Javatan

By: Trevor Crystal, Christopher Lamberston, Chuanfeng Xiong, Ruben McWilliams

Overview (Trevor)

Catan is one of the most popular board games out today. It was originally created by Klaus Teuber and released in Germany in 1995. Since then, its international popularity has skyrocketed. Many different expansions have also been released. Some allow for more players to join the game. Some add massive new mechanics while others are simpler variants or scenarios. For this assignment, we just focused on the base game.

General Base Game Rules

The base game involves three or four players attempting to settle on an island called Catan. This is done by placing settlements and roads, and collecting resources. For each settlement you have, you have one victory point. Settlements can be upgraded to cities which are worth two victory points. Roads are built to reach new places to build new settlements, as there must be at least one empty space between settlements. The first to reach ten victory points is the winner. Each turn starts with a dice roll done by the player whose turn it is. Two dice are rolled, and the values rolled are summed up. Each tile (except for the desert) has a number two through twelve on it (except seven). For every tile with the number equal to the dice roll sum, any player with a settlement adjacent to that tile will receive the corresponding resource. If that settlement is a city, then two of that resource is received. If a seven is rolled, then things work a little differently. First, any player with more than seven resource cards must discard half of them. This does round down, so a player with nine resource cards must discard four of their choosing. Then, whoever rolled the seven gets to move the robber to a tile, which accomplishes two things. First, the player may steal a random card from one player with a settlement or city adjacent to that tile. Second, the robber will stay on that tile until another seven is rolled, and while it is there, no player may receive resources from that tile. There is another way to move the robber without rolling a seven, however, and that is with the knight development card. You may use it to move the robber when it is your turn. Development cards in general are another thing you may spend resources on, and knights are just one of three kinds. Another kind is progress cards, which help you with building more roads or receiving more resources. The last kind is victory point cards, allowing for another way to receive victory points aside from settlement and city building. There is actually one more way to receive victory points: special cards. There are two different special cards. The first is Longest Road. This is given to the first player to have a continuous road that is at least five segments long. If another player builds a longer continuous road than the player with the Longest Road card, then that new player receives the longest road card. The Largest Army special card functions similarly to Longest Road, except it is based off of the number of knights used, and the number a player must use for it to be initially claimed is three. The only other major feature I have not yet

mentioned is trading. When it is a player's turn, that player may offer to trade resource cards with any other player. Also, that player may exchange four of any one of a different kind of resource card. Additionally, if the player has a settlement located next to a port, then they can use that port for an even better trading rate. That is as brief of a summary of the fundamental rules of Catan as possible. The full rule manual is around 15 pages long, and can be found here along with rules for the various gameplay expansions.

Contributions Overview

Initially, we set out to implement all of the features of the base game. We all met and collaborated on deciding what the core classes should be named and what functionality they should contain. The original idea was that we would all work on all aspects of the implementation (the core, GUI, networking, and documentation). As the semester progressed, we realized that we were all too busy to divide the work in that way, so we ended up mostly focusing on different chunks. Ruben focused on the GUI, Chuanfeng, focused on the networking, Chris focused on the core and stitching the different parts together, and Trevor focused on simpler parts of the core, along with documentation. The repository is available on Github, and while it does not show who contributed to what parts of the documentation, it does show who contributed to what exact parts of the code. It can be viewed here. TGCrystal is Trevor, rmcwilliams is Ruben, JumpingKangaroo is Chris, and MaxXiong666 is Chuanfeng. For some reason, some commits do not have a listed author. These appear to be commits made by Chris.

Implementation Differences

While we had initially planned for our implementation to have full feature parity with the base game, we did eventually realize that this was way too in depth for us to handle given the amount of time we had. We decided that not only was it easier to not hardcode every single rule, it would also make it more enjoyable for casual play. During a real life, over the board game, it is possible for a player to make a mistake, whether intentionally or unintentionally. That kind of thing can make for a much more interesting and exciting game between friends. In terms of features not included, we decided that trading resource cards with other players would probably be the most difficult single feature to implement, and thus it fell to the bottom of our priority list. The robber, trading with ports, and development cards were also time consuming in terms of connecting their functionality through all components, although the core code does contain support for them. The core does check to see if the robber is blocking a tile, and roads do store data about whether or not a port covers the same location. Players can also have development cards, but there is no implementation of their functionality. Those are the main differences between our implementation and the actual game.

