Chad

Team WareWolves

A Simple and Elegant game of Tactical and Strategic depth

Ben Goodwin, Joshua Munoz, Josiah May, Luis Rodriguez, Miles Wood

Challenges and Lessons Learned

Table of Contents

Challenges

Server/Database

- Getting MySQL server connected
    - We had issues getting the mysql java connector working independently in the maven build. Leveraging code from our CS314 experience, we adapted the methods and structure to work with our new database, but had to build default returns and values for queries and updates that didn't affect any rows in the database.
- Defining the database tables and columns
    - When we were given MySQL access we quickly defined a working database of tables(users, invites, activeGames, and pastGames) with columns that we thought covered all our bases. As we continued development we had to add additional columns and expand varchar ranges to fit the whole system as it evolved.
- Working with Java NIO packaged and IO together
    - For the client side NetworkManager we used old Java IO packages with plain old sockets and data streams, while on the server side we used newer Java NIO packages that utilized SocketChannels, Selectors, and SelectorKeys. Getting these two different packages working together proved challenging at times and created issues that weren't necessarily foreseen or needed.
- Defining and finalizing NetworkMessage classes
    - Early on in the project we defined a set of NetworkMessages that we thought covered all needed cases. As we continued to work on finishing the rest of the system, additional cases were found that we needed to cover and we continuously added more NetworkMessage classes. Additionally, as we fleshed out the presenter and views, changes to already set concrete NetworkMessge classes needed to be made to accommodate what was needed in the rest of the system.

Presenter

- Communicating with the Model, View, and Network Manager
    - As the system became more detailed the number of messages increased as well. For each message some knowledge of the logic for other components was needed to send the appropriate information. This required communication

between team members to fully understand the implementations of the other components.
    - A message factory was created in the beginning for creating view messages from the presenter and view. This factory was unused in the end of the project as it was outdated and did not allow for the creation of the messages in the way that the implementation called for.

UI

- Building two different User interfaces concurrently.
    - Building the Command Line and the GUI at the same time proved challenging to keep them both in sync. For example, as the development furthered there were changes necessary for the ViewMessage classes that needed to be updated in both.
- Scanner issues
    - Java's Scanner threw an exception for a while stating there was data in its buffer already while it was being invoked for input. The Scanner had to be moved into its own thread for this to work. We still do not know why this exception was being thrown.
- Learning proper coding practices with Java Swing
    - Most of the team had little to no experience with Java Swing which meant we needed to learn by the Java tutorials. While those tutorials could teach us how to uses ActionListeners, JTable, and Layouts, all the tutorials assumed that all the logic would be in one class. So, making ActionListeners events pass information to other JPlanels had to be done by chaining message between several JPanels. There has to be a cleaner way to do this but we could not find it.

AI

- Integrating java and python
    - I wrote the AI in python in order to take advantage of the great packages available there, but because of that executing the python from a java game client is system dependent, and only works if the packages are in exactly the right place. This made it impossible to make the AI run in the client for offline play, instead it runs as a separate "user" logged into the system.
- Problem scale
    - The opening move of chad has 56 possibilities, and the second move has up to that many retaliations. Because of the effective branching factor of the problem it is impractical to fully traverse the problem tree when trying to discover the best next move.
    - Training a neural network to approximate the value function for the game was the only option, but to train a network requires a large number of games to be played

to generate data to train against. This took a lot of time and the resulting network is not all that good at playing the game.

Game Logic

- Interfacing with the controller.
    - The Game package was designed on its own and with the idea of sending the game Strings around as the only communication out of it. This caused problems when we were trying to then decode those Strings in the interface layers, and possibly a lot of code duplication. We solved this problem by moving the Point class out of the Game package and into the Client package so that it could be used in both places as a way to translate strings to integers.
- Proving the correctness of the game model.
    - There are a lot of edge cases in the Chad game and all of them needed to be tested for before the model could be used correctly. This required manually creating game board situations and the valid moves that should result from them then using these as unit test cases.
    - I was still finding new edge cases throughout development that we had not thought of, for example initially walls were all considered to be the same as we thought that a piece could not get from the other wall into a castle. This caused a problem because a Queen, moving on the diagonals, could capture pieces in the other castle. This was fixed by separating the walls and checking only the wall associated with its own castle.

Lessons Learned

Server/Database

- Pick NIO or IO for networking interactions, not both.
    - Translating between the two became a larger issue than it ever should have. Picking one package from the start would have made the networking part of the system more coherent and easier to maintain. Picking NIO and sticking with it would also have made the networking component more future proof.
- Define Networking Messages and MySQL Tables after the larger system is closer to final
    - The database tables should have been defined once the greater system was defined well enough to play test, not before.
    - The network messages should have been defined once the presenter and view interactions were better defined and not before. With the network messages being changed throughout the end of development there were issues of coherence and getting the network package behaving as it should.

Game Model

- Define the representation of the data first
    - We ended up finding a problem with and then changing the representation of our data in the middle of the development process, meaning that the implementation of that had to be changed. If we had thought through all of the consequences of out representation and defined it better in the planning stages it could have avoided that.
- Defining the game model first makes it much easier to create the rest of the system.
    - We ended up defining the entire game model before finalizing other parts of the system, but even if it had taken longer defining an interface that the rest of the system can assume works even if the logic isn't there yet makes the development of the rest of the system much easier.

AI

- Try and keep all the parts of a project in compatible languages.
    - I had several issues in integrating the Java game code with the Python AI. Also, because I wrote the AI in python I had to recreate the Game logic in both the Java and Python, leading to code duplication and potential problems if the core game logic changed in one place and not the other.

- A machine learning AI for a game like Chad needs a large amount of game data to train, and generating that data has been a slow process, leading to the poor performance of the AI agent.

Presenter

- The development of the messages to the Viewer should have been created alongside new features in the GUI/CLI
    - Each message should have been added to the system when a new feature was created instead of creating them all as a whole in the beginning. This would have allowed the messages to not send unneeded information to the View or to the Presenter. This would have also allowed us to avoid creating messages that were ultimately unused in the end.
    - The message factory should have been created near the end when implementation was understood to better cater the message factory to the needs of the Presenter and View.

UI

- Prioritizing the complete functionality of an interface one at a time
    - Both interfaces were being built concurrently. This wasn't an issue given we were using the same messages for both interfaces, but this made us spread out our resources. Whereas two people would have finished one UI, we had one person working on each UI.
    - Going of the above stated point in Presenter, completing one UI (or getting the skeleton designed for it) would also have helped with creating/modifying any new messages that were needed. This could then contribute to the second UI that we would decide to create.
- Java Swing is not best UI for a pure MVP model
    - By requiring the presenter to be useable by two UI's, we need two message systems. One for messages between the UI elements and the presenter, and another for all the messages sent and received to the server. This meant for most message sent or received we needed the UI controller to read the message, act on the messages information, and then pass the message to the presenter. The presenter then needed to read the message again, act on the messages information, and pass a new message based on the first message to server. This made the presenter and Swing controller both need logic for acting on the same message. This could be done cleaner in the Swing controller, but the MVP model did allow us to make to UI models.