Tracy L Clark
Senior Project Paper
Computer Science
Fall 2016

Catan

Learning Outcomes

I learned that organization is key to completing a project of this scale. The amount of time I spent planning and organizing my information was greater than the amount of time spent coding. Well designed constructors for my map elements made the coding for accessing and updating the map much easier. A well designed API helped with organizing and distributing information flow and function calls.

I didn't know about favicons until this project, but in the course of programming my client side I ended up looking into and discovering their history, which led to me making one for my webpage. In my research I learned about RESTful programming, though I chose not to use it for game security reasons.

During the course of this project I discovered more about the HTML5 canvas, including programming dice animations. I also discovered that divorcing game logic from the render loop meant that I had more flexibility in what I wanted to put on the canvas. I could simply update the data models and see the changes reflected in the rendered map. I used the same design principle with the dice rolls, where I simply sent the result and the animation updated when ready. In previous projects involving canvas, I only rendered when updates were received. In this project I allow the browser to handle the render loop. With this pattern, the display changes automatically with any changes to the underlying objects.

This project gave me the avenue to explore the functional aspects of JavaScript. I much better understand the practical aspects of functional programming from this experience. Functional programming worked best for interacting with the large data structures used to represent the map. By utilizing functional programming, I was able to complete this project with only a single *for* loop!

Most importantly I learned that even with good planning and organization that you can't plan for every eventuality and must be able to reassess and refactor based on changing conditions. I started out with an agile design based on perceived difficulty and realized throughout the course of the project that some things were much more difficult that they seemed at first and had to readjust my expectations and plan based on those discoveries. Using an agile outlook helped keep my project on course for completion. Even if I wasn't able to finish every proposed aspect, I was able to complete as much of the finished project as I had committed to.

Technologies Used

I used JavaScript (ES 6), HTML5, and CSS3 to build my web version of Catan, which I titled Pioneers. I used JavaScript(ES 6) in Node.js running express, socket.io, and http to handle the server side of my project because then I had a consistent language in both the front and back end of my code.

The reason that I chose to use the newest version of JavaScript was that it provided me with better access to the functional aspects of the language. Also, I wasn't as familiar with the newest features of the language, so I wanted to use this project as an opportunity to utilize some of the newest functionality of a rapidly growing language. Those features gave me the ability to allow the language to do what was necessary, not force it.

I used HTML5 with canvas for my graphics because the DOM is the user interaction model with which I have the most experience. The portability of HTML5 and the consistency of appearance between platforms is desirable. It has a plethora of options without the use of additional libraries. The standard modern way of handling animations in JS is to use the requestAnimationFrame system provided by the browser. However, not all browsers handle this the same or provide the same level of consistent behavior. To provide the best user experience across platforms, I used an open source shim to provide consistent performance. This shim, titled rAF.js in my source code, was written by Erik Möller with fixes from Paul Irish and Tino Zijdel and released under the MIT license.

I used MongoDB for a login and account creation database, setting it up to use unique indexes for username entries so that only one entry per username was allowed to be created. I chose MongoDB because I had used it previously and knew it was quick to set up and for the use case of my program it was a good choice. My use case was small scale access without complex or large amounts of data that would require a relational database.

Finally, for source control purposes, ease of deployment, and ease of access from multiple systems I used a git repository hosted at github.com/tracylclark/SP/. This enabled me to track code changes with ease. It also allowed for quick deployment of code changes I made on my local machine to the linux server that I have hosting my database and code.

Software Organization

I organized the software using modules. I approached this by planning out how to separate the various duties the program needed to perform and making modules for them based on the distinct duties of each. A basic model of my project files is pictured in *Figure 1*.

