Starcraft

Overview

The specification required that a simulator and build order optimiser be produced. The program should simulate a Starcraft II game from a single players perspective and should suggest ways in which a player could reach specified game states.

The following extensions were implemented

- Addition of resource suppply
- Tech lab and reactor addon
- All Unit types
- All Upgrades
- Orbital Command
- Mule
- Constructing buildings with workers
- Extra Bases and Base Depletetion
- Realistic Timing (adjustable game second)

Assumptions:

- Ghost Academy and Fusion core are permanent upgrades https://liquipedia.net/starcraft2/Ghost_Academy_(Legacy_of_the_Void)
- contrary to specification and further supported by sources on:
    - barracks
    - factory
    - starport

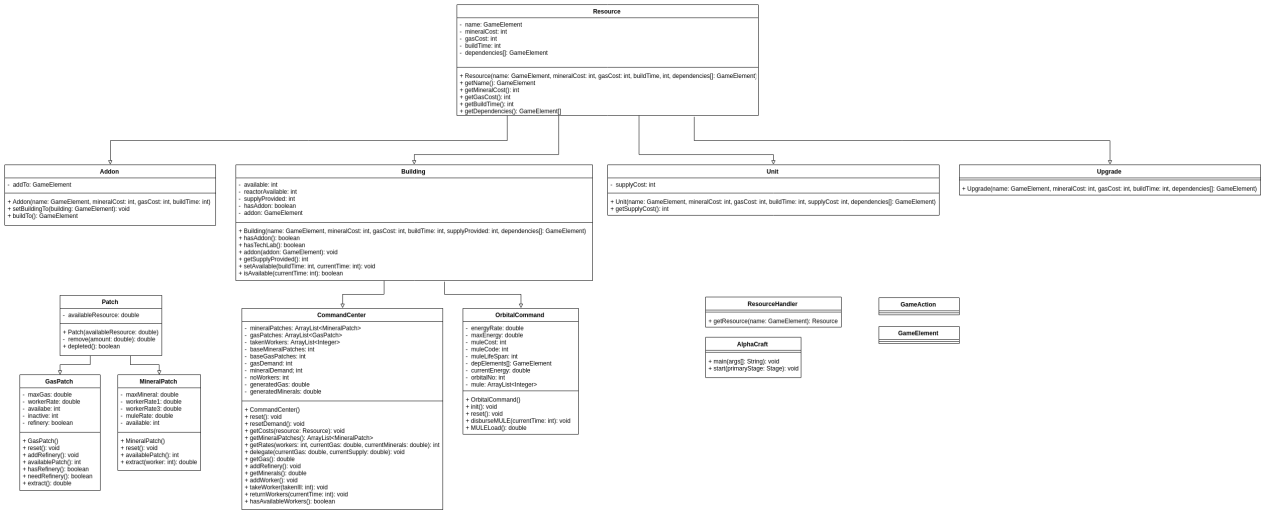
  support addons
- Buildings can only have one addon (supported by sources)
- There can be multiple mules on one patch
- Builds which cost over 200 supply can be instantly rejected as game this max supply.
- You require units to actually upgrade.
- The program can run infinitely it long update the current Optimal when it finds one;

Problem Decomposition:

- implement mechanism to find the possible decision
- implement mechanism to execute a decision
- implement way of represent of the game objects

