Hanabi Client: Test Plan

CMPT 370 - Group F1

Minh Hoang - mph952
Sharjeel Humayun - shh738
Donovan Lavoie - dbl599
Georgi Nikolov - gnn308
Faiz Qureshi - faq588
Jason Wong - jaw845

Contents

1. Introduction

This document is the test plan for the Hanabi Client. It describes a set of tests that check whether the Client implements the specifications in the Client's Design Document (the DD) and satisfies the functionality in the Client's Requirements Document (the RD). These tests are structured into 3 levels. End-to-end tests check that the Client serves all the use cases in the RD, while integration testing checks that the individual steps in the use cases, particularly those involving communication between architectural components, work correctly and unit testing checks whether the methods of different components function properly.

2. End-To-End Tests

2.1 Successful Game Creation

This test checks for full implementation of the Main Sequence in Requirement 4.2 of the RD. The display initially shows the Main Menu view; after a simulated click of the Create Game button, it should shift to the Create Game view. The fields of the Create Game view used for data input are then set to the settings of a 5 Player game with a 60 second timeout, no Force option, and a Player NSID of abc123. Another simulated click of the Create button should then send a game creation message with those settings to a mock Server, which should respond with a message about successfully creating a game with a mock Game ID and Token of 1 and “sample”; this message should also cause the display to change to the Lobby view.

Once the game has been created, the mock Server should then send messages about 4 other Players joining the game, with the Player count increasing by 1 each time. Once all the fake Players join the game, the mock Server will send a message about the game starting with dealt hands of rank 1-5 cards of each colour to the Players; the Game Creator should be Player 1 and will be given an “empty” hand with no information. The game should then start with the Player’s hands set to the dealt hands, the information and fuse token counts set to 8 and 3, and empty discard and fireworks piles plus an empty log. The “Discard Visible” and “Log Visible” flags should be off, and it should be Player 1’s turn. Finally, the display should then change to the Game view.

2.2 Forceful Game Creation

Another test of game creation is whether a Game Creator can leave one game and forcefully create another game with their NSID, which combines the 6th Error Sequence and Steps 1-6 of the Main Sequence of Requirement 4.2 in the RD. The first half of this test is the same as Test 2.1 for successful game creation, but with the addition of the mock Server keeping track of the created game instance under the Game Creator’s NSID. One simulated Player join should occur, and then a simulated click of the Leave Game button on the Lobby view should cause the Game Creator to leave the game and make the display change to the Main Menu view.

A simulated click of the Create Game button should then bring the display to the Create Game view. Two attempts are then made to create another game where the data inputs are set to another 5 Player game with a 60 second timeout and an NSID of abc123. The first attempt should not set the Force option before simulating a Create button click, which will send a game creation message to the mock Server and receive an error message about an extant game under the NSID of abc123. The failed attempt should keep the display at the Create Game view. The second attempt will be similar but now the Force option is set so that the simulated Create button click will send another game creation message to the mock Server, which now destroys the pre-existing game and creates a new game under the NSID of abc123. The Client will receive a message about the new game being created with a mock Game ID and Token of 2 and “resample” and the display will change to the Lobby view, ending the test.

2.3 Human Player Joining a Game

As a test of the full Main Sequence and 3rd Error Sequence of Requirement 4.3 in the RD, the display should start at the Main Menu view and a mock Server should contain an instance of a game with a Game ID and Token of 1 and “sample”, a Player count of 3, and 2 slots left. A simulated Join Game click should bring the display to the Join Game view, where the fields for data input are set to a Game ID and Token of 3 and “unsample” and an NSID of def456. Once a simulated Join button is clicked, a game join message is sent to the mock Server, which responds with a message about failing to join a non-existent game. This should keep the display at the Join Game view.

For a successful join, the NSID is again set to def456, but the Game ID and Token are now set to 1 and “sample”. After a simulated Join button click, another game join message is sent to the mock Server, which responds with a successful join message and adds the Player to the game. Another simulated Player join should then occur, accompanied by a received message, and cause the game to start as the mock Server sends a message with hands of rank 1-5 cards of Red, Green, and Blue cards. Taking the Client’s Player as Player 1, the game should then start with the Player’s hands set to the dealt hands, the information and fuse tokens set to 8 and 3, an empty log, and empty discard and fireworks piles. The “Discard Visible” and “Log Visible” flags should be off, and it should be Player 1’s turn. Once the display changes to the Game view, the test ends.

