS = 6;

/**
 * The initial number of "Defect" actions of my Opponent,
 * I will consider, to began to do hard "Calm Down" processes,
 * instead of, the soft ones
 */
private static final int NUM_INITIAL_DEFECTS_BEFORE_HARD_CALM_DOWN = 8;
```

```
/**
 * The factor number of minimum "leeway" considered to be safe to try to do some "Cooperate" actions
 */
private static final int FACTOR_OF_MINIMUM_LEEWAY_OF_COOPERATES_DIFFERENCE = 2;

/**
 * The beta-value for the minimum probability value to play safe
 */
private static final double SAFE_BETA_PROBABILITY_TO_CONTINUE = ( 1 / 3 );



// Global Instance Variables:

/**
 * The numbers of Defects of the both, my Players
 */
private int[] currentNumMyselfDefects = {0, 0};

/**
 * The numbers of Cooperates of the both, my Players
 */
private int[] currentNumMyselfCooperates = {0, 0};

/**
```

```
 * The numbers of Defects of the both, Opponent's Players
 */
private int[] currentNumOpponentDefects = {0, 0};

/**
 * The numbers of Cooperates of the both, Opponent's Players
 */
private int[] currentNumOpponentCooperates = {0, 0};

/**
 * The numbers of Cooperates of the both, Players' Opponents
 */
private int[] currentNumOpponentConsecutiveCooperates = {0, 0};

/**
 * The numbers of Defects remaining, in a Defecting process,
 * for the both Players
 */
private int[] numDefectsRemaining = {0, 0};

/**
 * The numbers of Cooperates remaining, in a "Calm Down" process,
 * for the both Players
 */
private int[] numCooperatesRemaining = {0, 0};
```

```
/**
 * The boolean values, too keep the information about if,
 * there's some Punishments currently pending or not,
 * for the both Players
 */
private boolean[] pendingPunishments = {false, false};



// Methods/Functions:


/**
 * Returns true if, the Player related to a given number,
 * it's currently Defecting and false, otherwise
 *
 * @param numPlayer the number of the Player,
 *        that it's pretending to be verified
 *        if it's currently Defecting or not
 *
 * @return true if, the Player related to a given number,
 *         it's currently Defecting and false, otherwise
 */
private boolean currentlyDefecting(int numPlayer) {
        return (numDefectsRemaining[ (numPlayer - 1) ] > 0);
}
```

```
/**
 * Returns true if, the Player related to a given number,
 * it's currently "Calming Down" and false, otherwise.
 *
 * @param numPlayer the number of the Player,
 *        that it's pretending to be verified
 *        if it's currently "Calming Down" or not
 *
 * @return true if, the Player related to a given number,
 *         it's currently "Calming Down" and false, otherwise
 */
private boolean currentlyCalmingDown(int numPlayer) {
        return (numCooperatesRemaining[ (numPlayer - 1) ] > 0);
}

/**
 * Returns true if, the Player related to a given number,
 * it's currently Defecting or "Calming Down" and false, otherwise.
 *
 * @param numPlayer the number of the Player,
 *        that it's pretending to be verified
 *        if it's currently Defecting or "Calming Down", or not
 *
 * @return true if, the Player related to a given number,
 *         it's currently Defecting or "Calming Down" and false, otherwise
 */
```

```
private boolean currentlyDefectingOrCalmingDown(int numPlayer) {
        return ( this.currentlyDefecting(numPlayer) || this.currentlyCalmingDown(numPlayer) );
}

/**
 * Starts a set of punishments, by doing, a given number of Defects and then,
 * "Calm Down" (2 Consecutive Cooperates).
 *
 * @param numPlayer the number of the Player, that it's pretended
 *                  to be started a set of punishments
 *
 * @param numDefects the number of Defects to be
 *       associated to this set of punishments
 */
private void startDefectAndCalmDownAsPunishment(int numPlayer, int numDefects) {
        this.numDefectsRemaining[ (numPlayer - 1) ] = numDefects;

        // If the Opponent, until the moment, made less than 10 Defects
        if(numDefects < NUM_INITIAL_DEFECTS_BEFORE_HARD_CALM_DOWN) {
                this.numCooperatesRemaining[ (numPlayer - 1) ] = NUM_COOPERATES_SOFT_CALM_DOWN;
        }

        // If the Opponent, until the moment, made 10 or more Defects
        else if(numDefects >= NUM_INITIAL_DEFECTS_BEFORE_HARD_CALM_DOWN) {

                // Every time that, my Opponent made 10 or more consecutive Cooperates,
```

```
                    // I will reconsider my concept of "Calming Down" by soft it again
                    if( (this.currentNumOpponentConsecutiveCooperates[ (numPlayer - 1) ] >=
NUM_CONSECUTIVE_COOPERATES_TO_APPLY_HARD_CALM_DOWNS) ) {

                        this.numCooperatesRemaining[ (numPlayer - 1) ] = NUM_COOPERATES_SOFT_CALM_DOWN;
                    }