Design

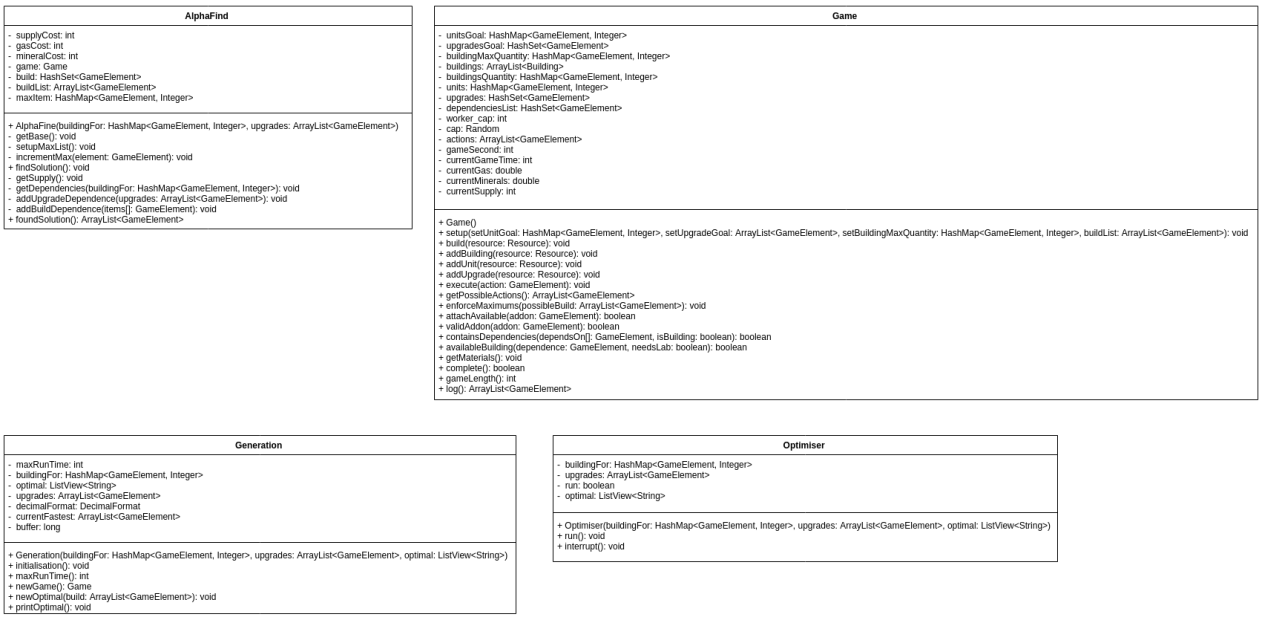
Resource



All the gas resource shared command attributes of their gas and mineral cost and build time so a class was made to represent this. Then from there they were broken into smaller sub groups. Since all the units share the exact same attributes they were generalised into a single class and all the different unit types were just instances of the Unit class. The same was done for upgrades and addons. Almost all the builds exist as just instances of a the building class but Command Center and Orbital Command which since both have special unique actions and properties were modelled into their own classes. It was decided that resource collection would be done via static methods from the Command Center as it would allow for resource to be collected for all the command centers and their bases and all for workers to be evenly distributed across of all the bases allow for maximum resource withdraw from patches. It was decided to model the Patches as class as they two had their own attributes such as the resource they had remaining and in the case of gas if it had a refinery or not. The attributes and functions Gas Patches and Mineral Patches shared in common were then generalised into a class from which then.

To be able to identify a resource a Use was made of Enum which were set to the name of the instances of the respective objects and also a `ResourceHandler` was made using case statements to return `new` instances of the resources. This allowed them to be ready used through out the program. As the object type could just be referenced directly from the enum.