2.4 AI Player Joining a Game

This test covers the Main Sequence of Requirement 4.4 in the RD; it also covers Steps 3-5 of the Alternative Sequence, but Steps 1-2 aren’t tested since they are essentially a manual version of Step 2 in the Main Sequence.

The display should start at the Lobby view and a mock Server should have a 3-Player game with 2 slots left and a Game ID and Token of 1 and “sample”. A HanabiController should also know about this game and know that the Player has an NSID of abc123. A simulated click of the Add AI button should result in the HanabiController giving the Game ID “gid”, Token “tok”, and Player NSID “nsid” to another instance of the Client, which it creates by invoking the terminal command “hanabiClient -g gid -t tok -n nsid”. This AI Player should successfully join the game, which the mock Server will see as another Player sending a game join message, and the original Client should receive a message about another Player joining the game. During this process, the display stays at the Lobby view.

Once another mock Player join occurs and a message is received about that, another message with 3 hands of rank 1-5 Red, Blue, and Green cards should be received, and a game should start, taking the original Client’s Player as Player 1. The game should start with the Player’s hands as the dealt hands, the information and fuse tokens set to 8 and 3, an empty log, and empty discard and fireworks piles. The “Discard Visible” and “Log Visible” flags should be off, and it should be Player 1’s turn. Once the display changes to the Game view, the test ends.

2.5 Discard Card

The Discard Card test covers the Main Sequence and the 2nd and 4th Alternative Sequences in Requirement 4.5 of the RD. The initial state involves the display showing the Game view while the HanabiController has a connection to a mock Server. The game state contains 2 Players whose hands both consist of a Yellow 1 in the first index of their hands. The information token count is set at 6 while the discard pile and action log are empty; the fuse token count and fireworks piles don't matter for this test. The draw pile that the mock Server holds contains a single Green 1 card. The Client handles Player 1 while it is currently the turn of Player 2, whom is controlled by a mock Player.

The mock Player will tell the mock Server that they are discarding their first card and the Client should get a response from the Server saying that the current Player discarded their first card (a Yellow 1) and drew a Green 1 to replace it. Once the model updates based on the move, Player 2's hand should have a Green 1 in the first index, the discard pile should contain a Yellow 1, the information token count should now be 7, and an action for Player 2 discarding a Yellow 1 from their hand's first index should be in the log. A Server message should then be received about it being Player 1's turn.

Player 1's first card button should get a simulated click, causing the Game view to show the prompt that asks whether to play or discard that card. A simulated click of the Cancel button should occur first, which should clear the prompt and produce no change in the game state, before the button for Player 1's first card is clicked again. Another simulated click of the Discard button should then send a message to the mock Server about discarding the first card from Player 1's hand. A Server message should be received in response confirming the Discard action of their first card (a Yellow 1) and saying that there was no replacement card. Once the model responds to this, Player 1's hand should have no card in the first index, the discard pile should have 2 Yellow 1s, the information token count should now be 8, and an action for Player 1 discarding a Yellow 1 from their hand's first index should be in the log. If these are all true, the test ends.

2.6 Play Card

The Play Card test covers the Main Sequence and the 2nd, 3rd, 4th, and 6th Alternative Sequences in Requirement 4.6 of the RD. The initial state once again involves the display showing the Game view and a HanabiController with a connection to a mock Server. The game state contains 2 Players whose hands consist of a Red 2 and a Red 5 in the first index of their hands. The information and fuse token counts are set at 7 and 3 while the Red fireworks pile has rank 1-4 Red cards and the discard pile and action log are empty. The draw pile that the mock Server holds contains a single Green 1 card. The Client handles Player 1 while it is currently the turn of mock Player 2.

The test starts with the mock Player playing their first card from their hand. A message should be received from the Server saying that the current Player played their first card (a Red 5) and drew a Green 1 to replace it. The model should then be updated with this information; afterwards, Player 2's hand should have a Green 1 card in the first index, the Red firework pile should have rank 1-5 Red cards, the information token count should be at 8, and an action for Player 2 playing a Red 5 from their hand's first index should be in the log. It should then be Player 1’s turn, as indicated by a received Server message.

A simulated click of the button corresponding to Player 1’s first card should occur and trigger the “Play or Discard” prompt to appear. Simulating a click of the Cancel button should close the prompt with no change to the state of the game, after which the prompt is brought back with another click of Player 1’s first card. Once a simulated Play button click happens, a message should be sent to the mock Server about playing the first card in the Player’s hand (a Red 1) and another message should be received about that card being burned and discarded and not drawing a replacement card. The model will update based on this, after which Player 1’s hand should have no card in the first index, the discard pile should contain the Red 1, the fuse count should be 2, and an action for Player 1 playing a Red 1 from their hand’s first index should be in the log. Once these are true, the test is over.

