Examining the Evolution of Social Behaviors

New Mexico
Supercomputing Challenge
Final Report
April 4, 2007

Team 94
Sandia Preparatory School

Team Members:
Alex Clement
Amy Clement
Greg Fenchel
Jeff Fenchel
Jayson Lynch

Teacher:
Neil McBeth

Mentor:
Mark Smith

Table of Contents

Executive Summary:

This program explores the evolution of the discriminatory social behavior ethnocentrism in a given geographical location. Ethnocentrism is one of several social behaviors currently used to describe the general pattern of interaction amongst human beings. It is a pattern of social interaction in which people of the same group show favoritism toward others of the same group. In this model ethnocentrism is demonstrated by agents who tend to cooperate with others of their same ethnicity while tending to defect with agents of different ethnicities. The preliminary objective of this project was to design a program in which a pattern of interaction similar to ethnocentrism emerges. In order to do this, we replicated the results of Ross A. Hammond and Robert Axelrod as found in their paper, *The Evolution of Ethnocentrism*.

In addition to exploring the model developed by Hammond and Axelrod, the team made two significant original modifications. First the team modified the program so that agents along a certain region could interact. This modification simulates a geographical feature that facilitates movement or trade. In this model ethnocentrism still emerged as a dominant behavioral strategy, but the "rivers" did reduce the percentage of ethnocentrism. Our second modification allowed an agent to distinguish the ethnicity of the agent with whom he is interacting. In addition to our four primary strategies, several other tendencies emerged in the second modification. In many ways, this model more accurately represents the current patterns of interaction in a society than the original program. Ultimately we confirmed our hypothesis: ethnocentrism emerges spontaneously under mild conditions. By studying ethnocentrism and its effect on a developing society, we have improved our understanding of the social divisions in our world.

Problem Statement:

The goal of this project is to explore the hypothesis that ethnocentrism is an emergent discriminatory behavior by developing a computational model in which ethnocentrism is an emergent property. By modeling the social development of a society, we hope to explore the evolution of this behavior and its effect on history, politics, and communication. By better understanding the conditions that cause ethnocentrism, we hope to increase our understanding of the racial, religious, and historical tensions that divide our planet.

Description of the Basic Program:

This model was created in Microsoft Visual Studio C#. In an attempt to improve our knowledge of programming, we chose to use object oriented programming to create this model. In object-oriented programming, the action of the program is divided into a series of classes. A main class dictates the order in which these classes run. Within the classes, there are a series of smaller objects which perform very specific actions. Due to the complexity and length of our program, object-oriented programming was an essential organizational tool. Also, we were able to divide the programming equally amongst team members. Each team member was assigned a specific class to create, and the classes where combined to form a complete program. In addition to the basic code dictating immigration, interaction, reproduction, and death, we have created a user interface, a numerical readout of the results, a graph that shows the evolution of ethnocentrism, and a graphic representation of the array which shows the ethnicity and strategy of each agent.

There are four main steps involved in this model: immigration, interaction, reproduction, and death. At each time-step the model goes through each of these steps one time. The standard length for the model is 1000 time-steps.

Immigration is the first step in the model. An 'immigrant' of random ethnicity and strategy is put on a randomly selected, unoccupied square of the array. The immigration rate determines how many 'immigrants' arrive at each time-step. The immigration step populates the world and provides a supply of people of differing strategies and ethnicities, both of which are essential to the model.

Interaction allows the agents to interact with each other the results of these interactions modify their chance to reproduce. This modification allows for those who did well in the interactions to produce more offspring allowing for the most successful strategy to emerge. The interaction step is based off of the game "The Prisoner's Dilemma". In this game, players have two options: cooperate or defect. There are three possible situations to the Prisoner's Dilemma. One outcome occurs when both agents cooperate. This strategy causes both agents to receive a two percent benefit to their chance of reproduction. This strategy yields the highest net gain, yet is not the most beneficial for the agent. The most beneficial outcome for the agent occurs when the agent defects and the other agent it plays the prisoners dilemma with cooperates. In this possibility, the agent defecting receives and three percent increase to their chance of reproduction and the agent cooperating receives a one percent decrease to their chance of reproduction thus making this strategy the most beneficial for the agent defecting. The last situation occurs when both agents defect, which yields no change to both of their chances of reproducing. The Prisoners Dilemma game creates complexity when deciding what strategy to implement because, from a selfish standpoint, to defect yields the best outcome, but there is a greater net benefit from cooperation.

In our model, the strategy used in each interaction is linked to the ethnicity of the other agent compared to the ethnicity of the individual agent deciding what strategy to use. Our basic model only acknowledges if the ethnicities are different or the same; there is no distinction between the different ethnicities. The four different strategies we have grouped the agents possible methods of interaction into are: Ethnocentric, Altruist, Egoist, and Cosmopolitan. Agents grouped under the ethnocentric strategy cooperate only with agents of their same ethnicity and defect with all other agents. Altruists cooperate with everybody: those of their

ethnicity and those of a different ethnicity. Egoists defect with all other agent essentially become free riders off of other agent's cooperation. Cosmopolitans cooperate with everybody except those of their same ethnicity. After each agent has interacted once with each of its four neighbors, the agents reproduce.

After the Interaction step modifies the reproduction rate of each agent, each agent is given an opportunity to reproduce based on its chance of reproduction. The default reproduction rate is 12%. However, this chance becomes modified in the interaction step so that the most successful in the interaction reproduce more. The program cycles through all the agents in a random order and determines if there are empty squares adjacent to each agent. If there is at least one empty square next to the agent, there is a chance of a person being 'born' one of those squares based on the agent chance of reproducing. When offspring are produced they inherent their parents ethnicity and strategy. Incorporated into reproduction is the chance of mutation, an important aspect that helps facilitate evolution among the population. Mutation is incorporated when an agent is 'born' by allowing for a chance that person will have a randomly determined strategy and/or ethnicity instead of inheriting his parent's traits. The standard mutation rate is 0.5%. This step is an essential part of an evolutionary model because without mutation, new strategies will not arise and thus there will be no opportunity for evolution.

After reproduction each agent has a chance of being removed from the array to replicate death within a population. This is essential for allowing the population to renew itself, change, and make room for an evolved agent. The default death rate is 10%, two percent less than the default reproduction rate, a difference that results in exponential population growth until the area encompassing the population becomes filled.

This program repeats each of these steps for a given number of steps and averages the results over a predefined number of runs. The default number of steps is 1000 allowing for significant evolution among the population. This simulation is then repeated over the default setting of 10 runs to average the results ensuring that they are not a abnormal set of data created by the random number generator.

This program incorporates a pseudorandom number generator called the Mersenne Twister instead of the standard random number generator found in C#. Although the addition of the Mersenne Twister does not significantly alter our results or conclusions, it increases the accuracy of our results. Appendix VII examines the effect a changing seed in the Mersenne Twister on the program.

Results:

In the simulations run the ethnocentric strategy became the dominant strategy. After this, the altruist was most successful, followed by egoist and finally cosmopolitan. The default run with a seed of 0 yielded an average of 73.4% ethnocentric strategy over 10 runs with a standard deviation of 9.2%. The altruist strategy averaged 18.4% with 8.4% standard deviation, the egoist strategy averaged 6% with 1.5% standard deviation, and the cosmopolitan strategy averaged 2.3% with 0.7% standard deviation. The results obtained when the random number generator seed was altered were consistent with the above results in that the ethnocentric strategy clearly became dominant within the model.

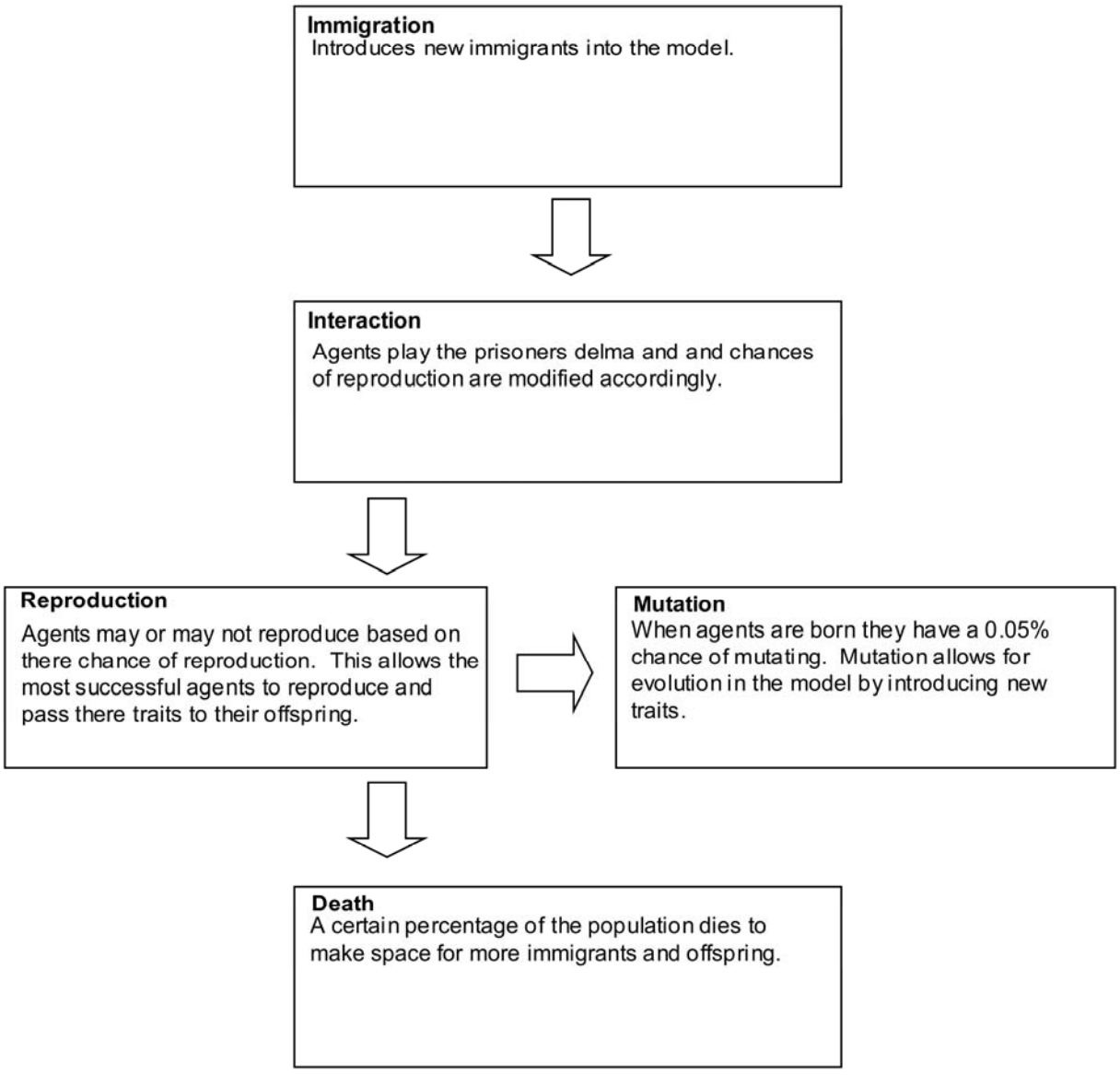
Conclusion:

In this model ethnocentrism arises under mild conditions. Further it becomes a dominant strategy governing interaction. This verifies the results of Hammond and Axelrod despite the slight deviation of percentages.

There is a logical explanation for why ethnocentrism becomes dominant within this model. It depends on the localization within the interaction and reproduction steps. When reproduction occurs the offspring are put into a square adjacent to its parent and also has the same traits as the parent. If these two subsequently cooperate they will have a higher chance of reproducing. This will occur if their strategy is ethnocentric or altruist, explaining the success of these strategies. Further the ethnocentric strategy defects with other ethnicities and will gain a larger benefit from interacting with those of different ethnicity than the altruists would. This also reduces the chance that agents of other ethnicities will reproduce, giving ethnocentrics more room into which they may expand. The combination of benefiting from one's offspring and maximizing the benefit of interacting with other ethnicities allows the ethnocentric strategy to become dominant.

One real world example that supports this conclusion is the formation of gangs within a city. People of one ethnicity or religion tend to interact more freely with those of a similar ethnicity or religion. Over time, these people find that their interactions amongst people of their own kind are more successful, and they tend towards one area. As new generations take over, the same tendencies dominate social interaction. Soon, several distinct groups emerge in a city. Because they do not interact well with individuals of a different ethnicity, violence often erupts when "defections" occur. Often, the formation of gangs polarizes a city and results in increased crime rates. Historically, ethnocentrism has played a hand in the formation of borders between countries, but it also affects society on a much smaller scale in cities and small towns.

Program Flow Chart



Description of Geographical Modification

The geographical modification allows further interaction amongst the agents bordering a "river". A "river" is basically a straight line through the array of agents. All the agents who are on the line interact with all the other agents along the "river". This modification simulates how rivers serve as major transportation arteries allowing more interaction amongst the people who live on the river.

Results:

In a run with five rivers and a seed of 0, we found that 64.5% ± 12.1% of the population was ethnocentric, proving that ethnocentrism still emerges as the dominant strategy even though the rivers reduce the localization of specific ethnicities. Cosmopolitanism emerged as the next most popular strategy with 19.7% ± 7.6% of the population. Altruism was almost as popular with 12.5% ± 6.7% of the population. Egoism was again the least popular strategy with 3.3% ± 2.8% of the population.

Conclusion:

This modification highlighted the importance of localized population centers in which agents have similar ethnicities. As ethnocentrics reproduce, their offspring remain in adjacent squares and interact with their parents, strengthening the effect of ethnocentrism. When agents are interacting with individuals all over the grid due to increased "trade" along the "rivers," the effects of ethnocentrism are weakened. Although it still emerges as the dominant strategy, the increased interaction amongst people of different ethnicities discourages ethnocentrism along the "banks" of the "rivers."

In real life, we see that rivers tend to encourage cultural and religious diversity along their banks. River towns and seaports are famous for their diverse communities. This modification shows the second major step in the social evolution of a society. As a community first develops, it is often isolated from other communities and major trade routes. As the community grows into a bustling city, members of the city begin to look outward. As a results, trade flourishes, and more interaction between individuals of different ethnicities takes places, discouraging ethnocentrism in major trade centers.

Description of "Ethnicity Recognition" Modification

The "Ethnicity Recognition" Modification allows an agent to recognize the specific ethnicity of the agent with whom he is interacting. In the normal program agents can only recognize if the agent with whom they are interacting has the same ethnicity or a different ethnicity. This modification was written to determine whether ethnocentrism will still arise if the agents can distinguish between other agents' ethnicities, and thus is not dependent on an "us and them" system. This model could also lead to unexpected phenomena occurring. Our group predicted a variety of things in conjunction with this model including weaker ethnocentrism, stronger ethnocentrism, and "alliances" evolving between ethnicities. Finally, this model is an attempt to more closely model the real world because in the real world people can generally distinguish between groups of other people.

Results:

The results within the ethnic distinction model varied widely but had a few noticeable trends. First, ethnocentrism still became a successful strategy. It was used by approximately 20% of all the agents in all of the simulations except in the run with seed zero, where it was only

14.59%. Even so, in all of our runs it was far more prevalent than altruism, egoism, and cosmopolitanism. Altruism ranged between 4% and 8%, Egoism ranged between 1.3% and 4.2%, and Cosmopolitan ranged between 0.55% and 1.5%. Ethnocentrism was not always the dominant strategy in the model. In several runs, the dominant strategy consisted of the cooperation with two or threes other ethnicities and defecting with the rest. For example the CCDC strategy was used by 21.25% of the population in the run with seed zero and in seed four, strategy CDDC was 0.3% higher than the ethnocentric strategy. In this model all strategies which involve defecting with one's own ethnicity were unsuccessful. Even though these account for half of the possible strategies in each run they only accounted for 8% to 13% of the population.

Conclusion:

This model still contained ethnocentrism as an emergent property. Further, the ethnocentric strategy became a dominant strategy despite the agents' ability to distinguish ethnicities, although this did dampen the success of the ethnocentric strategy. This reduction in the percentage of the population using the ethnocentric strategy could be explained by the large increase in possible strategies, four in the original model and the number of ethnicities squared in this modification: this would be 16 for the default parameters. It may also be due to the functional similarity between ethnocentrism and other strategies in certain situations. For example, if a group of agents with ethnicity 1 who cooperate with ethnicities 1 and 2 but defect with ethnicities 3 and 4, never encounter agents of ethnicity 2 it would have been no different if they instead used the ethnocentric strategy.

It was also interesting to note the unexpected success of some strategies that involved cooperating with two or three other ethnicities and defecting with the rest. This appears to be the

"alliances" between ethnicities that was hypothesized, although more analysis, including that of the geographical relation between the agents using this strategy and the ethnicities involved, is necessary for a more definite conclusion. At the moment there is an explanation for this behavior. One may recall that ethnocentrism is successful in the original model because the offspring cooperate with themselves and with the parents, giving everyone a higher reproduction rate and thus producing more offspring. It also discourages the intrusion of other ethnicities which may defect with it and reduce its reproductive success. With this in mind let us look at a theoretical case in the new model. Assume there is an agent of ethnicity 1 which cooperates with ethnicities 1 and 2, and defects with 3 and 4. Now assume there is an agent of ethnicity 2 with the same strategy as the former agent. Both of the agents cooperate with each other. If they reproduce their offspring will cooperate with everyone involved. In addition any interference from ethnicities 3 and 4 will be discouraged. The relationship between ethnicities 1 and 2 has now become almost identical to that of one ethnicity with the ethnocentric strategy. This behavior suggests that complex social behaviors between people such as alliances might arise from the simple conditions of interaction, distinction between groups of people, and evolution.

Our hypothesis was correct; ethnocentrism emerges spontaneously under mild conditions. In our basic model, a pattern of social interaction similar to that of ethnocentrism always becomes the dominant strategy. As predicted, our geographical modification decreased the effect of ethnocentrism. The addition of "rivers" forced agents to interact with all the other agents along the river, increasing interaction amongst agents of different ethnicities and strategies. In this modification, we attempted to model the effect of trade on a developing society. Our ethnicity recognition modification revealed more complex patterns of interaction.

Ethnocentrism still emerges as a popular strategy, but in addition, we saw the formation of alliances between two ethnicities. These alliances indicate that ethnocentrism might evolve into a more complex pattern of interaction in which agents favor both their ethnicity and one other over all other ethnicities. This project taught us about the importance of race, religion, and culture in both our historical world and our modern world.

Bibliography:

Axelrod, Robert. The Evolution of Cooperation. New York: Basic Books, 1984.

Axelrod, Robert. Harnessing complexity: organizational implications of a scientific frontier. New York: Free Press, 1999.

Axelrod, Robert, and Ross Hammond. "The Evolution of Ethnocentrism." Journal of Conflict Resolution. 50.6 (Dec 2006): 926-936.

Brundage, Michael L. "Mersenne Twister." Random Number Generation. http://www.qbrundage.com/michaelb/pubs/essays/random_number_generation.html.

Matsumoto, Makoto. "Mersenne Twister with Improved Initialization." Mersenne Twister Homepage. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html.

Special Thanks to...
Mark Smith, our mentor, who has seen us through everything. Thanks so much for teaching us about object oriented programming and C#. We appreciate everything you do for us.

Neil McBeth, our sponsoring teacher, who has kept us on track with deadlines. Thank you for dealing with all the paperwork that we would have forgotten about.

Anita and Eric Gallagher for allowing us to make use of the conference facilities at all hours of the night.

