Bonks and Zaps

pit@aber.ac.uk
Pip Turner

May 5th, 2016

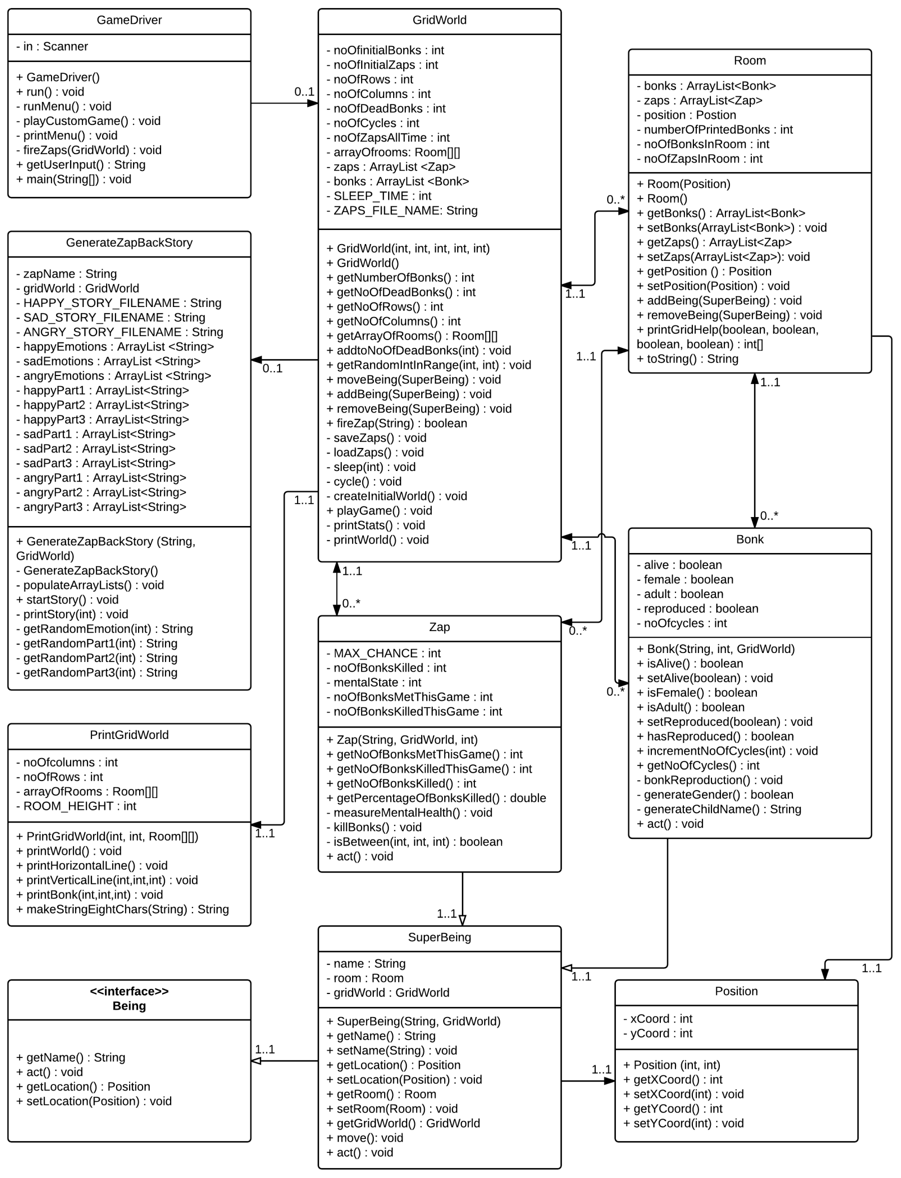
Contents

1 Introduction

Within this document there are three sections; design, testing and evaluation. Using each section I hope to present and explain my program to a high level, using a multitude of diagrams within the Design section, screenshots of testing methods, including a comprehensive description of my testing methods and an honest evaluation that confidently discusses the difficulties, flaws and blue sky thinking of my program.