                    // Since that, the Opponent aren't doing 10 or more consecutive Cooperates, at the moment,
                    // I will reconsider my concept of "Calming Down" by the hard way
                    else {
                        this.numCooperatesRemaining[ (numPlayer - 1) ] = NUM_COOPERATES_HARD_CALM_DOWN;
                    }
            }
}

/**
 * Performs a punishment, associated to a given Player.
 *
 * @param numPlayer the number of the Player,
 *        to who be applied this punishment
 */
@SuppressWarnings("unused")
private void defectAndCalmDownAsPunishment(int numPlayer) {

            // I still have some previous consecutive Defects to do
            if(numDefectsRemaining[ (numPlayer - 1) ] > 0) {
```

```
                // I will Defect
                numDefectsRemaining[ (numPlayer - 1) ]--;
        }

        // I'm not currently consecutively Defecting,
        // but probably, I'm currently "Calming Down"
        // (2 consecutive Cooperates)
        else if((numDefectsRemaining[( numPlayer - 1) ] == 0) &&
                        (numCooperatesRemaining[ (numPlayer - 1) ] > 0)) {

                // I'm "Calming Down", so, I will Cooperate
                numCooperatesRemaining[ (numPlayer - 1) ]--;
        }
}

/**
 * Performs a Hard Punishment, associated to a given Player.
 *
 * @param numPlayer the number of the Player,
 *        to who be applied this punishment
 */
private void defectAndCalmDownAsPunishmentHard(int numPlayer) {

        // I still have some previous consecutive Defects to do
        if(numDefectsRemaining[ (numPlayer - 1) ] > 0) {
```

```
                // I will Defect
                numDefectsRemaining[ (numPlayer - 1) ]--;
        }

        // I'm not currently consecutively Defecting,
        // but probably, I'm currently "Calming Down"
        // (2 consecutive Cooperates)
        else if((numDefectsRemaining[( numPlayer - 1) ] == 0) &&
                        (numCooperatesRemaining[ (numPlayer - 1) ] > 0)) {

                // I'm "Calming Down", so, I will Cooperate
                numCooperatesRemaining[ (numPlayer - 1) ]--;
        }
}

/**
 * Performs a Soft Punishment, associated to a given Player.
 *
 * @param numPlayer the number of the Player,
 *        to who be applied this punishment
 */
private void defectAndCalmDownAsPunishmentSoft(int numPlayer) {

        // I still have some previous consecutive Defects to do
        if( (numDefectsRemaining[ (numPlayer - 1) ] > 0 ) &&
```

```
    ((numDefectsRemaining[ (numPlayer - 1) ] % 2) == 0) ) {

            // I will Defect
            numDefectsRemaining[ (numPlayer - 1) ] -= 2;
        }

        // I'm not currently consecutively Defecting,
        // but probably, I'm currently "Calming Down"
        // (2 consecutive Cooperates)
        else if((numDefectsRemaining[( numPlayer - 1) ] == 0) &&
                (numCooperatesRemaining[ (numPlayer - 1) ] > 0)) {

            // I'm "Calming Down", so, I will Cooperate
            numCooperatesRemaining[ (numPlayer - 1) ]--;
        }
    }

    /**
     * The method to perform a possible Cooperate action, knowing that my Opponent Cooperate in the previous round.
     *
     * @param myStrategy the Strategy's object, that's currently being used
     *
     * @param myselfPlayerNum the number of the Myself's Player, that's being analysed the possible move
     *
     * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move
     *
```

```
 * @param possibleMove the possible Move, that's being analysed

 */

private void possibleCooperateActionKnowingThatMyOpponentCooperateInPreviousRound(PlayStrategy myStrategy, int myselfPlayerNum,

                                                                                   int opponentPlayerNum, String possibleMove) {


        // The probability to continue playing to the next iteration

        double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();



        // I'm deciding if I Cooperate,

        // knowing that my Opponent Cooperate in the last round


        // I detect a Cooperate action made by my Opponent in the last round

        this.currentNumOpponentCooperates[ (opponentPlayerNum - 1) ]++;


        // I detect one or more than one consecutive Cooperate actions

        this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ]++;



        // The maximum number of iterations remaining for the current Game

        int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();


        // If I'm playing the last round, I will make always a "Defect" action

        if( numMaxIterationsRemaining == 1 ) {


            // I'm playing the last round of the current Game
```

```
                System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
                System.out.println();

                // I'm Cooperating,
                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(0.0));
                System.out.println("Setting " + possibleMove + " with probability of 0.0");
        }

        // Otherwise, I will consider other current aspects of the current Game
        else {

                // If the probability of continue in the next round,
                // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
                if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {

                        // But, I'm not currently consecutively Defecting neither
                        // currently "Calming Down" (2 consecutive Cooperates)
                        if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {

                                // My Opponent was revealing some Cooperating "patterns"
                                // (I will consider, 4 consecutive "Cooperate" actions),
                                // so, maybe, it's a good opportunity to be a little severe
                                // and try to maximise my gain since now on
                                if( (this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] ) >=
NUM_CONSECUTIVE_COOPERATES_TO_TRY_MAXIMISE_MY_GAIN) {
```

```
                                System.out.println("I'm not currently consecutively Defecting neither currently
\"Calming Down\", \n"
                                                + "and my Opponent was being
Cooperating, recently!!!");
                                System.out.println();

                                // If my Opponent Cooperated such or more than the times that he Defected,
                                // I will Defect because I have some "leeway" to do Defect actions
                                if( (this.currentNumOpponentCooperates[ (opponentPlayerNum - 1) ]) >=
                                                ( (this.currentNumOpponentDefects[
(opponentPlayerNum - 1) ]) /
FACTOR_OF_MINIMUM_LEEWAY_OF_COOPERATES_DIFFERENCE ) ) {

                                        // I'm Cooperating,
                                        // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 1.0]
                                        myStrategy.put(possibleMove, new Double(1.0));
                                        System.out.println("Setting " + possibleMove + " with probability of
1.0");
                                }

                                // If my Opponent Cooperated less than the times that he Defected,
                                // I will Defect because I have some "leeway" to do Defect actions
                                else {
                                        // I'm Cooperating,
                                        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                                        myStrategy.put(possibleMove, new Double(0.0));
```

```
                                        System.out.println("Setting " + possibleMove + " with probability of
0.0");
                                    }
                                }

                                // My Opponent wasn't being Cooperating recently,
                                // so, may it's a good idea to punish him, just if, perhaps, he have some pending
Punishments
                                else {

                                    // I have some pending Punishments,
                                    // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                                    if(pendingPunishments[ (opponentPlayerNum - 1) ]) {
                                        this.startDefectAndCalmDownAsPunishment(opponentPlayerNum,
this.currentNumOpponentDefects[ (opponentPlayerNum - 1) ]);
                                        pendingPunishments[ (opponentPlayerNum - 1) ] = false;

                                        System.out.println("I'm not currently consecutively Defecting neither
currently \"Calming Down\", \n"
                                                + "but I have some pending
Punishments!!!");
                                        System.out.println();

                                        // I'm Defecting,
                                        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                                        myStrategy.put(possibleMove, new Double(0.0));
                                        System.out.println("Setting " + possibleMove + " with probability of
0.0");
                                    }
```

```
                    // I'm not currently consecutively Defecting neither
                    // currently "Calming Down" (2 consecutive Cooperates),

                    // So, I will do the same of my opponent in the previous round
                    // by mimic (Cooperate)
                    else {
                        System.out.println("I'm not currently consecutively Defecting neither currently \"Calming Down\", \n"
                                + "so I will mimic and Cooperate!!!");

                        // I'm Cooperating,
                        // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfCooperates[ ( myselfPlayerNum - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                    }
                }
            }

            // Possibly, currently consecutively Defecting or
            // currently "Calming Down" (2 consecutive Cooperates)
            else {

                // I still have some previous consecutive Defects to do
                if(this.currentlyDefecting(opponentPlayerNum)) {
```

```
                    // I'm Cooperating,
                    // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                    myStrategy.put(possibleMove, new Double(0.0));
                    System.out.println("Setting " + possibleMove + " with probability of 0.0");
                }

                // I'm not currently consecutively Defecting,
                // but probably, I'm currently "Calming Down"
                // (2 consecutive Cooperates)
                else if(this.currentlyCalmingDown(opponentPlayerNum)) {

                    // I'm "Calming Down",
                    // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 0.0]
                    myStrategy.put(possibleMove, new Double(1.0));
                    this.currentNumMyselfCooperates[ ( myselfPlayerNum - 1) ]++;
                    System.out.println("Setting " + possibleMove + " with probability of 1.0");
                }
            }
        }

        // Otherwise, if the probability of continue in the next round,
        // it's lesser than 0.3333%, I will play a "Defect" action
        else {

            // I'm Cooperating,
```

```
                    // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                    myStrategy.put(possibleMove, new Double(0.0));
                    System.out.println("Setting " + possibleMove + " with probability of 0.0");
                }
            }
        }

    /**
     * The method to perform a possible Cooperate action, knowing that my Opponent Defects in the previous round.
     *
     * @param myStrategy the Strategy's object, that's currently being used
     *
     * @param opponentPlayerNum the number of Myself's Player, that's being analysed the possible move
     *
     * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move
     *
     * @param possibleMove the possible Move, that's being analysed
     */
    private void possibleCooperateActionKnowingThatMyOpponentDefectInPreviousRound(PlayStrategy myStrategy, int myselfPlayerNum,
                                                                                   int opponentPlayerNum, String possibleMove) {

        // The probability to continue playing to the next iteration
        double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();
```

```
// The maximum number of iterations remaining for the current Game
int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();