Appendix I:

Code of Normal Program Separated by Classes

Program Class:

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Threading;

namespace EvoEthnocentrism
{
    static class Program
    {
        delegate void That();
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()//This is the piece of code which ties all the classes together
        {
            Parameters p = new Parameters();
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Input(p));

            int run = 0;
            ResultsHandler rh = new ResultsHandler(p, run);
            Life l = new Life();
            Interaction interact = new Interaction();

            for (run = 0; run < p.runNumber; run++)
            {
                Population pop = new Population(p);
                Graph graph = new Graph(p, pop);
                Graphic graphic = new Graphic( p, pop);
                if (p.runNumber - 1 == run)
                {
                    graph.Show();
                    graphic.Show();
                }

                for (int time = 1; time <= p.runLength; time++)
                {
                    l.immigration(pop,p, run);
                    Interaction.interaction(pop, p);
                    l.reproduction(pop,p);
                    l.death(pop, p);
                    if (time >= p.runLength - 100)
                        rh.countResults(pop, p, run);
                    if (p.runNumber - 1 == run)
                    {
                        graph.Refresh();
                        graphic.Refresh();
                    }
                }
               p.seed++;
               Console.Write(".  ");
            }

            ResultsHandler.StandardDeviation(rh, p);
            Application.Run(new Results(rh));
        }
    }
}
```

Input Class:

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace EvoEthnocentrism
{
    public partial class Input : Form
    {
        Parameters p;

        public Input(Parameters pinput)
        {
            InitializeComponent();
            p = pinput;
        }

        private void resetDefault_Click(object sender, EventArgs e)
            //This portion of the code resets the default parameters.
        {
            arrayInput.Text = (50).ToString();
            ethnicityInput.Text = (5).ToString();
            immigrationInput.Text = (1).ToString();
            deathInput.Text = (0.10).ToString();
            mutationInput.Text = (0.005).ToString();
            ptrInput.Text = (0.12).ToString();
            costInput.Text = (0.01).ToString();
            benefitInput.Text = (0.03).ToString();
            runLengthInput.Text = (2000).ToString();
            runNumberInput.Text = (10).ToString();
            seedInput.Text = (0).ToString();
        }

        public void go_Click(object sender, EventArgs e)
        //Here the input parameters are sent to the other classes in the program.
        {
            p.ptr = double.Parse(ptrInput.Text);//ptr to Life
            p.arraySize = int.Parse(arrayInput.Text);//array size to Population
            p.mutationRate = double.Parse(mutationInput.Text);//mutation rate to Life
            p.deathRate = double.Parse(deathInput.Text);//death rate to Life
            p.ethnicityNumber = int.Parse(ethnicityInput.Text);//number of ethnicities to
Population
            p.immigrationRate = int.Parse(immigrationInput.Text);// immigration rate to Life
            p.runLength = int.Parse(runLengthInput.Text);//run Length to main program
            p.runNumber = int.Parse(runNumberInput.Text);//run Number to main program
            p.cost = double.Parse(costInput.Text);//cost to Interaction
            p.benefit = double.Parse(benefitInput.Text);//benefit to Interaction
            p.seed = int.Parse(seedInput.Text);//seed input to Population
            Close();
        }
        private void arrayInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void ethnicityInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void immigrationInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void deathInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void mutationInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void ptrInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void costInput_TextChanged(object sender, EventArgs e)
```

```
        {
        }
        private void benefitInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void runLengthInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void runNumberInput_TextChanged(object sender, EventArgs e)
        {
        }
        public void seedInput_TextChanged(object sender, EventArgs e)
        {
        }
        private void Input_Load(object sender, EventArgs e)
        {
        }
    }
    public class Parameters
        //This class allows the input parameters to be shared throughout the program.
    {
        public int arraySize;
        public double deathRate;
        public double mutationRate;
        public double cost;
        public int seed;
        public double benefit;
        public int runLength;
        public int runNumber;
        public int immigrationRate;
        public int ethnicityNumber;
        public double ptr;
    }
}
```

Population Class:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{
    public class Population
    {
        public RandTool Rand;
        Parameters p;
        MT RandNumGen;
        public Person[,] people;
        public enum Strategies { DEFECT = 1, COOPERATE }

        public Population(Parameters p)
        {
            this.p = p;
            RandNumGen = new MT(p.seed);
            Rand = new RandTool(RandNumGen);
            people = new Person[p.arraySize, p.arraySize];
        }

        public class Person
        {
            public Strategies strategySame;
            public Strategies strategyDifferent;
            public int ethnicity;
            public double chanceOfReproducing;
            public ResultsHandler.StrategyNames strategyName;
        }

        public void DefaultTraits(int i, int e)
        {
            people[i, e].strategySame = (Strategies)Rand.RandBetween(1, 3);
```

```
            people[i, e].strategyDifferent = (Strategies)Rand.RandBetween(1, 3);
            people[i, e].chanceOfReproducing = p.ptr;
            people[i, e].ethnicity = Rand.RandBetween(0, p.ethnicityNumber);
        }

        public void Mutation( int currentx, int currenty)
        {
            if (p.mutationRate >= Rand.RandDouble())
            {
                people[currentx, currenty].ethnicity = Rand.RandBetween(0, p.ethnicityNumber);
            }
            if (p.mutationRate >= Rand.RandDouble())
            {
                people[currentx, currenty].strategyDifferent = (Strategies)Rand.RandBetween(1,
3);
            }
            if (p.mutationRate >= Rand.RandDouble())
            {
                people[currentx, currenty].strategySame =
(Population.Strategies)Rand.RandBetween(1, 3);
            }
        }

        public void CopyTraits(Population pop, int parentx, int parenty, int offspringx, int
offspringy)
        {
            pop.people[offspringx, offspringy].strategySame = pop.people[parentx,
parenty].strategySame;
            pop.people[offspringx, offspringy].strategyDifferent = pop.people[parentx,
parenty].strategyDifferent;
            pop.people[offspringx, offspringy].chanceOfReproducing = p.ptr;
            pop.people[offspringx, offspringy].ethnicity = pop.people[parentx,
parenty].ethnicity;
        }

        public static void GraphicMethod(Parameters p, Population pop, int time)
        {
            Graphic graphic = new Graphic( p, pop);
            graphic.Refresh();
        }

    }
}
```

Interaction Class:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{
    class Interaction
    {
        static void Play(Population.Person self, Population.Person other, Parameters p)
            //This code tells the agents how to interact with their neighbors.
            //They compare their strategies with their neighbor's and either cooperate or defect
based on their strategies.
            //Their chance to reproduce is then modified.
        {
            if (self.ethnicity == other.ethnicity)
            {
                if (other.strategySame == Population.Strategies.COOPERATE)
                    self.chanceOfReproducing += p.benefit;

                if (self.strategySame == Population.Strategies.COOPERATE)
                    self.chanceOfReproducing -= p.cost;
            }
            else
            {
                if (other.strategyDifferent == Population.Strategies.COOPERATE)
```

```
                    self.chanceOfReproducing += p.benefit;

                if (self.strategyDifferent == Population.Strategies.COOPERATE)
                    self.chanceOfReproducing -= p.cost;
            }

        }
        public static void interaction(Population pop, Parameters p)
            // This portion of the code ensures that everybody will interact only once in each
array.
        {
            for (int i = 0; i < p.arraySize; i++)
            {
                for (int e = 0; e < p.arraySize; e++)
                {
                    if (pop.people[i, e] != null)
                    {
                        if ( pop.people[(i + 1) % p.arraySize, e] != null )
                            Play( pop.people[i, e], pop.people[(i + 1) % p.arraySize, e], p );
                        if ( pop.people[(i + (p.arraySize - 1)) % p.arraySize, e] != null )
                            Play( pop.people[i, e], pop.people[(i + (p.arraySize - 1)) %
p.arraySize, e], p );
                        if ( pop.people[i,  (e + 1) % p.arraySize] != null )
                           Play( pop.people[i, e], pop.people[i, (e + 1) % p.arraySize], p );
                        if ( pop.people[i, (e + (p.arraySize - 1)) % p.arraySize] != null )
                           Play( pop.people[i, e], pop.people[i, (e + (p.arraySize - 1)) %
p.arraySize], p );

                    }
                }

            }

        }

    }
}
```

Life Class:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{
    public class Life
    {
        public class PersonCords
        {
            public int xCord;
            public int yCord;
        }

        public void immigration(Population pop, Parameters p, int run)
        {
            PersonCords[] nullCords = new PersonCords[0];
            int currentx, currenty;
            for (currentx = 0; currentx < p.arraySize; currentx++)
            {
                for (currenty = 0; currenty < p.arraySize; currenty++)
                {
                    if (pop.people[currentx, currenty] == null)
                    {
                        Array.Resize(ref nullCords, nullCords.Length + 1);
                        nullCords[nullCords.GetUpperBound(0)] = new PersonCords();
                        nullCords[nullCords.GetUpperBound(0)].xCord = currentx;
                        nullCords[nullCords.GetUpperBound(0)].yCord = currenty;
                    }
                }
```

```
            }


            pop.Rand.RandPermutation(nullCords);

            int newImmigrants = p.immigrationRate;
            if (p.immigrationRate > nullCords.Length)
                newImmigrants = nullCords.Length;


                for (int immigrants = 0; immigrants < newImmigrants; immigrants++)
                {
                    pop.people[nullCords[immigrants].xCord, nullCords[immigrants].yCord] = new
Population.Person();
                    pop.DefaultTraits(nullCords[immigrants].xCord, nullCords[immigrants].yCord);

                }
        }

        public void reproduction(Population pop, Parameters p)
        {
            PersonCords[] nonNullCords = new PersonCords[0];
            int currentx, currenty;
            int x = 0;
            for (currentx = 0; currentx < p.arraySize; currentx++)
            {
                for (currenty = 0; currenty < p.arraySize; currenty++)
                {
                    if (pop.people[currentx, currenty] != null)
                    {
                        Array.Resize(ref nonNullCords, nonNullCords.Length + 1);
                        nonNullCords[x] = new PersonCords();
                        nonNullCords[x].xCord = currentx;
                        nonNullCords[x].yCord = currenty;
                        x++;
                    }
                }
            }

            pop.Rand.RandPermutation(nonNullCords);

            for (int i = 0; i < nonNullCords.Length; i++)
            {
                if (pop.Rand.RandDouble() < pop.people[nonNullCords[i].xCord,
nonNullCords[i].yCord].chanceOfReproducing)
                {
                    int[] neighborx = new int[4];
                    int[] neighbory = new int[4];
                    int neighborCount = 0;
                    if (pop.people[(nonNullCords[i].xCord + 1) % p.arraySize,
nonNullCords[i].yCord] == null)
                    {
                        neighborx[neighborCount] = (nonNullCords[i].xCord + 1) % p.arraySize;
                        neighbory[neighborCount] = nonNullCords[i].yCord;
                        neighborCount++;
                    }
                    if (pop.people[nonNullCords[i].xCord, (nonNullCords[i].yCord + 1) %
p.arraySize] == null)
                    {
                        neighborx[neighborCount] = nonNullCords[i].xCord;
                        neighbory[neighborCount] = (nonNullCords[i].yCord + 1) % p.arraySize;
                        neighborCount++;
                    }
                    if (pop.people[(nonNullCords[i].xCord + (p.arraySize - 1)) % p.arraySize,
nonNullCords[i].yCord] == null)
                    {
                        neighborx[neighborCount] = (nonNullCords[i].xCord + (p.arraySize - 1)) %
p.arraySize;
                        neighbory[neighborCount] = nonNullCords[i].yCord;
                        neighborCount++;
                    }
```

```
                    if (pop.people[nonNullCords[i].xCord, (nonNullCords[i].yCord + (p.arraySize -
1)) % p.arraySize] == null)
                    {
                        neighborx[neighborCount] = nonNullCords[i].xCord;
                        neighbory[neighborCount] = (nonNullCords[i].yCord + (p.arraySize - 1)) %
p.arraySize;
                        neighborCount++;
                    }
                    if (neighborCount > 0)
                    {
                        int randomNeighbor = pop.Rand.RandBetween(0, neighborCount);
                        pop.people[neighborx[randomNeighbor], neighbory[randomNeighbor]] = new
Population.Person();
                        pop.CopyTraits(pop, nonNullCords[i].xCord, nonNullCords[i].yCord,
neighborx[randomNeighbor], neighbory[randomNeighbor]);
                        pop.Mutation(neighborx[randomNeighbor], neighbory[randomNeighbor]);
                    }
                }
                pop.people[nonNullCords[i].xCord, nonNullCords[i].yCord].chanceOfReproducing =
0.12;
            }
        }

        public void death (Population pop, Parameters p)
        {
            int currentx, currenty;
            for (currentx = 0; currentx < p.arraySize; currentx++)
            {
                for (currenty = 0; currenty < p.arraySize; currenty++)
                {
                    if (pop.Rand.RandDouble() < p.deathRate)
                    {
                        pop.people[currentx, currenty] = null;
                    }
                }
            }
        }
    }
}
```

Mersenne Twister (MT) Class:

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace EvoEthnocentrism
{
    //The following code is based off of that by Michael Brundage as found in
    //"Mersenne Twister." Random Number Generation.
http://www.qbrundage.com/michaelb/pubs/essays/random_number_generation.html.
    public class MT
    {

        private uint mt_index;
        private uint[] mt_buffer = new uint[624];

        public MT(int seed)
        {
            Random r = new Random(seed);
            for (uint i = 0; i < 624; i++)
                mt_buffer[i] = (uint)r.Next();
            mt_index = 0;
        }

        public uint Random()
        {
            if (mt_index == 624)
            {
                mt_index = 0;
                uint i = 0;
```

```
                uint s;
                for (; i < 624 - 397; i++)
                {
                    s = (mt_buffer[i] & 0x80000000) | (mt_buffer[i + 1] & 0x7FFFFFFF);
                    mt_buffer[i] = mt_buffer[i + 397] ^ (s >> 1) ^ ((s & 1) * 0x9908B0DF);
                }
                for (; i < 623; i++)
                {
                    s = (mt_buffer[i] & 0x80000000) | (mt_buffer[i + 1] & 0x7FFFFFFF);
                    mt_buffer[i] = mt_buffer[i - (624 - 397)] ^ (s >> 1) ^ ((s & 1) *
0x9908B0DF);
                }

                s = (mt_buffer[623] & 0x80000000) | (mt_buffer[0] & 0x7FFFFFFF);
                mt_buffer[623] = mt_buffer[396] ^ (s >> 1) ^ ((s & 1) * 0x9908B0DF);

                s ^= (s >> 11);
                s ^= (s << 7) & 0x9D2C5680;
                s ^= (s << 15) & 0xEFC60000;
                s ^= (s >> 18);
            }
            return mt_buffer[mt_index++];
        }
    }
}
```

Mersenne Twister Tools Class:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{
    public class RandTool
    {
        MT RandNumGen;

        public RandTool(MT m)
        {
            RandNumGen = m;
        }

        public int RandBetween(int lowerBound, int upperBound)
        {
            if (lowerBound > upperBound)
            {
                int x = lowerBound;
                lowerBound = upperBound;
                upperBound = x;
            }
            int randNum;
            randNum = (int)(RandNumGen.Random());
            randNum = randNum < 0 ? -randNum: randNum;
            randNum %= upperBound - lowerBound;
            randNum += lowerBound;
            return randNum;

        }

        public void RandPermutation(object[] theArray)
        {
            for (int arrayPermute = 0; arrayPermute < theArray.Length - 1; arrayPermute++)
            {
                object i = theArray[arrayPermute];
                int j = RandBetween(theArray.GetLowerBound(0), theArray.GetUpperBound(0));
                theArray[arrayPermute] = theArray[j];
                theArray[j]  = i;
            }
        }
```

```
        public double RandDouble()
        {
            double randNum = (-(double)RandNumGen.Random() / Int32.MinValue);
            return randNum;
        }
    }
}
```

ResultsHandler Class:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{

    public class ResultsHandler
    {
        public enum StrategyNames { ETHNOCENTRIC = 0, EGOIST, ALTRUIST, COSMOPOLITAN }
        public double[] stdevp;
        public double[] stderr;
        public double[] average;
        public int[,] strategyCounter;
        public int[] peopleCounter;


       public ResultsHandler( Parameters p, int run )
       {
           strategyCounter = new int[4, p.runNumber];
           peopleCounter = new int[p.runNumber];
           peopleCounter[run] = 0;
           strategyCounter[(int)StrategyNames.ALTRUIST, run] = 0;
           strategyCounter[(int)StrategyNames.COSMOPOLITAN, run] = 0;
           strategyCounter[(int)StrategyNames.EGOIST, run] = 0;
           strategyCounter[(int)StrategyNames.ETHNOCENTRIC, run] = 0;
       }

       public void countResults(Population pop, Parameters p, int run)
           //This class determines the strategy of the agents and then stores the data in an
array.
        {

            for (int i = 0; i < p.arraySize; i++)
                for (int e = 0; e < p.arraySize; e++)
                {
                    if (pop.people[i, e] != null)
                    {

                        peopleCounter[run]++;

                        if (pop.people[i, e].strategySame == Population.Strategies.COOPERATE)
                        {
                            if (pop.people[i, e].strategyDifferent ==
Population.Strategies.COOPERATE)
                            {
                                pop.people[i, e].strategyName = StrategyNames.ALTRUIST;
                                strategyCounter[(int)StrategyNames.ALTRUIST, run]++;
                            }
                            if (pop.people[i, e].strategyDifferent ==
Population.Strategies.DEFECT)
                            {
                                pop.people[i, e].strategyName = StrategyNames.ETHNOCENTRIC;
                                strategyCounter[(int)StrategyNames.ETHNOCENTRIC, run]++;
                            }
                        }

                        if (pop.people[i, e].strategySame == Population.Strategies.DEFECT)
                        {
```

```
                            if (pop.people[i, e].strategyDifferent ==
Population.Strategies.COOPERATE)
                            {
                                pop.people[i, e].strategyName = StrategyNames.COSMOPOLITAN;
                                strategyCounter[(int)StrategyNames.COSMOPOLITAN, run]++;
                            }
                            if (pop.people[i, e].strategyDifferent ==
Population.Strategies.DEFECT)
                            {
                                pop.people[i, e].strategyName = StrategyNames.EGOIST;
                                strategyCounter[(int)StrategyNames.EGOIST, run]++;
                            }
                        }
                    }
                }
        }
        public static void StandardDeviation(ResultsHandler rh, Parameters p)
            //This class calculates the standard deviation and the standard error.
        {
            rh.stdevp = new double[4];
            rh.stderr = new double[4];
            rh.average = new double[4];

            for (int i = 0; i < 4; i++)
            {
                double fraction;
                double total = 0;
                double totalSquars = 0;

                for (int j = 0; j < p.runNumber; j++)
                {

                    fraction = ((double)rh.strategyCounter[i, j]) / rh.peopleCounter[j];
                    total += fraction;
                    totalSquars += fraction * fraction;
                }
                rh.average[i] = total / p.runNumber * 100;
                rh.stdevp[i] = Math.Sqrt(totalSquars / p.runNumber - ((total * total) /
(p.runNumber * p.runNumber))) * 100;
                rh.stderr[i] = rh.stdevp[i] / Math.Sqrt(p.runNumber - 1);
            }
        }
    }
}
```

Results Class:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace EvoEthnocentrism
{
    public partial class Results : Form
    {
        public Results(ResultsHandler rh)
        {
            InitializeComponent();
            PrintResults(rh);
        }

        private void PrintResults(ResultsHandler rh)
        {
            label12.Text = Math.Round(rh.average[(int)ResultsHandler.StrategyNames.ETHNOCENTRIC],
1).ToString() + "%";
```

```
            label13.Text = Math.Round(rh.stdevp[(int)ResultsHandler.StrategyNames.ETHNOCENTRIC],
1).ToString() + "%";
            label14.Text = Math.Round(rh.stderr[(int)ResultsHandler.StrategyNames.ETHNOCENTRIC],
1).ToString() + "%";
            label15.Text = Math.Round(rh.average[(int)ResultsHandler.StrategyNames.ALTRUIST],
1).ToString() + "%";
            label16.Text = Math.Round(rh.stdevp[(int)ResultsHandler.StrategyNames.ALTRUIST],
1).ToString() + "%";
            label17.Text = Math.Round(rh.stderr[(int)ResultsHandler.StrategyNames.ALTRUIST],
1).ToString() + "%";
            label18.Text = Math.Round(rh.average[(int)ResultsHandler.StrategyNames.EGOIST],
1).ToString() + "%";
            label19.Text = Math.Round(rh.stdevp[(int)ResultsHandler.StrategyNames.EGOIST],
1).ToString() + "%";
            label20.Text = Math.Round(rh.stderr[(int)ResultsHandler.StrategyNames.EGOIST],
1).ToString() + "%";
            label21.Text = Math.Round(rh.average[(int)ResultsHandler.StrategyNames.COSMOPOLITAN],
1).ToString() + "%";
            label22.Text = Math.Round(rh.stdevp[(int)ResultsHandler.StrategyNames.COSMOPOLITAN],
1).ToString() + "%";
            label23.Text = Math.Round(rh.stderr[(int)ResultsHandler.StrategyNames.COSMOPOLITAN],
1).ToString() + "%";
        }

        private void Results_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Graphic Class:

```
using System;
using System.Windows.Forms;
using System.Drawing;


namespace EvoEthnocentrism
{
    public partial class Graphic : Form
    {
        Population pop;
        Parameters p;

        public Graphic(Parameters p, Population pp)
        {
            InitializeComponent();
            pop = pp;
            this.p = p;
            this.ClientSize = new System.Drawing.Size(p.arraySize * 10, p.arraySize * 10);
        }


        private void Form2_FormClosing(Object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            this.Hide();
        }


                private void drawObject(Population.Person person, Graphics draw, SolidBrush
brushColor, Pen penColor, int locationi, int locationj)
        {

            if (person.strategyName == ResultsHandler.StrategyNames.ALTRUIST)
            {
                draw.DrawEllipse(penColor, 10 * locationi, 10 * locationj, 6, 6);
            }
            if (person.strategyName == ResultsHandler.StrategyNames.COSMOPOLITAN)
            {
```

```
                draw.FillEllipse(brushColor, 10 * locationi, 10 * locationj, 6, 6);
            }
            if (person.strategyName == ResultsHandler.StrategyNames.EGOIST)
            {
                draw.FillRectangle(brushColor, 10 * locationi, 10 * locationj, 9, 9);
                        }
            if (person.strategyName == ResultsHandler.StrategyNames.ETHNOCENTRIC)
            {

                draw.FillRectangle(brushColor, 10 * locationi, 10 * locationj, 9, 9);
                draw.FillEllipse(Brushes.Black, 10 * locationi + 1, 10 * locationj + 1, 6, 6);
            }
        }
        Pen penColor = new Pen(Brushes.Blue, 1f);

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics draw = e.Graphics;
            draw.Clear(Color.Black);


            for (int i = 0; i < p.arraySize; i++)
                for (int j = 0; j < p.arraySize; j++)
                    if ( pop.people[i, j] != null)
                        switch (pop.people[i, j].ethnicity)
                        {


                            case 0:
                                SolidBrush brushColor = new SolidBrush(Color.FromArgb(255, 255,
0, 0));
                                Pen penColor = new Pen(Color.FromArgb(255, 255, 0, 0), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                            case 1:
                                brushColor = new SolidBrush(Color.FromArgb(255, 0, 255, 0));
                                penColor = new Pen(Color.FromArgb(255, 0, 255, 0), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                            case 2:
                                brushColor = new SolidBrush(Color.FromArgb(0, 0, 255));
                                penColor = new Pen(Color.FromArgb(1, 0, 255), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                            case 3:
                                brushColor = new SolidBrush(Color.FromArgb(255, 255, 255, 255));
                                penColor = new Pen(Color.FromArgb(255, 255, 255), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                        }
        }

        protected override void OnPaintBackground(PaintEventArgs e)
        {
        }

        private void Graphic_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Graph Class:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

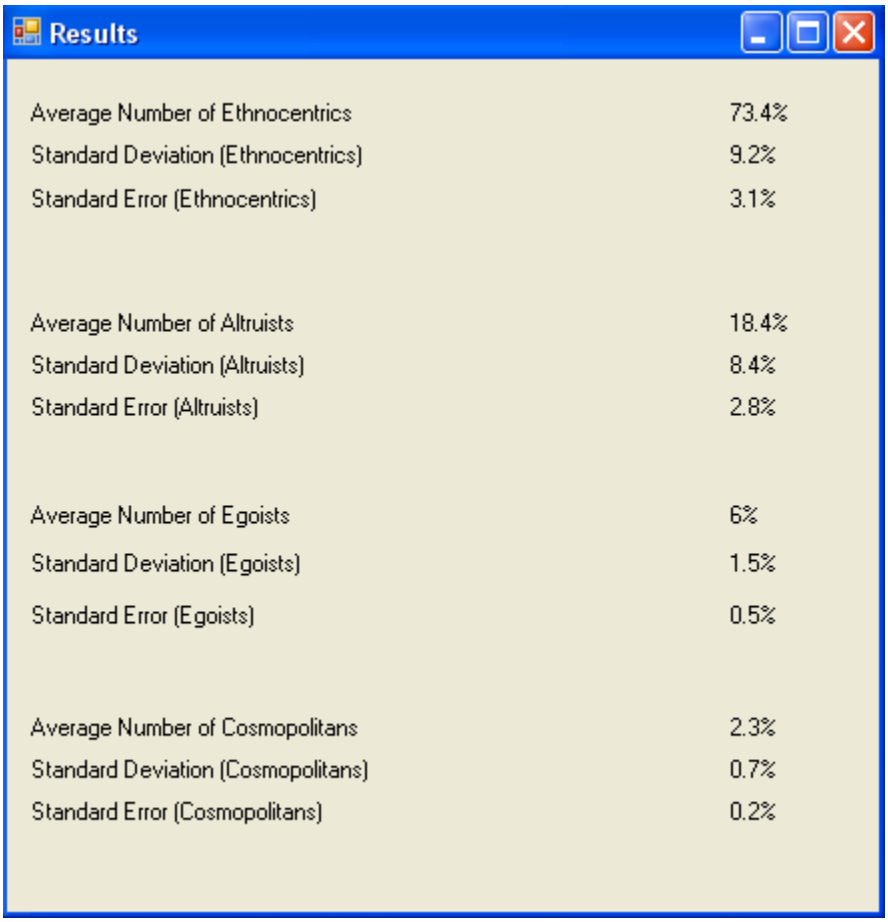
```
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;

namespace EvoEthnocentrism
{
    public partial class Graph : Form
    {
        //imported information
        Parameters p;
        Population pop;

        //x cordinates
        int time = 0;

        //spaceing
        const float spaceOnBottom = 12;
        const float SpaceOnBottom2x = 2 * spaceOnBottom + 1;

        //point for lines
        PointF f;

        //text font
        Font drawFont = new Font("Arial", 8);

        //backround color
        SolidBrush drawBrush = new SolidBrush(Color.Black);

        //different pen colors
        Pen blackPen = new Pen(Brushes.Black, 1f);
        Pen greenPen = new Pen(Brushes.Green, 1f);
        Pen redPen = new Pen(Brushes.Red, 1f);
        Pen bluePen = new Pen(Brushes.Blue, 1f);

        //lines
        GraphicsPath a = new GraphicsPath();
        GraphicsPath b = new GraphicsPath();
        GraphicsPath c = new GraphicsPath();
        GraphicsPath d = new GraphicsPath();

        public Graph(Parameters p, Population pop)
        {
            this.p = p;
            this.pop = pop;
            InitializeComponent();
            ResizeRedraw = true;
        }

        private void Form3_FormClosing(Object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            this.Hide();
        }

        protected override void OnPaint(PaintEventArgs paint)
        {
            //creates data set
            ResultsHandler rh = new ResultsHandler(p, p.runNumber - 1);
            rh.countResults(pop, p, p.runNumber - 1);

            //creates basic unfilled graph
            string xlabel = "time(" + time + ")";
            Graphics table = paint.Graphics;
            //create basic unfilled graph
            table.Clear( Color.White );
            table.DrawRectangle( blackPen, spaceOnBottom, spaceOnBottom, ClientSize.Width -
SpaceOnBottom2x, ClientSize.Height - SpaceOnBottom2x );
            table.DrawString( "Population", drawFont, drawBrush, 0, 0 );
            table.DrawString( "0,0", drawFont, drawBrush, 0, ClientSize.Height - spaceOnBottom -
1 );
```

```
            table.DrawString( xlabel, drawFont, drawBrush, ClientSize.Width - xlabel.Length * 8,
ClientSize.Height - spaceOnBottom - 1 );

            //moves the drawing area
            table.TranslateTransform( spaceOnBottom, ClientSize.Height - spaceOnBottom - 1 );

            //increments time
            if (time != p.runLength)
                time++;

            //scales graph
            float xTransform = Math.Min( 1f, (float)ClientSize.Width - SpaceOnBottom2x / time );
            float yTransform = -(float)(ClientSize.Height - SpaceOnBottom2x) /
rh.peopleCounter[p.runNumber - 1];
            paint.Graphics.ScaleTransform( xTransform, yTransform );

            //adds points to lines

            f.X = time;
            f.Y = rh.strategyCounter[(int)ResultsHandler.StrategyNames.ALTRUIST, p.runNumber -
1];
            a.AddLine(f, f);
            f.Y = rh.strategyCounter[(int)ResultsHandler.StrategyNames.COSMOPOLITAN, p.runNumber
- 1];
            b.AddLine(f, f);
            f.Y = rh.strategyCounter[(int)ResultsHandler.StrategyNames.EGOIST, p.runNumber - 1];
            c.AddLine(f, f);
            f.Y = rh.strategyCounter[(int)ResultsHandler.StrategyNames.ETHNOCENTRIC, p.runNumber
- 1];
            d.AddLine(f, f);

            //Draws lines
            table.DrawPath( blackPen, a );
            table.DrawPath( greenPen, b );
            table.DrawPath( redPen, c );
            table.DrawPath( bluePen, d );
        }
    }
}
```

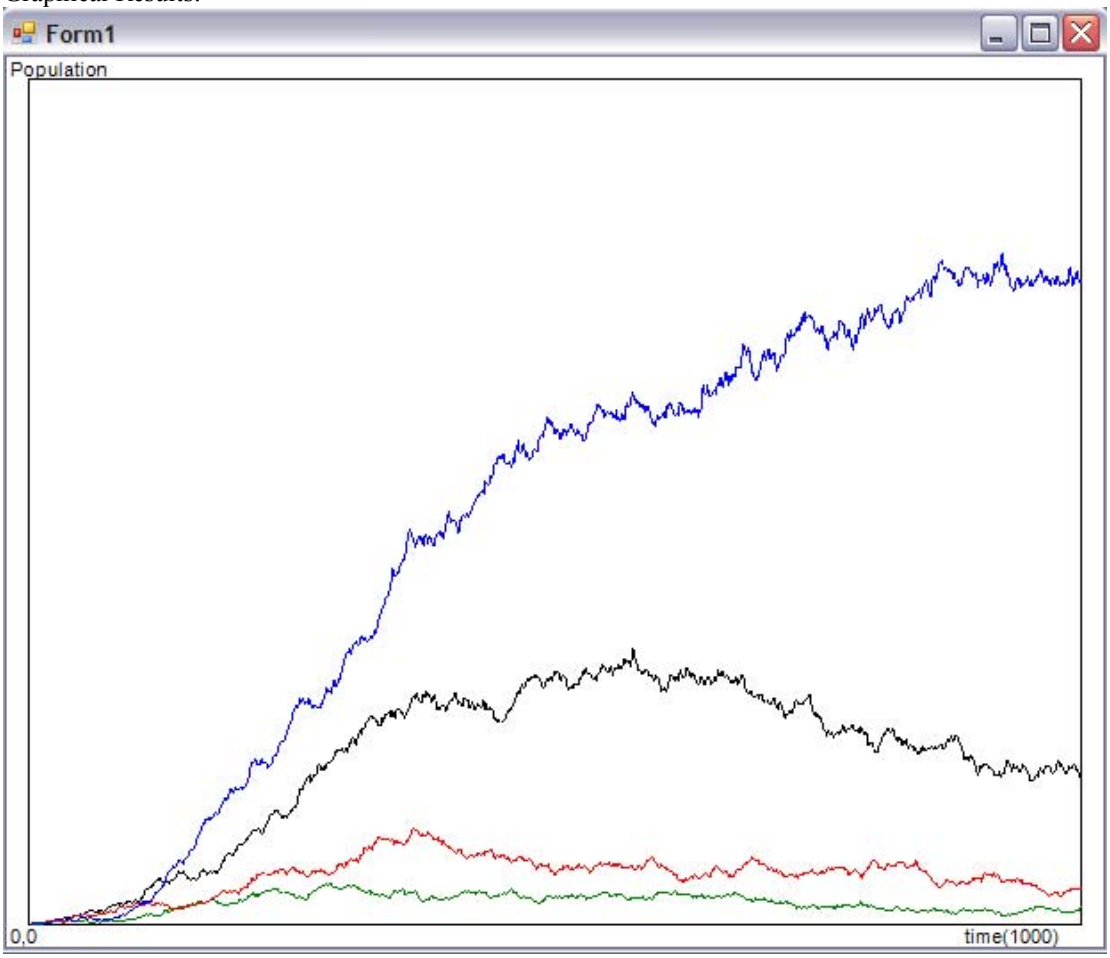
Appendix II:

Results of a Normal Run with Seed 0

Numerical Results:



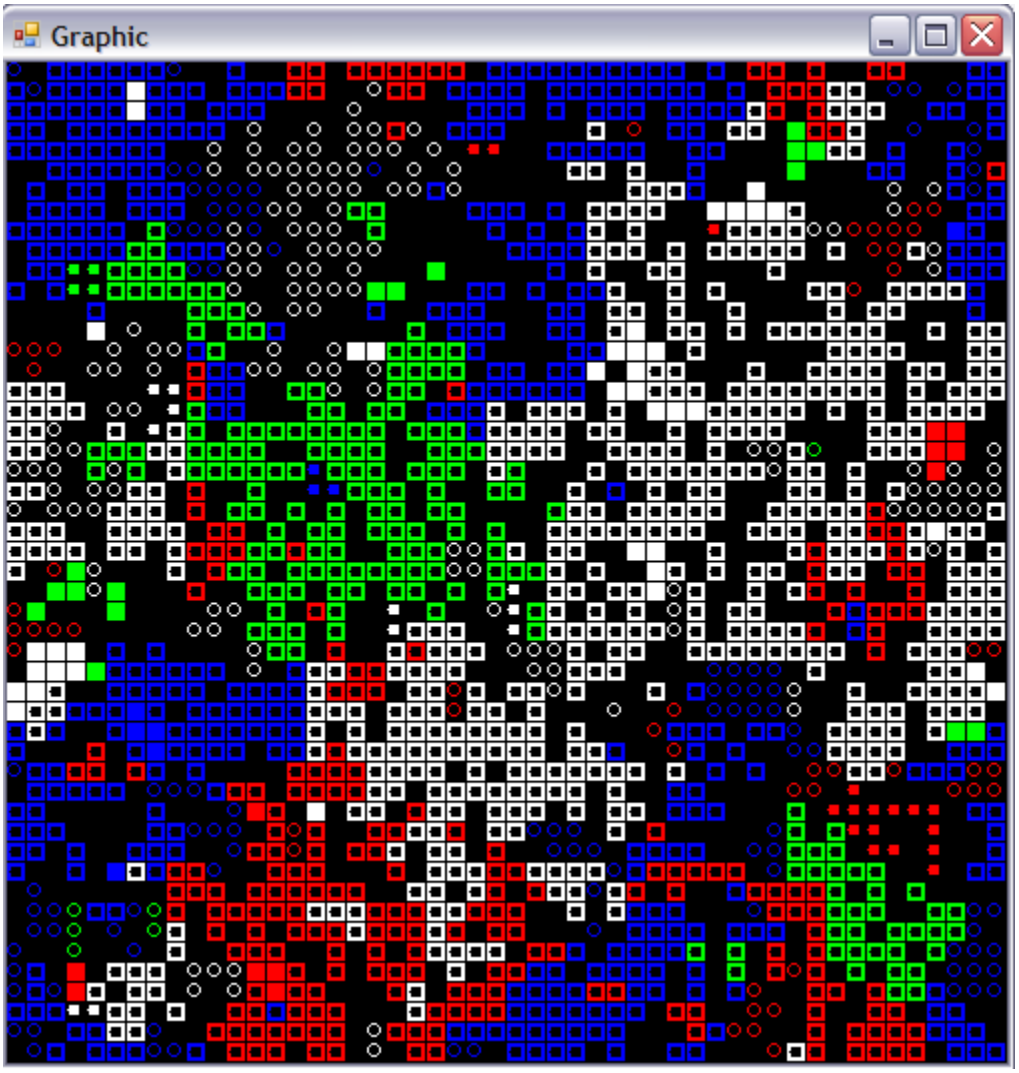
Graphical Results:



Key:

Ethnocentric Strategy
Altruist Strategy
Egoist Strategy
Cosmopolitan Strategy

Graphic:



Key:

Ethnocentrics

Altruists

Egoists

Cosmopolitans

(Color represents different ethnicities)

Appendix III:

River Interaction Modifications

***Note: The only classes that are modified are the Interaction and Graphic classes.

Interaction Class:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{
    class Interaction
    {
        static void Play(Population.Person self, Population.Person other, Parameters p)
            //This code tells the agents how to interact with their neighbors.
            //They compare their strategies with their neighbor's and either cooperate or defect
based on their strategies.
            //Their chance to reproduce is then modified.
        {
            if (self.ethnicity == other.ethnicity)
            {
                if (other.strategySame == Population.Strategies.COOPERATE)
                    self.chanceOfReproducing += p.benefit;

                if (self.strategySame == Population.Strategies.COOPERATE)
                    self.chanceOfReproducing -= p.cost;
            }
            else
            {
                if (other.strategyDifferent == Population.Strategies.COOPERATE)
                    self.chanceOfReproducing += p.benefit;

                if (self.strategyDifferent == Population.Strategies.COOPERATE)
                    self.chanceOfReproducing -= p.cost;
            }

        }
        public static void interaction(Population pop, Parameters p)
            // This portion of the code ensures that everybody will interact only once in each
array.
        {
            for (int i = 0; i < p.arraySize; i++)
            {
                for (int e = 0; e < p.arraySize; e++)
                {
                    if (pop.people[i, e] != null)
                    {
                        if ( pop.people[(i + 1) % p.arraySize, e] != null )
                            Play( pop.people[i, e], pop.people[(i + 1) % p.arraySize, e], p );
                        if ( pop.people[(i + (p.arraySize - 1)) % p.arraySize, e] != null )
                            Play( pop.people[i, e], pop.people[(i + (p.arraySize - 1)) %
p.arraySize, e], p );
                        if ( pop.people[i,  (e + 1) % p.arraySize] != null )
                           Play( pop.people[i, e], pop.people[i, (e + 1) % p.arraySize], p );
                        if ( pop.people[i, (e + (p.arraySize - 1)) % p.arraySize] != null )
                           Play( pop.people[i, e], pop.people[i, (e + (p.arraySize - 1)) %
p.arraySize], p );
                       if (i == 2)
                           riverInteraction(i, e, p, pop);
                       if (i == 7)
                           riverInteraction(i, e, p, pop);
                       if (i == 16)
                           riverInteraction(i, e, p, pop);
                       if (i == 22)
                           riverInteraction(i, e, p, pop);
                       if (i == 27)
                           riverInteraction(i, e, p, pop);
                    }
```

```
                }

            }

        }

        private static void riverInteraction(int x, int y, Parameters p, Population pop)
            // This code allows agents to interact along "rivers".
            //Rivers allow non-adjacent agents on the river to interact as if they were adjacent.
        {
            for (int i = 0; i < p.arraySize; i++)
            {
                if (i != y)
                {
                    if (pop.people[x, i] != null)
                    {
                        Play(pop.people[x, y], pop.people[x, i], p);
                        if (pop.people[x + 1, y] != null)
                            Play(pop.people[x + 1, y], pop.people[x, i], p);
                    }

                }

                if (pop.people[x + 1, i] != null)
                    Play(pop.people[x, y], pop.people[x + 1, i], p);
                if (pop.people[x + 1, y] != null)
                    if (pop.people[x, i] != null)
                        Play(pop.people[x + 1, y], pop.people[x, i], p);
            }
        }
    }
}
```

Graphic Class:

```
using System;
using System.Windows.Forms;
using System.Drawing;


namespace EvoEthnocentrism
{
    public partial class Graphic : Form
    {
        Population pop;
        Parameters p;

        public Graphic(Parameters p, Population pp)
        {
            InitializeComponent();
            pop = pp;
            this.p = p;
            this.ClientSize = new System.Drawing.Size(p.arraySize * 10, p.arraySize * 10);
        }


        private void Form2_FormClosing(Object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            this.Hide();
        }


        private void drawObject(Population.Person person, Graphics draw, SolidBrush brushColor,
Pen penColor, int locationi, int locationj)
        {

            if (person.strategyName == ResultsHandler.StrategyNames.ALTRUIST)
            {
                draw.DrawEllipse(penColor, 10 * locationi, 10 * locationj, 6, 6);
            }
```

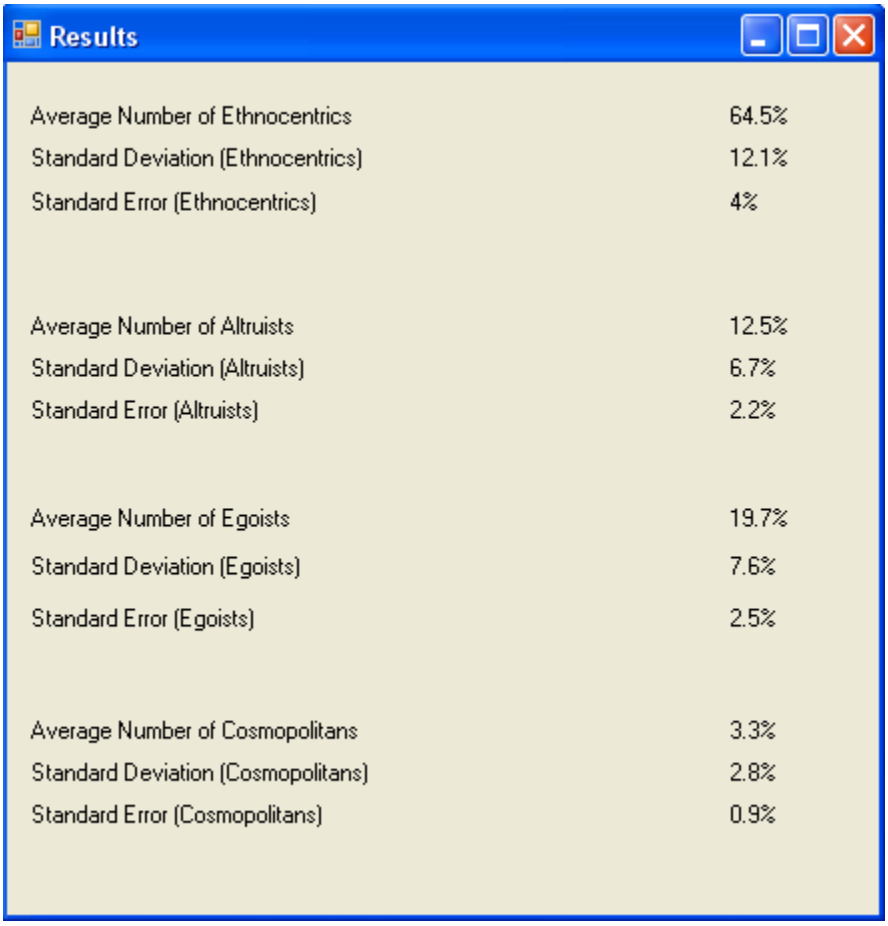
```
            if (person.strategyName == ResultsHandler.StrategyNames.COSMOPOLITAN)
            {
                draw.FillEllipse(brushColor, 10 * locationi, 10 * locationj, 6, 6);
            }
            if (person.strategyName == ResultsHandler.StrategyNames.EGOIST)
            {
                draw.FillRectangle(brushColor, 10 * locationi, 10 * locationj, 9, 9);
                        }
            if (person.strategyName == ResultsHandler.StrategyNames.ETHNOCENTRIC)
            {

                draw.FillRectangle(brushColor, 10 * locationi, 10 * locationj, 9, 9);
                draw.FillEllipse(Brushes.Black, 10 * locationi + 1, 10 * locationj + 1, 6, 6);
            }

        }
        Pen penColor = new Pen(Brushes.Blue, 1f);

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics draw = e.Graphics;
            draw.Clear(Color.Black);
            Pen orange = new Pen(Brushes.Orange, 1f);



            for (int i = 0; i < p.arraySize; i++)
                for (int j = 0; j < p.arraySize; j++)
                    if ( pop.people[i, j] != null)
                        switch (pop.people[i, j].ethnicity)
                        {


                            case 0:
                                SolidBrush brushColor = new SolidBrush(Color.FromArgb(255, 255,
0, 0));
                                Pen penColor = new Pen(Color.FromArgb(255, 255, 0, 0), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                            case 1:
                                brushColor = new SolidBrush(Color.FromArgb(255, 0, 255, 0));
                                penColor = new Pen(Color.FromArgb(255, 0, 255, 0), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                            case 2:
                                brushColor = new SolidBrush(Color.FromArgb(0, 0, 255));
                                penColor = new Pen(Color.FromArgb(1, 0, 255), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                            case 3:
                                brushColor = new SolidBrush(Color.FromArgb(255, 255, 255, 255));
                                penColor = new Pen(Color.FromArgb(255, 255, 255), 1f);
                                drawObject(pop.people[i, j], draw, brushColor, penColor, i, j);
                                break;
                        }

            e.Graphics.DrawLine( orange, 2 * 10, 0, 2 * 10, p.arraySize * 10 );
            e.Graphics.DrawLine(orange, 7 * 10, 0, 7 * 10, p.arraySize * 10);
            e.Graphics.DrawLine(orange, 16 * 10, 0, 16 * 10, p.arraySize * 10);
            e.Graphics.DrawLine(orange, 22 * 10, 0, 22 * 10, p.arraySize * 10);
            e.Graphics.DrawLine(orange, 27 * 10, 0, 27 * 10, p.arraySize * 10);

        }

        protected override void OnPaintBackground(PaintEventArgs e)
        {
        }

        private void Graphic_Load(object sender, EventArgs e)
        {
```

```
        }
    }
}
```

