Diego San Miguel

dsanmigu@andrew.cmu.edu

Section K

Mentor: raunaksg@andrew.cmu.edu

Design Proposal

Project description:

My project is going to be an online multiplayer monopoly game. At MVP I hope to have a fully functional monopoly game that allows up to four players to connect and play against each other. The game screen will consist of two main parts; on the left will be the game board with all the pieces, and houses currently on it, on the right will be an activity log that lists all the events that occurred in that round, a profile for each player that shows all their properties and cash, and a button to roll dice. Further steps after MVP would be to add an optional trading feature between players, audio sound effects to the game, and then hopefully make the board CMU/112 themed.

Competitive Analysis:

Obviously, there are lots of online monopoly games out there. From what I’ve seen in my research, many versions of the game will use 3D graphics to get a bird’s eye view of the monopoly board, and have sophisticated animations to show rolling dice and moving pieces. What I have not seen thus far is something akin to the activity log that I mentioned in my project description, and the trading feature between players. I think that the activity log stands out as a good idea because it allows players to be updated on what’s been going on in the game if they had to stop playing for a while. Most other features that I mentioned however have been already done by competitors which is to be expected given the fact that I am implementing a popular boardgame into python

Structural Plan:

To get to MVP, I will likely need 4 separate python files, and a folder full of PNG images related to monopoly. One file will be the server which will handle receiving and sending information from all clients so that everyone’s game is up to date in terms of stats and players positions. Another file will be the cards file. This file has several dictionaries that sort all the monopoly cards into their respective type, so there will be a dictionary of property cards, community chest cards, etc. In these dictionaries, all the hard-coded values such as rent, price, mortgage value, etc. will be stored in lists. Finally, this file will also have a list that is every spot in the monopoly board in order. My third file will hold the player class. In this class I will define all the properties of a player, such as their list of properties, cash, etc, I do not think however that I will be defining methods that correspond to player actions here since it would be best to rather have those in the client file. The client file will be my most important file. Here I will have the Monopoly class which will store all the data of the game’s current status and have defined all the methods which are actions that players can make. Here I will also be connecting to the server and sending encoded messages everytime an action is performed

Algorithmic Plan:

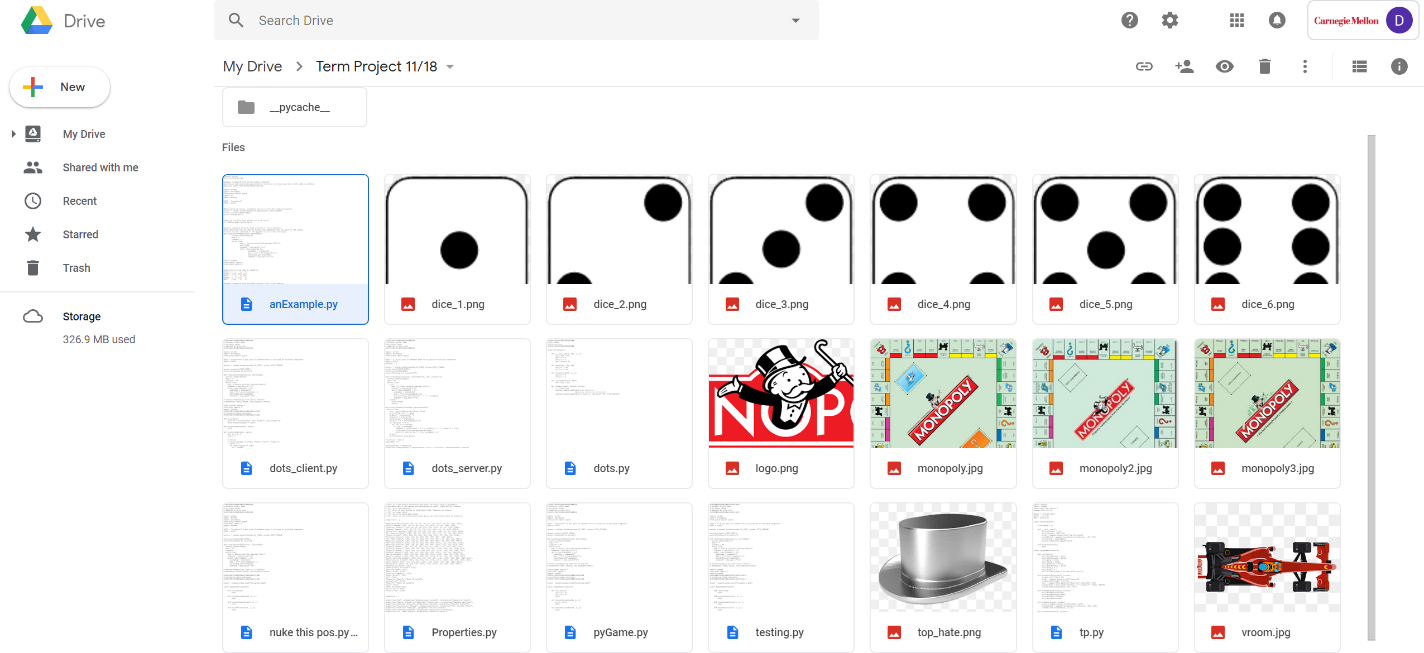
The trickiest part of this project will be making sure that each client has their game up to date and is exactly the same as everyone else’s game. I have to break down this problem into two different parts, one will be receiving messages from the server and the the other will be sending messages to the server. The only time I’ll need to send messages to the server to update the game state will be when a player performs an action which can be any in the following list: Buy a property, pay rent, sell a property, offer a trade, or move their piece. To do this efficiently, I will have a variable called “msg” in every method that updates the game state, the msg variable will be built up by the specification of the actions that occur within the method, and then will be sent out to other clients using the command “self.server.send(msg.enccode()).” To receive messages from the server I will have to modify timerfired so that it is constantly checking for received msg encodes and can classify them based on their data so that I can properly update the clients game based on the received message. This method is inspired heavily from the dots\_client and dots\_server example given in the 112 sockets manual.

