The Genesis Project

DJ Richardson, Sarah Babski, Alice Nin

Introduction

The Genesis Project was supposed to be an extension of a Java-based population simulation engine that I started building at the beginning of this semester. Specifically the extension would have added goal-oriented planning to the citizens of Genesis, having them make intelligent decisions on who they marry and mate with as well as add new features such as careers and skills. Those intelligent systems could have used the Millington and Funge discontent tree or used partial order planners, both of which would've used a quantity of action points to limit actions. The systems changed over time several times. In a perfect world we would have also added a non-text based UI and there would be some Maxis-inspired flavor text. A passion project of mine would have turned into a graded labor of love by me and two people who voiced interest on Piazza.

Instead, this is a bit of an extension. It is much more a system of dice-rolling citizens that do what they do based on romantic, platonic, career, and narrative roles assigned to them. There is some flavor text in the form of life events but those were there before this project started. There is more or less a proper attraction system instead of fairly blind, heteronormative matchmaking. (The attraction system is still heteronormative unfortunately). There is now a career system as well as defined goals based off to roles. The success of accomplishing these goals are outputted to a CSV as well as a bunch of other files that document what happened in the land of Genesis.

Instructions

Clone the GitHub repo from the blackboard link. You can run the code two ways:

The IDE Way

Import this project into your favorite Java IDE. Alice and Sarah used IntelliJ. I used Eclipse Mars. If it is complaining about dependencies, make sure you have a dependency on the *gson-2.8.2.jar* that is also in the repo.

The Command Line Way

Included in the repo is JAR called genesis.jar. You can run this jar as you normally would (java –jar genesis.jar (<program arguments>)). If you don't want to clutter your console, I would recommend directing the output of the run to a text file. All the output just prints out the current year in the simulation, the number of people who have ever lived in the simulation, the current living population number, and the youngest generation alive.

For either way above, you will need to feed in three program arguments into the program. In the IDE you will set this up in the run configuration where on command line they will follow right after genesis.jar.

END CONDITION TYPE	END CONDITION (integer)	FILE PREFIX
Population	The maximum historical population (the simulation will	The beginning string of all output files.

	end after this many people have lived in the simulation or no one is alive)	
Generation	The maximum generation. (The simulation will after once the first person of this generation is born or no one is alive)	
Year	The last year to run the simulation	

Once the simulation is over, it will output a few files to the output file. (Examples of these are in output/sampleout in the repo)

- prefix-goalStats.csv has information on the different goals that people had in the simulation and how successful they were at accomplishing them.
- prefix-relationshipInfo.csv has information on the final state of relationships once the relationship is over. It does not have a history of relationships.
- Prefix-founderInfo.csv has information on the original 500 founders of the simulation. This is legacy from the original engine that I used to just see what the genetics pool was like. You're free to look at it.
- Prefix-personalInfoStats.csv has information on *everyone* that has lived in the simulation. Be careful of opening this in Excel because there is a limit on the number of rows per spreadsheet.
- Prefix-lifeEvents.csv is legacy file output. It is not all life events in the simulation.
- Prefix-familyTree.fs is a FamilyScript file. You can import it into https://www.familyecho.com/ to see the lineages formed by the program. This is legacy from the engine but is honestly one of my favorite parts of the pre-project engine.
- Prefix-diverseSamples folder contains 20 randomly selected people from the simulation that got to at least age 50 so you can see what actions they did. These actions include:
    - Marriage
    - Having a child
    - Joining a Career
    - Taking Leave from a Job
    - Quitting a Job
    - Retiring from a Job
    - Changing relationship status (platonic or romantic) with another

Looking at these above files is what helped us glean (and what can help you glean) what went well (and not well) with the system.

Systems

Unfortunately there wasn't really a successfully implemented AI system in the final version of this project. The system is really just rolling dice against a static field of a Person, such as their extraversion or their desired depth of friendship. There isn't really much of a choice made.

However there were several systems we attempted to do:

Partial Order Planners

Alice found an open source Java partial order planning library (http://www.philippe-fournier-viger.com/plplan/) and both of us attempted to hack at it to convert it to our needs. I personally tried to have the states be a String form of all the relationships the person had; the actions meeting, befriending, and romancing others; and the pre-conditions stopping persons from trying to romance strangers and more importantly stop them from using more than their allotted action points. Alice and I both ran into issues getting "no solution" from the library and sometime the error was in French. Due to the lateness of the attempt, we stashed it and hoped we could talk to a TA about it. We could not. You can see at least my attempt on the *Partial-Order-Roles* branch on the repo.