2.7 Give Info

This test of the Give Info action covers the Main Sequence and 2nd Alternative Sequence in Requirement 4.7 of the RD. The display starts at the Game view and a HanabiController exists with a connection to a mock Server. The model game state contains 2 Players; the first Player’s hand has 5 unknown cards that Player 2 can see as 3 Red 1s and 2 Red 2s while Player 2’s hand has 3 Blue 1s and 2 Blue 2s. The information token count is set at 8 while the action log is empty; the fuse tokens and the fireworks and discard piles don’t matter for this test, nor does the draw pile that the mock Server holds. The Client handles Player 1 while it is currently the turn of mock Player 2.

The mock Player will tell the mock Server that they are telling Player 1 about the rank 1 cards in their hand and the Client should get a response from the Server saying that the current Player told Player 1 that their first 3 cards are rank 1 cards. Once the model gets updated, Player 1’s first three cards should be replaced by cards with a rank of 1 but an unknown colour, those cards should have their “Known Rank” flag set, the information token count should now be 7, and an action for Player 2 telling Player 1 about their rank 1 cards should be in the log. A Server message should then be received about it being Player 1’s turn.

The button for Player 2’s first card should get a simulated click, bringing up a Game view prompt asking whether to tell them about the colour or rank of that card. The Cancel button should be clicked first to clear the prompt and check that the game state doesn’t change, and then 2 sequential simulated clicks should happen: a click of Player 2’s first card button and the Colour button in the prompt. A Server message should then be sent indicating that Player 2 is being told about their Blue cards and, unlike the Play and Discard actions, no other message should be received from the mock Server. After the model responds to the action, all 5 of Player 2’s cards should have the “Colour Known” flag set, the information token count should now be 6, and an action for Player 1 telling Player 2 about their Blue cards should be in the log. If these are true, the test finishes.

2.8 Leave Game via Button

As a test of Requirement 2.8 (which only has a Main Sequence) in the RD, the display should start at the Game view while a HanabiController has a connection to a mock Server. There should be an ongoing 2-Player game where the Player's hands have rank 1-5 Red and Blue cards, the fireworks piles all have a single rank 1 card in them, and the discard pile has a single Green 2 card. The action log should have the back and forth plays of Players 1 and 2 plus the discard action from Player 2 necessary to establish the given fireworks and discard piles. The information and fuse tokens are at 8 and 3 and Player 1 has the current turn.

The test is set off by a simulated click of the Leave Game button on the Game view, which should then lead to the connection to the mock Server being destroyed and the model's game state being cleared due to the premature end of the game. Once this is all done, there should be 0 Players in the game and no Player objects in the model, the fireworks and discard piles should all have 0 cards, the action log should have 0 actions, the information and fuse token counts should be 0, and the "Discard Visible" and "Log Visible" flags should be turned off. The Game ID and Token of the game should also be cleared in the model and controller, as should the NSID field in the controller. Once these are true and the display is at the Main Menu view, the test ends.

2.9 Toggle Discard Pile View

To test Requirement 2.9's Main Sequence in the RD, the display should once again start at the Game view while a HanabiController has a connection to a mock Server. An ongoing 2-Player game should have rank 1-5 hands of Red and Blue cards for Players 1 and 2, the discard pile should have a set of rank 1-4 Green cards in it, and the White fireworks pile should have a single White 1 in it. The log, meanwhile, should have a "Give Info" action where Player 1 tells Player 2 about the Blue cards in their hand and a "Play" action where Player 2 plays a White 1 that goes onto the White fireworks pile, followed by a sequence of back-and-forth "Discard" actions between Players 1 and 2 that place the rank 1-4 Green cards into the discard pile. The other fireworks piles and the token counts do not matter for this test, but the "Discard Visible" and "Log Visible" flags should be off.

The test itself starts with a simulated click of the discard pile's button, which should turn the "Discard Visible" flag on. The Game view should then change to show a small inset window with the discarded rank 1-4 Green cards. Another simulated click of the discard pile should turn the "Discard Visible" flag back off and make the inset discard window disappear. If these happen, then the test is successful.

