Test Revision 0

Safwan Hossain `hossam18`
Tyler Magarelli `magarelt`
Eamon Earl `earle2`

March 11th, 2022

1 Introduction

Our methodology for testing will involve a few different approaches, tailored to the nature of our project and its use cases. The basis of our testing will also relate to the architecture of our program, where different methods will be used for the model, view, and controller respectively. Ultimately, our goal will be, as it should be with all software testing, to achieve full branch and condition coverage, and ensure our program is valid.

2 Methodology

2.1 FR Testing

For the model, we will start with unit testing, likely with a combination of boundary and partition tests, including system response to invalid inputs and error states. That way we can ensure that our controller is contrived in such a way that any risk of errors is avoided and designed around. This also allows us, once the data model classes and their methods are verified in their behaviour, to treat said classes as black boxes when analyzing our sequences and control flow graphs from the controller.

As stated, the controller will be modelled via a control flow graph that includes relevant and auxiliary methods in the view and data. In this way we can visualize both the general and specific sub-routines of our program and ensure full coverage of statements and sequences.

2.2 NFR Testing

The view, being purely aesthetic and the most important part of the program for the user, will be tested experimentally, where both the developers and eligible end users unrelated to the project will interface with the game and evaluate the presentation of information on the basis of both its clarity and its format. We will use this feedback to make any necessary changes.

For the server, we intend to engage in exploratory and experiential testing, with an emphasis on stress testing. Once again, we can allow end users to try playing with each other online, and rate and describe the experience. Regarding the stress testing, we intend to explore the fidelity of the server both on LAN and over the internet, by seeing specifically how many players can connect and engage in the game without causing notable latency issues, within the bounds of some metric that will be determined by the development team in conjunction with opinions from eligible end users.

3 Test Tools

For unit testing we will be using JUnit which allows for automated testing. For stress testing the server we will generate large number of virtual clients and make them all connect to a single server to observe the effects on performance. For system testing we will execute the application at specific states of the game to ensure functionalities are working correctly.

4 Extent

The entirety of the project will be tested.

5 Data Recording

Each different testing method (e.g., system testing) will record its results in its own testing log which will contain information such as person running test, purpose of test, results of test, and a summary of what the results reflected.

6 Business Functions

- At startup the program will display the main menu which will present the option to start or connect to a lan server
- When connecting to a server the user will input the server's IP address and the program will try to establish a local network connection to that address
- The server host will be asked to set up the game rules (e.g., minimum number of chips, max players)
- Once a player gets to zero chips they will be given the option to leave or continue to watch the game
- The game states will change in orderly fashion according to the rules of poker (e.g, the river cards cannot be presented before the turn or flop cards)

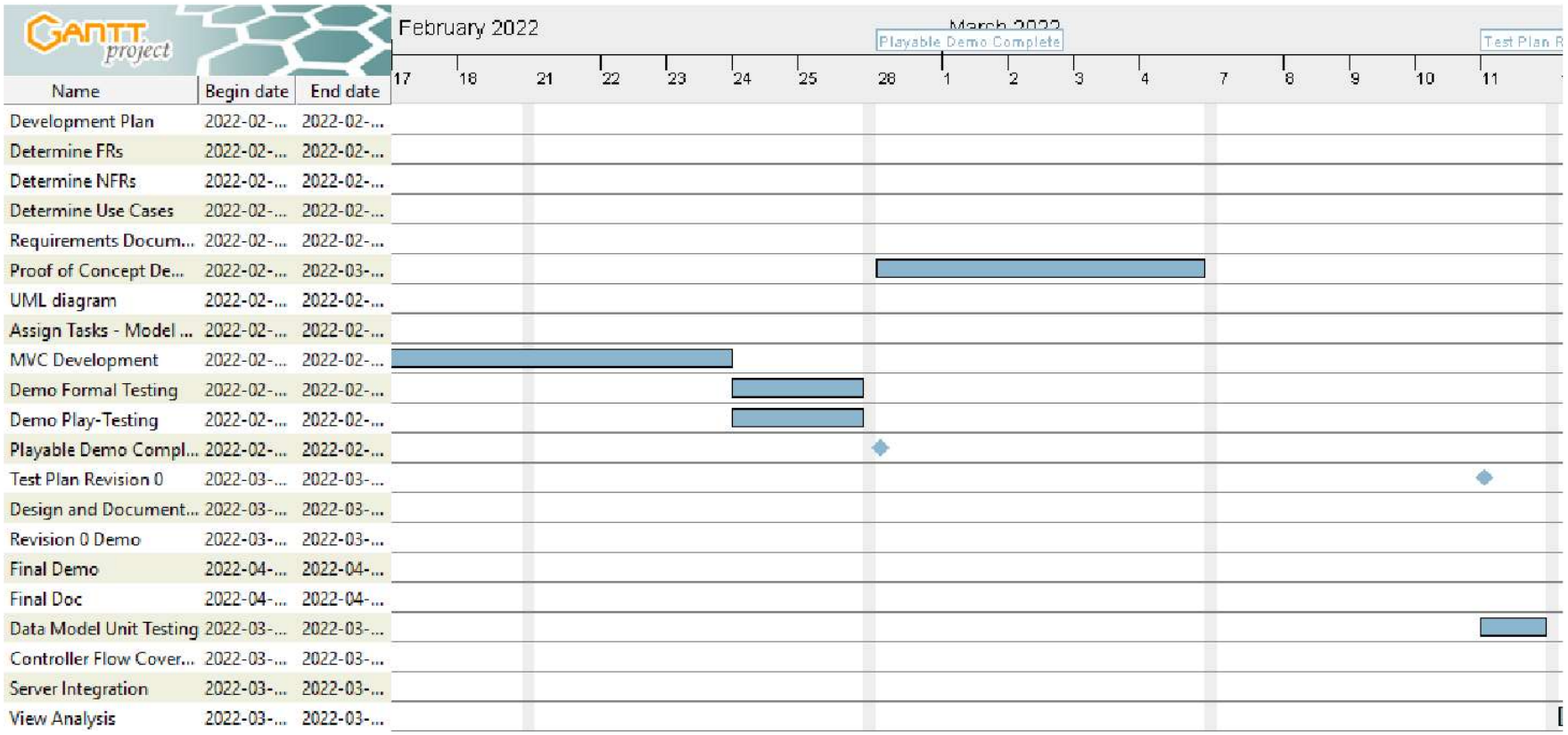
- At the beginning of the game each player will start with the same amount of money and be given 2 cards each
- Only one player can have a turn at any given turn (no two players will have a turn at the same time)
- If the host leaves or closes the game then the server will shut down and all players will be kicked out

