1.0 Graph to represent the Behavior of a Single Automat via Nassi-Schneiderman

Figure.1 Nassi-Schneiderman's diagram graph shows the behavior of a single automat. This updates for different generations. The field is updated with numbers "**0 and 1**". Where "**0**" represents the state of the automat as dead and "**1**" represents that the automat is alive.

This method used to represent the behavior is called "**UpdatePopulation**" and it is implemented in the **PopulationBuilder** C# class.

Figure.2 is a method "**CalculateGenerations**" used to represent the behavior for a specified number of generation/evolution.

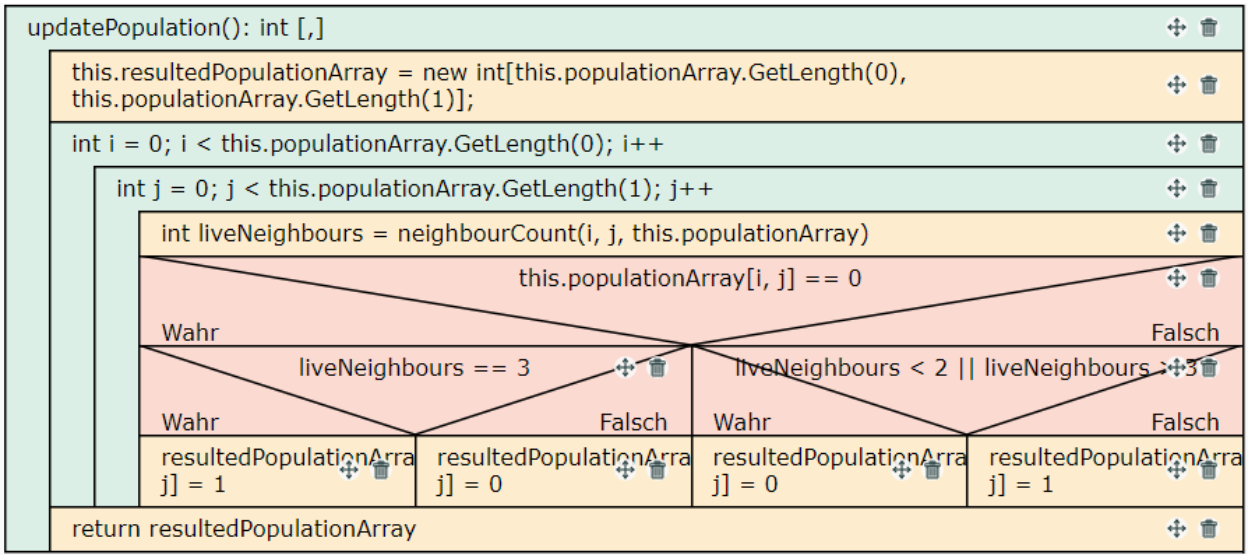

Figure 1: Nassi-Schneiderman Diagram showing rule for updating the state of a single automat in a field.

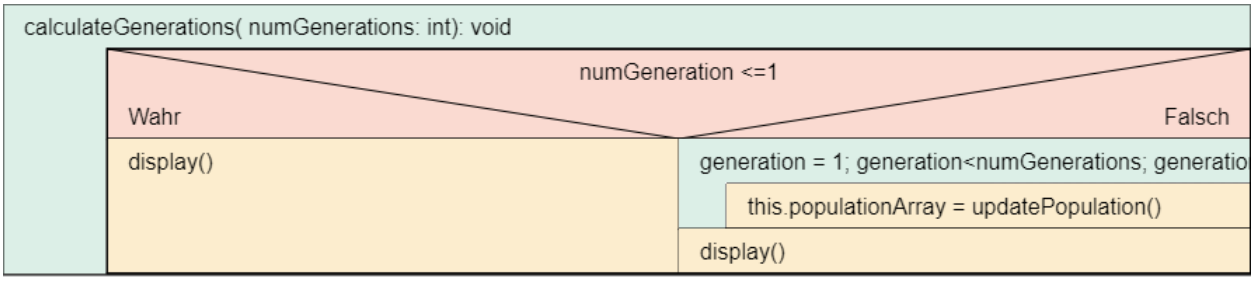

Figure 2: Nassi-Schneiderman Diagram showing method to update state of an automat for specified number of generations.

2.0 Nassi-Shneiderman Diagram for Calculation of the Whole Field

The whole field is implemented in a C# class called "**PopulationBuilder**", and it consists of a constructor and different methods used to calculate the whole fields. These methods include:

- **The constructor**, which initializes a 2D array of zeros which implies the whole field is initially in a dead state. The Nassi-Schneiderman's diagram to illustrate this is shown in **Figure 3**.
- A method "**Live**" which is used to set the state of an automat as alive. This automat is given a value "**1**" to indicate live. The Nassi-Schneiderman's diagram to implement this is shown in **Figure 4**.
- A method "**NeighbourCount**" which is used to count the numbers of live neighbours around a cell automat. The Nassi-Schneiderman's diagram used to implement this is shown in **Figure 5**.
- A method "**UpdatePopulation**" which is used to update the state of the automat based on the specified rules. The Nassi-Schneiderman's diagram used to implement this is shown in **Figure 1**.
- A method "**CalculateGenerations**" which is used to calculate the state of the automats in the field for a specified number of generations/evolution. The Nassi-Schneiderman's diagram used to implement this is shown in **Figure 2**.
- A private method "**Print**" which is just used to print the resulting array to unity console to check if the populations are updating correctly as per specified rules. The Nassi-Schneiderman's diagram used to implement this is shown in **Figure 6**.

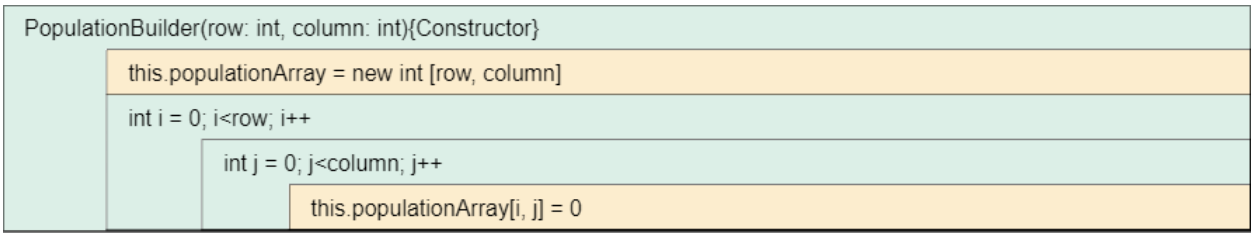

Figure 3: Nassi-Schneiderman Diagram of the Constructor used to initialize with field with zeros meaning dead state.

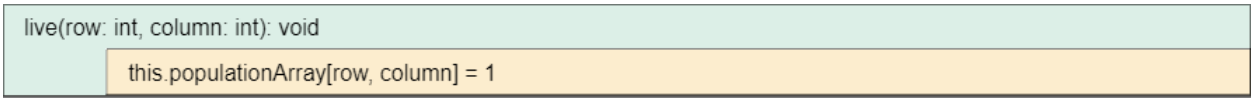

Figure 4: Nassi-Schneiderman Diagram of method used to set the state of an automat as live with number 1.

