Settlers of Catan

Emily Guthrie, Josh Langowitz, Ankeet Mutha, Brooks Willis

Project and Goals

This semester we created a software implementation of the board game Settlers of Catan. The game Settlers of Catan is a popular multiplayer civilization game for 3-6 people. Each player represents a group of settlers who are racing to expand and develop using the resources at their disposal. Each player has a turn in which they roll dice, collect resources based on the value of the roll, and buy materials to aid in their expansion. They can build new things onto their civilization during their turn and also can engage in trades with other players for desired resources. We wanted to create a means of playing this game without the need for the physical board and pieces, and ideally without needing all the players in the same room.

At a minimum we wanted our version of the game to include background python code to establish rules, game play, and interaction between players as well as a graphical interface to visualize the python code. The board would display graphically the different tiles' values and resources, as well as where players had built on the board. Hands would be displayed and players would interact with these during game play. Assuming completion of the minimum goals we had for the game, we had several additional features we wanted to add including a server with multiple lobbies, a mobile phone application to allow for private viewing of hands, and a means of keeping track of statistics on players in the game

Our final product is a python game that is playable on a web interface, which will allow it to be easily expanded into our reach goals in the future. We have all of the basic functions of a game of Catan, which was our intended minimum deliverable.

Final Refinement of the design:

Our project had a unique design process due to where we foresaw issues arising in development. We all felt confident that the work required to make the game objects and game play function would be manageable and that we should gear our design toward more advanced final deliverables like the server hosting, mobile application, and statistics tracker.

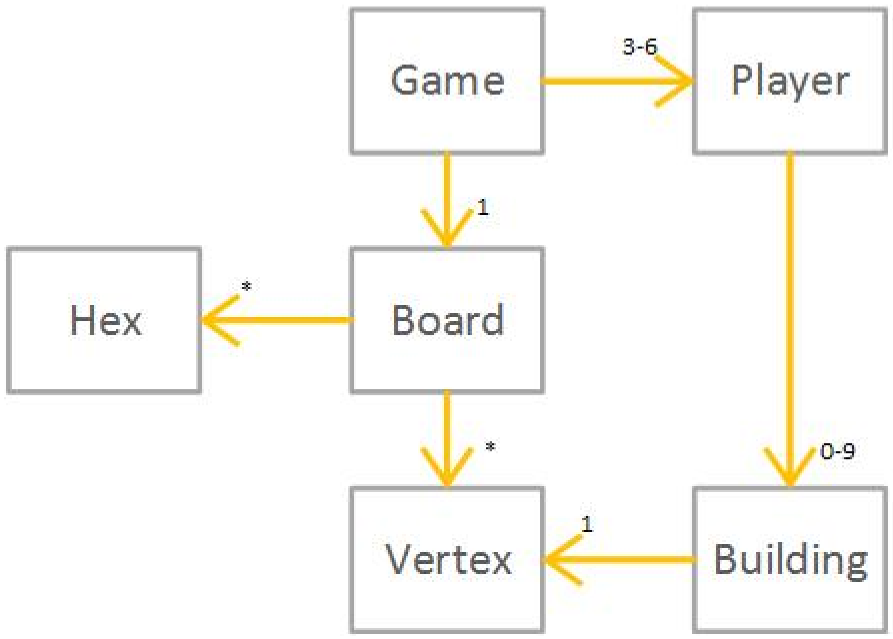
Our first priority was to provide a web server to play the game online, ideally with multiple lobbies to allow more than one game to be played at once. The drawback was that it required us to acquire more knowledge external to the scope of the course and that it didn't demonstrate what we learned in the class. As a compromise, we hosted it locally and used a service called ngrok which tunnels to the internet and allows the game to be accessed by other people.

We contemplated making a mobile web application that will allow each player to view their hand of cards individually on their phone. The idea was that either a group of players will get together and display the board on a computer or projector, using phones for their hands, or each player will have the board and his/her hand on a computer, allowing the game to be played online at a distance. The problem was this would also involve creating a more involved trading system that allows several trades to occur at the same time or a queue to keep track of the order in which trade requests are made. It also was going to require us to really expand beyond the python knowledge we acquired in the course. Because of these issues, we completely avoided implementing this idea.

