**Statement of language used**

Processing-java which is an object oriented language and is used to show graphical images and animations

**Naming of all necessary libraries**

Processing

**System diagram**

**Full outline of each function**

|  |  |  |
| --- | --- | --- |
| Function Name | Parameters | What it does |
| setup | ------- | Sets up the background and the board. Also sets up the menu and GUI. |
| draw | ------- | Render the game and updates it |
| god | List of entities | Checks if there is a pattern, that causes an entity to be created or destroyed and change their colour to signify that. Then returns the new list |
| mousePressed | -------- | This senses when a mouse is pressed and checks if it has pressed any interactive objects |
| checkKeys | -------- | This checks if any keys have been pressed and, if any of the arrow keys have been pressed, move the screen location |
| readFromFile | Filename | This reads from a file and turns it into a 2d array that then can be used to place in later on. |
| clearBoard | -------- | This makes all the values of the board set to 0 (representing an empty cell location) |
| randomBoard | -------- | This randomises the Board |
| startGame\_random | -------- | This runs the randomBoard and starts the game |
| startGame\_gun | -------- | This starts the game with only a glider gun |
| startGame\_glider | -------- | This starts the game with only one glider |
| startGame\_file | -------- | This starts the game from a saved file |
| render | -------- | This controls all the rendering |
| renderBoard | -------- | This renders the board |
| renderGUI | -------- | This renders the GUI |
| renderMenu | -------- | This renders the menu |
| saveGame | -------- | Saves the current board to a file |
|  |  |  |
|  |  |  |

**Pseudo Code or Flowcharts for each function**

**setup()**

Set canvas size to 1080x1920

Draw the background as white

Create cellList

Add 20 cells in random positions

**draw()**

draw the background

FOR each in cellList

Set colour to each.getColour()

Draw each

CALL each.Update()

IF the colour of each is red THEN

Delete each

ENDIF

ENDFORLOOP

Set cellList equal to the return of god(cellList)

**god(cellList)**

create an an empty list called cellCreationList

FOR each in cellList

Check local entities to see if they create the pattern to create a cell

IF they do create a pattern THEN

Add co-ordinates to cellCreationList

ENDIF

Check local entities to see if they create the pattern to delete a cell

IF they do create a pattern THEN

Delete the cell

ENDIF

ENDFORLOOP

FOR each in cellCreationList

Create new cell with the co-ordinates from each

Add the new cell to the cellist

ENDFORLOOP

Set cellList equal to the return of sortList(cellList)

Return cellList

**sortList(cellList)**

FOR each in cellList

IF cellList[each].getX() < cellList[each – 1].getX() THEN

Go down the list until the statement is for filled and insert cellList[each]

Set each to the given value

ENDIF

IF cellList[each].getY() < cellList[each – 1].getY() THEN

Go down the list until the statement is for filled and insert cellList[each]

Set each to the given value

ENDIF

ENDFORLOOP

Return cellList

**Cell.update()**

IF this.colour == Blue THEN

IF this.x is out of bounds or this.y is out of bounds THEN

Set this.colour to Red

ELSE

Set this.colour to Green

ENDIF

ENDIF

**Cell.getX()**

Return this.x

**Cell.getY()**

Return this.y

**Cell.getColour()**

Return this.colour

**Sketch of User Interface**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |