



CS12320 MAJOR ASSIGNMENT

Bonks & Zaps

6TH MAY 2016

Jake McDonagh
jam93@aber.ac.uk

Table of Contents

Contents

Introduction	2
Descriptions of Classes and relationships	3
GameEngine.java	3
GridWorld.java	3
BeingInterface.java	3
Bonk.java	3
Zap.java	3
Location.java	4
GameLabels.java	4
Gender.java	4
Testing	5
Screenshot 1	5
Screenshot 2	6
Screenshot 3	7
Evaluation	8

Introduction

The following information within this PDF will contain the information that is relevant to the Bonks and Zaps project that was assigned to us on the 7th March 2016.

The PDF will consist of 3 major parts in its entirety; the *Design*, the *Testing* and the *Evaluation* processes that took place during my time and development on this project.

- The Design section of this file will focus on the classes and the relationships that the classes have for each other respectively. As well as this, there will also be several diagrams that will visually represent the program, including a UML case diagram and a sequence diagram.
- The Testing section will prioritise on displaying several screenshots of the prototype in action. Throughout this section I will be showcasing the different stages of the program's creation and will also be discussing the test data that was used and the results of said data.
- The Evaluation will primarily consist of the overall summary of the project. Here I will be describing how I initially started the project and the methods that I used in order to make progress within it. I will also be discussing the hurdles that I encountered along the way and the solutions I tried to deal with them. I will also be discussing about how I felt I did in this assignment and where I could have improved upon if I were to have a second chance at this.

Descriptions of Classes and relationships

GameEngine.java

The Game Engine class is the class that is used to ensure that the game activity happens within the program. It first begins by creating a new Game Engine object, which then allows it to also create a new Grid World object for further use within the program as well as a new scanner to allow the user to have manual input within the project.

GridWorld.java

The Grid World class is effectively the main centrepiece of the program, as it is essential to the initialisation of the first 20 bonks and 5 zaps, as well as allowing the UI to constantly change and update for the user as the bonks and zaps take their actions at every cycle. The class is linked to the BeingInterface interface (which in turn, the bonks and zaps inherit most of their methods) as well as the actual Bonk class, due to Grid World needing to get the ArrayList from the class itself.

BeingInterface.java

The Being Interface class is abstract due to the fact that it does not use any methods itself, but instead because it's the interface that both the bonks and zaps inherit from. It holds the main methods that are needed in order to allow them to act, as well as constantly receive the info about their current location. The interface is linked to both the bonks and zaps classes for the reasons mentioned above, as well as the location class.

Bonk.java

The Bonk class is the class that is used for the methods and the data that are quintessential to them. The class contain many different methods such as constantly receiving new locations as it's moving around (due to the inheritance from the BeingInterface class). The bonk class inherits the BeingInterface class along with GameLabels, as the bonks need a visual representation on the grid.

Zap.java

Like the Bonk class, the Zap class is primarily used for the methods and data that belong to them in the first place. This also includes the gets/sets for the location. It also contains the methods needed to move and act just like the bonks. The Zap class also inherits from the interface as well as the GameLabels class in order to have a visual representation.

Location.java

The Location class, although small, allows us to create the location co-ordinates for the bonks and zaps respectively. The class also allows us to return the current bonks/zaps row or column integer values, making it essential in order to allow them to act correctly. The Location class is linked to the Beings Interface since both the bonks and zaps need to have positions within the grid.

GameLabels.java

The GameLabels.java is an enumeration which is used in order to give the beings (either bonks or zaps) a visual representation within the grid. This is important as without this the entire grid would be blank and thus, the user would have no idea what is happening within the program. It is linked to both the Bonks and Zaps classes, as well as the Grid World class in order to apply the changes.