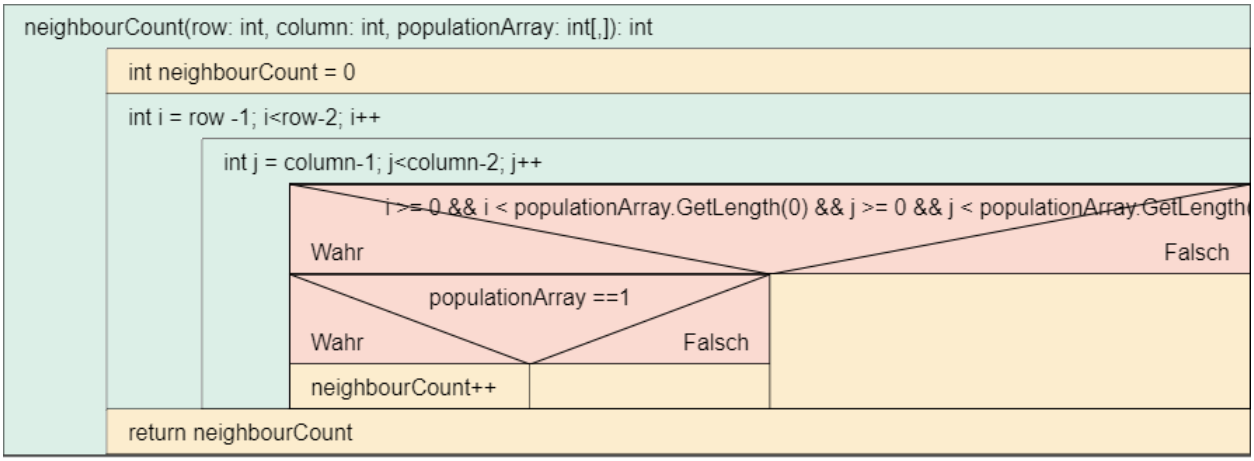

Figure 5: Nassi-Schneiderman Diagram of method used to count the number of live neighbour around an automat.

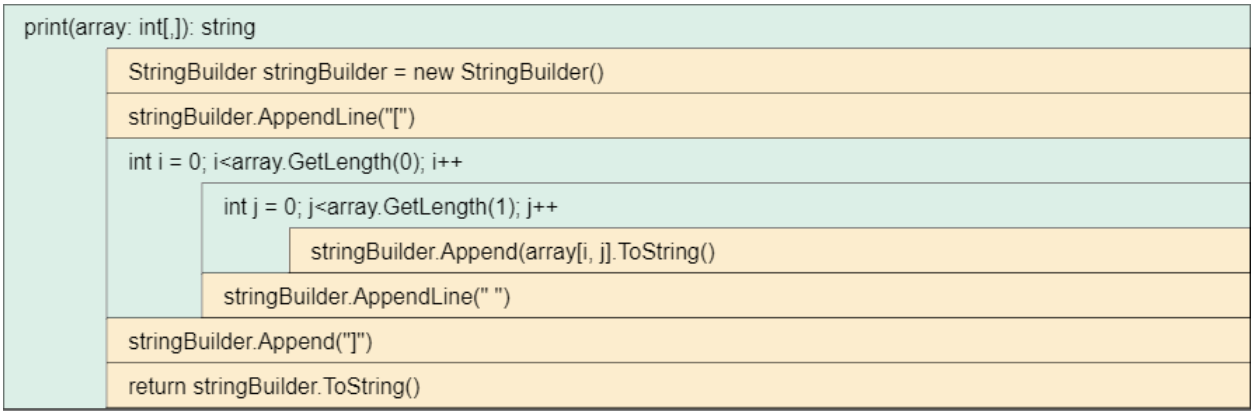

Figure 6: Nassi-Schneiderman Diagram of a private method used to print result of the field to Unity Console

3.0 Nassi-Schneiderman Diagrams used for Visualization of the Field

A C# rectangle array class called "**RectangleArrayBuilder**" was created for visualization of the state of the field as a rectangle. An object instance of the class was created in the "**PopulationBuilder**" C# class to display the rectangles and state in Unity Scene. This method is called "Display" which the Nassi-Schneiderman's diagram used to implement is shown in **Figure 7.**

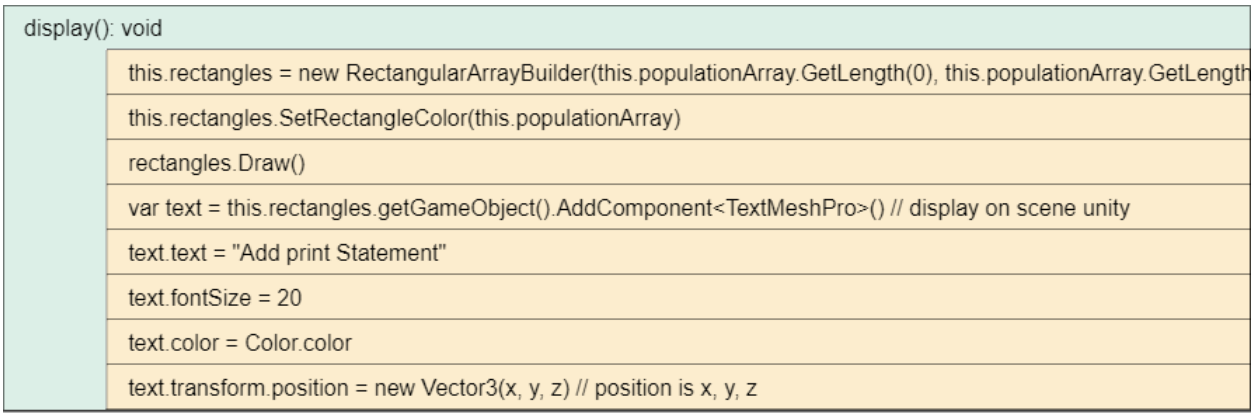

Figure 7:Nassi-Shneiderman Diagram of Display method in Population Builder Class used to Display Rectangles

From the "**RectangleArrayBuilder**" C# class for visualization, the methods implemented include:

- **A Constructor** used to initialize a 2D array of rectangles from existing rectangle classes. The Nassi-Schneiderman's diagram to implement this is shown in **Figure 8**.
- A method "**SetRectangleColor**" is created to set the colors of rectangles based on the state, with **0 for dead and 1 for live**. The Nassi-Schneiderman's diagram used to implement this is shown in **Figure 9**.
- A method "**Draw**" is created to draw the array of rectangles in Unity Console. The Nassi-Schneiderman's diagram used to implement this is shown in **Figure 10**.
- A method "**GetGameObject**" is created to return the gameObject which is used to display in Unity Scene. The Nassi-Schneiderman's diagram used to implement this is shown in **Figure 11**.

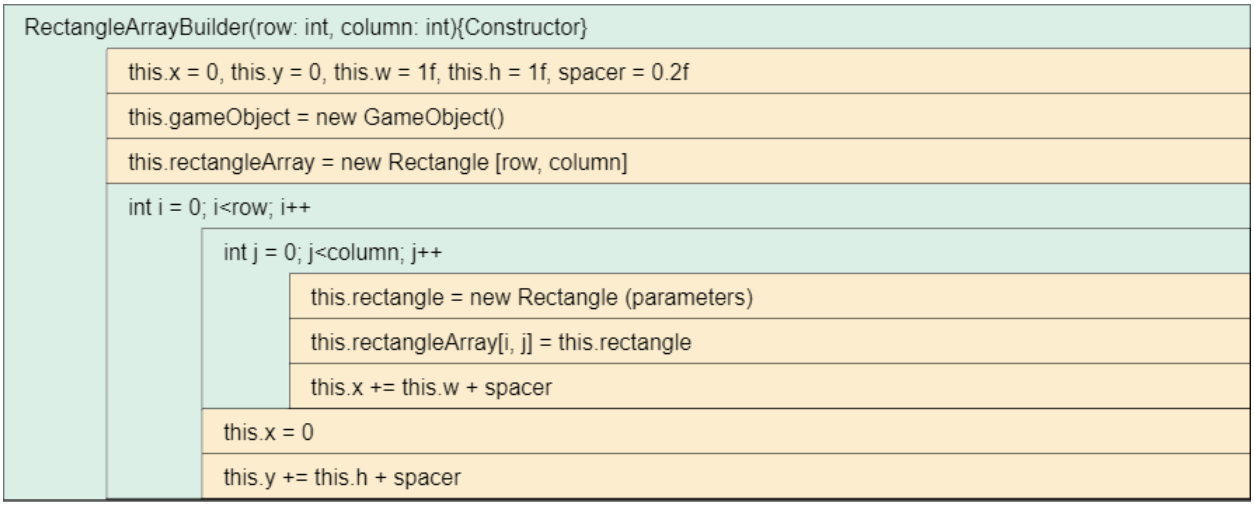

Figure 8: Nassi-Schneiderman's Diagram of the constructor to create 2D Array of Rectangles

