CS2 JA16 Java Coursework 1

Abstract

This assignment was to design and implement a virtual life simulator in Java. The simulator had to handle several different types of entities as well as outputting their activity to the user graphically. There were several different iterations of the simulator before the final project. Firstly, the method of display was through the console and the program only handled a couple of basic entity behaviours. Next, reading and saving from a text file was implemented, followed by adding more control features such as pausing and resetting the environment. Finally, the GUI was applied. The results were a fully functioning Java executable which simulated several basic lifeforms and inanimate objects. There was also a successful set of controls which modify the parameters of the world.

Introduction

This project was begun by designing what the input the program would need to handle as well as the desired output. The project was then split into several sections to help break it down into manageable steps. The main two steps were the logic and the graphics. First the logic must be implemented and tested successfully before the graphics overlay will be designed over it. The logic layer will include the behaviours and movements of the entities and the world they live in. For ease of testing, the logic layer will output the world to a console. The additional features and a GUI will then be created to create the final project.

Design

Classes

There are several classes that were designed for this application:

**GUIentity**: This abstract class will contain all the necessary properties that all entity’s must have. These properties will be; a species, an image which represents the entity, which world the entity will be in, a variable of which type of other entity that the entity pursues to eat, a compass direction, and a position of where the entity is in the world. All entities also will have an energy count. The other types of entity will all inherit from this class, as it has the properties they all must have. The “Species” field will be the unique identifier for each type of entity.

The responsibilities of this class will be:

* To provide all essential properties for every type of entity whom will inherit from this class.
* To provide default values for these properties where the inheriting class does not define their own value.

**AMagikarp**: This class is for the Magikarp entity (which will be the prey in the simulation). This class will inherit from the GUIentity class. They will hunt and try to find food to increase their own energy. The class does not have any variables except those from the GUIEntity class but has its own energy level, image and of course having a unique identifier. Its behaviour is that it will navigate around the world randomly until it finds a food within 5 spaces horizontally or vertically.

The Magikarp Class will be responsible for:

* Managing the way that Magikarp entities look and act in the world. The class handles what they hunt and how they move if they have no food in their line of sight.
* Eating any neighbouring Pokefood and adding the consumed food to itself

**ASharpedo:** This class is for the Sharpedo entity (which will be the predator in the simulation). It will be one of the child classes from GUIentity. These entities will hunt the Magikarp and will attempt to follow them around the world. When they are in a one block radius of a Magikarp entity they will eat the Magikarp (remove it from the world) and then move to the coordinates where the Magikarp was.

The Sharpedo Class will be responsible for:

* Managing how the Sharpedo entities look and act in the world. They will look like sharks and hunt any Magikarp’s in their line of sight, which is five squares vertically and horizontally.
* Eating any neighbouring Magikarp and add the consumed energy to itself.

**PokeFood:** This class will represent the food in the world. It will be immobile and does not consume any other type of entity. It will not regenerate but has energy which will be added to Magikarp when it is eaten.

The PokeFood Class will be responsible for:

* Managing how the PokeFood entities look and act in the world. They are visually represented by a blue berry. They don’t move and their position is set when they are spawned
* Maintaining the same amount of energy as they are not moving

**Obstacle:** This class will be for the obstacles which will block certain cells so that no other entities can pass through that cell. They will share the trait will the food, and they do not move. However, unlike the food, they cannot be consumed by any type of entity.

The Obstacle class will be responsible for:

* Managing the way that the Obstacle entity will look and interact with the world. They will not move and will prevent any entity from passing through them.
* Not being eaten by any other entity.

**GUIWorld:** This class needs to be created so that the entities can have a world to interact and move about in. It will have several properties such as its size, an integer which points to a current selected entity and an ArrayList, which will store all the entities in that world. It also has functions to instantiate a new entity which will be stored in its ArrayList. The GUIWorld will also have some other functions which will handle running the simulation. For example, it will have the code for doing one simulation cycle (this will probably include moving all the entities one space around the world).

The GUIWorld class will be responsible for:

* Allowing the entities to move and interact with each other.
* Providing a function which moves the entities around the world based on their behaviour.
* Providing a function which checks the bounds of the world and prevents an entity moving out of it.

**GUIinterface:** This class needs to be created to provide the interface and control features. These are necessary for the user to be able to manipulate the world effectively and exactly as they want. This class will handle all the graphical user interfacing as well as import and export the configuration files.

The GUIinterface class will be responsible for:

* Handling the user’s commands and changing the world and the entities so that it matches the user’s requirements.
* Displaying the world and therefore the entities as well.
* Opening a file and changing the world to match it.
* Saving the current world into a current file or creating a new file with the configuration.
* Creating a configuration file with new perimeters that the user shall input.

UML Diagram

**GUIEntity**

- Species: String

- Positionx: Int

- Positiony: Int

- Energy: int

- imageType: String

- whoihunt: String

- compassPos: Compass

- currentworld GUIWorld

+ toText()

+ smellFood(char direction, int range)

+ move(Compass dirtomove)

**AMagikarp**

- Species: String

- Positionx: Int

- Positiony: Int

- Energy: int

- imageType: String

- whoihunt: String

- compassPos: Compass

-currentworld GUIWorld

String

- Positionx: Int

- Positiony: Int

- Energy: int

- imageType: String

- whoihunt: String

- compassPos: Compass

-currentworld GUIWorld

- Species: String

- Positionx: Int

- Positiony: Int

- Energy: int

- imageType: String

- compassPos: Compass

-currentworld GUIWorld

**PokeFood**

- Species: String

- Positionx: Int

- Positiony: Int

- Energy: int

- imageType: String

- whoihunt: String

- compassPos: Compass

-currentworld GUIWorld

String

- Positionx: Int

- Positiony: Int

- Energy: int

- imageType: String

- whoihunt: String

- compassPos: Compass

-currentworld GUIWorld

**ASharpedo**

**Obstacle**

- Species: String

- Positionx: Int

- Positiony: Int

- Energy: int

- imageType: String

- compassPos: Compass

-currentworld GUIWorld

**GUIWorld**

Inherits

Lives in

- xSize: int

- ySize: int

- CurrentEntities: Arraylist<GuiEntity>

- randomnum: Random

- currentEntity: int

+ NewMagikarp()

+ NewObstacle()

+ NewPokeFood()

+ NewSharpedo()

+ Checker(int x, int y)

+ Reportfood(int x,int y)

+ entityCanmove(int x, int y, GUIEntity passedEnitity, String hunted)

+ runOnce()

+ worldstats()

**GUIInterface**

- width: int

- height: int

- currentWorld: GUIWorld

- currentfile: String

- show: Boolean

- isPaused:Boolean

- foodcount: int

-obcount:int

- magicount:int

- sharpcount:int

+ start(final Stage primaryStage)

+setupEntities(Page canvas)

+ renderEntities(GraphicsContext gc)

+ fromText(String input)

+ toText(String input)

+ importFile(Stage stage)

+ entityCount()

Interface controls World

Demonstration

The starting menu:

See appendix 1

Normally, there is a default file which is loaded in. Should you need to load a new file in:

See appendix 2

To run the simulation:

See appendix 3

To create a new simulation:

See appendix 4

Saving a file:

See appendix 5

Saving a file with a new name:

See appendix 6

Displaying configuration:

See appendix 7

Editing configuration:

See appendix 8

Display information about life forms:

See appendix 9

Display information about map:

See appendix 10

Modify current life form parameters:

See appendix 11

Remove current life form:

See appendix 12

Add a new life form:

See appendix 13

Pause:

See appendix 14

Reset:

See appendix 15

Display each map at each iteration:

See appendix 16

Display information about application:

See appendix 17

Display information about author:

See appendix 18

Testing

For a program to be properly implemented it must be tested to confirm that it functions to match the requirements and does not cause damage to other components or systems. The first step would be to produce a test plan to make sure as many avenues for error are accounted for.

Test Plan

1.Does the program function run the simulation without crashing? Test 3 times for a minute.

2.Do all the entities update their position every cycle?

3.Can the simulation be paused?

4. Can the simulation be un-paused?

5.Can the animation be reset?

6. Does the saving feature work as expected?

7. Can you open a different saved file successfully?

8. Can the animation be turned off without stopping the simulation?

9. Can lifeforms be added without needing to open a new file?

10. Can lifeforms be removed with needing to open a new file?

11. Can any lifeform be modified (by changing its x/y position and species)?

12.When trying to modify a certain lifeform, does the program handle a lifeform ID which is not present?

13. When the displaying the configuration is the data correct?

14.Can you edit the world configuration correctly?

15. Does the information about displaying lifeforms only contain ‘living’ entities?

Test Results

Test 1. This test was successful and in all attempts the program did not crash and the entities acted as they should

See appendix 19 for screenshots

Test 2. This test was a success as all entities in the test moved.

See appendix 20 for screenshots

Test 3. This test was a success as the entities stopped moving when the pause button was pressed

See appendix 21 for screenshots

Test 4. This test was a success as the entities then restarted to move after the button was re-clicked.

See appendix 22 for screenshots

Test 5. This test was successful as the world reset with the same amount of and type of entity as when it started.

See appendix 23 for screenshots

Test 6. This test was a success as the default world was saved successfully as a separate file name.

See appendix 24 for screenshots

Test 7. This test was a success as the new file created in the previous test was opened successfully and then run.

See appendix 25 for screenshots

Test 8. This test was a success as the animation was turned off for 10 seconds and then turned back on. The entities had moved and some had been eaten and removed from the world.

See appendix 26 for screenshots

Test 9: This test was successful as 3 new entities were added into the world without it starting again

See appendix 27 for screenshots

Test 10: This test was also successful as these entities were then removed from the world.

See appendix 28 for screenshots

Test 11: This test was a success as the first entity in the array normally of type Pokefood was changed to a Magikarp and had its position changed.

See appendix 29 for screenshots

Test 12: This test was a success as the number 999 was entered and a suitable error message was created.

See appendix 30 for screenshots

Test 13: The configuration output by the program matches the file, so the test was a success.

See appendix 31 for screenshots

Test 14: The configuration was changed successfully

See appendix 21 for screenshots

Test 15: The lifeform box correctly only shows the living entities (Magikarp and Sharpedo)

See appendix 22 for screenshots

Discussion

Through the development of this project several points of interest have arisen. Firstly, simulating simple lifeforms are relatively simple and the project could be expanded further to simulate more complicated behaviour. However, at this complexity there are several simple types of microbes whose actions are very close to those of the prey and predator which were created in this project. If there was more time the program would be up-scaled to gradually include more complex entities which match the behaviour of real world lifeforms. The 2D world would be changed to include depth and breadth to better match real life. This would allow more realistic simulations which, with a simulation, is what you strive for.

More complex control features would also be added in the future so that users can change the software more according their needs. An example of this is a feature where a custom entity can be created by the user. Changing the menu system so that the GUI is easier to use and making the whole experience faster so that users do not get confused and need support to operate.

Conclusion

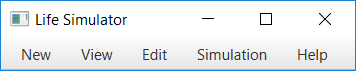
To conclude, the output of this project is a full-functioning virtual life simulator. There are several improvements that could have been made that were not due to time constraints e.g. more complex entities and greater user control. Nevertheless, there are minimal bugs and matches the majority of the requirements. The most notable omission is of poisonous food and renewable food. Overall, the program runs smoothly and so a decision was made to stabilise this version instead of rushing the implementation of the other entities mentioned.

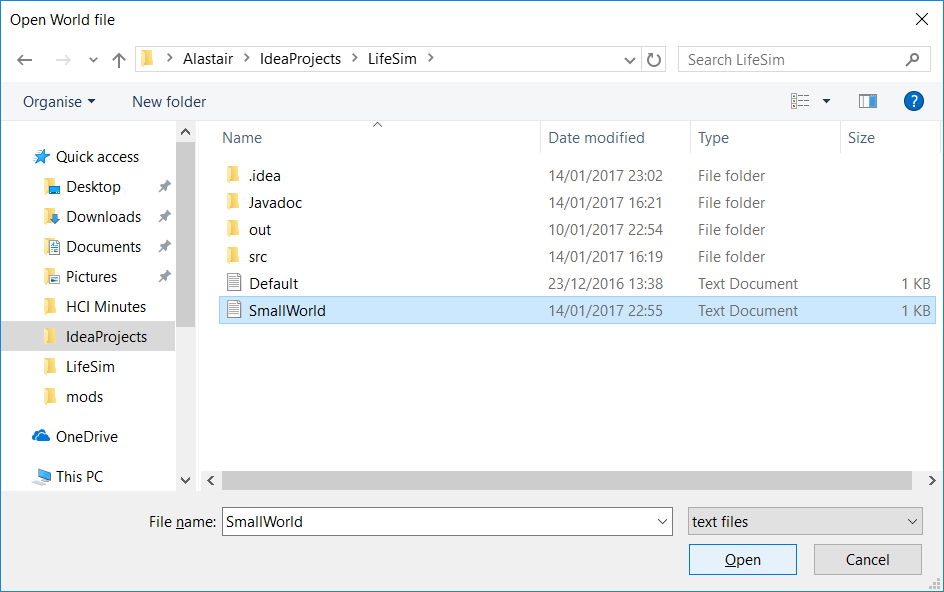
Personal Reflections

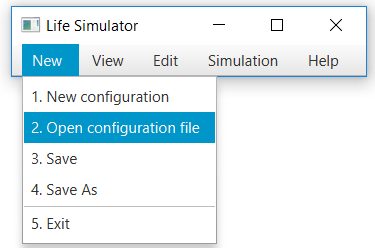
I feel that, overall, this project went fairly well. Most of the objectives were met successfully and the program is relatively bug free. I feel the program code is well laid out which aids the readability of my work.