Gender.java

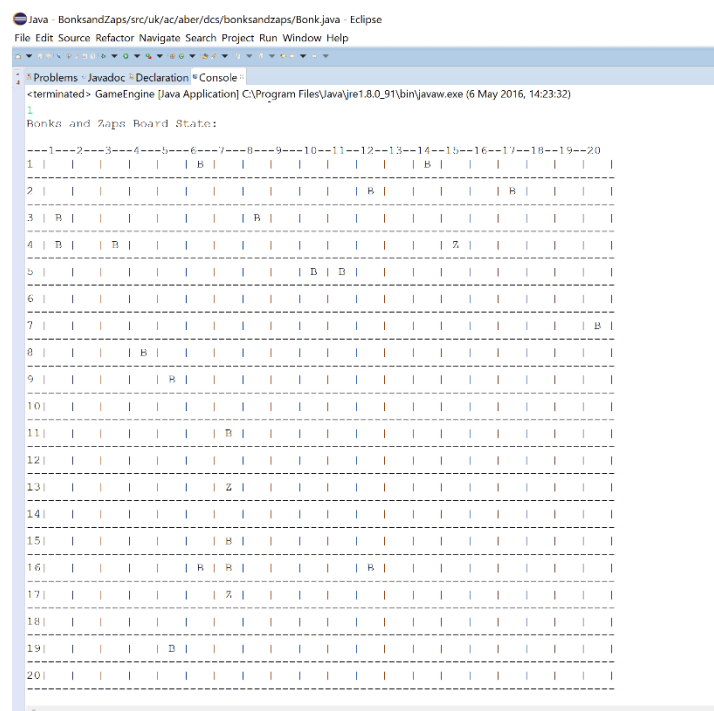
The Gender enumeration is used to give each being within the world a gender (for bonks this would be male/female, however, zaps are always given the gender of zap). This is used to allow the bonks to reproduce as well as to allow the zaps to kill the bonks, not other zaps. The Gender is only linked to the interface, as both the bonks and zaps inherit from it.

Testing

Within this section of the file, I will be displaying several screenshots that were taking during the course of the program's development. Some of the screenshots will be showing the visual improvement of the program, whilst others will be showing some examples of the problems that I encountered along the way and the solutions used to fix them.