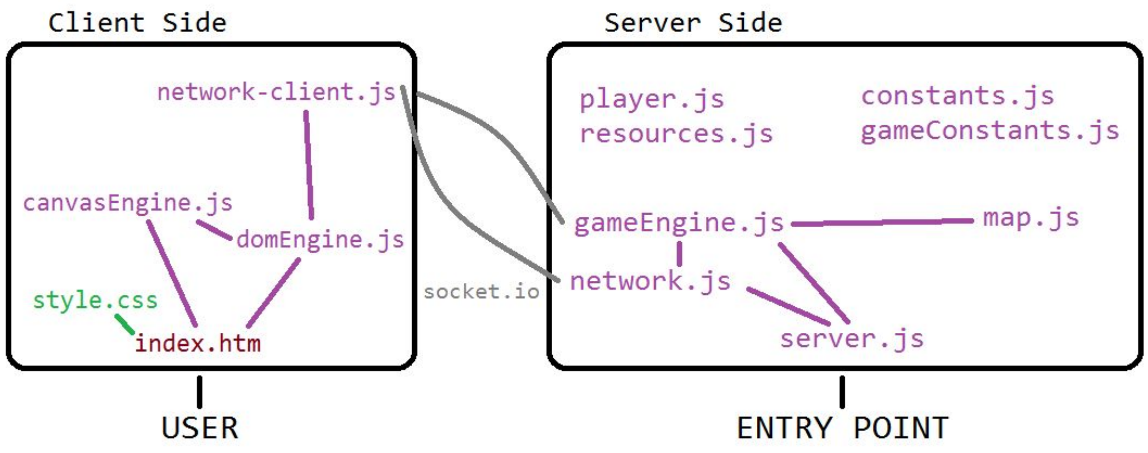

Figure 1 : Interaction Diagram

Server side : server.js

The server module serves as an entry point for the program, and is responsible for setting up the initial socket communication and basic file serving. It starts the express, http, and socket.io protocols and initializes the game engine and network. The network module is passed the socket object so that it can handle all incoming communication from the client, and both the gameEngine and network modules are given a reference to each other.   Each of these modules shares a common array which is used to hold Player objects. The player module is a constructor function for new Player objects.

Server side : network.js:

The network module is where the socket API is set up; it handles all incoming communication. It is responsible for connecting to the mongoDB database and handles all non-game related logic, such as logging on and account creation. It holds the player and spectator information and sanitizes any raw data received from the users such as chats to protect against injection attacks. The network is initialized by the server and sent a reference to the game engine for communication about any requests that are related to the game logic.

Server side : gameEngine.js

The gameEngine module handles all game related logic. It holds the game state, including the sole reference to the current game map. It occasionally sends socket messages to specific users, and it can invoke functions on the network module to send messages to all connected sockets.

Server side : constants.js, gameConstants.js

Two modules, constants.js and gameConstants.js are used to hold hard-coded data, effectively in JSON format. The constants module contains information used to initialize a new map module. The gameConstants module holds information used by the game engine, such as the starting development deck and the cost to build of various improvements in the game.

Server side : player.js

The player module is a constructor to represent each player of the game. It holds network related information, such as the socket object used to communicate with that player's browser, game related information such as their current resources or built infrastructure, and a handful of methods for modifying this data.

Server side : resources.js

The resources module is a constructor function used for creating resources objects. These are used to represent all resources in the game. Players have a resources object for their current resource totals. The costs of improvements are stored as resource objects, which can be subtracted from the Player's resources.

Server side : map.js

Finally, the map module holds the complicated data structure that is the Catan map. The map is depicted in this module as a graph connecting three types of objects: Tiles, Edges, and Vertices. Each of these objects has relevance within the game rules of Catan, and so each is their own class of object.

