Abby Winegarden
CS 480 Senior Project
Apr. 22, 2019

Predictors Of Catan

Introduction

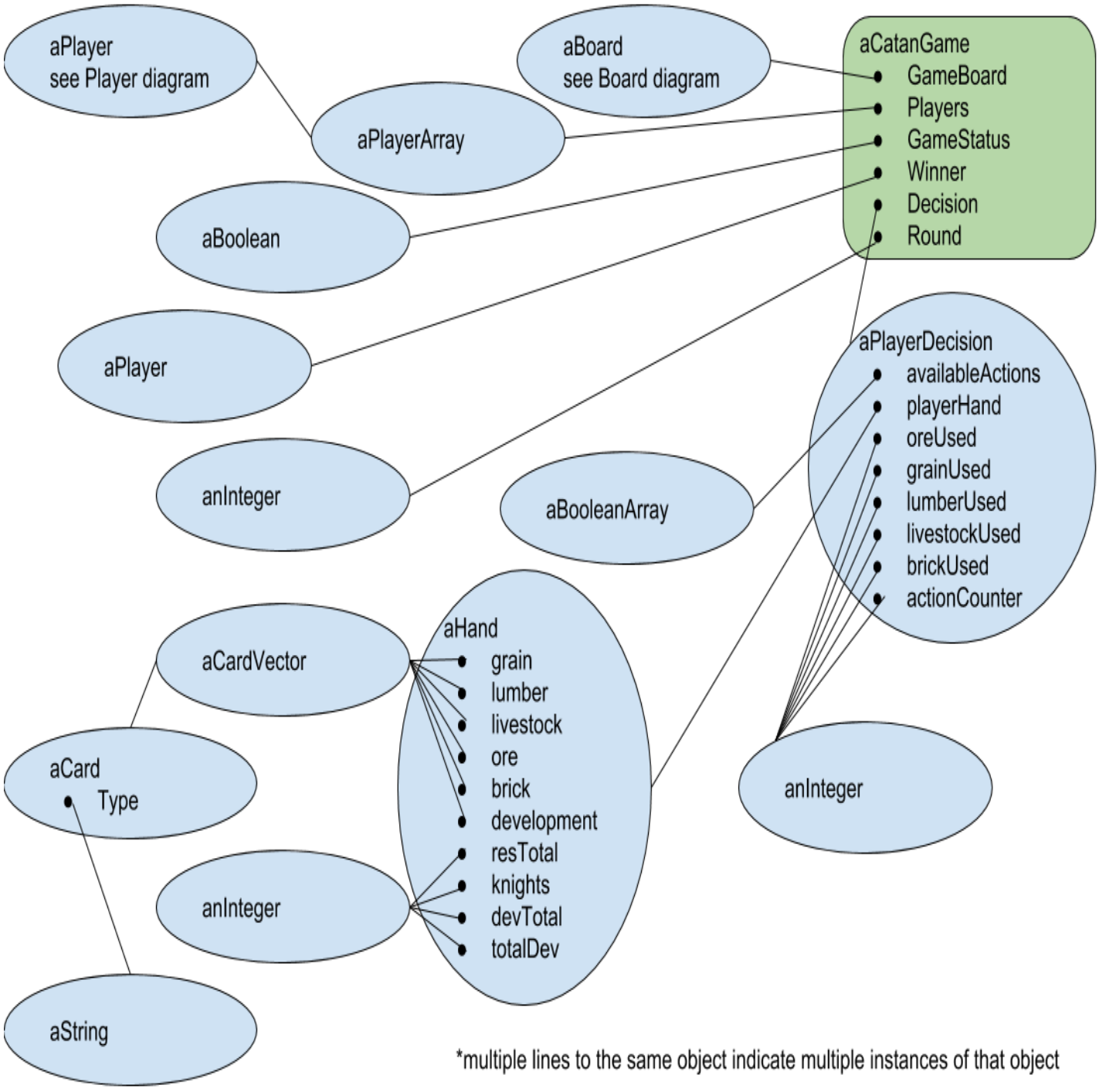
The aim of this project is to predict, using machine learning, the winner of a game of *Settlers of Catan* (https://www.catan.com/en/download/?SoC_rv_Rules_091907.pdf) in any given game state. McKenzie Shores and I chose this project because we are both fans of the game and both wanted to learn how to implement machine learning. We decided to use a generator we found on GitHub (https://github.com/sorinMD/MCTS) known as *Smart Settlers* and a game generator that we built ourselves to gather the wide range of data necessary to make our predictions. Then, using our collected data, process it and use Google's TensorFlow API to make our predictions.