Screenshot 1

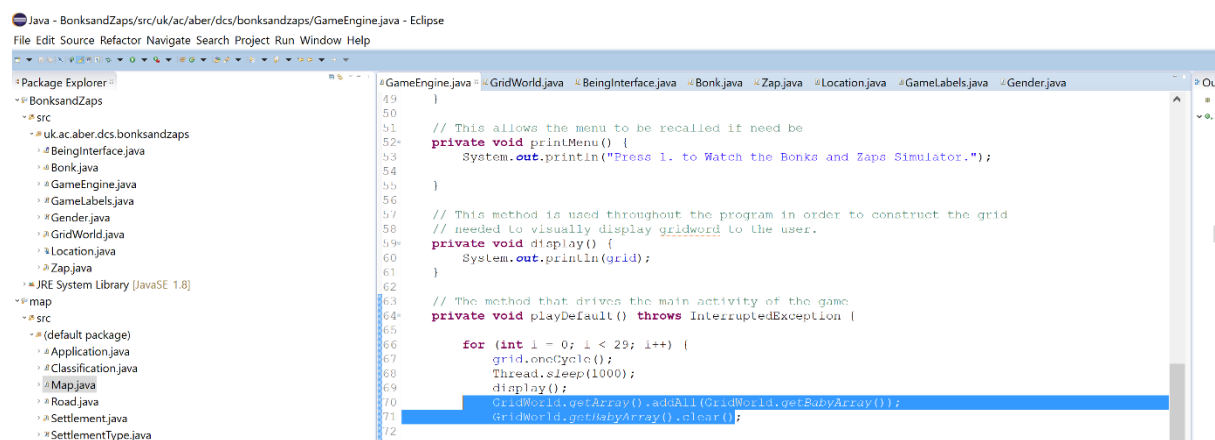
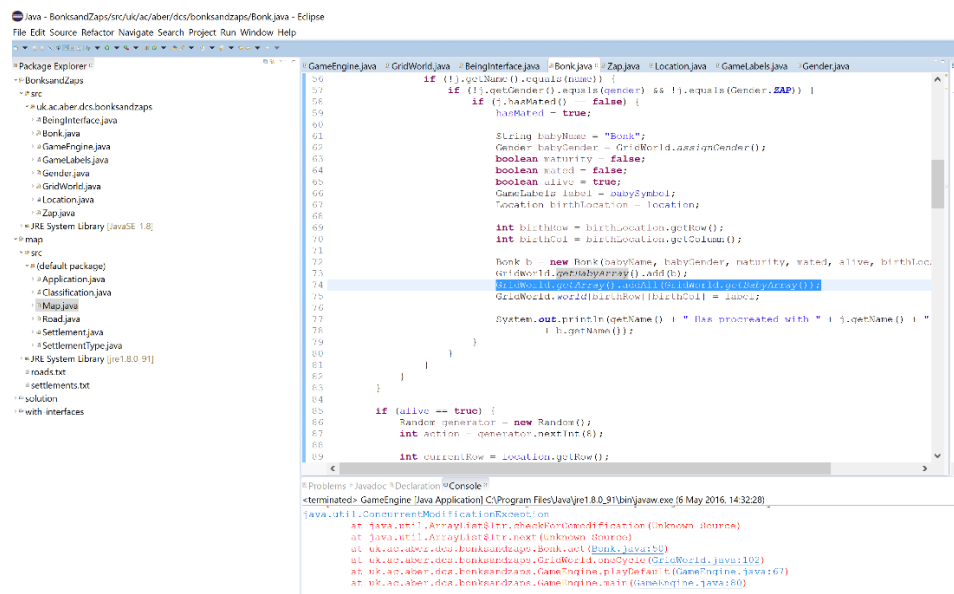
The following screenshot was the first time when I implemented the grid to allow the user to visually see the grid world. Although the grid is very basic and is not flashy, it however does display the location of the bonks and the zaps.



```
Java - BonksandZaps/src/uk/ac/aber/dcs/bonksandzaps/Bonk.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
<terminated> GameEngine [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (6 May 2016, 14:23:32)
Bonks and Zaps Board State:
---1---2---3---4---5---6---7---8---9---10---11---12---13---14---15---16---17---18---19---20
1 | | | | | | B | | | | | | | | | | | | | | | |
2 | | | | | | | | | | | | B | | | | | B | | | |
3 | B | | | | | | | | B | | | | | | | | | | |
4 | B | | B | | | | | | | | | | | Z | | | | |
5 | | | | | | | | | B | B | | | | | | | | | |
6 | | | | | | | | | | | | | | | | | | | | |
7 | | | | | | | | | | | | | | | | | | | B |
8 | | | | B | | | | | | | | | | | | | | | |
9 | | | | R | | | | | | | | | | | | | | | |
10| | | | | | | | | | | | | | | | | | | | |
11| | | | | | B | | | | | | | | | | | | | | |
12| | | | | | | | | | | | | | | | | | | | |
13| | | | | | Z | | | | | | | | | | | | | | |
14| | | | | | | | | | | | | | | | | | | | |
15| | | | | | B | | | | | | | | | | | | | | |
16| | | | | B | B | | | | | B | | | | | | | | |
17| | | | | | Z | | | | | | | | | | | | | | |
18| | | | | | | | | | | | | | | | | | | | |
19| | | | B | | | | | | | | | | | | | | | |
20| | | | | | | | | | | | | | | | | | | | |
```

Screenshot 2

In this screenshot, we can see that at some point during the development of my code, I was receiving a `ConcurrentModificationException` error in what appeared to be at random times during the running of my code. After some quick searching I found out that it was due to the fact that the loop (which is used in the `act()` method) causes an error when it finds out that the number of beings within the arraylist changes during the loop. In order to fix this problem, I had to take the piece of code that is highlighted below into the `GameEngine` class, after the point where the first cycle is completed and thus, at the point where the loop is no longer in use. I also made sure to clear the arraylist so that multiple babies with the same properties weren't being duplicated.