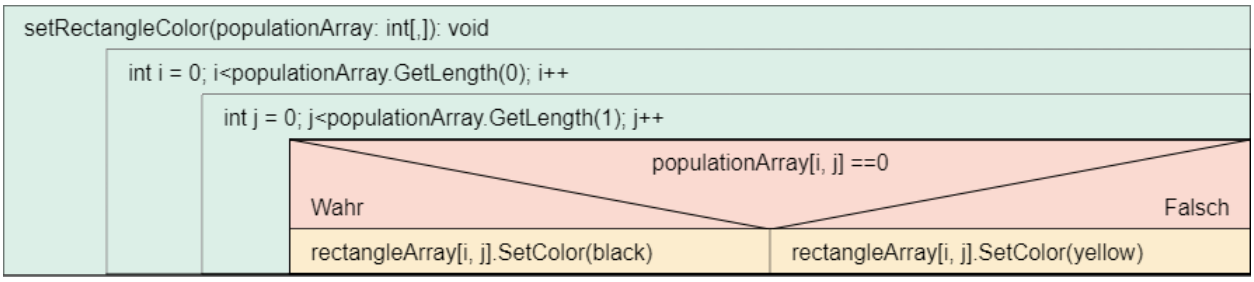

Figure 9: Nassi-Schneiderman's Diagram to set colors of Rectangles.

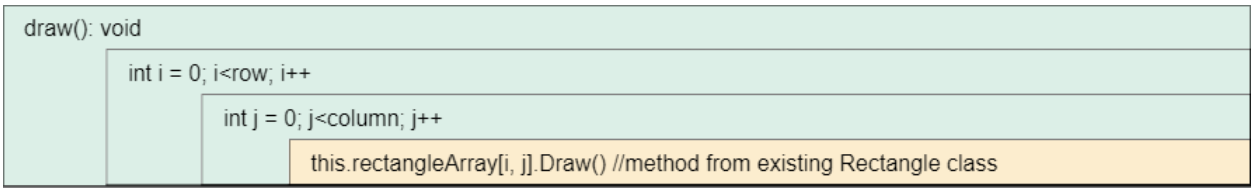

Figure 10: Nassi-Schneiderman's Diagram to draw Rectangles.

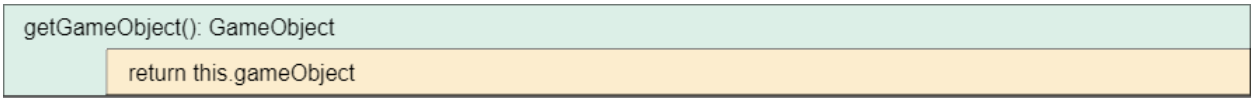

Figure 11: Nassi-Schneiderman's Diagram to get Game Object.

4.0 Class Diagrams for Implemented Classes

Two classes are used to implement the simple population problem and they are "**RectangleArrayBuilder**" for visualization and "**PopulationBuilder**" for calculation. Figure 12 and Figure 13 are the UML/Class Diagrams.