7 System Tests

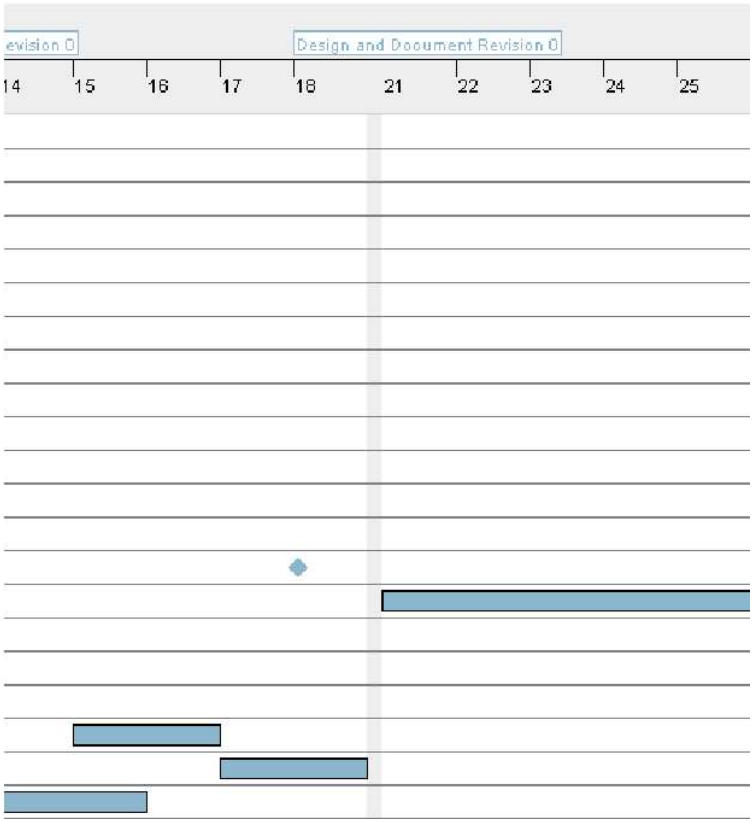
When testing the entirety of the system we must consider the varying states of the program. Luckily, due to the cyclic nature of card games, we truly only have a small handful of looping states, with an exit condition achieved in a finite number of rounds. As such, the main overarching system tests we need to fulfill are basic turn based choices, namely folding, raising, or calling the bet. They all start from the same state, and this process is initiated by the server when it is time for the player's turn. Calling and raising take separate transitions but lead to identical states, namely a stalling state until their next turn or until the round ends, while the folding option enters a similar state and only exits when the round ends. This represents the overarching behaviour of the system, and affects the individual state of the data for said player. In terms of the specific system tests on the data model, we have previously discussed the methodology we will use, but a specific example for the draw_hand(int n) method will be shown:

- Test 1: Starting state: a full deck of cards
  Input : n = 52
  Expected Output and End State: An array with all 52 unique cards present in some random order, and the stack pointer s is equal to 52
- Test 2: Starting state: stack pointer at index 32
  Input : `n > 20`
  Expected Output and End State: IndexOutOfRange Error reached, caught and the remaining cards are output
- Test 3: Starting state: any stack
  Input : n = 0
  Expected Output and End State: Empty array returned and stack pointer unchanged

8 Gantt Project



Name	Begin date	End date
Development Plan	2022-02-...	2022-02-...
Determine FRs	2022-02-...	2022-02-...
Determine NFRs	2022-02-...	2022-02-...
Determine Use Cases	2022-02-...	2022-02-...
Requirements Docum...	2022-02-...	2022-02-...
Proof of Concept De...	2022-02-...	2022-03-...
UML diagram	2022-02-...	2022-02-...
Assign Tasks - Model ...	2022-02-...	2022-02-...
MVC Development	2022-02-...	2022-02-...
Demo Formal Testing	2022-02-...	2022-02-...
Demo Play-Testing	2022-02-...	2022-02-...
Playable Demo Compl...	2022-02-...	2022-02-...
Test Plan Revision 0	2022-03-...	2022-03-...
Design and Document...	2022-03-...	2022-03-...
Revision 0 Demo	2022-03-...	2022-03-...
Final Demo	2022-04-...	2022-04-...
Final Doc	2022-04-...	2022-04-...
Data Model Unit Testing	2022-03-...	2022-03-...
Controller Flow Cover...	2022-03-...	2022-03-...
Server Integration	2022-03-...	2022-03-...
View Analysis	2022-03-...	2022-03-...


evision 0
Design and Document Revision 0
14
15
16
17
18
21
22
23
24
25

9 Revision History

- Feb 18th - View analysis and server viability test
- Feb 20th - Requirements review and comparison, model unit testing (partial)
- Feb 23rd - Demo controller tests