Screenshot 3

In this screenshot, we can see the first time when I give text info towards the user whenever an important event happens within the program (in these cases, when two bonks give birth to a new bonk and when a zap also kills a bonk). The following pictures are below:

```
17| | | | | | | | | B | | | | | | | | |
-----
18| B | | | | B | | | | | | | | | | | |
-----
19| | | | | | | | | B | | | | | | | | |
-----
20| | | | | | | | | | | | | | B | | | |
-----

Zap3 has zapped Bonk18 to death
Bonk18 has died

16| | | | | | | | | | | | | | B | | | |
-----
17| | B | | | | | | | | | | | | | |
-----
18| | | | | | | | B | | | | | | | | |
-----
19| | | | | | B | | Z | | | | | | | | |
-----
20| | B | | | | | B | | | | | | | | |
-----

Zap4 has zapped Bonk10 to death

16| | | | | | | | | | | | | B | | | |
-----
17| | | | | | | | | | | B | | Z | | | |
-----
18| B | | B | | | B | | | | | | | | |
-----
19| | | B | | Z | | | | | | | | | | |
-----
20| | B | | | | B | | | | | | | | |
-----

Bonk1 Has procreated with Bonk3 to give birth toBabyBonk2
Bonk1 Has procreated with BabyBonk1 to give birth toBabyBonk3
```


Evaluation

In this section, I will be evaluating the project that I have created and the steps that I took during the creation of it. I will also be discussing the faults with the program and how I could've improved on the program in general as well as the parts of the project which I found easy as well as difficult.

I first began this assignment by implementing the code which I was certain that I was capable of implementing it fairly easy. Because of my experience with the worksheets throughout the entire semester as well as the mini assignment 2, which consisted of us creating a program where we had to create a system involving roads and settlements, I knew that being able to initialise all of the object within the GameEngine class was probably going to be the best place to start. Once I initialised the vast majority of the objects, I then focused on initialising the instance variables that I might need. Looking through the BeingInterface class that we were given before this assignment, it was clear to me that I would need to manipulate the locations of the bonks and zaps before I started implementing the code for it. I also noticed that the interface also used a getName() method, which suggested to me that the names of the bonks and zaps are also vitally important. With this knowledge beforehand, I initialised these variables within their respective classes. Finally, within the Grid World class I created an arraylist that was going to contain all of the information about every being that lives in Grid World. I felt that using an arraylist for this was the most effective choice, as the flexibility of the arraylist to be capable of being any size was going to be the most efficient for this type of programming. The different possibilities that could exist within Grid World could mean that the number of bonks that exist in Grid World have such a large range that using a regular array would definitely be inefficient with our memory.

When it came to dealing with a singular arraylist in a class where the array was initialised, I found that particularly easy due to my experiences with learning how to handle it within the Roads and Settlement assignment. However, one problem that I faced with this assignment was learning how to deal with an arraylist that was not initialised within the same class that I needed it for. However, after a quick search I found out that it was fairly easy to deal with as it simply required another get() method but for the arraylist.

However, one of the biggest challenges that I had to face was trying to figure out the logic when using multiple arraylists together within the same class. For example, within the Bonk and Zaps classes, I had trouble trying to make sure that the Bonks were breeding with the right type of bonks and the zaps were killing the right type of bonks respectively. Although the code that is being used within the program now is a better attempt than what I initially started with, I know that the code is not as robust and efficient as I would have liked it to be. I feel like I've learned a lot about the logic that is needed when it comes to using multiple arrays lists within the same class, and if I were to be given a second attempt at this assignment, I would continue to improve upon my Bonk and Zaps classes to ensure that when they breed and kill respectively, that it would consistently be on the right type of bonks. Another problem I had encountered was trying to make a GUI for the assignment. Due to my lack of knowledge with JFrame I was not capable of making a good start on the GUI and scrapped the idea early on. I hope that if I were to be given the chance to have a second attempt at this system, I would be capable of creating an attractive GUI for the user.

If I were to give myself a mark for this assignment, I would give myself a mark of roughly 55%. I feel that the Design and the Documentation for this assignment is fairly strong and that the

implementation of my program is good. I also feel like that I used inheritance to a great effect. However, I know that my code is not perfect and that at times the bonks and zaps reproduce and kill the wrong beings, which therefore means that the `act()` method still needs to be more robust before I consider it complete. Finally, losing 20% of the marks due to a lack of flair is also going to reduce my mark significantly. The lack of any diagrams will also count against my mark.