2.10 Toggle Action Log View

A test of Requirement 2.10's Main Sequence in the RD uses the same starting state as Test 2.9 above. The test itself changes and uses a simulated click of the Log button on the Game view, which should turn the "Log Visible" flag on and cause the Game view to show a different inset window with a list of statements, one for each of the actions in the log. Another simulated click of the Log button should turn the “Log Visible” flag back off and make the inset log window disappear. If these hold true, then the test is successful.

2.11 End Game Sequence

Testing the End Game Sequence covers the 5th, 7th, and 3rd Alternative Sequences of Requirements 4.5 - 4.7 in the RD. The initial state is a 2-Player game that has finished due to running out of cards; both Players have an empty card slot in the first index of their hands and arbitrary cards in the other 4 slots. The fireworks piles each have rank 1-4 cards of their colour and the discard pile and log have arbitrary cards and actions in them. The information and fuse token counts don’t matter, even though running out of fuse tokens will end the game; the other end conditions of finishing all the fireworks and running out of fuses will ultimately be recognized by the Server, which starts the End Game Sequence.

The End Game Sequence test starts when a message is received from the mock Server about the game ending. The controller should then destroy the Server connection, clearing the Game ID, Token, and NSID from the controller, and the display should transition to the Game Over view, which displays a message with the game’s final score (the total number of cards in the fireworks piles). A simulated click of the OK button should then cause the game state to clear itself; the number of Players should be 0, there should be no Player objects, the fireworks and discard piles and the log should be empty, the information and fuse token counts should be 0, the “Discard Visible” and “Log Visible” flags should be off, and the Game ID and Token should be cleared. If all of these are true and the display has transitioned to the Main Menu view, then the test has passed.

3. Integration Tests

3.1 Creating a Game

This integration test covers the subset of end-to-end test 2.1 where Server communication occurs to create a game and have the Game Creator join their new game. This corresponds to Steps 5-6 of Requirement 4.2 in the RD. The display initially shows the Create Game view while the HanabiController doesn't yet have a connection to a mock server and both it and the model have no recorded Game ID or Token. The Creator's game settings are presumed to be a game with 4 Players, a 60 second timeout, an NSID of abc123, and no Force option. The mock Server should not have any other games under the NSID of abc123.

A simulated click of the Create button should then cause the mock Server to receive a message about a game being created with the parameters specified above. The controller in the Client should then in turn receive a message confirming that a new game has been created with a Game ID of 1 and a Token of "game1". In reaction to that message, the controller and model should then have the Game ID and Token stored, the controller should know about the Game Creator's NSID, and only 1 Player should be in the game. If those are true and the display shows the Lobby view, the test ends.

3.2 Starting a Game

To cover the game starting portion of end-to-end test 2.1, which matches Steps 8-9 in Requirements 4.2 and 4.3 and Steps 3-4 in Requirement 4.4 of the RD, the display now starts at the Lobby view. The model, controller, and mock Server should know about a 4 Player game with 3 Players in it and an arbitrary Game ID and Token. The test starts with a 4th and final Player joining the game where the controller should receive a message with the starting hands: a set of 5 empty cards for the Client Player as Player 1, and then hands of [1R, 1R, 3G, 3B], [1R, 1Y, 1G, 1B], and [4R, 5Y, 2G, 4G] for Players 2-4. The model should then start the game: there should be 4 Players whose hands are equal to the ones given earlier, the discard and fireworks piles are empty, as is the log, the information and fuse token counts should be 8 and 3, and it should be Player 1's turn. The "Discard Visible" and "Log Visible" flags should also be off. Once these are true and the display shows the Game view, the test ends.

3.3 Human Joining a Game

Another integration test covers the part of end-to-end test 2.3 from the Join button being clicked to having the Player join the game; this matches Steps 5-7 in Requirement 4.3 of the RD. The display should be showing the Lobby view while the mock Server is aware of a 4 Player game with 1 Player in it and a Game ID and Token of 1 and "game1". The joining Player is assumed to be giving the same Game ID and Token using an NSID of abc124. A simulated click of the Join button should then send a join game message to the mock Server, which should have the Game ID, Token, and NSID given above. The controller should then receive another message saying that the Player successfully joined that game. The controller and model should then know about the same 4 Player game with the same Game ID and

Token, and that there are currently 2 Players in the game. The display should also still be at the Lobby view.