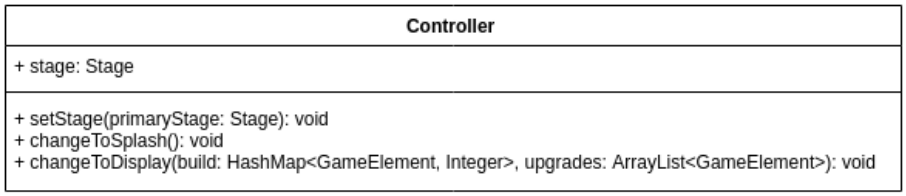
Search Strategy

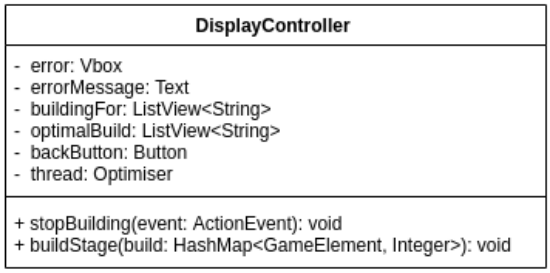


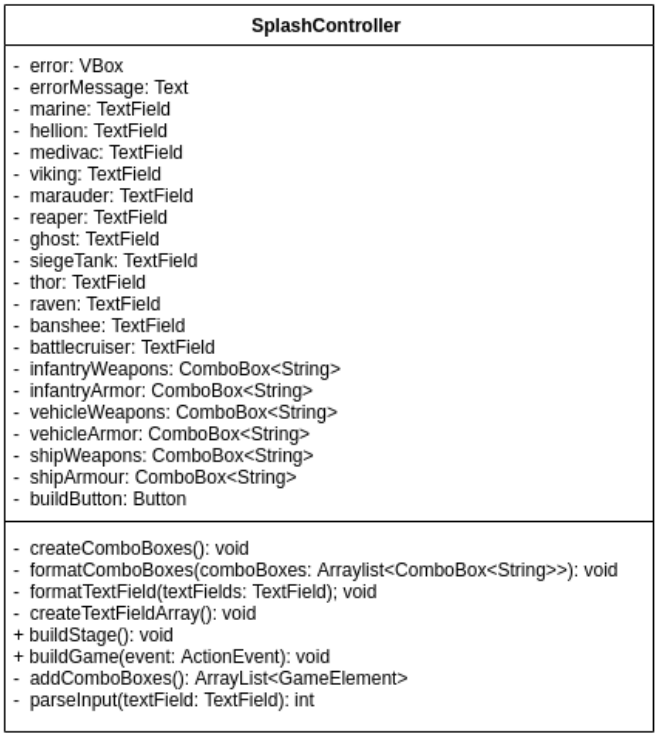
The search strategy used was a mixture of Heuristics and Random search. The program does not randomly assign workers but does it based on the demand for gas and minerals by the things it potential has to build. So when there is nothing with gas to build and especially when no refinery the program assigns all the workers to the mineral patches. Also the program only builds items which are dependencies of the units and upgrades it is attempting to build. This drastically reduces the search space and allows for random search to become a viable option. Also to decrease the search space first a Worse Case solution is found by only building one of the each of the dependencies of the unit and not buy any workers and this becomes the maximum time a random search is able to go for. When a solution is ran again the length of the new solution becomes the new maximum for which the search shall run for.

It was found that when this algorithm would run the algorithm would always over investing workers, potentially increasing length of solution. So it was decided max number of workers would use a random number between (1-30) *multipled by Command Center that will be built as the maximum amount of workers which would allow for a shorter solution to be found. It was also determined via heuristic the maximum buildings required depending on what they were building. Ghost academy and Fusion Core were if they were a dependence capped at 1 since they unlocked permanent upgrades. Engineering bays had a global cap of 2 as they could only build Infantry Armor and Infantry Weapons concurrently. Amoury was capable at 4 with similar logic.

GUI







To present the information and allow for input JavaFX was used. This allowed for a easy format for the user to be able to input data and a easy way for the data to be outputted.

Multithreading

To allow the `GUI` to and optimisation algorithm to exist concurrently use was made of multi threading. When the use wanted to find a solution a new thread would be created and would allow for the calculation to run independently of the GUI, but occasionally there seen to be access conflict error when trying to update items from GUI from instead the thread. So the use of a try catch to make thread fill the `listview` when the resource is not busy. Also to stop the input from continuous flashing so fast user can not see it is on update via a buffer at regular intervals

Testing

Test 1:

This test was used to ensure that the program produces build order for 6 marines:

Expected

The program shoud produce a build order which outlines the best method in which to build six marines.