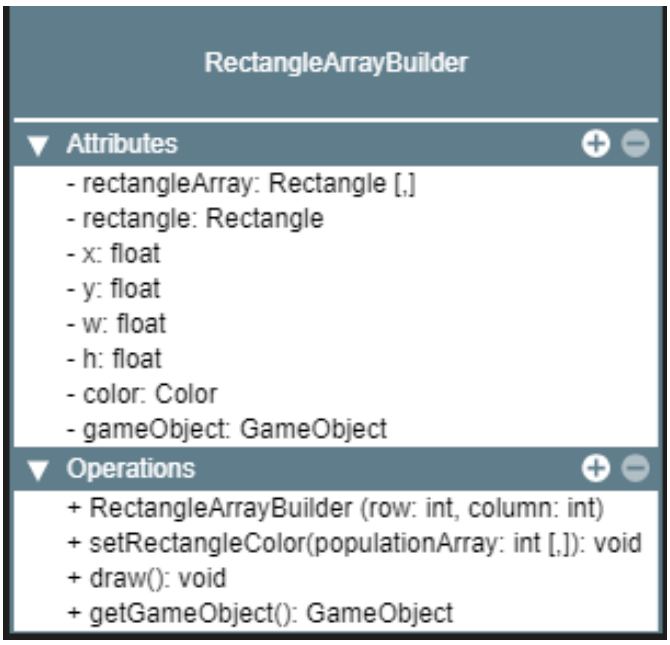

Figure 12: UML Diagram for RectangleArrayBuilder Class

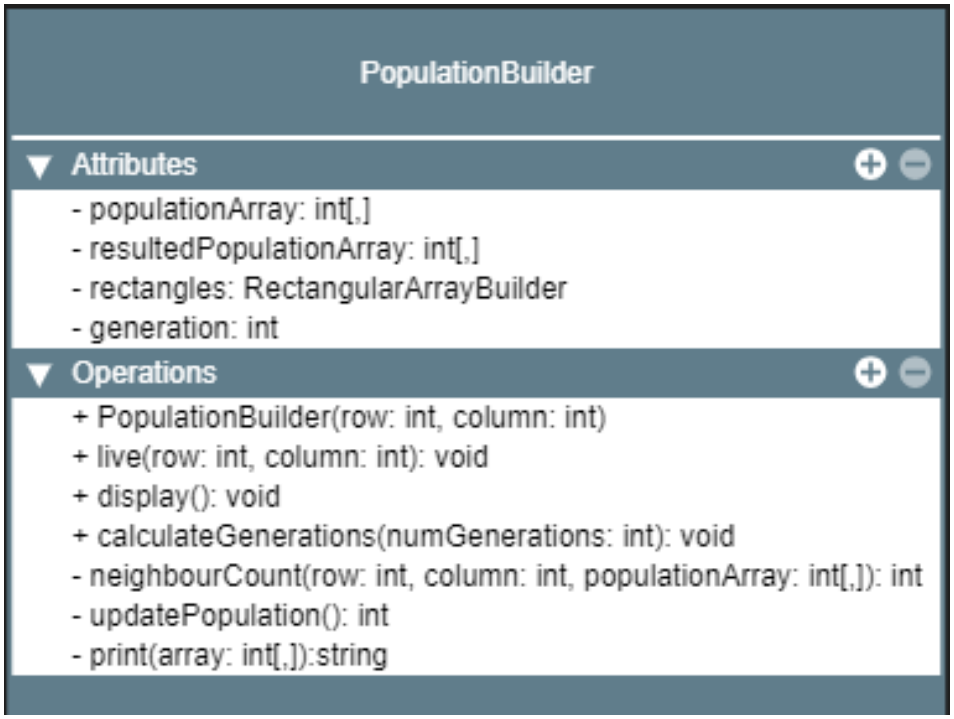

Figure 13: UML Diagram for Population Builder Class

5.0 Pattern 1 Evolution Results

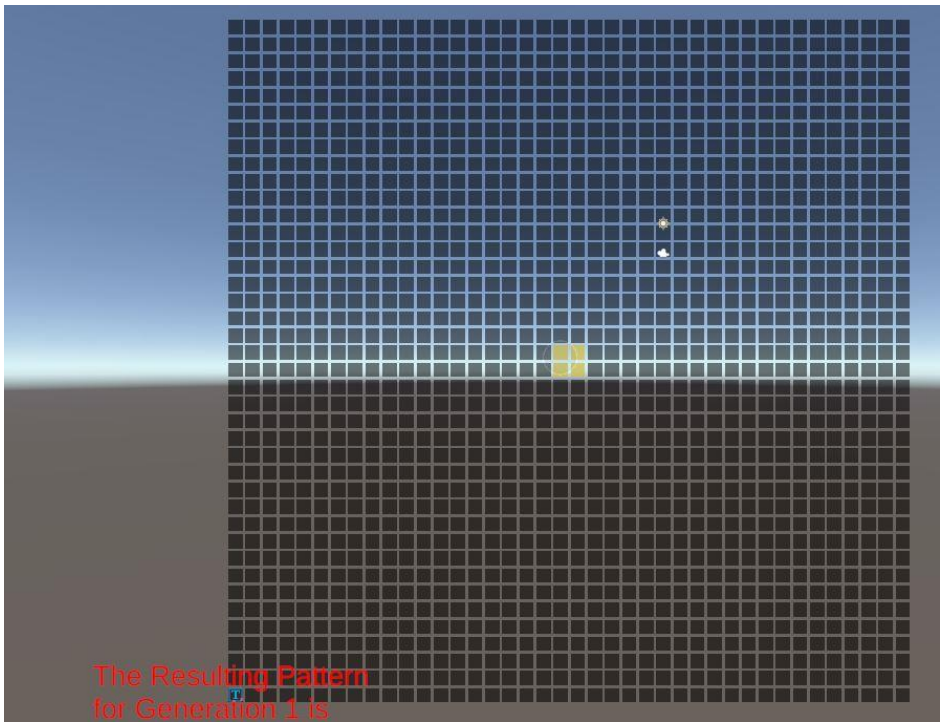

Figure 14: Plot Showing Pattern 1 result for 1st Generation (initial).

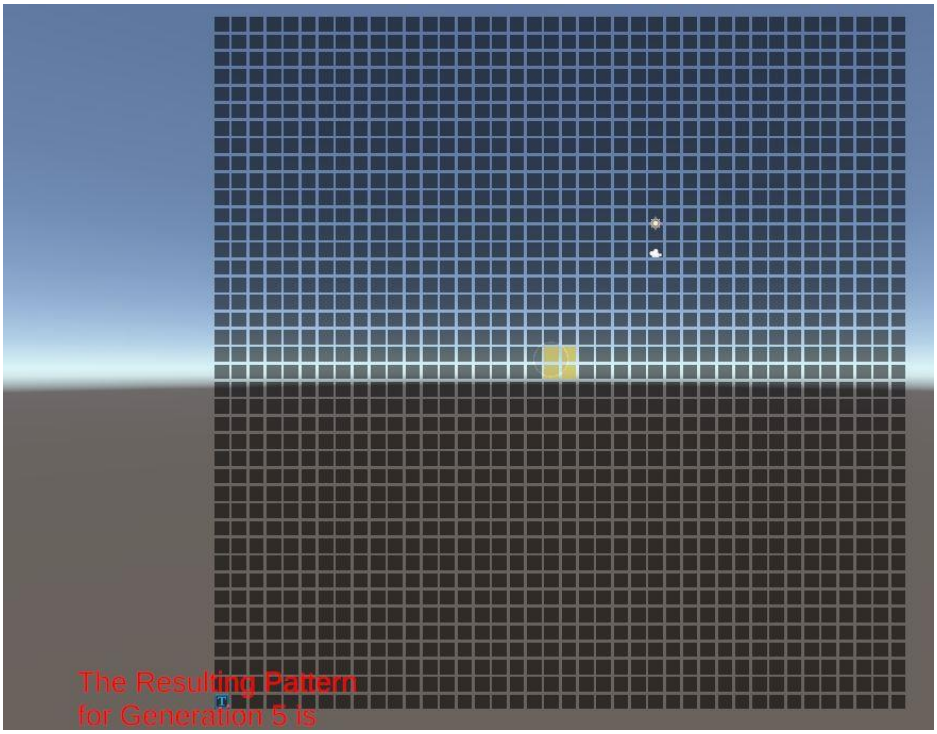

Figure 15: Plot Showing Pattern 1 result for 5th Generation.

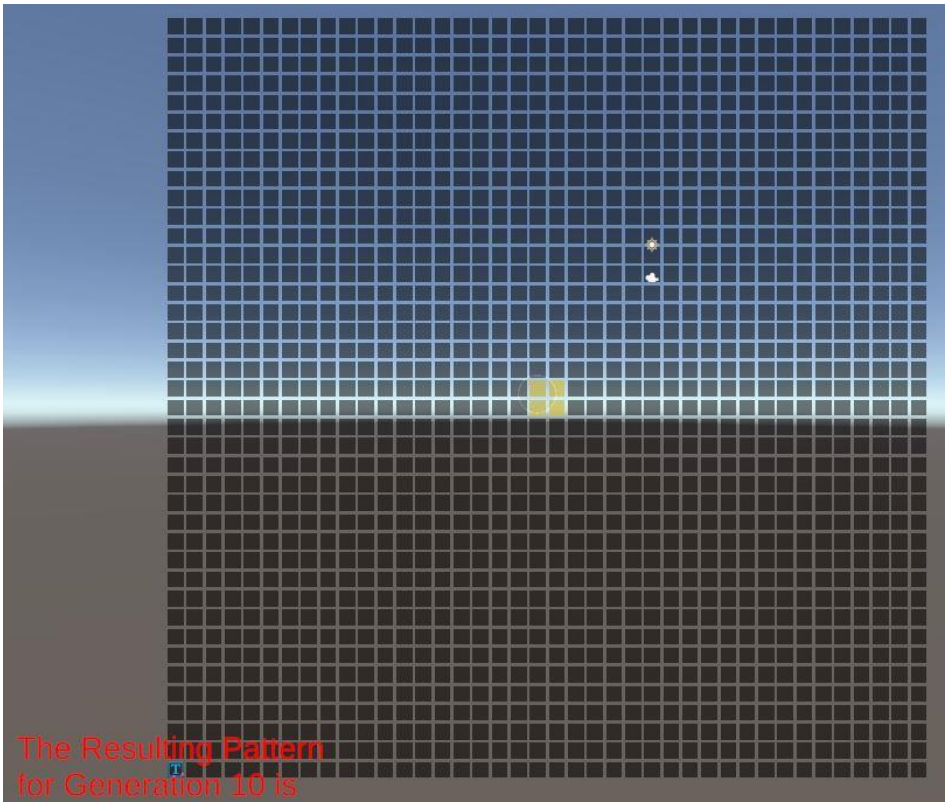

Figure 16: Plot Showing Pattern 1 result for 10th Generation.