2 Design

2.1 Class Diagram


GameDriver
- in : Scanner
+ GameDriver()
+ run() : void
- runMenu() : void
- playCustomGame() : void
- printMenu() : void
- fireZaps(GridWorld) : void
+ getUserInput() : String
+ main(String[]) : void
0..1
GridWorld
- noOfinitialBonks : int
- noOfInitialZaps : int
- noOfRows : int
- noOfColumns : int
- noOfDeadBonks : int
- noOfCycles : int
- noOfZapsAllTime : int
- arrayOfrooms: Room[][]
- zaps : ArrayList <Zap>
- bonks : ArrayList <Bonk>
- SLEEP_TIME : int
- ZAPS_FILE_NAME: String
+ GridWorld(int, int, int, int, int)
+ GridWorld()
+ getNumberOfBonks() : int
+ getNoOfDeadBonks() : int
+ getNoOfRows() : int
+ getNoOfColumns() : int
+ getArrayOfRooms() : Room[][]
+ addtoNoOfDeadBonks(int) : void
+ getRandomIntInRange(int, int) : void
+ moveBeing(SuperBeing) : void
+ addBeing(SuperBeing) : void
+ removeBeing(SuperBeing) : void
+ fireZap(String) : boolean
- saveZaps() : void
- loadZaps() : void
- sleep(int) : void
- cycle() : void
- createInitialWorld() : void
+ playGame() : void
- printStats() : void
- printWorld() : void
Room
- bonks : ArrayList<Bonk>
- zaps : ArrayList<Zap>
- position : Postion
- numberOfPrintedBonks : int
- noOfBonksInRoom : int
- noOfZapsInRoom : int
+ Room(Position)
+ Room()
+ getBonks() : ArrayList<Bonk>
+ setBonks(ArrayList<Bonk>) : void
+ getZaps() : ArrayList<Zap>
+ setZaps(ArrayList<Zap>): void
+ getPosition () : Position
+ setPosition(Position) : void
+ addBeing(SuperBeing) : void
+ removeBeing(SuperBeing) : void
+ printGridHelp(boolean, boolean, boolean, boolean) : int[]
+ toString() : String
0..*
1..1
1..1
1..1
GenerateZapBackStory
- zapName : String
- gridWorld : GridWorld
- HAPPY_STORY_FILENAME : String
- SAD_STORY_FILENAME : String
- ANGRY_STORY_FILENAME : String
- happyEmotions : ArrayList <String>
- sadEmotions : ArrayList <String>
- angryEmotions : ArrayList <String>
- happyPart1 : ArrayList<String>
- happyPart2 : ArrayList<String>
- happyPart3 : ArrayList<String>
- sadPart1 : ArrayList<String>
- sadPart2 : ArrayList<String>
- sadPart3 : ArrayList<String>
- angryPart1 : ArrayList<String>
- angryPart2 : ArrayList<String>
- angryPart3 : ArrayList<String>
+ GenerateZapBackStory (String, GridWorld)
- GenerateZapBackStory()
- populateArrayLists() : void
+ startStory() : void
- printStory(int) : void
- getRandomEmotion(int) : String
- getRandomPart1(int) : String
- getRandomPart2(int) : String
- getRandomPart3(int) : String
0..1
1..1
0..*
Bonk
- alive : boolean
- female : boolean
- adult : boolean
- reproduced : boolean
- noOfcycles : int
+ Bonk(String, int, GridWorld)
+ isAlive() : boolean
+ setAlive(boolean) : void
+ isFemale() : boolean
+ isAdult() : boolean
+ setReproduced(boolean) : void
+ hasReproduced() : boolean
+ incrementNoOfCycles(int) : void
+ getNoOfCycles() : int
- bonkReproduction() : void
- generateGender() : boolean
- generateChildName() : String
+ act() : void
1..1
1..1
0..*
0..*
0..*
Zap
- MAX_CHANCE : int
- noOfBonksKilled : int
- mentalState : int
- noOfBonksMetThisGame : int
- noOfBonksKilledThisGame : int
+ Zap(String, GridWorld, int)
+ getNoOfBonksMetThisGame() : int
+ getNoOfBonksKilledThisGame() : int
+ getNoOfBonksKilled() : int
+ getPercentageOfBonksKilled() : double
- measureMentalHealth() : void
- killBonks() : void
- isBetween(int, int, int) : boolean
+ act() : void
PrintGridWorld
- noOfcolumns : int
- noOfRows : int
- arrayOfRooms : Room[][]
- ROOM_HEIGHT : int
+ PrintGridWorld(int, int, Room[][])
+ printWorld() : void
+ printHorizontalLine() : void
+ printVerticalLine(int,int,int) : void
+ printBonk(int,int,int) : void
+ makeStringEightChars(String) : String
1..1
1..1
SuperBeing
- name : String
- room : Room
- gridWorld : GridWorld
+ SuperBeing(String, GridWorld)
+ getName() : String
+ setName(String) : void
+ getLocation() : Position
+ setLocation(Position) : void
+ getRoom() : Room
+ setRoom(Room) : void
+ getGridWorld() : GridWorld
+ move(): void
+ act() : void
1..1
1..1
Position
- xCoord : int
- yCoord : int
+ Position (int, int)
+ getXCoord() : int
+ setXCoord(int) : void
+ getYCoord() : int
+ setYCoord(int) : void
1..1
<<interface>>
Being
+ getName() : String
+ act() : void
+ getLocation() : Position
+ setLocation(Position) : void
1..1