Actual

2:46 BARRACKS: Quantity 3
2:51 SCV: Quantity 9
2:55 MARINE: Quantity 1
3:09 SCV: Quantity 10
3:13 BARRACKS: Quantity 4
3:20 MARINE: Quantity 2
3:26 SCV: Quantity 11
3:33 MARINE: Quantity 3
3:43 ORBITAL_COMMAND: Quantity 1
3:45 MARINE: Quantity 4
3:55 MARINE: Quantity 5
3:58 MARINE: Quantity 6

The program successfully produced a build order for 6 marines

Test 2:

This test was used to ensure that the program produces build order for 16 marines:

Expected

The program should produce a build order which outlines a build order for 16 marines and the time it takes to build the 16 marines using that specific build order.

Actual

4:13 MARINE: Quantity 7
4:15 MARINE: Quantity 8
4:26 SCV: Quantity 14
4:29 MARINE: Quantity 9
4:30 MARINE: Quantity 10
4:31 MARINE: Quantity 11
4:41 MARINE: Quantity 12
4:42 MARINE: Quantity 13
4:43 SCV: Quantity 15
4:54 MARINE: Quantity 14
4:56 MARINE: Quantity 15
4:57 MARINE: Quantity 16

The program successfully produces a build order for 16 marines

Test 3:

This test was used to ensure that the program produces build order for 50 marines:

Expected

The program should produce a build order which outlines a build order for 50 marines and the time it takes to build the 50 marines using that specific build order.

Actual

```
275:08 MARINE: Quantity 39
276:37 MARINE: Quantity 40
277:58 MARINE: Quantity 41
279:11 MARINE: Quantity 42
280:16 MARINE: Quantity 43
281:13 MARINE: Quantity 44
282:08 MARINE: Quantity 45
282:49 MARINE: Quantity 46
283:22 MARINE: Quantity 47
283:47 MARINE: Quantity 48
284:12 MARINE: Quantity 49
284:37 MARINE: Quantity 50
```

The program successfully produces a build order for 50 marines

Test 4:

This test was used to ensure that the program produces build order for 10 hellions:

Expected

The program should produce a build order which outlines a build order for 10 hellions and the time it takes to build the 10 hellions using that specific build order.

Actual

```
8:18 HELLION: Quantity 9
8:23 SCV: Quantity 31
8:28 SCV: Quantity 32
8:31 BARRACKS: Quantity 10
8:32 SCV: Quantity 33
8:35 BARRACKS: Quantity 11
8:40 BARRACKS: Quantity 12
8:41 SCV: Quantity 34
8:43 ORBITAL_COMMAND: Quantity 4
8:45 ORBITAL_COMMAND: Quantity 5
8:47 BARRACKS: Quantity 13
8:48 HELLION: Quantity 10
```

The program successfully produces a build order for 10 hellions.

Test 5:

This test was used to ensure that the program produces build order for 6 marines and 4 hellions:

Expected

The program should produce a build order which outlines a build order for 6 marines and 4 hellions and the time it takes to build the 6 marines and 4 hellions using that specific build order.

Actual

```
10:37 BARRACKS: Quantity 1
12:06 MARINE: Quantity 1
13:27 MARINE: Quantity 2
14:40 MARINE: Quantity 3
15:45 MARINE: Quantity 4
16:49 MARINE: Quantity 5
17:38 MARINE: Quantity 6
19:53 FACTORY: Quantity 1
21:02 HELLION: Quantity 1
21:57 HELLION: Quantity 2
22:32 HELLION: Quantity 3
23:02 HELLION: Quantity 4
```

The program successfully produces a build order for 6 marines and 4 hellions.

Test 6

This test was used to ensure that the program produces build order for 8 marines and 2 medivacs:

Expected

The program should produce a build order which outlines a build order for 8 marines and 2 medivacs and the time it takes to build the 8 marines and 2 medivacs using that specific build order.