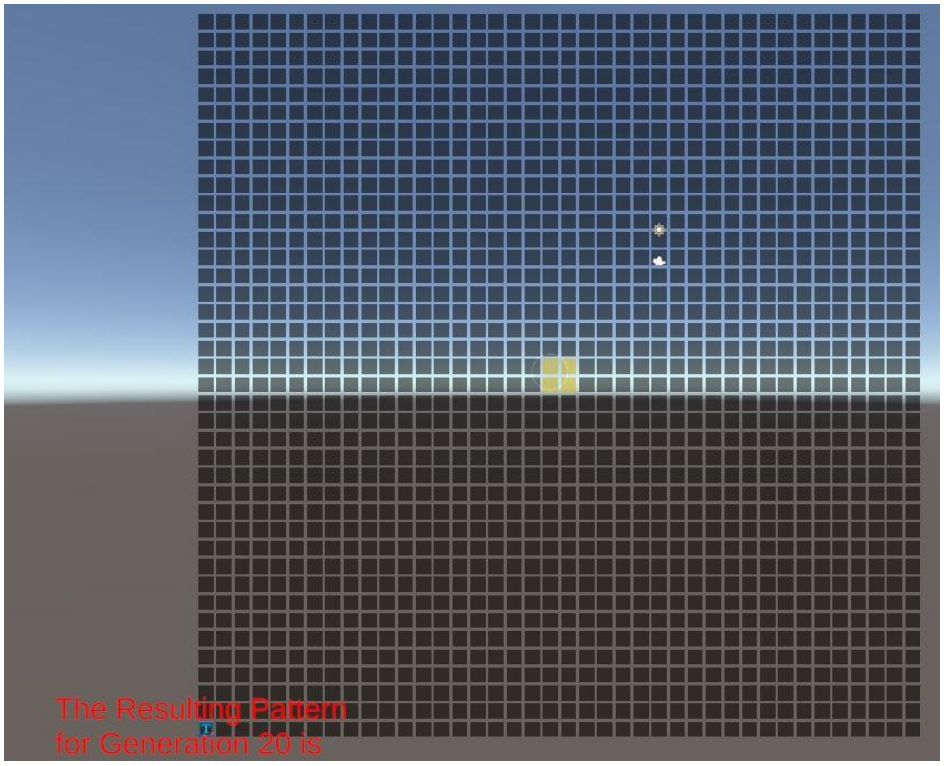

Figure 17: Plot Showing Pattern 1 result for 20th Generation.

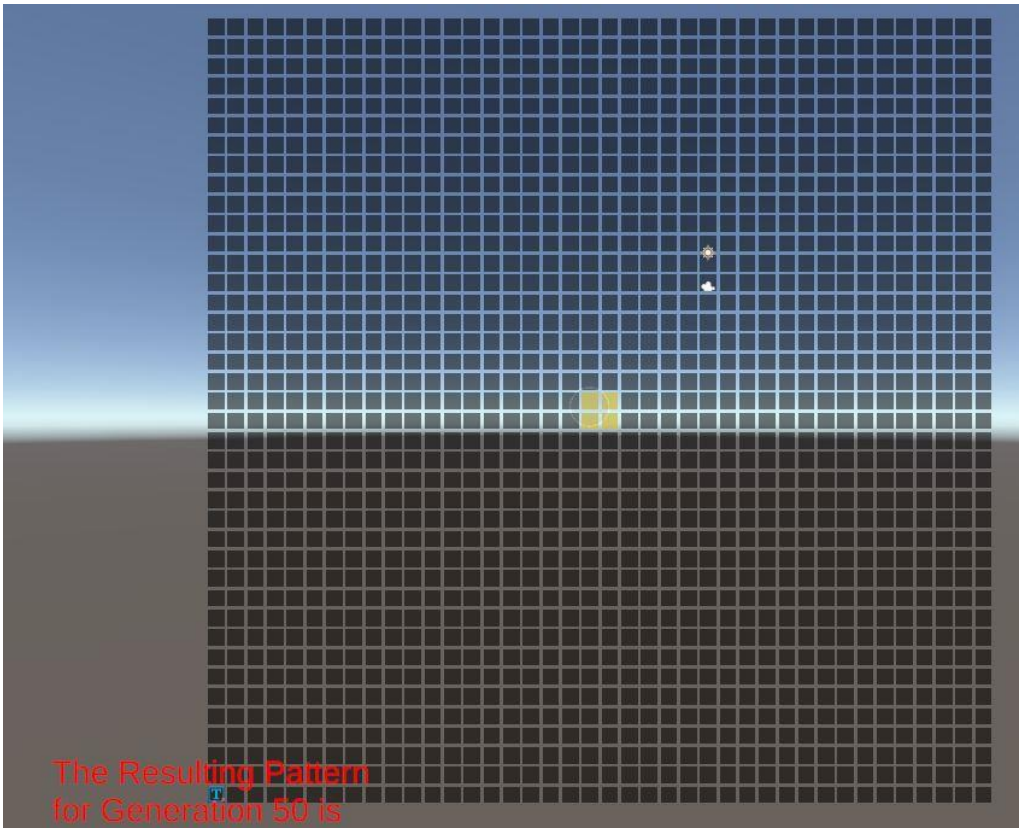

Figure 18: Plot Showing Pattern 1 result for 50th Generation.

6.0 Pattern 2 Evolution Results

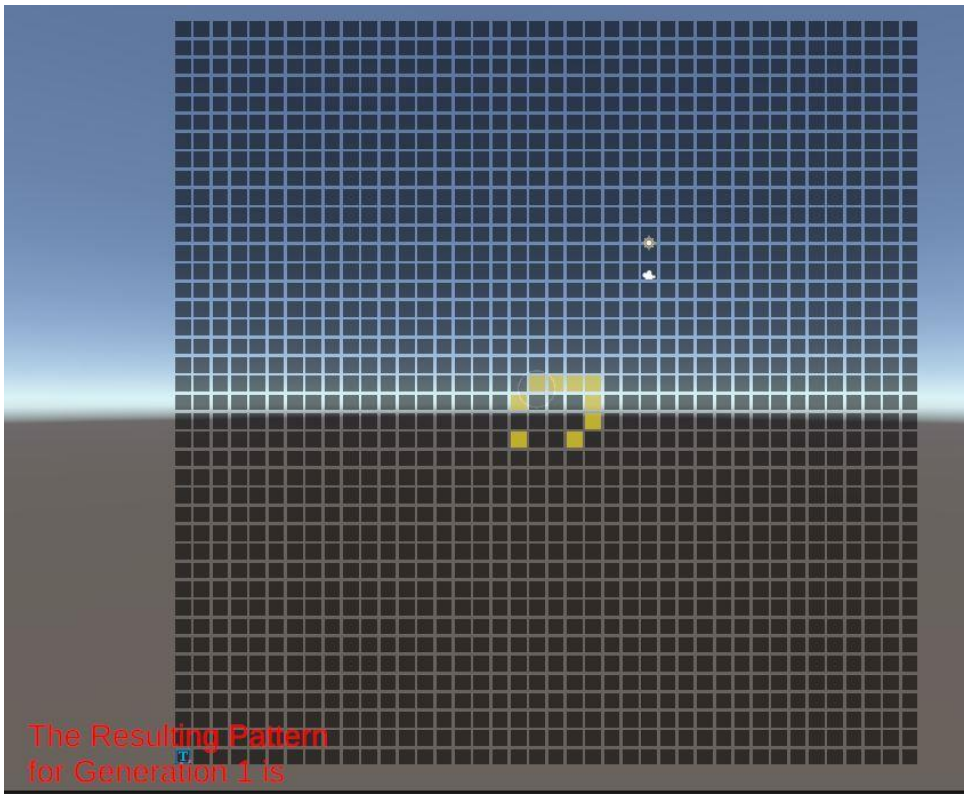

Figure 19: Plot Showing Pattern 2 result for 1st Generation (initial).

