Project 3

Added by Christopher Hoder, last edited by Christopher Hoder on Oct 02, 2011

The goal of this project was to continue and expand on last weeks Game of LIfe. In this simulation we are learning about how different "Social Agents" will interact with each other on a grid. In this simulation, the grid will be populated with a given number of different social agent types and they will remain in place if they are surrounded by more neighbors of the same type than neighbors of a different type. If they are surrounded by more neighbors of a different type than the social agent will try to move. The movement will be anywhere in the 9-grid neighborhood of the Social Agent's location.

My solution was implemented by updating the previous weeks code to be able to deal with null spaces in the grid and likewise deal with social agents of different types. While most methods required only minor changes, the major changes were in the advance method. The way that advancing a step is done in this simulation is not done with each cell acting in parallel with eachother (like in the Game of Life). Instead, the elements advance sequentially with a random elements advancing until all the populated elements in the grid have advanced. In doing this, I added all the SimObjects in the grid to a LinkedList. I chose to use a linked list over an array list or an array because of the need to remove elements within this list (that are at a random index) and the linked list allowed for the easiest, fastest removal of elements. Arrays and ArrayLists both store data in arrays and to remove an element from the middle of the array would require all of the elements to be shifted over and this can be time consuming with large grids. Once a Linked List has been created, a random integer is picked between 0 and the number of SimObjects remaining in the Linked List. This number corresponds to the index in the LinkedList and this SimObject is removed, and its state updated.

Another method that required significant change was the initializeRandom method in the SocialSimulation class. Now it must both determine whether a cell in the grid is occupied and what type of SocialAgent will be in the cell if it is going to be occupied. The way that I did this was to loop through each cell in the grid and then if a random number is less than the density it will be occupied (same as before). Then another random number is picked between 0 (inclusive) and number of types (exclusive) and that will by the type of Social Agent occupying the cell in the grid.

As part of the extensions I made it possible for the user to change a few parameters of the game. First, the user can change the probability that a "happy" socialAgent will try to move to a new space around it. This is done by storing a public static parameter in the abstract SimObject class so that all the SimAgents will have this new probability if it is changed. Second, before the simulation starts, the user can choose to create alliances. This is done by the user inputting the two types that should be aligned. You can set any number of alliances. However, an alliance with a type that is not the same as it will only count toward 0.5 of a like neighbor. The way this is done is by storing an array of ArrayLists in the landscape class. The index of the array corresponds to the SocialAgent of that type. The ArrayList is basically an Array of integers that correspond to the types of SocialAgents that the given SocialAgent is an ally with (this includes itself). The reason I sued an ArrayList here and not just an array of Integers was to allow for varying number of alliances. With an array of integers, I would have to take into account the size of each array and this would depends on the number of alliances the user wanted to create.

As another extension, I modified the way the Life Simulation is run and you can now run both Simulations from the same program. The user can choose between the 2 simulations either in the interactive set-up (discussed later) or on the command line. The game of life is run by having a null (emtpy) cell in the grid correspond to a dead cell and then populating the grid with only one type. A different advance method was also needed, called advanceLifeSim. However, since dead cells correspond to empty cells in the grid the size of the list returned by getNeighbors is equal to the number of alive neighbors that the cell element has. A similar method to last weeks project, creating a second grid and stepping through the each element in the new grid was used to advance every cell in parallel. You can also still load up a pre-designed map from a file. This mode can be reached by starting the program with no inputs and following the instructions.

Running the SocialSimulation can be done in 3 different ways. If you run the program with no command inputs, you will be prompted in the terminal to give parameters for everything you want to do. It would look something like this

```
chris-hoders-macbook-pro-4:Lab03 chris$ java SocialSimulation
You started the simulation Program!
 Choose the type of game you would like to play
 1) Game of Life 2) Social Simulation
2
How many types?         4

 What do you want the probability( i.e. 0.05 == 5%) that a happy agent moves to be?  .05
 How many trials?        200

 Pause Time (ms)?        0

 How many rows?          80

 How many cols?          80

 How dense?     0.4

 Would you like to set alliances? (1) Yes (2) No        1
Input the allinances you want to create. There are 4 different types on the grid. To create an alliance,
use the following syntax
 [1] [2] to create an alliance between types 1 and 2, then hit enter. After you are done, hit enter on an
 empty line. Types are from 0 (inclusive) to types (exclusive)
1 2
2 3
```

In this mode you can load maps for the Game of Life from a text file. You can not do this by command line parameters.