We broke the project down into five parts: 1.) break down the *Smart Settlers* generator to understand how it works and modify it to output data, 2.) build our own generator, 3.) gather data from both generators and process/format it, 4.) a. research TensorFlow, neither McKenzie nor I had ever used TensorFlow before; 4.) b. build an algorithm in TensorFlow and make predictions based on our data. According to our proposal McKenzie would handle #1, we would pair program #2 with myself as the lead, I would be responsible for #3, and we would pair program #4 with McKenzie as the lead. However, due to time constraints from our respective schedules, we could only get together once a week to pair program for roughly two hours. In that time we discussed design and any issues we may have been having as well as writing and executing tests. There was also a hidden complexity in the creation of our own game generator that neither of us realized. The actual division of work came down to McKenzie tackling #1, #3, and #4 while I handled #2 and assisted with #4 when I was able. The game generator ended up being over 2,200 lines of code, which I wrote largely in my own time, not including the tests that McKenzie and I wrote during our meetings which were roughly 800 lines of code.

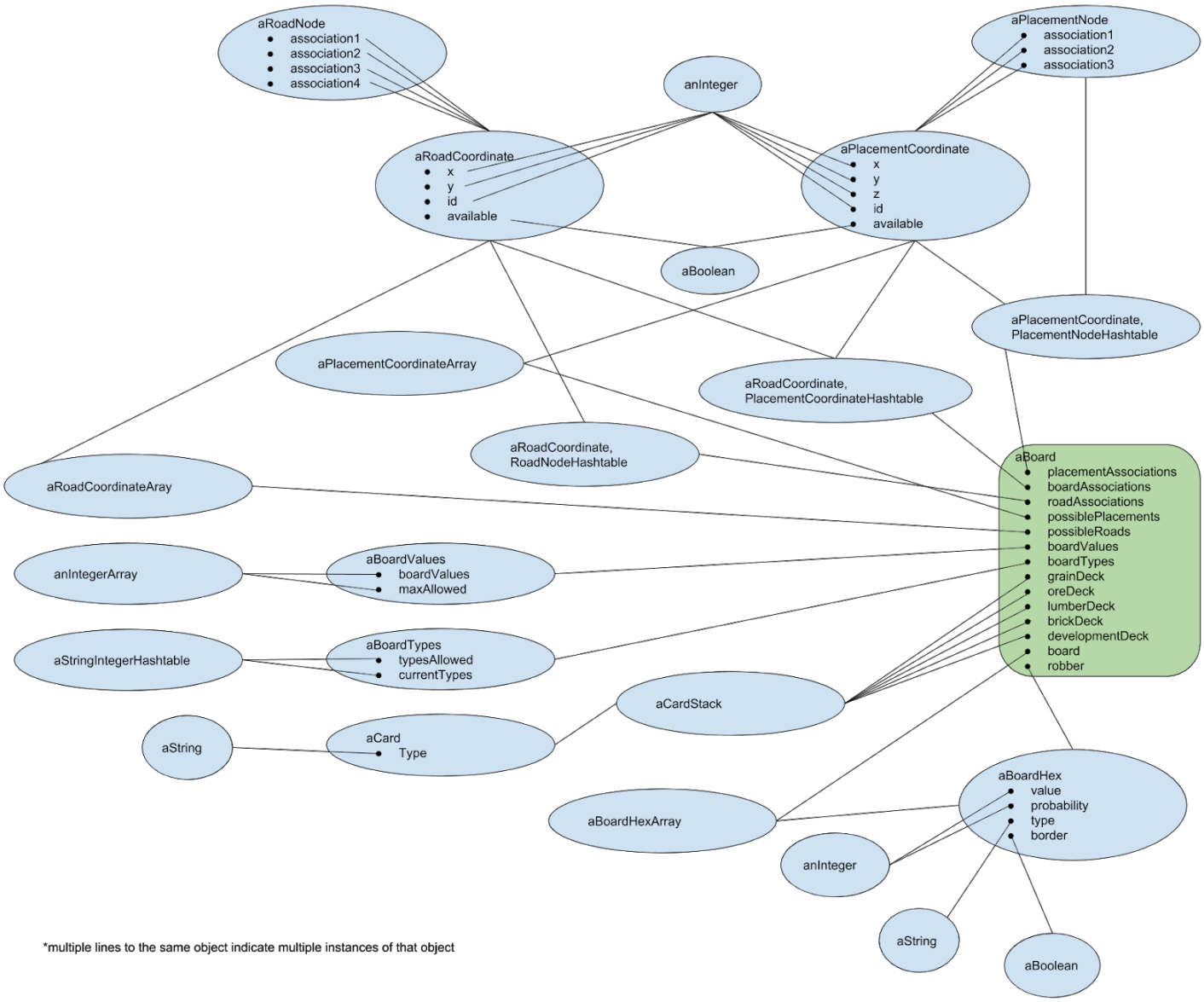
Game Generator

The construction of the game generator seemed simple enough when McKenzie and I first began to think about how we would proceed with this project. In our proposal I outlined the objects I felt would be necessary to create a working *Settlers of Catan* game generator. Unfortunately, it quickly became clear once I began to work that that simple architecture would not be sufficient and that I would have benefitted from taking more time to design the object flow during the proposal stage. However, after much thought I was able to break the game down into three major objects and design the game around them.

1.) The CatanGame object - this object creates and keeps track of both the Board object and all four of the Player objects that actually play the game. The object diagram:

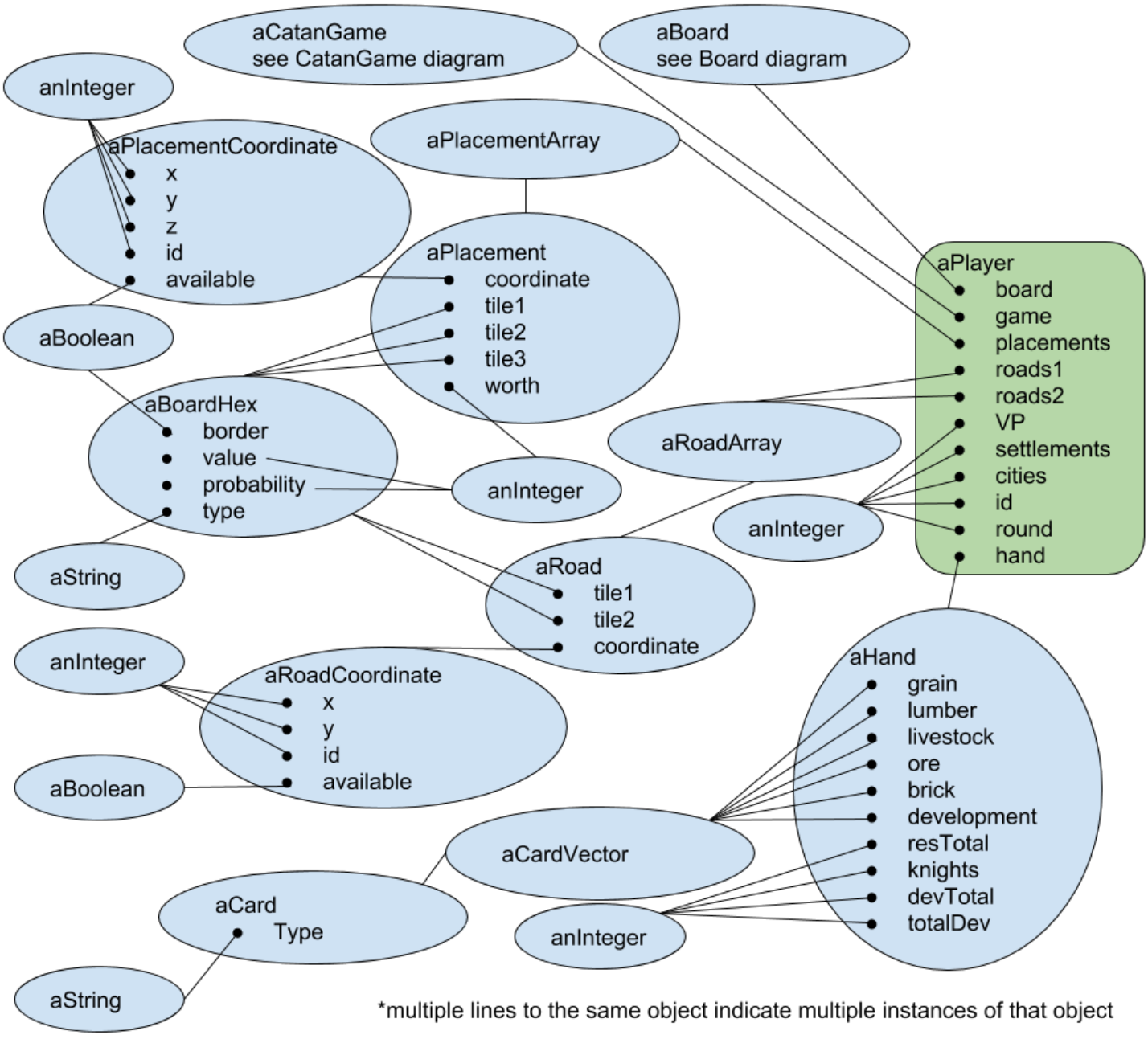


2.) The Board object - this object keeps track of all objects it takes to understand the physical board as well as keeping track of the associations between the possible locations a player can place a settlement/city or a road. The object diagram:



*multiple lines to the same object indicate multiple instances of that object

3.) The Player object - this object knows and follows the rules of the game and uses the decision maker held by its CatanGame object to build and acquire Victory Points. The object diagram:



The complexity of this setup far surpassed what I had anticipated in the early stages of this project. Due to this complexity, there was one object mentioned in the proposal that I left out: ports. Ports can be a very important part of the game, however McKenzie and I both felt that adding them to the design would further complicate our system and would take longer than was worth the effort in the time we had left. We decided it would be better if we left them out and move on to the machine learning aspect of the project. This also means that we did not implement trading with ports, but we did implement trading with the board.

The actual execution of the game is not observable as we only needed the data from these games. When the game has reached a conclusion the program will print which player won and how many rounds it lasted. It writes all of the necessary data out to an output file using code McKenzie wrote. On average, the games take about 58 rounds to declare a winner. The program terminates in under a second, something that neither I nor McKenzie expected. We thought it would be incredibly slow, but our generator doesn't make completely random moves, as we had originally anticipated, like the *Smart Settlers* generator.

The speed that we are able to get from our program comes from the decision maker object. The game holds on to just one instance of the PlayerDecision object and passes it to each player when it tells them to take a turn. The player then gives the PlayerDecision its hand and it goes through and checks what actions, if any, are available. There are only six possible actions a player can take during their turn and the decision maker checks for them in this order: 1.) play a development card, 2.) upgrade a settlement to a city, 3.) build a settlement, 4.) draw a new development card, 5.) build a road, and 6.) trade with the board. This order is to ensure that cards are not wasted. For example building a road spends two of the resources that it takes to build a settlement so we check for the settlement first. This is what makes our program run so fast, it's not wasting cards and accrues points much faster than the *Smart Settlers* generator which actually selects actions randomly.