// If I'm playing the last round, I will make always a "Defect" action
if( numMaxIterationsRemaining == 1 ) {

        // I'm playing the last round of the current Game
        System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
        System.out.println();

        // I'm Cooperating,
        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
        myStrategy.put(possibleMove, new Double(0.0));
        System.out.println("Setting " + possibleMove + " with probability of 0.0");
}

// Otherwise, I will consider other current aspects of the current Game
else {

        // I'm deciding if I Cooperate,
        // knowing that my Opponent Defect in the last round


        // If the probability of continue in the next round,
        // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
        if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {
```

```
// But, I'm not currently consecutively Defecting neither
// currently "Calming Down" (2 consecutive Cooperates)
if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {

        // I will make so many Defects as my Opponent,
        // and after that, I will "Calm Down"
        // (2 consecutive Cooperates)

        // I'm Cooperating,
        // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
        myStrategy.put(possibleMove, new Double(0.0));
        System.out.println("Setting " + possibleMove + " with probability of 0.0");
}

// Possibly, currently consecutively Defecting or
// currently "Calming Down" (2 consecutive Cooperates)
else {

        // I still have some previous consecutive Defects to do
        if(this.currentlyDefecting(opponentPlayerNum)) {

                // I'm Cooperating,
                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(0.0));
                System.out.println("Setting " + possibleMove + " with probability of 0.0");
```

```
                    }


                    // I'm not currently consecutively Defecting,
                    // but probably, I'm currently "Calming Down"
                    // (2 consecutive Cooperates)
                    else if(this.currentlyCalmingDown(opponentPlayerNum)) {


                        // I'm "Calming Down",
                        // so, I will Cooperate, C = 1.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfCooperates[ ( myselfPlayerNum - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                    }
                }
            }


            // Otherwise, if the probability of continue in the next round,
            // it's lesser than 0.3333%, I will play a "Defect" action
            else {


                // I'm Cooperating,
                // so, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(0.0));
                System.out.println("Setting " + possibleMove + " with probability of 0.0");
            }
        }
```

```
}

/**
 * The method to perform a possible Defect action, knowing that my Opponent Cooperate in the previous round.
 *
 * @param myStrategy the Strategy's object, that's currently being used
 *
 * @param myselfPlayerNum the number of Myself's Player, that's being analysed the possible move
 *
 * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move
 *
 * @param possibleMove the possible Move, that's being analysed
 */
private void possibleDefectActionKnowingThatMyOpponentCooperateInPreviousRound(PlayStrategy myStrategy, int myselfPlayerNum,
                                                                               int opponentPlayerNum, String possibleMove) {

    // The probability to continue playing to the next iteration
    double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();

    // The maximum number of iterations remaining for the current Game
    int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();

    // If I'm playing the last round, I will make always a "Defect" action
    if( numMaxIterationsRemaining == 1 ) {
```

```
            // I'm playing the last round of the current Game
            System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
            System.out.println();

            // I'm Defecting,
            // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
            myStrategy.put(possibleMove, new Double(1.0));
            System.out.println("Setting " + possibleMove + " with probability of 1.0");
        }

        // Otherwise, I will consider other current aspects of the current Game
        else {

            // I'm deciding if I Defect,
            // knowing that my Opponent Cooperate in the last round


            // If the probability of continue in the next round,
            // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
            if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {

                // But, I'm not currently consecutively Defecting neither
                // currently "Calming Down" (2 consecutive Cooperates)
                if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {

                    // My Opponent was revealing some Cooperating "patterns"
```

```
                    // (I will consider, 4 consecutive "Cooperate" actions),
                    // so, maybe, it's a good opportunity to be a little severe
                    // and try to maximise my gain since now on
                    if( (this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] ) >=
NUM_CONSECUTIVE_COOPERATES_TO_TRY_MAXIMISE_MY_GAIN) {

                        System.out.println("I'm not currently consecutively Defecting neither currently
\"Calming Down\", \n "
                                + "and my Opponent was being
Cooperating, recently!!!");
                        System.out.println();

                        // If my Opponent Cooperated such or more than the times that he Defected,
                        // I will Defect because I have some "leeway" to do Defect actions
                        if( (this.currentNumOpponentCooperates[ (opponentPlayerNum - 1) ]) >=
                                ( (this.currentNumOpponentDefects[
(opponentPlayerNum - 1) ]) /
FACTOR_OF_MINIMUM_LEEWAY_OF_COOPERATES_DIFFERENCE ) ) {

                            // I'm Defecting,
                            // so, I will Cooperate, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                            myStrategy.put(possibleMove, new Double(0.0));
                            System.out.println("Setting " + possibleMove + " with probability of
0.0");
                        }

                        // If my Opponent Cooperated less than the times that he Defected,
```

```
                    // I will Defect because I have some "leeway" to do Defect actions
                    else {
                        // I'm Defecting,
                        // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        System.out.println("Setting " + possibleMove + " with probability of
1.0");
                    }
                }