3.4 AI Joining a Game

A test for AI Players joining a game during end-to-end tests 2.1 and 2.4, unlike with Human Players, covers Steps 1-3 in Requirement 4.4 of the RD. The display starts at the Lobby view while a HanabiController has a mock Server connection and both them and the model know about 1 Player in a 5 Player game. The game's Game ID and Token are 123 and "game" and the Client Player's NSID is abc123. A simulated click of the Add Computer button should create another instance of the Client which is given the game's Game ID and Token and the Client Player's NSID above, and this new Client should send a join game message to the mock Server using those parameters to join the same game. The controller should then receive a message saying that a player successfully joined, and in response to this the model should know that there are 2 Players in the game. The display should still be at the Lobby view, which shows the new Player count.

3.5 AI Getting a Move

Another test for AI Players covers the coordination between the HanabiGame, HanabiController, and AIController classes (see sections 3.1.9, 3.3.1, and 3.3.2 in the DD) when an AI Player makes a move. Each kind of play is similar in terms of this integration, so this test only gives the AI Player a game state where it should play a card. In a 2 Player game, the fireworks piles are empty except for a Red 1 in the red pile, while the discard pile has a single Green 3 and the AI has a 5-card hand where it knows the 2nd card is Red. The other Player's hand has rank 1-5 White cards and there are 6 information tokens.

The test starts with a message from a mock Server saying that it is the AI Player's turn. The AIController should then be given a reference to the game and its state should then be obtained from the model. This obtained state should match the state given above: an AI hand of [R?, ??, ??, ??, ??], the other Player's hand of [W1, W2, W3, W4, W5], a Red fireworks pile with a Red 1 plus 5 empty fireworks piles, an empty discard pile, and 6 information tokens. The AIController should then use that state to generate a best move of playing the 2nd index card from the AI Player's hand and give that to the controller. If it gets back this move, the test is successful.

3.6 Move Communication

This test covers the process of communicating a move to the Server and other Players as part of end-to-end tests 2.5-2.7 and Step 5 of Requirements 4.5-4.7 in the RD. The display starts at the Game view in the middle of a 2-Player game with empty fireworks and discard piles, an empty log, information and fuse token counts of 7 and 3, and five arbitrary cards in each player's hand except for Player 1's 3rd card, which is a Blue 1 which they know is Blue. The controller has a connection to a mock Server that will get and send Server messages. It is Player 1's turn.

Using the example of a discard action, a pair of simulated clicks on Player 1's third card button and the Discard button in the "Play or Discard" prompt should tell the controller that Player 1 is discarding their 3rd card. The controller should then send a message to the mock Server, which should get a message about the current Player discarding their 3rd card. Another message should be sent by the mock Server to the controller, which indicates that their discard of a Blue 1 card was accepted and that the replacement card was a Yellow 2. If the controller can eventually receive this message, then the test succeeds.

3.7 Move Application

The final integration test covers the process of applying a move to the model, which occurs during end-to-end tests 2.5-2.7 and matches with Steps 6-7 of Requirements 4.5-4.6 and Steps 7-8 of Requirement 4.7 of the RD. The initial state is the same as with integration test 3.6 above, except the controller has received the message about Player 1 successfully discarding a Blue 1 as their 3rd card and drawing a Yellow 2 as a replacement. The controller should then tell the model to update itself in response to this move. After the update, Player 1's third card should now be a Yellow 2, the discard pile should contain a single Blue 1, the information token count should be 8, and an action for Player 1 discarding a Blue 1 from the 3rd index of their hand should be in the Log. The display should still be on the Game view, but the new information token count and new state of Player 1's hand should be reflected in it. If these are true, the test succeeds.

4. Unit Tests

4.1 Model Unit Tests

4.1.1 HanabiGame

These tests cover the HanabiGame Class (in section 3.1.9 of the DD) of the set of methods it contains. The tests come in 4 sets: one for creating a starting a game, one for applying moves, one for toggling view flags, and one for ending the game. At the very start of these tests the Game ID and Token should be 0 and “”, there should be 0 Players and no Player objects, empty discard and fireworks piles, and empty log, and 0 information and fuse tokens. The current Player should be “Player” 0 and the “Discard Visible” and “Log Visible” flags should be off.

