Written By:

# Group # 2

# Besufekad Debebe

# Jose Martinez

# Merhawi Gebre

Contribution

It is mostly a collaboration of work.

List of reading/materials

https://www.codeproject.com/Articles/199525/Drawing-nearly-perfect-D-line-segments-in-OpenGL

http://ogldev.atspace.co.uk/

Youtube

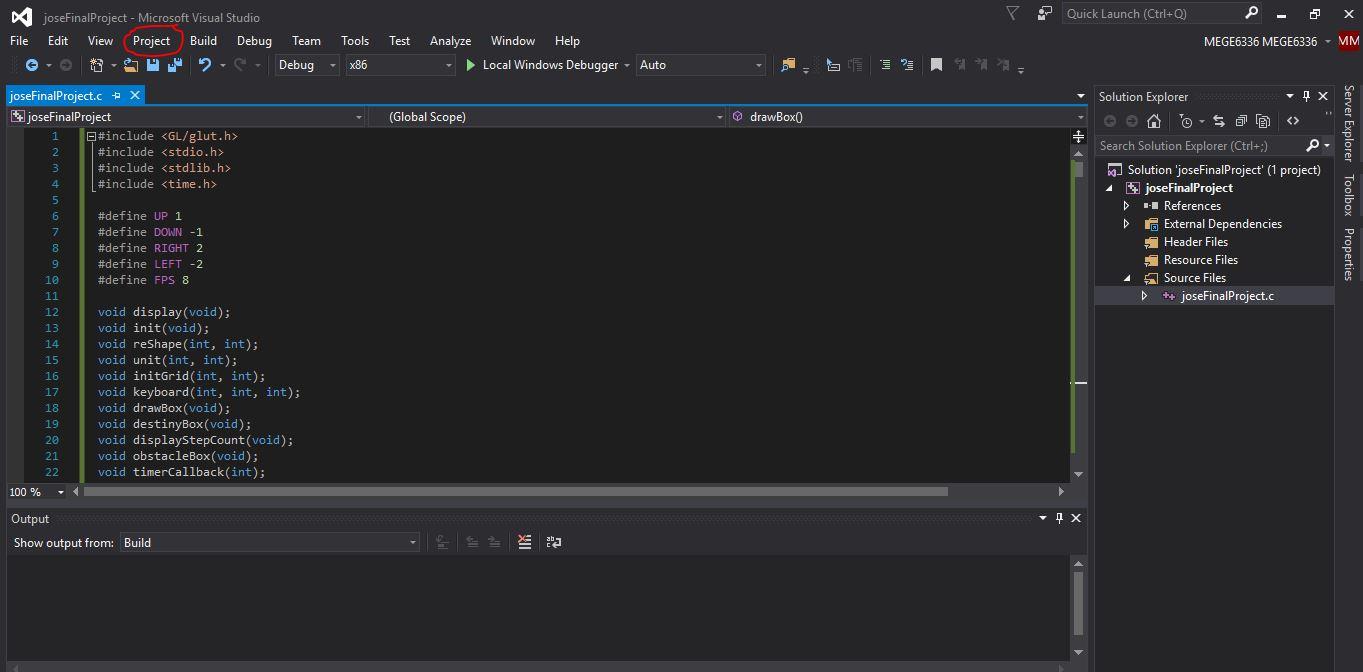
<https://learnopengl.com/#!Getting-started/Textures>

OpenGL Development Cookbook (Textbook)

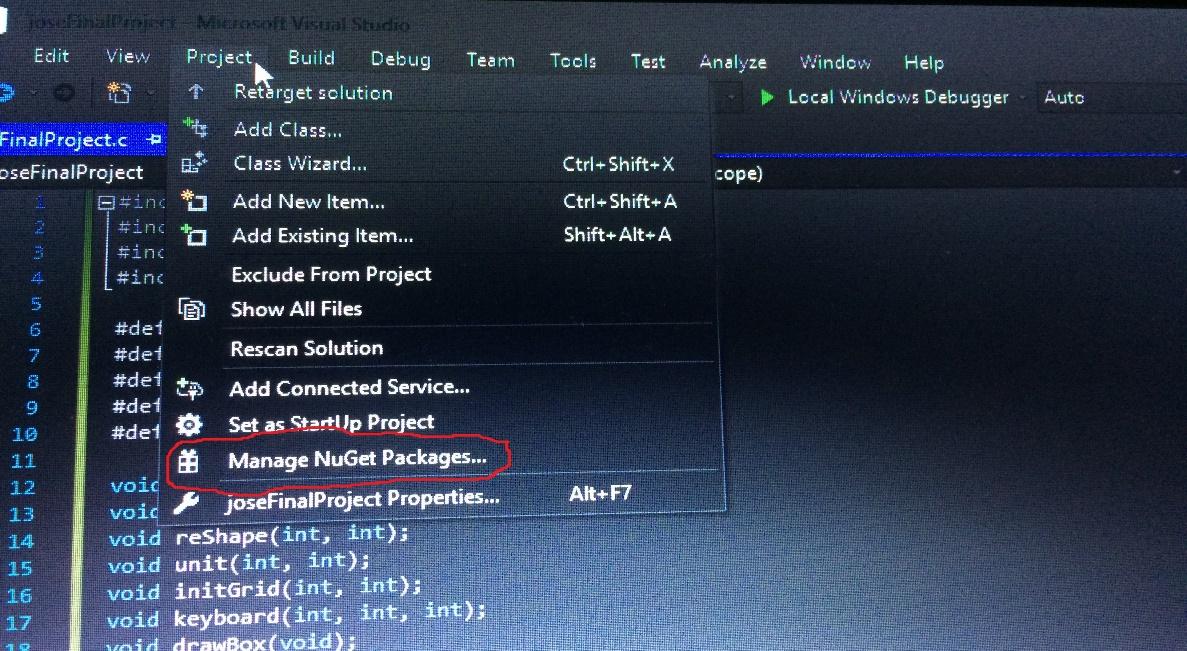
Compatibility

The OpenGl library version we used is 0.1.0.1. This library adds the various libraries to your project that are required to build OpenGl applications on both the x84 and x64 architectures. Includes FreeGLUT, GLFW, and GLEW.

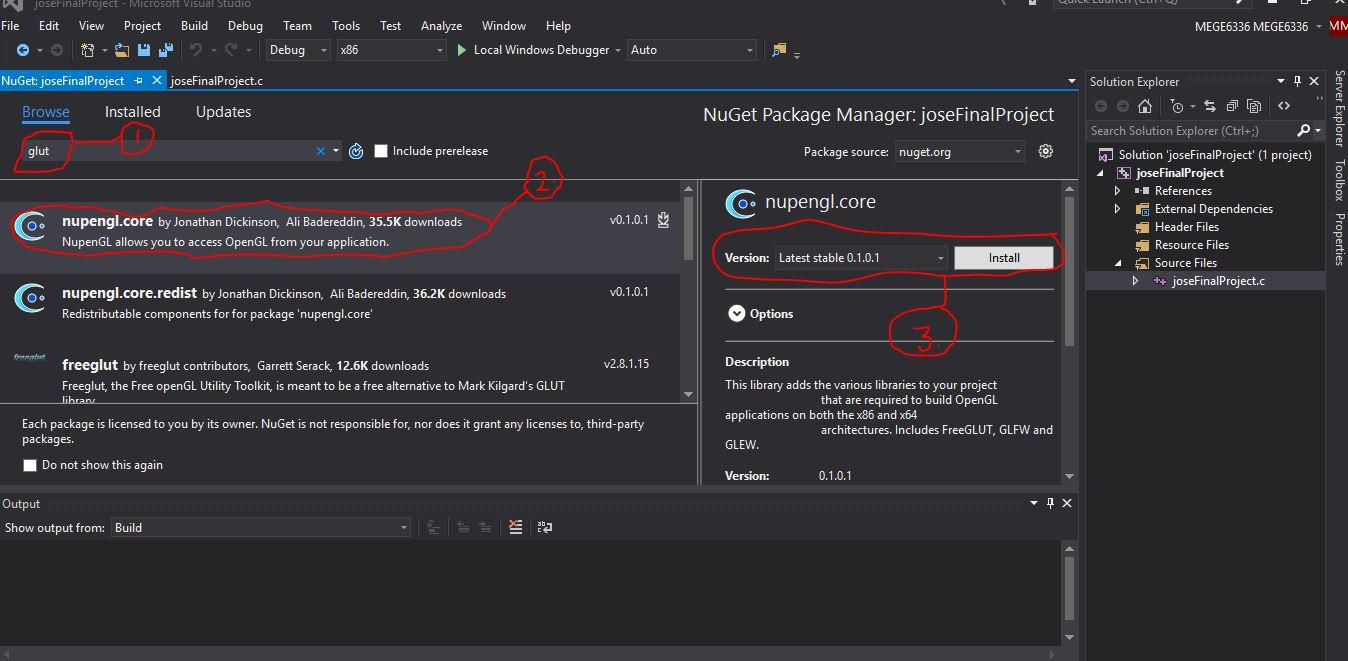
Setting up OpenGL on Visual Studio and downloading the Glut library



1st step is to open the project and go to Project tap.



2nd step is to go to Project/Manage NuGet Packages Tab.



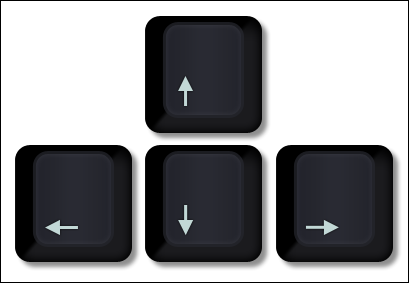
Final step is to go to the search bar and search for “glut”( which is numbered on the picture as number 1). Then click the “nupengl.core” ”( which is numbered on the picture as number 2). Finally click the “install” button ”( which is numbered on the picture as number 3).

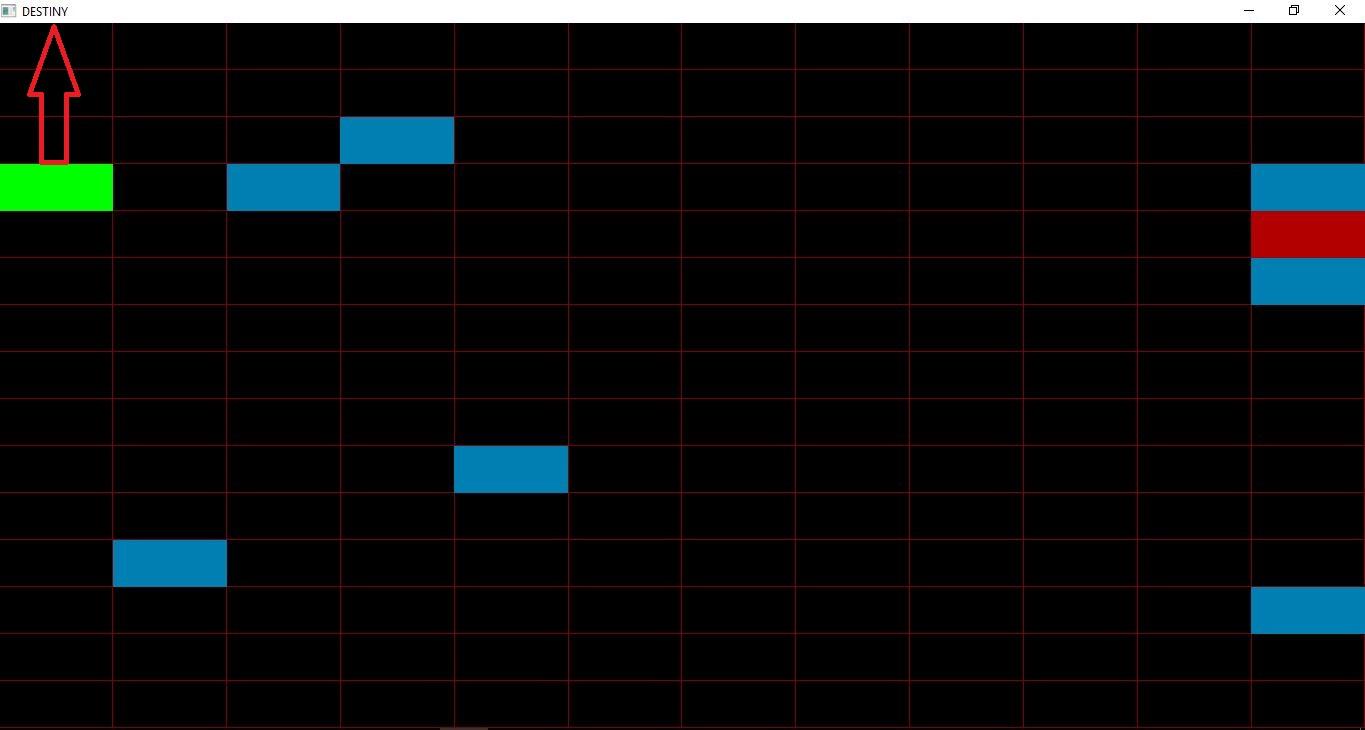
Improvements to the game

Anyone interested in the game may include to add the option to play all the puzzles subsequently instead of re-running the program every time each puzzle is solved. Adding textures to the blocks is another option to add.

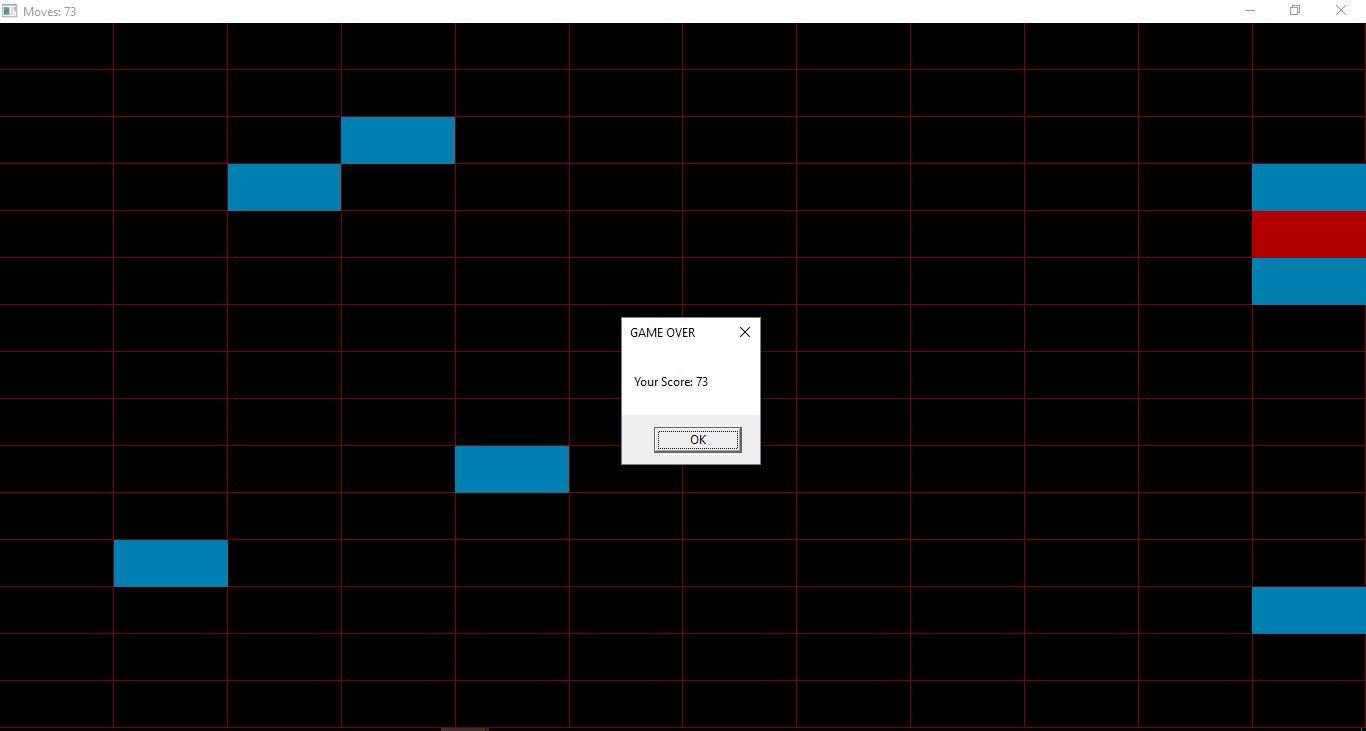
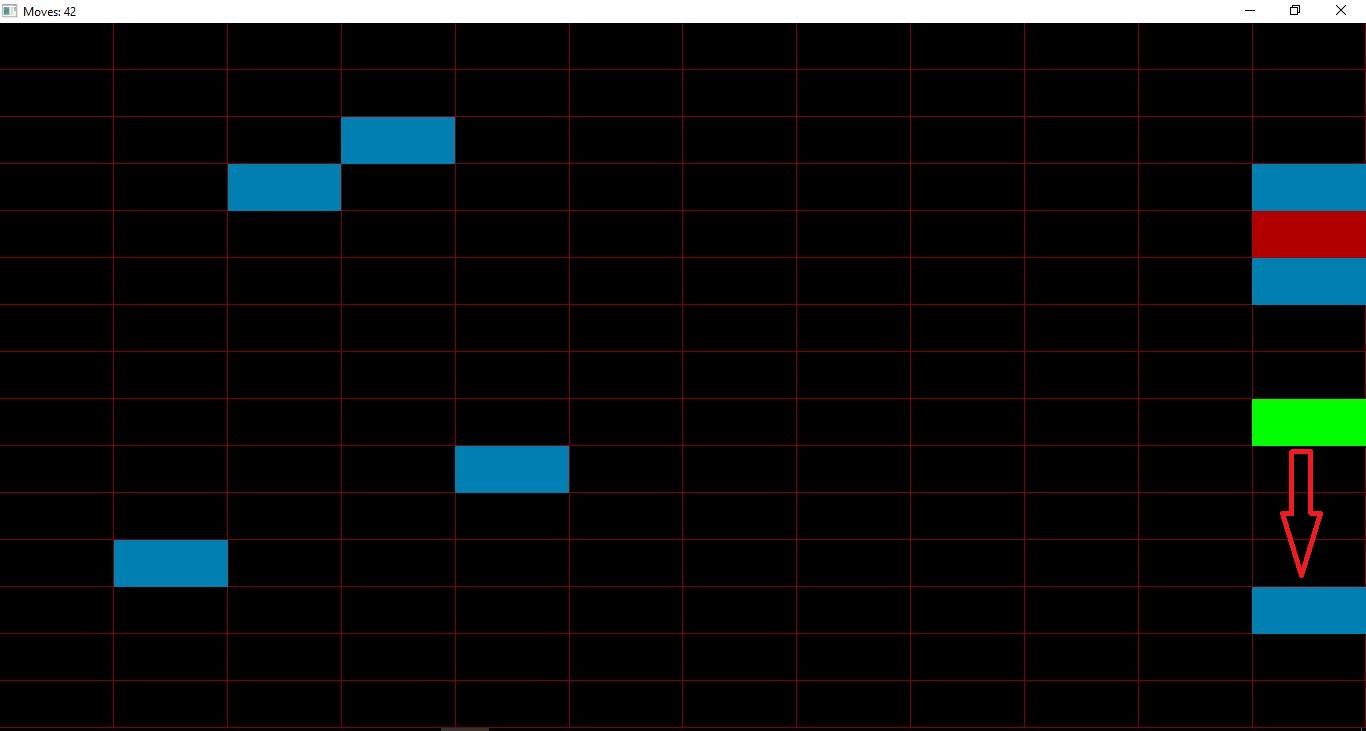
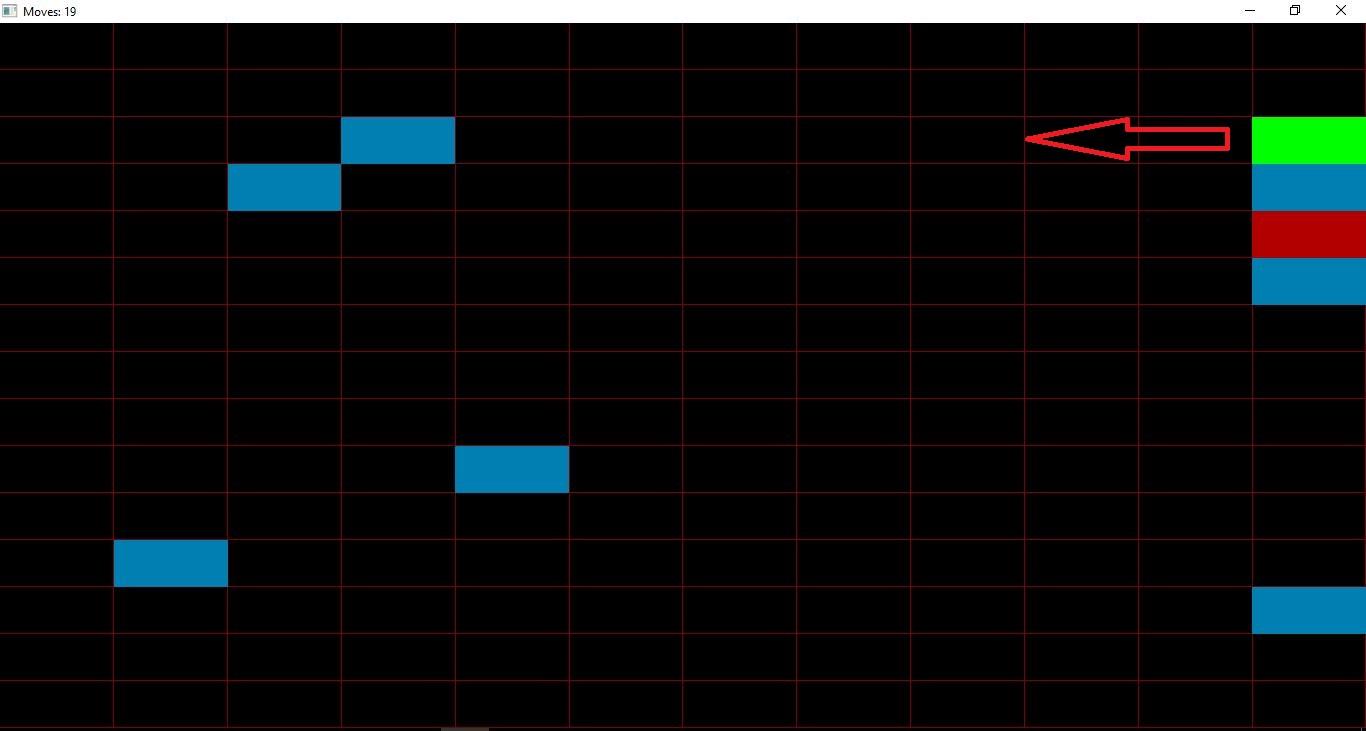
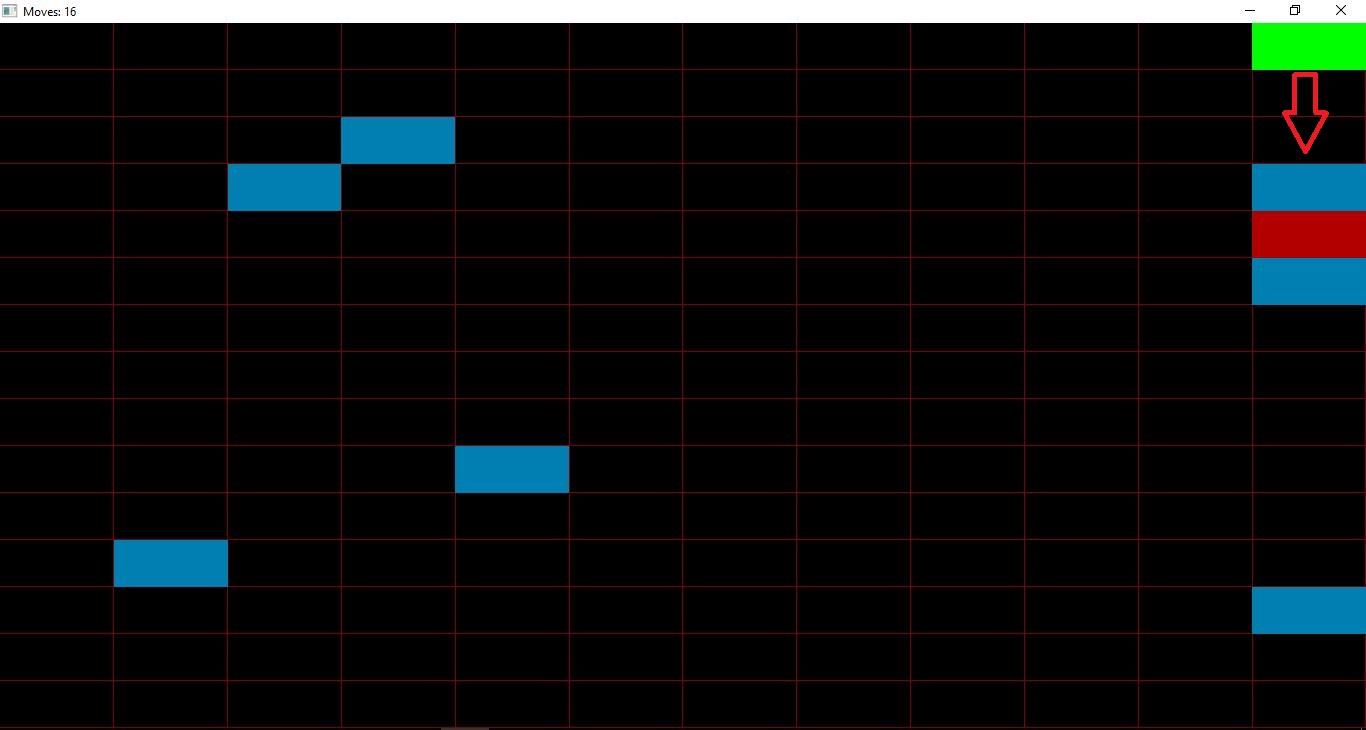
Demo of the Game with Pictures and Instructions

To play the game use the four buttons UP, DOWN, RIGHT, and LEFT arrows on the keyboard to move the GREEN BLOCK to the DESTINATION which is the RED BLOCK by using the BLUE BLOCKS or the BORDER on the SCREEN as a DIRECTION CHANGER. Once you chose a direction you cannot change the direction while the BLUE BLOCK is moving.









Issued we faced when creating the game

Handling the collision of the blocks took us time to come up with the solution. Other than that everything else was not that hard in terms of coming up with the solution to solve the problem at each phase of the game.

Implementation and logic

1. Creating and displaying the board
2. Creating the objects
   1. Player box
   2. Destiny box
   3. Obstacles
3. Moving the Player box
4. Hitboxes
5. Creating solvable puzzles
6. Displaying random puzzle each time code is ran
7. **Creating and displaying the board**

Our game starts with creating the setting up of the display window first. We initialized the window by using the glutInitWindowSize(), with a rectangular dimension of 750 \* 450. Afterwards, we created the grid individually, drawing a single unit box and duplicating it to 12 columns and 15 rows.

1. **Creating the objects**

The objects in our game includes, the Start/Player, the Obstacles, and the Destiny.

* 1. **Player box**

The Player/Start is the green box, whose position is dynamic. Its position can be manipulated by the user using the arrow keys from the keyboard. It is constructed as the same way as the unit grid is built. It moves only in one of the four directions, left, right, up and down. Once it is directed to move in a certain direction, it increments the coordinate by one in the direction it moves.

* 1. **Destiny box**

The destiny box is the one which is red. It is constructed the same way as Start/Player is constructed. Unlike the Start its position is static. Destiny has two functions in the game. First, the whole game is logic is built on it. This means, the objective of the game is trying to get in the destiny. Second, it ends the game once it gets hit.

* 1. **Obstacles**

The obstacles are the one which are located in a blue box. Their construction is the same as both destiny and start. Additionally, their position is static. Functionally, they have two purposes in the game. First, they are the path or the route to change a direction for the player in order to get into the destiny. In brief, they are direction changers. Second, they make the game interesting for the player. Because, they put the user/gamer into puzzle. Not all of the obstacles are helpful to play the game, the can also trick the player without helping them.

1. **Moving the Player box.**

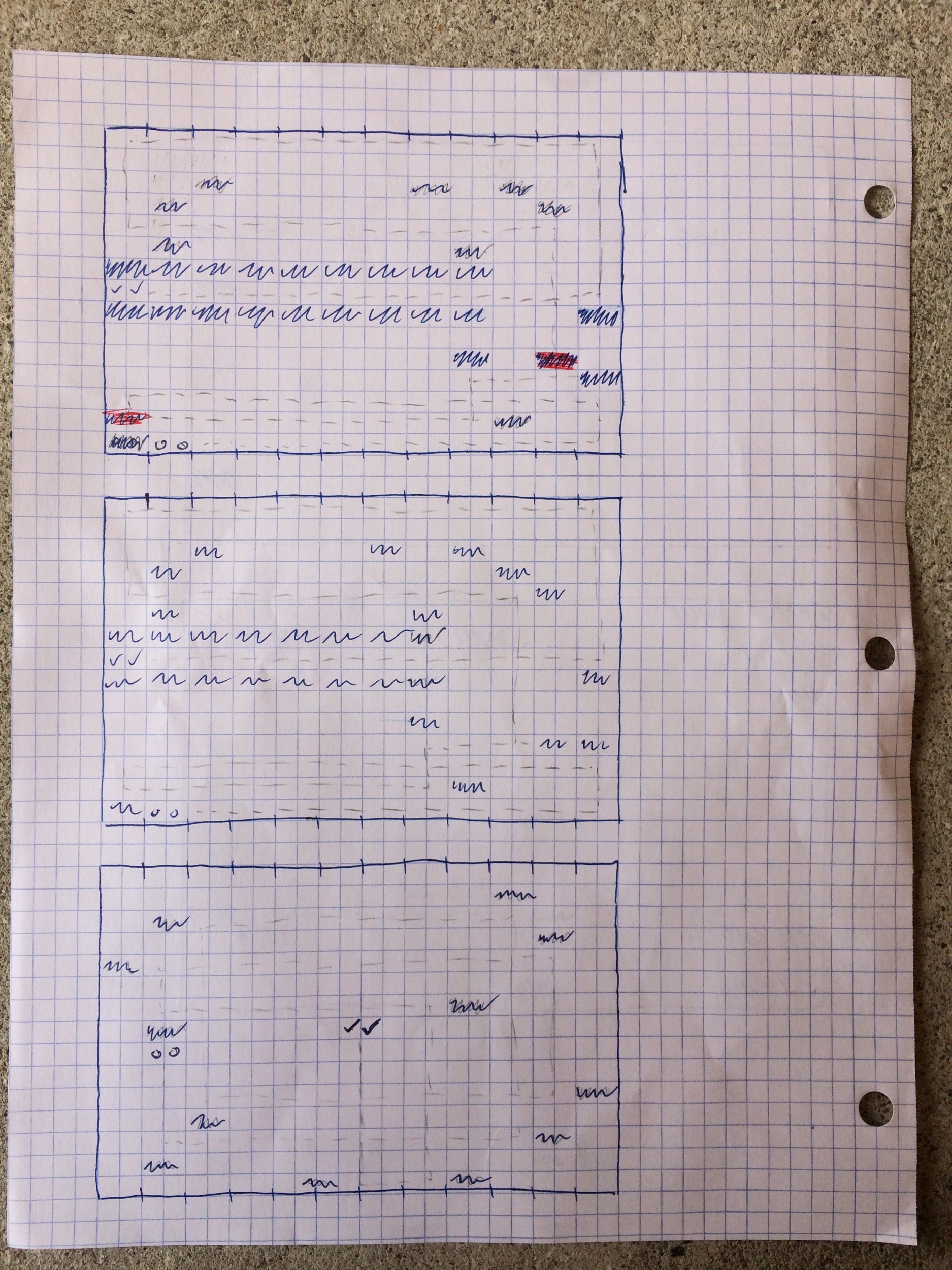
Arrow keys change one of two variables. One variable is for left/right movement, the other is for up/down movement. The current position values is incremented by the value of the variables. The Player box can only move if both of these variables are 0, or in other words the box is stationary. When it detects hitbox collision, it sets the values of the variables to 0 and stops the movement. The box moves continuously until otherwise, like if it were on ice or a frictionless surface. GrowthUp being a value of 1 means upward movement, while a value of -1 means downward movement. GrowthRight having a value of 1 means rightward movement, while a value of -1 means leftward movement.