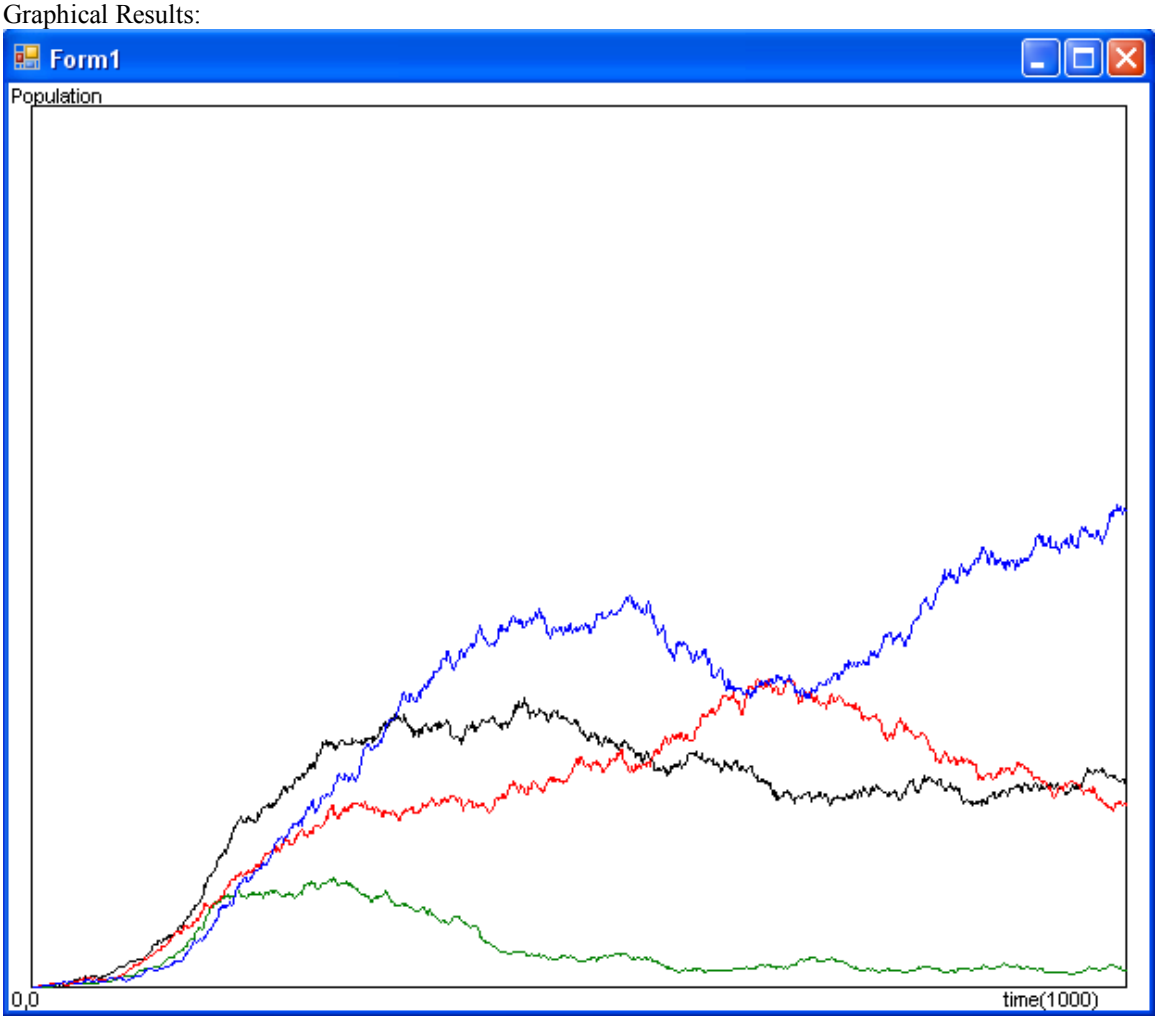
Appendix IV:

Results of River Interaction Run with Seed 0

Numerical Results:



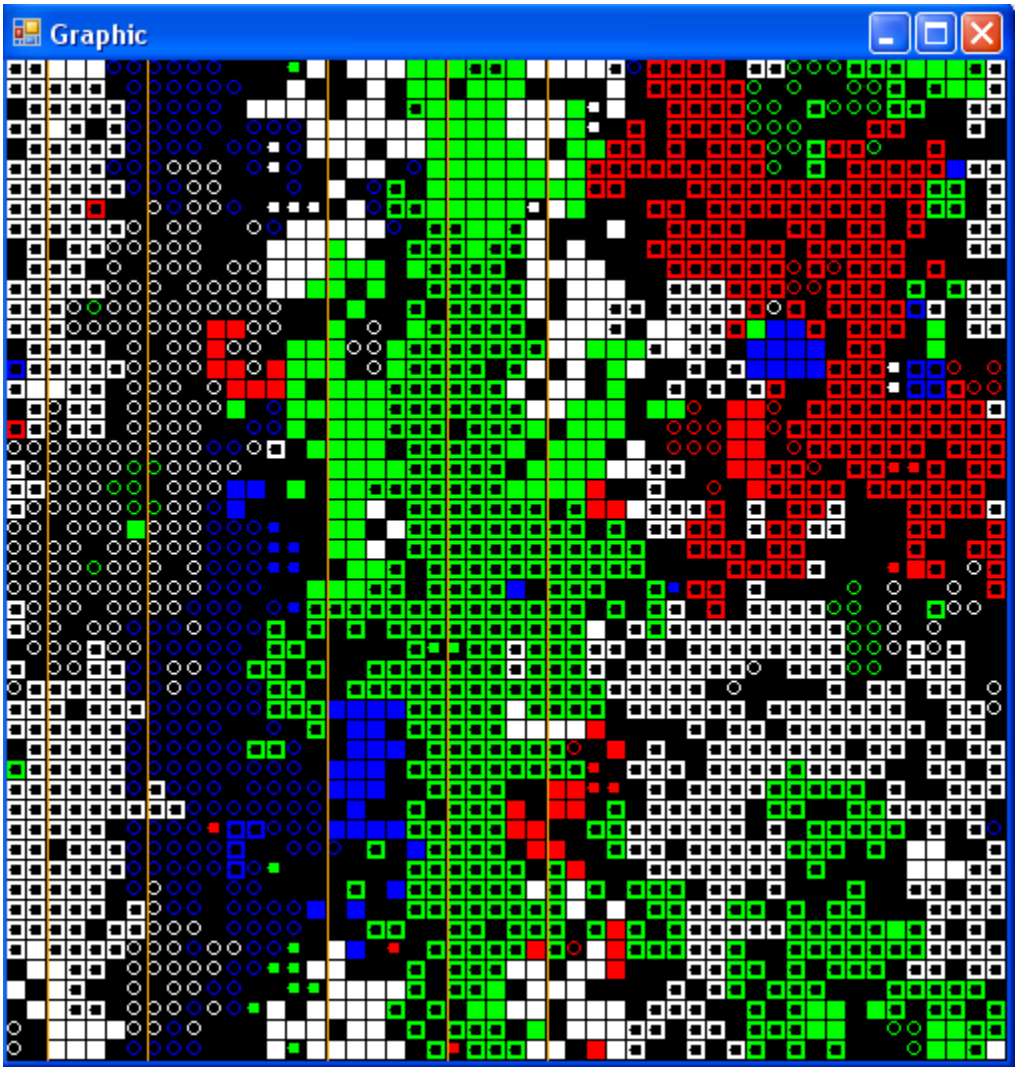
Graphical Results:



Key:

Ethnocentric Strategy
Altruist Strategy
Egoist Strategy
Cosmopolitan Strategy

Graphic:



Key:

Ethnocentrics

Altruists

Egoists

Cosmopolitans

(Color represents different ethnicities)

Appendix V:

Code for Ethnicity Distinction Modification with Excel Output

***Note: Only classes in which modifications are made are displayed below. Modifications are highlighted.

Program Class:

```
using System;
using System.Threading;
using System.Collections.Generic;
using System.Windows.Forms;

namespace EvoEthnocentrism
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Parameters p = new Parameters();
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Input(p));

            int run = 0;
            ResultsHandler rh = new ResultsHandler( p, run );
            ResultsHandler graphData = new ResultsHandler(p, run);
            Life l = new Life();
            Interaction interact = new Interaction();
            DataOutput excel = new DataOutput( p, rh );
            Graph[] ethnicGraphs = new Graph[p.ethnicityNumber];

            for ( run = 0; run < p.runNumber; run++ )
            {
                Population pop = new Population( p );
                for ( int i = 0; i < p.ethnicityNumber; i++ )
                {
                    ethnicGraphs[i] = new Graph( p, pop, graphData, i );

                }
                Graphic graphic = new Graphic( p, pop);
                if (p.runNumber - 1 == run)
                {
                    for (int i = 0; i < p.ethnicityNumber; i++)
                    {
                        ethnicGraphs[i].Show();
                    }
                    graphic.Show();
                }

                for (int time = 1; time <= p.runLength; time++)
                {
                    l.immigration(ref pop, p, run);
                    Interaction.interaction(ref pop, p);
                    l.reproduction(ref pop, p);
                    l.death(ref pop, p);
                    if (time >= p.runLength - 100)
                        rh.countResults(ref pop, p, run);
                    if (p.runNumber - 1 == run)
                    {
                        graphData.countResults( ref pop, p, run );
                        for (int i = 0; i < p.ethnicityNumber; i++)
                        {
                            ethnicGraphs[i].Activate();
                            ethnicGraphs[i].Show();
```

```
                        ethnicGraphs[i].Refresh();
                    }

                    graphic.Activate();
                    graphic.Show();
                    graphic.Refresh();
                }
            }
            p.seed++;
            Console.Write(".  ");
        }

        ResultsHandler.StandardDeviation(rh, p);
        excel.ExcelOutput();
        Application.Run(new Results(rh));
    }
  }
}
```

Population:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{
    public class Population
    {
        public RandTool Rand;
        Parameters p;
        MT RandNumGen;
        public Person[,] people;
        public enum Strategies { DEFECT = 1, COOPERATE }
        public int[,] ethnicityColor;

        public Population(Parameters p)
        {
            this.p = p;
            RandNumGen = new MT(p.seed);
            Rand = new RandTool(RandNumGen);
            people = new Person[p.arraySize, p.arraySize];
            ethnicityColor = new int[p.ethnicityNumber * p.ethnicityNumber, 3];
            for (int i = 0; i < ethnicityColor.GetLength(0); i++)
                for (int j = 0; j < 3; j++)
                    ethnicityColor[i, j] = Rand.RandBetween(0, 255);

        }


        public class Person
        {
            public Strategies[] ethnicStratagies;
            public int ethnicity;
            public double chanceOfReproducing;
        }

        public void DefaultTraits(int x, int y)
        {
            people[x, y].ethnicStrategies = new Strategies[p.ethnicityNumber];
            for (int i = 0; i < p.ethnicityNumber; i++)
                people[x, y].ethnicStrategies[i] = (Strategies)Rand.RandBetween(1, 3);
            people[x, y].chanceOfReproducing = p.ptr;
            people[x, y].ethnicity = Rand.RandBetween(0, p.ethnicityNumber);
        }

        public void Mutation( int currentx, int currenty)
        {
            if (p.mutationRate >= Rand.RandDouble())
            {
```

```
                people[currentx, currenty].ethnicity = Rand.RandBetween(0, p.ethnicityNumber);
            }
            for (int i = 0; i < p.ethnicityNumber; i++)
                if (p.mutationRate >= Rand.RandDouble())
                {
                    people[currentx, currenty].ethnicStrategies[i] =
(Strategies)Rand.RandBetween(1, 3);
                }
        }

        public void CopyTraits(Population pop, int parentx, int parenty, int offspringx, int
offspringy)
        {
            pop.people[offspringx, offspringy].ethnicStrategies = new
Strategies[p.ethnicityNumber];
            for ( int i = 0; i < p.ethnicityNumber; i++ )
                pop.people[offspringx, offspringy].ethnicStrategies[i] = pop.people[parentx,
parenty].ethnicStrategies[i];
            pop.people[offspringx, offspringy].chanceOfReproducing = p.ptr;
            pop.people[offspringx, offspringy].ethnicity = pop.people[parentx,
parenty].ethnicity;
        }
    }
}
```

Interaction:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{
    class Interaction
    {
        static void Play(Population.Person self, Population.Person other, Parameters p)
        {
            for (int i = 0; i < p.ethnicityNumber; i++)
                if (other.ethnicity == i)
                    if (self.ethnicStratagies[i] == Population.Strategies.COOPERATE)
                    {
                        self.chanceOfReproducing -= p.cost;
                        break;
                    }

            for (int i = 0; i < p.ethnicityNumber; i++)
                if (self.ethnicity == i)
                    if (other.ethnicStratagies[i] == Population.Strategies.COOPERATE)
                    {
                        self.chanceOfReproducing += p.benefit;
                        break;
                    }
        }

        public static void interaction(ref Population pop, Parameters p)
        {
            for (int i = 0; i < p.arraySize; i++)
            {
                for (int e = 0; e < p.arraySize; e++)
                {
                    if (j.people[i, e] != null)
                    {
                        if ( j.people[(i + 1) % p.arraySize, e] != null )
                            Play( j.people[i, e], j.people[(i + 1) % p.arraySize, e], p );
                        if ( j.people[(i + (p.arraySize - 1)) % p.arraySize, e] != null )
                            Play( j.people[i, e], j.people[(i + (p.arraySize - 1)) % p.arraySize,
e], p );
                        if ( j.people[i,  (e + 1) % p.arraySize] != null )
                           Play( j.people[i, e], j.people[i, (e + 1) % p.arraySize], p );
```

```
                        if ( j.people[i, (e + (p.arraySize - 1)) % p.arraySize] != null )
                           Play( j.people[i, e], j.people[i, (e + (p.arraySize - 1)) %
p.arraySize], p );
                    }
                }
            }
        }
    }
}
```

Result Handler:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EvoEthnocentrism
{

    public class ResultHandler
    {
        Parameters p;
        public enum StrategyNames { ETHNOCENTRIC = 0, EGOIST, ALTURIST, COSMOPOLITAN }
        public int[,] strategyCounter;
        public int[] peopleCounter;
        public int[] storedPeopleCounter;
        public int[] strategyArray;
        public Results[] resultData;
        public Results[,] storedResultData;
        public int[] ethnicityCounter;
        public string[] nameOrder;
        int resultCounter = 0;
        public int[] coperationStrategyCount;
        public int[,] storedcoperationStrategyCount;
        public int[,] storedEthnicityCount;


        public ResultHandler( Parameters p, int run )
        {
            this.p = p;
            strategyCounter = new int[4, p.runNumber];
            peopleCounter = new int[p.runNumber];
            peopleCounter[run] = 0;
            strategyCounter[(int)StrategyNames.ALTURIST, run] = 0;
            strategyCounter[(int)StrategyNames.COSMOPOLITAN, run] = 0;
            strategyCounter[(int)StrategyNames.EGOIST, run] = 0;
            strategyCounter[(int)StrategyNames.ETHNOCENTRIC, run] = 0;
            strategyArray = new int[p.ethnicityNumber * p.ethnicityNumber];
            resultData = new Results[p.ethnicityNumber * p.ethnicityNumber];
            storedResultData = new Results[(int)p.runLength / 100, p.ethnicityNumber *
p.ethnicityNumber];
            ethnicityCounter = new int[p.ethnicityNumber];
            nameOrder = new string[resultData.GetLength(0)];
            storedPeopleCounter = new int[(int)p.runLength / 100];
            cooperationStrategyCount = new int[p.ethnicityNumber];
            storedcoperationStrategyCount = new int[(int)p.runLength / 100, p.ethnicityNumber];
            storedEthnicityCount = new int[p.runLength / 100, p.ethnicityNumber];
        }

        public class Results
        {
            public string name;
            public int[] count;
        }

        public void recordOrder()
        {
            bool k = false;
            int f = 0;
            string onestring = "";
```

```
            for (int i = 0; i < resultData.GetLength(0); i++)
            {
                k = false;

                if (resultData[i] != null)
                    for (int j = 0; j < nameOrder.GetLength(0); j++)
                    {
                        if (nameOrder[j] != null)
                            if (string.Equals(resultData[i].name, nameOrder[j]) == true)
                            {
                                break;
                            }
                        if (nameOrder[j] == null)
                        {
                            f = j;
                            onestring = resultData[i].name;
                            k = true;
                            break;
                        }
                    }
                else
                    break;

                if (k == true)
                {
                    nameOrder[f] = onestring;

                }
            }

        }

        public void restoreOrder()
        {
            Results[] correctOrder = new Results[resultData.GetLength(0)];

            for (int i = 0; i < resultData.GetLength(0); i++)
                for (int j = 0; j < resultData.GetLength(0); j++)
                    if (resultData[j] != null)
                            if (string.Equals(resultData[j].name, nameOrder[i]) == true)
                            {
                                correctOrder[i] = resultData[j];
                                break;
                            }


            resultData = correctOrder;

        }

        public void StrategyCooperationCounter(Population.Person currentPerson)
        {
            for (int i = 0; i < p.ethnicityNumber; i++)
            {
                if (currentPerson.ethnicStrategies[i] == Population.Strategies.COOPERATE)
                    cooperationStrategyCount[i]++;
            }
        }

        public void countEthnicity(Population.Person currentPerson, int run)
        {

            if (currentPerson.ethnicStrategies[0] == Population.Strategies.COOPERATE &&
currentPerson.ethnicStrategies[1] == Population.Strategies.COOPERATE &&
currentPerson.ethnicStrategies[2] == Population.Strategies.COOPERATE &&
currentPerson.ethnicStrategies[3] == Population.Strategies.COOPERATE)
                strategyCounter[(int)StrategyNames.ALTURIST, run]++;
            if (currentPerson.ethnicStrategies[0] == Population.Strategies.DEFECT &&
currentPerson.ethnicStrategies[1] == Population.Strategies.COOPERATE &&
```

```
currentPerson.ethnicStrategies[2] == Population Strategies.COOPERATE &&
currentPerson.ethnicStrategies[3] == Population.Strategies.COOPERATE)
                strategyCounter[(int)StrategyNames.COSMOPOLITAN, run]++;
            if (currentPerson.ethnicStrategies[0] == Population.Strategies.COOPERATE &&
currentPerson.ethnicStrategies[1] == Population.Strategies.DEFECT &&
currentPerson.ethnicStrategies[2] == Population.Strategies.DEFECT &&
currentPerson.ethnicStratagies[3] == Population.Strategies.DEFECT)
                strategyCounter[(int)StrategyNames.ETHNOCENTRIC, run]++;
            if (currentPerson.ethnicStratagies[0] == Population.Strategies.DEFECT &&
currentPerson.ethnicStrategies[1] == Population.Strategies.DEFECT &&
currentPerson.ethnicStratagies[2] == Population.Strategies.DEFECT &&
currentPerson.ethnicStrategies[3] == Population.Strategies.DEFECT)
                strategyCounter[(int)StrategyNames.EGOIST, run]++;

        }

        public void specificStrategyCount(Population.Person currentPerson)
        {
            string strategyName = "";

            for (int i = 0; i < p.ethnicityNumber; i++)
                switch (currentPerson.ethnicStrategies[i])
                {
                    case Population.Strategies.DEFECT:
                        strategyName += new string('D', 1);
                        break;

                    case Population.Strategies.COOPERATE:
                        strategyName += new string('C', 1);
                        break;
                }

            for (int i = 0; i < resultData.GetLength(0); i++)
            {
                if (resultData[i] != null)
                    if ( string.Equals( resultData[i].name, strategyName ) == true )
                        {
                            resultData[i].count[currentPerson.ethnicity]++;
                            break;
                        }

                if (resultData[i] == null)
                {
                    resultData[i] = new Results();
                    resultData[i].name = strategyName;
                    resultData[i].count = new int[p.ethnicityNumber];
                    resultData[i].count[currentPerson.ethnicity]++;
                    break;
                }
        }

        }

        public void countResults( ref Population pop, Parameters p, int run, bool eraseData )
        {
            if (eraseData == true)
            {
                resultData = null;
                resultData = new Results[p.ethnicityNumber * p.ethnicityNumber];
                peopleCounter[0] = 0;
                for (int i = 0; i < p.ethnicityNumber; i++)
                    ethnicityCounter[i] = 0;
            }
            resultCounter++;

            for ( int x = 0; x < p.arraySize; x++ )
                for ( int y = 0; y < p.arraySize; y++ )
                    if ( j.people[x, y] != null )
                    {
                        peopleCounter[run]++;
                        specificStategyCount( pop.people[x, y] );
```

```
                        for (int k = 0; k < p.ethnicityNumber; k++)
                            if ( pop.people[x, y].ethnicity == k )
                            {
                                ethnicityCounter[k]++;
                                break;
                            }
                        if ( eraseData == true )
                        {
                            if (resultCounter % 100 == 0)
                                StrategyCooperationCounter(pop.people[x, y]);
                        }
                        else
                            StrategyCooperationCounter(pop.people[x, y]);

                    }

            recordOrder();
            restoreOrder();

            if (eraseData == true)
            {
                if (resultCounter != 0)
                    if (resultCounter % 100 == 0)
                    {
                        storedPeopleCounter[resultCounter / 100 - 1] = peopleCounter[0];

