**Introduction**

This document describes the test plan for the Hanabi game client. It describes each level of testing: a high level end-to-end test case where it tests whether the client itself is functional, integration testing based on the model-view-controller and unit testing that describes each methods functionalities.

**End-To-End Testing**

* Join game
  + Test to see if a player can join a game with Join game button.
  + Precondition: Player is at join game screen
  + Input: simulate button click -> simulate inputs to join a game -> Simulated button click  -> Simulated player joins
  + Mock components: mock communication with the server?
  + Path:
    - Wrong inputs -> sent back to main menu
    - Right inputs -> simulate players in a game lobby -> successfully joined a game
* Join game (AI) Button
  + Button for a game creator to add an AI to the game
  + Command line to add an AI to the game
* Join game (AI) Command Line
  + Command line to add an AI to the game
* Discard a card
  + Test to see if a player discard a card during the game
  + Precondition: game is running, player is given the option to discard a card
  + Input: simulate button click on the card to discard?
  + Mock Components: mock server simulates a message that a card has been discarded from the players hand
  + Path:
    - wrong inputs -> player still has original hand (nothing in the hand has changed)
    - Right inputs -> simulate a player discarding a card -> successfully discarding a card
* Play a card
  + Test to see if a player can play a card during a game
  + Precondition: game is running, player is given the option to play a card
  + Input: simulate card click selection on the card to be played
  + Mock components: mock server simulates a message that a card has been played from the players hand
  + Path:
    - Wrong inputs -> player still has original hand (nothing in the hand has changed)
    - Right inputs -> simulate a player playing a card -> successfully playing a card
* Give info
  + Test to see if a player can give information during a game
  + Precondition: game is running, player is given the option to give information
  + Input: simulate give information selection on the card the player wants to give
  + Mock components: mock server simulates a message that information has been giving to a player
  + Path:
    - Wrong inputs -> player does not know any information yet?
    - Right inputs -> simulate a player giving information -> successfully giving information
* Leave game
  + Testing to see if the player can leave a game or lobby properly with leave game button.
  + Precondition: The player has the leave game button visible?
  + Input: Simulated button click -> Simulated inputs to leave a game/lobby
  + Mock Components: Mock server that simulates messages for leaving a game/lobby
  + Path:
    - Wrong inputs -> still in lobby/game running
    - Force leave a game/lobby
    - Right inputs -> simulated player leaving a game/lobby -> successfully left a game/lobby
* View action log
  + Testing to see if the players can see the action log of what has happened during the game
  + Preconditions: a game is running, action log is not expanded out/option to click it to expand
  + Input: simulated action log button click -> simulate action log displaying properly
  + Mock components: mock server that simulates messages for an action log?
  + Path:
    - Wrong inputs: -> nothing is displayed on the action log
      * + -> action log button not functional

- Right inputs -> simulate player with an expanded action log -> successfully expanded an action log

**3. Integration Testing**

**3.4 Leave Game**

In the GameView the leaveButton is always visible for a player to leave a game early (see Section 3.2.5 of DD). During a game, a player clicked on the leaveButton and the game screen should go back to the main menu of the client. To test this the player should disconnect from the current game that is in play and the game should clear the models that are displayed on the player’s screen and update the display to MainMenuView (see Section 3.2.1 in DD).

**4. Unit Tests**

Prevent invalid action, check corner cases and working cases

## 4.1 Model Unit Tests

4.1.1 HanabiGame

To test the HanabiGame class there are going to be a handful of methods to be tested. For the  enterGame(int ID, String token) method should test if each player have an ID and token set to their respective selves. As for the addPlayer() and removePlayer() the game would see if a player has been added or removed from the game. In addition, to the add test cases the game must have room to add a player or if the game has the maximum amount of players no additional players can be added. The idea for the remove players would be the same assuming there are players to be removed from the game and if the game does not have empty players. For example, if there were 5 players in a game and we would like to remove a player from the current game it would become 4 players. This would be the same for adding a player to the game assuming it is not at its max capacity. For the toggleDiscardView() and toggleLogView() the game would determine if the views would actually toggle the discard and log views to display their respective information. As for the endGame() method, once the game has ended the player should return to the main menu and everything stored in the game will be cleared. Overall, the playCard(), discardCard(), and giveInfo() methods would all be similar in a way where the game has started and the player can decide which option to take. As an example, a player decides to give information to another player the unit test would see if the correct information was given to the other players.

4.1.2 Token Addition and Removal

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

4.1.3 Adding and Removing Cards from a Hand

This test checks the addCard() and removeCard() methods from the Hand class for correct functionality (see Section 3.1.2 in the DD). The test should add a card object to a player’s hand once the player has room for the additional card. Therefore, the hand count should vary from 4-5 cards in a player's hand based on the game size. Another test case for removeCard() would be if the player were to discard or play a card, the card would be removed from the hand and and another card would be added to the hand.

* Log
  + Testing can be done with HanabiGame
* Action
  + Testing can be done with HanabiGame
* Token
  + Testing if addToken() is functional, in the case there are max info tokens available and a functional range from 0-7 tokens available
    - Case 1
      * Precondition: 0-7 info tokens available
      * Input: void
      * Mock components: mock token object with tokens that range from 0-7
      * Path: adding a token, result in an additional token in game play
    - Case 2
      * Precondition: max info tokens (8 tokens)
      * Input: void
      * Mock components: mock token object with 8 tokens
      * Path: adding token is not allowed
  + Testing if removeToken() is functional, in the case there are no info tokens available and a functional range from 1-8 tokens available
    - Case 1:
      * Precondition: 0 info tokens available
      * Input: none
      * Mock components: mock token object with zero tokens
      * Path: remove a token is not allowed
    - Case 2
      * Precondition: 1-8 tokens available
      * Input: none
      * Mock components: mock token objects with 1 to 8 tokens
      * Path: remove a token, resulting in one less token available