The first tests start with the creation of a new game and the entrance of the Game Creator as the first Player. This is done by calling enterGame() with a Game ID and Token of 1 and “sample” and calling addPlayer() once. The Game ID and Token fields should be set to the same Game ID and Token and the number of Players should be 1; there are no Player objects yet. addPlayer() should then be called 1 more time to add 1 more Player, incrementing HanabiGame’s Player count, before calling removePlayer() to bring it back down to 1, and then calling addPlayer() 2 more times to get a Player count of 3. These are followed by calling startGame() with an empty hand for the Client’s Player, [], and 2 sets of hands that are rank 1-5 cards of a single colour: [R1-R5] and [B1-B5]. This should result in the creation of 3 Player objects; the first one should have a hand of 5 cards with a rank of 0 and a colour of Null, all of which have the rank and colour information flags off, while the next 2 should have hands of cards that match the given ones and have the rank and colour information flags off. The information and fuse token counts should be set to 8 and 3 and the current Player tracker should be set to 1.

The second test set involves having sequential info, discard, and play actions occur. giveInfo() will get called to say that the current Player (P1) is telling Player 3 about rank 1 cards in their hand. This should cause the information token count to fall to 7, the rank information flag on Player 3’s first card (a Blue 1) to be set, the log to gain an info action about the move, and the current Player tracker to change to 2 without changing the rest of the state. Then discardCard() gets called to say that the current Player (P2) is discarding their 2nd card and drew a Yellow 4 as a replacement. Without changing anything else, Player 2’s hand should have a Yellow 4 instead of a Red 2 as their 2nd card (with no information flags set), the information token count should be 8, the discard pile should have a Red 2, the log should have the first action plus a new discard action about the move, and the current Player tracker should be 3. Finally, playCard() should get called to say that the current Player (P3) played their first card and got a Blue 4 as a replacement. This should, without other changes, change Player 3’s first card from a Blue 1 to a Blue 4 (without information flags set), add a Blue 1 to the Blue fireworks pile, add a play action to the log, and change the current Player tracker back to 1.

The next set of tests is simpler and simply calls toggleDiscardView() twice and toggleLogView() twice. These should make the Discard Visible and Log Visible flags be set and then unset again. These are followed by a final call to endGame(), which should clear the game state. Clearing the state should mean that every field of HanabiGame should be set back to what it originally was at the beginning of these unit tests. If they are, then all the HanabiGame unit tests have passed.

4.1.2 Token

This test checks that the methods of the Token class (in Section 3.1.6 in the DD), addToken(), removeToken(), and getToken(), work correctly. The test should create a Token object that starts with a token count of 0 and getToken() should return 0. After sequential calls to addToken() and removeToken, the token count should change to 1 and then 0 respectively, with getToken() returning those values as well. Another call to removeToken() should raise an exception about having no more tokens, after which the test is complete.

4.1.3 Hand

A similar set of tests is to check the methods of the Hand class (in Section 3.1.2 in the DD), addCard(), removeCard(), getCard(), getCards(), and getSize(). The test starts by creating a Hand with a size of 4, after which getSize() should return 4, getCards() should return a set of 4 nulls, and getting the first card with getCard() should return a null. Four different card objects should then be created and added to the four slots in the hand and after each addition getCard() should return the same card for that index; after all four are added, getCards() should return the set of all four new cards. A call to removeCard() should then remove the third card, and getCard() should return a null again afterwards. Since a 5-card hand is also possible, a final test should try to add a card to the fifth slot, which should return an exception. Once these all pass, the Hand class passes its unit tests.

4.2 View Unit Tests

Some common elements while testing the GUI are to ensure that all elements are completely visible to the user. This means checking all the panels with buttons, texts, dialog boxes, etc. for the size, position, and inputs of their elements. Any text used to convey information from any view should be clearly visible and readable with proper alignment, font, and colour. Any images used should be clear and visible. Since a single window frame is created and encapsulated by Display, all views will share the same window size of 1280x720p. The MainMenuView, CreateGameView, JoinGameView, and GameOverView all use a common fireworks image in their background that fills the entire screen. They will also have a centred 330x300p panel which will have a centre aligned title text field. All buttons used in these four views have sizes of 200x50p. For allowed inputs, CreateGameView and JoinGameView's NSID input should allow alphanumeric characters in the format “xxxyyy” for variable letters “x” and numbers “y”.

4.2.1 MainMenuView

The title of this view is “Main Menu”. The view contains three buttons to let the user either create or join a game or to close the client. The title and buttons should be vertically aligned and centred to the panel. The “Create Game” and “Join Game” buttons should successfully transfer the player to the

CreateGameView and JoinGameView respectively, while the “Exit Game” button should successfully terminate the client.