2.2 Textual Description of Classes

2.2.1 Beings

Being Being.java is the interface provided by Chris Loftus. It contains a set of methods that Chris wanted every Being to contain. It is implemented within the SuperBeing class.

SuperBeing SuperBeing.java is a class that represents a generic Being. It holds variables and methods that all Beings need to use - e.g. a Name, a Room, a GridWorld, getters and setters for these variables and a move() method. Currently, only Bonks and Zaps inherit SuperBeing, but it would be very easy to add a third Being thanks to both SuperBeing and Being. Whilst I could have also included act() within SuperBeing as all Beings have an act() method, As each being's act method is so different, I preferred to have it within the Bonk/Zap itself. I also have an empty setLocation() method in here as I used position differently to what the Interface implied.

Bonk Bonk.java is one of two SuperBeings within my project, the other being Zap.java. A Bonk has a Name, a Room and a GridWorld (all of which are inherited from SuperBeing), a Number of Cycles and a multitude of booleans: alive, adult, reproduced. All of these variables have their own getters and setters. Bonk also contains methods that help it act, such as generateGender() and generateChildName(). Bonk inherits SuperBeing and is used by GridWorld and Room.

Zap Zap.java is the second of the two SuperBeings within my project. As well as the inherited variables, it has noOfBonksKilled, noOfBonksMetThisGame, noOfBonksKilledThisGame and mentalState. As well as setters and getters for these variables, the Zaps have getPercentageOfBonksKilled(), measureMentalHealth(), killBonks() and act(). Zaps have a guilty conscience, the more they kill, the less they want to. Zap inherits SuperBeing and is used by GridWorld and Room.

2.2.2 GridWorld

GameDriver GameDriver.java is the first class called when BonksAndZaps is run. The only instance variable it contains is a scanner called in. Its purpose is to control the GridWorld and handle user input. It prints menus and lets the user decide what to choose. This includes giving the user the option to play a standard game or custom sized game and whether to fire a zap or not. It can have a gridWorld attached to it, depending on whether the user decides to quit or now.

GridWorld GridWorld is the biggest class in my project. It contains all the instance variables for the gridworld, getters and setters for them, and then various methods which set up the gridWorld, handle the movement of beings within the gridWorld, add and remove beings from the gridWorld, handles the saving and loading of zaps and runs the general cycle of the gridWorld. It is arguably the Cornerstone of the project, without it, nothing would have purpose. GridWorld can have a GenerateZapBackStory, has one PrintGridWorld, can have many bonks and zaps and can have many Rooms. All of these also contain one GridWorld.