Core (Chris)

The core of the application is where most of the game logic is held. This can be considered the "model" of the application in a traditional model-view-controller system. There are a few different main parts of it, with varying levels of complexity. Board is the most complex,

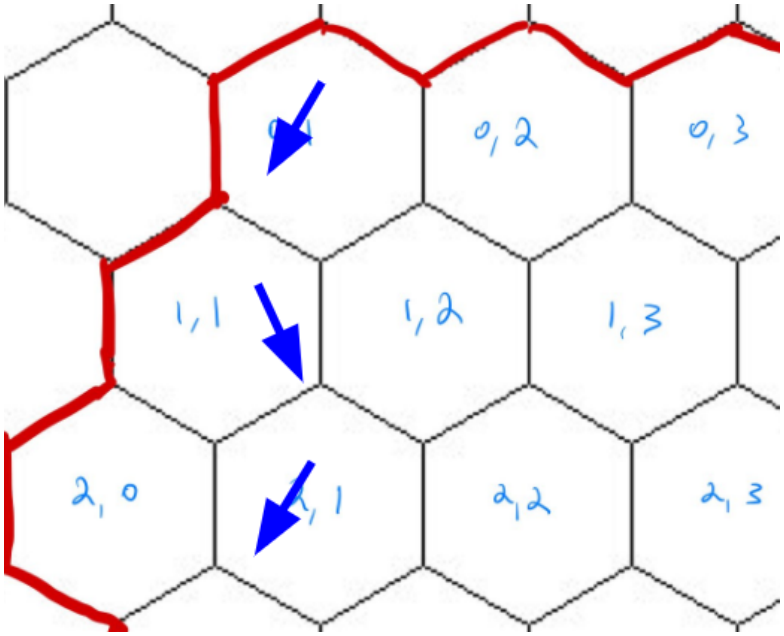
and handles the storage of the hex board along with checking the validity of road and building placements. Player keeps track of a player's resource cards as well as victory points. There are some other classes that are used to keep track of different game mechanics, such as DevelopmentCard, Building, and Card. These do not contain much logic, and are mainly used by the other classes for storing data. This section will go into greater detail some of the logic used in the core classes.

Board

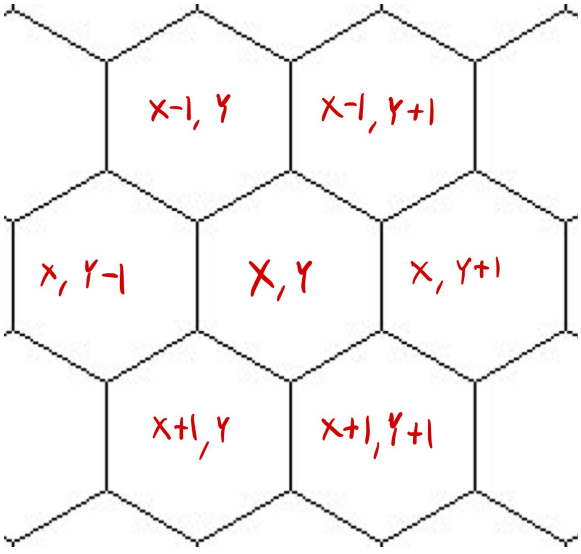
The main work in implementing a Catan board in Java is that a Catan board is made up of hexes. Since these are not square and have six sides, they cannot be trivially stored in arrays. To design the storage and access of the Catan board, first we figured out what we would need to access from what parts of the board. We formulated these requirements from what operations would need to be done in the normal running of a Catan game. We determined that we would need to be able to access the roads adjacent to a settlement spot, access the settlement spots around a given hex, get the buildings adjacent to a road, and get the roads adjacent to a road. After translating this to a graph, we determined that we would need the ability to: access the adjacent vertices given a vertex, access the vertices of a given hex, and access edges adjacent to a given edge. Additionally, we wanted edges and vertices to be shared between adjacent hexes so that we didn't need to do any copying between hexes to update roads or settlements.