                        for (int k = 0; k < resultData.GetLength(0); k++)
                            storedResultData[resultCounter / 100 - 1, k] = resultData[k];
                        for (int k = 0; k < p.ethnicityNumber; k++)
                        {
                            storedcooperationStrategyCount[resultCounter / 100 - 1, k] =
cooperationStrategyCount[k];
                            cooperationStrategyCount[k] = 0;
                            storedEthnicityCount[resultCounter / 100 - 1, k] =
ethnicityCounter[k];
                        }
                    }
            }
        }
    }
}
```

Graphic Class:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;

namespace EvoEthnocentrism
{
    public partial class Graph : Form
    {
        Parameters p;
        Population pop;
        ResultHandler data;
        GraphicsPath[] strategyLines;
        int displayedEthnicityNum;

        int time = 0;
        const int borderShift = 12;
        const int BorderShift2x = 2 * borderShift + 1;
        PointF f;
        Font drawFont = new Font("Arial", 8);
        SolidBrush drawBrush = new SolidBrush(Color.Black);
        Pen[] strategyPens;
```

```
        SolidBrush[] strategyBrushes;
        Pen blackPen = new Pen(Brushes.Black, 1f);
        int keySize;


        public Graph( Parameters p, Population pop, ResultHandler data, int DispEthnNum )
        {
            this.p = p;
            this.pop = pop;
            this.data = data;
            displayedEthnicityNum = DispEthnNum;
            InitializeComponent();
            ResizeRedraw = true;
            strategyLines = new GraphicsPath[p.ethnicityNumber * p.ethnicityNumber];
            strategyPens = new Pen[p.ethnicityNumber * p.ethnicityNumber];
            strategyBrushes = new SolidBrush[p.ethnicityNumber * p.ethnicityNumber];
            for (int i = 0; i < strategyPens.GetLength(0); i++ )
            {
                strategyPens[i] = new Pen(Color.FromArgb(pop.ethnicityColor[i, 0],
pop.ethnicityColor[i, 1], pop.ethnicityColor[i, 2]), 1f);
                strategyBrushes[i] = new SolidBrush( Color.FromArgb (pop.ethnicityColor[i, 0],
pop.ethnicityColor[i, 1], pop.ethnicityColor[i, 2]));

            }
            this.Text = "Graph of Ethnicity " + (DispEthnNum + 1);
            keySize = p.ethnicityNumber * 12 + 40;
        }



        private void Form3_FormClosing(Object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            this.Hide();
        }

        protected override void OnPaint(PaintEventArgs paint)
        {
            string xlabel = "time(" + time + ")";
            Graphics table = paint.Graphics;
            //create basic unfilled graph
            table.Clear(Color.White);
            table.DrawRectangle(blackPen, borderShift, borderShift, ClientSize.Width -
BorderShift2x - keySize, ClientSize.Height - BorderShift2x);
            table.DrawString("Population", drawFont, drawBrush, 0, 0);
            table.DrawString("0,0", drawFont, drawBrush, 0, ClientSize.Height - borderShift - 1);
            table.DrawString(xlabel, drawFont, drawBrush, ClientSize.Width - xlabel.Length * 8 -
keySize, ClientSize.Height - borderShift - 1);

            //key
            table.TranslateTransform((float)ClientSize.Width - keySize, 0);
            paint.Graphics.ScaleTransform(1, (float)1 / (p.ethnicityNumber * p.ethnicityNumber /
15.8f)); //scale for number of lables

            for (int i = 0; i < data.resultData.GetLength(0); i++)
                if (data.nameOrder[i] != null)
                {
                    table.FillRectangle(strategyBrushes[i], 5, i * 25 + 95, 15, 15);
                    table.DrawString(data.nameOrder[i], drawFont, drawBrush, 35, i * 25 + 95);
                }

            paint.Graphics.ScaleTransform(1, (float)(p.ethnicityNumber * p.ethnicityNumber /
15.8f) / 1); //unscale


            table.TranslateTransform(borderShift - ClientSize.Width + keySize, ClientSize.Height
- borderShift - 1);  //scale graph to array size
            float xTransform = Math.Min(1f, (float)(ClientSize.Width - BorderShift2x - keySize) /
time);
            float yTransform = -(float)(ClientSize.Height - BorderShift2x) /
data.peopleCounter[p.runNumber - 1];
```

```
            paint.Graphics.ScaleTransform(xTransform, yTransform);

            if (time != p.runLength)
            {
                time++;

                for (int i = 0; i < strategyLines.GetLength(0); i++)
                {
                    if (strategyLines[i] == null)
                    {
                        strategyLines[i] = new GraphicsPath();
                        strategyLines[i].AddLine(0, 0, 0, 0);
                    }
                    if (data.resultData[i] != null)
                        f.X = time;
                    if (data.resultData[i] != null)
                        f.Y = data.resultData[i].count[displayedEthnicityNum];
                    else
                        f.Y = 0;
                    strategyLines[i].AddLine(f, f);
                }
            }

            for (int i = 0; i < strategyLines.GetLength(0); i++)
                if (strategyLines[i] != null)
                    table.DrawPath(strategyPens[i], strategyLines[i]);
        }
    }
}
```

Graph Class:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;

namespace EvoEthnocentrism
{
    public partial class Graph : Form
    {
        Parameters p;
        Population pop;
        ResultHandler data;
        GraphicsPath[] strategyLines;
        int displayedEthnicityNum;

        int time = 0;
        const int borderShift = 12;
        const int BorderShift2x = 2 * borderShift + 1;
        PointF f;
        Font drawFont = new Font("Arial", 8);
        SolidBrush drawBrush = new SolidBrush(Color.Black);
        Pen[] strategyPens;
        SolidBrush[] strategyBrushes;
        Pen blackPen = new Pen(Brushes.Black, 1f);
        int keySize;


        public Graph( Parameters p, Population pop, ResultHandler data, int DispEthnNum )
        {
            this.p = p;
            this.pop = pop;
            this.data = data;
            displayedEthnicityNum = DispEthnNum;
            InitializeComponent();
```

```
            ResizeRedraw = true;
            strategyLines = new GraphicsPath[p.ethnicityNumber * p.ethnicityNumber];
            strategyPens = new Pen[p.ethnicityNumber * p.ethnicityNumber];
            strategyBrushes = new SolidBrush[p.ethnicityNumber * p.ethnicityNumber];
            for (int i = 0; i < strategyPens.GetLength(0); i++ )
            {
                strategyPens[i] = new Pen(Color.FromArgb(pop.ethnicityColor[i, 0],
pop.ethnicityColor[i, 1], jeff.ethnicityColor[i, 2]), 1f);
                strategyBrushes[i] = new SolidBrush( Color.FromArgb (pop.ethnicityColor[i, 0],
pop.ethnicityColor[i, 1], jeff.ethnicityColor[i, 2]));

            }
            this.Text = "Graph of Ethnicity " + (DispEthnNum + 1);
            keySize = p.ethnicityNumber * 12 + 40;
        }



        private void Form3_FormClosing(Object sender, FormClosingEventArgs e)
        {
            e.Cancel = true;
            this.Hide();
        }

        protected override void OnPaint(PaintEventArgs paint)
        {
            string xlabel = "time(" + time + ")";
            Graphics table = paint.Graphics;
            //create basic unfilled graph
            table.Clear(Color.White);
            table.DrawRectangle(blackPen, borderShift, borderShift, ClientSize.Width -
BorderShift2x - keySize, ClientSize.Height - BorderShift2x);
            table.DrawString("Population", drawFont, drawBrush, 0, 0);
            table.DrawString("0,0", drawFont, drawBrush, 0, ClientSize.Height - borderShift - 1);
            table.DrawString(xlabel, drawFont, drawBrush, ClientSize.Width - xlabel.Length * 8 -
keySize, ClientSize.Height - borderShift - 1);

            //key
            table.TranslateTransform((float)ClientSize.Width - keySize, 0);
            paint.Graphics.ScaleTransform(1, (float)1 / (p.ethnicityNumber * p.ethnicityNumber /
15.8f)); //scale for number of lables

            for (int i = 0; i < data.resultData.GetLength(0); i++)
                if (data.nameOrder[i] != null)
                {
                    table.FillRectangle(strategyBrushes[i], 5, i * 25 + 95, 15, 15);
                    table.DrawString(data.nameOrder[i], drawFont, drawBrush, 35, i * 25 + 95);
                }

            paint.Graphics.ScaleTransform(1, (float)(p.ethnicityNumber * p.ethnicityNumber /
15.8f) / 1); //unscale


            table.TranslateTransform(borderShift - ClientSize.Width + keySize, ClientSize.Height
- borderShift - 1);  //scale graph to array size
            float xTransform = Math.Min(1f, (float)(ClientSize.Width - BorderShift2x - keySize) /
time);
            float yTransform = -(float)(ClientSize.Height - BorderShift2x) /
data.peopleCounter[p.runNumber - 1];
            paint.Graphics.ScaleTransform(xTransform, yTransform);

            if (time != p.runLength)
            {
                time++;

                for (int i = 0; i < strategyLines.GetLength(0); i++)
                {
                    if (strategyLines[i] == null)
                    {
                        strategyLines[i] = new GraphicsPath();
                        strategyLines[i].AddLine(0, 0, 0, 0);
```

```
                    }
                    if (data.resultData[i] != null)
                        f.X = time;
                    if (data.resultData[i] != null)
                        f.Y = data.resultData[i].count[displayedEthnicityNum];
                    else
                        f.Y = 0;
                    strategyLines[i].AddLine(f, f);
                }
            }

            for (int i = 0; i < strategyLines.GetLength(0); i++)
                if (strategyLines[i] != null)
                    table.DrawPath(strategyPens[i], strategyLines[i]);
        }
    }
}
```

Excel Class:

```
using System;
using System.Collections.Generic;
using System.Text;
using Excel = Microsoft.Office.Interop.Excel;
using System.Reflection;

namespace EvoEthnocentrism
{
    class DataOutput
    {
        Parameters p;
        ResultHandler rh;
        ResultHandler allResults;

        Excel.Application app;
        Excel._Workbook wbk;
        Excel._Worksheet wSheet;

        public DataOutput(Parameters p, ResultHandler rh, ResultHandler allResults)
        {
            this.p = p;
            this.rh = rh;
            this.allResults = allResults;
        }

        public void ExcelOutput()
        {
            //Start Excel
            app = new Excel.Application();
            app.Visible = true;

            //workbook
            wbk = (Excel._Workbook)(app.Workbooks.Add(Missing.Value));
            wSheet = (Excel._Worksheet)wbk.ActiveSheet;
            wSheet.Name = "Basic Results";

            detailedExcelOutput();
            detailedOutputs();
            PeramitersOutput();


            //Excel visible user control
            app.Visible = true;
            app.UserControl = true;

        }

        void detailedExcelOutput()
```

```
        {
            //Activate next sheet
            //Excel.Worksheet wSheet = (Excel.Worksheet)app.ActiveWorkbook.Sheets[2];
            //wSheet.Name = "Detailed Results";

            //Colume headers
            for (int i = 0; i < p.ethnicityNumber; i++)
            {
                wSheet.Cells[1, i + 2] = "Ethnicity";
                for (int j = 0; j < rh.resultData.GetLength(0); j++)
                    if (rh.resultData[j] != null)
                        if (rh.resultData[j].count != null)
                            wSheet.Cells[j + 2, i + 2] =
Math.Round((double)rh.resultData[j].count[i] / (double)rh.peopleCounter[0] * 100, 2) + "%";
            }
            for (int i = 0; i < rh.resultData.GetLength(0); i++)
                if (rh.resultData[i] != null)
                {
                    wSheet.Cells[i + 2, 1] = rh.resultData[i].name;
                    wSheet.Cells[i + 2, p.ethnicityNumber + 2] = calculateAvgOfStrategy(i) + "%";
                }

            wSheet.Cells[rh.resultData.GetLength(0) + 2, 1] = "Total% Cooperating w/ Ethnicity";
            wSheet.Cells[rh.resultData.GetLength(0) + 3, 1] = "Total% Defecting w/ Ethnicity";
            wSheet.Cells[rh.resultData.GetLength(0) + 4, 1] = "Ethnicity%";
            wSheet.Cells[1, p.ethnicityNumber + 2] = "Strategy Total";
            for (int i = 0; i < p.ethnicityNumber; i++)
            {
                wSheet.Cells[rh.resultData.GetLength(0) + 2, i + 2] =
Math.Round((double)rh.coperationStrategyCount[i] / (double)rh.peopleCounter[0] * 100, 2) + "%";
                wSheet.Cells[rh.resultData.GetLength(0) + 3, i + 2] = 100 -
Math.Round((double)rh.coperationStrategyCount[i] / (double)rh.peopleCounter[0] * 100, 2) + "%";
                wSheet.Cells[rh.resultData.GetLength(0) + 4, i + 2] =
Math.Round((double)rh.ethnicityCounter[i] / (double)rh.peopleCounter[0] * 100, 2) + "%";
            }


            wSheet.Cells.EntireColumn.AutoFit();
        }

        void PeramitersOutput()
        {
            //activate next sheet
            Excel.Worksheet wSheet = (Excel.Worksheet)app.ActiveWorkbook.Sheets[3];
            wSheet.Name = "Parameters";

            //set up colums / rows
            wSheet.Cells[1, 1] = "Parameters";
            wSheet.Cells[2, 1] = "array size";
            wSheet.Cells[3, 1] = "benefit";
            wSheet.Cells[4, 1] = "cost";
            wSheet.Cells[5, 1] = "death rate";
            wSheet.Cells[6, 1] = "number of Ethnicities";
            wSheet.Cells[7, 1] = "immigration Rate";
            wSheet.Cells[8, 1] = "mutation Rate";
            wSheet.Cells[9, 1] = "chance of Reproduction";
            wSheet.Cells[10, 1] = "run length";
            wSheet.Cells[11, 1] = "number of runs";
            wSheet.Cells[12, 1] = "seed";

            //fill rows
            wSheet.Cells[2, 2] = p.arraySize;
            wSheet.Cells[3, 2] = p.benefit;
            wSheet.Cells[4, 2] = p.cost;
            wSheet.Cells[5, 2] = p.deathRate;
            wSheet.Cells[6, 2] = p.ethnicityNumber;
            wSheet.Cells[7, 2] = p.immigrationRate;
            wSheet.Cells[8, 2] = p.mutationRate;
            wSheet.Cells[9, 2] = p.ptr;
            wSheet.Cells[10, 2] = p.runLength;
            wSheet.Cells[11, 2] = p.runNumber;
```

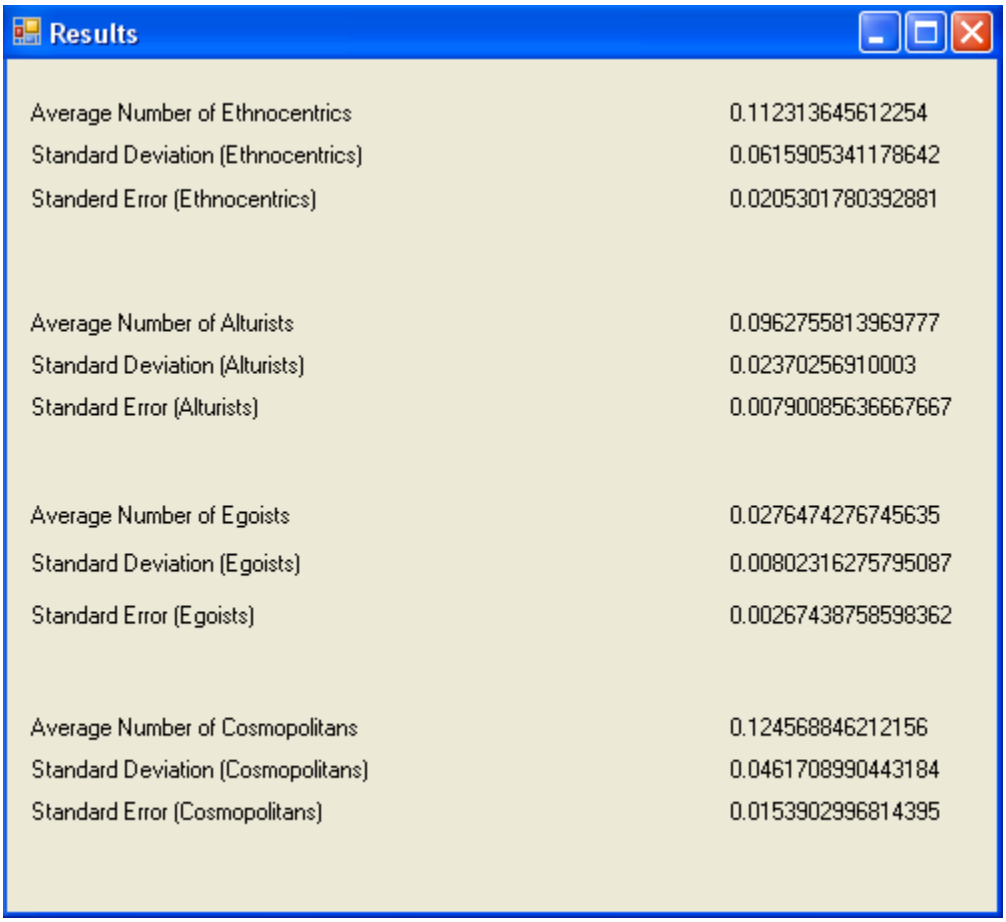
```
            wSheet.Cells[12, 2] = (p.seed - p.runNumber);

            wSheet.Cells.EntireColumn.AutoFit();
        }

        void detailedOutputs()
        {
            //activate next sheet
            Excel.Worksheet wSheet = (Excel.Worksheet)app.ActiveWorkbook.Sheets[2];
            wSheet.Name = "Data Each 100 steps";

            for (int k = 0; k < allResults.storedResultData.GetLength(0); k++)
            {
                wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + 1, 1] = "Data
for " + (k + 1) * 100 + " run";
                for (int i = 0; i < allResults.storedResultData.GetLength(1); i++)
                {
                    if (allResults.nameOrder[i] != null)
                        wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + i + 2,
1] = allResults.nameOrder[i];
                    if (allResults.storedResultData[k, i] != null)
                        wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + i + 2,
p.ethnicityNumber + 2] = calculateAvgOfStrategy(i, k) + "%";
                    else
                        wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + i + 2,
p.ethnicityNumber + 2] = 0 + "%";
                }

                wSheet.Cells[1, p.ethnicityNumber + 2] = "Strategy Total";

                for (int i = 0; i < p.ethnicityNumber; i++)
                {
                    wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + 1, i + 2] =
"Ethnicity " + (i + 1);
                    for (int j = 0; j < allResults.storedResultData.GetLength(1); j++)
                        if (allResults.storedResultData[k, j] != null)
                            wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + j +
2, i + 2] = Math.Round((double)allResults.storedResultData[k, j].count[i] /
(double)allResults.storedPeopleCounter[k] * 100, 2) + "%";
                        else
                            wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + j +
2, i + 2] = 0 + "%";
                }

                wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k +
rh.resultData.GetLength(0) + 2, 1] = "Total% Cooperating w/ Ethnicity";
                wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k +
rh.resultData.GetLength(0) + 3, 1] = "Total% Defecting w/ Ethnicity";
                wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k +
rh.resultData.GetLength(0) + 4, 1] = "Ethnicity%";
                wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k + 1,
p.ethnicityNumber + 2] = "Strategy Total";
                for (int i = 0; i < p.ethnicityNumber; i++)
                {
                    wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k +
rh.resultData.GetLength(0) + 2, i + 2] =
Math.Round((double)allResults.storedcoperationStrategyCount[k, i] /
(double)allResults.storedPeopleCounter[k] * 100, 2) + "%";
                    wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k +
rh.resultData.GetLength(0) + 3, i + 2] = 100 -
Math.Round((double)allResults.storedcoperationStrategyCount[k, i] /
(double)allResults.storedPeopleCounter[k] * 100, 2) + "%";
                    wSheet.Cells[(allResults.storedResultData.GetLength(1) + 5) * k +
rh.resultData.GetLength(0) + 4, i + 2] = Math.Round((double)allResults.storedEthnicityCount[k, i]
/ (double)allResults.storedPeopleCounter[k] * 100, 2) + "%";
                }
            }
            wSheet.Cells.EntireColumn.AutoFit();
        }

        double calculateAvgOfStrategy(int strategyNum)
```

```
        {
            double avg = 0;

            for (int i = 0; i < p.ethnicityNumber; i++)
                avg += rh.resultData[strategyNum].count[i];
            avg /= rh.peopleCounter[0];
            avg = Math.Round(avg * 100, 2);
            return avg;
        }

        double calculateAvgOfStrategy(int strategyNum, int outputNum)
        {
            double avg = 0;