Room Room.java is used for holding all the information about a room in GridWorld. It is what GridWorld is made of. As well as instance variables for an array list of bonks, an array list of zaps and the room's position (all with getters and setters) there are also methods to add and remove beings, and a method that helps the PrintGridWorld class by storing useful variables. Room has one GridWorld, zero or many Zaps, zero or many Bonks and one Position. In turn, all of those objects apart from GridWorld have one room. GridWorld has zero or many.

Position Position.java simply contains an X and a Y integer, with appropriate setters and getters. It's simply used for holding a location which can then be used to get the room.

2.2.3 Useful Classes

CannotActException I wasn't sure what to do with CannotActException. As it was in the given interface, I made a class, but I left it unpopulated.

GenerateZapBackStory GenerateZapBackStory.java is a class that prints out a randomised back story for a Zap, once it has been fired. There are three different types of story that a Zap can have - Happy, Sad or Angry. Each of these have their own text file containing a list of appropriate emotions, a list of first parts, second parts and third parts. It randomly chooses a type of story, before randomising the emotion and one bit from each part, after which it prints it all. It contains one gridWorld, which, in turn, contains one GenerateZapBackStory.

PrintGridWorld PrintGridWorld.java handles printing the world into the console. It gets the array of rooms and cycles through them, printing appropriate information. If there is a zap in the room, it prints it first. Otherwise, it prints bonks out in order until there is no space left in the cell, upon which it displays +x, where x is the remaining alive Beings in the room. PrintGridWorld contains one GridWorld, which, in turn, contains one PrintGridWorld

2.3 Pseudo-code

2.3.1 Move Being

Algorithm 1 Move Being

```
if Number of Rows and Number of columns != 1 then
  GET Being's Location
  if Being is in the 1st row then
    New Row = Get either 0 or 1
  else if Being is in the last row then
    New Row = Get either Max Row-1 or Max Row
  else
    New Row = Get either currentRow +0 or +1 or -1
  end if
  if Being is in the 1st column then
    New Column = Get either 0 or 1
  else if Being is in the last Column then
    New Column = Get either Max Column-1 or Max Column
  else
    New Column = Get either currentColumn +0 or +1 or -1
  end if
  if Being is a bonk then
    Remove Bonk from Current Room
    Put Bonk in new room, using New Row and New Column as the index
  else if Being is a Zap then
    Remove Zap from Current Room
    Put Zap in new room, using New Row and New Column as the index
  end if
end if
```

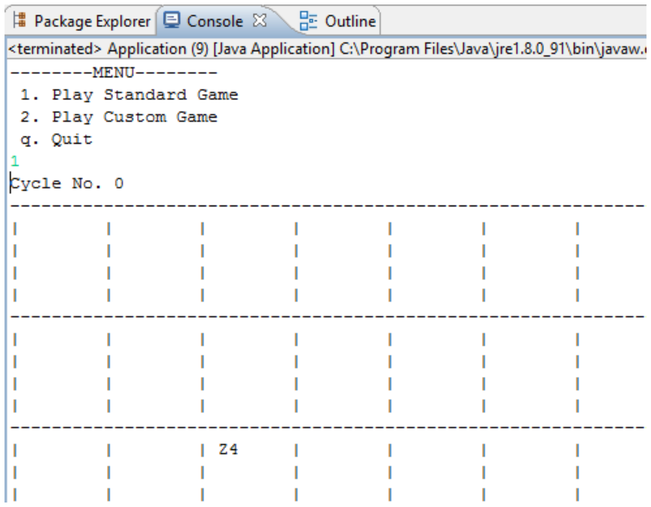
2.3.2 Bonk Reproduction

Algorithm 2 Bonk Reproduction

```
for all Bonks in the same room do
    if Bonk is of the opposite gender AND has not reproduced AND is not a
    baby AND is alive then
        Create new Bonk Name
        Add Baby Bonk to current room
        Add Baby Bonk to GridWorld array
        Set partner Bonk to reproduced = true
        Set this to reproduced = true
    end if
end for
```

