Hanabi Testing Plan

Mels Kolozyan
Xudong Li
Xufeng Lin
Evan Semenoff
Tiandu Xie

Part I
Introduction

This document outlines the testing plan which will be used during the implementation phase. It is split into three parts, the first will cover end-to-end tests, which are directly related to use cases found in the requirements documentation. Next, the integration testing will be covered, this will reflect the architectural decisions made in the design document. Finally, unit testing will address the testing of the classes and methods which were outline in the design document.

Part II
End-To-End Testing

Each of the following tests reflects a specific use-case as seen in the requirements documentation (page 15). Each use-case will have two cases which will be covered: cases which cause the use-case to happen, and cases which won't. These will be denoted as positive (+) and negative (-) cases.

1 Create Table

+ Well Formed Input: The user has given a valid string to create a game with
+ Can Be Made: The game can be made.
+ Auto Join: Given that both of the previous conditions have been met, the user is joined into the game they created.
- Malformed Input: The user has given an invalid string.
- Connection Error: A connection to the server cannot be established.
- Game Can't Be Made: The server is unable to create the game.

2 Play Card

+ Was Valid: This causes two unique things to happen
  - The card is played
  - The score is incremented
+ Wasn't Valid: This causes two unique things to happen
  - The card is put into the discards
  - A fuse is removed

  Both of these actions cause a set of four actions that they share:
  - A new card is dealt
  - The players turn ends
  - The card pile is decremented
  - The timer is reset

3 Hint

+ Hint Given: This causes a hint to be given to another player, this causes four things to happen
  - The turn is changed
  - A hint is sent
  - The number of hint tokens goes down
  - The timer is reset
- No Tokens: This stops the player from giving a hint (by making the option unavailable)

4 Discard

+ Discard: This is the basic discard use-case, it causes many things to happen.

  - The tokens go up
  - A new card is drawn
  - A card is appended to the discards
  - The number of cards left is decremented
  - The turn ends
  - The timer is reset

- Tokens Full: This stops the player from discarding (by making the action into a null action)

5 View Discards

+ View Discards: The discards are drawn as an overlay on-top of the screen.

- Drawn Improperly: There are a few cases to consider
  - Not drawn in the correct order
  - Not drawing the correct cards
  - Not drawing the discards (drawing some other cards)

6 Personal Notes

+ Personal Notes: Drawn as an overlay on top of the screen

+ Cyclical hints: The hints cycle properly when clicking on them

- Drawn Improperly: The hint graphics don't reflect their state

- Broken Button: The button doesn't work

7 Start AI

+ AI Starts: The AI takes over the the game

- AI Won't Start: The AI won't start

- AI Won't Play: The AI won't make moves

8 End AI

+ AI Stops: The AI lets the human player play again

- AI Won't Stop: The AI won't stop playing

9 Exit

+ Exit: The connection is severed and the player returns to the main menu

- Can't Exit: The user can't exit the game without closing the window, or the exit button doesn't return them to the main menu.

10 Join Table

+ The player can successfully join a table given he meets these preconditions:
  - The game is not full
  - The input is well formed
  - The game exists

- Malformed Input: The string entered is not valid

- Connection Error: Cannot connect to the server

- Game Doesn't Exist: No game exists with those credentials

- Game is Full: There is no more room at the given table

Part III
Integration Testing

The integration tests were developed by breaking down the cases of the end-to-end tests, and outlining how they fit into the architecture. For brevity, the symbols C, M, V, and S will refer to the controller, the model, the view, and the server, respectively. This phase will outline how the code fits together and will allow us to verify that the modules fit together during the scaffolding phase.

1 Create Table

+ Well Formed Input: C → C
  The controller takes in a valid string, and creates a game
+ Can Be Made: S → C → M → V
  The controller receives verification from the server, a GameState is instantiated, and the view begins to draw the table.
+ Auto Join: C → S
  The controller joins the game which was created.
- Malformed Input: C → V
  The controller detects an invalid string, a window is drawn in the view to inform the user
- Connection Error: C → M → V
  The controller cannot connect to the server, this is reflected in the view via a pop-up
- Game Can't Be Made: C → M → V
  For some reason, the game cannot be made. The user is informed via pop-up

2 Play Card

+ Was Valid: C → S → C → M → V
  The controller sends a valid move to the server, which returns a JSON

packet. The packet is parsed, and the game state is updated, which is reflected in the view.

+ Wasn't Valid: C → S → C → V
  Exactly the same as was valid, but different things are updated.

