Student ID: 170006953, 160027195

Mercurial link: https://zz35.hg.cs.st-andrews.ac.uk/Project3/

Project3: StarCraft II

Instruction

This programs runs by simply compiling and running through terminal. Upon start up, the program will open a GUI that keeps track of the elapsed run-time of the game, as well as the amount of minerals and vesper gas available to the player. Control of the program is achieved through terminal input and the program's responses are printed out to the terminal.

Usage:

1. javac *.java
2. java Main.
3. select the goal you want.
4. enter the game you would try.
5. If you want to quit, close the GUI.

Design

- Simulation

Overall:
There are 13 classes used in this section. The most important files are "process.java" and "typeOfConstruction.java". "Process.java" serves as the center of control for the whole project, including gathering the information from a user and calling different methods in reaction to instructions from the user. The other classes are used to represent the different characters in this game, all of which extend from "typeOfConstruction.java". This is because all of the characters share some of the same features in this program, and thus it makes sense for them to inherit from this parent class.

Timer:
In the simulation, the time operates on real time, which means the real game can be accurately attempted by the user with the suggestions of the optimizer. A variable called "secondsTotal" is created to keep track of the elapsed run time. It is increased by one in every second in real time by using a "TimerTask".

Labelling units and buildings:
All the units and buildings in this program are labelled with an integer. Begining with 1000, and increasing by 1000 for each units or building. For instance, the potential IDs of the Nexus range from 1001 to 1999, and the potential IDs of the Probe range from 2000 to 2999. The reason for this is that it simplifies calculation of unit quantities. The program directly use the biggest ID of this units/buildings minus the smallest one. Another reason is that using integers make it easier to find the element in the array list and hash map which will be discussed in the next section.

Array List and Hash Map presenting the whole game:
A Hash Map and two Array Lists are used to store the information of the game. A Hash Map called "totalMap" is used in this program. Its key is the unit ID, and the corresponding value is the time when it began construction. Moreover, an array list called "idList" is declared to store the newest

ID of each unit/ building, as well as another array list called “initialList”, which is immutable, that stores the smallest value for each units/ building.

The function of the Hash Map is that every key, (units/ buildings), has its corresponding value (beginning construction time). By using the current time in the game minus the beginning construction time, which equals to the existing time. With getting its existing time, the program can judge whether it can be used or it is under constructing.

Furthermore, in order to find the specific units/buildings in the Hash Map and the number of its, “initialList” and “idList” are used. Every units/ building have an immutable position in the array list. For example, in this program, the position of 0 in the array list must be Nexus, and the position of 1 in the array list must be Probes. With using these array list the number of specific units/buildings. For instance, if getting the position 0 in the “initialList” is 1000, and “1003” in “idList”, the program using 1003 minus 1000 to get there are 3 Nexus in the game. Similarly, if Nexus need to be found in the Hash Map, using “idList” and “initialList”, we can get there exist 1001, 1002, 1003. Therefore, three Nexus are gotten including their status whether it is under construction or can be used.

The Construction of the Buildings/units:

In this part, different buildings are classified into various types according to their dependence. By classifying them into different part can share some same classes and methods. The result of constructing the buildings and units is that its beginning time together with it ID is put into Hash Map and Array List.

In the following sections, a representative unit is taken to demonstrate the design of them. The rest of the buildings/ units which is not mentioned share the similar situation.

Probes:

There are three functions of this unit, which are building probes, gathering minerals and gathering gas. There is a class called “probe” that achieves the all the mentioned above functions.

Firstly, for building probes, there are two things need to be checked. For one thing, whether it is enough minerals to construct. This judgment is completed by a method called “constructionJudgement()”. This method will return true or false to present whether it is possible.

For another thing, checking whether Nexus is occupied by another probe constructing. This function is achieved by the method called “checkFacilityAvailable()”. In the specification, in the process of constructing the unit, buildings cannot be used to do another thing. To satisfy this function, a HashMap is created for Nexus, whose key is the ID, and the value is Boolean. Ture presents it is allowable to use this facility to build units. For example, if there is a Nexus, the Nexus Hash Map will be {1001=>true}. If the Nexus is used to build a probe whose ID is 2001, the Nexus Hash Map will be {2001=>false}. Moreover, the process of updating the Nexus Map is conducted by a method called “updateNexusMap()” in “process.java”. This method will be called if the user wants to build the new probe. The main idea of this method is that according to its ID in the Nexus Hash Map, its beginning time will be found via “totalMap”, and if its existing time larger than its constructing time, the value in Nexus Map will be set to true, which means Nexus can be used to build next probe.