Actual

```
13:03 MARINE: Quantity 1
14:32 MARINE: Quantity 2
15:53 MARINE: Quantity 3
17:06 MARINE: Quantity 4
18:19 MARINE: Quantity 5
19:16 MARINE: Quantity 6
20:05 MARINE: Quantity 7
20:46 MARINE: Quantity 8
22:34 FACTORY: Quantity 1
23:58 STARPORT: Quantity 1
24:49 MEDIVAC: Quantity 1
25:31 MEDIVAC: Quantity 2
```

The program successfully produces a build order for 8 marines and 2 medivacs

Test 7

This test was used to ensure that the program produces build order for 8 marines, 2 medivacs and 2 vikings:

Expected

The program should produce a build order which outlines a build order for 8 marines, 2 medivacs and 2 vikings and the time it takes to build the 8 marines, 2 medivacs and 2 vikings using that specific build order.

Actual

```
18:29 MARINE: Quantity 3
20:00 MARINE: Quantity 4
21:33 MARINE: Quantity 5
22:48 MARINE: Quantity 6
23:55 MARINE: Quantity 7
24:54 MARINE: Quantity 8
27:36 FACTORY: Quantity 1
29:56 STARPORT: Quantity 1
31:07 MEDIVAC: Quantity 1
32:04 MEDIVAC: Quantity 2
33:21 VIKING: Quantity 1
34:03 VIKING: Quantity 2
```

The program successfully produces a build order for 8 marines, 2 medivacs and 2 vikings.

Test 8:

This test was used to ensure that the program produces build order for 16 Marines, 8 Hellions and 3 Medivacs:

Expected

The program should produce a build order which outlines a build order for 16 Marines, 8 Hellions and 3 Medivacs and the time it takes to build the 16 Marines, 8 Hellions and 3 Medivacs using that specific build order.

Actual

```
7:23 ORBITAL_COMMAND: Quantity 29
7:24 SCV: Quantity 40
7:27 SCV: Quantity 41
7:28 ORBITAL_COMMAND: Quantity 30
7:29 ORBITAL_COMMAND: Quantity 31
7:30 SCV: Quantity 42
7:31 SCV: Quantity 43
7:32 MEDIVAC: Quantity 3
7:33 ORBITAL_COMMAND: Quantity 32
7:34 MARINE: Quantity 15
7:35 HELLION: Quantity 8
7:36 MARINE: Quantity 16
```

The program successfully produces a build order for 16 Marines, 8 Hellions and 3 Medivacs.

Advanced Goals

Test 9

This test was used to ensure that the program produces a build order to that outlines how to build: 2 Marines, 8 Hellions, 10 Siege Tanks and 2 Thors.

Expected

The program should output an optimal build order that shows how 2 Marines, 8 Hellions, 10 Siege Tanks and 2 Thors can be produced in a shown time.

Actual

```
342:39 SIEGE_TANK: Quantity 2
347:39 SIEGE_TANK: Quantity 3
352:02 SIEGE_TANK: Quantity 4
355:59 SIEGE_TANK: Quantity 5
359:38 SIEGE_TANK: Quantity 6
362:43 SIEGE_TANK: Quantity 7
365:28 SIEGE_TANK: Quantity 8
367:41 SIEGE_TANK: Quantity 9
369:32 SIEGE_TANK: Quantity 10
370:53 ARMORY: Quantity 1
372:42 THOR: Quantity 1
373:42 THOR: Quantity 2
```

The program successfully produces a build order that outlines how to create 2 Marines, 8 Hellions, 10 Siege Tanks and 2 Thors.

Test 10

This test was used to ensure that the program produces a build order to that outlines how to build: 16 Marines and 8 Marauders.

Expected

The program should output an optimal build order that shows how 16 Marines and 8 Marauders can be produced in a shown time.

Actual

```
42:31 MARINE: Quantity 14
43:05 MARINE: Quantity 15
43:36 MARINE: Quantity 16
44:12 BARRACKS_TECH_LAB: Quantity 1
45:17 MARAUDER: Quantity 1
46:14 MARAUDER: Quantity 2
47:03 MARAUDER: Quantity 3
47:44 MARAUDER: Quantity 4
48:17 MARAUDER: Quantity 5
48:48 MARAUDER: Quantity 6
49:19 MARAUDER: Quantity 7
49:49 MARAUDER: Quantity 8
```

The program successfully produces a build order that outlines how to create 16 Marines and 8 Marauders.