                // My Opponent wasn't being Cooperating recently,
                // so, may it's a good idea to punish him, just if, perhaps, he have some pending
Punishments
                else {

                    // I have some pending Punishments
                    // so, I will Defect, D = 0.0, accordingly to [C = 0.0; D = 1.0]
                    if(pendingPunishments[ (opponentPlayerNum - 1) ]) {
                        this.startDefectAndCalmDownAsPunishment(opponentPlayerNum,
this.currentNumOpponentDefects[ (opponentPlayerNum - 1) ]);
                        pendingPunishments[ (opponentPlayerNum - 1) ] = false;

                        // I'm Defecting,
                        // So, I will Defect, C = 0.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfDefects[ ( myselfPlayerNum - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of
1.0");
```

```
                                        // Attempts to make a Defect and Calm Down punishment,
                                        // accordingly to the Gradual strategy
                                        System.out.println();
                                        System.out.println("My opponent made " +
this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ]
                                                + " consecutive
Cooperates!!!");

                                        System.out.println();

                                        // If my Opponent made 10 or more consecutive Cooperates,
                                        // I will Defect less and apply a softest Punishment
                                        if(this.currentNumOpponentConsecutiveCooperates[
(opponentPlayerNum - 1) ] >=
NUM_CONSECUTIVE_COOPERATES_TO_APPLY_HARD_CALM_DOWNS) {

this.defectAndCalmDownAsPunishmentSoft(opponentPlayerNum);
                                        }

                                        // If my Opponent made less than 10 consecutive Cooperates,
                                        // I will Defect less and apply a hardest Punishment
                                        else {

this.defectAndCalmDownAsPunishmentHard(opponentPlayerNum);
                                        }
                                }
```

```
                    // So, I will do the same of my opponent in the previous round
                    // by mimic (Cooperate)
                    else {

                        // So, I will Cooperate, D = 0.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(0.0));
                        System.out.println("Setting " + possibleMove + " with probability of
0.0");

                    }
                }
            }

            // Possibly, currently consecutively Defecting or
            // currently "Calming Down" (2 consecutive Cooperates)
            else {

                // I still have some previous consecutive Defects to do
                if(this.currentlyDefecting(opponentPlayerNum)) {

                    // I'm Defecting,
                    // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                    myStrategy.put(possibleMove, new Double(1.0));
                    this.currentNumMyselfDefects[ ( myselfPlayerNum - 1) ]++;
                    System.out.println("Setting " + possibleMove + " with probability of 1.0");

                }
```

```
                // I'm not currently consecutively Defecting,
                // but probably, I'm currently "Calming Down"
                // (2 consecutive Cooperates)
                else if(this.currentlyCalmingDown(opponentPlayerNum)) {

                        // I'm "Calming Down",
                        // so, I will Cooperate, D = 0.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(0.0));
                        System.out.println("Setting " + possibleMove + " with probability of 0.0");
                }

                // Attempts to make a Defect and Calm Down punishment,
                // accordingly to the Gradual strategy

                // If my Opponent made 10 or more consecutive Cooperates,
                // I will Defect less and apply a softest Punishment
                if(this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] >= 10) {
                        this.defectAndCalmDownAsPunishmentSoft(opponentPlayerNum);
                }

                // If my Opponent made less than 10 consecutive Cooperates,
                // I will Defect less and apply a hardest Punishment
                else {
                        this.defectAndCalmDownAsPunishmentHard(opponentPlayerNum);
                }
        }
```

```
                }

                // Otherwise, if the probability of continue in the next round,
                // it's lesser than 0.3333%, I will play a "Defect" action
                else {

                        // I'm Defecting,
                        // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                }
        }
}

/**
 * The method to perform a possible Defect action, knowing that my Opponent Defect in the previous round.
 *
 * @param myStrategy the Strategy's object, that's currently being used
 *
 * @param myselfNumPlayer the number of the Myself's Player, that's being analysed the possible move
 *
 * @param opponentPlayerNum the number of the Opponent's Player, that's being analysed the possible move
 *
 * @param possibleMove the possible Move, that's being analysed
 */
private void possibleDefectActionKnowingThatMyOpponentDefectInPreviousRound(PlayStrategy myStrategy, int
myselfNumPlayer,
```

```
                                        int opponentPlayerNum, String possibleMove) {

// The probability to continue playing to the next iteration
double probabilityToContinueToTheNextIteration = myStrategy.probabilityForNextIteration();



// I'm deciding if I Defect,
// knowing that my Opponent Defect in the last round

// I detect a Defect action made by my Opponent in the last round
this.currentNumOpponentDefects[opponentPlayerNum - 1]++;

// I will reset the counter for the consecutive Cooperates made by my Opponent, until the moment
this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] = 0;

// I will pass to have a new pending Punishments
this.pendingPunishments[opponentPlayerNum - 1] = true;

// The maximum number of iterations remaining for the current Game
int numMaxIterationsRemaining = myStrategy.getMaximumNumberOfIterations();