Secondly, as for gathering gas and minerals, there are three functions in this part. Firstly, if the number of probes excessed the limitation of the minerals patch and gas geysers, there will be an altar by this program. In order to complete this function, there are two array lists “mineralPatchList” and “gasList”. The reason to create this list is to calculate the number of probes working for minerals and gas. For example, if there is 8 minerals patch, the “mineralPatchList” will be {0, 0, 0, 0, 0 , 0, 0, 0}. If there is one probe working for the first mineral patch. The patch list will be changed to {1, 0, 0, 0, 0, 0, 0, 0}. Secondly, in order to calculate the velocity of gathering minerals, because the velocity is different until the third probe in a patch. A method called

“MineralCalculator()” is used. If the element in the patch list is equal to 3 (patch list: {3, 0, 0, 0, 0, 0, 0, 0}). the variable called “numberMaxVelocity” will be 2, which means there are two probes working with the highest velocity. On the other hand, “numberMinVelocity” will be 1, which means there is one probe working with the lowest velocity.
Thirdly, if some probes are assigned to gather minerals, the program will automatically arrange them the higher velocity minerals patch. For example, if the minerals path list is {2, 1, 0, 0, 0, 0, 0, 0}, and the user wants to set a probe to gather mineral. The probe will be set to the third patch rather than the first patch, because the velocity will be slower, coming to 3 in the first patch. To satisfy this function, the smallest element will be found every time, and the probe will be assigned to it. Moreover, if the number of the free probe is not enough to assign to gather minerals, the program can switch some probes from gas geysers to minerals patch

- Optimizer

Overview:

In this part, it is processed in the class called “optimiser”, which is extends from the class “process”. All the five goals are process by different methods.

The following results are shown the time to finish each goal.

1st goal costs 7 mins 28s. **2nd goal costs 7 mins 57s.**

3rd goal costs 8 mins 2s. **4th goal costs 8 mins 27s.**

5th goal costs 13 mins 21s.

Timer:

In this part, the time is gaming time. A while loop is used to model the whole game. If the loop is finished each time, one second passes. Until the goal is finished, the while loop will stop.

Goal:

Instead of completing the goal directly, small goals are set in different time. Time can be divided into two stages. The first stage is accumulating minerals and gas by building probes, and the second stage is achieving the goal. After setting different number of building probes, a conclusion is drawn that only building 12 more probes are suitable for finishing all the goals.

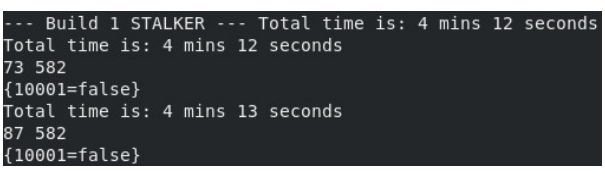
Design:

In the whole process, if the minerals and gas are left very close to zero, it is more likely to have the fastest way of finishing the goal. Therefore, the balance of minerals and gas together with building order are important in this optimizer. Because the velocity of gathering minerals is much faster than gas, the program will be set the probes to gather gas, when the minerals are much larger than gas. On the other hand, the building order is changed according to different simulation. Finally, the strategy is present in the terminal.

- GUI

The GUI was designed in order to allow the user a greater gameplay understanding of the optimization. Using the NetBeans Platform IDE Swing Framework, a jTextArea, three jLabel values, a series of buttons, as well as a jTextField are shown to the user.

After the optimization code determines the best build order and the terminal prints out the order and timings, the GUI begins to keep track of the game’s play time, the amount of minerals the player has, and the amount of vespene gas. It does this by creating a new object of the process class as well as a timer. The timer then checks the static time, mineral and gas variables of the process class and updates the GUI accordingly using a set text method.



This is as far as the GUI implementation was able to go, although in dummied out code, it is possible to see the intended next steps which were to move the text being printed to the console over the jTextArea by writing this text to a document and displaying it using the .read method. Then, by using buttons and the jTextField, the user would have complete control over the simulation, thus circumventing the need for direct terminal interaction.

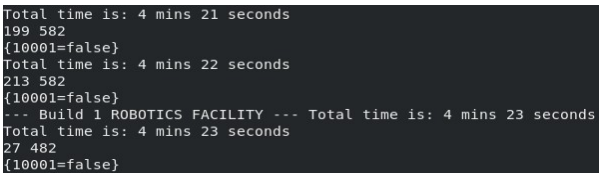
Testing

1. ***Optimizer Testing***

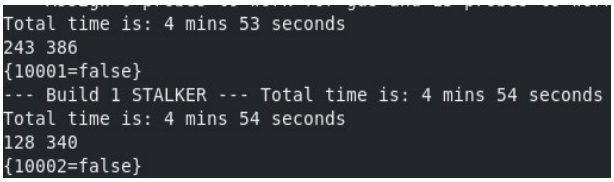
1.1 Deduce the minerals and gas.