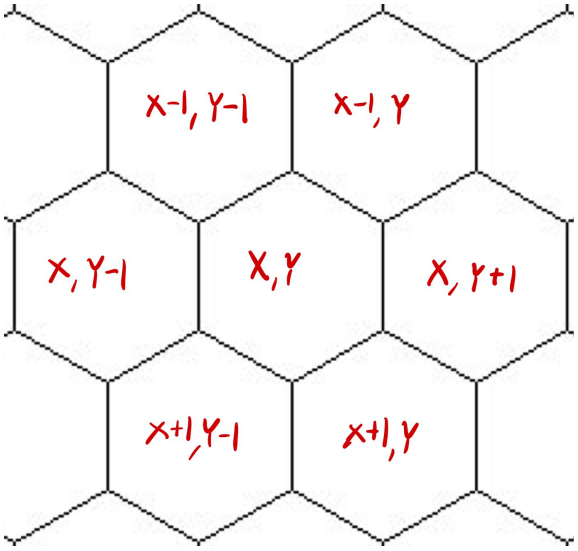
To do this, we considered a few different options. First, we considered having each hex store references to each of its edges and vertices. Each vertex and edge would also need to store the adjacent objects. This was quickly discarded, as keeping track of what referenced what would have been very complicated. Additionally, this would likely have required a lot of manual linking of adjacent hexes and edges, since programmatically assigning references with this approach was difficult.

Eventually we arrived at our final answer. We would store the hexes in a two-dimensional array, with each column being slightly offset. This is best illustrated in the below figure. On each hex, you can see its (x,y) coordinate. The x coordinate is the row, and the y coordinate is the column. The rows are numbered as expected, however the columns are not. The top left hex for example, is actually in column 1. This is due to the board being a slightly rounded size. The blue arrows on the diagram show the hexes in column 1. As you can see, column 1 is offset slightly between each row of hexes. This results in the rows with an odd x coordinate being offset compared to the rows with an even x coordinate. This will be important later, since this affects the equations needed to translate between hex, edge, and vertex coordinates.

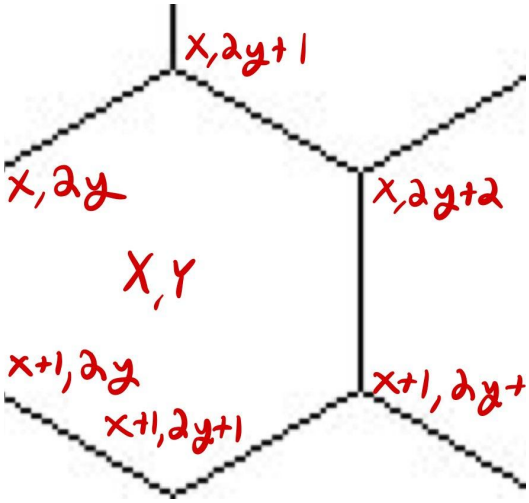


From this, we were able to determine equations to get adjacent hexes from even and odd rows. Even though this wasn't strictly necessary, this made it easier to get equations for edges and vertices that allowed us to have shared vertices and edges between hexes. Note that as you can see in the two separate diagrams below, even and odd rows require slightly different equations. The top image shows the equations for adjacent hexes in even rows, and the bottom image shows the same for odd rows. The adjacent hexes on the same row don't change between the two, however due to the offset odd rows need to subtract from the y index and even rows need to add to the y index.



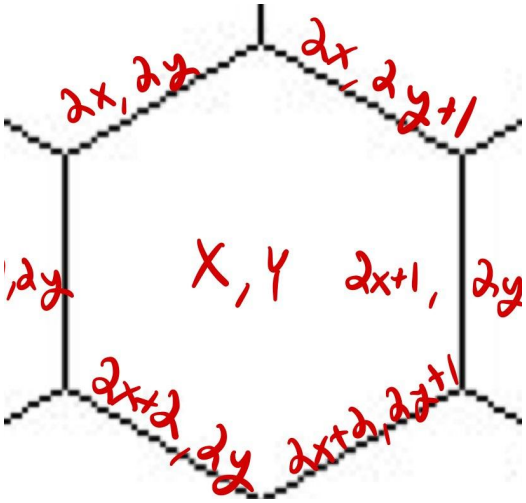


Next, we looked at how to store settlements (Buildings). These vertices should be shared between adjacent hexes, so we determined to use the hex coordinates to generate vertices. Since odd rows are offset from even rows, this meant that we needed slightly different coordinate conversion from odd and even row indices. From here, we arrived at the following conversions for even row hexes. For odd row hexes, the equations are very similar. One adjustment needs to be made for the y coordinate. On odd rows, we determined that we needed to subtract one from the y coordinate. Below are the equations for even rows, so for x rows we used the same equations with an offset of -1 to the y coordinate.