Test 11

This test was used to ensure that the program produces a build order to that outlines how to build: 16 Marines 8 Marauders and 4 Medivacs.

Expected

The program should output an optimal build order that shows how 16 Marines 8 Marauders and 4 Medivacs can be produced in a shown time.

Actual

```
124:42 MARAUDER: Quantity 3
127:54 MARAUDER: Quantity 4
130:59 MARAUDER: Quantity 5
133:37 MARAUDER: Quantity 6
135:58 MARAUDER: Quantity 7
138:09 MARAUDER: Quantity 8
140:50 FACTORY: Quantity 1
143:05 STARPORT: Quantity 1
144:18 MEDIVAC: Quantity 1
145:10 MEDIVAC: Quantity 2
145:53 MEDIVAC: Quantity 3
146:35 MEDIVAC: Quantity 4
```

Test 12

This test was used to ensure that the program produces a build order to that outlines how to build: 2 Marines, 3 Hellions, 8 Tanks and 8 Vikings.

Expected

The program should output an optimal build order that shows how 2 Marines, 3 Hellions, 8 Tanks and 8 Vikings can be produced in a shown time.

Actual

```
287:44 SIEGE_TANK: Quantity 6
292:50 SIEGE_TANK: Quantity 7
297:19 SIEGE_TANK: Quantity 8
301:22 STARPORT: Quantity 1
304:59 VIKING: Quantity 1
308:02 VIKING: Quantity 2
310:45 VIKING: Quantity 3
312:56 VIKING: Quantity 4
314:45 VIKING: Quantity 5
316:04 VIKING: Quantity 6
316:57 VIKING: Quantity 7
317:39 VIKING: Quantity 8
```

Test 13

This test was used to ensure that the program produces a build order to that outlines how to build: 16 Marines, 4 Banshees and 4 Vikings.

Expected

The program should output an optimal build order that shows how 16 Marines, 4 Banshees and 4 Vikings can be produced in a shown time.

Actual

```
43:14 MARINE: Quantity 16
45:46 FACTORY: Quantity 1
48:06 STARPORT: Quantity 1
49:09 STARPORT_TECH_LAB: Quantity 1
49:38 BANSHEE: Quantity 1
50:04 BANSHEE: Quantity 2
50:27 BANSHEE: Quantity 3
50:52 BANSHEE: Quantity 4
51:29 VIKING: Quantity 1
52:12 VIKING: Quantity 2
52:55 VIKING: Quantity 3
53:37 VIKING: Quantity 4
```

Extension Testing:

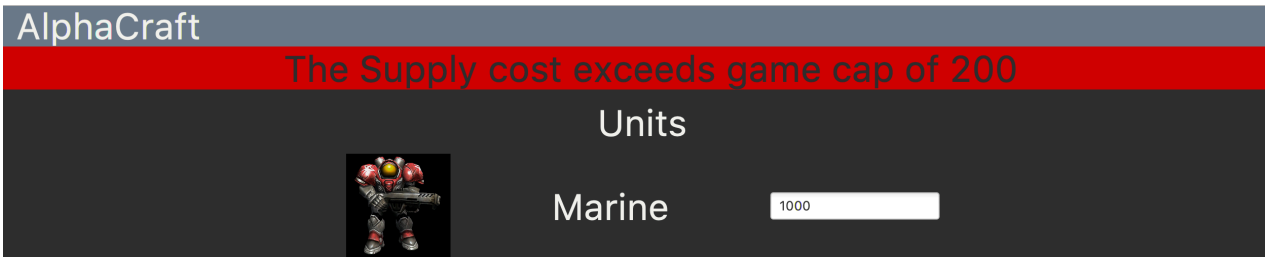
Test 14 - Supply

This test was used to ensure that the program prevents the user from building, for example, 1000 marines as this exceeds the maximum supply level of 200

Expected

The program should display an error message as this amount (1000) of marines would use more than 200 supply.