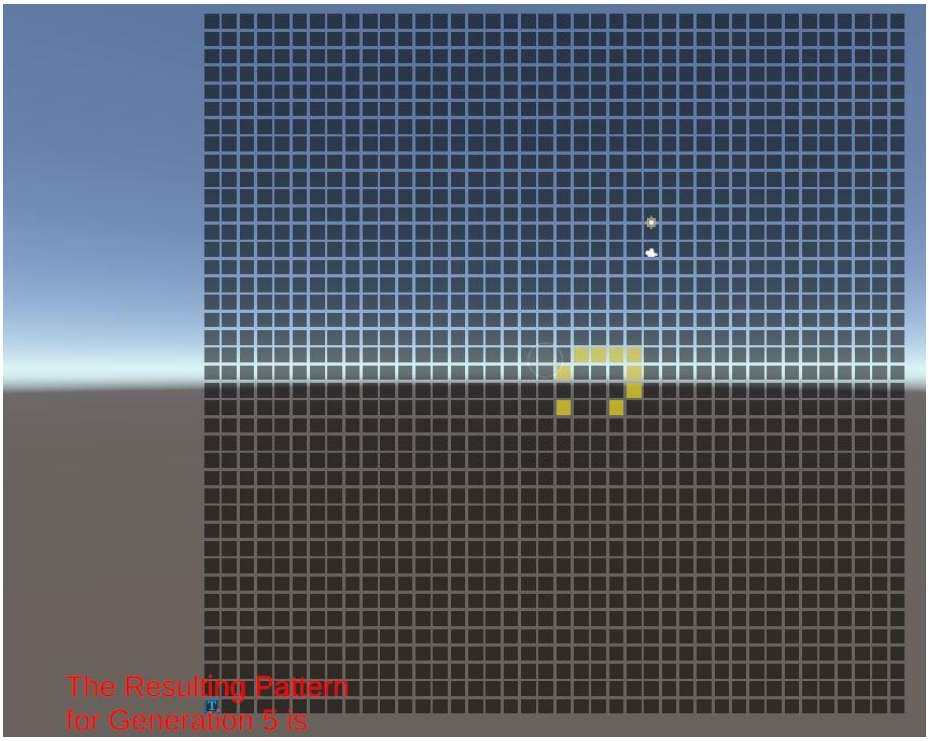

Figure 20: Plot Showing Pattern 2 result for 5th Generation.

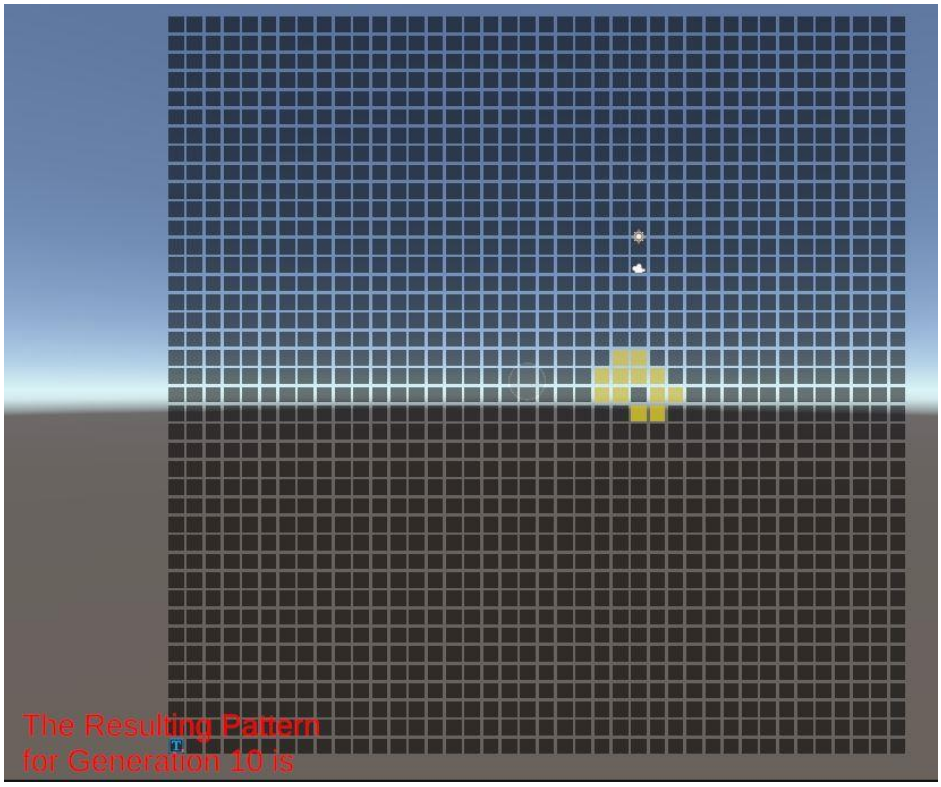

Figure 21: Plot Showing Pattern 2 result for 10th Generation.

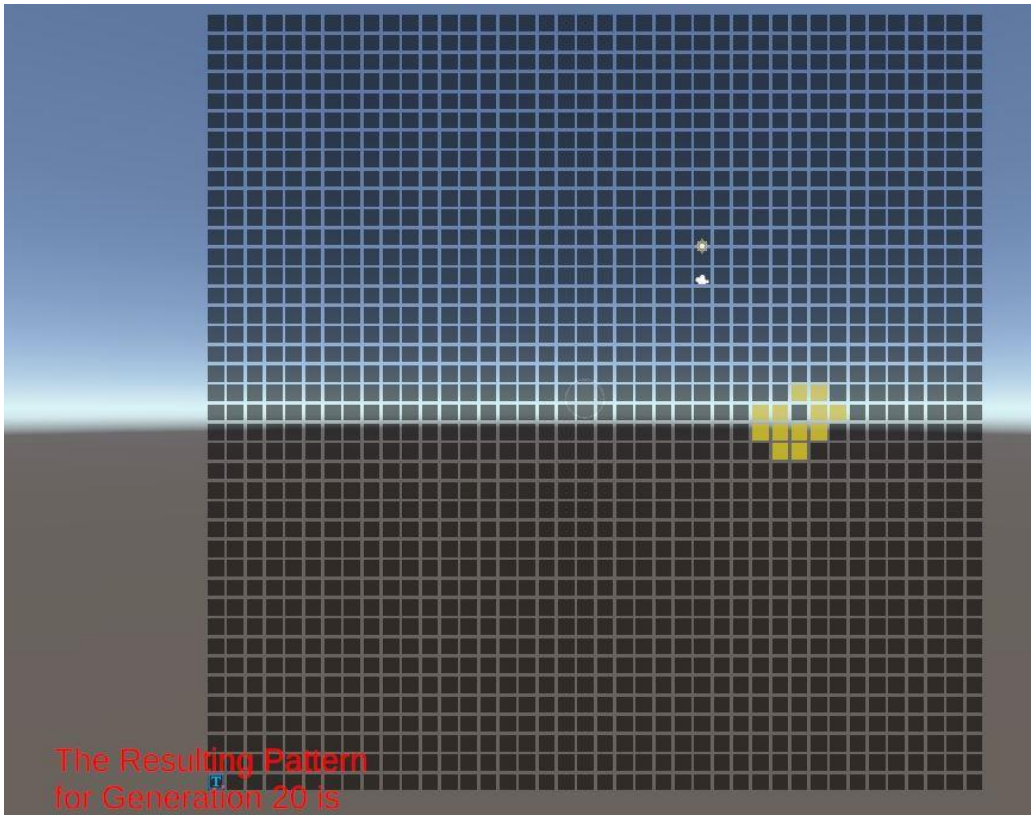

Figure 22: Plot Showing Pattern 2 result for 20th Generation.