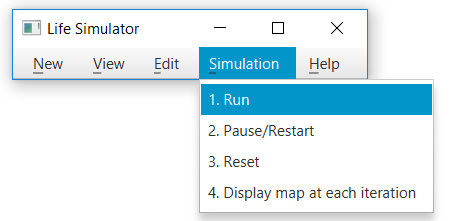
There are several points to be to be improved and these have been outlined in the discussion. The highest priority would be the additional types of entity as outlined in the original specification. This would be followed by the additional control features. I would have also written JavaDoc comments simultaneously to the classes and methods as this would help prevent forgetting what code does.

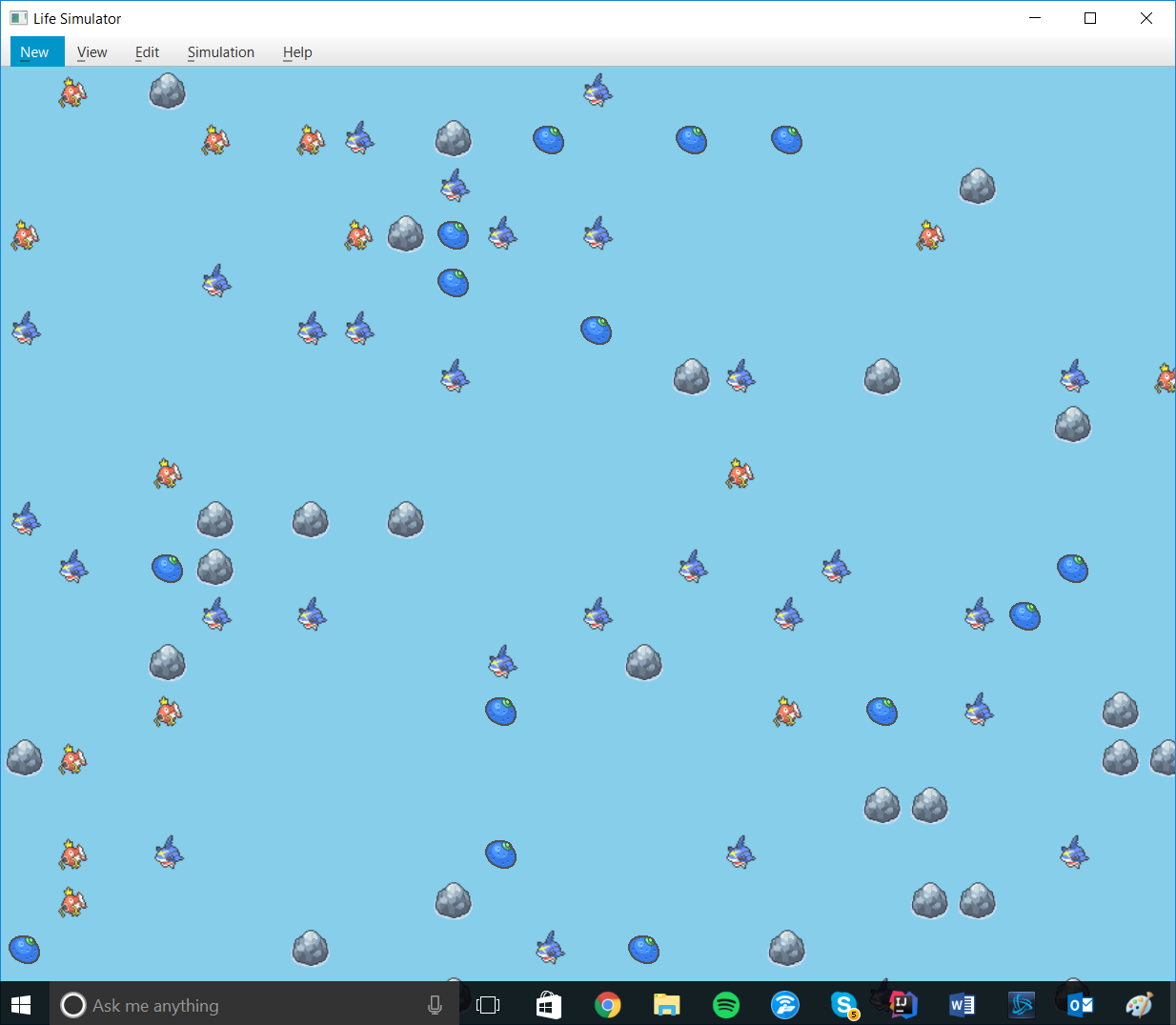
Appendix

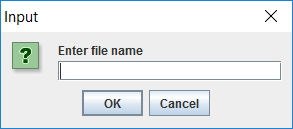
1.

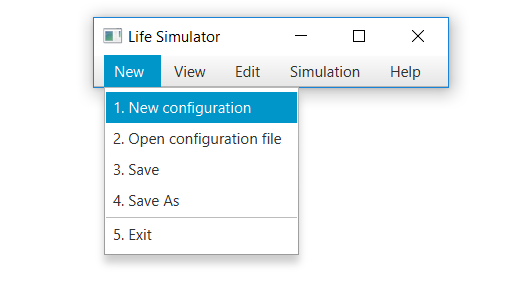
2.

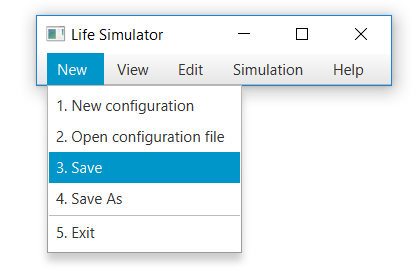


3.

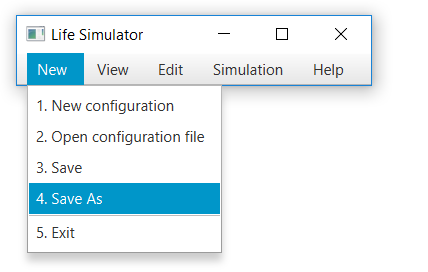


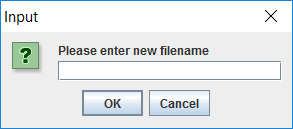
4.

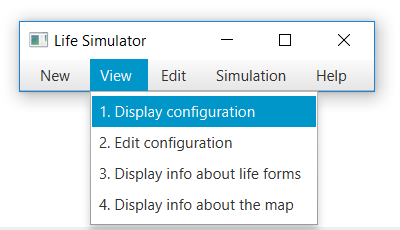
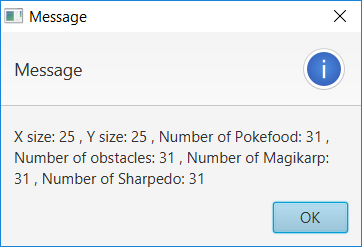
5.



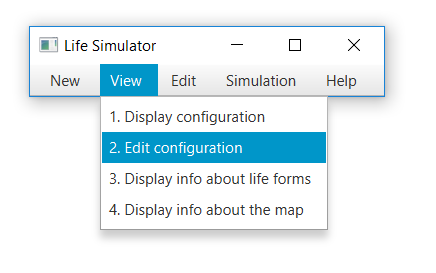
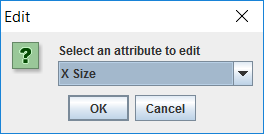
6.

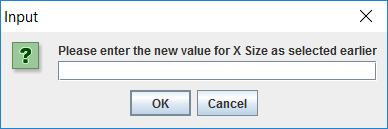


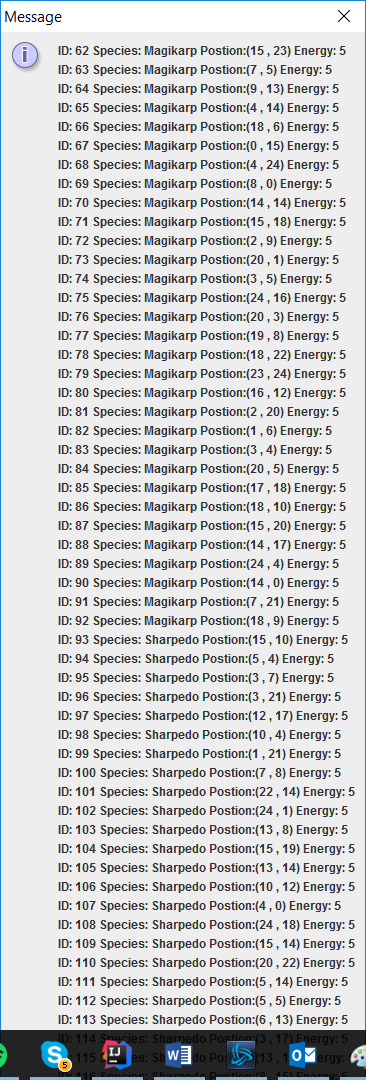


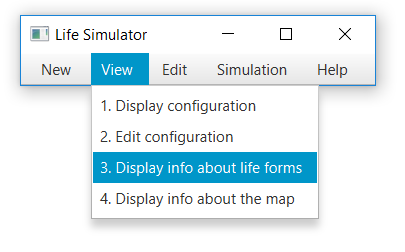
7.

8.

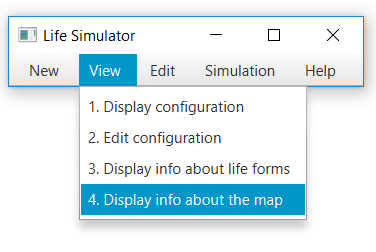
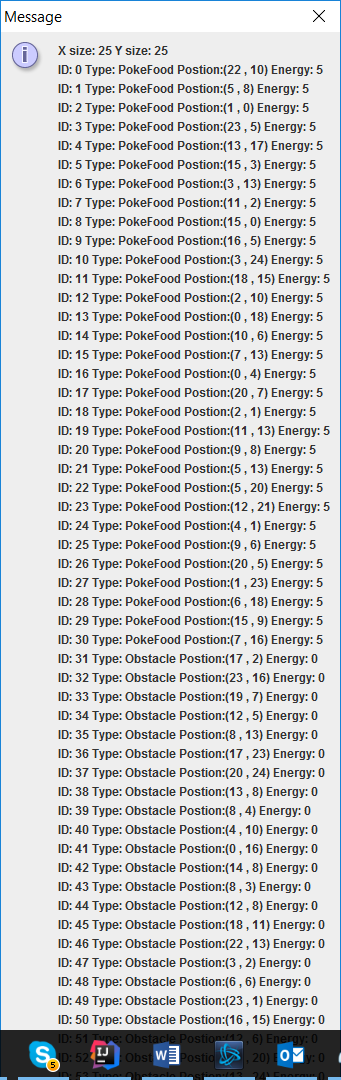


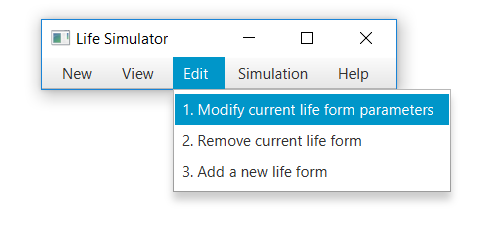
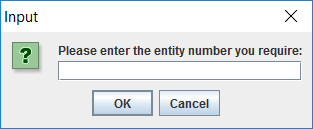
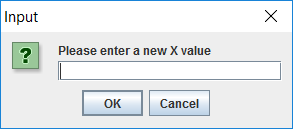
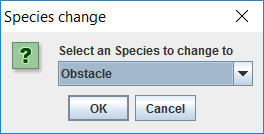


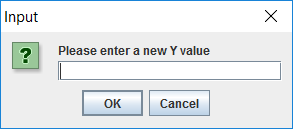
9.

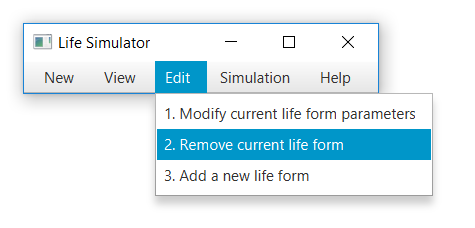


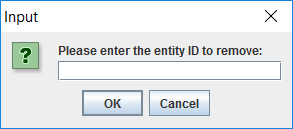
10.