Finally, we thought we would like to keep track of statistics about the game, such as what resources players get and use during the game on average, and how that compares between winning and losing players. This requires us to have a persistent database and since we're hosting it locally we decided not to do this.

Our ultimate reason for scaling back, however, wasn't related to the difficulty of our advanced goals. Our main impetus for refinement was that the minimum value product we selected was actually very difficult. We knew that there would be lots of labor to get the code working but we assumed that there wouldn't be large logical challenges. Early on in the project we realized that laying out the board objects was an algorithmic challenge and that the relationship and referencing between our game objects would be a difficult problem. It's for these reasons we focused on our MVP with the few additional things we discussed before.

Our greatest challenge has been figuring out the various gameplay elements of Catan and how best to represent them with Python data structures in an organized and readable fashion. The main elements of a Catan board are hexes, edges, vertices, ports, and buildings. We quickly realized that the ports and edges can be tied to vertices, but that still left us with hexes, vertices, and buildings to deal with. We thought about making a hex object that keeps track of it's own edges and vertices and also the neighboring hexes, but we determined that would make it hard to deal with vertices and edges that belong to more than one hex. For now, we have decided to go with a board class that keeps track of where everything is and a class for hexes, a class for vertices, and a class for buildings. Because edges can only have roads, really we don't need to keep track of them, as a road just needs to know which 2 vertices it connects to. We have a grid system so that the board knows which vertices and hexes go together, and the players will be able to interact with the individual vertices and hexes on the board. We ended up with 6 classes in total, four of which directly represent pieces of the game and two of which are higher level classes used for organization. The direct representation of game pieces boiled down to Player, Building, Hex, and Vertex. Player contains all the information about that player and the methods used for trading, and resource management. The two high level objects are Game and Board. Game contains a Board, as many Players as are in the game, ports, and all of the methods dictating rules, turn order, and other higher level function, including building buildings and playing dev cards. Board, as you might expect, has all of the Hex and Vertex classes, as well as the functionality for setting up the board at the start of the game. The full class diagram can be seen in figure 1 below



(Figure 1 This figure shows the full UML class diagram)

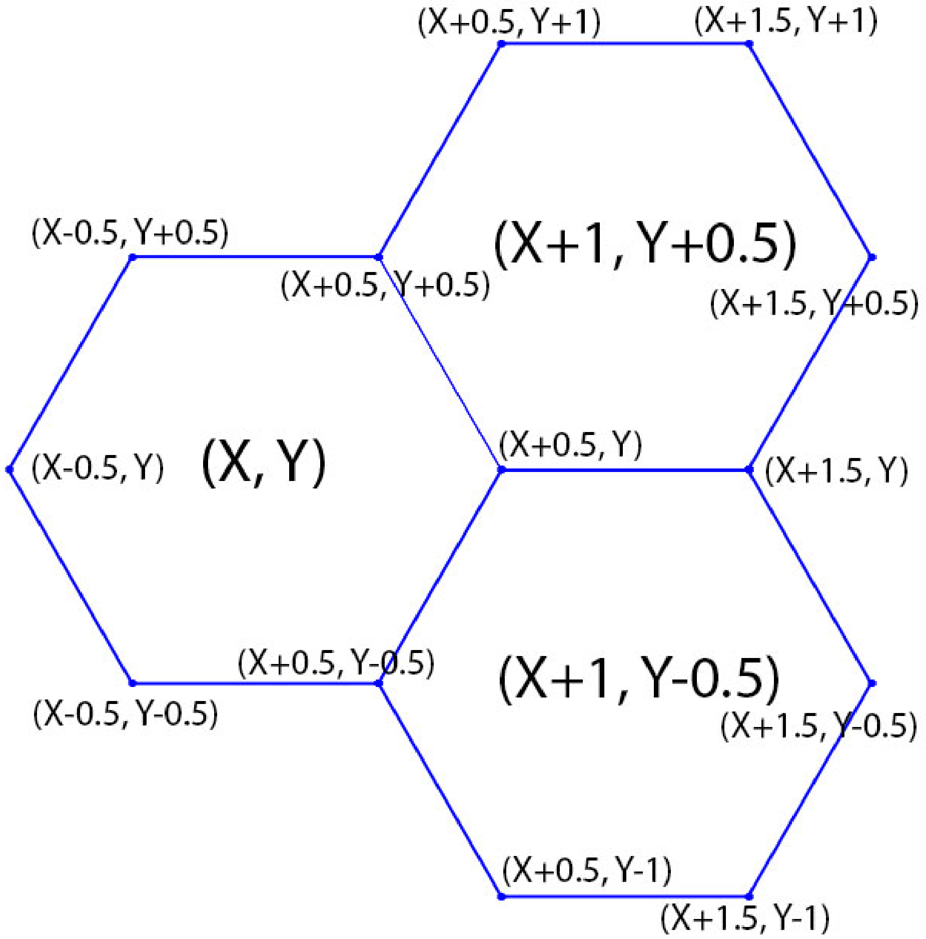
The next problem we encountered that changed our design plan was the code required to work with the javascript in the GUI. Our original class structure had a board object that was supposed to hold the information about the board but it didn't have all of the functionality to communicate to the javaScript in the GUI. To fix this we redesigned our system to have a game class that inherits everything and provides wrapper functions that allow us to execute all the functions for game play together and to communicate with the GUI better.