            for (int i = 0; i < p.ethnicityNumber; i++)
                avg += allResults.storedResultData[outputNum, strategyNum].count[i];
            avg /= allResults.storedPeopleCounter[outputNum];
            avg = Math.Round(avg * 100, 2);
            return avg;
        }
    }
}
```

Appendix VI:

Results Ethnicity Distinction Run with Seed 0

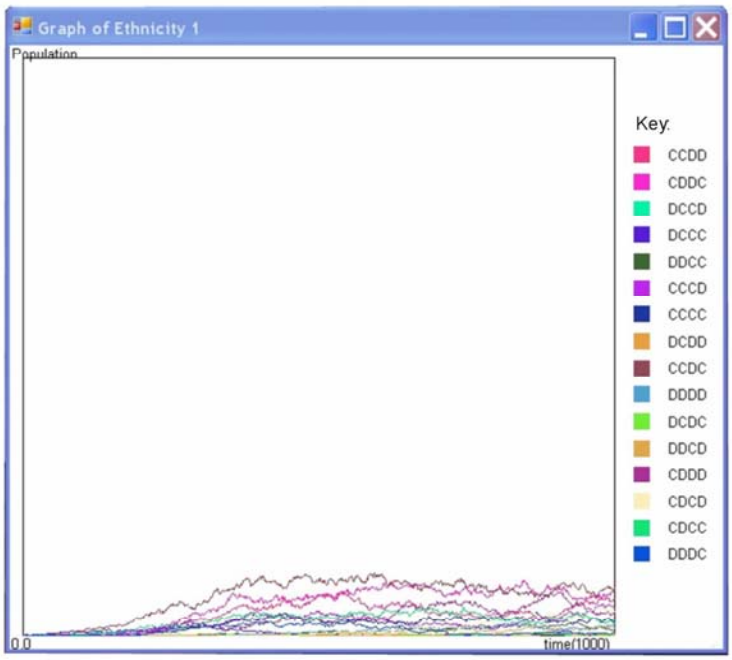
Numerical Results:

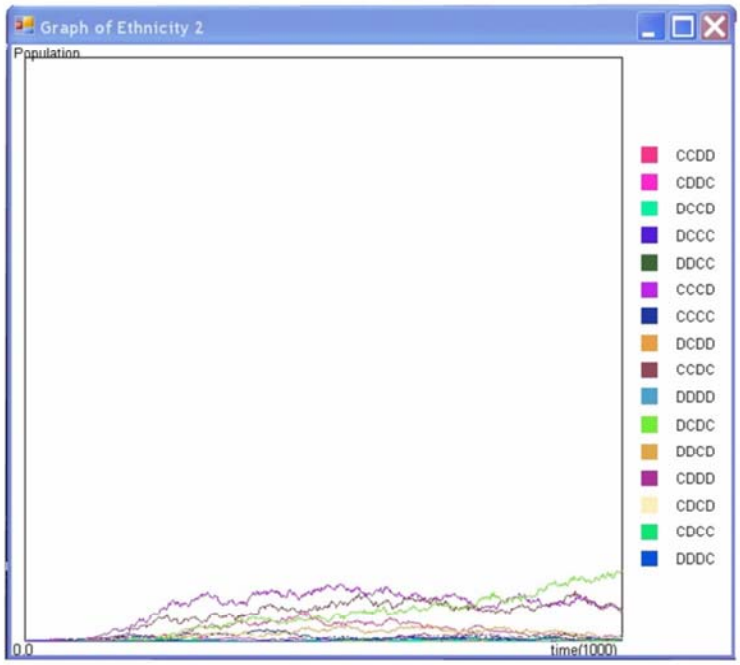


Ethnicity Recognition Modification Results

Data set 1

Page A





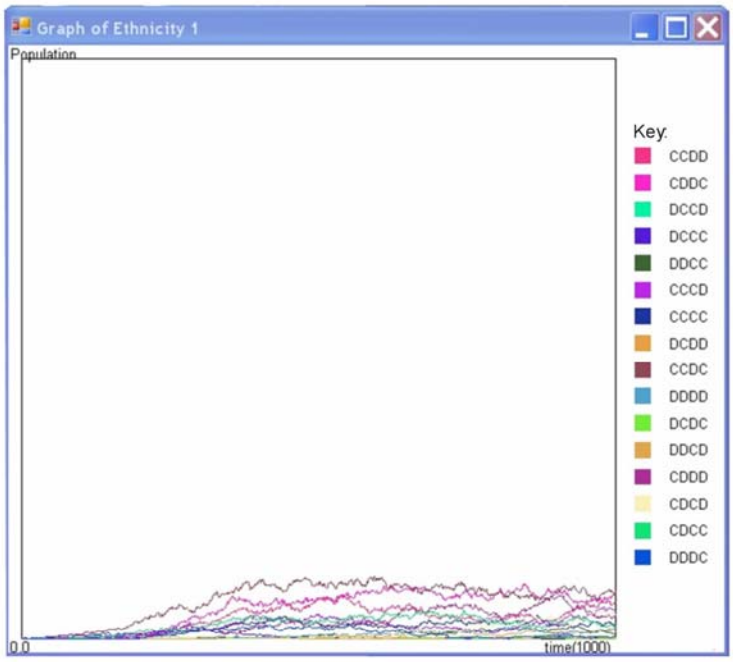
Key explanation

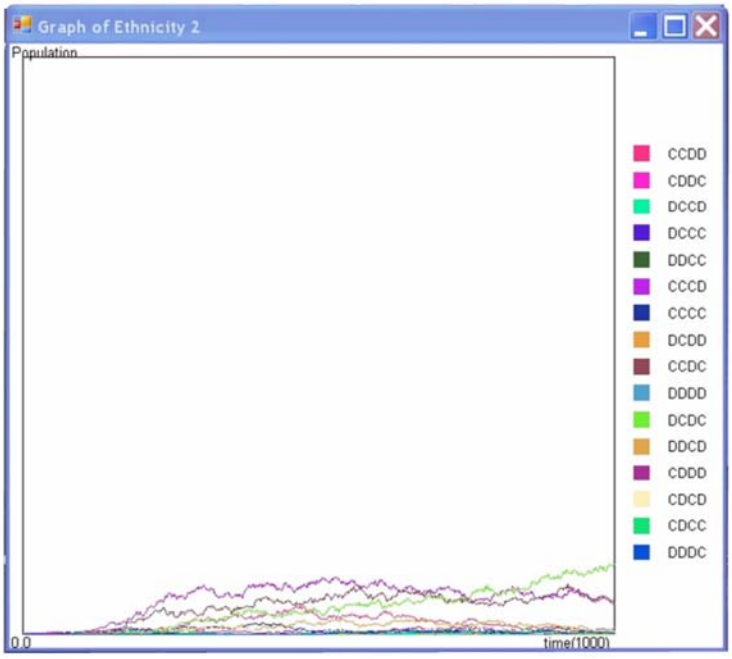
On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 1

Page B





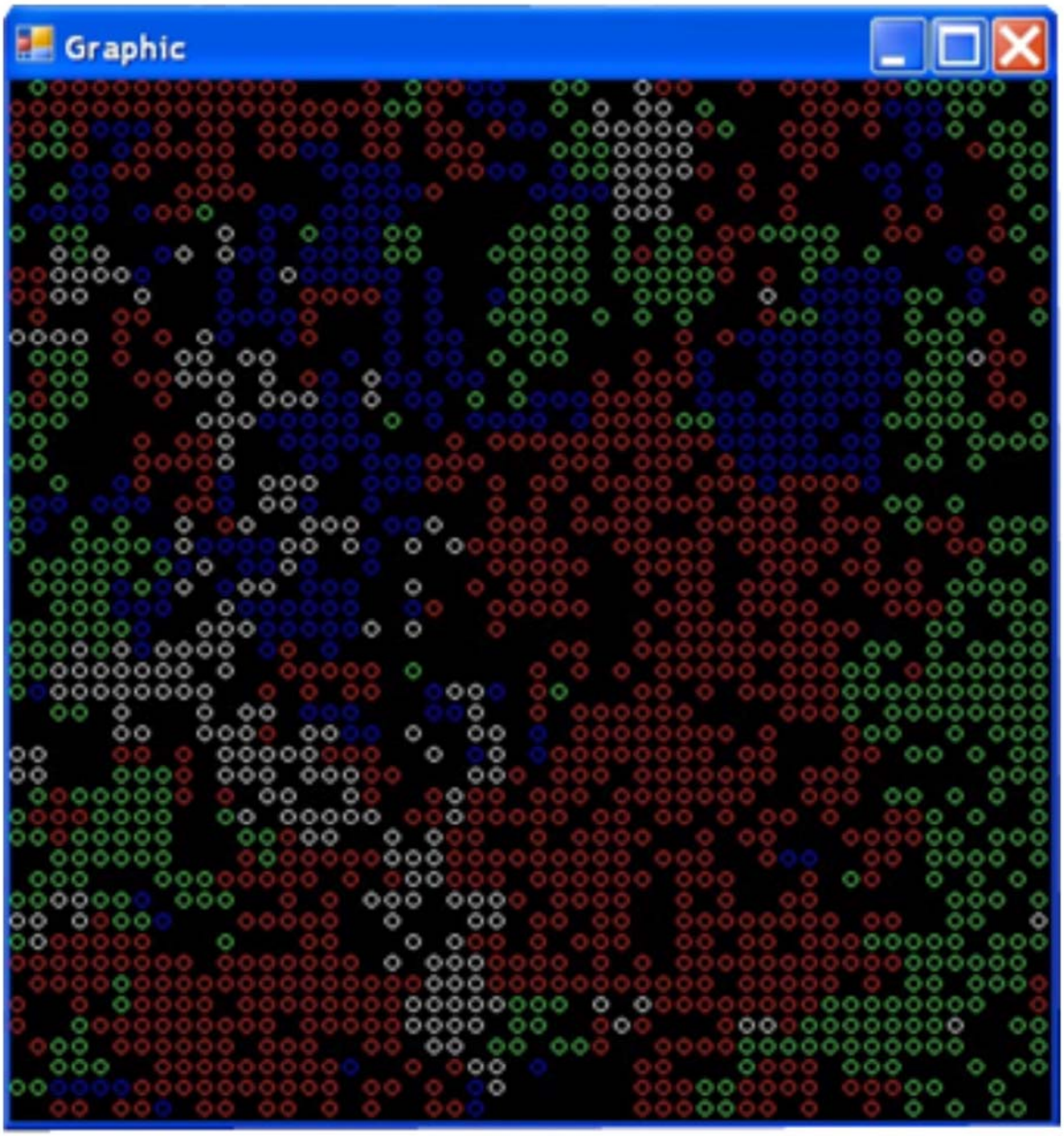
Key explanation

On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 1

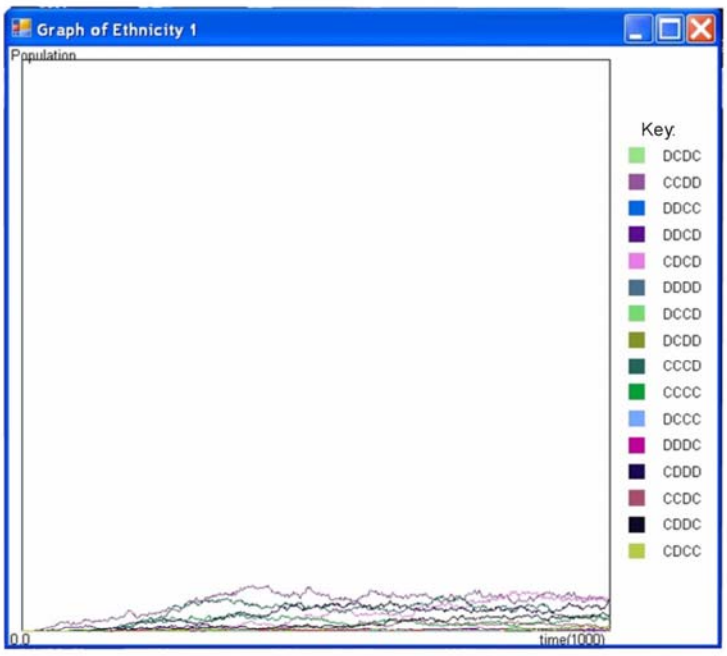
Page C

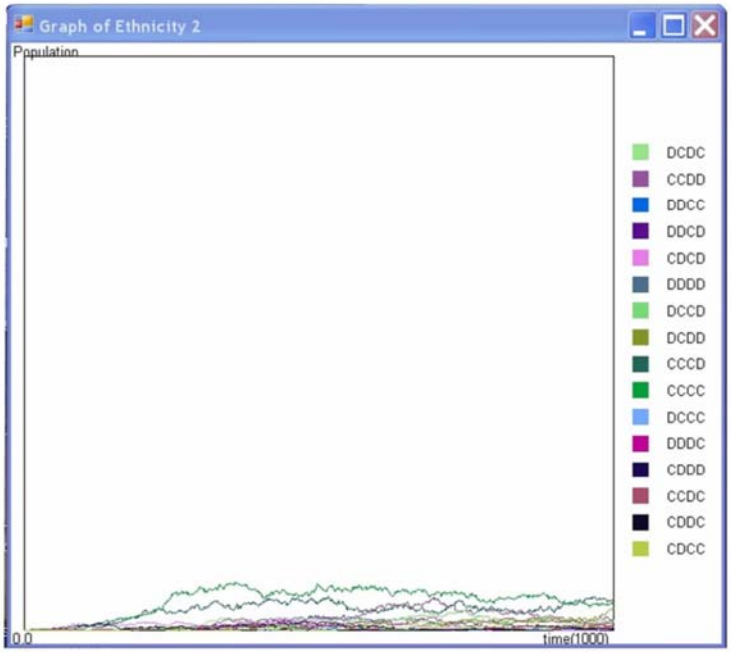


Ethnicity Recognition Modification Results

Data set 2

Page A





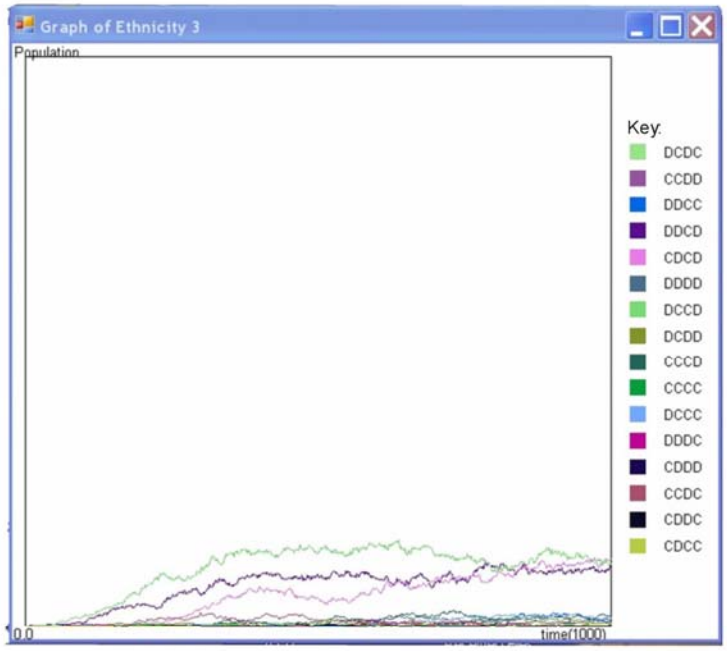
Key explanation

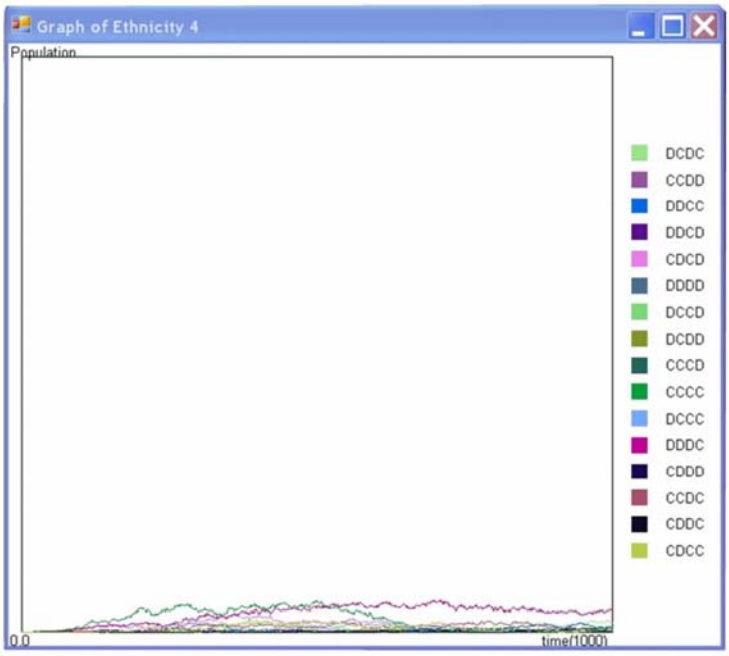
On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 2

Page B





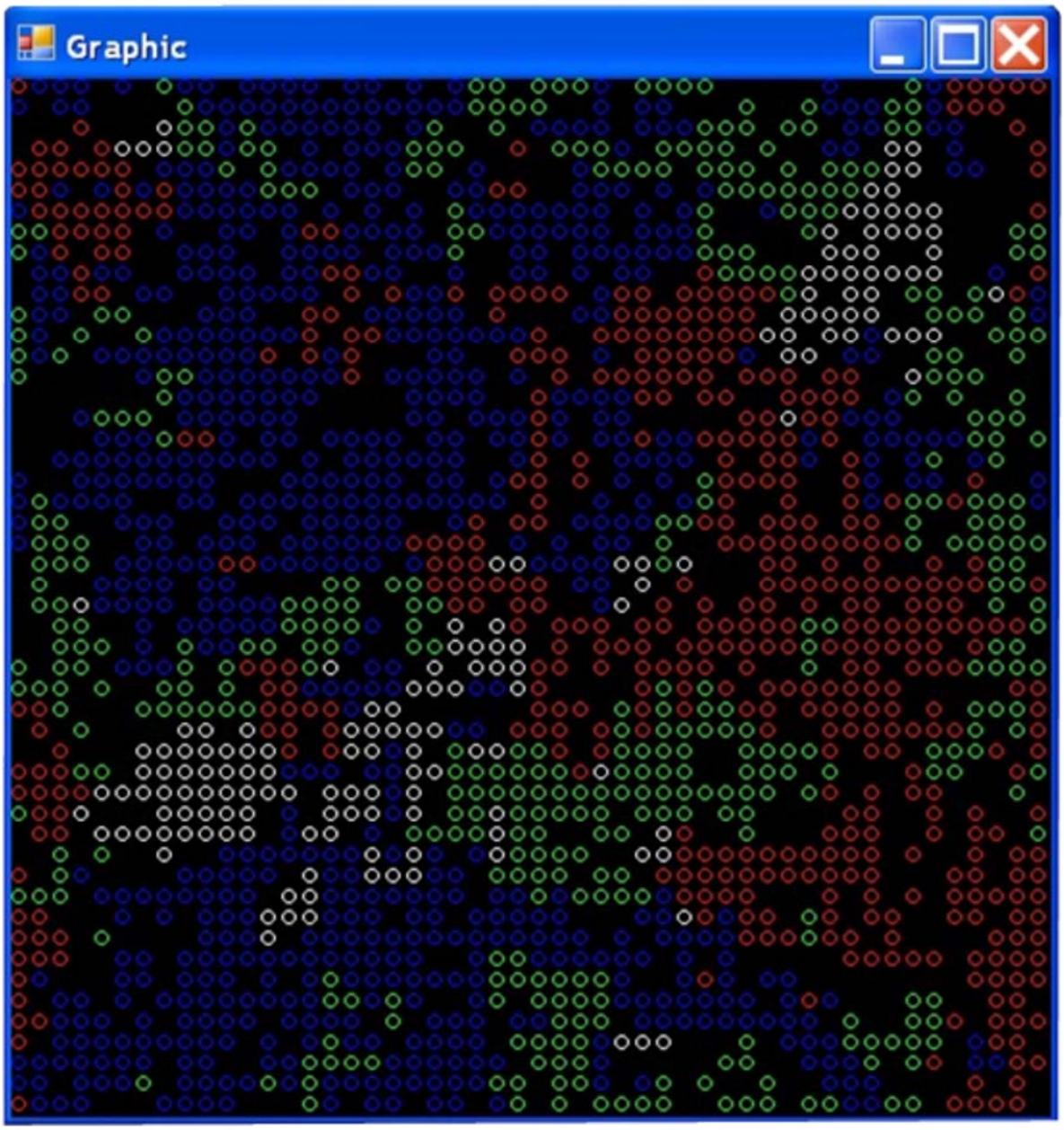
Key explanation

On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 2

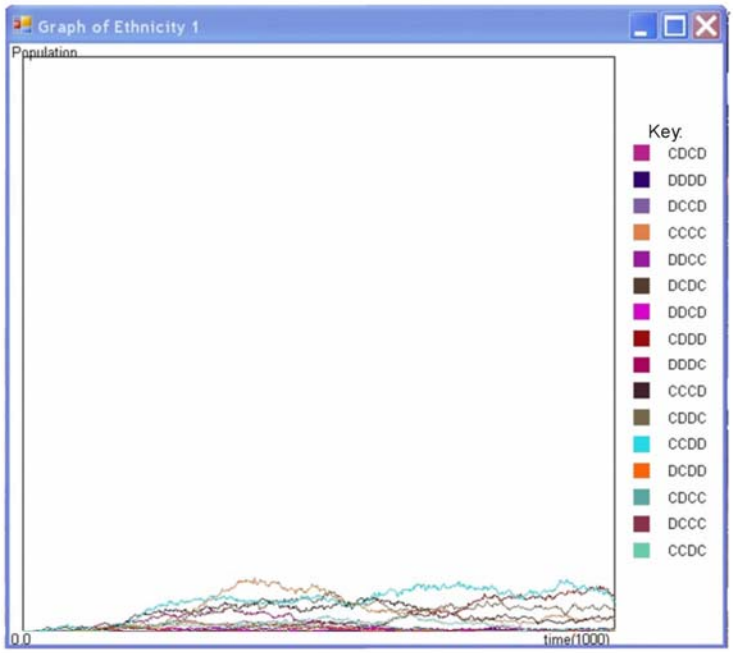
Page C

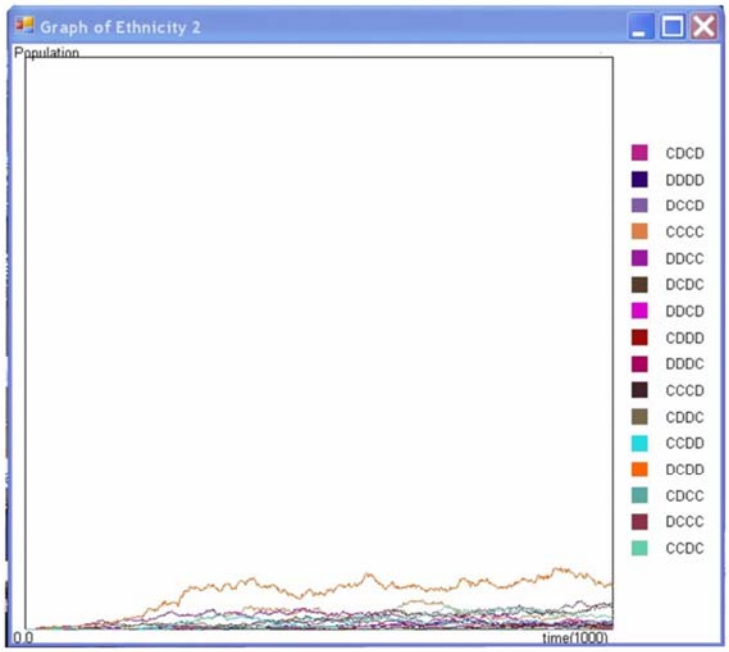


Ethnicity Recognition Modification Results

Data set 3

Page A





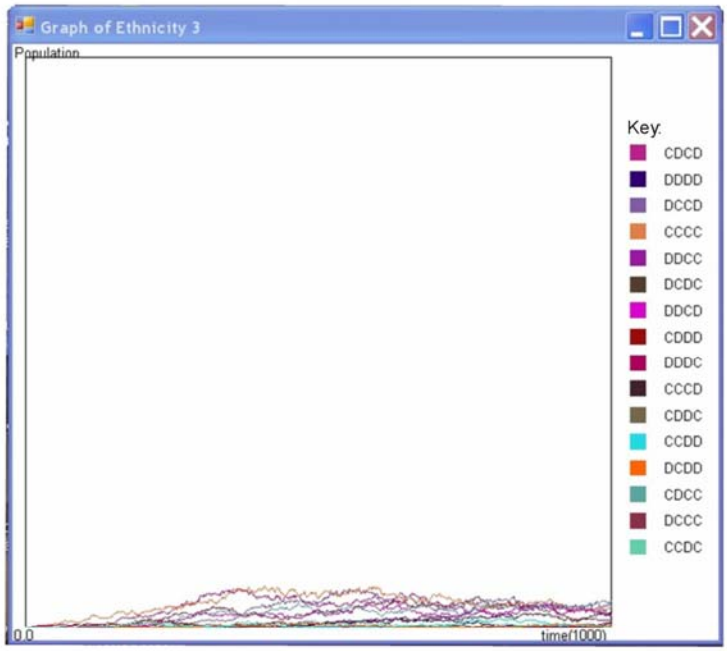
Key explanation

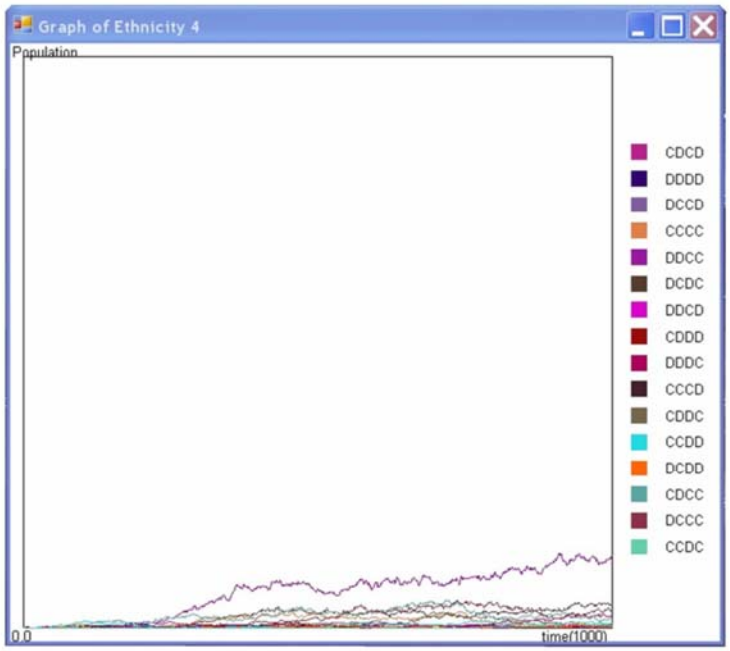
On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 3

Page B





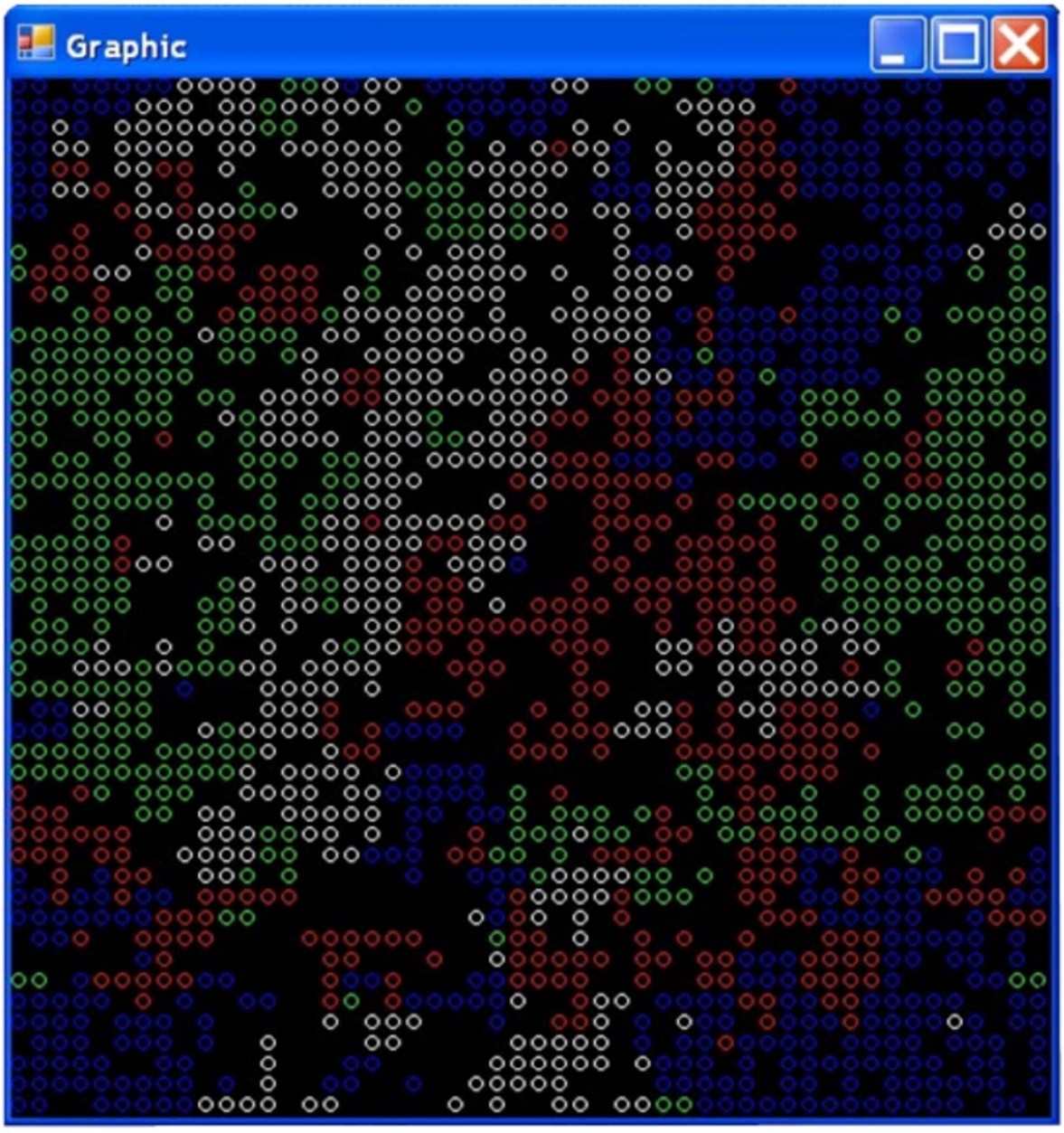
Key explanation

On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 3

Page C

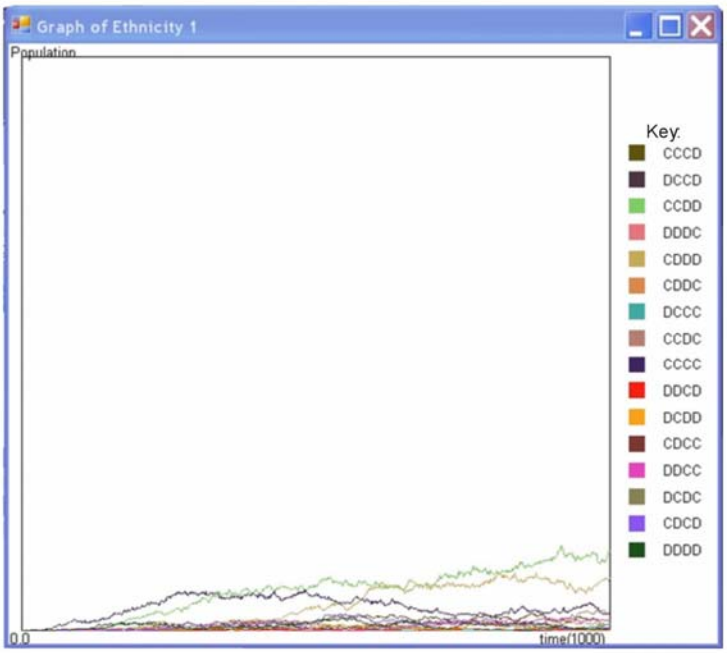


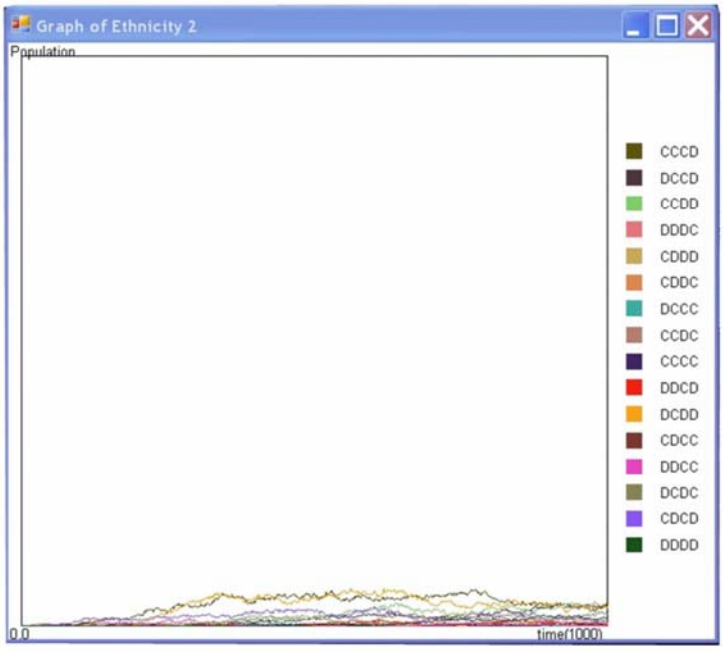
Ethniciy 1 Ethniciy 2 Ethniciy 3 Ethniciy 4

Ethnicity Recognition Modification Results

Data set 4

Page A





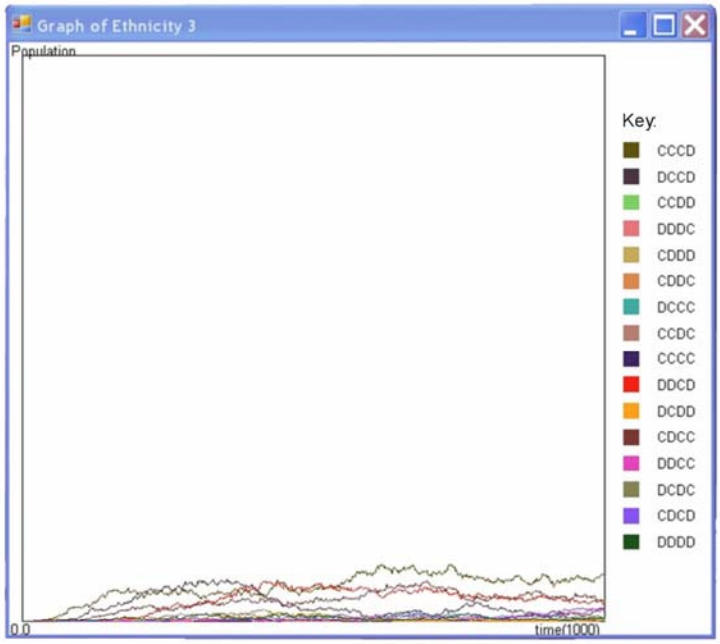
Key explanation

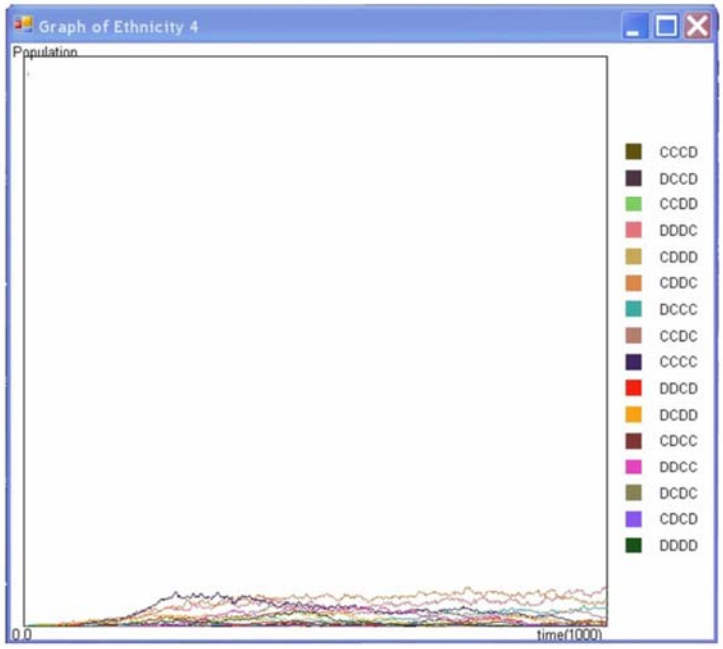
On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 4

Page B





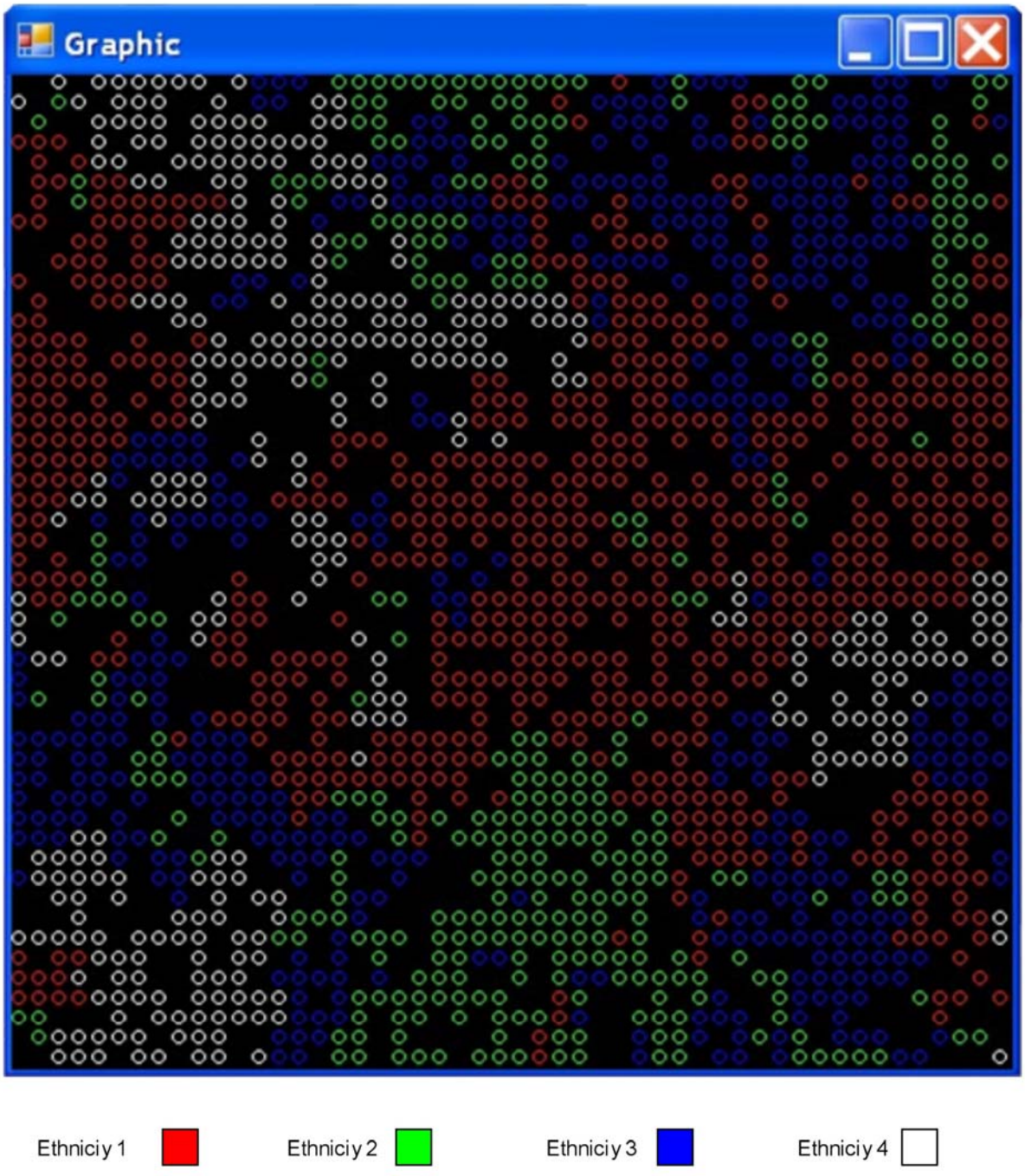
Key explanation

On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 4

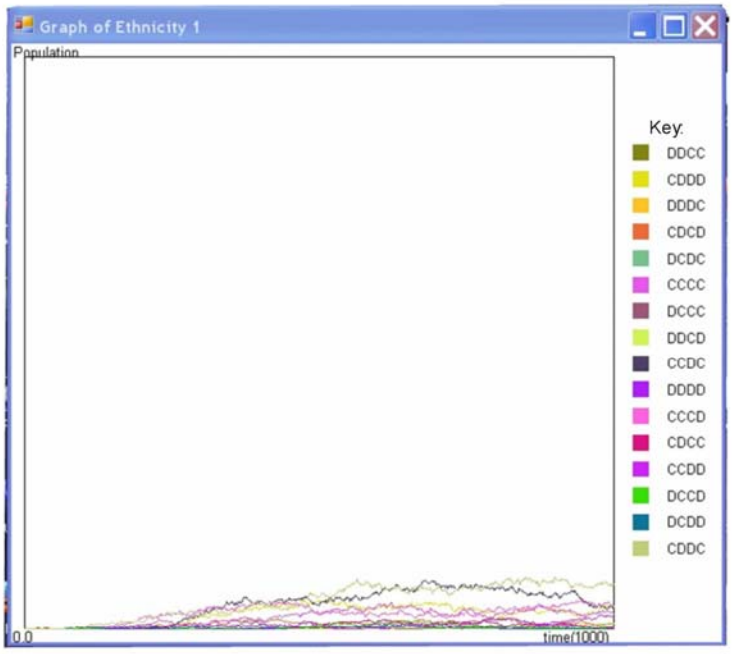
Page C

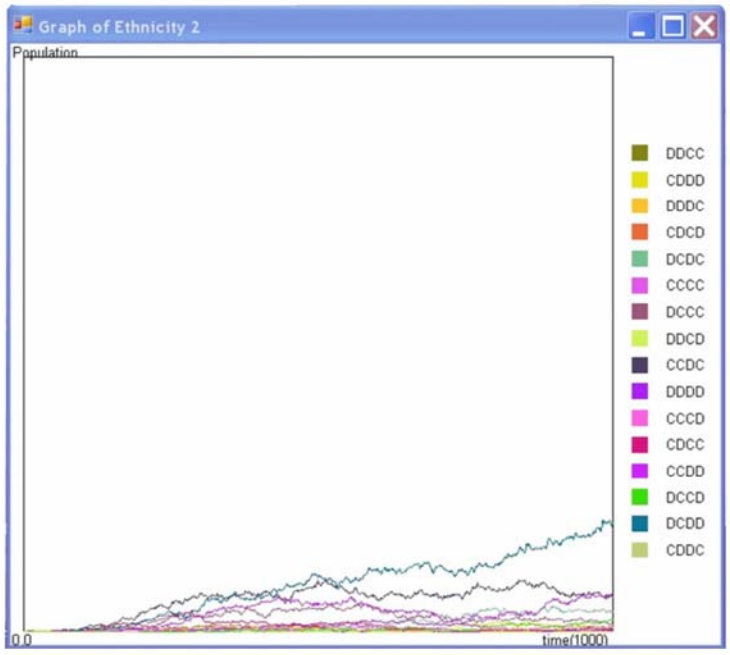


Ethnicity Recognition Modification Results

Data set 5

Page A





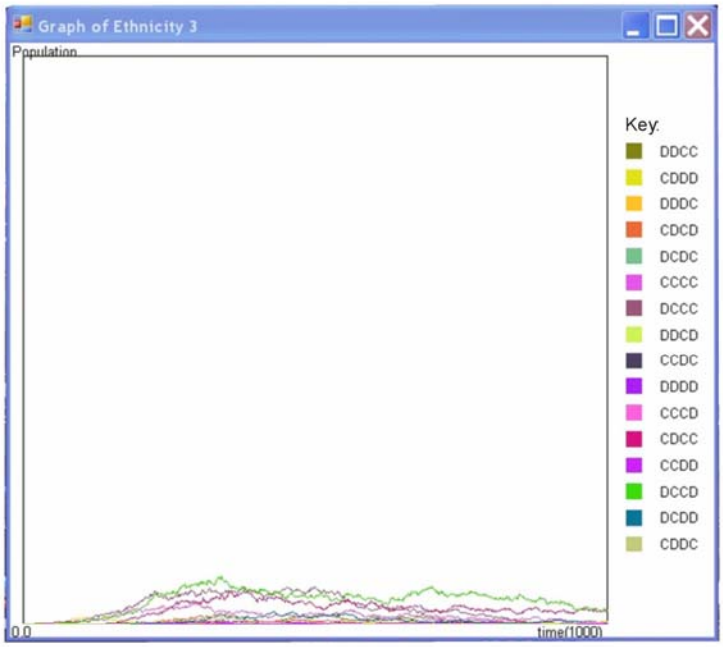
Key explanation

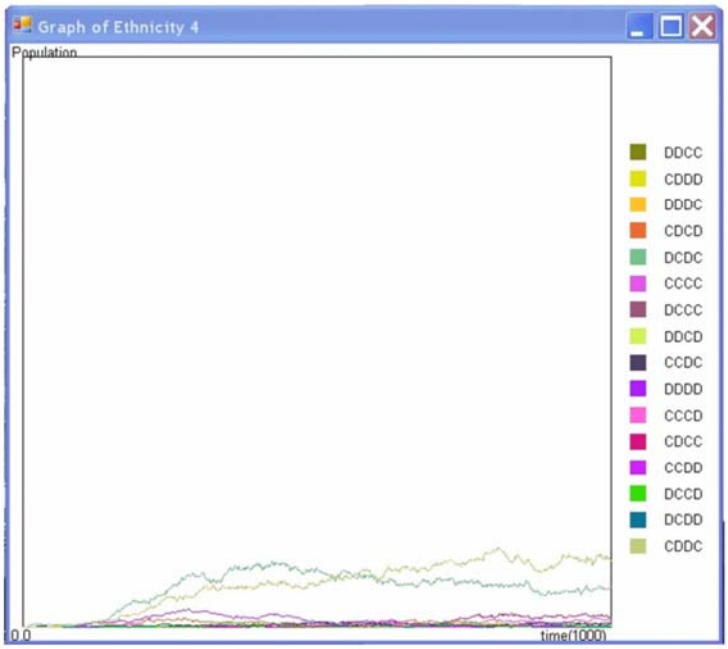