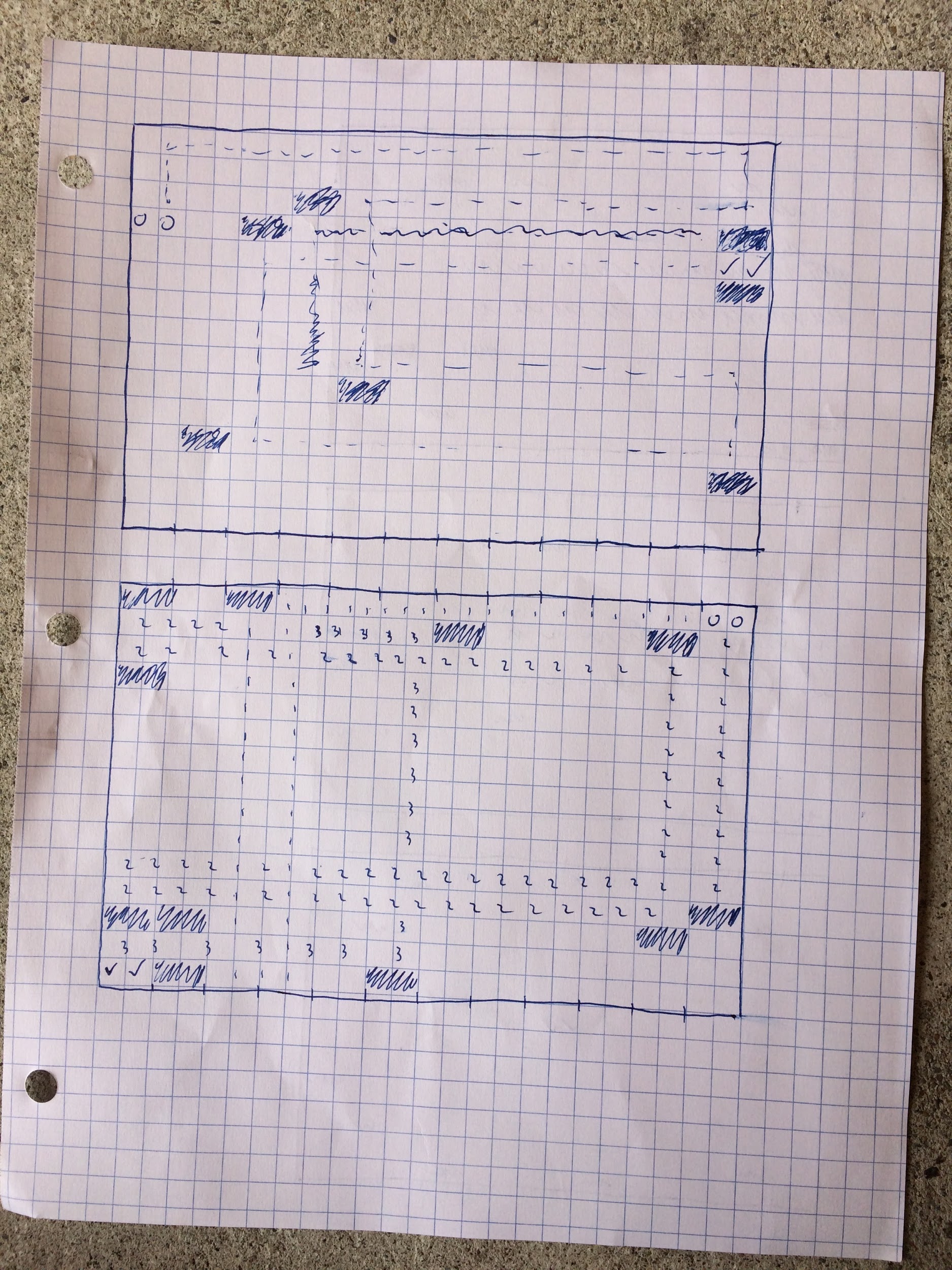
1. **Hitboxes on obstacles.**

the way we gave hitboxes to the obstacle boxes was a long list of if statements, this checks if the X and Y position values of the player box equals to the X and Y values of the different obstacles. Then based on the direction the Player box was moving, the Player box gets moved one spot before the obstacle box. Also if the Player box goes outside of the board displayed, it similarly gets set back. the operations happen fast enough that it looks like the Player box stopped a spot before a wall/obstacle instead of overlapping and being set back.

1. **Making solvable puzzles.**

The process to make a solvable puzzle was creating a grip of the same dimensions of the game board on a piece of graph paper. Then we picked a spot on the grid for the destiny box and the Player box. Then we worked backwards from what would be the final step of the puzzle and added obstacles to obstruct/lengthen the path to this final step. Then it’s simply changing the appropriate variables in the code that correspond to the different game objects.





1. **Random puzzle every time code is ran.**

We wanted to add more complexity to the game beside just one puzzle. The solution we arrived to was to display a random puzzle from a list of puzzles we came up with. to get a random value we simply used the rand function seeded with the computer’s time, then modded this value by 10 and added 1. Like this a random value from 1 to 10 is generated, which corresponds to 1 of 10 puzzles. Then the selected puzzle is displayed.