CMPT 370 Final Report

Group H2

Amanda Zimolag
Brendan Nykoluk
George Shi
Spencer O’Lain
Xinnan Xie
Zeshan Ahmad

1. User Documentation

1.1 User Guide

Our program will be supplied to the user in the form of the .jar file at https://git.cs.usask.ca/370-19/h2/blob/master/HanabiFinal/HanabiProjectV4%203/out/artifacts/Hanabi_jar/Hanabi.jar. Since it was compiled in java, a modern version of the Java Runtime Environment (11 or greater) is required to run the program. There is no installation or configuration files required to run the program.

From the start menu that shows up when the jar file is executed, the user can create or join a game by selecting the “create server” or “choose server” buttons. The create menu allows the user to select the number of players and turn length from drop down menus. The join menu requires the user to enter the game ID and the secret token to join a specific game.

Once the game begins, the user can make all of their moves using the text box and buttons at the bottom of the screen. For play and discard, the user should enter the position of the card they wish to play. The positions start at “1” for the card on the left and increase going right. The hint action works similarly, except the number entered now corresponds to the player that the user wishes to give a hint to. The numbers start at “1” for the player on the left of the user’s cards and increase in a clockwise order.

1.2 As-Built Requirements

Our Hanabi client differs from the client outlined in our requirements document in a few ways. The first difference is that some features were cut from the start menu. The tutorial and settings menu were given a lower priority during development due to time constraints. The other major change between our requirements document and the final client is the lack of a “take a break” button in the in-game interface. This was also deemed low priority due to time constraints, and ultimately never completed.

The changes introduced mid-way through the coding process are also not implemented at this time. This implementation of the rainbow cards was given a low priority because our implementation of the base game took longer than expected, and there was no time remaining when we finally were ready to implement the rainbow cards.

2. Programmer Documentation

2.1 Compilation Instructions

Our Hanabi client can be built easily using the builder in IntelliJ. We never attempted to build it manually since IntelliJ took care of the building for us. The main class for the program is Main.java. Running the main method of Main.java will start the program in a similar fashion to running the jar file specified in section 1.1 of this document.

Our client requires two libraries to compile. The first is GSON, which was used for operations involving json messages. It can be found at https://github.com/google/gson. The other library required is JavaFX. It can be found at https://gluonhq.com/products/javafx/. JavaFX requires some set-up to be used with IntelliJ, instructions for which can be found at https://openjfx.io/openjfx-docs/#introduction.

2.2 As-Built Design

Our design changed a fair bit from the initial design. Most classes changed from what was represented in our document at least a little bit. The start menu, join menu, and create menu were combined into one class because JavaFX allowed all of the user interfaces to be combined in a way that made them generally more simple. The fields that were stored in the join and create menus are no longer fields since they can be read from the interface as necessary.

Several classes were removed from the final product. The tutorial class was never implemented, and as such doesn't exist. The artifacts interface was never created, and the functions just implemented in other classes as necessary. The messages class was removed because it ended up serving no real purpose.

Our JsonOps class was modified because of a lack of understanding by the group going into the project. The changes were due to the fact that it was more simple to create separate methods for each kind of message being sent out then to have one monster class to build them all. The rest of the functions ended up serving a similar purpose.

A few more classes were added to the play field for the user interface. This was required because java swing requires separate classes for the frame and panel. These were not in the design document because of our overall inexperience with user interfaces. Some extra classes were required for the fireworks display for a similar reason. A new class was also added for cards. This allowed us to store the rank and suit of the cards using enumerations to cut down on the amount of errors that could be produced.

2.3 JavaDocs

The JavaDocs for our Hanabi client can be found at https://git.cs.usask.ca/370-19/h2/tree/master/HanabiFinal/HanabiComented/HanabiJavaDoc.

2.4 Known Bugs, Incomplete Features, and Workarounds

2.4.1 Known Bugs

The most game breaking bug in our project currently has to do with the buttons in the play field. When it is the user's turn, the buttons can be pressed and the corresponding action will be taken successfully. When it is not the user's turn, pressing any of the play, discard, or hint buttons will cause the program to crash.

Our client also has a couple of visual bugs with the play field. The first of these bugs has to do with the discard pile. As of right now, it only updates with the cards the user burns or discards. The cards the other players discard or burn do not show up on the user's discard pile.

The other visual bug occurs when a player plays a card that gets burned. On occasion, a fuse token is not removed from the user interface, despite the play field class updating its field. This means the game can end while the screen still shows fuse tokens remaining.

The final visual bug at the time of release has to do with the end game screen. The end game screen is only appearing on one type of game ending message from the server. If the game is cancelled, the screen does not appear, and the user is simply left waiting.

2.4.2 Incomplete Features

The largest incomplete feature of our client is the Automated Intelligence. At the time of release, the AI player does not functionally exist in any capacity. Some of the building blocks exist, but due to time constraints the full implementation was never completed.

The second incomplete feature is the lack of a message system to the user. As of right now, the user has no way of knowing that they have successfully connected or that a game was cancelled without looking at the command line. In the future, this should be presented in the user interface.

The other incomplete feature of our client is the Tutorial. In the current implementation the tutorial exists as a blank white screen. Unfortunately the time required to create the tutorial was instead spent refining other parts of the project instead.

2.4.3 Workarounds

We have one major workaround in the client right now. When the user attempts to create or join a game, the play field opens up automatically. This is an issue because it does this even when incorrect information was sent to the server and an invalid reply is received. The user will be sent to a blank play field and given no notice that their message was invalid.

2.5 Highlights

Our client has one feature that we feel is a highlight of our program: the end game fireworks display. It is capable of taking the results of the game, and producing a display that corresponds directly with the final score of each colour. Every successfully played card changes the display by upgrading the firework to be displayed.