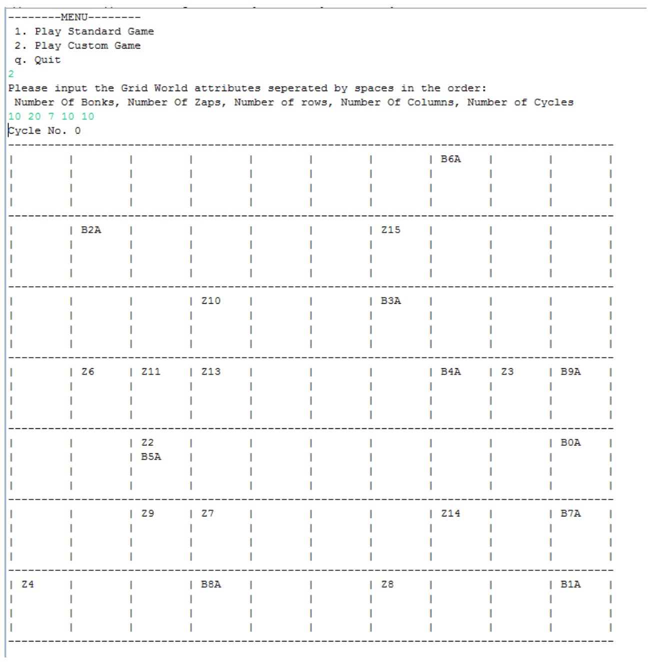
3 Testing

I tested by using correct values, then extreme values, then incorrect values. My testing worked and showed that my code is flexible and can cope with a high variation of values. However, it struggles with large numbers, taking a while to handle them.



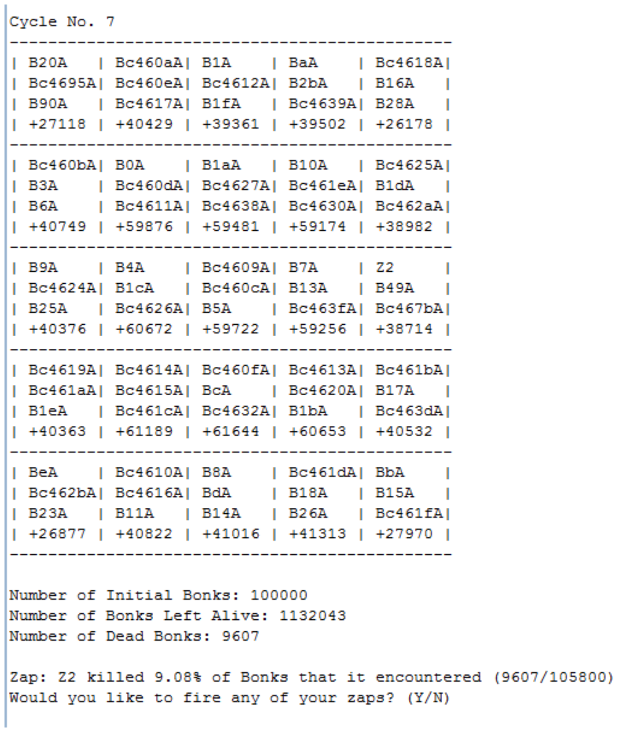
```
<terminated> Application (9) [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
1
Cycle No. 0
-------------------------------------------------------------
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
-------------------------------------------------------------
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
-------------------------------------------------------------
|        |        | Z4     |        |        |        |
|        |        |        |        |        |        |
|        |        |        |        |        |        |
```