// If I'm playing the last round, I will make always a "Defect" action
if( numMaxIterationsRemaining == 1 ) {

        // I'm playing the last round of the current Game
        System.out.println("I'm currently playing the last round,\nso I will play safe for me and I will Defect!!!");
```

```
                System.out.println();

                // I'm Defecting,
                // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(1.0));
                System.out.println("Setting " + possibleMove + " with probability of 1.0");
        }

        // Otherwise, I will consider other current aspects of the current Game
        else {

                // If the probability of continue in the next round,
                // it's greater or equal than 0.3333%, I will consider the current aspects of the Game
                if( probabilityToContinueToTheNextIteration >= SAFE_BETA_PROBABILITY_TO_CONTINUE ) {

                        // So, I will Defect and continue to do it, so,
                        // until I done so many Defects as my Opponent at the moment,
                        // and after, I will "Calm Down" (2 consecutive Cooperates)
                        if(!this.currentlyDefectingOrCalmingDown(opponentPlayerNum)) {

                                // I will make so many Defects as my Opponent,
                                // and after that, I will "Calm Down"
                                // (2 consecutive Cooperates)
                                this.startDefectAndCalmDownAsPunishment(opponentPlayerNum,
this.currentNumOpponentDefects[opponentPlayerNum - 1]);

                                // I'm Defecting,
```

```
                // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                myStrategy.put(possibleMove, new Double(1.0));
                this.currentNumMyselfDefects[ ( myselfNumPlayer - 1) ]++;
                System.out.println("Setting " + possibleMove + " with probability of 1.0");
        }

        // Possibly, currently consecutively Defecting or
        // currently "Calming Down" (2 consecutive Cooperates)
        else {

                // I still have some previous consecutive Defects to do
                if(this.currentlyDefecting(opponentPlayerNum)) {

                        // I'm Defecting,
                        // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                        myStrategy.put(possibleMove, new Double(1.0));
                        this.currentNumMyselfDefects[ ( myselfNumPlayer - 1) ]++;
                        System.out.println("Setting " + possibleMove + " with probability of 1.0");
                }

                // I'm not currently consecutively Defecting,
                // but probably, I'm currently "Calming Down"
                // (2 consecutive Cooperates)
                else if(this.currentlyCalmingDown(opponentPlayerNum)) {

                        // I'm "Calming Down",
```

```
                        // so, I will Cooperate, D = 0.0, accordingly to [C = 1.0; D = 0.0]
                        myStrategy.put(possibleMove, new Double(0.0));
                        System.out.println("Setting " + possibleMove + " with probability of 0.0");
                }
        }

        // Attempts to make a Defect and Calm Down punishment,
        // accordingly to the Gradual strategy

        // If my Opponent made 10 or more consecutive Cooperates,
        // I will Defect less and apply a softest Punishment
        if(this.currentNumOpponentConsecutiveCooperates[ (opponentPlayerNum - 1) ] >= 10) {
                this.defectAndCalmDownAsPunishmentSoft(opponentPlayerNum);
        }

        // If my Opponent made less than 10 consecutive Cooperates,
        // I will Defect less and apply a hardest Punishment
        else {
                this.defectAndCalmDownAsPunishmentHard(opponentPlayerNum);
        }
}

// Otherwise, if the probability of continue in the next round,
// it's lesser than 0.3333%, I will play a "Defect" action
else {
```

```
                    // I'm Defecting,
                    // so, I will Defect, D = 1.0, accordingly to [C = 0.0; D = 1.0]
                    myStrategy.put(possibleMove, new Double(1.0));
                    System.out.println("Setting " + possibleMove + " with probability of 1.0");
                }
            }
        }

    /**
     * Returns the reverse path, by backward, from a given current Game Node.
     *
     * @param current the current Game Node,
     *        from it's being calculated the reverse path, by backward
     *
     * @return the reverse path, by backward, from a given current Game Node
     */
    private List<GameNode> getReversePath(GameNode current) {
        try {
            GameNode n = current.getAncestor();

            List<GameNode> l =  getReversePath(n);
            l.add(current);

            return l;
        }
        catch (GameNodeDoesNotExistException e) {
```

```
                List<GameNode> l = new ArrayList<GameNode>();
                l.add(current);

                return l;
        }
}

/**
 * Computes the Game Strategy, that I defined previously. It's here where will be applied all the computation for my strategy.
 *
 * @param listP1 the list of Game Nodes of my Game Tree, as Player no. 1
 *
 * @param listP2 the list of Game Nodes of my Game Tree, as Player no. 2
 *
 * @param myStrategy the computational strategy, that I defined previously and that will be used by me for the current Game
 *
 * @param random a Secure Random object, to calculate random numbers' operations
 *
 * @throws GameNodeDoesNotExistException a GameNodeDoesNotExist to be thrown if
 *         the a certain Game Node don't exist in the current Game
 */
private void cumputeStrategy(List<GameNode> listP1, List<GameNode> listP2,