When working with our project, our language has been mostly based on all of the Catan terms. All of the people who are working on this project know how to play Catan and understand the language. We have been using Catan terms to dictate most of what we are doing. For example, we have several new classes and functions like board, player, hex, drawDev, buildCity, etc. We initially had some difficulty transferring this knowledge over to python in some cases. Trying to create a hex grid instead of a normal grid required us to change each vertex on the board into a system of points described by the hexes next to it and their distance from the center of the board. Additionally, we also could not use hex as function, because it already exists in Python. But we have not had many other language barriers. For any Catan player and programmer, the code is simple to understand.

Final Product and Success

Our final product as stated above is a playable version of settlers with most of the functionality we deemed useful. We widely consider it to be a success. The refining of our design goals really helped us realize how much work we did to make a successful project.

We have 1-3 fairly complicated algorithms for our game, all of which we were able to successfully implement. The first one builds the boards, assigns coordinates to all the hexes and vertices, and creates references between adjacent vertices and hexes. Our coordinate system is an x, y coordinate system with staggered columns of hexes, it is illustrated in figure 2. Moving up or down within a column of hexes changes the y coordinate by 1, and moving to a column to the left or right changes the x coordinate by 1. However, columns that are next to each other use offset coordinates, so moving from one hex to an adjacent hex to the side changes the (x,y) coordinate pair by (1,.5), with signs depending on the direction of movement. The center column of hexes defines x=0, and the center hexes in columns with an odd number of hexes define y=0. Vertices are defined in a similar way, with columns of vertices having x values in between the hex columns the divide (so they are ...-1.5,-.5,.5,1.5…), and moving up and down in a vertex column changes the y values by .5, rather than 1 for hexes. The figure below (figure 2) shows a small section of board and the relative values of hex and vertex coordinates.



(Figure 2 The figure shows how our coordinate system works in relation to a set of hexes)

Once the board is created, this coordinate system allows easy identification of which hexes connect to which vertices, so we go through all the hex-vertex pairs and reference them to each other. We then go through each vertex to find it’s neighbor vertices. Getting the vertices in the same column is easy, but finding out whether to go left or right for the third vertex is a bit tougher. First, we check to see if the x coordinate, rounded down to an integer, is even. Then we check if the y coordinate is an integer. If neither or both of those are true, the correct neighbor vertex is to the right. If only one is true, the correct neighbor vertex is on the left.