You can also run this program from simple command line parameters. In this project there is no error checking so all parameters must be correct and present. The syntax is as follows:

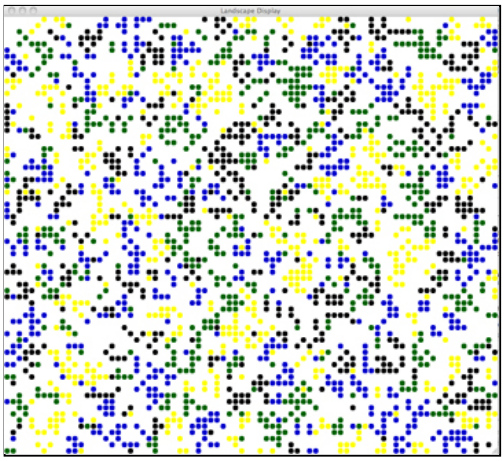
java SocialSimulation [Mode] ?[rows ?] [cols] [iterations] [pauseTime] [density] [Types] [changeProbability]

Mode should either be (1) for Social Simulation or (2) for the Game of Life. If you are running the Game of Life the last 2 are not needed and should not be input into the command line because these correspond only to the SocialSimulation game. If the user chooses to run a Social Simulation it will be prompted to create alliances before the simulation starts.

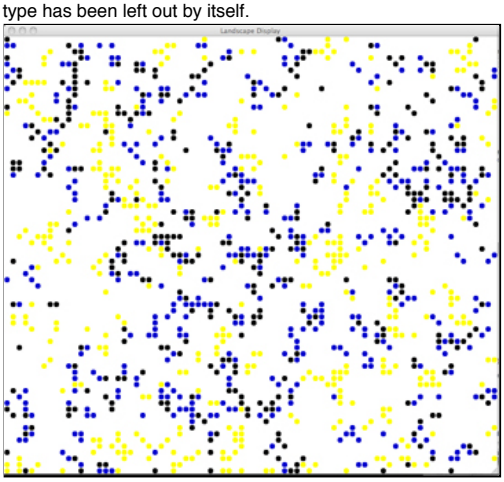
With this model we can observe some group dynamics. One early observation is that if one sets the changeProbability to 0 so that all happy socialAgents remain in place, the game will quickly come to a stationary image where all socialAgents are in a happy spot. However, by simply adding a 5% chance that a happy cell will move makes the Simulation never reach a stationary arrangement. Also, if the simulation is run for a large amount of trials this allows the different types of

Social Agents to slowly gather together and group in bigger groups.

Also, if the density is set really low, you can see that some elements will travel across large empty spots of the grid to reach clusters of similar elements. This allows for easier clustering of elements over time if the simulation is run for long enough. You also can notice that you tend to have little "villages" which have around 10 or so elements of a specific type spread around the grid and then you have traveler of each type running between them where they join up and then eventually another traveller breaks off and heads off on its own. (this is with a 0.05 change probability of a happy agent). I would not have guessed that this would have happend just from the rules. I would have assumed that the different types would cluster together (with the inner cells being completely surround and thus unable to move). The below image was run for 50000 trials and you can see a bit of clustering, even given the density which makes movement difficult.