Finally, we needed to determine equations for edges. These are similar to the vertex equations, in that for odd rows an offset of -1 is necessary on the y coordinate. Here, there is also the added complexity of vertical edges. These are indexed in such a way that only every

other spot in the array is used. This allows us to maintain the sharing between hexes, even though there are half as many vertical edges as diagonal ones.

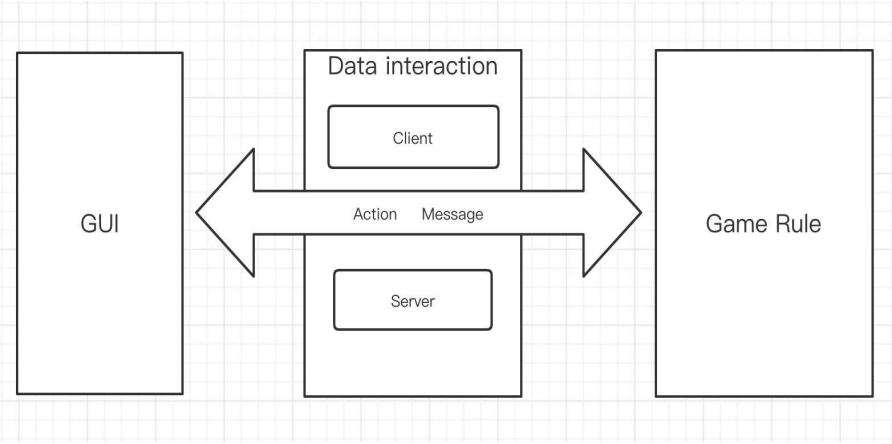


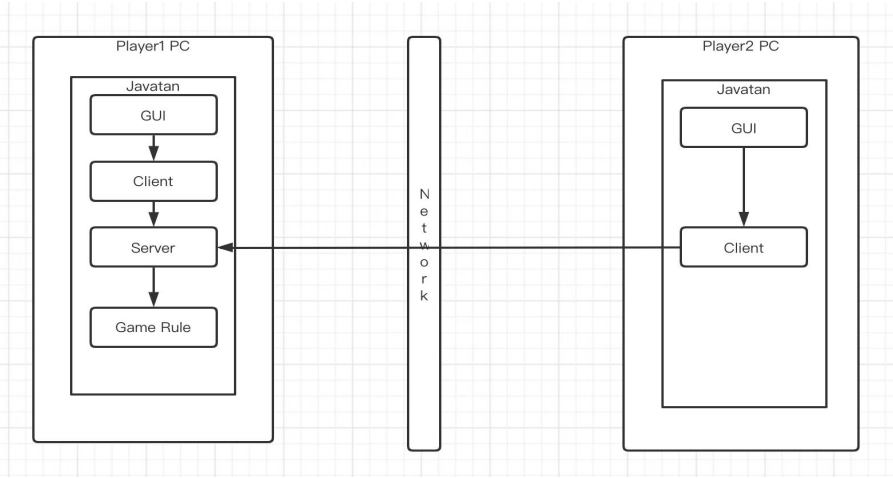
A key part of implementing the Board algorithms and data structure was testing. Comprehensive tests of adding roads, buildings, and gathering resources helped ensure that the implementation was correct. This also allowed us to test further improvements and iterations of the code and ensure that the core logic was still correct. For example after adding the subtraction of resources from a player to the purchase of a road, we were able to quickly run unit tests to verify this did not introduce any new bugs into the functionality already existing.

After figuring out the board storage, we implemented the operations to add roads and settlements. These operations used the aforementioned equations in order to check adjacent vertices, edges, and hexes as required by the game logic. For example, when adding a settlement it needs to be adjacent to a road of the same type, and it cannot be adjacent to any other settlement. Thus, when adding a settlement the Board class will check adjacent roads to ensure it is next to one of the same color, and check adjacent vertices to make sure there are not any settlements already there.