Millington and Funge Goal-Oriented Planning

Knowing full well that this was thought too simple from a group of our size, we attempted to cobble something similar to the discontentment solution as shown in lecture. Instead of reducing discontentment, we wanted to maximize reward, since we already could calculate the success of a goal. This was attempted again for relationships and not for careers since they were not merged into master yet. The way it would work is it would create a shallow graph where the states were the goal progress and actions were either meeting or progressing relationships. At first, the only vertices were a start goal progress state and the goal progress state after taking exactly one of all the possible actions (a random sample of strangers to meet and progressing a relationship with a known person). From there, we would do a depth first search, creating new vertices as we went down. If we ran into a state with a similar progress and similar number of action points left, we would not expand upon it. Whichever chain of actions that led to the best state was applied to the person.

Unfortunately, due to the mutative nature of applying actions to the person, calculating the progress, and then reversing the actions, it got much messier. Due to holes in the relationship code, randomness in our code, overall intertwining of much of the code, there were mutation bugs practically everywhere. To see the work I did on it (with some git merge in the way), you can look on branch *Attempt-At-MF-Discontent.*

Failures and Setbacks

Above I listed some of the failures we faced when implementing AI. Other issues we ran into were purely communication. I was much more active in our Messenger group than the others which meant information was not shared in a timely way. Furthermore, while I definitely tried to encourage posting designs ahead of time so we had time to review them, deadlines were often not followed (even by myself). I will also say that I was focused much more on the engine than I was on applying the AI in class and that may have steered us in the wrong direction.

Also there are some out of memory errors if your parameters are too ambitious (like running for 100 years with the last-minute changes.)

Successes

While it may not be much, it looks like the romantic relationships formed between people had a higher initial desire computation (which takes into account attraction to physical traits, personality, romantic role compatibility) than non-romantic relationships. Sarah reported similar results with the career

system, with more tenacious (hard working) workers climbing the career ladder more quickly and more focused workers staying in the same career more than their counterparts.

Evaluation

Sarah came up with the following metric for how careers should work in the system:

> *People with high Openness will join the Journalism and Visual Art careers and be less likely to join the Medical and Law Enforcement careers. People with high Extraversion will join the Journalism, Law Enforcement, and Teaching careers and avoid the Visual Art career. People with high Conscientiousness will join the Medical and Teaching careers and be less likely to join the Visual Art career. People with high Agreeableness will join the Medical and Teaching careers. People with high Neuroticism are less likely to join the Teaching and Law Enforcement careers.*

> *Workaholics will stay in their occupation longer, advance faster in their careers, and retire later Slackers will stay in their occupation longer, get fired more often, and retire later Job Security Seekers will stay in their occupation longer and retire later Jack Of All Trades will advance faster in their careers.*

I asked her if that seemed to be the case and she said that based off what we saw in the output that seemed to work.

Below are Alice's metrics on relationships:

With action points, people's actions should depend on their roles.

> ***Romantic goals:***
>
> *People with higher monogamy values should be more likely to have a significant other, or a high desire with one other person*
>
> *People with higher flirtation values should be more likely to have medium desire with a lot of different people*
>
> ***Friendship goals:***
>
> *People with higher friendliness values should be more likely to have more relationships with positive regard values*
>
> *People with higher depth values should be more likely to have a few friends with extremely high regard values*
>
> *Without action points, people's actions shouldn't differ too much, but should be logical.*
>
> *People should only get into romantic relationships with people who were previously their friends*

People should be more likely to get married if they have a regard and desire values

According to the goal stats outputted in the test output in the repo, these are the averages for persons who got to at least age 50:

Goal Type	Mean Success
Romantic	0.613171
Platonic	0.486096
Career	0.582708

The results are not stellar but I do wonder if it's because the goal generation is just too extreme for the number of times we the people do anything per year.

Lessons Learned

The Final Project prompt said to be honest in this section. I've come away from this project very bitter. I know I mismanaged my time and could have done much better, but I feel like working on a team and trusting them to do things and work by deadlines has left a very bad taste in my mind. So I suppose I need to check in more often with group projects. I've also learned to be a bit more discerning with code reviews. I think due to our collective procrastination I had to be quick but it ended up costing me later.

I think the biggest lesson I learned was that I shouldn't mix personal projects with school work, especially when working with people who I've never worked with before. I still plan on expanding this project when this is all done and doing it on my own terms, but I will likely revert much of what this project has done. Who knows, maybe I'll actually get some working Goal-Oriented Planning in.