Python Programming Neema Badihian
12/13/2018

Final Project

For my project, I decided to simulate the trading card game Magic the Gathering (MTG). MTG is a fantasy-based game with many rules and conditions. It has been around for a couple decades and it is still coming out with new cards to this day. I knew my first big problem was going to be getting all the data on those cards. Luckily, I found a JSON repository of the cards at *mtgjson.com*. This data was invaluable, but it was missing something. It didn't have the images for the cards. Apparently there used to be a website that worked with *mtgjson.com* to provide images for developers, but it had been shut down due to copyright infringement. I found another site, though, called *scryfall.com*. Scryfall had a lot of the data that MTGJSON had but it wasn't as well put together. Unlike MTGJSON, however, it had images. I decided to use these two JSON repositories hand in hand. I would get the data on each card from MTGJSON and I would get the images from Scryfall.

Now that I had my repositories, I had to learn how to actually access them. I have dabbled with JSON before in other languages like Swift and JavaScript, but I was very inexperienced with it and I had never used it with Python. I did some research and I figured out how to get data from the JSON files. It was very straightforward with MTGJSON because the cards were organized by name, as opposed to index. That meant I could easily access any card as long as I had its name instead of having to iterate through the entire data until I found my card. Scryfall was not organized this way. It was organized by index. So first I found the data on the cards through MTGJSON and then I would iterate through the entire Scryfall JSON string until I found the card I was looking for, finally grabbing its image's URL.

In order to use the URLs for the card images, I needed to learn how to use Tkinter. Our graphics.py library was very helpful, but it didn't support getting an image with a URL. It only supported getting a local image with a file path. I went online and did some digging. I found different blocks of code that should have solved my problem but of course they did not. I pieced them together and finally was able to get something that resembled what I wanted. Instead of just using Tkinter, I also had to use the PIL library, the io library, and the urllib library. What I made with this code wasn't exactly right though. It displayed the card images in a table of sorts that determined the size of the window. I was able to move an image to a different point on the window, but its canvas still took up the whole window so that its background color filled everything while the image of the card was on the bottom right corner. I decided to leave it be for now and work on another part of the project because this was taking too much of my time. I came back to this much later when I decided to use graphics.py. At that point, I combined my code with the code from the graphics library and was able to make it work. I also had to modify the graphics library a little to allow PIL images to be passed to it. I will talk more on this later in the paper after I have explained how the other parts work.

My initial goal was to make an AI that I could play MTG against. I knew nothing about AI so I went to Udemy to see what I could find. Udemy is an online course website that I often use to expand my knowledge. There was a course that seemed to be well reviewed and was focused on reinforcement learning. I knew that reinforcement learning was what I needed to know if I wanted to create something similar to Google's AI that plays Go. I spent some time on the course but eventually I realized that this just was not going to be possible in the time I had available. I already knew I wouldn't be able to make a great AI in this little time, but I thought I

would be able to make something rough. I decided to alter my plan and get rid of the AI aspect for now. Instead, I would make a game where the computer always has the same cards and the user is able to choose his/her cards from a selection screen as is done with Hearthstone.

This also proved to be too much, though. As I went through the cards, it hit me just how many different effects there are. Many cards have unique effects and I could not account for all of them. Once again, I decided to modify my project plan. I finally decided on using the same deck for the player as I would for the computer. There is a website called *mtgdecks.com* that has various deck designs that people have uploaded. I went with one of the more popular and easier to learn decks known as White Weenie. I, myself, started playing MTG with a variation of a White Weenie deck so I figured it would be something I understand well.

Inside the file 'buildDecks.py', I created a class called 'whiteWeenie' that creates a White Weenie deck. The class works like as follows. Upon initialization, it creates a list of all the card names in a White Weenie deck. It then creates a list of cards of the class 'Card' using these names. Finally, it shuffles its cards.

The 'Card' class was one of my favorite designs. It only requires one parameter, the name of a card, and it will create an object with all the card's data from the JSON repositories. I created individual methods to get each attribute of the card and then placed these methods inside of a method called 'getCard()'. The 'getCard()' method runs all these individual methods and set's the card's attributes accordingly. I also gave the 'Card' class methods to display all of its attributes. Just like with the 'getCard()' method, the 'showCard()' method accesses each of the 'show' methods to display all the card's attributes.