4.2.2 CreateGameView

The title of this view is “Game Options”. All its elements should be horizontally aligned to the left side of the panel and to each other in rows. Below the title should be a text field saying “Number of players:”, under which are four radio buttons (two per line) with the labels [“2”, “4”] and [“3”, “5”]. The radio buttons should be vertically aligned to each other in pairs (2-3) and (4-5) and should all initially be de-selected, but only one can be selected at a time to select the Player count prior to creating a game. Below the radio buttons should be an input box of size 100x50p with the label “Timeout Period (seconds):”, and a faded placeholder name of “Value”. The timeout period accepts only numbers within the range of 1-120. Under that should be a check box with the label “Force” that defaults to being unchecked and specifies the Force option when creating games. Further below that should be another input field with the label “Your NSID:”. The bottom of the panel should have 2 buttons: “Create”, which should successfully transfer the player to the LobbyView after a mock update is received from the model about a game being created, and “Back”, which should successfully transfer the Player to the MainMenuView.

4.2.3 JoinGameView

The title of this view is “Join Game”. It should contain three input fields for taking the Game ID, Token, and NSID. All these elements should be vertically aligned and centred within the panel. The Game ID and Token fields should have a size of 280x50p and have faded placeholder values of “Enter Game ID” and “Enter Game Token”, whereas the NSID field should be 100x50p in size and have a label of “Your NSID”. The bottom of the panel should have 2 buttons: “Join”, which should successfully transfer the Player to the LobbyView after a mock update is received from the model about a game being joined, and “Back”, which should successfully transfer the Player to the MainMenuView.

4.2.4 LobbyView

The LobbyView should contain fixed text fields that show the game’s Game ID and Token and its current number of Players. It should have two buttons at the bottom right and left corners: “Leave Game”, which should successfully transfer the player to the MainMenuView after a mock model update is received about leaving the game, and “Add Computer”, with a size of 200x100p, which should increment the number of Players after another mock model update is received about another Player joining the game.

4.2.5 GameView

The GameView has four possible layouts based on the number of Players in the game. Cards in the Player’s hands should be represented by 85x140p buttons, with the Client’s Player’s own cards showing only the known properties of the card: their numeric rank and their colour. There should be text labels

“Information Tokens: x” and “Fuse Tokens: y” that display the game’s token counts. A button with a picture of a pile of cards labeled “Discard Pile” should also be visible and clicking it should expand and shrink a sub-window showing the cards in the discard pile when a mock model update toggles its state. The top and bottom right corners of each layout should have two standard buttons: “Log”, which should expand and shrink the Action Log sub-window which displays moves when a mock model update toggles its state, and “Leave”, which should return the player to the MainMenuView after another mock model update about leaving the game.

4.2.6 GameOverView

The title of this last view is “Game Over”. Below that should be a vertically aligned text field labeled “Your Score: x Points!” that displays the game’s final score “x”. At the bottom of the panel should be 2 horizontally aligned buttons: “Main Menu”, which should successfully transfer the Player to the MainMenuView, and “Quit”, which should successfully terminate the client.

4.3 Controller Unit Tests

4.3.1 AIController

These unit tests check the functionality of the AIController (in section 3.2.2 of the DD). In particular, getMove() is not tested in the unit tests since it mostly does integration level work in calling the other methods, bestPlay(), bestDiscard(), bestInfo(), and bestPlay(), and passing bestPlay()’s results to the HanabiController. Each of them is passed a set of game states that corresponds to the different cases for deciding on a given move type, which are described in section 2.6 of the DD.

To test bestPlay(), a set of four game states should be given to check its 4 possible cases, which depend on the AI Player’s hand and the fireworks piles. Each state has a set of fireworks piles where only the Blue pile has a rank 1 card, while the others are empty. The AI hand in the first state should be fully known rank 1-5 Blue cards (in that order in the hand), so the returned best play should be to play the 2nd card with a PlayValue of 0. The second state’s AI hand should only have knowledge that the 3rd card has a rank of 2, so its returned best play should be the 3rd card with a PlayValue of 1. The third state is similar except instead of knowing that the 3rd card is a 2, they only know that it is Blue, so the best play should still be the 3rd card but the PlayValue should change to 3. The last case should have a totally unknown AI hand, so that any hand index will work for a “best” play, but the returned PlayValue should be 3.

bestDiscard() should use a different set of 4 game states to test its cases, which use the AI’s hand and the fireworks and discard piles. The first test should use a fully known AI hand of 3 rank 1 Blue and 2 rank 1 Green cards while the fireworks piles have a single Blue 1 and the discard pile is empty; the best discard should be the 1st card with a PlayValue of 0. The next state should have an fully known AI hand of 1 Red 1, 2 Blue 1s, and 2 Green 1s, empty fireworks piles, and a single Blue 1 is in the discard pile, so that the best discard should be the 2nd card with a PlayValue of 1. A third state should have a single Blue card of unknown rank as the AI’s third card while the fireworks piles are empty and 2 Blue 1s and 2

Blue 2s are in the discard pile; the best discard should be the third card with a PlayValue of 2. Finally, a state should be given with a totally unknown AI hand and arbitrary fireworks and discard piles, since the best discard can be any card but should have a PlayValue of 3.