Figure 1: Example of User Input Working. The User input "1" to select option 1, and a standard game began



```
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
2
Please input the Grid World attributes seperated by spaces in the order:
 Number Of Bonks, Number Of Zaps, Number of rows, Number Of Columns, Number of Cycles
10 20 7 10 10
Cycle No. 0
-------------------------------------------------------------------------------------------
|        |        |        |        |        |        |        | B6A    |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
-------------------------------------------------------------------------------------------
|        | B2A    |        |        |        |        | Z15    |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
-------------------------------------------------------------------------------------------
|        |        |        | Z10    |        |        | B3A    |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
-------------------------------------------------------------------------------------------
|        | Z6     | Z11    | Z13    |        |        |        | B4A    | Z3     | B9A    |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
-------------------------------------------------------------------------------------------
|        |        | Z2     |        |        |        |        |        |        | B0A    |
|        |        | B5A    |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
-------------------------------------------------------------------------------------------
|        |        | Z9     | Z7     |        |        |        | Z14    |        | B7A    |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
-------------------------------------------------------------------------------------------
| Z4     |        |        | B8A    |        |        | Z8     |        |        | B1A    |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
|        |        |        |        |        |        |        |        |        |        |
-------------------------------------------------------------------------------------------
```

Figure 2: Example of a Custom GridWorld working, with the correct number of Bonks, Zaps, Rows and Columns

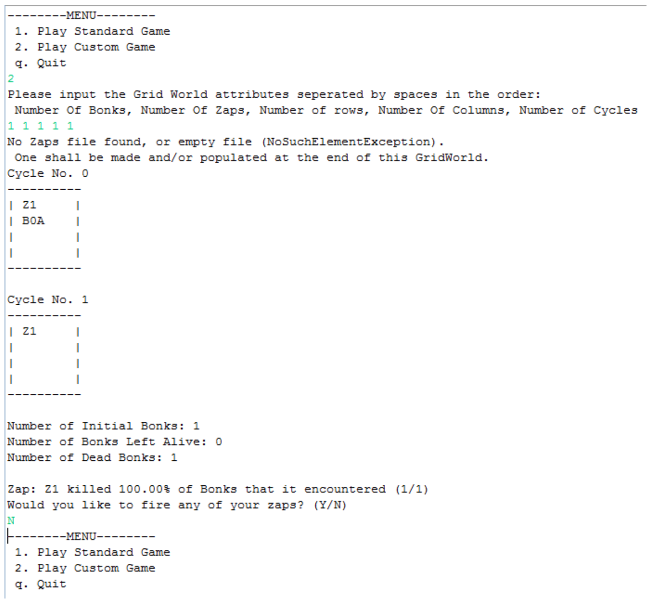


```
Cycle No. 7
---------------------------------------------
| B20A   | Bc460aA| B1A    | BaA    | Bc4618A|
| Bc4695A| Bc460eA| Bc4612A| B2bA   | B16A   |
| B90A   | Bc4617A| B1fA   | Bc4639A| B28A   |
| +27118 | +40429 | +39361 | +39502 | +26178 |
---------------------------------------------
| Bc460bA| B0A    | B1aA   | B10A   | Bc4625A|
| B3A    | Bc460dA| Bc4627A| Bc461eA| B1dA   |
| B6A    | Bc4611A| Bc4638A| Bc4630A| Bc462aA|
| +40749 | +59876 | +59481 | +59174 | +38982 |
---------------------------------------------
| B9A    | B4A    | Bc4609A| B7A    | Z2     |
| Bc4624A| B1cA   | Bc460cA| B13A   | B49A   |
| B25A   | Bc4626A| B5A    | Bc463fA| Bc467bA|
| +40376 | +60672 | +59722 | +59256 | +38714 |
---------------------------------------------
| Bc4619A| Bc4614A| Bc460fA| Bc4613A| Bc461bA|
| Bc461aA| Bc4615A| BcA    | Bc4620A| B17A   |
| B1eA   | Bc461cA| Bc4632A| B1bA   | Bc463dA|
| +40363 | +61189 | +61644 | +60653 | +40532 |
---------------------------------------------
| BeA    | Bc4610A| B8A    | Bc461dA| BbA    |
| Bc462bA| Bc4616A| BdA    | B18A   | B15A   |
| B23A   | B11A   | B14A   | B26A   | Bc461fA|
| +26877 | +40822 | +41016 | +41313 | +27970 |
---------------------------------------------

Number of Initial Bonks: 100000
Number of Bonks Left Alive: 1132043
Number of Dead Bonks: 9607

Zap: Z2 killed 9.08% of Bonks that it encountered (9607/105800)
Would you like to fire any of your zaps? (Y/N)
```

