Project Requirement 1 (PR1): Project Proposal

Task

Submit a set of GUI storyboards of your project that illustrates the requirements in items "b.2. User interaction and related requirements" and "b.3. Problem resolution"

User interaction and related requirements

The users are required to interact with your program for him/her to make attempts to solve the puzzle. Hence, your program is required to have the following components and operations:

I. **Puzzle structure** - your program should maintain a data structure that represents the structure and operations done on the puzzle

After analyzing the main github repository that I shall be referencing as I build linking-alzar-pipes, I have noted a few things with regards to data structure representation:

Note:

- Github repository used as reference: lwchkg/pipesgame
- Observations are mostly focused on the pipes.js file as it contains the pipes maze/board generation logic

Observations:

- There are no built in 2D arrays in Javascript as I understand it. Single-dimensional arrays are used to simulate 2D arrays, as shown in the following lines of code:

```
var added = new Array(hsize);
this.pieces = new Array(hsize);
this.states = new Array(hsize);

for (var x=0; x < hsize; x++) {
    added[x] = new Array(vsize);
```

Figure 1. *Observe var added, hsize, and vsize*

These arrays are used for storing information like which nodes to visit, which nodes are already visited, which piece type, and piece state. Currently, I have decided to put into consideration the utilization of Map or Object data structures. However, given the number of elements that must be tracked in pipe maze generation algorithms, changing the current data structure may be a lower priority at this stage.

So, for **puzzle structure**, data structure that represents puzzle structure and operations most likely will rely on arrays (e.g. simulated 2D arrays in Javascript).

Figure 1

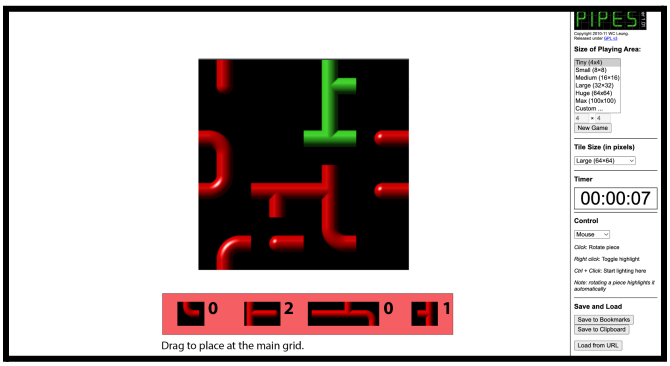

Figure 2

Configure *new game* options to turn empty canvas into the actual game canvas

Figure 1

- New Game

Figure 2

- After pressing the *new game* button, a fully connected pipe maze solution is generated. However, this solution is not readily placed on the grid at the start; instead, the player must drag the pipes to their appropriate positions and rotate them until all connections are completed. The data structure supporting this process is maintained using arrays.

II. **Puzzle information** - your program should save the latest configuration of the puzzle data incurred from the last action done by the user on the puzzle

III. **In-place solution building** - users are provided with all the components of the puzzle during the game. They may be placed on the puzzle board or are viewable on the screen and made available to the user to replace those which are on the board. There is no need for you to provide the users an aesthetically-nice GUI. Simply make sure that the information and controls to play and interact with the game are sufficient for the users.

Figure 3	Figure 4

Figure 3

- Before piece drag/rotation

Figure 4

- After piece drag/rotation
- Proof of a state-saver mechanism
- Data structure representation - Array in Javascript
- Color change on connection(e.g. {red, not connected}, {green, connected})