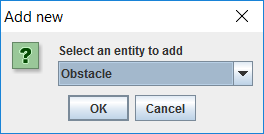


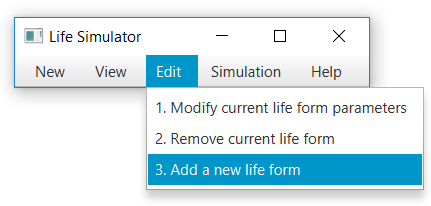
11.

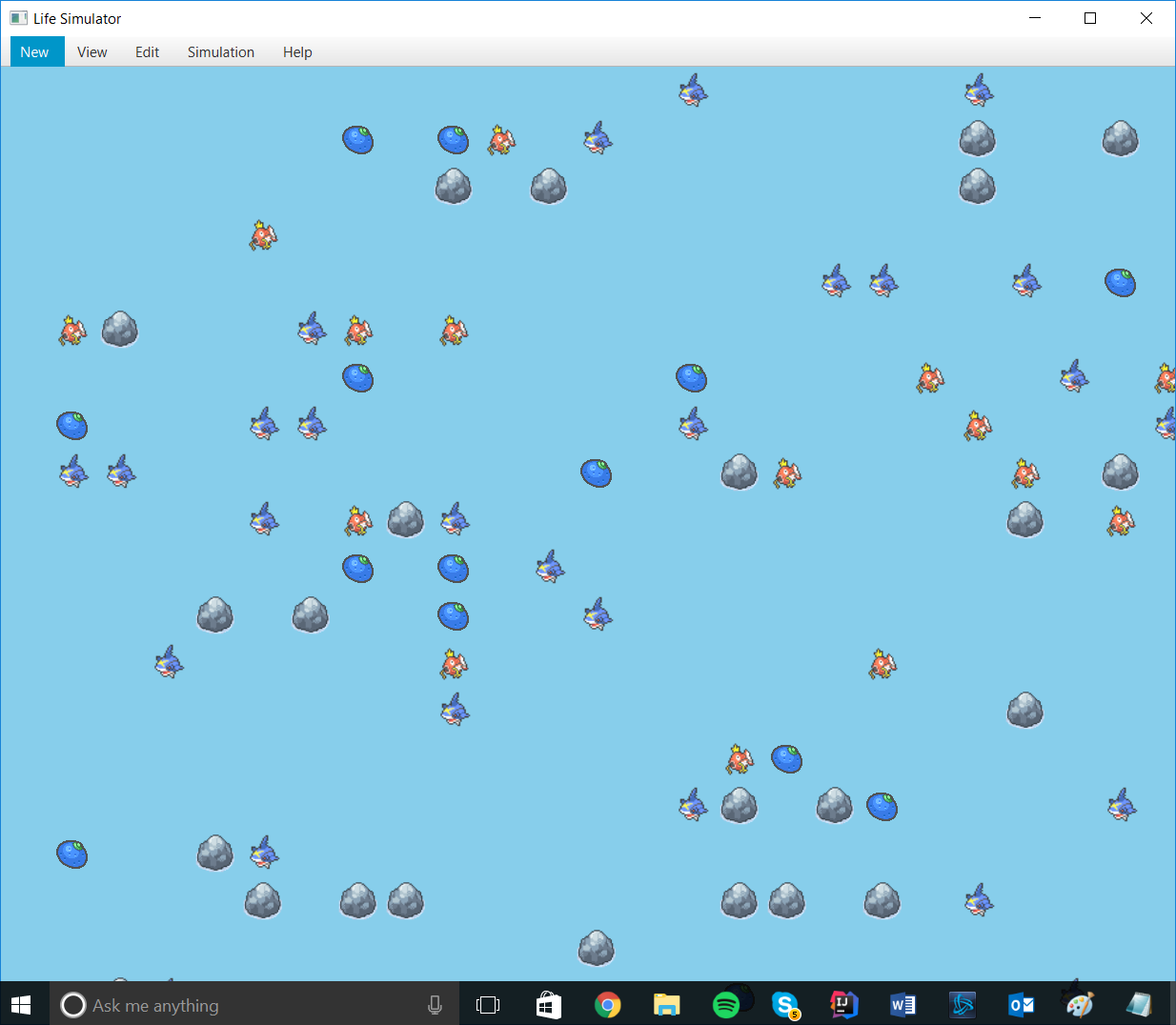


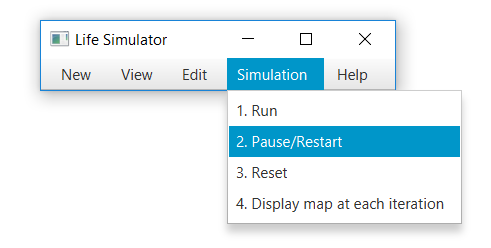
12.

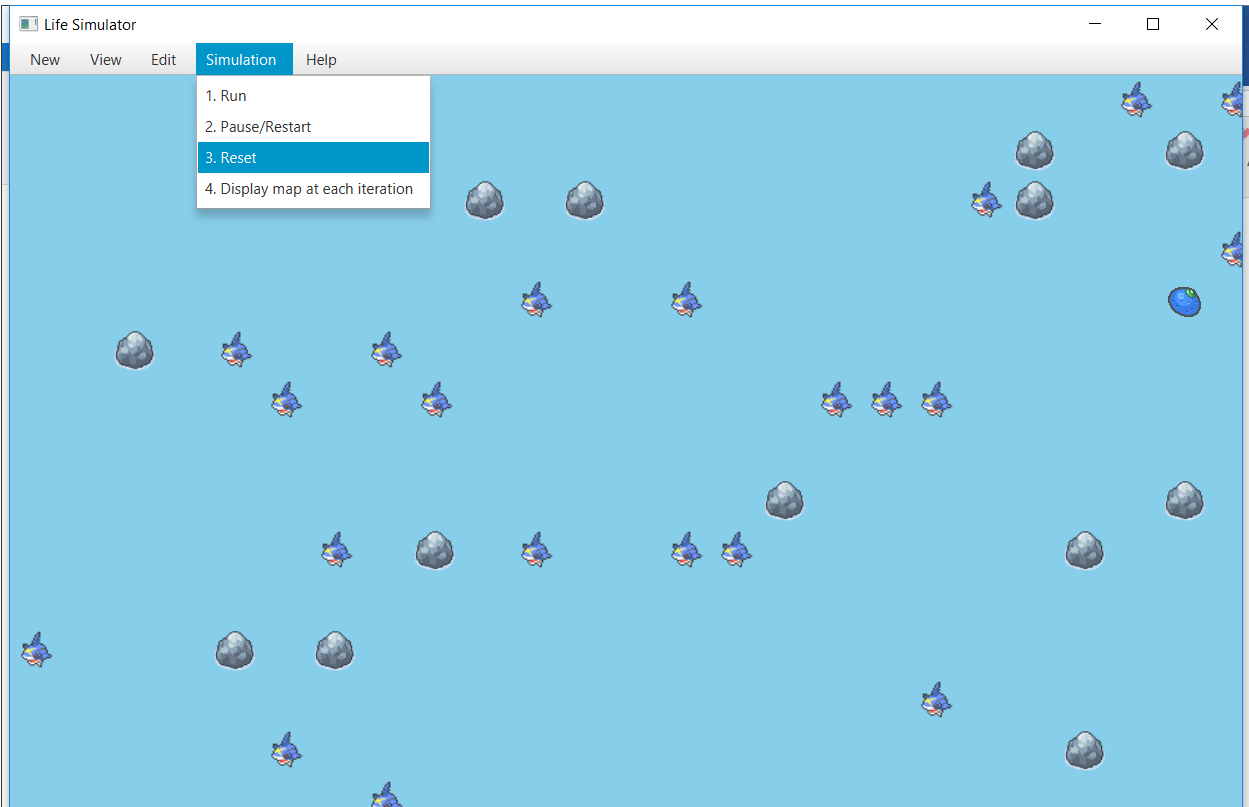
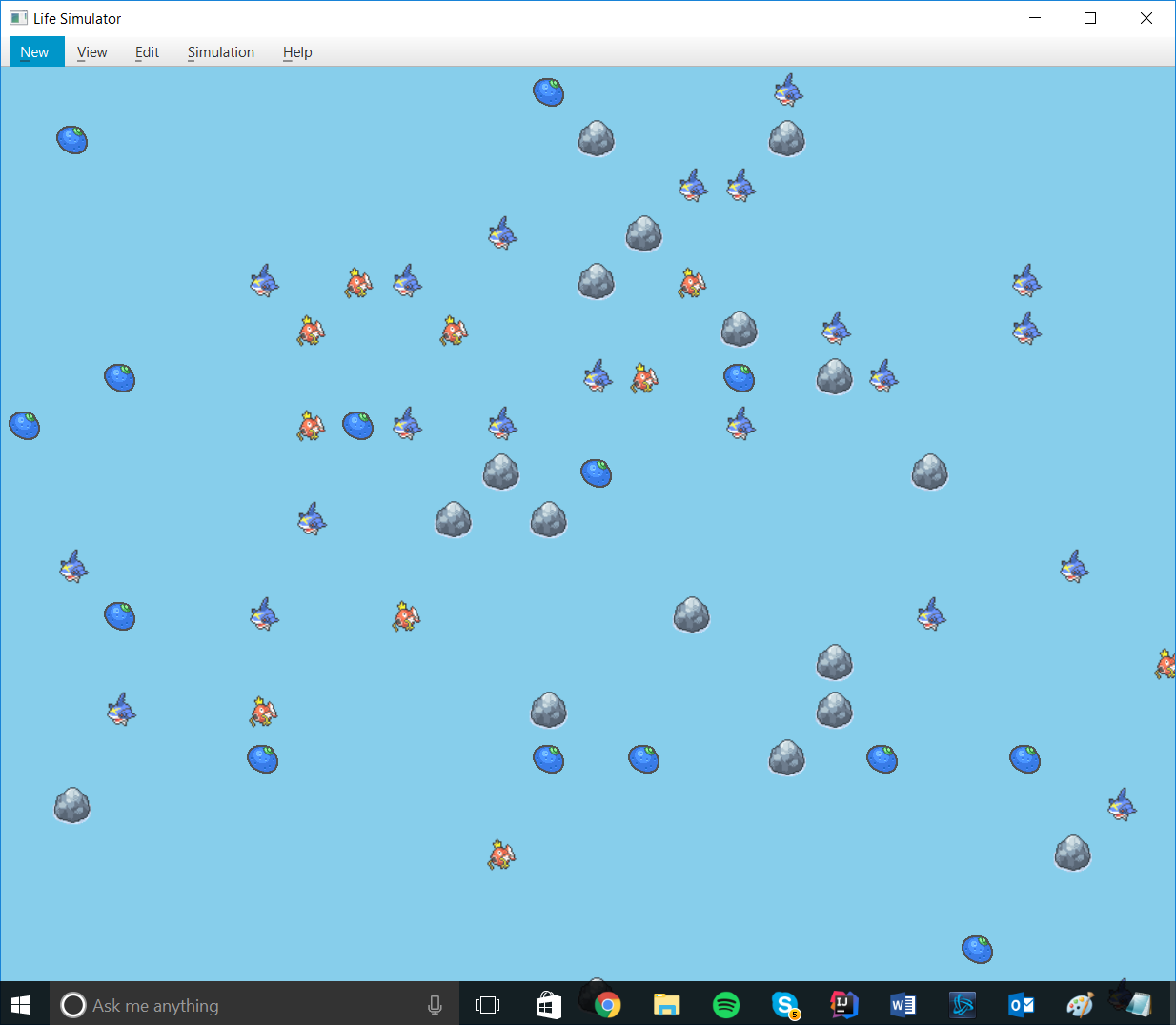


13.