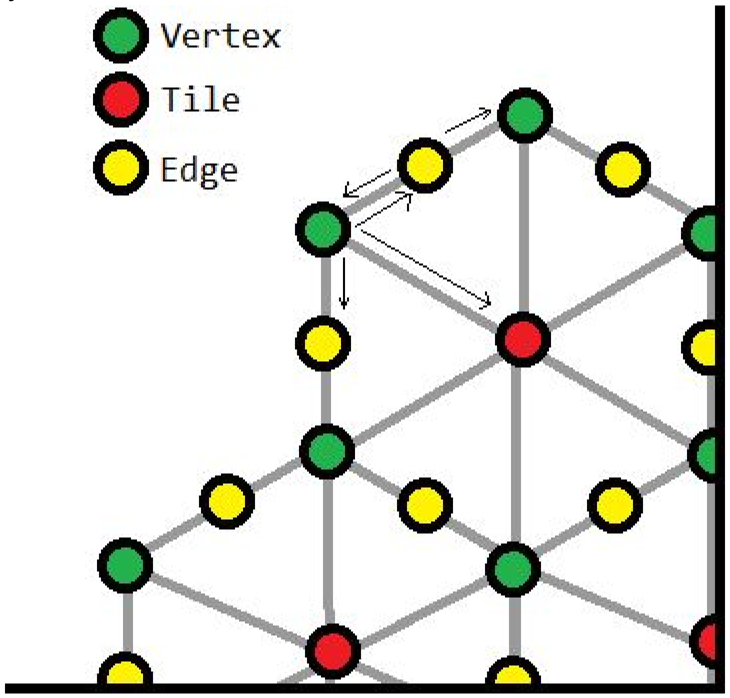

Figure 2 : Relational Diagram

```
function Vertex(coords){
	var invalidCoords = [
		{x:0, y:0}, {x:1, y:0}, {x:0, y:1}, {x:9, y:0}, {x:10, y:0}, {x:10, y:1},
		{x:0, y:4}, {x:0, y:5}, {x:1, y:5}, {x:9, y:5}, {x:10, y:4}, {x:10, y:5}
	];
	this.coords = coords || {x:0,y:0};
	this.accessible = true;
	if(invalidCoords.find((e)=>this.coords.x == e.x && this.coords.y == e.y)){
		this.accessible = false; //invalid coords are not accessible
	}
	this.tiles = []; //array of 18 tiles
	this.edges = [];
	this.owner = null;
	this.vendor = null;
	this.server = false;
	this.database = false;
}
```

Vertices hold a reference to the tiles and edges they are adjacent to. They are stored in a two dimensional array (in Javascript, this is an array of arrays) with their coordinates being usable as a reference to their position in the array. Due to the shape of the playing field and the coordinate system chosen (*Figure 4*), some coordinates within the range x:0-10, y:0-5 are not used by the game. These are hardcoded in the Vertex initializer. Valid vertices have their `accessible` property set to true, invalid coordinates have this property set to false. References to Edges are held as an array of integers; each being an index to an Edge in the edges array. References to Tiles are similarly held as a reference to the index of that tile in the array. Vertices may have 1,2, or 3 tiles, and 2 or 3 edges.

```
function Edge(u, v){
	this.u = u;
	this.v = v;
	this.owner = null;
}
```

Edges hold a reference to the two vertices they are in between, as well as one property for game logic. Edges are stored in a one-dimensional array. Their indices can be used to find them, as can a two-vector object.

```
function Tile(id, type){
	this.id = id;
	this.token = null;
	this.resource = type;
	this.hacker = type==="DarkNet";
}
```

Tiles hold only their own state: id, which is their index in the tile array (and also understood as their location), and three properties pertaining to the game logic (token, resource, and hacker). Tiles are stored in a one dimensional array where their index defines their position. This ‘coordinate’ pattern matches the way tiles are placed in the boardgame, spiraling around the board until meeting the middle(*Figure 4*).

Client Side :

The client side code is split into three modules. All three are included by index.html which also includes a small bit of Javascript to call on these modules once the page is fully loaded. From the client browser perspective, the global namespace has only four variables added to it: these three modules and one 'name' variable visible to all modules. The stylesheet, style.css, is used to set the initial properties of most DOM elements.

Client Side : network-client.js

This module handles all communication with the server. It calls on domEngine or canvasEngine as needed and holds no state of it's own besides a reference to the socket.