On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 5

Page B





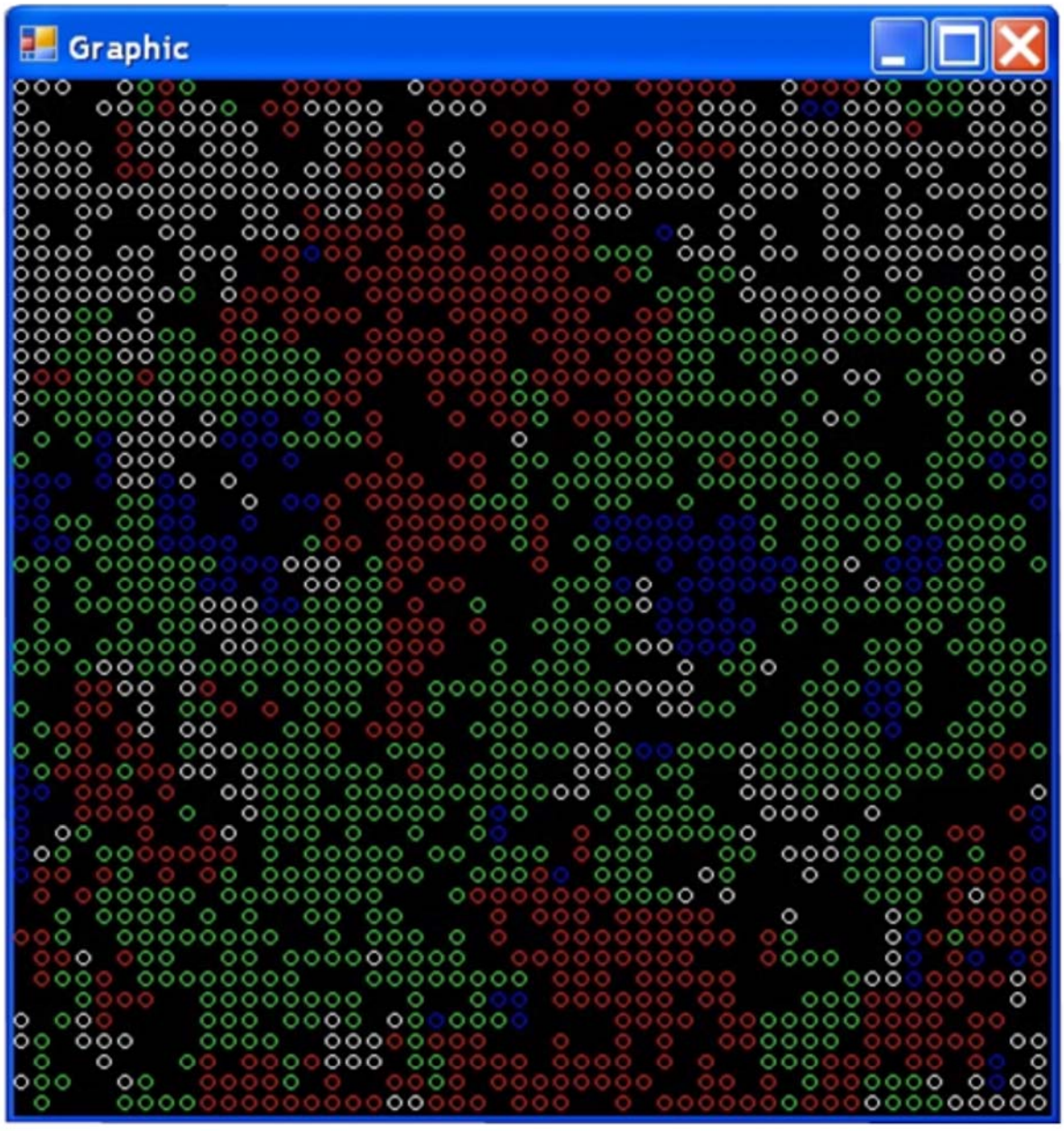
Key explanation

On the key the letter "C" represents cooperate and the letter "D" represents defect. The first letter of each sequence shows whether that specific ethnicity will always cooperate or always defect with ethnicity one the next three letters in the sequence show the method by which the strategy plays the modified Prisoner's Dilemma against the other three ethnicities.

Ethnicity Recognition Modification Results

Data set 5

Page C



Excel Output Charts:

Seed 0:

	Ethnicity	Ethnicity	Ethnicity	Ethnicity	Strategy Total
CCCC	2.51%	0.04%	0.20%	1.55%	4.30%
CCCD	4.14%	6.25%	2.29%	0.24%	12.91%
CCDC	7.95%	6.61%	0.78%	5.91%	21.25%
CCDD	5.29%	0.84%	0.57%	0.37%	7.07%
CDCC	2.89%	0.17%	0.27%	0.02%	3.35%
CDCD	3.65%	0.37%	3.98%	0.06%	8.07%
CDDC	6.90%	0.14%	0.18%	0.03%	7.25%
CDDD	4.98%	0.37%	0.03%	0.26%	5.63%
DCCC	0.31%	0.06%	0.23%	1.12%	1.72%
DCCD	0.06%	0.20%	0.63%	0.62%	1.51%
DCDC	0.10%	10.45%	0.35%	0.58%	11.48%
DCDD	1.26%	0.94%	0.01%	0.14%	2.34%
DDCC	1.55%	0.44%	0.04%	0.09%	2.12%
DDCD	0.21%	0.01%	7.27%	0.08%	7.57%
DDDC	0.33%	0.27%	0%	1.40%	2%
DDDD	1.22%	0.12%	0.01%	0.07%	1.42%
Ethnicity%	43.32%	27.29%	16.84%	12.55%	

Seed 1:

	Ethnicity	Ethnicity	Ethnicity	Ethnicity	Strategy Total
CCCC	2.24%	4.73%	0.20%	0.63%	7.80%
CCCD	2.79%	4.96%	1.74%	0.04%	9.53%
CCDC	1.03%	1.41%	0.07%	0.50%	3.01%
CCDD	6.14%	1.83%	0.71%	0.20%	8.88%
CDCC	0.45%	0.57%	0.26%	0.30%	1.58%
CDCD	5.87%	0.06%	11.26%	0.11%	17.30%
CDDC	2.73%	0.47%	0.02%	0.26%	3.48%
CDDD	4.39%	0.28%	0.02%	0.46%	5.16%
DCCC	0.31%	0.29%	1.83%	0.42%	2.84%
DCCD	0.01%	1.68%	12.30%	0.07%	14.04%
DCDC	0.08%	1.17%	0.01%	1.54%	2.80%
DCDD	0.89%	2.22%	0.90%	0.49%	4.51%
DDCC	0.09%	0.01%	1.65%	1.06%	2.81%
DDCD	0.32%	0.01%	10.02%	0.03%	10.37%
DDDC	0.13%	0.80%	0.05%	3.55%	4.52%
DDDD	0.01%	0.38%	0.92%	0.04%	1.35%
Ethnicity%	27.48%	20.87%	41.96%	9.69%	

Seed 2:

	Ethnicity	Ethnicity	Ethnicity	Ethnicity	Strategy Total
CCCC	2.22%	1.41%	3.26%	0.25%	7.14%
CCCD	2%	0.85%	1.76%	0%	4.60%
CCDC	0.39%	2.32%	0.32%	0.74%	3.78%
CCDD	7.27%	0.95%	0.26%	0.75%	9.22%
CDCC	0.15%	0.19%	3.84%	1.88%	6.05%
CDCD	0.05%	1.24%	3.03%	0.04%	4.36%
CDDC	3.93%	0.01%	0.39%	0.53%	4.86%
CDDD	6.74%	0.61%	0.01%	0.07%	7.43%
DCCC	0.21%	0.50%	1.65%	2.39%	4.76%
DCCD	0.04%	3.79%	3.16%	0.26%	7.26%
DCDC	0.03%	3.33%	0.40%	3.64%	7.40%
DCDD	0.08%	8.78%	0.41%	0.35%	9.61%
DDCC	0.14%	0.01%	3.62%	11.50%	15.28%
DDCD	0.03%	1.25%	2.67%	0.33%	4.28%
DDDC	0.02%	0.24%	0.32%	1.96%	2.55%