Figure 3: Example of a max value for Bonks. It was given 100000 Bonks and 1 zap as a starting point, with 7 cycles. It progressed slower and slower until eventually reaching Cycle 6 not updating for over 10 Minutes



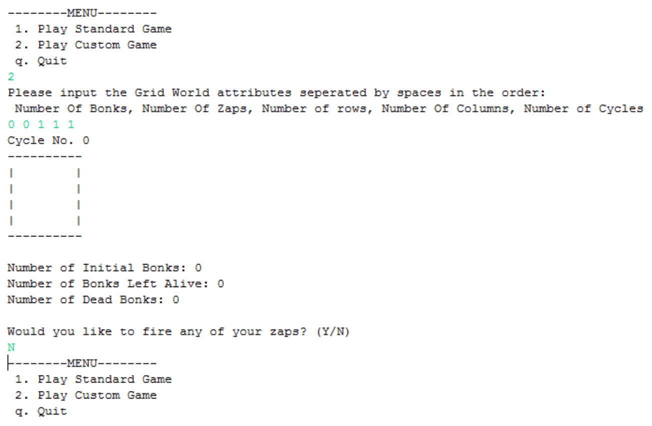
```
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
2
Please input the Grid World attributes seperated by spaces in the order:
 Number Of Bonks, Number Of Zaps, Number of rows, Number Of Columns, Number of Cycles
1 1 1 1 1
No Zaps file found, or empty file (NoSuchElementException).
 One shall be made and/or populated at the end of this GridWorld.
Cycle No. 0
----------
| Z1      |
| B0A     |
|         |
|         |
----------

Cycle No. 1
----------
| Z1      |
|         |
|         |
|         |
----------

Number of Initial Bonks: 1
Number of Bonks Left Alive: 0
Number of Dead Bonks: 1

Zap: Z1 killed 100.00% of Bonks that it encountered (1/1)
Would you like to fire any of your zaps? (Y/N)
N
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
```

Figure 4: Example of the smallest Grid possible.

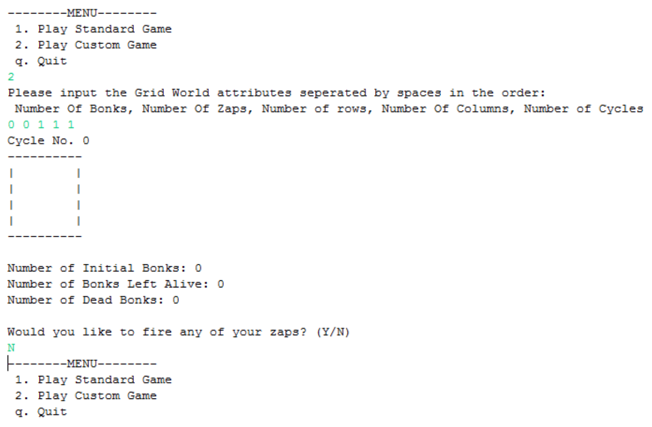


```
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
2
Please input the Grid World attributes seperated by spaces in the order:
 Number Of Bonks, Number Of Zaps, Number of rows, Number Of Columns, Number of Cycles
0 0 1 1 1
Cycle No. 0
----------
|        |
|        |
|        |
|        |
----------

Number of Initial Bonks: 0
Number of Bonks Left Alive: 0
Number of Dead Bonks: 0

Would you like to fire any of your zaps? (Y/N)
N
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
```