Client Side : canvasEngine.js

This module controls the canvas element of the DOM. It holds a simplified version of the server's map structure, modified to be just what the canvasEngine needs in order to draw it. Each Tile, Vertex, and Edge object is given it's own draw() function, so these map elements know how to render themselves on the canvas. The canvasEngine registers a render function with the browser's requestAnimationFrame (if not present in the browser, several fallback options are tried by rAF.js). This render function draws the whole current game map, and the dice when needed.

Client Side : domEngine.js

This module interacts with several DOM elements defined in index.htm. CSS is modified to display or hide different elements as a reaction to certain network events. The contents of these elements is changed to represent the game state and display it for the user. One interesting design of this domEngine is that it creates a tree structure to represent all the dom elements in the html file. For example, the password field of the login dialog is stored in the dom.login.password object.

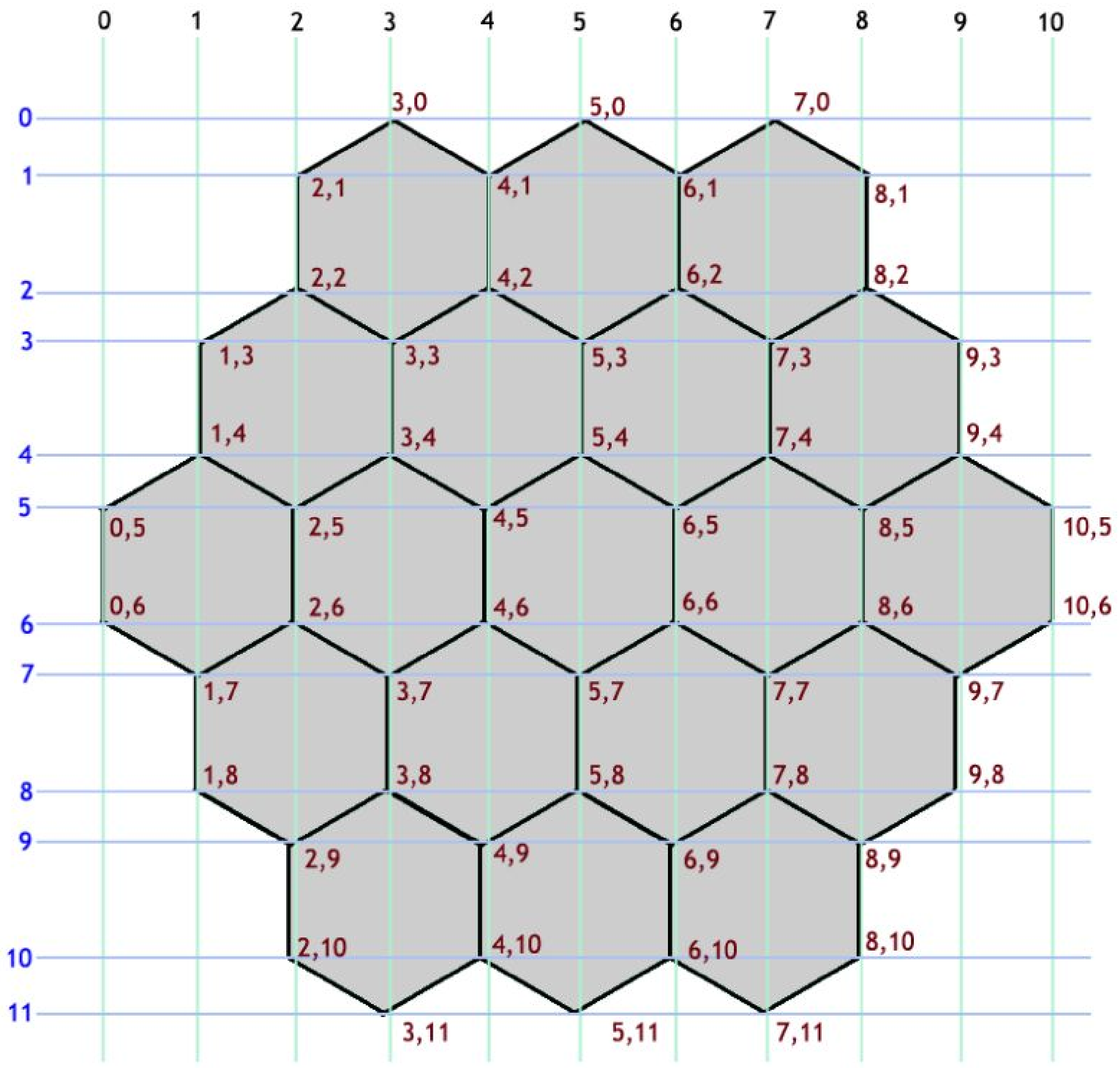

Figure 3: Initial Map Design, 11x12

INVALID COORDINATES:

0,0	0,1	0,2	0,3	0,4	0,7	0,8	0,9	0,10	0,11
1,0	1,1	1,2	1,5	1,6	1,9	1,10	1,11		
2,0	2,3	2,4	2,7	2,8	2,11				
3,1	3,2	3,5	3,6	3,9	3,10				
4,0	4,3	4,4	4,7	4,8	4,11				
5,1	5,2	5,5	5,6	5,9	5,10				
6,0	6,3	6,4	6,7	6,8	6,11				
7,1	7,2	7,5	7,6	7,9	7,10				
8,0	8,3	8,4	8,7	8,8	8,11				
9,0	9,1	9,2	9,5	9,6	9,9	9,10	9,11		
10,0	10,1	10,2	10,3	10,4	10,7	10,8	10,9	10,10	10,11

