Arimaa Final Report

Tayler How, Luke Miller, and Jesse Shellabarger

# State of Features

We implemented every feature that we enumerated in our problem statement. The basic features of the game are in place. You are able to move pieces, push and pull them, and pieces can be removed from the board on death.

Game states can be loaded and saved at any point of the game. This is accomplished by writing the current state of the game to a text file that the user specifies. The user is then able to load any game that has been previously saved by specifying the file it was saved to.

The game checks for all win conditions at the end of every move. These include the standard win condition, having a rabbit in the opponent’s home row, the loss of all the opponent’s rabbits, and the inability of the opponent to move.

Multiple players are able to play the game using “hot-seat” multiplayer on the same machine. It switches control of the pieces every four moves.

The game uses the user specified turn timer setting and then counts down in real time. When the turn timer expires, the user whose turn it is loses.

The board state is saved at the end of every user’s move, and the user is able to restart his turn to the initial state of the board when he started. This is useful in the case of a piece accidentally killed.

The GUI of the game shows both player’s names, allows the user to start games or load games. It displays whose turn it is and how much time is left in their turn. It also shows how many turns are left in the current move.

# Testing Strategies

~~The main testing technique in this project was scripted automated unit testing. This is because the test driven development structure of the project emphasized this technique. We used various methods as discussed in class to make this unit testing work such as dependency injection in our save and load testing. An example of this is one of the unit testing used for pushing pieces. In this testing, we created a board that has pieces in place to be pushed. We then push the pieces, and insure that the push function returned true, and that the pieces were in the correct place. We repeated this for every direction we could push a piece.~~

Since the focus of this project was the test-driven development process, the primary testing style we used was scripted automated unit testing. All of the game logic functions were easily unit tested, however the GUI could not be unit tested effectively. The unit tests for the save and load functions required a dependency injection to account for reading or writing to a text file. An example of our basic unit tests is testBasicPullLeft. For this test, we create an instance of the game with custom board that has pieces placed to execute a pull. Next, we pull the appropriate pieces and verify that the pull was executed successfully. We assert that the pull command returned true and the pieces are in the correct spots. We repeated this process for all possible directions, as well as for other caveats (i.e. can’t pull another of your own pieces, can only pull pieces strictly weaker than you).

~~We used big bang integration testing, as the complexity of the integrations was relatively low, and we only had a few parts to integrate. This led to any issues with the integration to be obvious, and we were able to fix any problems that appeared.~~

Another testing style we used was big-bang integration testing. There were only had three parts to integrate: the game model, the GUI, and .txt files for saving and loading games. We chose the big-bang approach because the complexity of our integrations was relatively low. It was very easy to debug any problems that occurred during integration. To perform integration testing, we simply ran our game with the GUI and hoped everything worked.

~~We did black-box exploratory acceptance testing for the completely integrated program. We used this technique because our design changed multiple times throughout the project, and so any scripted acceptance testing would be brittle and would have to be changed multiple times as the requirements changed~~

~~We also did independent verification and validation of our project by sending it to multiple third parties to play the game and report any bugs or issues with the gameplay experience.~~

Since the ultimate goal of the project focuses on the user’s experience playing the game, we did a lot of exploratory manual black-box acceptance testing. Testing the game from a user’s perspective helped us find and fix any bugs that we may have overlooked during unit testing and early development. To perform acceptance testing, we played through whole games countless times. We made sure to test all of our game model and GUI functionality, focusing on edge and corner cases. Towards the end of the project, we also tested our game through independent verification and validation. We had multiple third parties play the game and report any bugs or issues with their gameplay experience.

# Testing Thoroughness