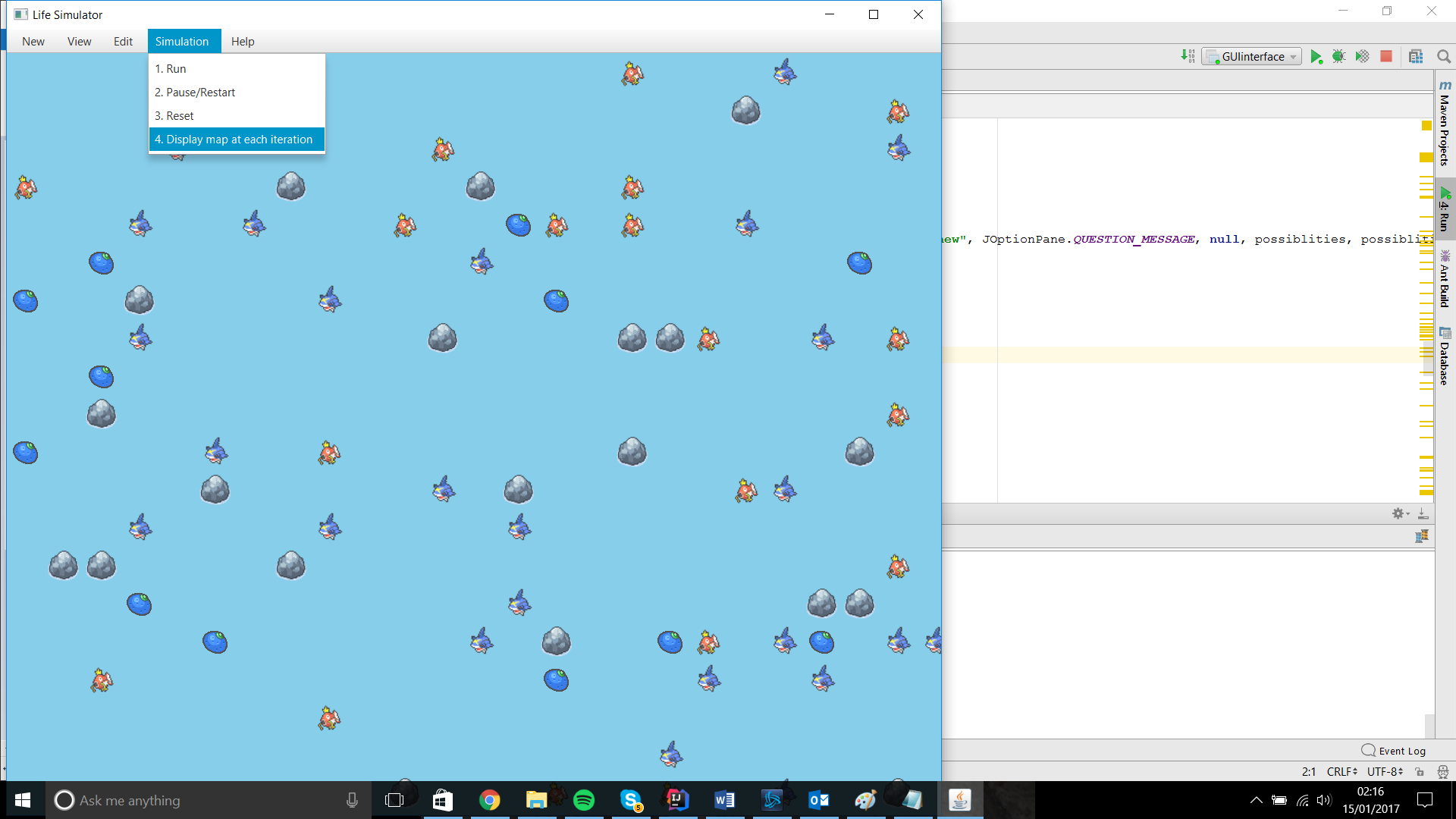
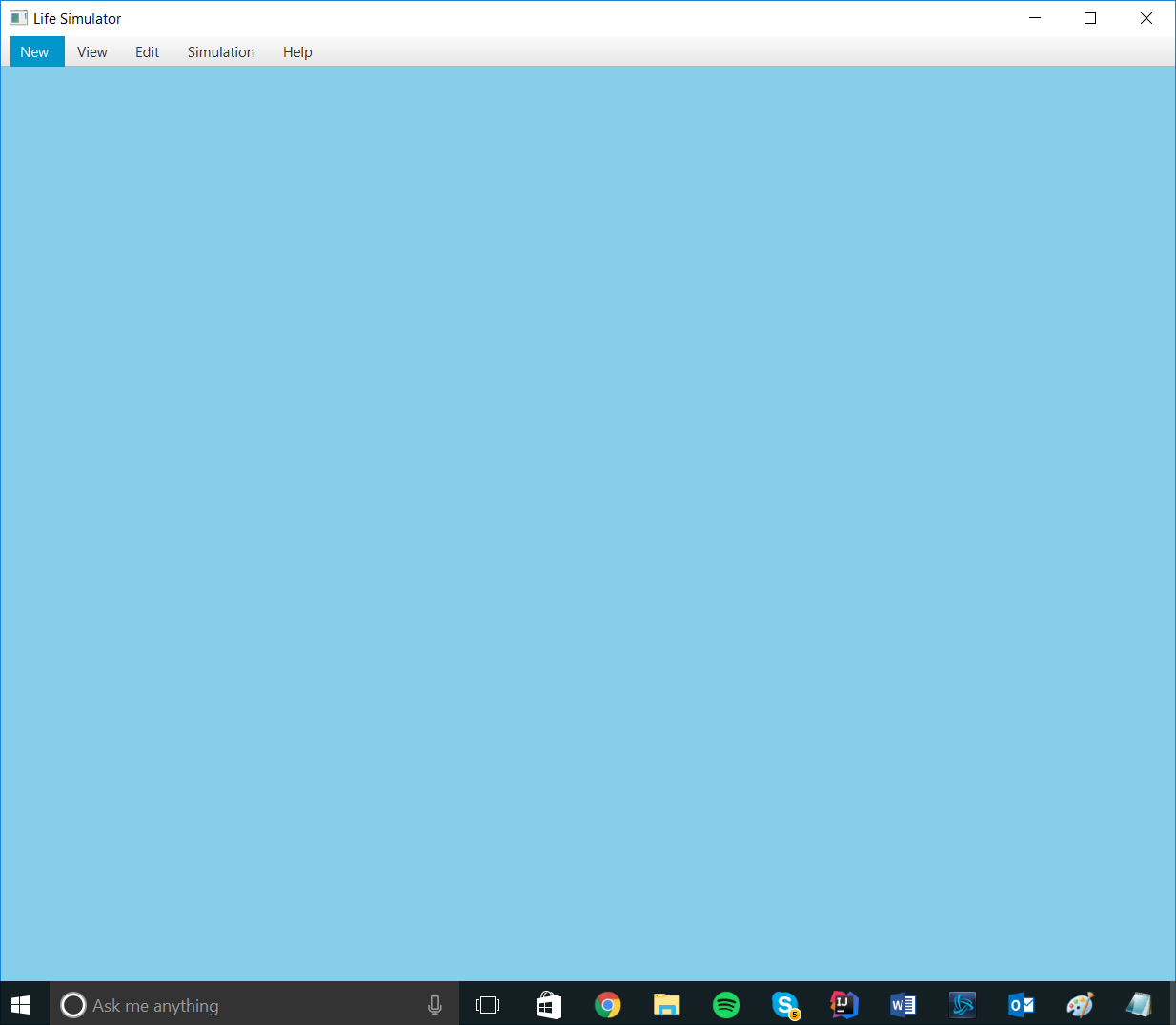


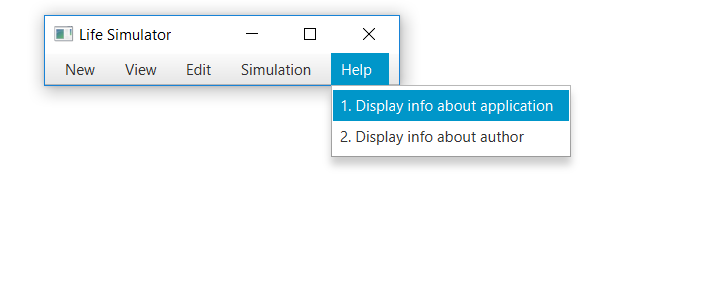
14.

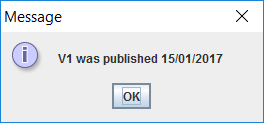


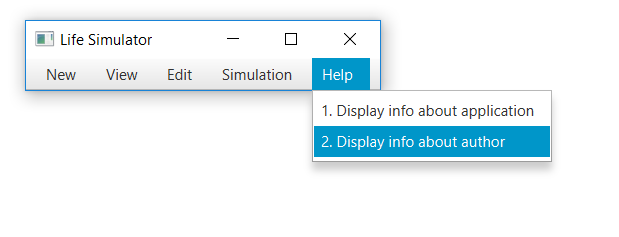
15.

16.

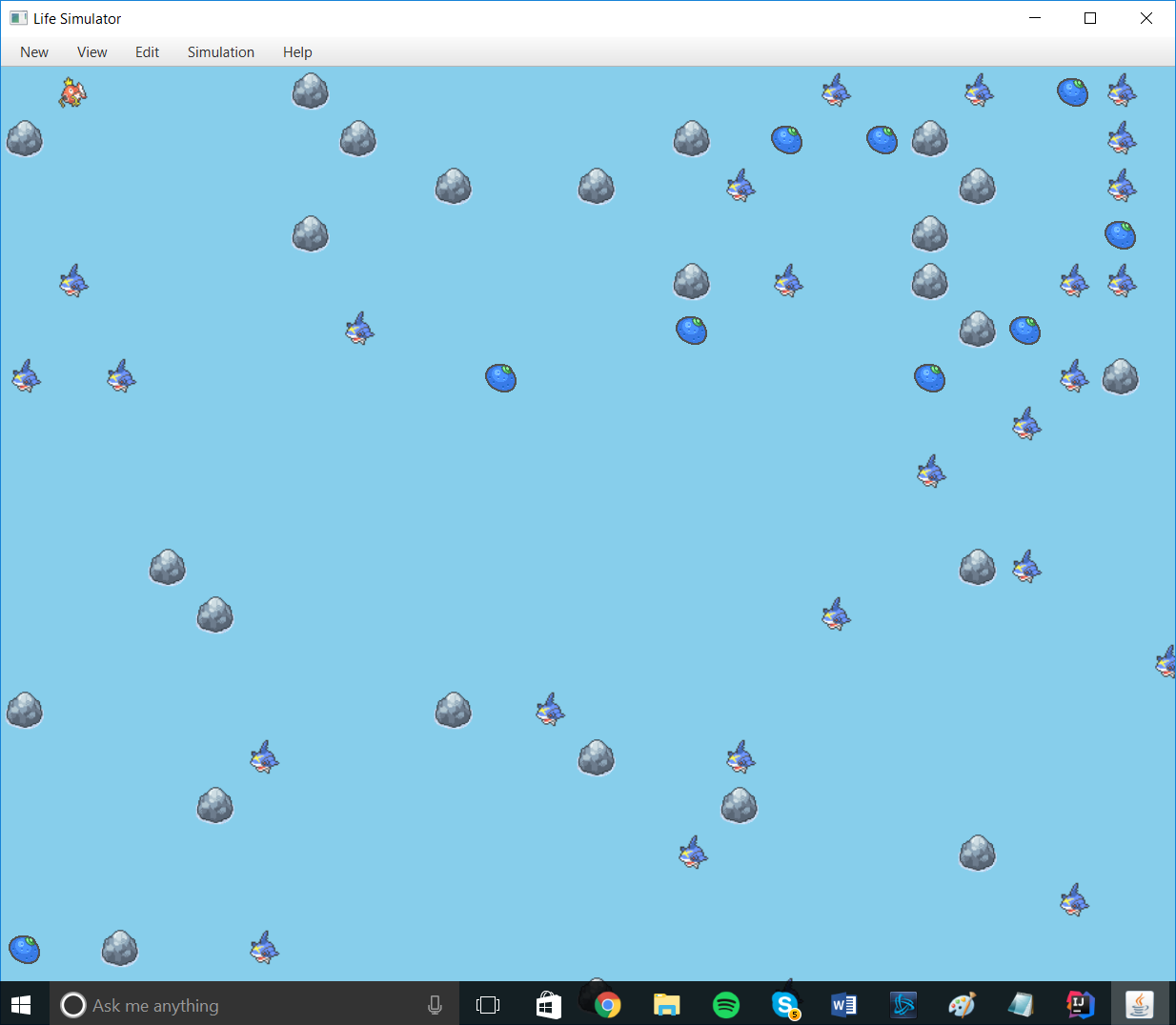
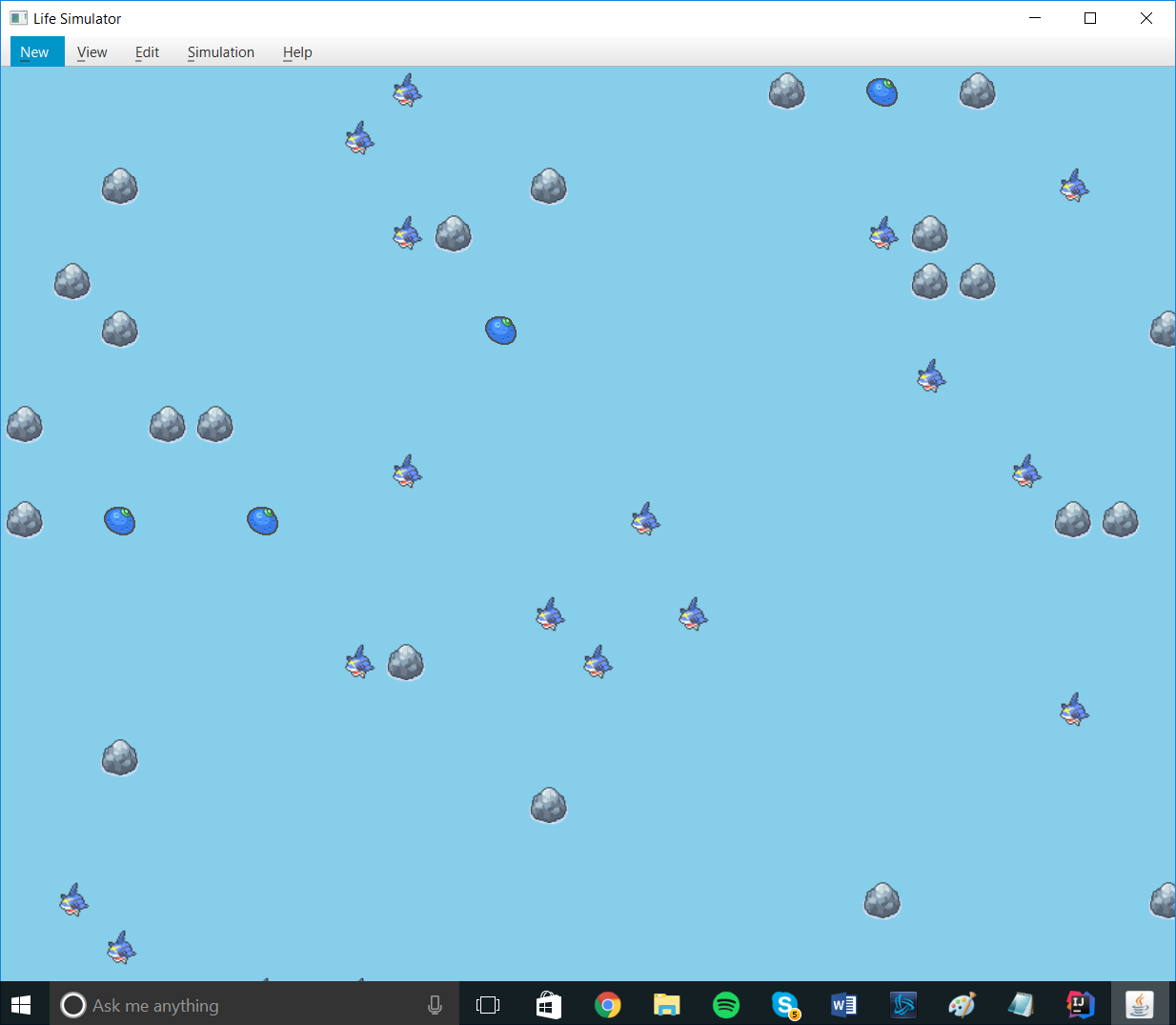
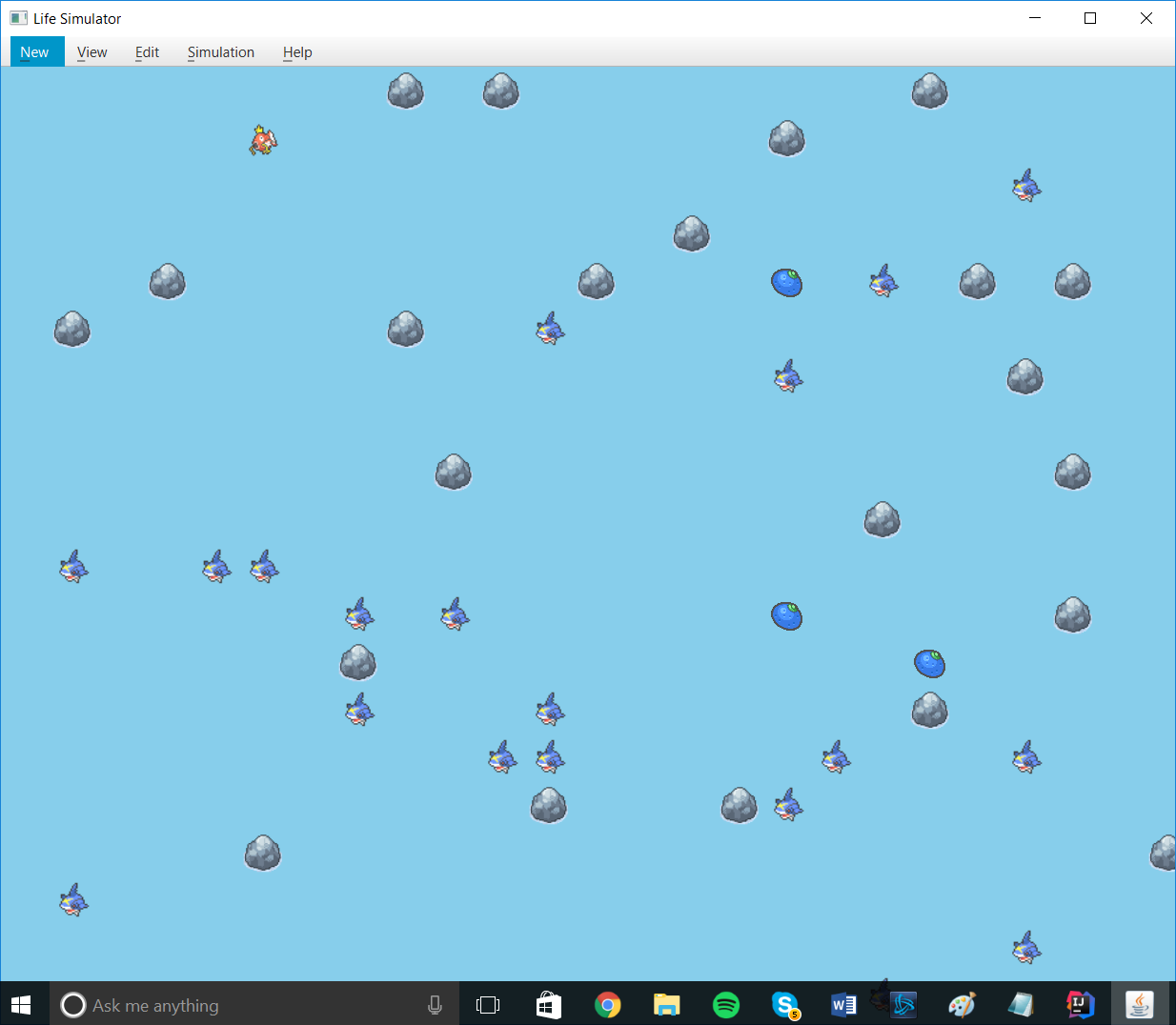


17.

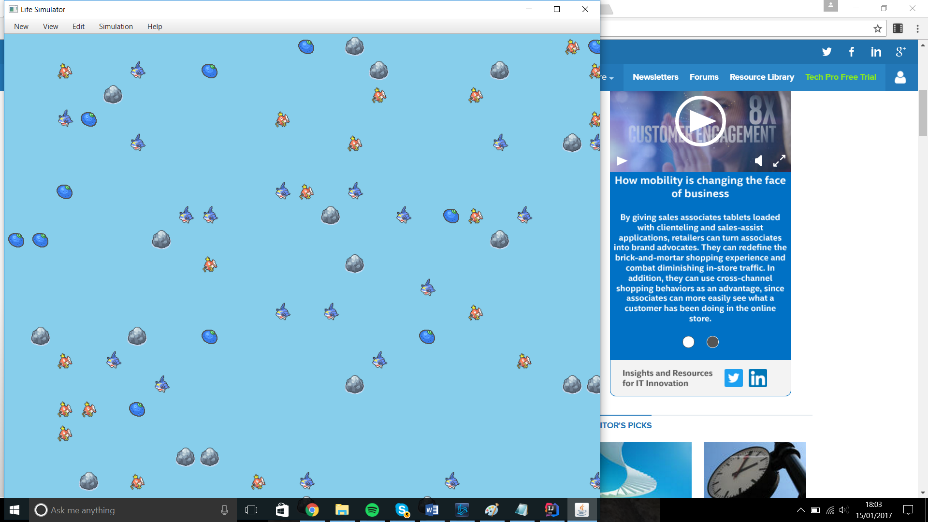
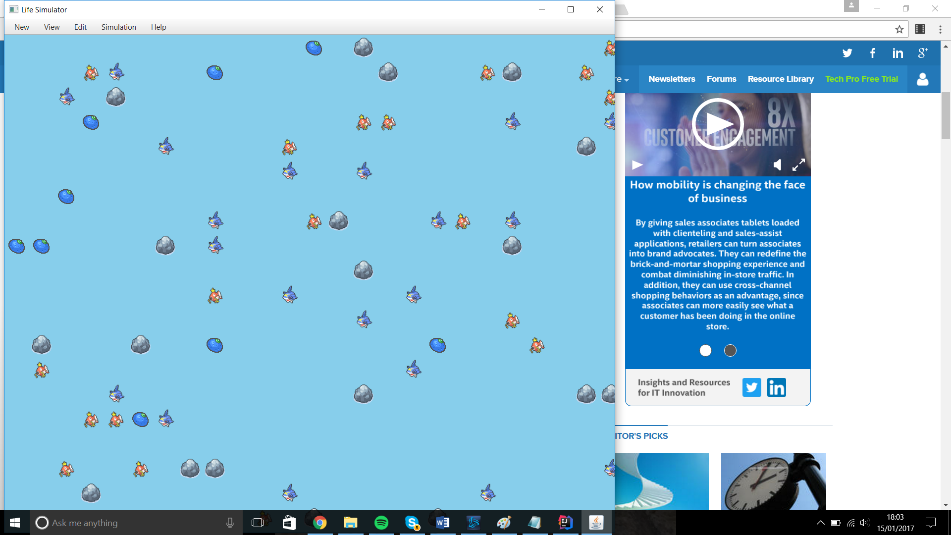


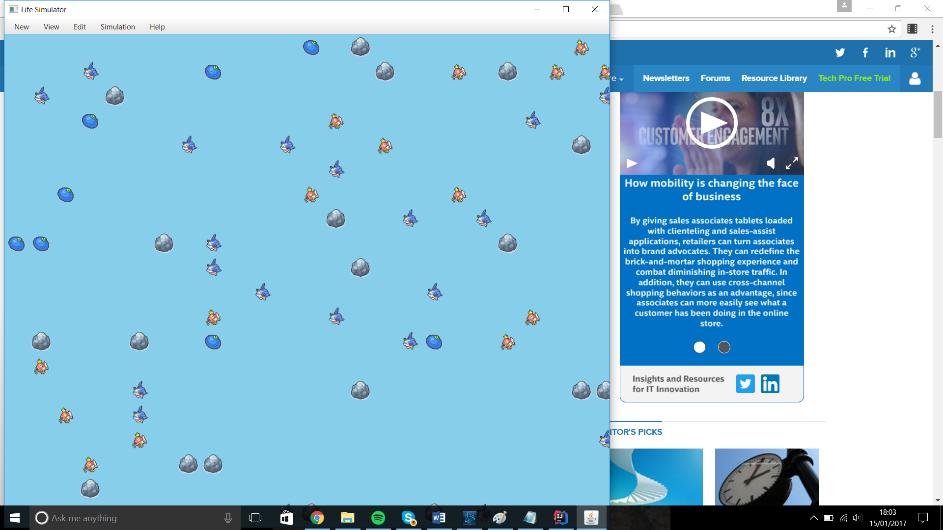
18.