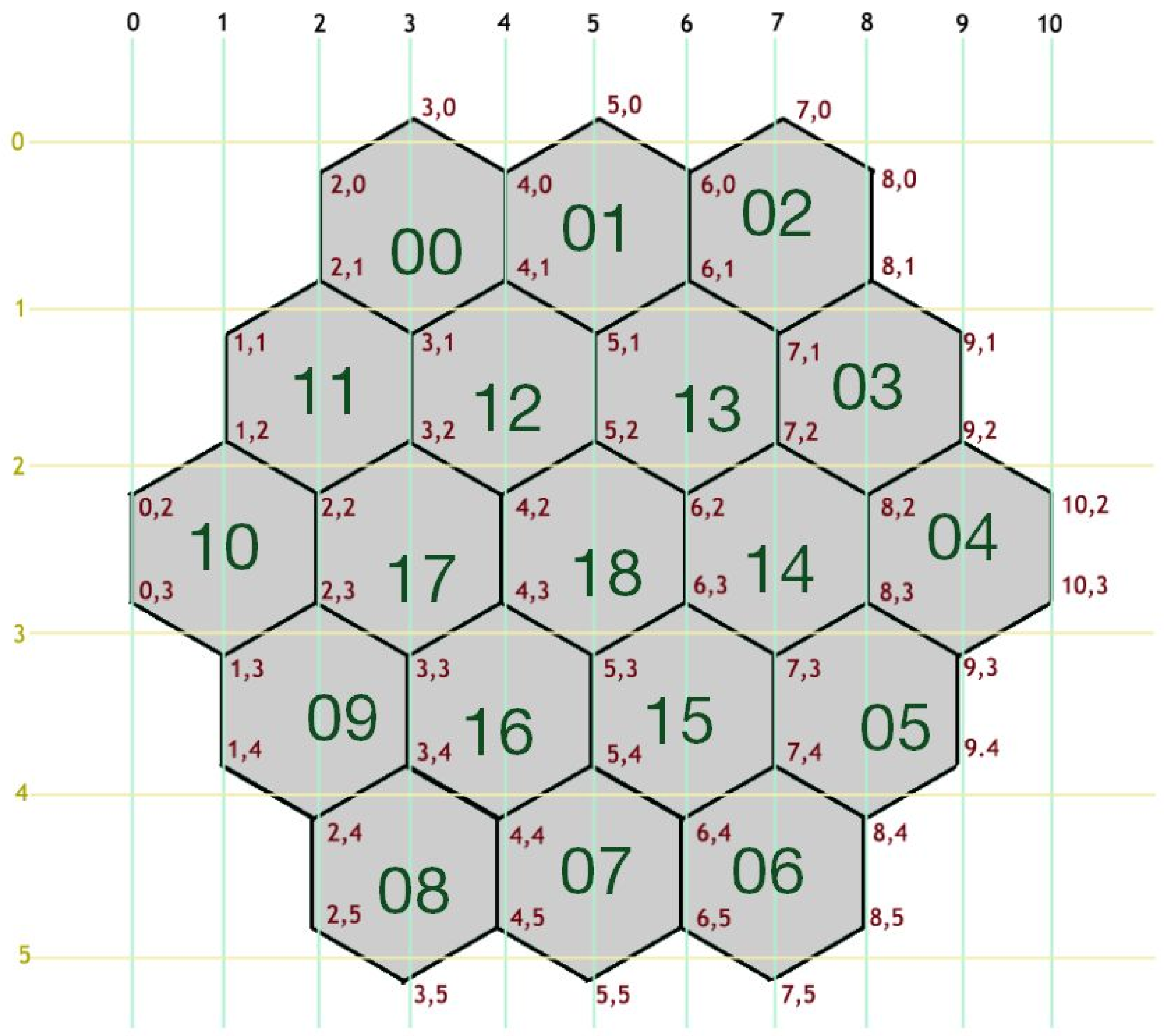

Figure 4: Revised Map Design, 11x6

INVALID COORDINATES:

0,0	1,0	0,1	9,0	10,0	10,1
0,4	0,5	1,5	9,5	10,4	10,5

Feature Point Synopsis

Grading Scale	Grade	Total
95+	A	105
90-94	B	
85-89	C	
80-84	D	
<80	E	

Core Game	Points	Points Earned
Create a default game map for beginner players	3	3
Create a randomized game map for advanced players	3	3
Implement a tiled map grid	3	3
Implement a hexagonal tiles for the map grid	8	8
Implement tile types	1	1
Implement resource number tokens	2	2
Implement resource generation through dice rolls	5	5
Track player resources	2	2
Implement development cards	5	5
Implement road building	2	2
Implement settlement building	3	3
Implement city upgrades	1	1
Implement robber	3	3
Implement victory point system (includes checking for a win)	2	2
Implement victory points for longest road: Longest Path Problem, NP-Hard	13	
Implement victory points for largest army	1	1
Implement Harbors	3	
Implement trading with other players	5	5
Implement 4:1 resource trade in	2	2
Implement turn system	2	2
Implement desert tile	2	2
Implement Set-up Phase	5	5
Implement roll-off for starting player	1	1
Implement starting resources	2	2
Implement even number token distribution	3	3
Implement random number token distribution	2	2
Implement game options for players to choose style of game play	3	3
Total Available for Core Game	87	71

