Number of players	40
Size of chromosome	64
Temptation	5
Reward	3
Punishment	1
Mutate probability	0.001
Crossover probability	0.95

Graph 1 (start)

Chromosome	Iteration	Sum
1	010110110	22
2	010101011	21
1	010010101	20
3	001010101	23
1	010100101	20
4	011100101	25
1	010111001	23
5	000111001	22
1	010001110	25
6	110011001	22
1	010011100	20
7	110011110	21
1	010111100	23
8	110010111	24
1	010011001	22
9	001111001	25
1	010111010	20
10	000111010	23

Graph 2 (medium)

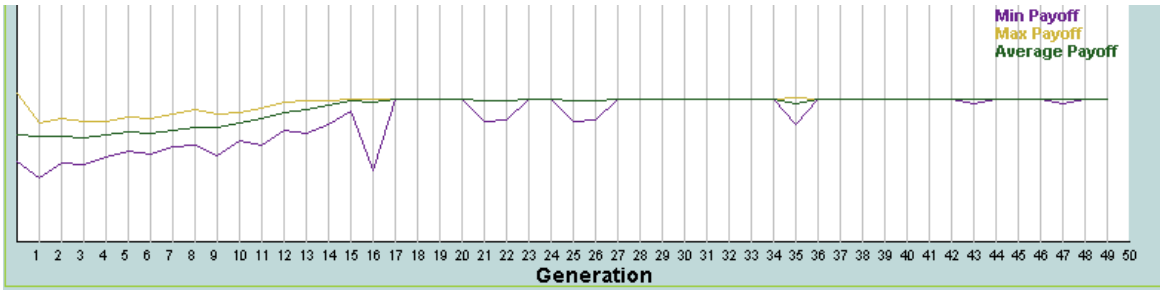
11	110010111	21
12	001111100	20
11	110111010	22
13	111010111	23
11	110001011	25
14	110101110	22
11	110010101	23
15	101000011	21
11	110110010	20
16	010101100	24
11	110111000	21
17	000011100	20
11	110000111	22
18	110000110	23
11	110110011	24
19	100100011	21
11	110001100	22
20	000001111	20

Graph 3 (finish)

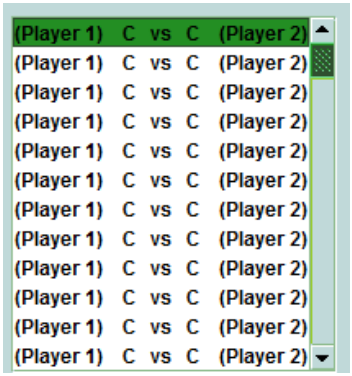
21	111000111	23
22	111001100	25
21	111111000	21
23	010111010	23
21	111010101	21
24	000111000	20
21	111101010	22
25	010101010	24
21	111110011	21
26	001100111	24
21	111001100	25
27	001110101	23
21	111001001	21
28	010101111	23
21	111110110	24
29	001101110	20
21	111000001	21
30	110000101	24

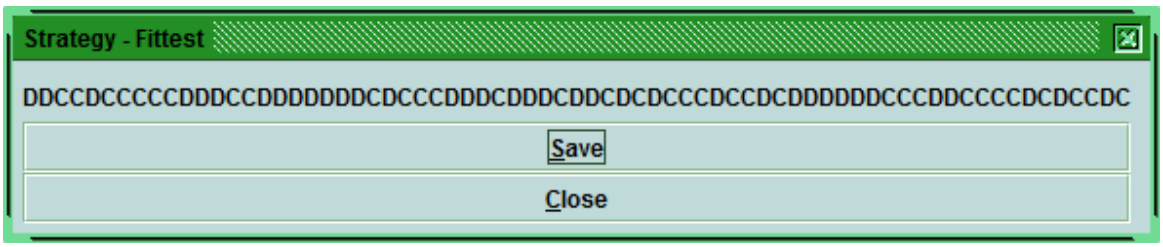
The tables show 3 samples: chromosome sequence number, chromosome selection at each iteration, the sum obtained by "playing" with another chromosome. So we get what were the chromosomes at the beginning of the dilemma, in the middle and at the end. The amount is needed in order to understand what fitness in the chromosome (we compare the amounts and get the fitness).