Networking (Chuanfeng)

An ideal structure of data transfer would be as depicted in the following graph.





As a board game with four players, there is a data interaction module that takes in a move made by the players and gives back the result of their movement depending on how game rules are implemented. Server and Client mainly do this work. Ideally (may not be the case currently) Client plays a role as an agent for each player. It is at the front line of receiving input from players and sending them for processing. Containing a “player” object, it keeps track of the identity of a player and his or her possessions.

Server plays the role as a spine of the program. It receives information and processes it by calling certain functions, makes changes to the Game object it stores which is the ultimate version of the game situation. It is also supposed to use a command line interface while the client uses a GUI.

Action and Message are objects that are implemented as packages of information. They are transferred between Client and Server. Both Client and Server can take out from or put in information into Action or Message. Action takes more

concrete information than Message. For example, when a player makes a movement, an Action will be formed containing what kind of the movement is (Building a house, adding a road, etc) is, or more information like what is the index of the movement. For now I have made an Action code such as 1 represents building a house, and 2 represents adding a Road. Server as a receiver would also have the same code for each movement. Action along with more information, for example playercolor (which is actually player identity) will be packed into Message and sent to Server. When Server receives the Message, it unpack this Message and get information. Then functions will be called according to the information. Updated information will be packed and returned to Client. Message can also carry player information like player color so that we can set which players actually receive certain updates.

Multithreading is used in Server. Once a Client connects to Server, a new thread would be created for accepting and handling Message from this Client. As a game usually with 4 players, multithreading can help avoid potential risks.

When testing whether Message can be transferred between Client and Server, a error was reported as

“writing aborted; java.io.NotSerializableException”

After looking it up It turns out that it was because Message contains core objects and they are not initialized with binary compatibility, which will trigger the error these objects are going through networking. So I had to make almost all the class implement Serializable and the test ended up passed.

For future works on networking, as the game becomes more complete and complicated, Action and Message would surely be implemented to carry more kinds of movement.

GUI (Ruben)

The GUI was implemented using JavaFX, it consists of some math to construct the hexagonal grid layout, along with the location of buttons and roads. If a user clicks on a location where they want to place a settlement, a settlement will appear, they can also place a road by doing the same thing. Settlements can also be upgraded to cities. I used Paint.NET as well as some icons/images I found online for the settlement icon, the city icon, and the resource cards. Also the GUI code consists of function calls used by other portions of the application to update the GUI the user sees. Examples of this are making a particular road visible, making a road not visible anymore, upgrading a settlement to a city, updating the amount of a resource card a user has, putting numbers on hex tiles, changing the color of a hex tile, and changing the color of a road.

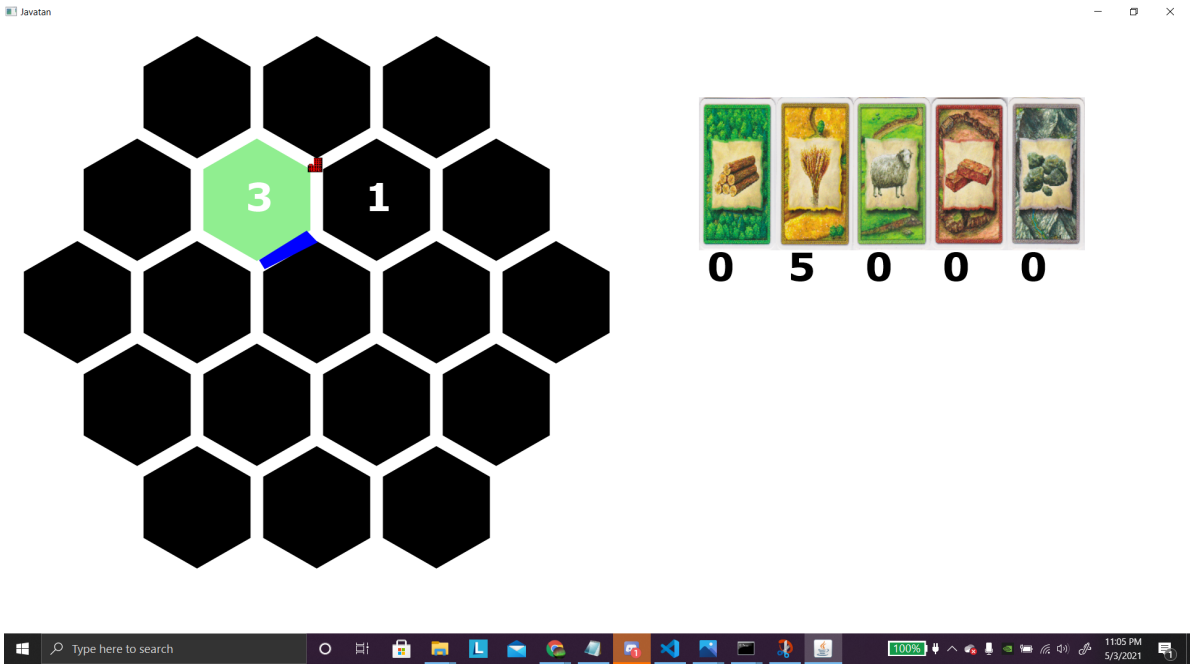
For future works, I would like to implement tokens on the board, harbors on the sides, the robber, domestic trading, maritime trading, development cards, progress cards, knight cards, text that shows the user who won the game, and text that tells the

