2.1.1 Client Model

The game’s model keeps track of the state of an active game of Hanabi by storing the very core information of the game.

Upon the creation of the game, the Game Creator will define the maximum number of players for it by assigning a value from 2 to 5. Each game has its own unique ID given upon creation with its own secret token for other players to join.

The current number of players is a value from 1 to the maximum number of players allowed (up to 5). This value is updated as Players join/leave the lobby or if AI players are added.

The in-game tokens are split into two categories: fuse and information. Fuse tokens starts with a value of 3 and can only be decreased upon Player’s mistakes. The number of information tokens, on the other hand, can go up or down based on Player’s discard move or give information move respectively.

Each card has its own colour from RED, BLUE, WHITE, YELLOW, GREEN (and depend on the game mode RAINBOW) and rank from 1 to 5.

The player hands, firework piles and discard pile can then be treated as containers, holding a different amount of cards. First, the player hands is equal to the current number of players ranging from 2 to 5. The amount of cards in each player’s hand is 5 except for the moment that they play a card, which will then be replenish right away by drawing a card. Next is the firework pile. The amount of this container is either 5 or 6 depends on how many suits are available. Each container must hold cards of different suits and increased in ranks (1 follows by 2, follows by 3 and so on) up to a maximum of 5. Last container is the discard pile that holds anywhere from 0 cards to all cards in the deck.

The model also contains information log, which are information of each turn in chronological order that can be turn into viewable information for the Players.

**Model**

The central component of the pattern. It is the application's dynamic data structure, independent of the user interface.[[4]](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) It directly manages the data, logic and rules of the application.

**View**

Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

**Controller**

Accepts input and converts it to commands for the model or view.[[5]](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)

Object Description Template

**Class Card**

**Description:** Card is a public class that is responsible for holding information for Rank and Color of a given card. It sets respective values for each card at game launch and provides access to them throughout the game.

**Fields:**

* int rank: Integer data member of Card that describes the value rank of the given card.
* String colour: String data member of Card that describes the color type of the given card.
* boolean rankKnown: Keep track if a Card’s rank is known to the player.
* boolean colourKnown: Keep track if a Card’s colour is known to the player.

**Methods:**

* int getRank(): Returns an integer value that represents the rank of a given card.
* String getColour(): Returns a string that represents the colour of a given card.
* Boolean[] getInfo(): Returns a list that contains rankKnown and colourKnown.

**Class AIController**

**Description:** Since the AI requires all the information about the current game’s state to make its move, we have decided that this class will not follow the Model-View-Controller structure. The class’s sole purpose is to compare the available moves against each other with a PlayValue to decide the best ones for the AI player when their turns come.

**Methods:**

* String[2], int getMove(): […]
* String[2], int bestMove() : Rank all the best moves from bestPlay(), bestDiscard() and bestInfo() together to pick out the best move possible. Return […]
* int[2] play: the index of the card being chosen and the PlayValue for such move
* int[2] discard: the index of the card being chosen and the PlayValue for such move
* int[2] info: the index of the player the AI want to give information to and the PlayValue for such move.
* String info: the information given to the player chosen in string form.
* int bestPlay(): Returns the best card to play from the AI’s hand as an integer. Also return a PlayValue to determine how good the move is.
* Card[4..5] handAI: the current cards on AI’s hand
* Card[0..5, 6] fireworks: the information about the current firework pile
* int bestDiscard(): Returns the best card to discard from the AI’s hand. Also return a PlayValue to determine how good the move is.
* Card[4..5] handAI: the current cards on AI’s hand
* Card[0..5, 6] fireworks: the information about the current firework pile
* Card[\*] discard: the information about the current discard pile
* String, int[2] bestInfo(): Return the best information as a string to the player at an integer index. Also return a PlayValue to determine how good the move is.
* Card[4..5, 1..4] handPlayers: the information about cards on other players’ hands
* Card[0..5, 6] fireworks: the information about the current firework pile

**Class HanabiController**

**Description:** This class taking inputs from the view and convert them into appropriate commands to notify the model to update.

**Fields:**

* game HanabiGame: […]
* String nsid: the NSID of the player who joins or create game
* Boolean isAI: Boolean value that is set to true when it is the AI’s turn.
* ServerComm server: […]

**Methods:**

* void createGame(): the method to create an instance of a game from a player.
* int numPlayers: the maximum number of players allowed to be in the game
* int timeout: the number of seconds for timeout period of the game
* String nsid: the nsid of the player who creates the game
* Boolean force: set to true to cancel the game
* void joinGame(): the method to join a game from a player.
* int id: the game’s id.
* String token: the game’s secret token.
* String nsid: the player’s nsid.
* void leaveGame(): the method to leave the game from a player.
* void addAIPlayer(): the method to add an AI player to the game.
* void tellPlayCard(): the method tells the model to update with a play card move
* int handIndex: the index of the card being played on the current turn
* void tellDiscardCard(): the method tells the model to update with a discard card move
* int handIndex: the index of the card being discarded on the current turn
* void tellGiveInfo(): the method tells the model to update with a give information move
* int playerIndex: the index of the player that the information is given to
* String property: the information given to the player, can be either rank or colour
* void tellToggleDiscard(): […]
* void tellToggleLog(): […]

As seen in the class diagram above, the Controller package contains four classes: HanabiController, ServerComm, JSONParser, and AIController. HanabiController is the main controller class that receives user input and coordinates server communication and model changes. HanabiController uses ServerComm to handle the task of maintaining a Server connection and sending and receiving Server messages. ServerComm itself then uses JSONParser to create and parse the JSON format used by Server messages. HanabiController is also dependent on AIController to determine the moves of AI Players whenever the Client is running with an AI Player.

The Controller package connects to the model mainly through an instance of the main model class, HanabiGame, that HanabiController maintains and uses to communicate state changes to the model. A reference to this instance is also given to AIController so that it can access the game state in the process of determining AI moves. AIController has some extra coupling too, as it depends on the Card class to encapsulate the game state passed around by its methods into fewer inputs.

3.5 AI Getting a Move

When it is an AI’s turn, the methods from AIController from Controller package will be called to decide the best play move possible with the information given at the current turn. Since play a card, discard a card and give information are very similar, we will test AI getting play a card method only. To test this method, we will simulate a situation where the best move is to play the next card in the fireworks pile. At the beginning of the AI’s turn, the fireworks pile will have a Red 1 and in the AI’s hand there is a known Red 2. The state will be given from the model for the AI’s turn and all information about the game (see Section 2.6 in DD). The overall information will then be evaluated in the AIController, resulting in the expected best move possible of playing Red 2.