The second major algorithm was the calculation for longest road. In short, it needs to take in a list of roads owned by a player and find the longest unbroken path through those roads, and who has the longest path. To do this we used a modified version of a breadth first search, since typically BFS looks for a specific result and does not record its path while we are looking for a path not an object. We chose breadth first over depth first (or another search method) because it allowed us to explore all paths at the same time and immediately knock out ones shorter than the longest. This greatly simplified the algorithm as a whole. We did not implement a feature that causes roads to be interrupted by opposing cities due to time reasons, but we believe that can be implemented fairly easily

Our final addition was to setup and use ports for our board. In this case, we didn’t try to algorithmically define where the ports should be because they should be in the same place for every board of one size. We hard coded in the list of the port coordinates and then using a list of possible ports, randomly assigned coordinates ports. Once the setup was done, if an individual settles a vertex with a port on it, the trade functions are able to update the conversion rates that that player has for bank trades.

Reflection on division of labor:

Our division of labor worked fairly well. Our project has a number of substantial sub-sections which we were able to write in parallel, such as the server, various classes, trading, longest road, etc. Early on we established naming and data structure conventions for almost everything we would encounter, this also allowed us to have a large amount of autonomy. Each of the team members was good about working independently and finishing assigned tasks on time. This allowed us to only have minimal face to face meeting time, which was used to update everyone on progress and re-evaluate schedule and tasks that would free up potential choke points.

The chunks we divided the project into were small enough that integration stayed minimal in most cases, except in between the backend and the server. This step of integration took significantly longer because a fair amount of work needed to be done in each before any form of integration was possible, even-still we were able to rapidly fix the problems because of our initial communication about data structures.

One of the strongest parts of our team’s dynamic was that we were able to work in parallel on specialized areas while being knowledgeable enough of every system to jump in as needed. We specialized based on what we liked to work on the most and what we most wanted to learn. Josh specialized in the gui work but played a crucial role in helping the communication between python and js work. As such he worked a lot with Ankeet, who worked mainly on game play functions like assigning turns, doing trades, and tracking state variables, all that required communication with the js and gui work Josh did. Ankeet’s work required communication with Brooks because the functions Brooks worked within the player class and the longest road function were very important for the functions that kept track of score and turn. Emily's work with the board setup and defining board object types worked heavily both with Ankeet and Josh because they needed to know how to access the objects and their attributes to write their functionality. This way we could be working on our own without being too isolated.

This is what lead us to be very proficient debuggers. We easily sat together and debugged the code as we developed it by providing scaffolding to test functions as we made them. Our other great trick was breaking up tasks for each other as we debugged. One person would open up the game and test functionality. Another person would have the python scripts open and another the js scripts. When something would break, the tester would read the error out and then the js and python debuggers would start working to figure out if it was on their end or in the communication between their ends.

Bug Report:

We did not run into any individual large bugs, just a multitude of small bugs. We did were religious about incremental debugging and going through logical checks of the code at regular intervals. As mentioned earlier we set up a number of conventions early on, which reduced the number of errors from incompatible data. Whenever we needed to integrate two larger pieces of code the two people who wrote those sections would sit down together and sort out all issues on the fly. This is one of the areas where we believe we performed best, our debugging method fit the project and our working styles exceptionally well and minimized what had the potential to be a large problem.

Analysis of the outcome:

We were able to reach our initial minimum viable product, but not much further. Going into the project we greatly underestimated the amount of work that would be required to implement the numerous rules and features of even a default game of Catan, which is the reason that we were only able to reach what we had originally called our minimum viable product. Were we to do this again, where we ended up would be a fairly high up goal, and a more bare bones game without trading, ports, or an automatic longest road would be the MVP. Everyone on the team put forth a considerable effort, which enabled us to get as far as we did, and we feel that we got as far as we reasonably could have given the timeframe of this project.

Reflection on your design:

The best decision we made was to re-scope our project early and often as we learned more about the requirements for making a full game of Catan. Had we been unwilling to re-scope we would have spent a lot of time chasing pieces of our original goals that weren't as important, and given the large scale of the project this likely would have caused us to either not finish or to be forced to sacrifice quality.

We could have done a much better job of file organization. We originally planned to have only a few files that contained all of the code, thinking that this would simplify searching for things. That was wrong, we ended up having a fair bit of trouble with our few scripts getting very long and unruly. If we were going to start over, we would have a separate file for each class, and possibly a few others containing major functions such as dev cards or longest road.

Git URL: https://github.com/JLangowitz/Catan.git