PlayStrategy myStrategy, SecureRandom random)

                                        throws GameNodeDoesNotExistException {
```

```
Set<String> opponentMoves = new HashSet<String>();

// When I played as Player no. 1, I'm going to check
// what were the moves of my opponent as Player no. 2
for(GameNode n: listP1) {
        if(n.isNature() || n.isRoot()) continue;

        if(n.getAncestor().isPlayer2()) {
                opponentMoves.add(n.getLabel());
        }
}

// When I played as Player no. 2, I'm going to check
// what were the moves of my opponent as Player no. 1
for(GameNode n: listP2) {
        if(n.isNature() || n.isRoot()) continue;

        if(n.getAncestor().isPlayer1()) {
                opponentMoves.add(n.getLabel());
        }
}

System.out.println();

System.out.println("My Opponent's Plays:");
for(String opponentMove : opponentMoves) {
```

```
            System.out.println("- " + opponentMove);
}

System.out.println();

Iterator<String> moves = myStrategy.keyIterator();

// I will analyse all the possible moves
while(moves.hasNext()) {

            // The current possible move
            String currentMove = moves.next();

            System.out.println();
            System.out.println();

            System.err.println("Analysing " + currentMove + " ...");

            System.err.println();

            String[] playStructure = currentMove.split(":");

            int currentOpponentPlayer = Integer.parseInt(playStructure[0]);

            int currentMyselfPlayer = (currentOpponentPlayer == 1) ? 2 : 1;
```

```
String currentAction = playStructure[2];

// Currently, analysing a possible Cooperate action,
// before I decide
if(currentAction.equalsIgnoreCase(COOPERATE)) {

    // In this case, my opponent Cooperates in the previous round
    if(opponentMoves.contains(currentMove)) {
        System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Cooperates in the last round!!!");
        System.out.println();
        this.possibleCooperateActionKnowingThatMyOpponentCooperateInPreviousRound(myStrategy, currentMyselfPlayer, currentOpponentPlayer, currentMove);
    }

    // In this case, my opponent Defect in the previous round
    else {
        System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Defects in the last round!!!");
        this.possibleCooperateActionKnowingThatMyOpponentDefectInPreviousRound(myStrategy, currentMyselfPlayer, currentOpponentPlayer, currentMove);
    }
}

// Currently, analysing a possible Defect action,
// before I decide
if(currentAction.equalsIgnoreCase(DEFECT)) {
```

```
                    // In this case, my opponent Defect in the previous round
                    if(opponentMoves.contains(currentMove)) {
                        System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Defects in the last round!!!");
                        System.out.println();
                        this.possibleDefectActionKnowingThatMyOpponentDefectInPreviousRound(myStrategy, currentMyselfPlayer, currentOpponentPlayer, currentMove);
                    }

                    // In this case, my opponent Cooperates in the previous round
                    else {
                        System.err.println("My Opponent (as Player no. " + currentOpponentPlayer + ") Cooperates in the last round!!!");
                        System.out.println();
                        this.possibleDefectActionKnowingThatMyOpponentCooperateInPreviousRound(myStrategy, currentMyselfPlayer, currentOpponentPlayer, currentMove);
                    }
                }
            }


            System.out.println();


            // Print my Player's statistics, related to the number of "Cooperate" and "Defect" actions
            System.out.println();
```

```
// Print the current number of Cooperates of Myself
System.out.println("Number of Cooperates of Myself as Player no. 1: " + this.currentNumMyselfCooperates[0]);
System.out.println("Number of Cooperates of Myself as Player no. 2: " + this.currentNumMyselfCooperates[1]);

System.out.println();

// Print the current number of Defect of Myself
System.out.println("Number of Defects of Myself as Player no. 1: " + this.currentNumMyselfDefects[0]);
System.out.println("Number of Defects of Myself as Player no. 2: " + this.currentNumMyselfDefects[1]);

System.out.println();


// Print Opponent's Player's statistics, related to the number of "Cooperate" and "Defect" actions
System.out.println();

// Print the current number of Cooperates of the Opponent
System.out.println("Number of Cooperates of the Opponent as Player no. 1: " + this.currentNumOpponentCooperates[0]);
System.out.println("Number of Cooperates of the Opponent as Player no. 2: " + this.currentNumOpponentCooperates[1]);

System.out.println();

// Print the current number of Defect of the Opponent
System.out.println("Number of Defects of the Opponent as Player no. 1: " + this.currentNumOpponentDefects[0]);
System.out.println("Number of Defects of the Opponent as Player no. 2: " + this.currentNumOpponentDefects[1]);
```

```
System.out.println();



