CSCI 1933 Project 4: Space Explorers

Due date: 04/29/2022, 11:55pm

Introduction

Up to this point in the semester, you've had a fairly specific format for your projects, with skeleton code that we've given you and specific methods that we've asked you to implement. This project is going to be a bit different. This time, your job is to implement a strategy for a game. You have quite a bit of flexibility in how you implement the strategy; we just have a specific way to call your strategy in the end. But more details on that later; first let's go over the game rules and how it works.

The Game

Premise

Space Explorers is a game where two players are pitted against each other in a system of interconnected planets. The players each start out on a home planet and the object of the game is to have the largest population by the end of the game.

Game Board

The game board is set up as a graph of planets interconnected by weighted edges.

- Population growth occurs at a set rate on each planet.
- If two planets are connected by an edge, then shuttles can be sent between these planets. The distance between the planets affects how many turns it takes for shuttles to move from one planet to the other.
- The planets vary by two factors:
    - Size correlates to the total population the planet can support. Once a planet hits its maximum population, population growth will cease and any population that exceeds the maximum will decrease by the rate described below.
    - Habitability correlates to the population growth rate on a planet. Population change after one turn for a given planet is defined below.
        - Let c = current population, m = max population, g = growth rate, and p = overpopulation penalty.
            - If $c < m$, pop next turn = $c*g$
            - If $c >= m$, pop next turn = $c - (c-m)*p$
        - Currently, p = 0.1 and g = 1 + (habitability / 100)
- Each player starts out with one planet with a given population. A player can send shuttles with explorers to neighboring planets on their turn.

- Each player receives information about the entire game board, including the planet IDs of all planets and which planets are interconnected. However, a player can only see detailed information (population percentages, size, habitability, incoming shuttles) about the planets that their people have the majority on and their neighboring planets.

Game Flow

In one turn, a player

- Receives information about the game state
- Adds moves to the event queue
    - A 'move' is sending a shuttle from one planet to a neighboring planet. The player can set how many explorers are sent in the shuttle.
    - A player can make as many moves as they want in a turn.
- Returns the event queue to the game engine

The game engine will then make all legal moves in the event queue, and allow one unit of time (one full turn cycle) to pass, in which

- Population growth (or decay, if overpopulation cap) occurs on all planets
- All shuttles move one step

The game play then passes to the other player.

Shuttles and Landings

A shuttle carries some amount of population from Planet A to Planet B. Let's say that Player 1 sent a shuttle with 100 explorers, and the distance between the planets is 2. The shuttle takes two full turn cycles to arrive at Planet B, and during that time, no population growth occurs on the shuttle. There are several possibilities when the shuttle arrives at Planet B.

Player 1 Has Majority on Planet B

Since Player 1's population is the majority on the planet, the explorers in the shuttle will always be added to the population as their people are able to accommodate some overcrowding with their own people if necessary.

Player 2 Has Majority on Planet B

If the population on Planet B has not hit the population cap, the explorers in the shuttle will simply be added to Player 1's population on the planet. If the population cap has been reached, the explorers on the shuttle will be lost since the player's population on the planet does not have resources necessary to accommodate overpopulation.

Planet B is Neutral

Player 1's explorers are added to the population if the population cap has not been reached.

Majority populations will give precedence to shuttles that they recognize (“friendly shuttles”), so all friendly shuttles will land before any others are able to land.

End of the Game

The game ends when one player has a majority population on all planets or a maximum number of turns is reached. If the maximum number of turns is reached, the player with the larger total population at that point wins the game.

For more information about the game flow, see the “SpaceExplorers_ExampleRound.pdf” document we posted on Canvas along with this assignment, which walks through an example round of the game.

Your Task

Write a class that implements `iStrategy`. As you will see when you look at `iStrategy`, there is one required method that needs to be written. (Of course, we strongly recommend that you do **not** include all of your logic in one method, due to various reasons that we have discussed throughout the semester.) At each turn, the `takeTurn` method in your strategy class will be called with parameters that include a list of the planets in the system and a queue for you to add your moves to. You will use the information in the list of planets and other information that you get from interacting with the system to make intelligent decisions about what moves to add to the queue.

We have provided you with a very simple strategy to give you an idea of how to interact with the game engine, add moves to the queue, and access information about the game space.

There are many different possible strategies, so each person/team’s approach may look very different. However, we do want each of your strategies to satisfy a few basic specifications:

- You must use the move queue properly and add moves at some point throughout each game.
- Not counting the move queue described above, you must use at least **three** of the following abstract data types in reasonable ways to help in implementing your strategy: lists, queues, dictionaries (maps), stacks, or graphs. Feel free to use Java built-in data structures that implement these ADTs (e.g. for a dictionary, you can use the HashMap class). **NOTE: we are not requiring you to implement your own data structures for these.**
- Your strategy must attempt traversal of the graph in some form. What exact traversal you use or what data you use in deciding how to traverse the planet graph is flexible, but your strategy should involve some attempt to traverse the graph.

In addition to the `takeTurn` method, you will need to implement two methods in your Strategy class that help us identify your submission:

`public String getName( )`: this method will be used as your team name on any scoreboards we report.

`public boolean compete( )`: return “true” in this method if you want your submission to compete in the class tournament, “false” if not.

Grading

Due to the unique nature of this project, you will be graded in a different manner than previously. Your strategy will be tested against several strategies that we have created. You will be graded based on performance against these strategies as well as on your proper use of data structures and the style of your implementation. We will provide you with these strategies in advance so you have a benchmark for how your strategy actually performs.

Grading components:

- Performance against our example strategies: 50 pts
  - If your solution can beat our RandomStrategy in > 70% of the trials (out of 100 random runs): 30/50 points
  - We will provide you with three additional non-random strategies (e.g. AI1Strategy, AI2Strategy, and AI3Strategy). For each additional strategy you can beat in > 70% of the trials (out of 100 random runs), you will earn an additional 10 pts toward the 50 total (a maximum of 50 pts can be earned).

- Appropriate use of requested data structures (at least **three** of lists, queues, dictionaries, stacks, or graphs, not counting the move queue): 30 pts

- Overall design and style of your strategy implementation: 20 pts

Running the Game

This example assumes that you’re using IntelliJ as your Java IDE.

1. Download the assignment from the class Canvas and unzip the folder
2. Import the folder “project4” as a project into IntelliJ
3. On the libraries screen, uncheck the “strategies” folder
   - This can be done later if necessary via File -> Project Structure -> Libraries
4. To add a strategy of your own, create a new java file in the folder project4/src/spaceexplorers/strategies
   - Use the provided example strategies as a reference
   - Note that you only need to edit the takeTurn method
5. To run the game WITH graphics...
   - Open the file project4/src/spaceexplorers/publicapi/Driver.Java

- Edit the file to call your strategy instead of the provided strategies:
  E.g. GameWindow window = new GameWindow(Strategy.class, false);
- To create a .jar file for your strategy:
  - For a strategy at project4/src/spaceexplorers/strategies/Strategy.java…
  - Create a .jar file for your strategy by going to File -> Project Structure -> Project Settings -> Artifacts -> Plus sign -> JAR -> from modules with dependencies and click 'OK'
  - Name your strategy so that it matches your class name (i.e. RandomStrategy -> RandomStrategy.jar), and set its output folder to project4/strategies
  - Press ok, go to the menu Build -> Build Artifacts -> Build
- Then, run the game by running the Driver.Java class, which has a main method. Be sure to set your Working directory to the parent folder for the project (this contains the subfolders src, strategies, graphs, etc). Set your Working directory by going to Run -> Edit Configurations and then updating the "Working directory" field.

6. To run the game WITHOUT graphics (this can be useful for testing your strategy)...
   - Open the file project4/src/spaceexplorers/core/SpaceExplorers.Java
   - Edit the file to call your strategy instead of the provided strategies
     - For a strategy at project4/src/spaceexplorers/strategies/MyStrategy.java…
     - Import your strategy with import spaceexplorers.strategies.MyStrategy
     - In the main method, change strategy1 to be an instance of MyStrategy
       - E.g. with IStrategy strategy1 = new MyStrategy();
     - Then, run the game by running the SpaceExplorers.Java class, which has a main method. Be sure to set your Working directory to the parent folder for the project (this contains the subfolders src, strategies, graphs, etc). Set your Working directory by going to Run -> Edit Configurations and then updating the "Working directory" field.

Public API

The API (interfaces you will be working with are listed below) - -

1. IEdge - Interface denoting an edge between two planets.
2. IEvent - An event is a wrapper around a population transfer from a source and destination planet.
3. IPlanet - A planet which has not been discovered yet.
4. IPlanetOperations - An interface denoting the events scheduling the movement of people.
5. IShuttle - An interface to represent the movement of people.
6. IStrategy - Your strategy class should implement this interface. Essentially, your strategy class should move a fixed number of people from planets where you have people to a destination planet such that you increase the number of planets you have population on as the game progresses.

7. IVisiblePlanet - A planet which you have conquered or is adjacent to you. You are able to see a complete set of characteristics for these planets.

Have a look at the RandomStrategy.java file for an example on how to use these interfaces.

Submission

Please write a brief description of your strategy, including what data structures you used to implement it, in the comments of your class that implements iStrategy. Then, zip all Java files (.java), including your Strategy class, and submit them through Canvas.

For this project, we request that you submit THREE different files:

(1) A .jar file of your strategy (see instructions above on how to build this)
(2) A .zip file with your entire project directory, including any external libraries you used. If you used IntelliJ/maven to get the external libraries, be sure you select the "download" checkbox so that the .jar file is actually included with your project code.
(3) A separate README.txt file that summarizes your strategy (see more details below)

Summary of what to include in your README file:

a. Your name(s) and x500.
b. Contributions of each group member (if working with a partner)
c. A paragraph summarizing your strategy.
d. A second paragraph that lists the data structures you used in implementing your strategy along with a brief justification why each data structure is appropriate for the task in which you used it.
e. If you created multiple strategies, specify which strategy (and the corresponding .java file) you want us to grade.
f. If you used external libraries for your implementation, please also describe these in the README and what you used the external libraries for.
g. A summary of any known issues with your strategy if they exist.
h. Any outside sources (aside from course resources) consulted for ideas used in the project, in the format:
    i. idea1: source
    ii. idea2: source
    iii. idea3: source
i. Include the statement:
“I(We) certify that the information contained in this README file is complete and accurate. I(We) have both read and followed the course policies in the ‘Academic Integrity - Course Policy’ section of the course syllabus.” and type your name(s) underneath.

Working with a partner

You may work with one partner to complete this assignment (max team size = 2). If you choose to work as a team, please only turn in one copy of your assignment. Please place your partner's and your names and x500s in a comment in each .java file you submit and in the README file. Do not share code with students other than your partner.