Actual



The program successfully displays an error message.

Test 15 - Orbital Command

This test was used to ensure that the program has the capability to upgrade Command Centers to Orbital Commands. In this test the program will try to produce the optimum build order for 100 marines.

Expected

The program should show use of orbital commands

Actual

```
52:07 REFINERY: Quantity 406
52:08 SUPPLY_DEPOT: Quantity 473
52:09 SUPPLY_DEPOT: Quantity 474
52:10 REFINERY: Quantity 407
52:11 REFINERY: Quantity 408
52:12 SUPPLY_DEPOT: Quantity 475
52:13 SUPPLY_DEPOT: Quantity 476
52:15 SCV: Quantity 463
52:16 ORBITAL_COMMAND: Quantity 432
52:18 SUPPLY_DEPOT: Quantity 477
52:19 COMMAND_CENTER: Quantity 412
52:20 MARINE: Quantity 100
```

The program successfully upgrades command centers to orbital commands depending on completed barracks.

Test 16 - Upgrades

This test was used to ensure that the selected upgrades were applied and shown on the outputted build order.

Expected

The build order should contain an upgrade for infantry weapons. To test this I will select the upgrade 'Infantry Weapons 3' In this test I will build 50 Marines, 8 Helion and 4 Tanks.

Actual

```
597:01 HELLION: Quantity 5
600:20 HELLION: Quantity 6
603:11 HELLION: Quantity 7
605:45 HELLION: Quantity 8
609:22 SIEGE_TANK: Quantity 1
612:25 SIEGE_TANK: Quantity 2
615:08 SIEGE_TANK: Quantity 3
617:19 SIEGE_TANK: Quantity 4
619:05 ARMORY: Quantity 1
620:39 INFANTRY_WEAPON_2: Quantity 1
623:07 COMMAND_CENTER: Quantity 2
623:49 INFANTRY_WEAPON_3: Quantity 1
```

The upgrade is successfully applied when appropriate.

Test 17 - Realistic Timing

This test is used to ensure that the game time is successfully modified depending on the users selection.

Expected

When the 'game second' is increased so too should the total build time.

Actual

```
0:00 COMMAND_CENTER: Quantity 1
0:54 REFINERY: Quantity 1
3:15 SUPPLY_DEPOT: Quantity 1
6:24 BARRACKS: Quantity 1
7:39 MARINE: Quantity 1
8:12 MARINE: Quantity 2
8:39 MARINE: Quantity 3
9:09 MARINE: Quantity 4
9:36 MARINE: Quantity 5
10:03 MARINE: Quantity 6
```

As expected the build time increases, giving allowing for a more realistic build time to be selected if neccecary.

Evaluation

The specification required that a simulator and build order optimiser be produced. It should simulate a Starcraft II game from a single players perspective and should produce ways in which a player can reach a specified game state. As shown from the testing above the program can successfully simulate resource gathering for one base and a variable number of workers, building and unit construction, contains a search strategy and outputs the build order. Thus fullfilling the basic deliverable. All extensions were implemented and a sophisticated search strategy was implemented as shown. From this it can be concluded that the program fullfills the Advanced Deliverable.

Conclusion

The specification was used successfully to produce a Starcraft II simulator and build optimiser. The program had to support simulation of resource gathering for one base, and a variable number of workers, including three per mineral patch, simulation of building and unit construction, a basic search strategy and should output this build order to fullfill the basic deliverable. As well as the basic solution all extensions were implemented and a complex search strategy was used to create and optimise the build order showing that the program fullfills the Advanced Deliverable.

Difficulties

- get input to show on list view with cause IllgalStateException
- Determination of the ideal ratio of mineral workers to gas workers.

With More Time

- More test cases could have been implemented to further prove that the program meets the specification.
- J Unit test could have been use for more vigorous testing
- Implementation of greater number of heuristic parameters to reduce search space.