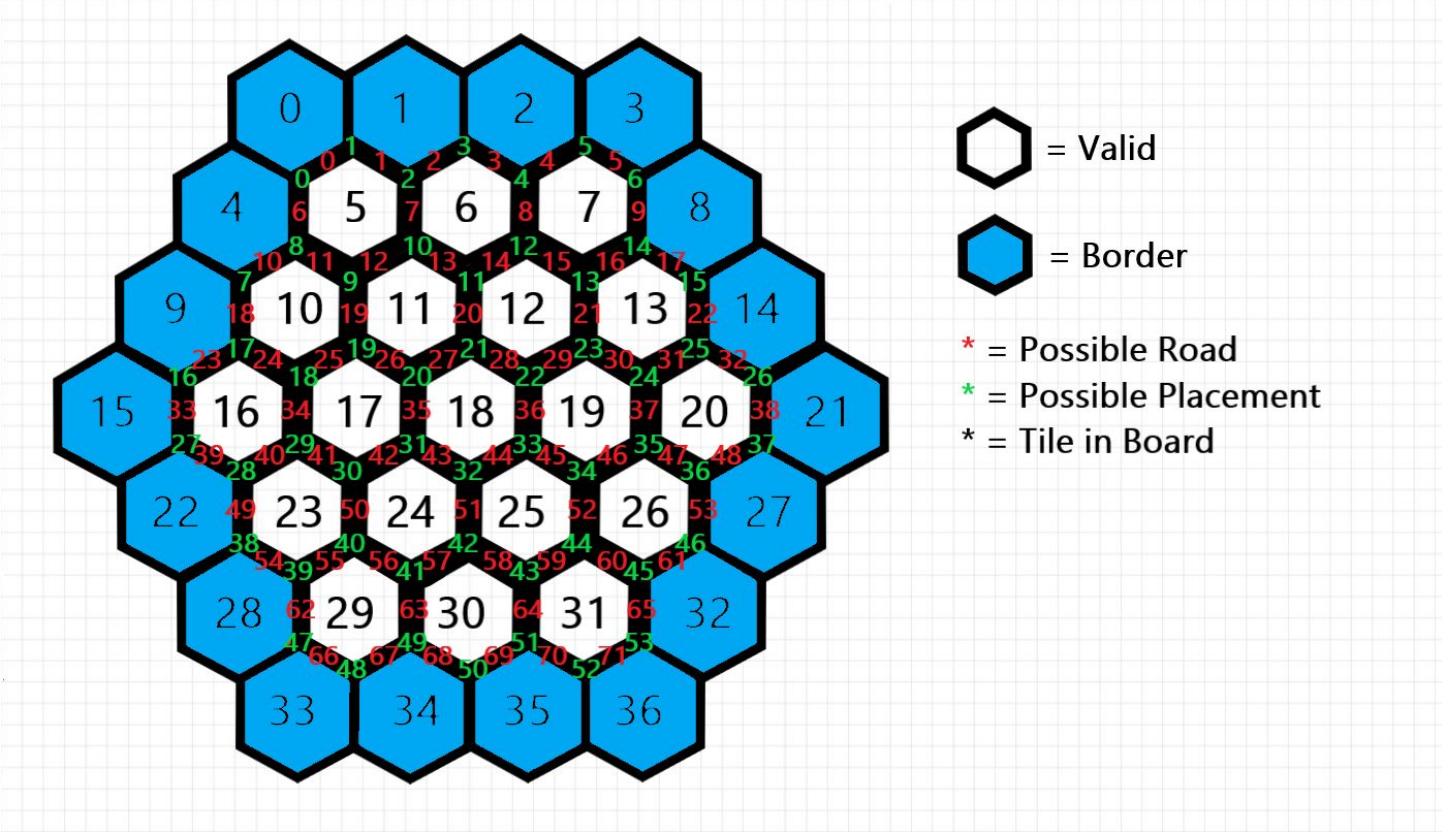
TensorFlow

Given that the vast majority of my time working on this project was spent on the generator, I did not have time to do the extensive research and practice with TensorFlow that McKenzie did. However, I still wanted to learn the technology so I found a simple tutorial of a boosted tree algorithm using TensorFlow and modified it to use our data (https://www.tensorflow.org/tutorials/estimators/boosted_trees). This allowed me to gain a very baseline understanding of how TensorFlow works without delving as deep as McKenzie. It also gave us something to compare with the work that she did. We were able to make comparisons about speed and accuracy.

Difficulties

There were two main challenges I encountered when I built our game generator. The first was how to manage the board in a way that would maintain the associations between roads and settlements and all of the possible locations both can be placed. It is

a rule in *Settlers of Catan* that if a player wishes to build a settlement it must be a minimum of two spaces away from every other settlement on the board including those belonging to that player. This presented a challenge because the board consists of hexagon-shaped tiles so knowing which spaces are next to yours can be difficult to calculate. I solved this by creating three hashtables, one to keep track of the associations between settlement locations, a second to keep track of the associations between road locations, and a third to keep track of the road locations associated with each settlement location. It led to a board layout that looked like this:



Each corner on a tile is a possible settlement location and each edge on a tile is a possible road placement. Each corner has an entry in one of the hashtables that details the other 2 or 3 corners that are directly next to it. The same goes for each edge with respect to its 2 to 4 related edges, and the last hashtable has an entry for each corner that details the 2 or 3 roads touching it. These hashtables allowed me to overcome the inherent difficulties a board like this presents.

The second notable difficulty that I faced was in creating the Player object. My original aspirations saw the Player as a much smaller object. With the emerging complexity of the system however, I realized that it would have to do a lot more. In the end the Player object is what actually understands and follows the rules of the game. This led to an object with over 800 lines of code in it and the decision making process was made into its own object that is used by all the Players in the game. The Player is

by far the most complicated part of the generator and it took a while to get this complexity smoothed out into a working object.