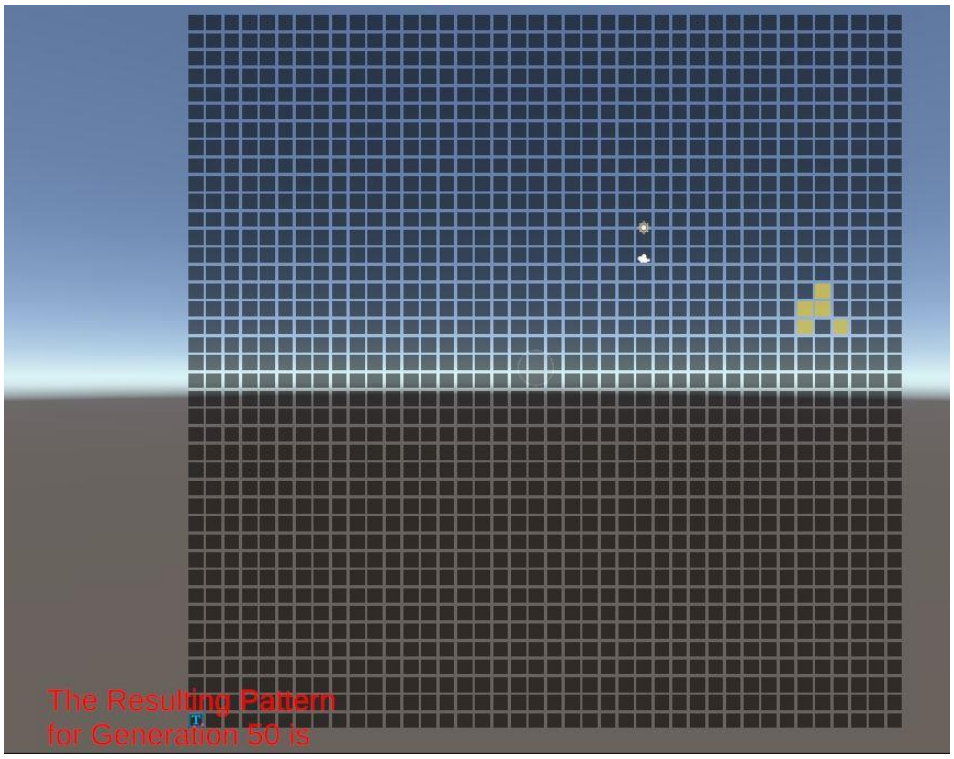

Figure 23: Plot Showing Pattern 2 result for 50th Generation.

7.0 Pattern 3 Evolution Results

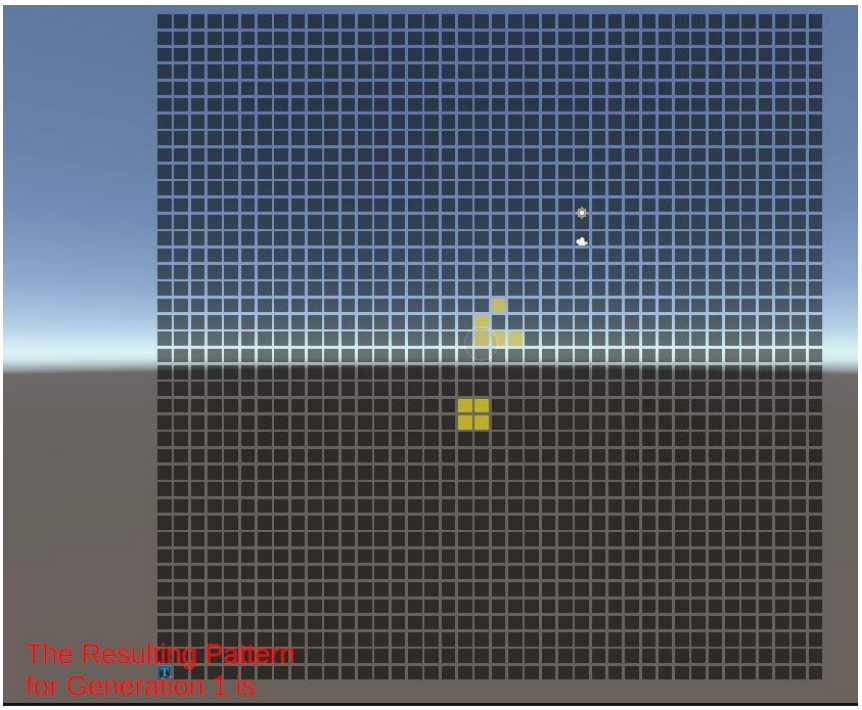

Figure 24: Plot Showing Pattern 3 result for 1st Generation (initial).

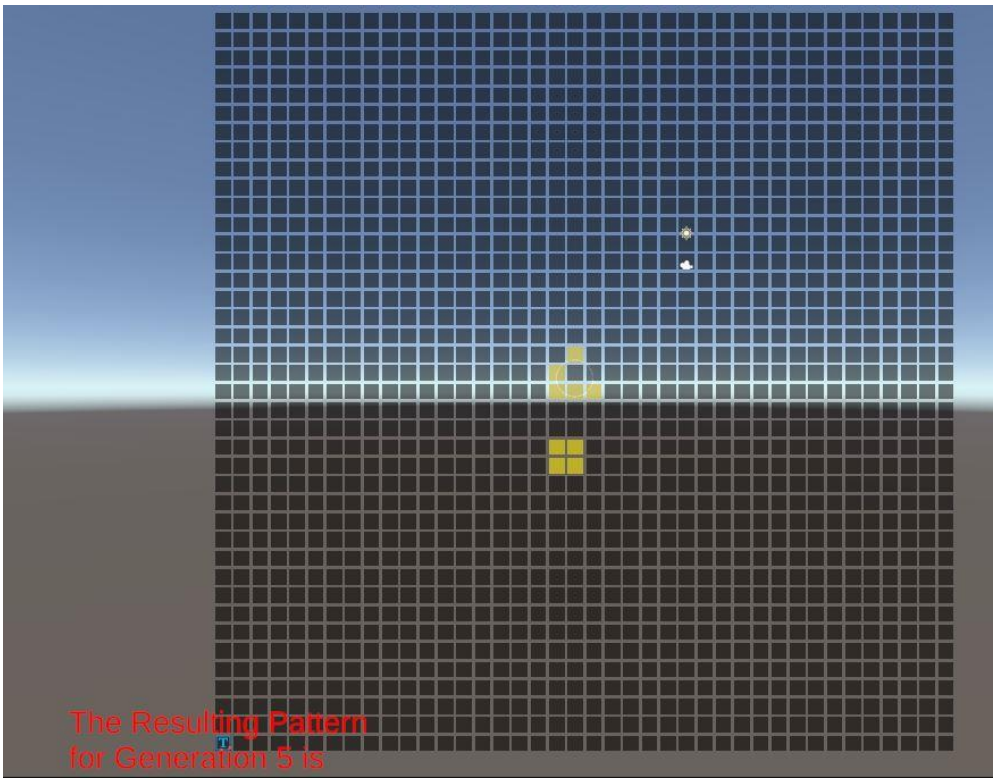

Figure 25: Plot Showing Pattern 3 result for 5th Generation.