Timeline Plan:

Getting a working player to player connection and having pygame working properly on this server connection is of the utmost importance and should be the first thing completed by TP1. After that. By TP2 a basic functioning monopoly should be implemented. I’m not worried about having very clean UI and lots of neat little features by this point such as requesting a trade, clicking on properties to see their stats etc. By TP2 I am just aiming to have, at the minimum, a playable monopoly game with up to 4 players on 4 different machines. This will hopefully be my MVP. By TP2 I would have liked to at this point have a very nice UI, to compliment small additional features, such as clicking on a player and a nice list will appear of all their properties, cash etc. I also plan to have sound effects in the game by this point such as rolling dice or other sound effects when you pay rent, pass go, etc. Lastly, if time permits, I plan to redo the theme of the board and make it CMU themed.

Version Control:

Because I am on Windows I cannot use iCloud to immediately back everything up. To fix this I have made a folder in my 112 folder named “Term Project.” Within this folder are all the files that I am currently working on and all the images I need in the game. Every time I finish a coding session I upload the entirety of the folder onto Google Drive, naming it a new version of the project. This way I not only have the most up to date version of my project but will have a folder filled with every previous iteration of the project.



Additional Modules:

The only additional module I am using for my MVP will be Pygame. I am also using sockets for the online component of the game. After MVP however I will likely use Pyaudio to implement sound effects.

Design Update:

Implementing trading between players has been far harder than previously expected. Because of this I do not have it by TP2 and instead not including it as part of m MVP. I am still aiming to implement audio after reaching MVP. Another design change I have made is in the rules of the game itself. I wanted to make the game faster-paced and cutthroat so I decided to change how the jail, chance, and community chest work. Now jail fines you $100 and sends you to jail, but you don’t waste time by losing a turn. The chance spot now teleports you a random number of spots (from 0 to 12) from where you are currently. The community chest will either fine or pay you a small amount of cash or, in rare cases, reward you with an unowned property on the board. Other than this, my design plans have not changed at all, all other features have been implemented to make a fully functional monopoly.

TP3:

All the features I have talked about previously have been added to the game, including trading and even auctioning. To make trading simpler and not take far too long to code, I made it a simple system of trading two cards between players. Another feature that I decided to add after reaching MVP was the ability to save your game. Because monopoly is usually quite a long game, it is reasonable to expect that people would want to drop the game sometimes and pick it up later. To do I this, I created a JSON file in the project folder that will save any relevant game data when the user decides to save, and then this data can be loaded up again on a new server and client. I also decided to add some small sound effects to the game just to keep it from being too boring audio wise.