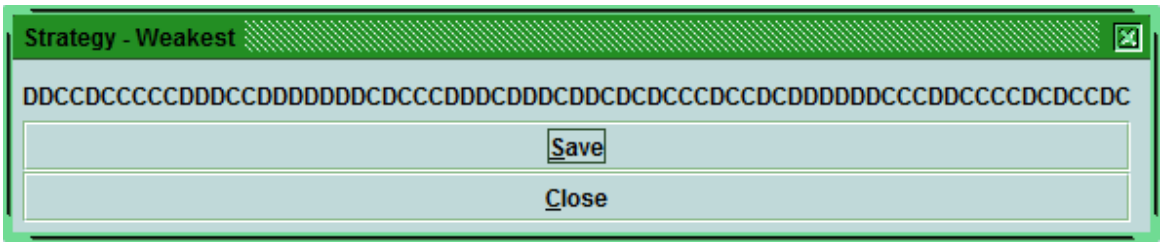
GRAPHICS



GAME HISTORY







C – 0 (COOPERATED)
D – 1 (DEFETEAD)


```
package ie.errity.pd;

import ie.errity.pd.graphics.*;
//Import GUI components
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.plaf.metal.*;


/**
 *MAIN class. Contains code which launches an instance of the Evolutionary
 *Prisoner's Dilemma application.
 *@author Andrew Errity 99086921
 */
public class epd
{
  /**
   *MAIN class. Contains code which launches an instance of the Evolutionary
   *Prisoner's Dilemma application.
   *@param args  command line agruments (not processed)
   */
  public static void main(String[] args)
  {
    //Make sure we have window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);
```

```
    //Create a frame and container for the game panels.
    MenuFrame progFrame = new MenuFrame("Evolutionary Prisoner's Dilemma");

        // THEMES
    // user selected theme
    MetalTheme theme = new EmeraldTheme();
    // set the chosen theme
    MetalLookAndFeel.setCurrentTheme(theme);
    try
    {
       UIManager.setLookAndFeel(
      UIManager.getCrossPlatformLookAndFeelClassName());
       SwingUtilities.updateComponentTreeUI(progFrame);
    }
    catch(Exception e){}

     // Exit when the window is closed.
     progFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Show the program
    progFrame.pack();
    progFrame.setVisible(true);

  }

}

package ie.errity.pd;

import java.util.BitSet;


/**
 *A game of the Iterated Prisoner's Dilemma between two
 *{@link  ie.errity.pd.Prisoner  Prisoners}
 *@author  Andrew Errity 99086921
 */
public class Game
{
  //Two Players
  private Prisoner P1;
  private Prisoner P2;

  //Histories from each players view
  private BitSet P1History;
  private BitSet P2History;

  //Payoffs recieved
  private int P1Score;
  private int P2Score;

  //Rules for the IPD
  private Rules rules;

  /**
   *Create a new game of the Iterated Prisoner's Dilemma between two players
   *@param p1 player 1
   *@param p2 player 2
   *@param r    the rules which govern the game
   */
  public Game(Prisoner p1, Prisoner p2, Rules r)
```

```
{
  P1 = p1;
  P2 = p2;
  rules = r;
}


/**Play a game of IPD according to the rules*/
public void Play()
{
  //Init
  int length = rules.getIterations();
  int iteration = 0;
  P1Score = 0;
  P2Score = 0;
  BitSet P1History = new BitSet();
  BitSet P2History = new BitSet();
  boolean P1move, P2move;

  //Play the specified number of PD games
  for(iteration = 0; iteration < length; iteration++)
  {
    //Get each players move
    P1move = P1.play(iteration,P1History);
    P2move = P2.play(iteration,P2History);

    //Update scores according to payoffs
    if(P1move && P2move) //CC
    {
      P1Score += rules.getR();
      P2Score += rules.getR();
    }
    else if(P1move && !P2move) //CD
    {
      P1Score += rules.getS();
      P2Score += rules.getT();

    }
    else if(!P1move && P2move) //DC
    {
      P1Score += rules.getT();
      P2Score += rules.getS();

    }
    else if(!P1move && !P2move) //DD
    {
      P1Score += rules.getP();
      P2Score += rules.getP();

    }

    //Update player histories
    if(P1move)
    {
      P1History.set(iteration*2);
      P2History.set((iteration*2)+1);
    }
    else
    {
      P1History.clear(iteration*2);
      P2History.clear((iteration*2)+1);
    }
```

```
      if(P2move)
      {
        P1History.set((iteration*2)+1);
        P2History.set((iteration*2));
      }
      else
      {
        P1History.clear((iteration*2)+1);
        P2History.clear((iteration*2));
      }
    }
    //Update each players score
    P1.updateScore(P1Score);
    P2.updateScore(P2Score);
  }

  /**
   *Get game results
   *@return array containing player 1 and player 2's score - [p1,p2]
   */
  public int[] getScores()
  {
    int   [] scores = {P1Score, P2Score};
    return scores;
  }
}
```

Conclusion: in the prisoner's dilemma, betrayal strictly dominates cooperation, so the only possible balance is the betrayal of both parties. Simply put, it does not matter what the other player will do, everyone will win more if he betrays. Since in any situation betraying is more profitable than cooperating, all rational players will choose betrayal.

Behaving individually individually, rationally, together the participants come to an irrational decision: if both betray, they will get a smaller payoff than if they were cooperating (the only equilibrium in this game does not lead to a Pareto-optimal decision). This is the dilemma.