Group #5: The Programmers Who Say C
Members: Tyler Poff, Mike Shoots, Travis Brown,
May 4th, 2015
Final Report for COSC 340: Software Engineering

Fill In The _____!

A Multiplayer Card Game App for Mobile Devices

Final Report

Table of Contents:

1. Introduction: **2**

2. Customer Value: **3**

3. Technology: **4**

4. Team: **8**

5. Project Management: **9**

6. Reflection: **10**

1. Introduction:

Fill in the _____! Is a multiplayer card game app for mobile devices similar to existing card games such as Cards Against Humanity. This report is to document the progress our team has made during the development of this app as well as give a final update on the project's status.

Our project is one that our team greatly enjoys working on. It involves many design problems that we have personally never faced before and constantly challenges us. During this iteration of our development cycle there were several major accomplishments that were achieved. The most notable accomplishment during this iteration of the development cycle was the development of the ability to have multiple concurrent games with multiple players each running their own copy of the client software connected to a single server hosted on Amazon Web Services. This networking update was the single most important update we have applied to our software during development. It allowed us to truly say that our project was a true multiplayer game.

During this phase of development most of the overall changes were made to the server aspect of the project. There were some minor changes made to the client software, however these were changes were mostly refactoring of code and adapting the server client communication protocols to accommodate the inclusion of multiplayer. The only major inclusion for the client was the ability to choose from a variety of game lobbies offered by the server.

The server had to be completely redone in order to add support for multiple players and concurrent games. The previous server architecture was ill suited for multiple clients connecting at once and we were forced to completely rewrite it. This lack of foresight on our part caused the team to fall behind schedule and as a result there were some elements from our initial design that were cut from the final version of the project. The features that had to be removed included plans to implement multiple game modes, the ability to dynamically create custom games that could be hosted by the server from the client, and an instant messaging system that would have allowed for players to communicate in game.

Although we were unable to include all of the features originally planned for this project, our team is satisfied with the product that we have developed over the past few months.

2. Customer Value:

Our original customer value was to create a game that could be played anywhere, set up quickly, and provide a fun social experience for a wide audience. Unfortunately, due to complications with the server development we fell behind schedule and were forced to cut the following features from the final product:

1. Instant Messaging System [removed 2016-04-08]
2. Monarchy Game Mode [removed 2016-04-22]
3. Dynamic game creation [removed 2016-04-24]

The most important feature that was cut was the dynamic game creation. This is a game that was intended to be fun to play with strangers, but even more fun when playing with friends. Although it is still possible to join the same game as a friend, it would have been much easier with the ability to create a private game and send an invitation.

An in-game messaging system would have contributed to the social factor of the game, but was ultimately cut from our final product. As we mentioned previously in our status report, a chat system would impede the core gameplay on a mobile app.

The other feature we decided to cut was the third game mode. Monarchy mode used a completely different voting system where only one player each round was allowed to cast a vote. At the end of each round, the player who submitted the winning card would become the “Monarch” who votes in the upcoming round. We realized that this would require additional communication from the server and additional screens to show the player which role they were playing each round. As we were already short on time, we decided to prioritize other features.

There were also some undefined features that we had hoped to explore, but had not officially stated that we would implement. These features included a more advanced particle system, a more detailed method of displaying the voting results, and more animation and movement for the various sprites that were drawn onscreen.

Although the removal of these features from our final product was undesirable, our team feels that our final product is still an enjoyable experience. We feel that customers can still get substantial entertainment from what we were able to complete.

3. Technology:

For this iteration of the project our team primary goal was implementing multiplayer functionality. This feature was considered the key aspect of our project as we wanted this to be a multiplayer app. As a result the majority of the work done over this iteration has been to the server side code, as it would be responsible for connecting players together. Although this multiplayer aspect was the primary focus there was some new features added to the client code as well.

3.1 Client Architecture:

There were only a few minor changes to the client architecture. In order to implement multiple concurrent games we had each game hosted on it's own port number on the server. In order for the client to connect to individual games we had to adjust the networking code to link to a variable port number.

We added 2 new classes the FindGameScreen class and the ProfileCreationScreen class. Each of these classes represents a new screen for the client. The FindGameScreen shows a list of various games that the client can connect to. The player can select an element from the list and then connect to that game. Fig 1 shows a screenshot of this screen. The other class that we had to implement for the client is the ProfileCreationScreen. This screen allows the player to create their own unique profile to use in the game. The player can adjust their profile name, icon image and icon color. These features allow each player to customize how they appear in game. Fig 2 shows a screenshot of the profile creation screen.

Fig 2 - This is the profile creation screen. This screen allows players to create their own unique profile to be used in the game.

Fig 1 - The find game screen for the client. It lists all games that the client can connect to.

Another minor addition was the ability for players to dynamically leave and join a game. This feature required a new message to be passed to and from the server. The

implementation of this message was fairly straightforward, the message would indicate a player is leaving and indicate the player's id, the server would search through its list of players and remove the player with the corresponding id from the list.

During early documentation many different screens were put into the plans, but were cut to keep the user experience simple and easy to pickup. A figure of all planned screens, both implemented and not implemented is presented in fig.3, To help prevent users from getting lost navigating the features of several screens were either merged into another screen, or implemented in a popup menu.

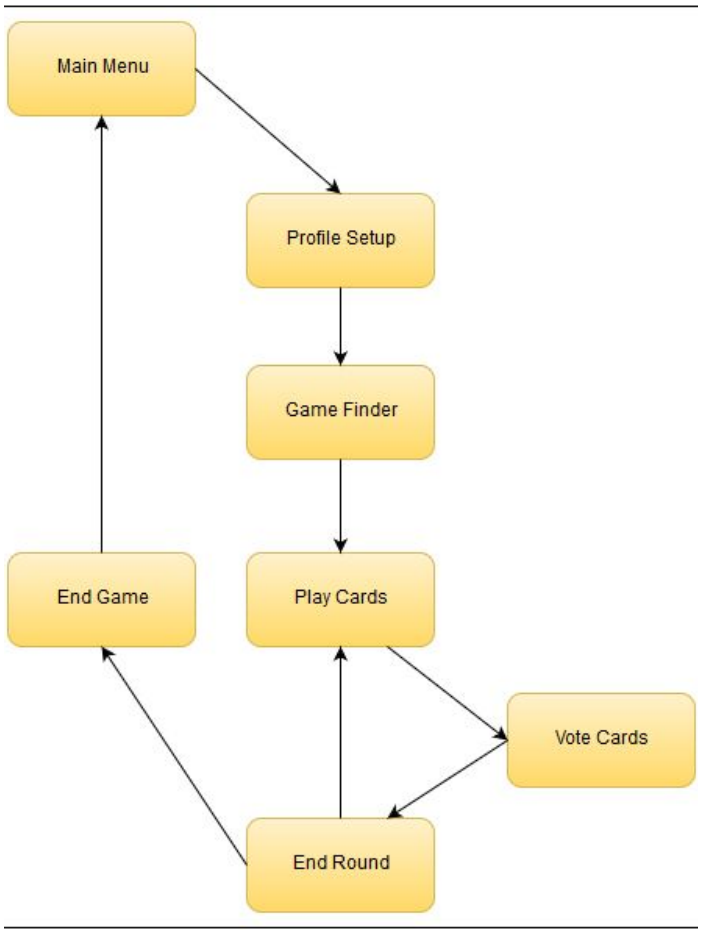

Fig 3 - The above diagram illustrates the flow of screens in the final Unity client. What is not shown is how there is a popup menu button on every screen to take the user back to the main menu at any time.

3.2 Server Architecture:

During this phase of the project we were forced to reprogram the server code from scratch. Our previous version of the server proved to be ill suited to handle multiple players in a single game. This ability to have multiple players playing the game together was the core of the project and so when it became apparent that the server architecture wasn't suited for this task we had to redo it.

To accommodate multiple players and multiple games, we created threadable classes to allow simultaneous play. Although we were starting from scratch, we stuck with a similar class structure. We had a Game class to run an instance of a game, and a PlayerThread class for each client that connected to the game. We also used another class for the AI to mimic a PlayerThread and a separate file to handle connections and start the games.

***PlayerThread* class**:

The PlayerThread class is the server side representation of a connected client. The class keeps track of all data associated with a player, including profile information, current cards, and the socket object associated with that player. This class functions as the "controller" of a MVC model, handling all interaction between the client and the server during a game. When the client sends a command, the PlayerThread class updates the appropriate game data. It also sends the client updates regarding the game state.

***Game* class**:

The Game class is the "model" of a MVC model. It maintains the game data and progresses the game state based on commands from the PlayerThread class. It maintains a PlayerThread object for each player currently connected to the game and manages all the data for the current game state such as card availability and game phase. It also keeps track of all the game settings such as game type, score threshold, and its assigned IP address and port number. We didn't have time to develop the client communication necessary for player-created games, but the interface is established to allow for customizable game creation.

Run_Server.py:

Run_Server.py is used as the insertion point for the server. This script creates 8 instances of the Game class with predefined game parameters. The script then begins each Game thread and then exits. In the original design this script was going to be far more complex. It's duties included creating a small pool of games with A.I. opponents at

start, handle dynamic game creation and deletion, and when a client connects it would have been responsible for sending the information for games that are currently being run. These features had to be cut due to time constraints however.

3.3 Testing and Debugging:

The setbacks during development led to a rather rushed testing phase. Each member incrementally tested the product while developing it in order to ensure that each module was working correctly and there were no unintended side effects that would impede other modules. This testing was done throughout and proved very effective at locating bugs in the client and server provided we were thorough in our examinations.