Info Gain

The biggest things that I learned while working on this project were time management, the advantages of taking the time to properly map out a problem, especially one as complicated as this, and a deeper understanding of Java in general. As previously stated, the intricacy of the game generator only grew the longer I spent on it. Looking back, I would have not only set aside more time to work on it but also to examine all aspects of the hierarchy. Part of my mistake was that when I began working on it I build the easiest parts first and didn't put much thought into how these simple parts would interact with the more complicated parts that came later. I found myself going back over code that I had written before and having to spend time modifying it to fit whatever new portion I was trying to code. I would have greatly benefitted from spending more time in the design phase before just jumping in. I might have been able to get the generator done faster that way.

I also gained a greater understanding of objects in Java. It had been several semesters since I worked heavily in Java and I found it difficult in the beginning to remember the specifics of the language. It was immensely helpful to review Java in general and to rediscover some of the objects that I had not had an opportunity to make much use of, like hashtables. This project helped me recement my foundation in Java and object design.

Conclusion

(20pts Total) Pre-Existing Game Generator - McKenzie (**20pts earned total**)

- (10pts) Using game generator found on Github: *Smart Settlers,* review code to understand how the game works. (**10pts earned**)
- (5pts) Get *Smart Settlers* to interact with our data collection methods i.e. modify/create code to output necessary data. (**5pts earned**)
- (5pts) Generate and run 10,000 games and extract data. (**5pts earned**)

(65pts Total) Building Game Generator - Abby~~ + McKenzie~~ (**63pts earned total**)

- (5pts) Create board object (**3pts earned**)
    - (3pts) 19 resource hex objects (**3pts earned**)
    - ~~(2pts) 9 port objects~~ (**0pts earned**)
- (5pts) Create resource decks (**5pts earned**)
- (5pts) Create development deck (**5pts earned**)
- (30pts) Create a player object (**35pts earned total**)
    - (2pts) Victory point count (**2pts earned**)
    - (2pts) 15 road objects (**2pts earned**)

    - ~~(2pts) 4 city objects~~
    - ~~(2pts) 5 settlement objects~~
    - (4pts) city and settlements were combined into one object (**4pts earned**)
    - (2pts) Deck/hand (**2pts earned**)
    - (5pts) Player understand rules of game and legal moves (**10pts earned**)
      ***This aspect of the Player was far more complex than anticipated**

      ***This aspect was moved into its own object**
    - (15pts) Decision making (**12pts earned total**)
        - (5pts) Resource use (**5pts earned**)
        - (4pts) Development use (**4pts earned**)
        - (6pts) Basic trading (**3pts earned**)
            - Trading is 4:1 to the board ~~or with any port object the player has access to~~.
- (2pts) Create robber object (**2pts earned**)
- (12pts) Gameplay (**10pts earned total**)
    - (4pts) Initial set up (**4pts earned**)
    - ~~(2pts) Randomly decide which player goes first~~ (**0pts earned**)
    - (2pts) Initial placement is also random (**2pts earned**)
    - (4pts) Resource allocation (**4pts earned**)
        - (2pts) Allocate starting resources (**2pts earned**)
        - (2pts) Allocate resources (if any) to each player on each die roll (**2pts earned**)
- (5pts) At the start of each round collect data from each player (see Data Collection) (**5pts earned**)
- (1pts) Run 10,000 games (**1pts earned**)

(**15pts Total**) Data Collection - ~~Abby~~ McKenzie (**15pts earned total**)

- (8pts) Scripts will be written to collect data from each player at each round in the game (**8pts earned**)
- (3pts) Collect data (**3pts earned**)
- (2pts) General overview of data to remove outliers (**2pts earned**)
- (2pts) Divide data into training set and testing set (**2pts earned**)

(**40pts Total**) Machine Learning Prediction - McKenzie ~~+ Abby~~ (**40pts earned total**)

- (15pts) Tensorflow research (**15pts earned**)
- (15pts) Write prediction algorithm (**15pts earned**)
- (3pts) Make predictions based on each generator individually (**3pts earned**)
- (4pts) Make predictions based on both generators (**4pts earned**)

- (3pts) Make comparisons about accuracy (**3pts earned**)

Total Points earned: 138
Original Grading Scale: 140pts Possible

≥ 135pts	A
134pts - 120pts	B
119pts - 100pts	C
≤ 99pts	D

Recommended Grade: A