The tests for bestInfo() should be split into 3 state sets, all assuming that the AI Player is Player 1. The 3 sets should all have 2 other Player's hands where they know nothing: the first one has rank 1-5 Green cards with no info, while the other one has 3 Yellow 1s, a Blue 4, and a Red 2. The first set's fireworks piles should only have up to a Blue 3 so that this state should result in a best info move, with a PlayValue of 1, of telling the 3rd Player about their Blue cards. The second set's fireworks piles should be empty instead; the best info move should then be to tell Player 2 about rank 5 cards in their hand, with a PlayValue of 2. The third set should then change the 2nd Player's knowledge so that they know about their 5th card being a 5 while still having empty fireworks piles, which should make the best info move, with a PlayValue of 3, of telling Player 2 about their Green cards.

Finally, bestMove() should be tested with three sets of move selections; the move parameters themselves are arbitrary since the PlayValues and information tokens are what matter in move selection. The first set should be play, discard, and info PlayValues of (0, 0, 2) with 8 information tokens, which should return a "Play" move with the number matching the given play move and the info string being blank. The next set should be PlayValues of (2, 1, 1) with 7 information tokens, which should return a "Discard" move with the number matching the given discard move and the info string still being blank. A third set should have PlayValues of (1, 1, 1) with 7 information tokens so that the returned move should be an "Info" move with the number and info string matching those given by the info move.

4.3.2 ServerComm

These tests cover the unit that handle communication with the Server, ServerComm (in section 3.3.3 of the DD). Only two of the sendX() methods, sendCreate() and sendJoin(), are tested along with receiveMessage() since the mechanisms for sending messages are similar between all of them. A mock Server should be used in these tests that knows about 1 Player in a 4-Player game with a Game ID and Token of 2341 and "somethingSecret". There should be no initial connection to the mock Server to begin with.

The test starts by checking connect() and sendCreate() while also checking receiveMessage()'s functionality. A call to connect() should manage to create a connection to the mock Server, which manifests as an existing TCP Socket in ServerComm. A new game should then be created with 5 Players, a 60 second timeout, an NSID of abc123, and no Force by calling sendCreate(), after which the mock Server should receive a JSON message for creating a game with those parameters and some timestamp and MD5 hash. The mock Server should then send back 2 JSON messages about a game being created with a Game ID and Token of 1 and "sample" and the Game Creator joining that game. Both of these messages should be picked up and parsed by calls to receiveMessage(), which should return String sets with the JSON tags and their values. The first sets should be ["reply", "game-id", "token"] and ["created", "1", "sample"], while the second sets should be ["reply", "needed", "timeout"] and ["joined", "4", "60"].

As part of leaving a game, a disconnect() call should then happen, after which ServerComm should no longer have a TCP Socket. sendJoin() is then used to join the mock Server's existing game (Game ID and Token of 2341 and "somethingSecret") using the same NSID of abc123 and the mock Server should receive a JSON message for joining a game with those parameters and some timestamp and MD5 hash. The mock Server should respond by sending another JSON message about successfully joining that game; a call to receiveMessage() should manage to parse that message and return String sets of ["reply", "needed", "timeout"] and ["joined", "2", "60"].

Finally, the mock Server should simulate having another Player join the existing game and send a JSON message to the Client saying that another Player joined and that 1 more is needed. This unprompted message should still manage to be parsed by receiveMessage(), which should return String sets of ["notice", "needed"] and ["player joined", "1"].

5. Summary

We have described a test plan for checking the functionality and completeness of the Hanabi Client. End-to-end tests have been given for ensuring that the Client will let Players create and join games, add AI Players, and make moves like playing, discarding, and giving info about cards as described in the use cases of our Requirements Document. A set of integration tests were also given for checking the functionality of the major steps of the use cases and unit tests to check the classes and methods involved in the integration tests were specified. Together, these should provide assurance that the Hanabi Client works correctly if it passes all these tests.