+ Null Case: C → M → V → M → V
  The user tries to play a move that isn't valid (perhaps attempting to discard when the tokens are full). The controller determines the move is invalid and reverts the client state to before the move was attempted.

3 Hint

+ Valid Hint: C → M → V → C → M → V → C → S
  The user selects the first card, and then creates a valid hint packet after enough cards are selected.

+ Invalid Hint: C → M → V → C → M → V
  The user selects a card that conflicts with the other selection while giving a hint. The other card is deselected and nothing is sent to the server.

+ Hint Received: S → C → M → V
  A player at the table receives a hint. The model is updated, and the view plays a short animation to convey this information.

- No Tokens: M → V If no tokens are available, this is reflected in the view.

4 Discard

+ Discard: C → S → C → M → V
  Whenever a discard happens, the controller sends a message to the server indicating this, once the message is received, the model is updated (discards are appended) and the view is updated.

- Tokens Full: M → V
  If the token counter is full, this is reflected in the view.

5 View Discards

+ View Discards: C → M → V
  The controller detects that the user has opened the discards menu, the client state is updated, and the view reflects this change.

6 Personal Notes

+ Personal Notes: C → M → V
  The user clicks on the notes menu, and the notes about the cards are updated in the client state, this is reflected in the view.

7 Start AI

+ AI Starts: C → M → V
  The user requests that the AI plays for them, this creates a new instance of the AI. The button to start the AI is changed to a button that says "Stop AI".

8 End AI

+ AI Stops: C → M → V
  The user requests to get control back. This is just like the start AI but in reverse.

9 Exit

+ Exit: C → M → V
  The user leaves the game, the model deletes the game state, and the game is set back to the main menu.

10 Join Table

+ Join Table: C → S → C → V
  The controller requests to join a game, the server responds accordingly,

and the controller instantiates a game state (which is immediately reflected in the view).

Part IV
Unit Testing

1 GameState

appendDiscard Adds a new discarded card to the linked list of discards.

- Linked list length Incremented: The linked list of discards should increment by one due to append.
- The Last Element: The last element in the linked list should be the recently discarded card.

updateCardsOnTable Updates cards played on the table after a legal card is played.

- New Card in Array: The played card can be found in the array of cards on table.
- Correct Index: The played card is in the correct index or position.

updatePlayerHand Updates cards in a player's hand in the array of players.

- Card Played and Replaced: The card that is played gets replaced by a new card (expect the last round).
- New Card in Hand: A new card is drawn by the player who played a card.

2 ClientState

selectCard Set the card at the given index of the array to Selected

- Correct Player: The player should be in the given player index of the player array.

- Correct Card: The card should be in the given card index of the player's hand array.
- Set Selected Cards: The Selected attributes of the selected cards should be set to True.

ToggleMenu Toggle the boolean value at the index of MenusOpen dictionary.

- Test the Toggle: the boolean at the MenusOpen index of the dictionary shall be toggled (set to True if was False, set to False otherwise)

3 Player

updateHand Updates cards in the player's hand in the array of players.

- Card Played and Replaced: The card that is played gets replaced by a new card (except the last round).
- New Card in Hand: A new card is drawn by the player who played a card.

4 Controller

RecieveServerData responsible for getting the data off of the buffer

- + JSON exists: check that a JSON packet is in the buffered reader. This method is called only when the socket receives a reply from the server, so there has to be an item stored on the buffer.

ParseJSON should be taking in a json message and outputting an appropriate java construct

- + Event Map is correct: check that the provided event map is of the proper format (required by the server).

EvalEvent Most of the logic of the controller is handled by this function, so there are a lot of test cases here to assure non-faulty communication with the other modules of the program.

- +Player joined
  - Send dialog update to let the player know about the new player joining
- +Player left
  - Send dialog update to let the player know about a player leaving the game before a game starts
- +Game Starts: a message is received about the start of the game which instantiates the game and the main program itself
  - instanties the rest of the game state (fields like player cards and turn)
  - the main loop begins
  - rtthread for the turn timer
- +Your Turn: signals the first players about the start of their turn
  - the local turn variable (in game state) set to yourself
- +Invalid Action: received when a sent action was determined to be invalid
  - As the GUI does not allow for illegal actions to be sent in the first place, this occurrence throws an exception
- +Game Cancelled: received when a player leaves the game early
  - Message(dialog): lets the player know why the game was stopped, test for the dialog box popping up
  - return to the main menu (test if the main menu was redrawn)
- +Discarded(notice): the server sends this notice to let them know about a discard that another player did. It also send the information on the new card and a lot of things in the game state must be updated.
  - AppendDiscards: test if the card was added to the discard pile
  - UpdatePlayer: test if the new drawn card is now part of the player's hand
  - Inc Turn: make sure the turn is given to the next player
  - DecCards: verify that the new cards count is decremented