Testing

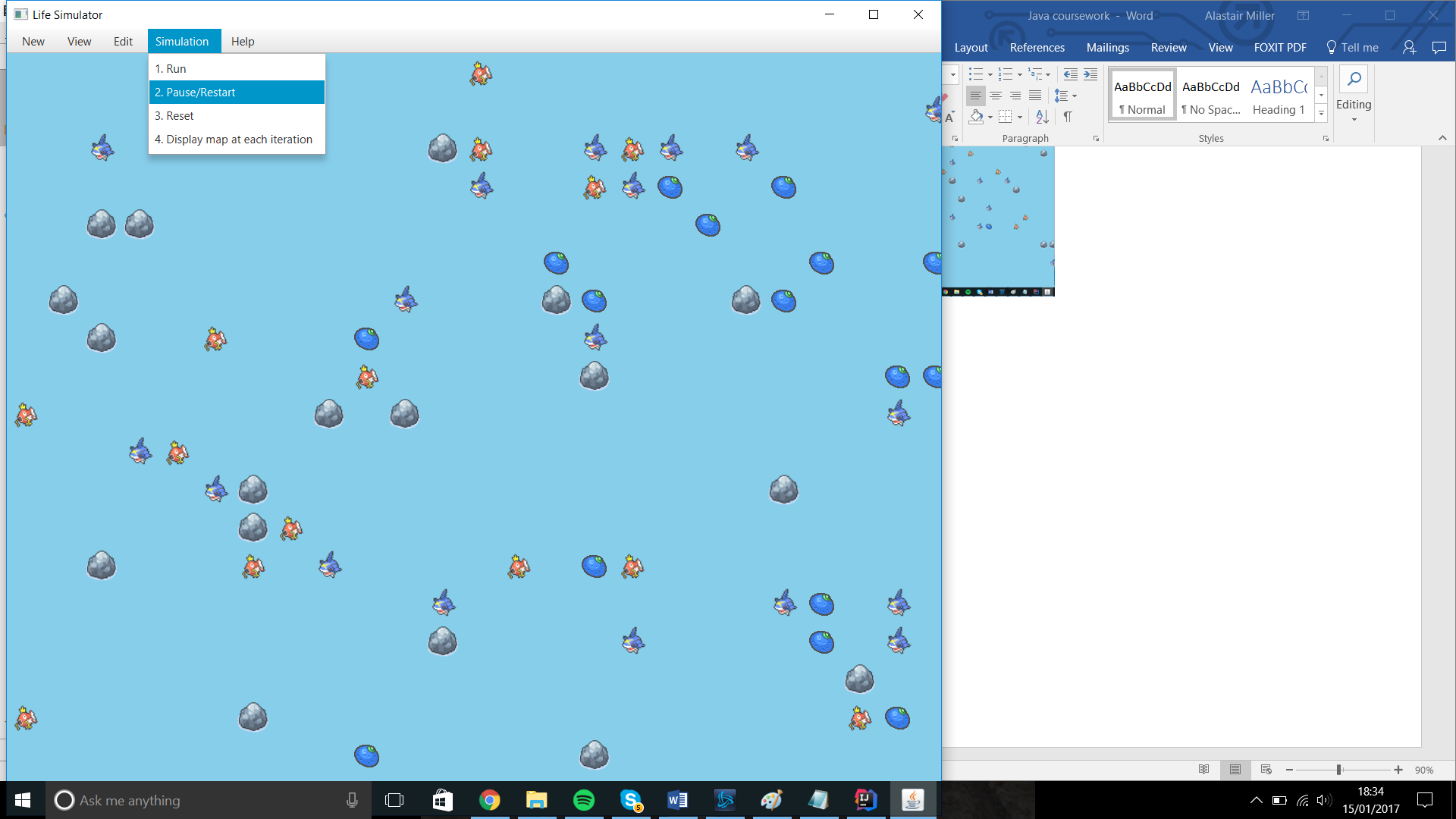
19. After ~1 min

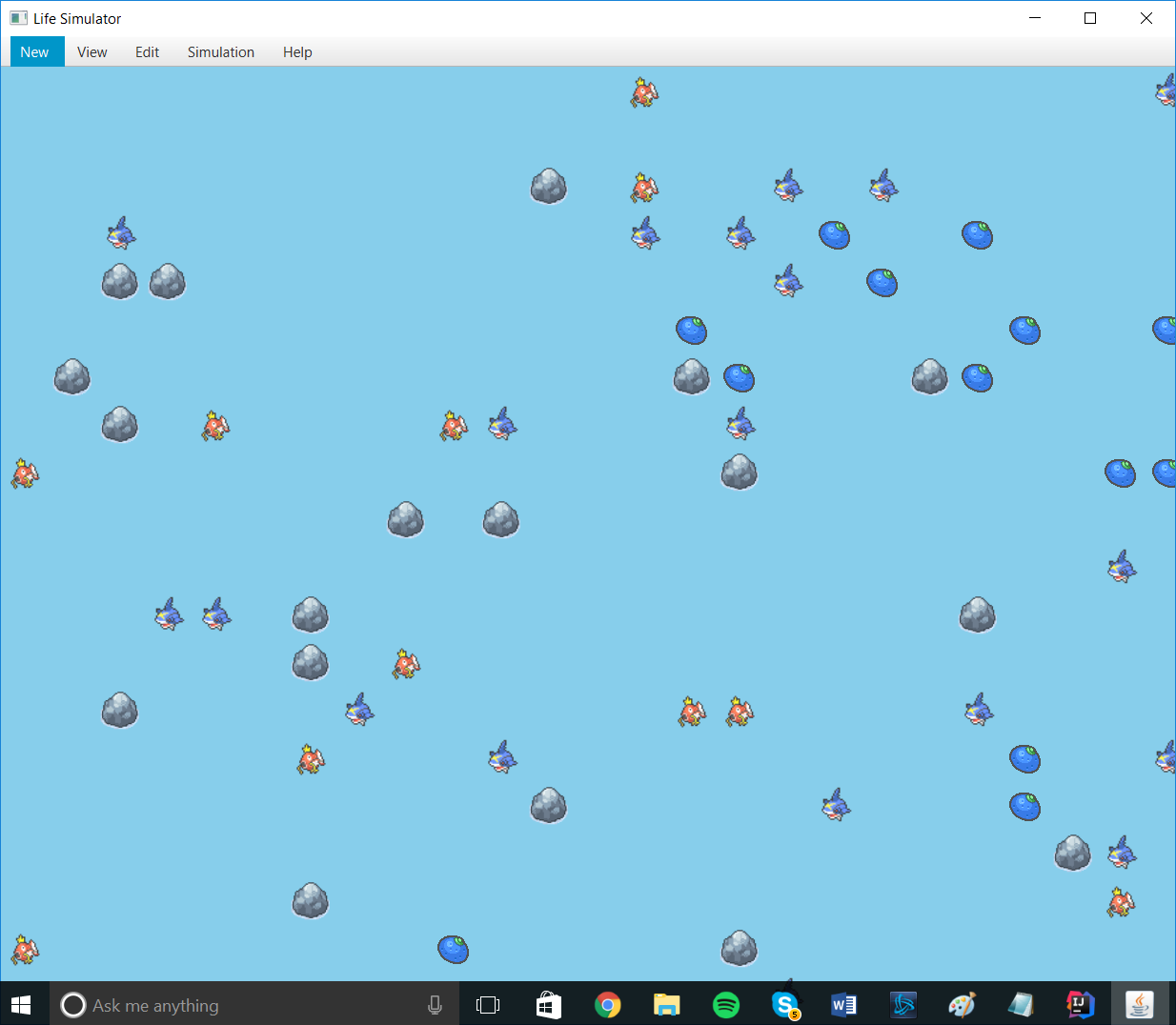
20.3 consecutive cycles



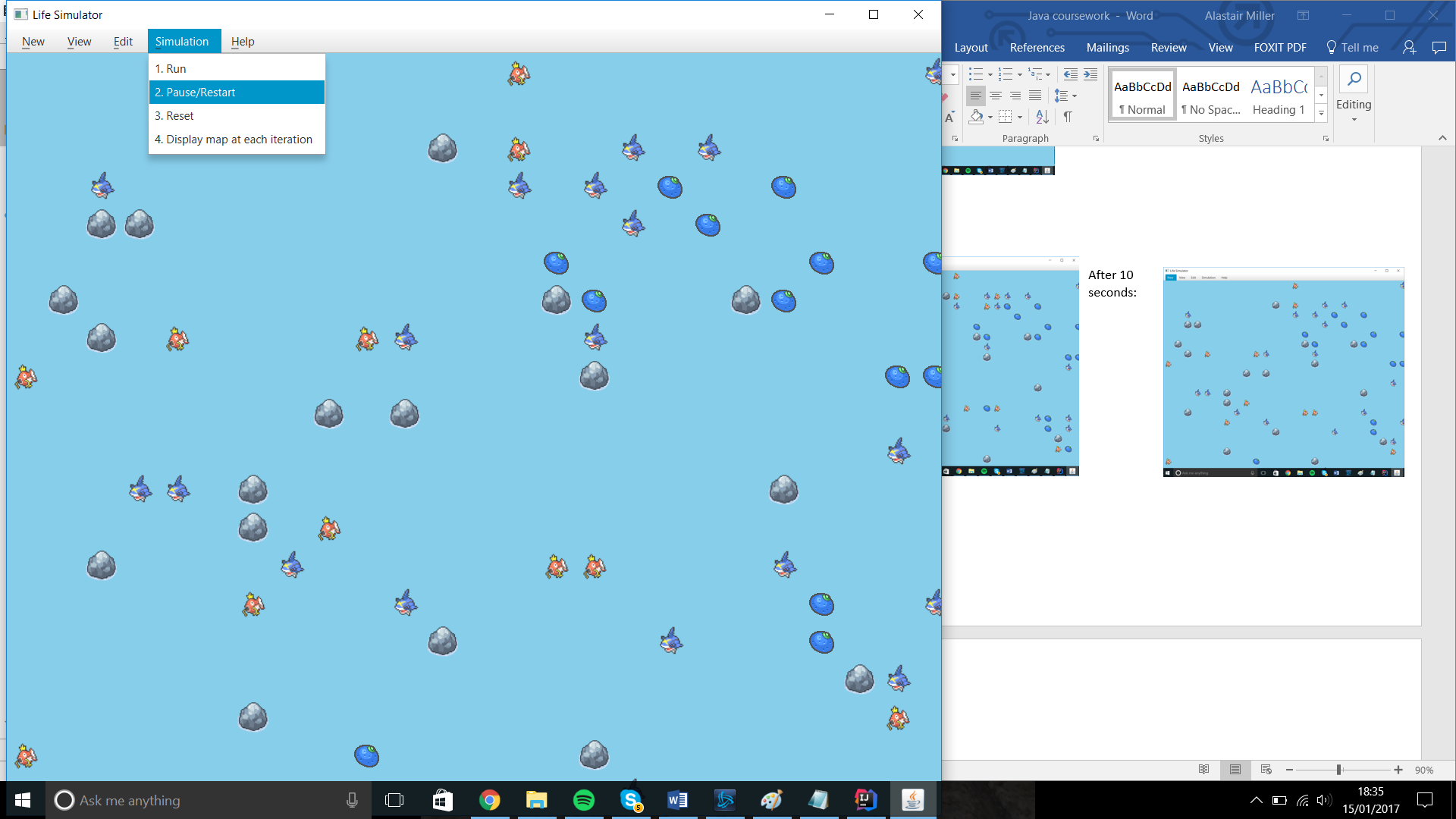
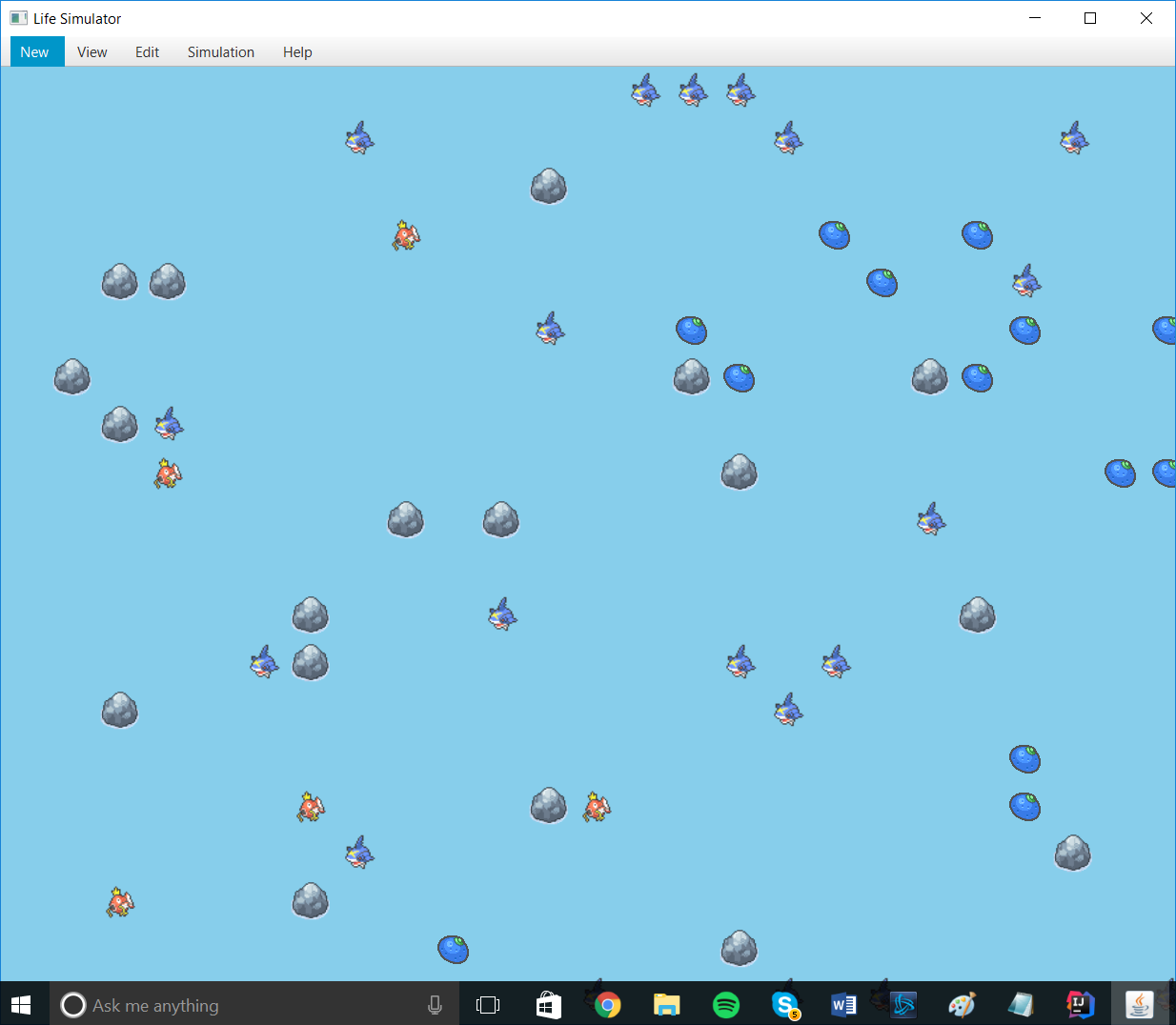