Add-ons

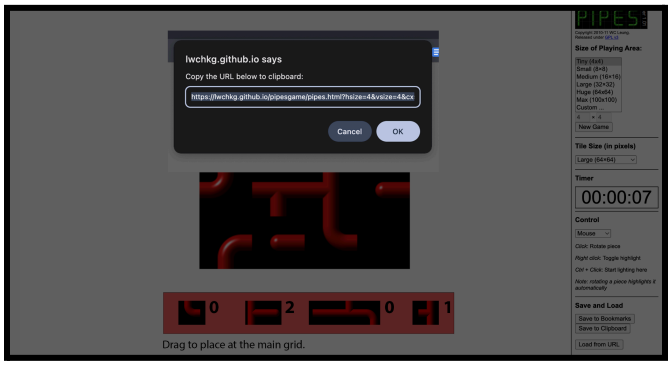

Figure 5

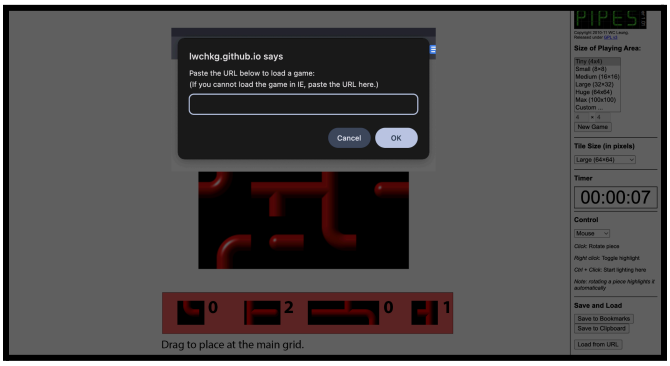

Figure 6

Progress saving mechanism from lwchkg/pipesgame using save to bookmarks, save to clipboard, and load from URL - options

IV. Invalid configuration attempts - your program should warn and forbid users should their attempted action be an invalid one based on the rules and constraints of your puzzle.

As noted in Sections II and III, invalid configuration attempts are represented by red-colored pipes, while valid connections are represented by green-colored pipes. These color assignments are still subject to change.

Note:

- Logic for handling incorrect attempts is tied to the pipe maze generation algorithm

V. Request puzzle solution - users can ask the program to solve the puzzle – starting from the latest puzzle information/configuration resulting from the user's interaction with the program.

/* Problem resolution

Upon the request of the user, your program should provide a solution to the problem instance. This solution must start at the latest puzzle information/configuration resulting from the user's interaction

with the program */

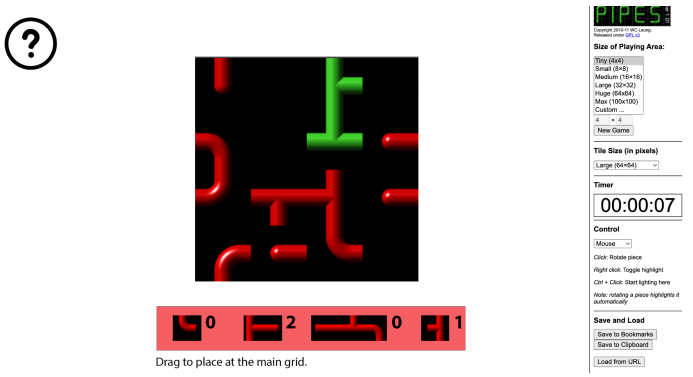

Figure 7

Click the *help* icon

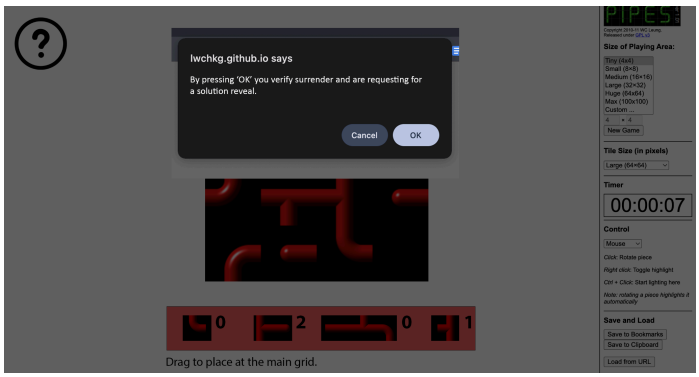

Figure 8

Confirmation asked before proceeding

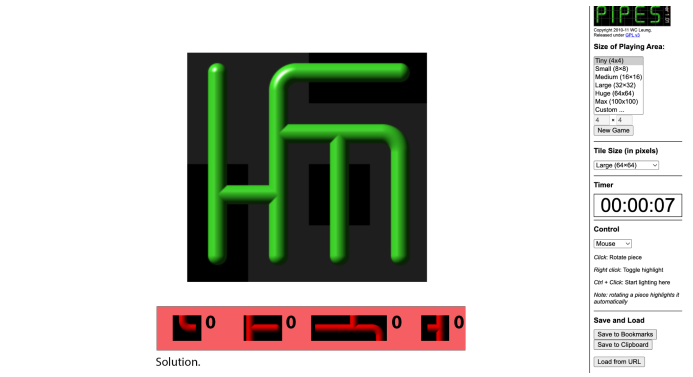

Figure 9

Solution reveal → a fully connected pipe maze