The number under the time is shown as the amount of minerals and gas at that time.

(Minerals and gas are deduced correctly by constructing Robotics Facility)



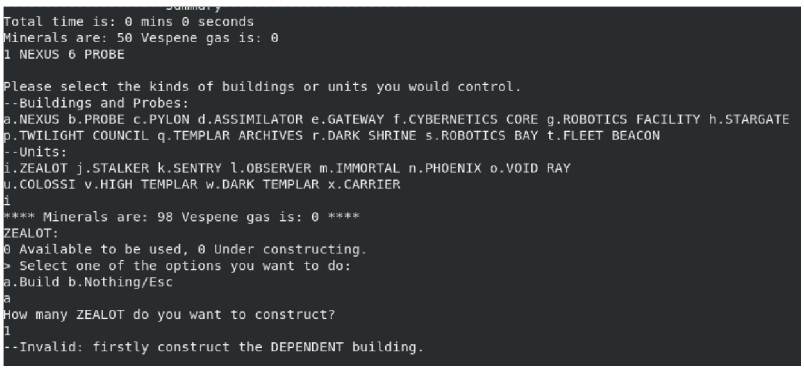
1.2 Constructing units. (The Gateway of Hash Map is changed into false, because stalker is under constructing.)



(After 42s, the stalker is finished. Therefore, another stalker can be built, and the Gateway Hash Map is changed to the second stalker ID.)

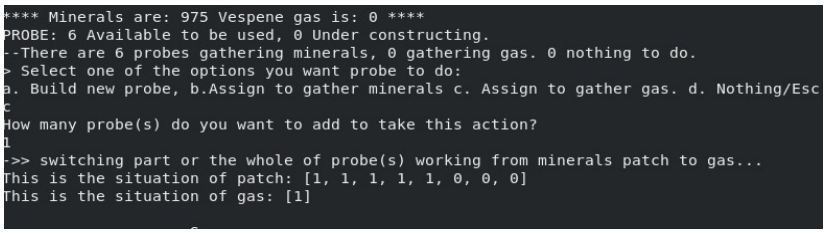
2. ***Simulation Testing***

2.1 Invalid building: lack of dependent building (example: user want to build the Zealot without gateway).

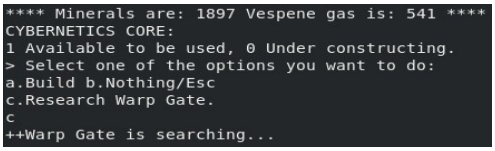


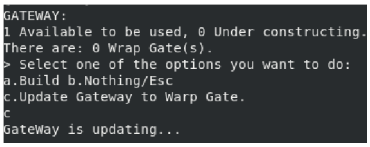
2.2 Assign one Probe to gather gas.

(There are 6 probes gathering minerals at first. After assigning task, there are 5 probes gathering minerals and one to gathering gas, which are shown in the two Lists at the end.)

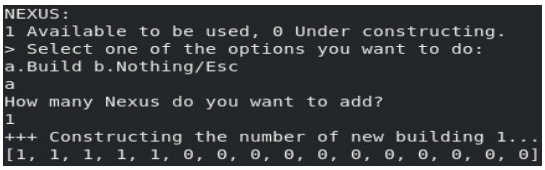


2.2 (Extension) Update To Wrap Gate. (When the cybernetics core is built, this option will be shown)





2.3 (Extension) Build Nexus (if another Nexus is built, the minerals patches will add more 8).



Extension Work

- Extra Bases and Resource Depletion
  when a new Nexus is constructed, 8 new mineral patches and 2 gas geysers are extended. At the same time, probes can be assigned to gather minerals and gas via this new nexus. This extension is done in the simulation.

- Upgrade Gateway to Warp Gate
  Before updating to Wrap Gate, it need to be researched in Cybernetics Core. And then, Gate Way can be upgraded, when the option is selected. The result of updating Gate Way is that the construction time of units is shorten. This extension is done in the simulation.

- Realistic Timing
  In the simulation of this project, players taking decisions is not assumed instantly. Therefore, even though user consume time to enter his/her instruction, the process of gathering minerals and gas never stop. Moreover, the time clock in this simulation is real time.