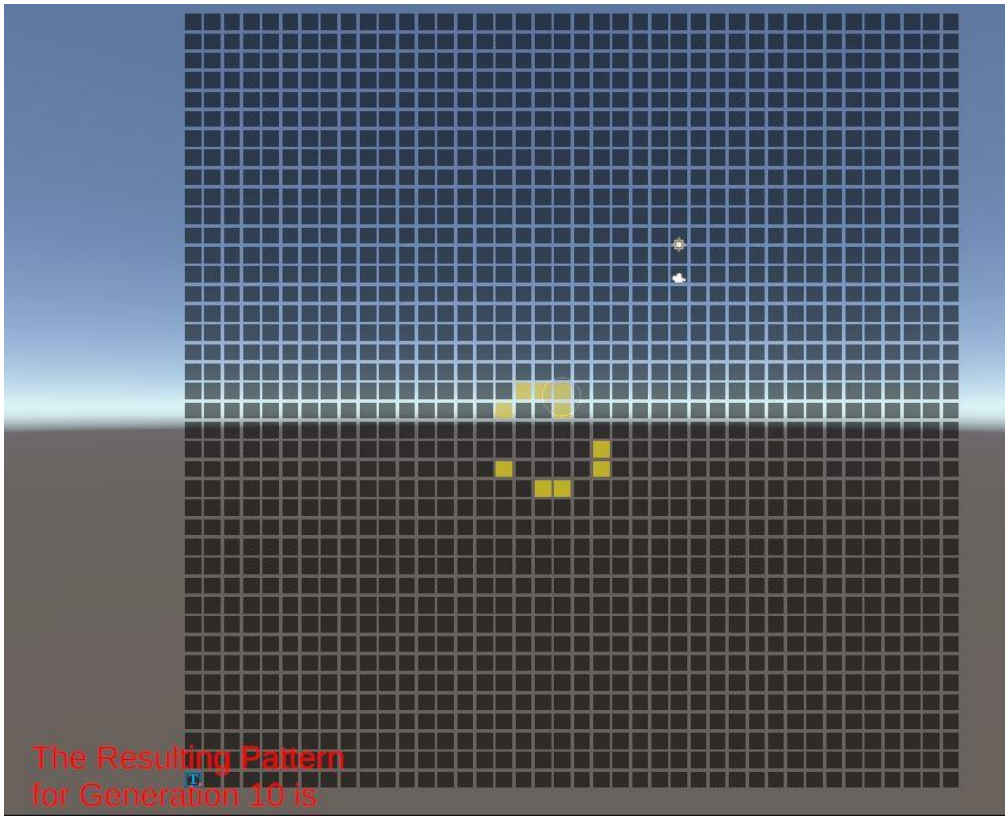

Figure 26: Plot Showing Pattern 3 result for 10th Generation.

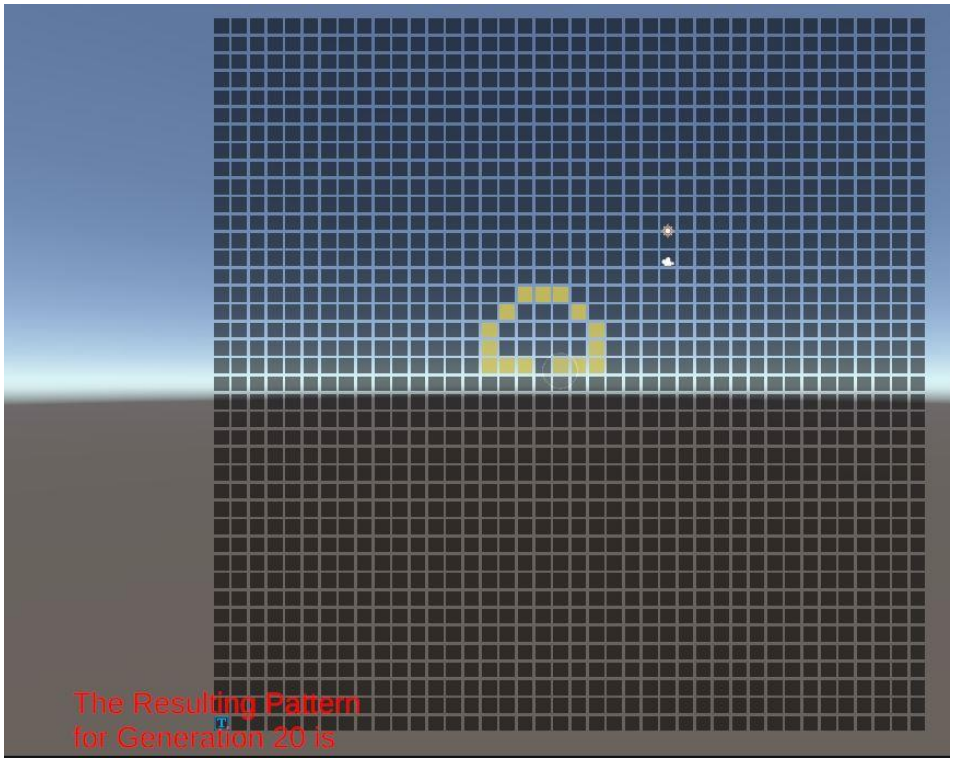

Figure 27: Plot Showing Pattern 3 result for 20th Generation.

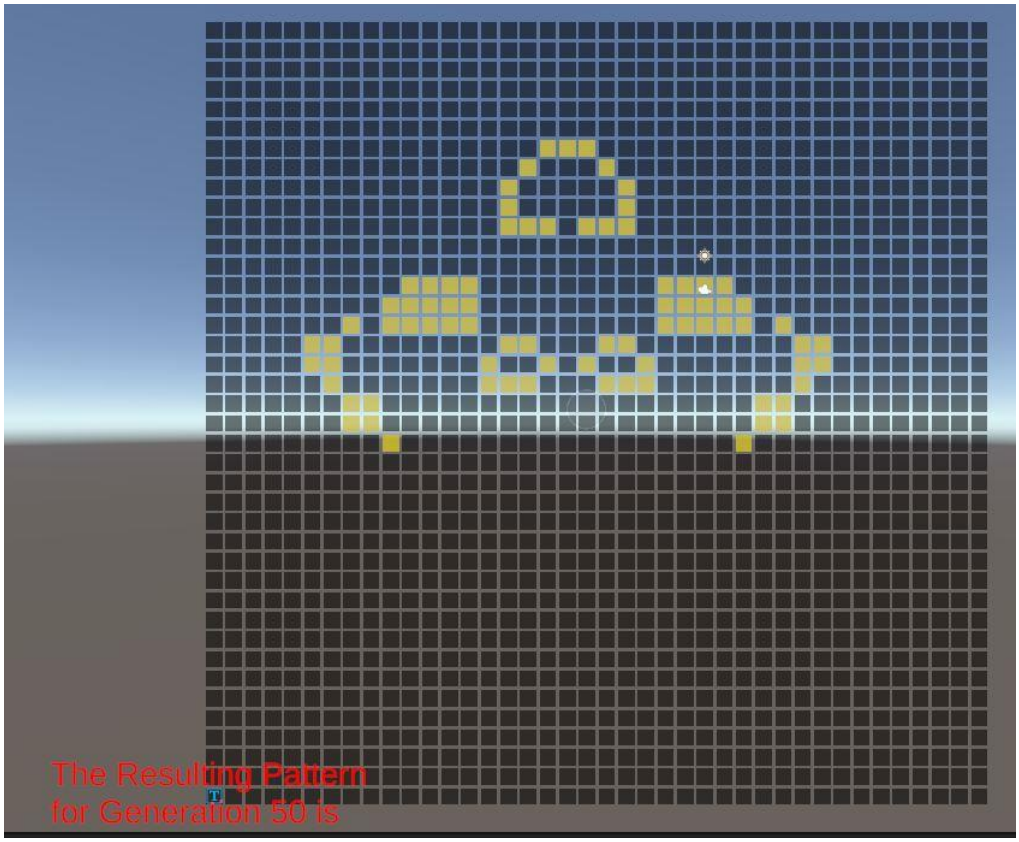

Figure 28: Plot Showing Pattern 3 result for 50th Generation.