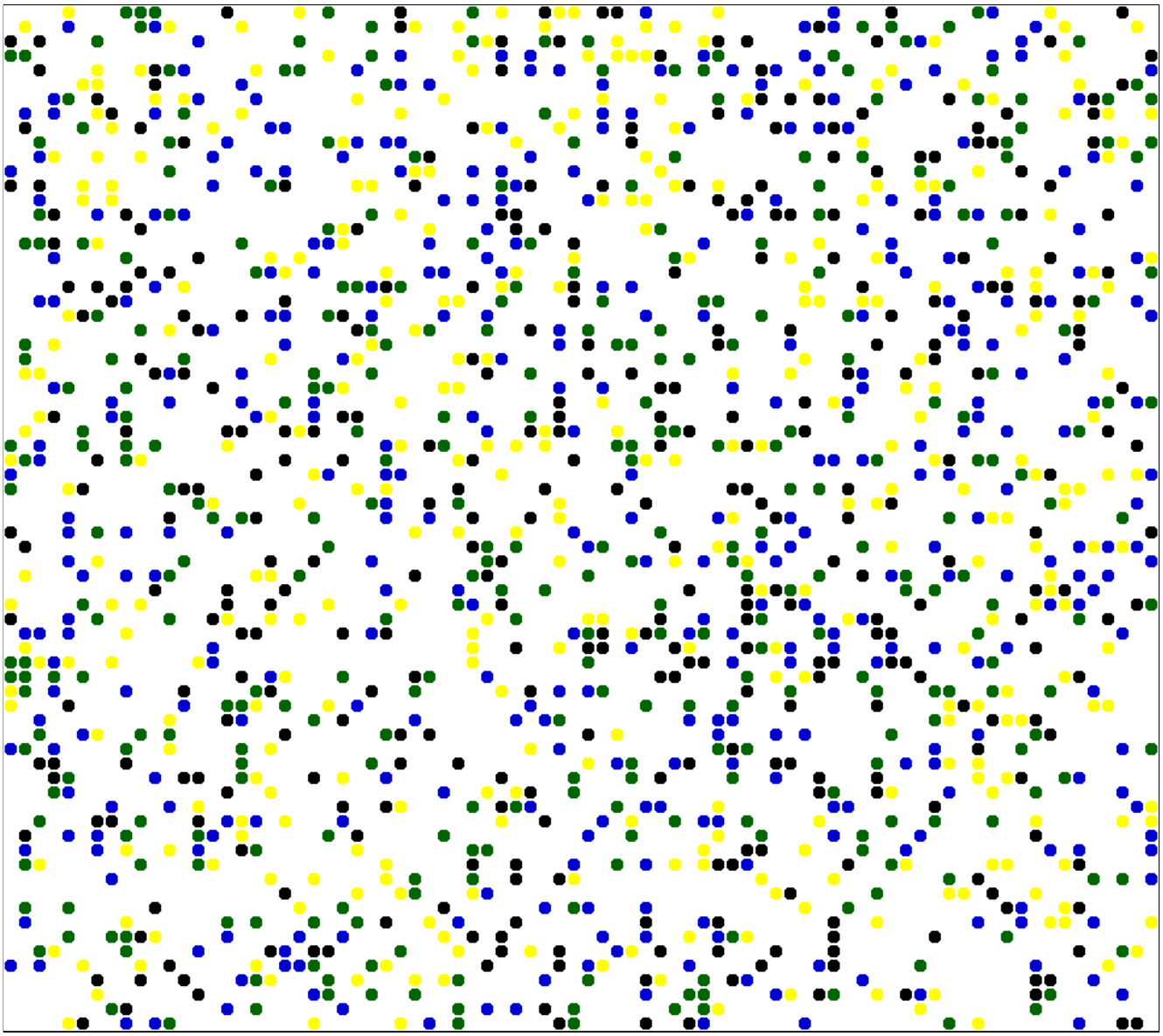


When playing around with adding in alliances to the mix of the SocialSimulation it is hard to tell what kind of difference is being made. However, you do now get various outliers of one type mingling with the agent it is allied with. Sometimes it will be completely by itself with no other similar colors in its neighborhood. An example is shown below. In this simulation the Black and blue were allied together and you can see that there is quite a bit of intermingling going on, while the third type has been left out by itself.



