# Part 2 Java Coursework – Artificial life simulation

Oliver Jones

24005044

Abstract:

The main aim of the project was to design and create a 2D interaction simulation between different entities. The way this was done was by assigning algorithmic behaviours to each class without any user input and watching it run. The way this was completed was by using a variety of standard libraries for the backend of the code and then using JavaFX for the front end GUI. This practical was very useful as it provides a good grasp of Java as a language and how it can be used.

Introduction:

The aim of this project was to create an artificial life simulator in a 2D world with various entity types. These entities were to have different traits and behave differently to each other. Stationary entities are in the form of obstacles, teleports and food. Obstacles are impassable meaning that any moving entity is unable to go through them and has to navigate around them. Teleport entities are also stationary; these allow the Mouse entities to teleport randomly to a new location on the map when a mouse is within range. Finally, for the stationary entities is Food. Food has an energy value, which when it is eaten by either a mouse or a cat the food entity is destroyed and the entity that destroyed the food gains the energy that the food had.

Requirements of the project:

* Be a user defined space
* The world must be generated from a configuration file that is saved
* The world must load the previously loaded file on start up
* GUI must but built from JavaFX and not use any API’s
* The code should use abstract classes
* The program has to use animation of some form

Description of OOP design:

Within this application there are 3 main classes I use with one of those classes being abstract enabling me to create children of that class (AnEntity).

The way that this project was undertaken was in a RAD (Rapid Application Development) style of development. For the first part of the project a prototype of the artificial life simulation had to be made. This was able to run in console and thus formed the backbone of the finished product. With this being rapid development style there was a lot of coding, testing, bug fixing and repeating this cycle over and over in a very short period of time.

Based upon the requirements that we set out at the very beginning of the project it was clear for me that the main inheritance and therefore abstracting would be able to take place upon my AnEntity class. This allowed me to use the same base behaviour for each of the animals (Cat and Mouse) but then have animal specific behaviours. The member variable in AnEntity could be accessed from the child classes. On creation of an entity it is assign a random position in the world by using a random function and check to ensure that the space that it is trying to be added to is free.

AnEntity also contains functions that can affect itself in terms of getters and setters.

AnEntity abstracts to ACat, AMouse, AFood, AnObstacle, APoison and ATeleport. This is a great example of the use of the construct polymorphism where a single method has many different applications and accessor.

AnObstacle, APoison and ATeleport are all constructor only classes as they do not have any behaviour or any movement. They are all created with 0 energy apart from APoison which has a value of -10000. This is to ensure that if ACat or AMouse “eat” it they will be destroyed.

ACat has access to the LookFor function in the parent class AnEntity, however it its class specific update function one of the traits ACat has is to be able to search for both AMouse and AFood. Once it has found either of these it will make its way towards its target and once it has reached there it will then consume the target if they collide and ACat will gain the energy of the target that was eaten.

Similarly, for AMouse it too can consume AFood entities. However, AMouse has also got access to ATeleport. This means when it is within a 3 block radius the mouse will be automatically moved to a random location meaning it has better chances of “survival”.

Below you will find my class listings and their respective functions.

Class: GUI

* **public void start(Stage PrimaryStage)** creates the stage and runs all subsequent functions that prepare the Javafx environment to be ready to be used.
* **public void ShowEntity(Image Image, int x, int y)**
* **public void CreateWorld()** creates the world in which the Entities are placed.
* **public void DrawWorld()** Each time this function is called it draws the current state of the world onto the canvas. It in turn calls other functions which add the correct images to the Entities, these images are then also shown upon the canvas in a grid like fashion.
* **public void LoadFile(String LoadFile)**
* **public void NewConfig()**
* **public void SaveFile()**
* **public void SaveFileAs(String SaveFile)**
* **public void ConfigureWorld**
* **public static void main(String[] args)**

Class: AWorld

* **public AWorld(int wx, int wy, int maxEntitys)** this is the constructor for the AWorld class.
* **public int Getx()** used to retun the Worldx value.
* **public int Gety()** used to retun the Worldy value.
* **public void AddEntity(String name, int Energy)** used to add objects of type AnEntity to the world.
* **public int Search(String species, int x, int y )** generic search function that takes children of AnEntity and in turn searches for them in the Entity arraylist. Returns position in Entity arraylist if found, returns -1 if not.
* **public boolean isSpaceFree(int ex, int ey)** generic space checking function. Checks to see whether the space at the provided parameters is free or not. Returns True or False.
* **public void Simulate()** function that calls the generic update function from AnEntity.
* **public void Stats()** functions shows how many of each life form there are in the world at the current time. This includes Food, Obstacles, Poison, Cats and Mice.
* **public void ShowEntitys(GUI GUI)**