Graphical Interface	**Points Possible**	**Points Earned**
Draw game map with HTML Canvas	8	8
Use image sprites instead of just drawing tiles	2	
Implement map camera (ability to pan and zoom on game map)	3	
Show graphical dice	1	1
Animate dice	5	5
Viewable current player scores	2	2
Login Screen	1	1
Rules Overlay	1	
Game play message displays	1	1
Use Socket.IO to keep game synchronized between players	3	3
Make a mobile friendly UI design	13	
Total Available for Graphical Interface	40	21

User Features	**Points Possible**	**Points Earned**
Implement spectators	3	3
Implement a “local hotseat” multiplayer option	5	
Implement “local hotseat” with online play (so that a single PC can be used for multiple players in in a single location playing against people in other locations)	8	
Saveable game state	8	
Persistent game state	3	
If a player leaves the ability for a spectator to take their place	1	
Create game lobby	5	
Create multiple play rooms for simultaneous games	8	
Users have the ability to “lock” a game and put a password on it	2	
Implement being able to choose to spectate or play the game	1	1
Implement a chat box	2	2
Implement voice chat	8	
Implement game login (with MongoDB validation)	2	2
Implement user account creation (with MongoDB validation)	2	2
Maintain server-centric security policy with clients handling no game logic	3	3
Total Available for User Features	61	13