  - Tokens Inc: discarded card should increment the amount of tokens
  - NewCardDrawn: Test if the new card is properly drawn on the game screen
  - TimeReset: the timer is reset for the next player
- +Accepted(reply): the reply is sent to the player who made the discard
  - AppendDiscards: test if the card was added to the discard pile
  - Inc Turn: make sure the turn is given to the next player
  - DecCards: verify that the new cards count is decremented
  - Tokens Inc: discarded card should increment the amount of tokens
  - TimeReset: the timer is reset for the next player
- +Played(notice): the server sends this notice to other players, and tell them what new card the player drew
  - UpdatePlayer: test if the new drawn card is now part of the player's hand
  - IncTurn: make sure the turn is given to the next player
  - DecCards: verify that the new cards count is decremented
  - TimeReset: the timer is reset for the next player
  - ScoreUp(If it is a legal play): make sure the score is correctly added
  - UpdateTable(If it is a legal play): verify that the played card is now placed on the correct position of the table
  - DecFuses(If it is NOT a legal play): the fuses should be decreased by 1
- +Built(reply): the reply is sent to the player who played card if the card is legal
  - IncTurn: make sure the turn is given to the next player
  - DecCards: verify that the new cards count is decremented
  - TimeReset: the timer is reset for the next player
  - ScoreUp: make sure the score is correctly added

  - UpdateTable: verify that the played card is now placed on the correct position of the table
  - TokenInc(If the card rank is equal to 5): successfully play a rank 5 card should increase the amount of tokens
- +Burned(reply): the reply is sent to the player who played card if the card is NOT legal
  - IncTurn: make sure the turn is given to the next player
  - DecCards: verify that the new cards count is decremented
  - TimeReset: the timer is reset for the next player
  - DecFuses: the fuses should be decreased by 1 if it is an illegal play
  - AppendDiscards: test if the card was added to the discard pile
- +GameEnds(notice): the server will notify all players if certain conditions are met
  - ShowGameOver: a dialog that contains game over information should be poped up
  - CloseConnection: connection to each player should be closed and game is destroyed
  - GoToMainMenu: users' screen should atomatically jump to main menu after connection is closed
  - SetGameStateNull: game state model should be set to null after previous game is destroyed
- +Inform(notice to informed player):
  - IncTurn: make sure the turn is given to the next player
  - TimeReset: the timer is reset for the next player
  - DecTokens: verify the amount of tokens is decreased after a hint is given
  - UpdateHands: make sure the player's hands is updated accroding to the hint
- +Inform(notice to other players):
  - IncTurn: make sure the turn is given to the next player
  - TimeReset: the timer is reset for the next player

  - DecTokens: verify the amount of tokens is decreased after a hint is given
  - Animation: an information should be given that indicates a hint is given by a player to another

SendMove responsible for sending data to the server

- + server responsed: this method will be test together with receive server data, it will check if there is a reply message sent by server after sendMove method is called. Server has to reply something after it sends a JSON package

PackStream convert a move message from the player's action into a JSON object which can be read by server

- + Right move is constructed: check the information that output object contains, a JSON object should be created with correct action, correct player, and correct card based on the user's input

handleInputEvent listens to mouse events and does functions based on different requests

- + Verify Mouse Event: several mock situation will be given to verify all five mouse events (mouseOver, mouseOut, mousePressed, mouseDraged, mouseReleased) are successfully handled

5 AI

devourStack Move through each move one by one from the stack of moves to create a statistical model for the AI.

- + Given Stack Get CPT: The conditional probability table (CPT) is created with correct dimensions and values.

updateCPT Update the conditional probability tables (CPT) of the AI.

- + Sum of all probabilities: the sum of probabilities from the CPT shall be equal to one.

Part V
Summary

This testing plan was designed to help aid in construction of the game, and guide the coding phase of the project. The end-to-end tests cover the actual usage of the game client, and come directly from the requirements documentation. The integration tests break down the various cases addressed in the end-to-end tests, and map them to corresponding parts in the architecture. Finally, the unit tests address the individual modules (classes and methods) and allow us to confirm correct implementation.