Class: AnEntity

* **public AnEntity(String specie, int AEx, int AEy, int EntityEnergy, AWorld TheWorld)** this is the default constructor for the AnEntity class.
* **public String Getspecies()** used to return the species string of the entity.
* **public int Getx()** used to return the x value of the entity.
* **public void Setx(int x1)** used to set the x value of the entity.
* **public int Gety()** used to return the y value of the entity.
* **public void Sety(int y1)** used to set the y value of the entity.
* **public boolean isHere(int Sx, int Sy)** used to check whether there is an entity at the specified x and y value passed as parameters.
* **public void DisplayEntity(GUI GUI, String Temp)**
* **public void SetImage(Image Image)**
* **public void update(int z)**
* **public boolean LookFor(String species, Direction D, int range)** this is a generic function that allows entities to search the world for an entity of the desired type (for example: Cats can search for mice and cheese, mice can search for teleporters and cheese). The other parameters that are passed are Direction, which is an enumeration value used to look North, East, South or West, and the range that the entity is able to look. If a target entity is within the range of entity running this function then this entity will move towards the target entity and lose an energy point as it has “moved”.

Class: ACat extends AnEntity

* **public ACat(String specie, int AEx, int AEy, int EntityEnergy, AWorld TheWorld)**  This is the super constructor for the Cat child class
* **public void update(int z)** This is the overridden function that is called when update(); is called from the parent class

Class: AMouse extends AnEntity

* **public AMouse(String specie, int AEx, int AEy, int EntityEnergy, AWorld TheWorld)** This is the super constructor for the Mouse child class
* **public void update(int z)** This is the overridden function that is called when update(); is called from the parent class

Class: AnObstacle extends AnEntity

* **public AnObstacle(String specie, int AEx, int AEy, int EntityEnergy, AWorld TheWorld)** This is the super constructor for the Obstacle child class

Class: AFood extends AnEntity

* **public AFood(String specie, int AEx, int AEy, int EntityEnergy, AWorld TheWorld)** This is the super constructor for the Food child class

Class: APoison extends AnEntity

* **public APoison(String specie, int AEx, int AEy, int EntityEnergy, AWorld TheWorld)** This is the super constructor for the Poison child class

Class ATeleport extend AnEntity

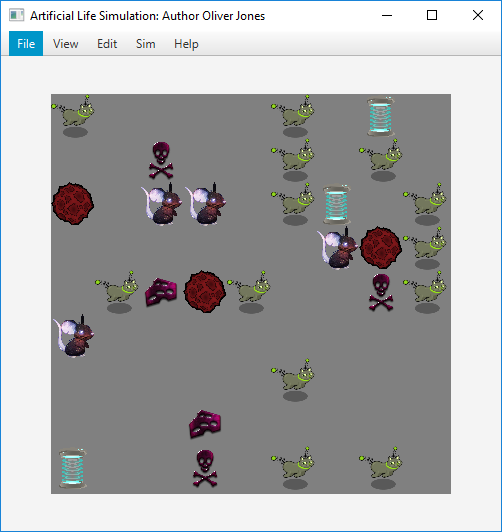
* **public ATeleport(String specie, int AEx, int AEy, int EntityEnergy, AWorld TheWorld)** This is the super constructor for the Teleport child class

UML Class Diagram



User Manual:

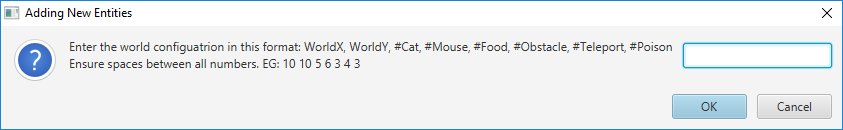
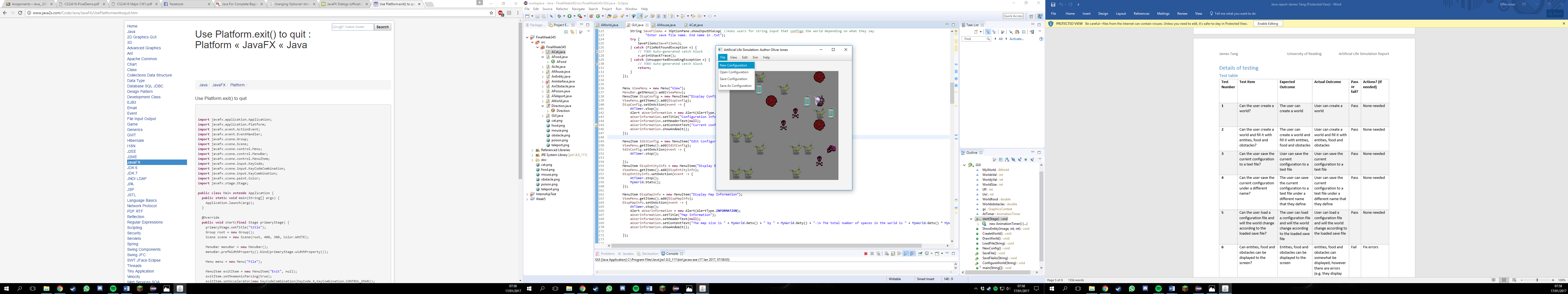
When starting the program the user is immediately brought into a simulation (this is using the previous configuration)



This pops up and starts cycling through with the cats chasing mice and eating cheese and mice eating cheese. Either after or during the initial simulation the user has a few options available to them.

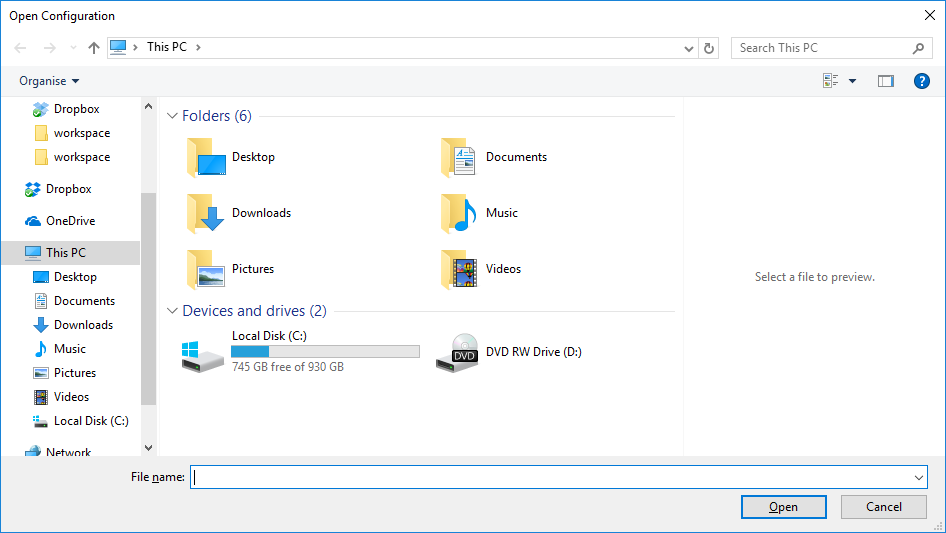
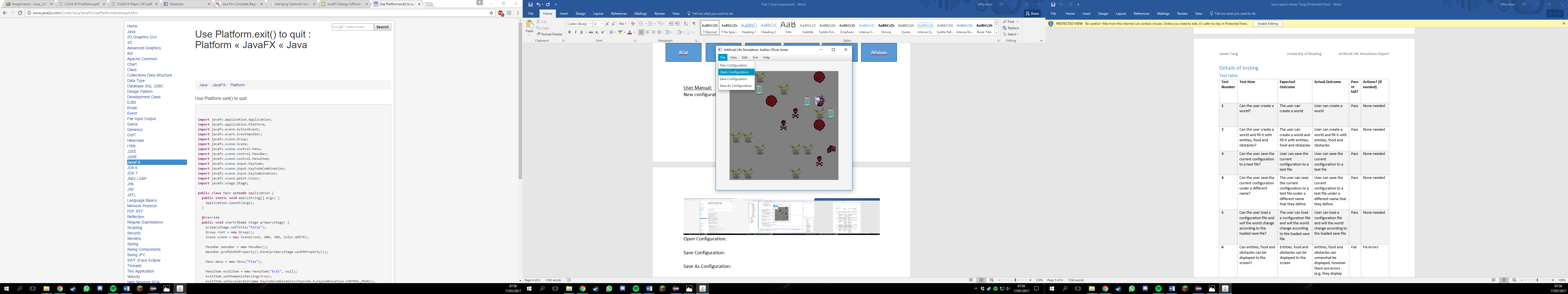
New configuration:

File -> New Configuration -> Adding New Entities



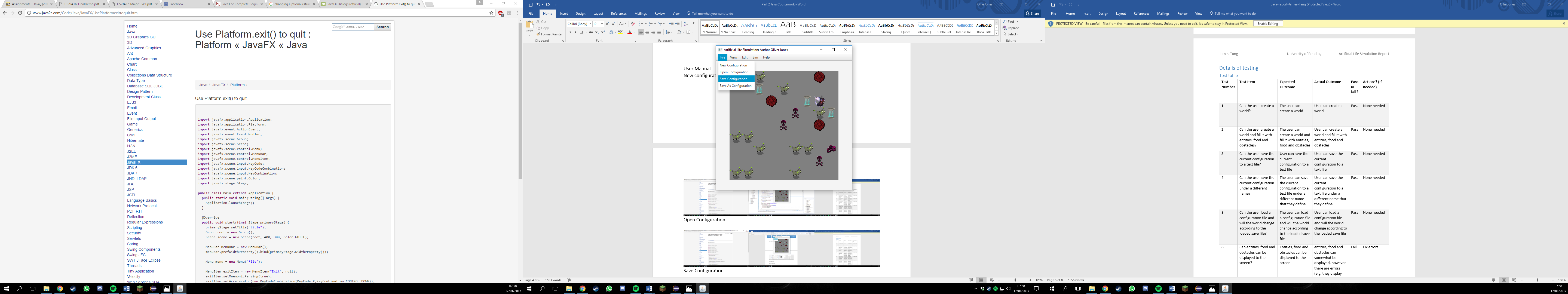
Upoin clicking on new configuration the user it met with an input box. This prompts the user to input space separated values and gives indication to what each number corresponds to.

Open Configuration:



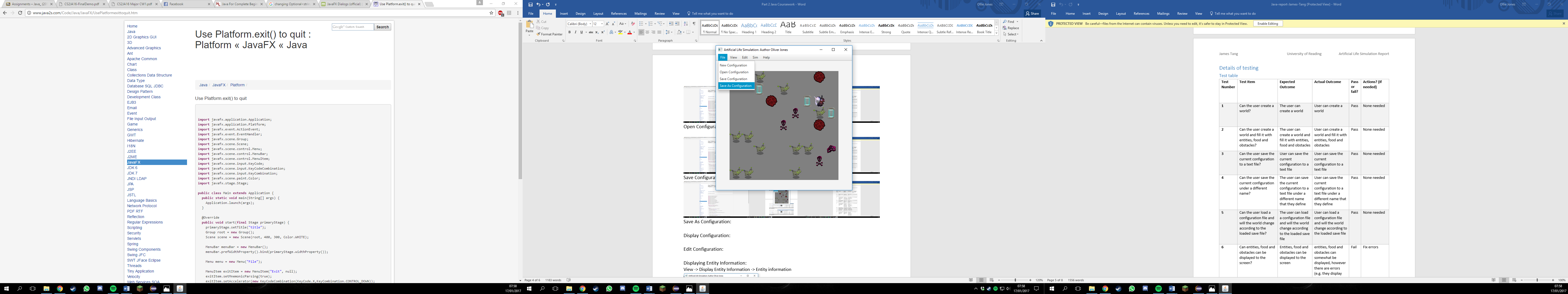
Opening a configuration file lets the user browse to a selected folder and then open the file. The valued list will then be pulled into the application and the world created

Save Configuration:



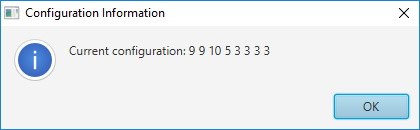
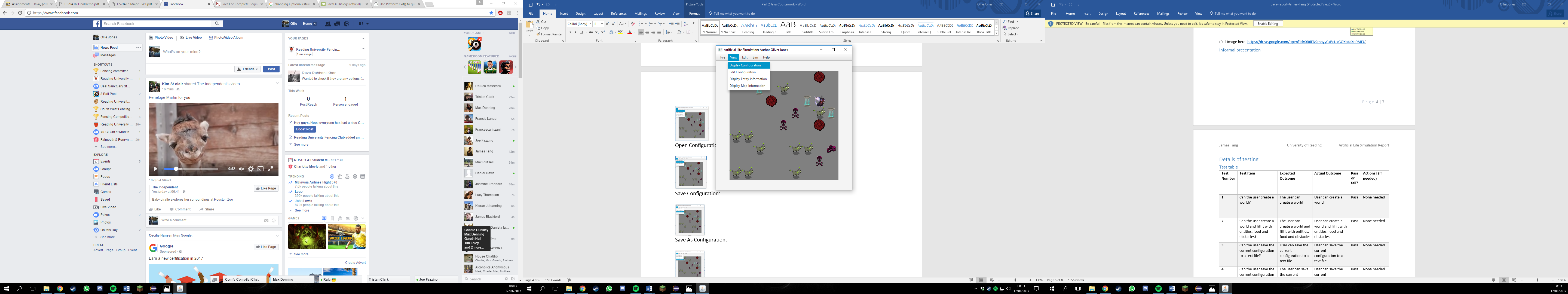
When saving a file the current configuration is saved and overwrites the old file currently stored within the SaveConfiguration.txt.

Save As Configuration:



When saving as the user has their own choice of file name as long as it ends with .txt. This is then saved and can be loaded whenever the user wishes.

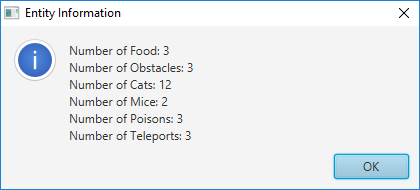
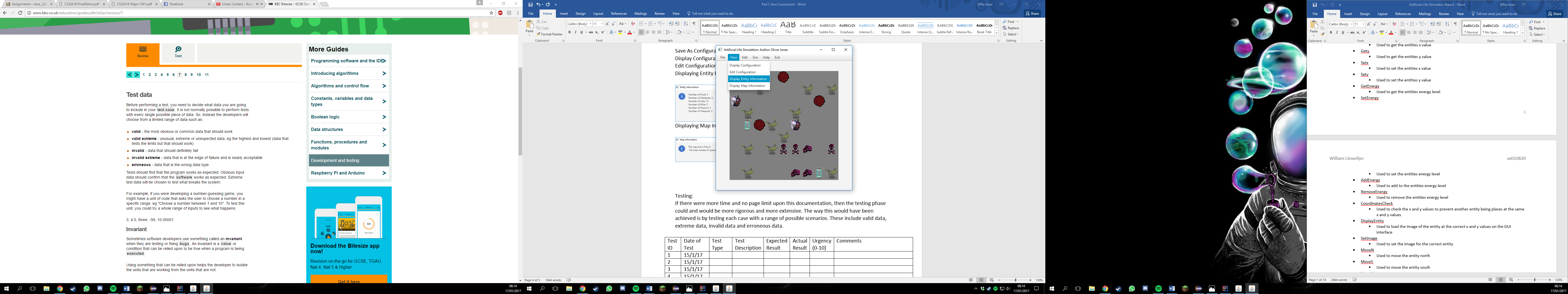
Display Configuration:



This outputs the current world configuration string. This is exactly what is saved into the file which the user is able to check to see if any errors have occurred.