Perceived Difficulty vs Reality

The design was the most complicated part of the project. The API interplay as well as how the modules split tasks and communicated with each other took the majority of my planning time. The relationship between vertices, edges, and tiles as well as how to represent them programmatically was a difficult process, which led to me considering multiple possibilities for grid coordinate systems. The two major grid systems I considered are represented in *Figure 3* and *Figure 4*. The initial grid was the first I considered, but I was able to extrapolate the much more streamlined revised grid. The initial grid is more easily accessible, but has more invalid coordinates. The second grid has an offset, however it only needs to be accounted for in the UI. This (www.redblobgames.com/grids/hexagons/) was a good resource that I used when considering my options and ultimitely led to my decision of the revised grid system.

The GUI was another difficult aspect, especially drawing a hexagonal grid with click handlers for tiles, vertices, and edges. The angled design of most edges and the hexagon design of the tiles meant I had to code *"hit boxes"* for them, which were only selectable during phases of the game where it was necessary. This prevented interference between selectable objects.

Implementing the development cards should have been separated into different tasks considering that each card implemented added a unique addition to the game play. The user interface should have also been several categories. When assigning points for the user interface, I only applied points for displaying the state of the game. I neglected to account for the fact that all user input programming didn't fall into these categories. If I had planned for these, I would have assigned them a 5 due the the amount of inputs and hiding and revealing them based on game state.

Spectators were easier to implement than I thought they would be, since I just created everyone as spectators and allowed them only to view the map and chatbox. I would have probably given it a 2 with the knowledge I have now.

I ran across an aspect of the game play that I had failed to account for in planning which I did not implement. If a 7 is rolled, all players with more than 7 resource cards must discard half of their cards (rounded down). If I had account for this and scored it I would have assigned it a 2.

Regrets

If I could do the project differently, I would have liked to add more polish to the final product. The communication is fairly streamlined, but could use some refactoring to make it flow better. Even with large amounts of planning I still had to stop and reassess every time something I hadn't considered came up and work it into the code base.

I would have liked to use actual references to the objects instead of array indexes in the map model. A coordinate system for tiles and edges would have allowed me to use more algorithmic methods for linking references between the *vertice*, *tile*, and *edge* objects rather than hardcoding them.

I didn't factor enough time for programming the client side display, so I wasn't able to show or test everything I had coded to see if it worked. I also put in UI displays for aspects of the game that I hadn't ended up not having time to complete, such as *longest path*. I input menu options for *harbors* as well, which I did code on the server side, but didn't allot enough time for rendering them on the client. These are the only two major aspects of the base game that I was unable to complete, but when I considered getting the finished product out the basic game display took priority. I needed to be able to show my project working, so completing a basic GUI was a necessity.

Conclusion

This entire project was a learning experience which bettered my understanding of design, agile development, and time management techniques. I had to deal with real world interference and breaking up a large scale project over several months while keeping track of implementation and API communication. Careful planning and design made implementation easier, but didn't eliminate the normal flaws present in a large scale project. I suffered from analysis paralysis at several points of the project, where I struggled with overthinking certain aspects and had to push through and make a decision, without struggling with whether it was the best one.

I did fall into the well known trap of trying to program everything right the first time instead of refactoring later, but careful prior planning helped keep this project on track and I ended up having very little code refactoring later. Most code changes later in the project had to do with things not working exactly as intended and revising them to perform their planned function. Given more time there are parts that could use some refactoring and be cleaned up better, but overall I'm pleased with the outcome. I did have several aspects where the planning really came through and I was able to implement functions that did exactly what was necessary. Overall I'm pleased with how the project came out. It's not the prettiest but then, I'm a programmer, not an artist!