// The following piece of code has the goal of checking if there was a portion
// of the game for which we could not infer the moves of the adversary
// (because none of the current Game's plays in the previous round pass through those paths)
Iterator<Integer> validationSetIte = tree.getValidationSet().iterator();
moves = myStrategy.keyIterator();

while(validationSetIte.hasNext()) {
        int possibleMoves = validationSetIte.next().intValue();
        String[] labels = new String[possibleMoves];
        double[] values = new double[possibleMoves];
        double sum = 0;

        for(int i = 0; i < possibleMoves; i++) {
                labels[i] = moves.next();
                values[i] = ((Double) myStrategy.get(labels[i])).doubleValue();
                sum += values[i];
        }

        if(sum != 1) {

                // In the previous current Game's play,
                // I couldn't infer what the adversary played here
```

```
                        // Will be applied a random move on this validation set
                        sum = 0;

                        for(int i = 0; i < values.length - 1; i++) {
                                values[i] = random.nextDouble();
                                while(sum + values[i] >= 1) values[i] = random.nextDouble();
                                sum = sum + values[i];
                        }

                        values[values.length - 1] = ((double) 1) - sum;

                        for(int i = 0; i < possibleMoves; i++) {
                                myStrategy.put(labels[i], values[i]);
                                System.err.println("Unexplored path: Setting " + labels[i] + " with probability of " +
values[i]);
                        }
                }
        }
}


@Override
public void execute() throws InterruptedException {

        SecureRandom random = new SecureRandom();

        while(!this.isTreeKnown()) {
```

```
                System.err.println("Waiting for the Game Tree to become available...");
                Thread.sleep(1000);
        }

        GameNode finalP1 = null;
        GameNode finalP2 = null;

        while(true) {

                PlayStrategy myStrategy = this.getStrategyRequest();



                // The current Game was terminated by an outside event
                if(myStrategy == null) {
                        break;
                }

                boolean playComplete = false;

                while(!playComplete) {
                        if(myStrategy.getFinalP1Node() != -1) {
                                finalP1 = this.tree.getNodeByIndex(myStrategy.getFinalP1Node());
                                if(finalP1 != null)
                                        System.out.println("Final/Terminal node in last round as P1: " + finalP1);
                        }
```

```
if(myStrategy.getFinalP2Node() != -1) {
        finalP2 = this.tree.getNodeByIndex(myStrategy.getFinalP2Node());
        if(finalP2 != null)
            System.out.println("Final/Terminal node in last round as P2: " + finalP2);
}

Iterator<Integer> iterator = tree.getValidationSet().iterator();
Iterator<String> keys = myStrategy.keyIterator();

if(finalP1 == null || finalP2 == null) {

        // This is the first round of the current Game
        while(iterator.hasNext()) {
            double[] moves = new double[iterator.next()];

            double probabilityToContinueInFirstRound =
myStrategy.probabilityForNextIteration();

            // If it's will be played more than one round
            if(myStrategy.getMaximumNumberOfIterations() > 1) {

                // If the probability of continue in the next round,
                // it's greater or equal than 0.3333%, I will play a "Cooperate" action
                if( probabilityToContinueInFirstRound >=
SAFE_BETA_PROBABILITY_TO_CONTINUE ) {
```

```
                    // Here, I will start to Cooperate, as both, Player no. 1 and
Player no. 2
                    moves[0] = 1.0;
                    moves[1] = 0.0;
                }

                // Otherwise, if the probability of continue in the next round,
                // it's lesser than 0.3333%, I will play a "Defect" action
                else {

                    // Here, I will start to Defect, as both, Player no. 1 and Player
no. 2
                    moves[0] = 0.0;
                    moves[1] = 1.0;
                }
            }

            // If it's will be played only one round
            else {

                // Here, I will start to Defect, as both, Player no. 1 and Player no. 2
                moves[0] = 0.0;
                moves[1] = 1.0;
            }

            for(int i = 0; i < moves.length; i++) {
```

```
                                        if(!keys.hasNext()) {
                                                System.err.println("PANIC: Strategy structure doesn't match
the current Game!!!");
                                                return;
                                        }

                                        String firstPlay = keys.next();

                                        System.out.println();
                                        System.out.println("My First Play - " + firstPlay + " with probability of "
+ moves[i]);

                                        myStrategy.put(firstPlay, moves[i]);
                                }
                        }
                }
                else {

                        // Let's, now, play the Gradual Strategy (at least what we can infer)
                        List<GameNode> listP1 = getReversePath(finalP1);
                        List<GameNode> listP2 = getReversePath(finalP2);

                        try {
                                cumputeStrategy(listP1, listP2, myStrategy, random);
                        }
                        catch (GameNodeDoesNotExistException gameNodeDoesNotExistException) {
                                System.err.println("PANIC: Strategy structure doesn't match the current
Game!!!");
```

```
                    }
                }

                try {
                    this.provideStrategy(myStrategy);
                    playComplete = true;
                }
                catch(InvalidStrategyException invalidStrategyException) {
                    System.err.println("Invalid Strategy: " + invalidStrategyException.getMessage());;
                    invalidStrategyException.printStackTrace(System.err);
                }
            }
        }
    }
}
```