player what color they are. For the robber icon, I'd probably just make one myself using Paint.NET. I'd probably place victory point development cards below resource cards and place progress cards below that, then finally knight cards. For harbors, I would look up exactly where they are supposed to be placed and when the user goes there, a popup window would appear that asks them which card they want to exchange for. I would also implement a trading with other players UI. open a side panel that is only accessible by the two players performing the trade and would allow them to negotiate a trade between what they have. A bad design for the trading UI would be a separate window as I think that would be annoying for the user. The other players would have to wait until the other two players have completed their trade to continue playing the game. As a last step in development after all of the functionality is implemented, I would focus fully on looks and make it look as professional and seamless as possible. I would probably utilize some of the CSS skills I have to do this since I might as well borrow that web development knowledge since JavaFX works with CSS. Also, one thing that could also definitely be improved is the responsiveness of the GUI. Currently, it is of fixed width and height, which might be a problem for users who have very small laptops or are using it on a tablet. It would be nice to have it be responsive so that everything adjusts proportionally in size based on the size the user has the window. This would conflict majorly with the current design as it relies on exact coordinates for the placement of things on the display. We would have to either do some math to make sure it adjusts proportionally or do some research to see how other people have implemented similar concepts.

Implementing the GUI was challenging as I had to do a lot of geometry that I had long forgotten since high school and some of it I actually implemented by clicking the sides of the polygon (I did this for the roads). The way I did this was I temporarily made my code so that it would output the coordinates that I clicked in the console, this allowed me to get all four corners of each road to construct the polygon. This worked reasonably well although definitely tedious but a lot of time GUI development ends up being like this. The result was a little bit ruggy looking but it got the job done and if I had more time I would definitely smooth them out a bit either by using a different approach or just very carefully calculating the exact coordinates of each side of each road. If I had more time to work on this, another thing I would do is look into how this GUI development process can potentially be sped up using tools out there in Eclipse or other software. I have heard that Eclipse has a drag and drop editor for JavaFX which sounded interesting to me and something that I would like to try. Oracle also provides a tool called Scene Builder (https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html) which looks very powerful and from what I've heard speeds up development by a lot. Another really cool feature of JavaFX that I liked was that you can actually utilize CSS when using it which is nice for someone like me who has done a lot of web

development. One feature of JavaFX that I actually didn't like was that sometimes it is a bit tedious when you want to make polygons. For example, when I was making the hex tiles, I found out that there isn't really a way to simply tell JavaFX that you want a hexagon, you have to do the math to calculate where the corners would be based on the center and do it this way. I think this is extremely tedious and if I would make a recommendation to JavaFX's developers it would be to add easier ways to add common polygons. However, I still think JavaFX is a huge step up from Swing so I still prefer it. In order to run JavaFX, we include a library called "javafx-sdk-11.0.2" in our directories. This allows JavaFX to run, if you are interested to see how JavaFX gets run, you can see the commands we call in our scripts related to it.

Overall, the GUI development process was largely trial and error as GUI development usually ends up being. I learned a great deal from this project. Below is a photo of the GUI:



As you can see the design is definitely simplistic but I definitely would add much more and polish it of course before the game would be released.