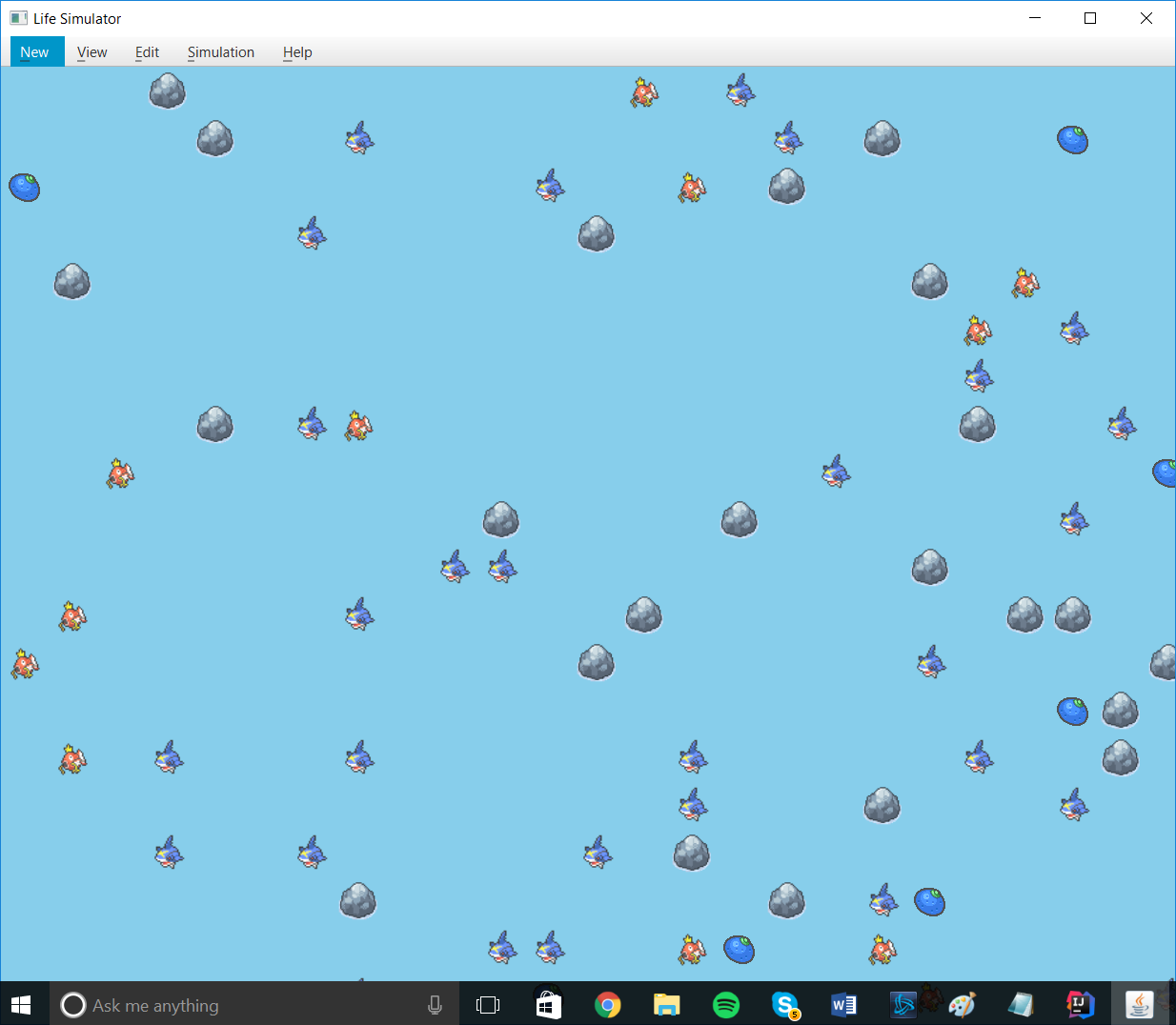
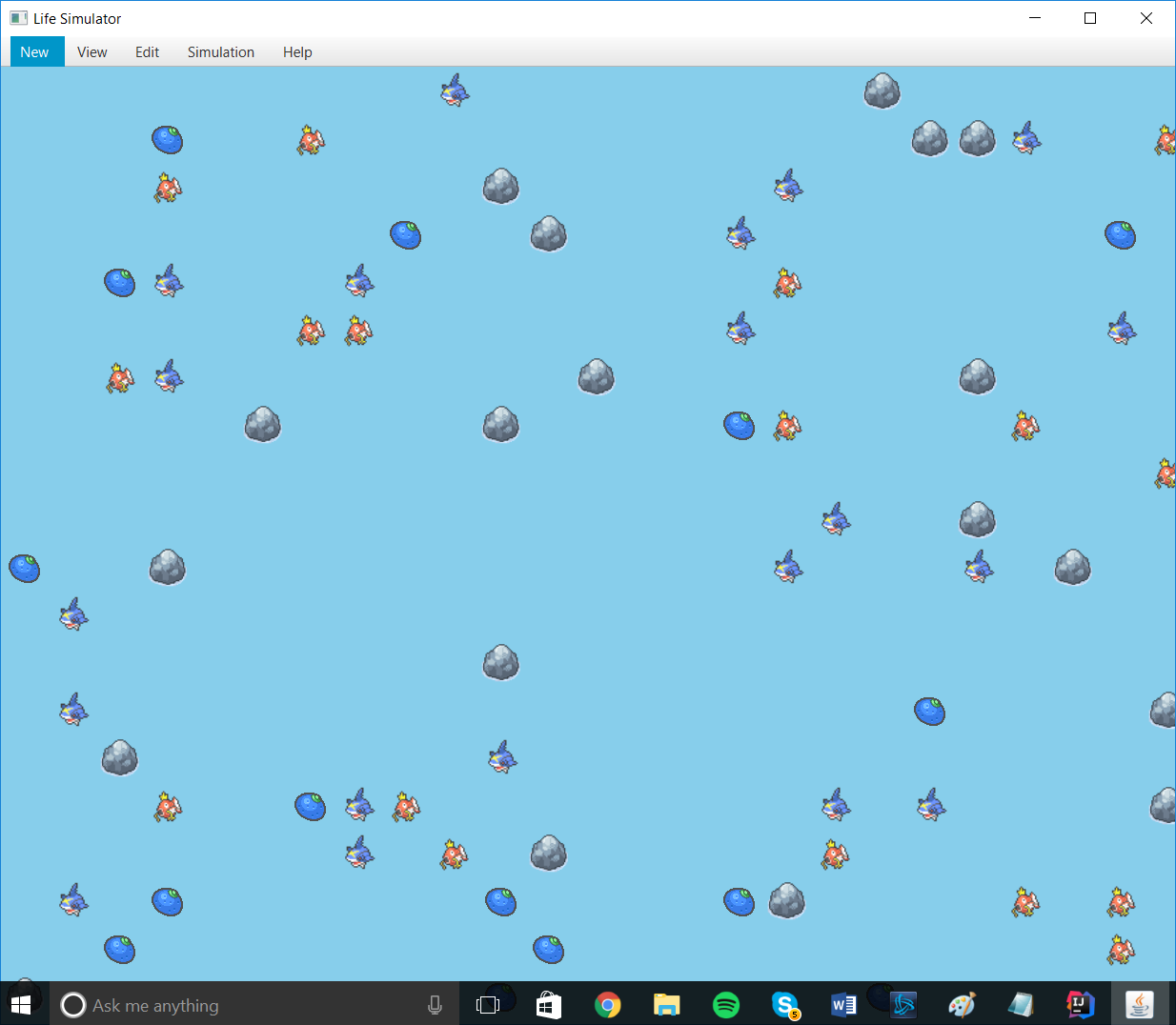
21.

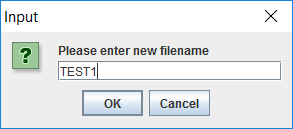
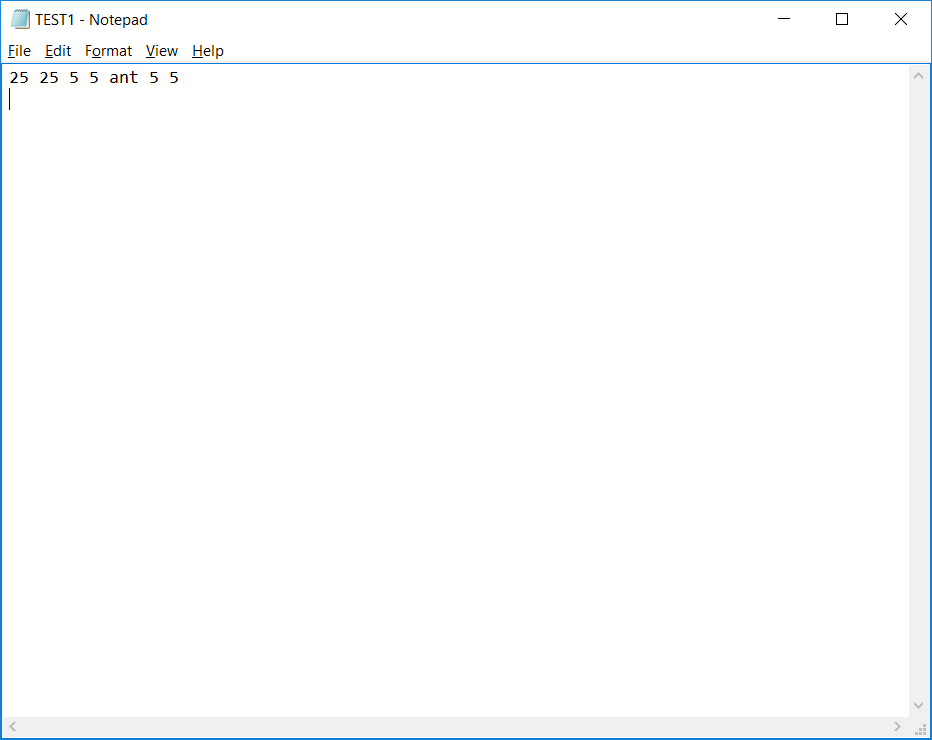


After 10 seconds:

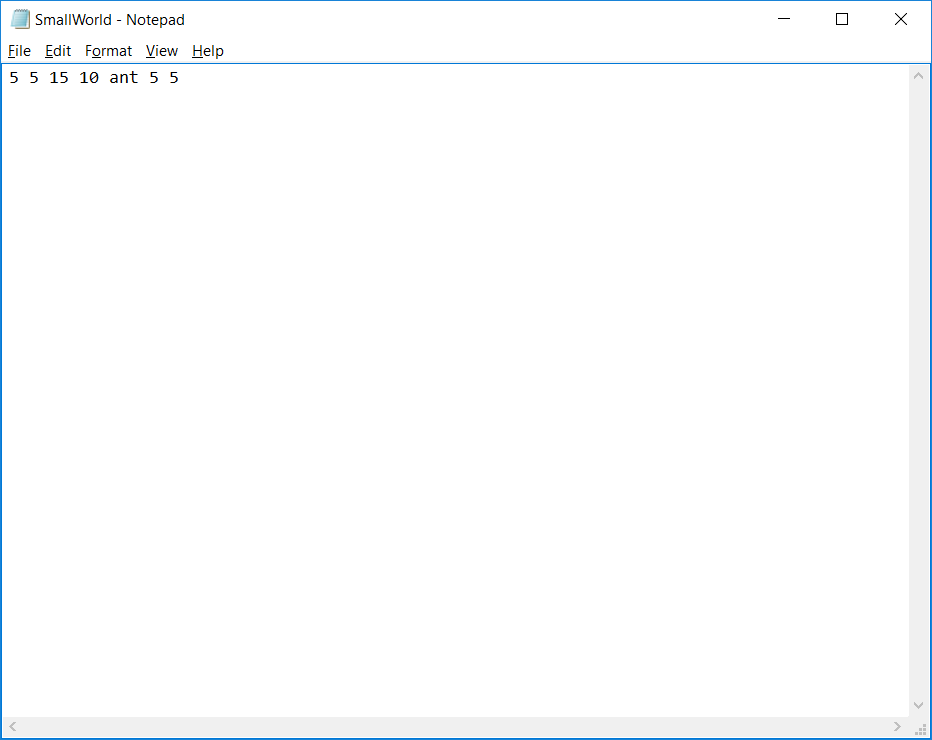
22.