Figure 5: Example of no Bonks or Zaps

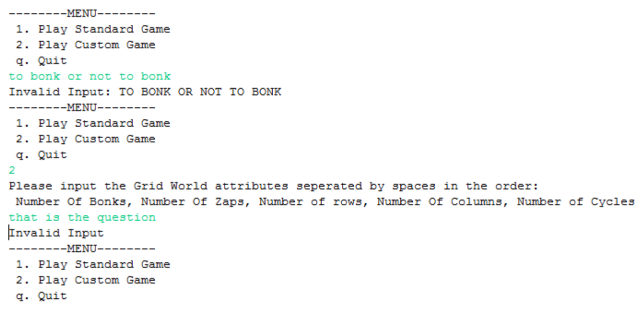


```
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
2
Please input the Grid World attributes seperated by spaces in the order:
 Number Of Bonks, Number Of Zaps, Number of rows, Number Of Columns, Number of Cycles
0 0 1 1 1
Cycle No. 0
----------
|        |
|        |
|        |
|        |
----------

Number of Initial Bonks: 0
Number of Bonks Left Alive: 0
Number of Dead Bonks: 0

Would you like to fire any of your zaps? (Y/N)
N
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
```

Figure 6: Example of no Bonks or Zaps



```
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
to bonk or not to bonk
Invalid Input: TO BONK OR NOT TO BONK
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
2
Please input the Grid World attributes seperated by spaces in the order:
 Number Of Bonks, Number Of Zaps, Number of rows, Number Of Columns, Number of Cycles
that is the question
Invalid Input
--------MENU--------
 1. Play Standard Game
 2. Play Custom Game
 q. Quit
```

Figure 7: Example of invalid input

4 Evaluation

The first thing I did when starting this assignment was creating a Room class and giving it a toString. I then made a two dimensional array of Rooms and looped through them, displaying their index and toString. This philosophy extended to the rest of the code. I knew what I needed to do, so I started it and kept following the logical steps until I reached a finished class. For Example, after making the two dimensional array List "rooms", I wanted to populate it with Bonks and Zaps. This then meant first creating a SuperBeing class, which lead to the Bonk class and the Zap class, which then led to randomly placing them.

The hardest part was starting and planning out how each class would link together. Once the foundation had been put in place, e.g. Bonk, Zap, Room, GridWorld classes, the rest all followed with not too much of a hassle. I often found, especially for working out the rules for moving, that using a pen and paper worked best. Visualising the problem helped to simplify, abstract and solve it. I also learnt a lot more about inheritance, finding the instanceof attribute was particularly useful. Another thing I struggled with was the consistent Zap naming. I wanted to name the Zaps so that they incremented with all zaps over all time, instead of just in the current session. However, this meant that every time I needed to create zaps, I had to load in all zaps, assign the relevant number rooms and leave the others in the memory, before saving them. This caused a lot of problems when calling zaps.size().

Whilst my GridWorld does work and displays a grid, I would like implement a GUI system into my GridWorld. I experimented with it but decided that within the timeframe I had left, that it would not be achievable. I would also like to expand the Zap's "mental-health", balance it more and add more features, such as not killing a Bonk if it had killed a lot from its family. On top of all this, I have some statistics at the end of my GridWorld and I would also like to improve and expand upon that. Another option would be to pair both JavaFX and these statistics together, to create easy to look at charts, that reflected a family tree, or bonk growth, or zaps' killing.

I also realise that my testing section in this writeup is lacking. I would greatly like to expand on it, using white box methods.

Overall, I would give myself 75%. My GridWorld moves Bonks and Zaps correctly and randomly, the zaps kill and the bonks reproduce and all of this is displayed on a readable grid in the console. However, I do think that with more planning I could have made my Program's methods and classes neater and more logical, using more inheritance to compact and neaten methods. I also accept that my flair, whilst fairly substantial, was not particularly technically complicated, when compared to a GUI using JavaFX. I also would like to use a more elegant solution within my GenerateZapBackStory class as it seems very inefficient at the moment. Another example of inefficiency, both processing and memory is that I have to load every zap into the code every time I make a new game, not just when needed. This could be improved.