There was also a third class I created in 'buildDecks.py', but I didn't get to use it. This class was called 'Token'. The 'Token' class was created for cards whose effect creates a token. A token can differ from card to card. One card might create a token with 1/1 stats, while another might make one with 0/3 stats that is also of type 'flying'. There can be all sorts of variations, so this class requires a lot of parameters to be passed for initialization. I had to make this class because there were no tokens in the JSON data, as far as I could tell.

The next thing I worked on were card effects. I wasn't able to implement this one but I did create some pseudocode and some functioning code for it. I started out by giving these card effects their own file called 'cardEffects.py'. I went through each card in the White Weenie deck and made a function for each unique effect. If I had more time, I would have also made a function that figures out when to use which function. I didn't understand the effects of several of these cards, since their descriptions can be a little lacking, so I had to do some research to figure out how they work exactly. Even though I have played the game many times, I never really understood how complicated it is. These effects seem easy to use in person, but when it comes to telling a computer how to use them, it become very complex.

Now I had a class for cards, for the deck, for tokens, and I had pseudocode for effects, but I did not have any code to describe a player. I created a new file called 'playerLib.py' and made a class called 'Player'. The 'Player' class only requires one parameter to be passed, the player's name. This way, I can differentiate between the computer and the player. When a Player is initialized, it immediately creates a deck of the class 'whiteWeenie'. It also is given 20 life points, which is the standard starting ground for all players.

Within the 'Player' class, I created a few different methods to deal with whatever situation might arise. First off, I would need to know who was making an action so I created the 'showName()' method. I would also need to keep track of the player's life so I created a 'showLife()' method. The player's life could both decrease or increase, that meant I needed a 'subtractLife()' method and a 'addLife()' method. The player had already created its deck, but it needed to be able to get cards from this deck which is where the 'drawCard()' method comes in. The 'drawCard()' method returns a card of type Card but that has two different uses. It could be used to simply get the card as an object, or it could be used to print the card's name. if a card of type Card is printed, its __str__() method will return the name of the card. Now that the user could draw a single card, I wanted it to be able to draw multiple cards for the beginning of games. At the start of every game, both players draw 7 cards. I created a method called 'initialDraw()' that instantly draws 7 cards and adds them to the player's hand. To add to the player's hand, I created a method called 'addHand()' that will append a Card object to the list called 'hand'. The 'getHand()' method returns each Card object in the player's hand. Meanwhile, the 'showHand()' method displays the images of each card in the player's hand on the game board. I was working on a couple methods to get the types of the cards in the player's hand and sort the hand by type, but I wasn't able to get that working just yet.

The last class I needed was the 'Board' class for the game board. When the 'Board' class is initialized, it immediately created a window using the graphics library. The size of the window is based on the size of the image I used for the background. I looked online and found a stock image that had a sense of fantasy to it for this. Each player, both human and computer, would

need a deck displayed on the game board so I found an image of MTG card back to display. The corners of the image were a little messed up so I used Photoshop to fix them.

I needed an area for the player's hand to be displayed. I thought old fashioned parchment would look good so I found another stock image online to work with. I wanted an area on the parchment that was just for text. This text would tell the user who's turn it was or whether it was his battle phase and so on. I happened upon an ornate vector frame on a site and decided to use that. I would also need to display the how much life both the player and the computer had left.

I didn't want to take the player away from the feel of the actual game, so I found an image of a 20-sided die as would normally be used. I had to make sure the die had no numbers on it, though, because I didn't want to use a different image for every side of the die. I figured the player and the computer shouldn't have the same color die either, since that so rarely happens in real life. I modified with Photoshop the blue die I found online and made a yellow version for the computer. I then placed the dice in the window with similar x and y coordinates as the decks. To display how much life the player and computer actually had, I used the graphics library's 'Text' method and placed the life points on each die.

The final file I made was the 'playGame.py' file. This file was only meant to run methods from other files. This way it would be easy to see what happens and in what order. Nobody likes spaghetti code. In the 'playGame.py' file, I created an instance of the Board(). I then created two instances of Player(), one for player1 and one for the computer. Then I displayed the life of both players and presented the player's hand. Finally, I waited for the user to click on the window before exiting.

I did not do what I said I was going to do with this project. I did not touch anything AI related, and I was not able to create a functional Magic the Gathering simulation. The game was far more complicated than I anticipated. Nevertheless, I learned a lot with this project. I learned about JSON, I learned how to use images with a URL, I learned how much detail is needed for a game, and I explored Python to a depth I previously had not. It was a great learning experience and I hope to work on this project more in the future as a personal goal.