Displaying Entity Information:

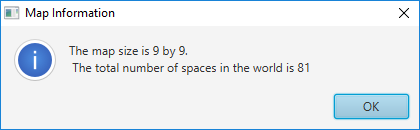
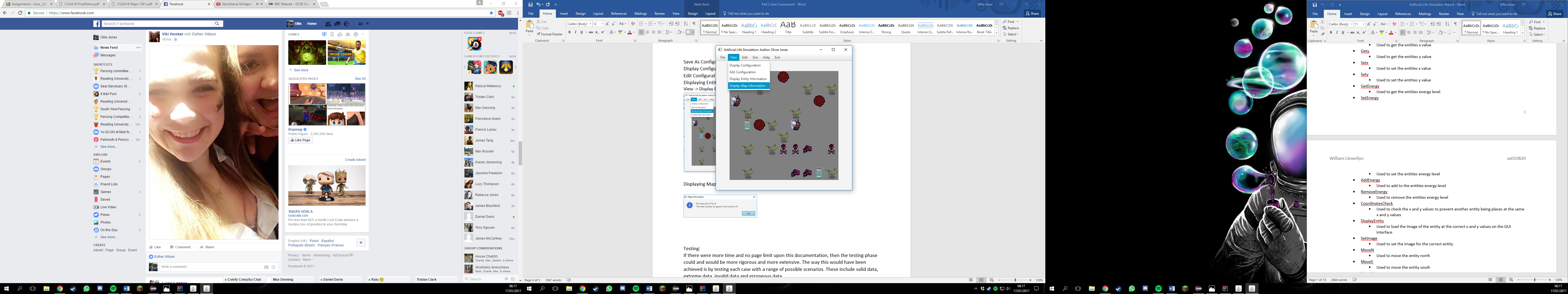
View -> Display Entity Information -> Entity information



This screen shows the user exactly how many of each type of entity are in the world at the time that the user stops the simulation. Upon running again and re-clicking the Display entity the numbers will change due to entities being destroyed and created.

Displaying Map Information:

View -> Display Map Information - > Map Information



This shows the map information. Tell you the size of the world and how many entities in total that you are able to have in the world.

Testing:

If there were more time and no page limit upon this documentation, then the testing phase could and would be more rigorous and more extensive. The way this would have been achieved is by testing each case with a range of possible scenarios. These include valid data, extreme data, invalid data and erroneous data.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID | Date of Test | Test Description | Expected Result | Actual Result | Urgency  (0-10) | Comments |
| 1 | 10/1/17 | Loading the last configuration | Loads last used configuration file | Loads last used configuration file | 5 | Pass |
| 2 | 10/1/17 | User can run simulation | Simulation runs | Simulation ran | 7 | Pass |
| 3 | 10/1/17 | User can pause/stop simulation | Simulation stopped | Simulation stopped | 2 | Pass |
| 4 | 10/1/17 | User can create a new configuration | New configuration was input | New configuration was loaded | 2 | Pass |
| 5 | 10/1/17 | User can open a configuration | Opened configuration | Had a path file error | 2 | Fail – changed around a variable and cast it to path, this test was then successful |
| 6 | 10/1/17 | User can save the current configuration | Current configuration was saved under default name | Current configuration was saved to the save file | 0 | Pass |
| 7 | 10/1/17 | User can save as a configuration | Current configuration was saved under custom name | File was saved | 0 | Pass |
| 8 | 10/1/17 | User can view details about the simulation | Viewed details about the |  | 0 | Pass |
| 9 | 10/1/17 | User can view details about the world | Viewed details about the word |  |  |  |
| 10 | 10/1/17 |  |  |  |  |  |

Discussion:

A thing I noticed with ACat being able to eat both AMouse and AFood is that AMouse would always nearly died out. Then ACat would rapidly increase due to lots of AFood spawning. Then once the number of ACat were too high and they were not consuming enough food the number depleted rapidly, which in turn lead to a vast increase of AFood, which in turn lead to ACat repopulating and this cycle continued. I found this interesting as this happens in the real world and

When reaching the closing stages of the project I realised some aspects could have been created more tidily and compact. This would have helped optimise the code and probably ran slightly more efficiently.

If there was more time available to develop the code I would have implemented other predators or food with different behaviours, zones on the map that only certain entities are able to go to. Little things that would enhance the artificial life aspect of this simulation.

Conclusion:

I found that this project was interesting and challenging at different points. It was interesting to see the development between the weeks and see how they all linked into each other, however when it came to changing from the console based application to a graphical one it was quite a challenge as almost all of the console interface was instantly redundant and having to start afresh was a little daunting. Once the initial daunt was over and I had gotten my head around the JavaFX API then the rest of the work was straight forward, although I did hit a wall when trying to work out my formula to get different sized grids to draw onto the same size canvas with different sized images. In the end if was all fine and supports different sized grids past 10\*10 and the entities are all graphically displayed also.