After 10 seconds

23.

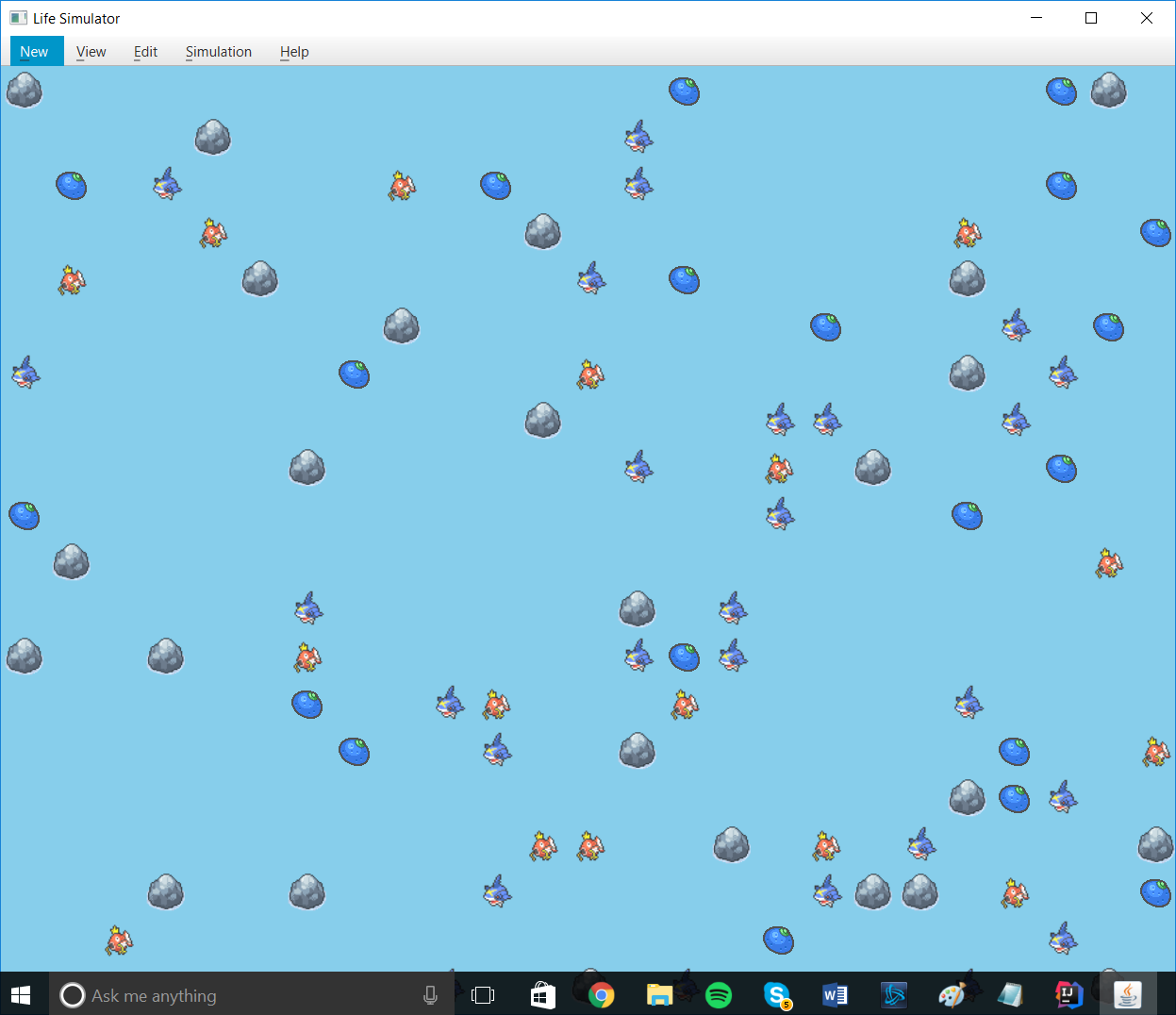
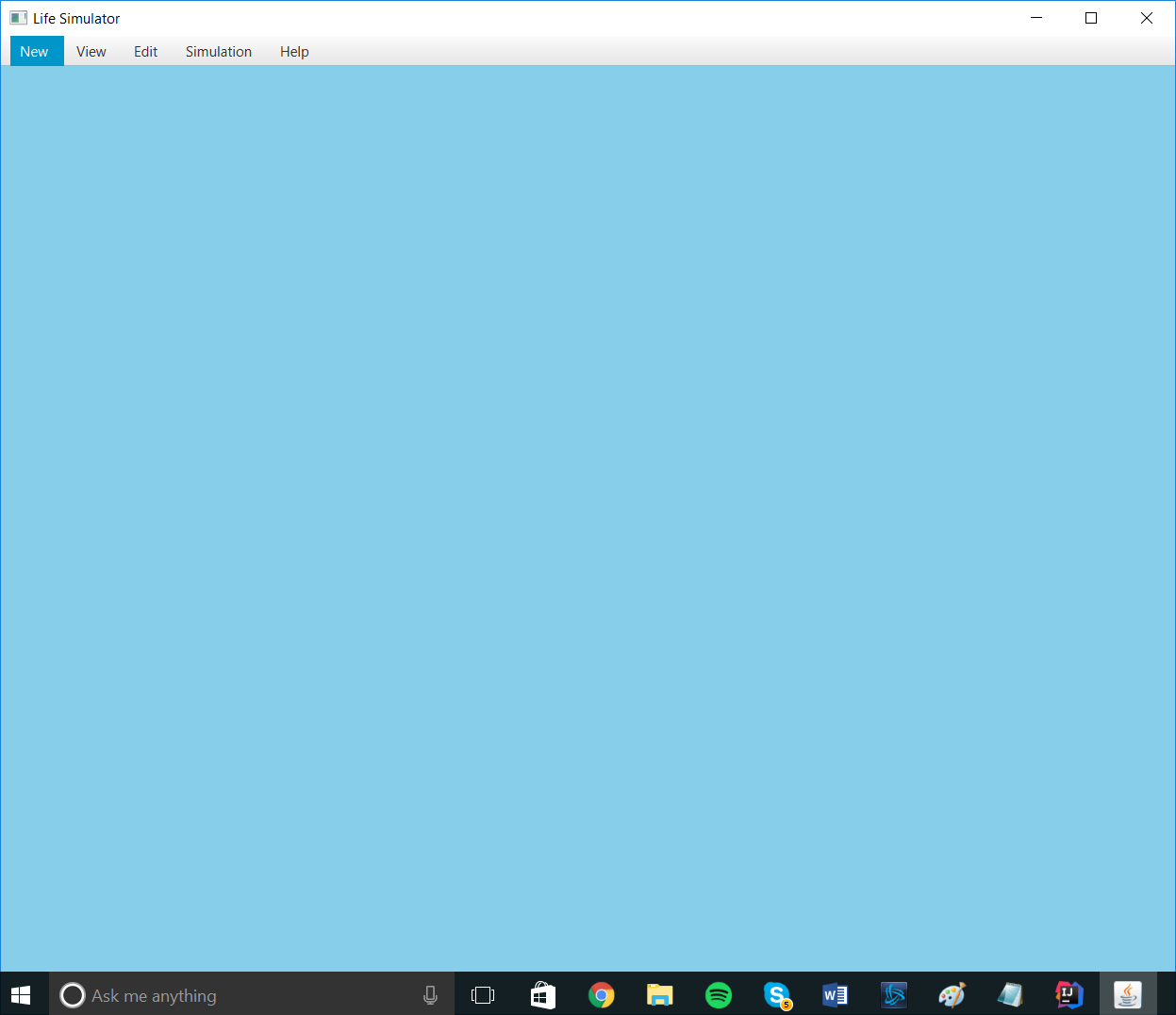
24.



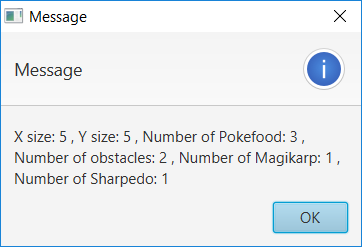


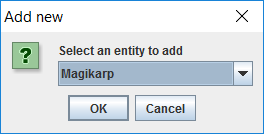
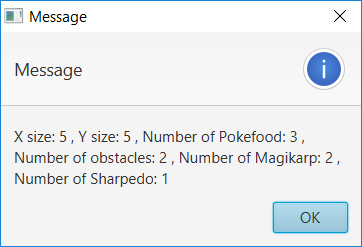
25.

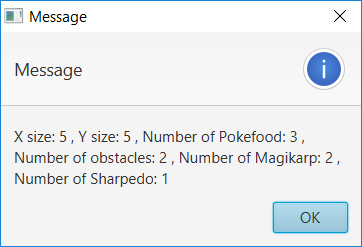
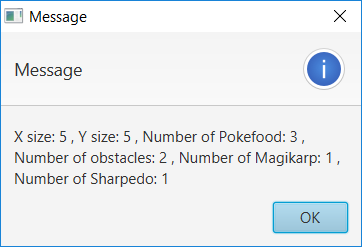
26.

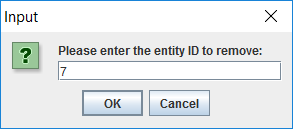


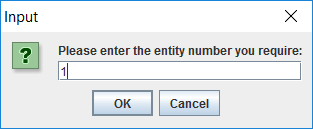
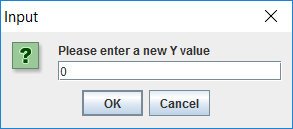
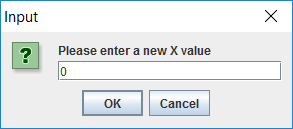
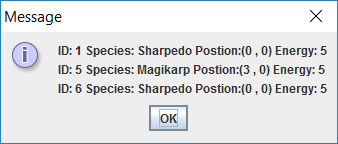
27.



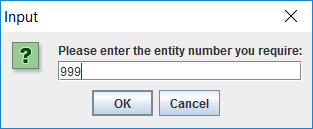
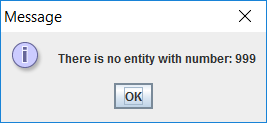


28.

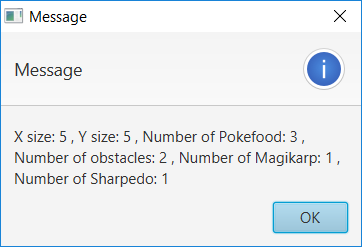


29.

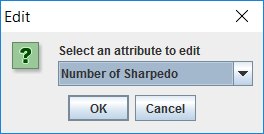
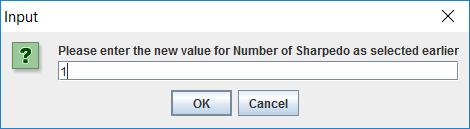
30.

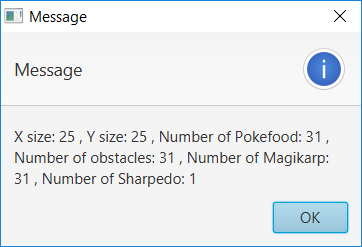


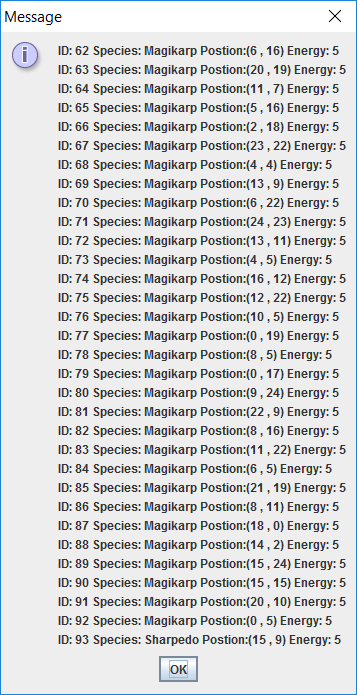
31.



=

32.



33.