DDDD	0.41%	0.85%	0.10%	0.05%	1.41%
Ethnicity%	23.69%	26.34%	25.21%	24.76%	

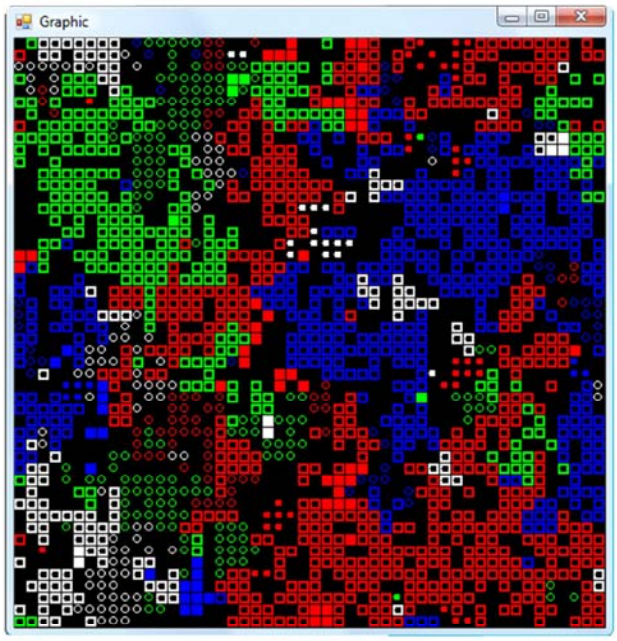
Seed 3:

	Ethnicity	Ethnicity	Ethnicity	Ethnicity	Strategy Total
CCCC	3.81%	0.30%	0.61%	1.01%	5.72%
CCCD	1.81%	3.26%	7.54%	0.03%	12.64%
CCDC	2.79%	0.41%	0.56%	2.29%	6.05%
CCDD	13.03%	3.48%	0.35%	0%	16.86%
CDCC	0.50%	0.10%	4.30%	0.17%	5.08%
CDCD	0.48%	1.13%	1.94%	0.15%	3.70%
CDDC	1.31%	0.93%	0.06%	4.67%	6.97%
CDDD	7.96%	0.06%	0.01%	0.12%	8.15%
DCCC	0.62%	0.01%	0.83%	2.88%	4.34%
DCCD	0.03%	3.29%	1.60%	0.32%	5.24%
DCDC	0.17%	1.67%	0.76%	0.71%	3.31%
DCDD	0.33%	3.27%	0.17%	0.76%	4.53%
DDCC	1.98%	0.05%	1.70%	0.08%	3.81%
DDCD	0.28%	0.35%	3.79%	0.59%	5.02%
DDDC	0.02%	0.49%	0.01%	5.62%	6.14%
DDDD	0.50%	1.48%	0.41%	0.04%	2.42%
Ethnicity%	35.63%	20.28%	24.65%	19.44%	

Seed 4:

	Ethnicity	Ethnicity	Ethnicity	Ethnicity	Strategy Total
CCCC	2.93%	0.16%	0.59%	1.13%	4.81%
CCCD	4.46%	0.80%	0.13%	0.01%	5.40%
CCDC	5.08%	6.38%	0.02%	0.43%	11.91%
CCDD	1.84%	5.65%	0.03%	0.06%	7.59%
CDCC	0.49%	0.42%	2.33%	1.96%	5.20%
CDCD	0.90%	0.08%	0.10%	0.05%	1.14%
CDDC	7.90%	0.57%	0.18%	11.73%	20.38%
CDDD	3.14%	0.89%	0.09%	0.64%	4.76%
DCCC	0.10%	1.88%	0.06%	0.02%	2.06%
DCCD	0.12%	1.38%	3.10%	0.27%	4.87%
DCDC	0.68%	3.55%	0.06%	6.32%	10.61%
DCDD	0.02%	16.83%	0.46%	0.05%	17.37%
DDCC	0.10%	0.50%	0.08%	0.40%	1.09%
DDCD	0.02%	0.01%	0.03%	0.20%	0.26%
DDDC	0.85%	0.17%	0.17%	0.08%	1.27%
DDDD	0.38%	0.29%	0.01%	0.63%	1.30%
Ethnicity%	29.02%	39.57%	7.43%	23.97%	

Normal run with seed of 1



Key:

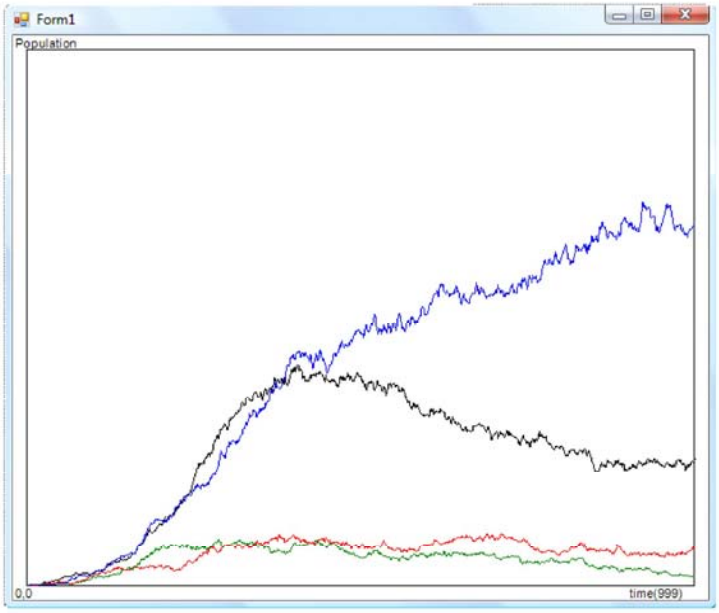
Ethnocentrics

Altruists

Egoists

Cosmopolitans

(Color represents different ethnicities)



Key:

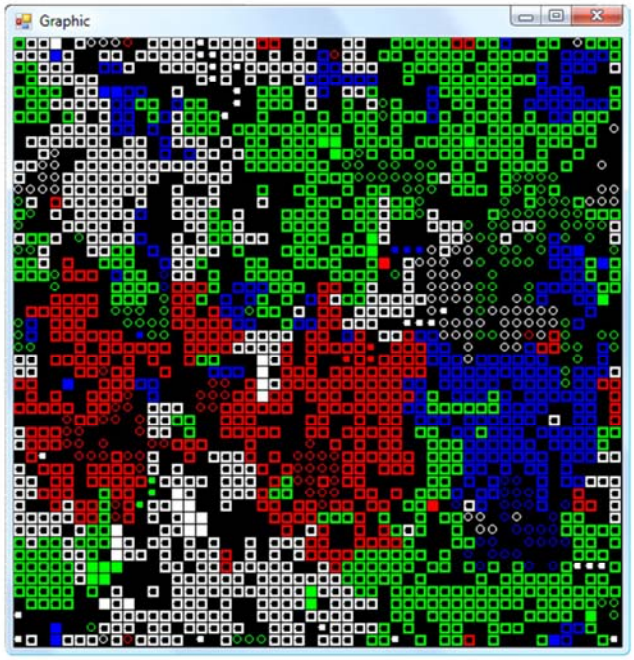
Ethnocentrics

Altruists

Egoists

Cosmopolitans

Normal run with seed of 2



Key:

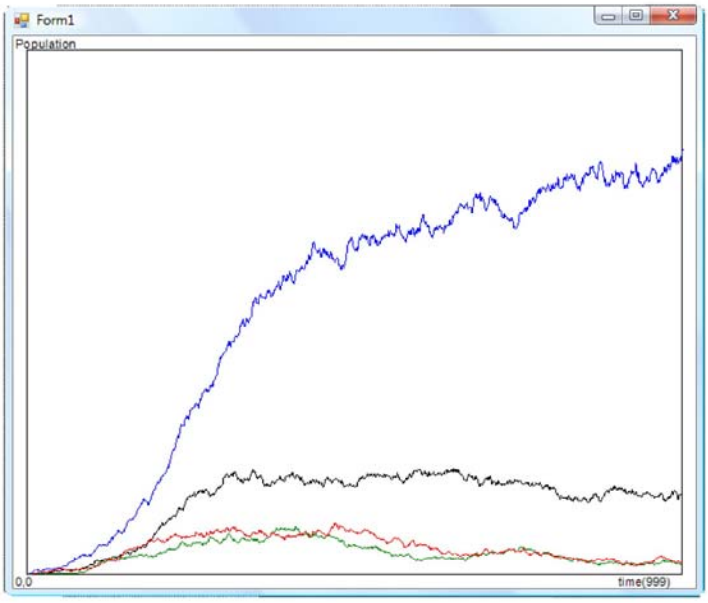
Ethnocentrics

Altruists

Egoists

Cosmopolitans

(Color represents different ethnicities)



Key:

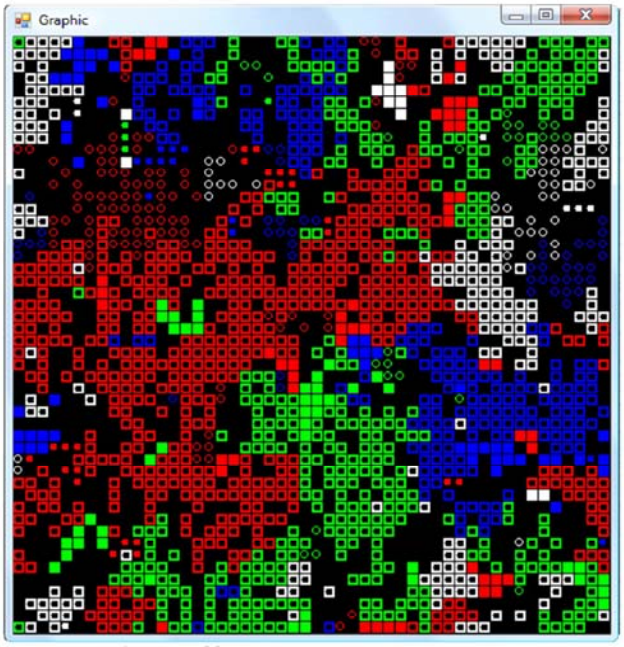
Ethnocentrics

Altruists

Egoists

Cosmopolitans

Normal run with seed of 3



Key:

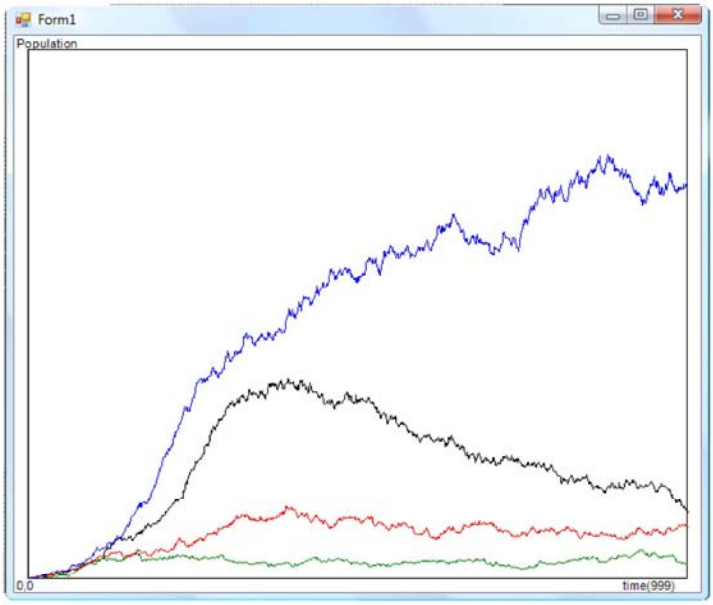
Ethnocentrics

Altruists

Egoists

Cosmopolitans

(Color represents different ethnicities)



Key:

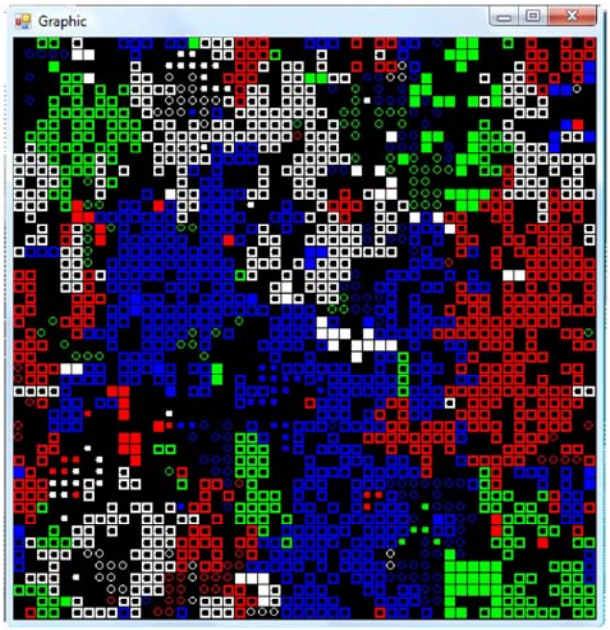
Ethnocentrics

Altruists

Egoists

Cosmopolitans

Normal run with seed of 4



Key:

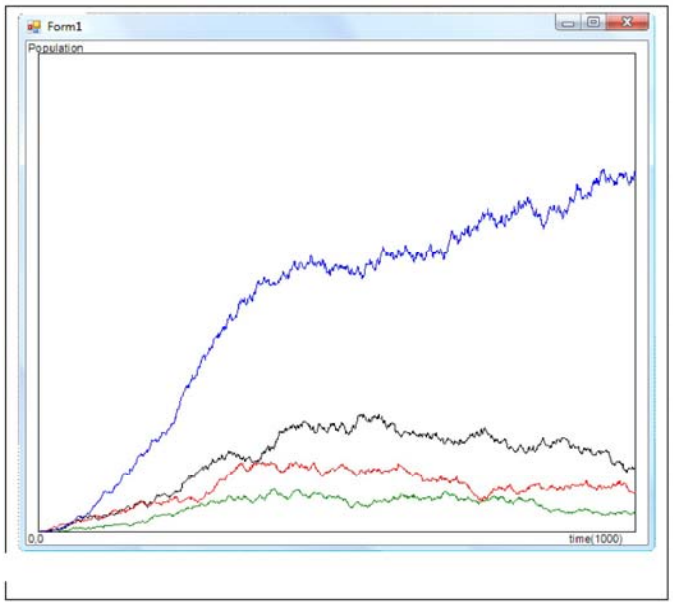
Ethnocentrics

Altruists

Egoists

Cosmopolitans

(Color represents different ethnicities)



Key:

Ethnocentrics

Altruists

Egoists

Cosmopolitans