Initially we were going to develop a series of scripts to test the maximum capacity of players that the server could handle. However, due to time constraints, we were unable to implement these tests. However our team is confident that our server architecture can handle a large quantity of players.

Despite the lack of time to implement proper stress testing we were able to do some player testing on a small scale. We were able to gather a small focus group consisting of 8 players that were willing to test our product. The feedback from this test group was very positive. They highly enjoyed the vibrant color scheme and praised the overall aspect of the game. However there was some negative feedback mostly regarding the way that scores were displayed on screen. Several testers commented that they would have liked a clearer explanation of what was expected at each stage of the game. Despite this lack of intuitive direction in the game all playtesters were able to quickly grasp the concepts behind each of the game’s phases and all were quoted as having fun while testing the product.

4. Team:

For this last iteration of the project our duties remained unchanged. Travis remained the primary server programmer, Mike was the primary client programmer and Tyler was responsible for the graphical component of the client as well as general programming for both the client and server. The team as a whole feels that each individual member contributed equally to the project. Each member contributed to the project in their own way and it was a combined effort of the entire team that resulted in the completion of this project. As a team we are satisfied with the way this project turned out.

5. Project Management:

During our initial report we outlined several goals for this project that we had hope to accomplish as well as a schedule of the milestones that outlined when we desired each aspect of our project would be done. This initial schedule included an additional buffer week that would allow our team to recover should we fall behind. However we encountered a severe problem with our plans for the server late in development that caused us to fall behind schedule.

These problems arose during the implementation of multiple players in a single game on the same server. In order to implement multiple players we decided to allow each player to have their own thread that would handle message passing to and from the server. However this multithreaded approach introduced multiple deadlocks and race conditions that we did not expect. Any attempt to fix or debug these problems introduced new errors. It was decided that in order to effectively solve these problems we would have to redesign the server from scratch, keeping these problems in mind while doing so. This redesign caused our team to fall an estimated 3 weeks behind schedule.

This delay in development meant that several features that were originally planned for the project had to be omitted in order to meet the deadline for the final product. Despite these setbacks and the removal of some aspects of our project, our team is happy to report that we were able to implement the vast majority of features originally outlined for this project.

6. Reflection:

Throughout this project our team encountered several difficulties pertaining to the project and organizing the group as a whole. Looking back there were several elements of the project that went very well and others that could have been improved upon.

6.1 What went well:

Both the client and server are reliable and robust. Despite some complications during implementation of some features the final product turned out to be a very elegant, lightweight but dependable system. With the current system that we have developed we could easily expand this project to include the features that had to be removed due to time constraints with relative ease. This system could also be ported to a variety of different platforms thanks to the Unity's cross platform capabilities.

During the course of development our team slowly learned how to work together to accomplish certain goals. With the nature of a networked app it was necessary for the various team members to coordinate the various messages to and from the server and ensure that both pieces of software interpreted them correctly. This required both planning and communication and the team was able to deliver on this important aspect of the project. The team also collaborated effectively to create the various documents required for the project.

Despite having little experience with some aspects of this project, such as networking and multithreading our team adapted quickly and learned these concepts as we coded the project. This ability to learn and integrate this new information quickly and efficiently was a key part in the success of this project.

6.2 What did not go well:

One of the major challenges of the project came from trying to develop a client and server at the same time. In many cases implementing the network commands required both programs to support the command before it could be properly tested. A way to help test and develop the server faster would be to a create a python version of the client that would behave just like the Unity client. The python client would not have any of the UI or advanced features that the client has, but would be purely command line based, which would allow for quick testing during development.

The server had the very difficult task to achieve multiple games and multiple players each with their own network connection. This introduced a type of problem that our team had little experience with: concurrent programming. During our initial planning phases, we were focused on the more immediate future with the idea that we could incrementally add features to our existing code. This shortsightedness ended up costing several additional hours of trouble and forced us to cut some features we had originally planned.

The Amazon Web Service we used for this project provided a workable solution, but not an optimal one. The services available from Amazon are all priced based on usage, which works great for a small project on the scale on this one. However, in order to create an account Amazon requires a valid credit card on file in case our data usage exceeds the free tier limit. This opened up security risks for a shared login that we could all have access to. If we had chosen a service like Heroku, we could all have access to test changes over a network instead of running everything on a local machine.

Another challenge the team faced was proper documentation of code. It was often the case where one team member would code a large portion of code but fail to add comments explaining what the code was supposed to accomplish. This led to confusion over the role of certain functions and classes and made debugger code written by other members difficult as they had to first decipher what the original intent was.

6.3 Was the final project a success

Despite the difficulties we've encountered this semester while developing this project, we have a product that is not only playable, but enjoyable as well. We have all accomplished something that we had not done before: we have created a multiplayer game that runs on multiple machines controlled by multiple users at the same time. With these accomplishments in mind, our team considers this project a success despite the inability to complete all of our intended goals.