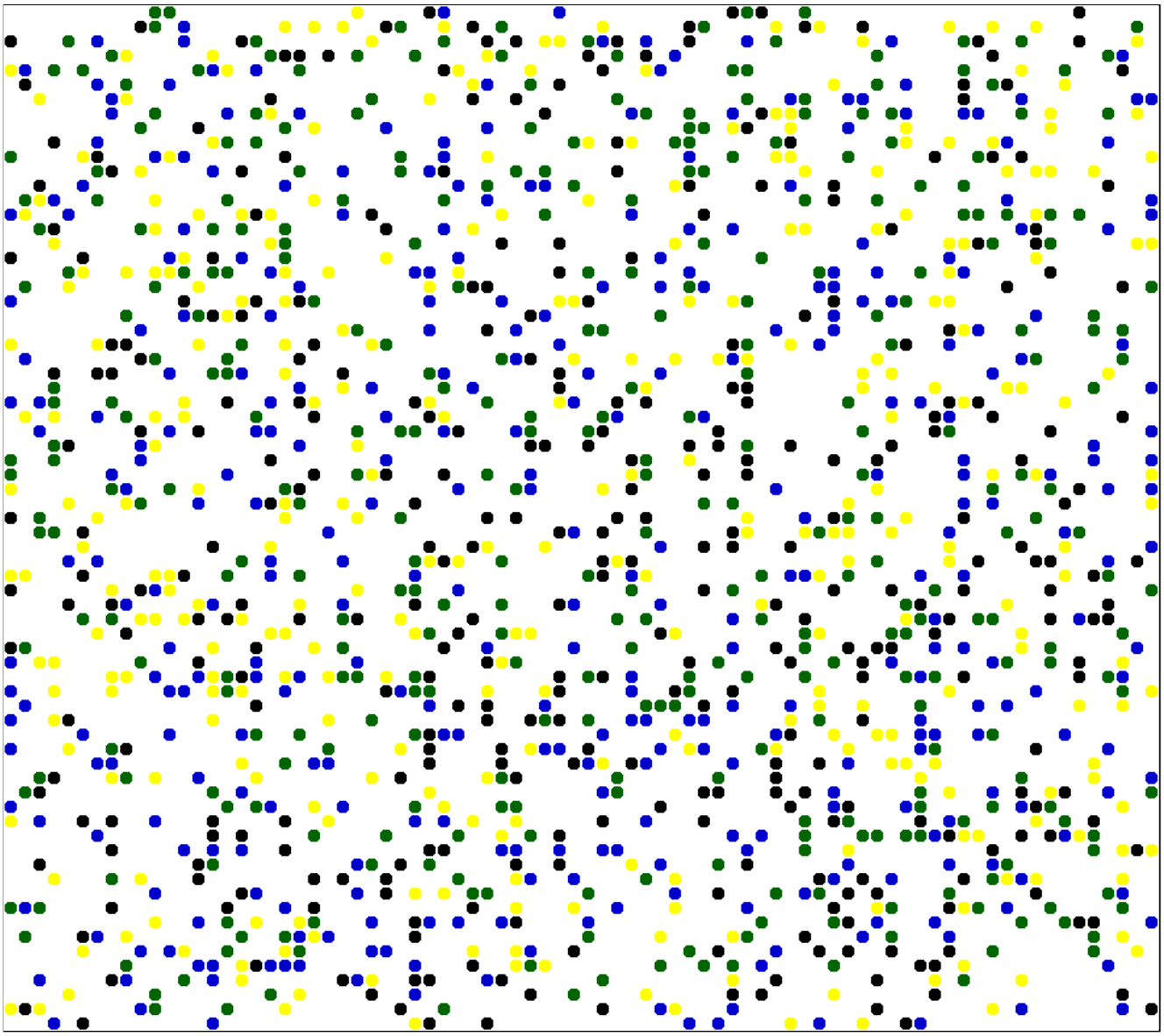
This gif below is a 500 turn simulation of the Social Simulation with a 0.05 change probability, 4 types and no alliances. The images were saved every 100 iterations. This was the command line executed

chris-hoders-macbook-pro-4:Lab03 chris$ java SocialSimulation 1 80 80 500 0 .25 4 0.05



This gif below was run with a change probability of a happy element set to 0. It was run also for 500 iterations saving every 100. As you can see with this setting the simulation reaches a steady state fairly rapidly where only a few elements remain moving. The more dense the starting board, the faster it reaches steady state. We can see that including this change probability is important in maintaining a fluid and changing simulation.There were no allinances for this run. The command line was this:

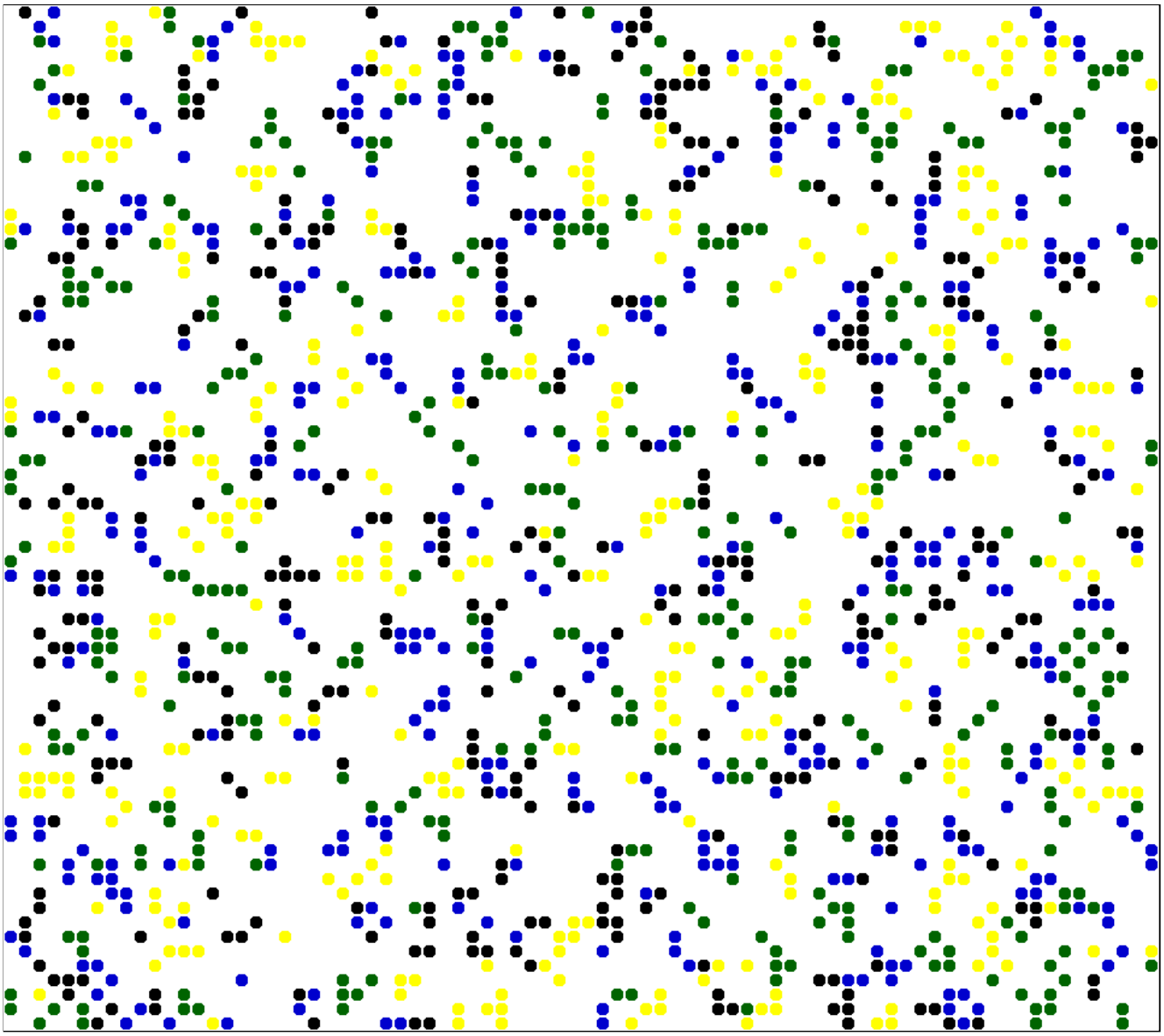
chris-hoders-macbook-pro-4:Lab03 chris$ java SocialSimulation 1 80 80 500 0 .25 4 0



This last gif was a SocialSimulation where there was an alliance set between the 0 and 1 types. The Change probability of a happy cell was set to 0.05. You can notice that in this one, unilke the other gifs we have mingling between the black and blue agents. Though also worth noting that this mingling is less when an element of another type is around, most likely due to the fact that an ally only counts toward 0.5 of a neighbor of the same type. The command line execution was this:

chris-hoders-macbook-pro-4:Lab03 chris$ java SocialSimulation 1 80 80 500 0 .25 4 0.05

Would you like to set alliances? (1) Yes (2) No 1
Input the allinances you want to create. There are 4 different types on the grid. To create an alliance, use the following syntax
[1] [2]
to create an alliance between types 1 and 2, then hit enter. After you are done, hit enter on an empty line. Types are from 0 (inclusive) to types (exclusive)
0 1



In conclusion, this project required me to think about the advantages of using certain data structures over others. For example, i chose to use a LinkedList when i needed to put all of the SimObjects in the grid into a list because of the ease of removing elements from the middle of a long list. On the other hand, I used an ArrayList for getNeighbors because it is simpler to get the size of it and to get a specific index. I also used an array of ArrayLists to store the different type of SocialAgents alliances. This data structure allowed me to more easily implement a dynamic list because you do not need to worry about the initial length of the ArrayList like you do with an array. Although I could have coded this using just arrays (which may have been slightly faster) it would have required a significant amount of more code to make sure that the proper lengths are created, especially since there is no limit to the number of alliances that can be formed. I also learned about abstract classes for this project. The ability to have a class that is inherited by all the social agents combined with the use of the static data declaration allowed me to be able to change the probability that a happy agent will want to move very easy.

Labels

cs231f11project3