**Model Description**

As shown in the class diagram above, the Model package stores the state of a Hanabi game.  It does this through encapsulating concepts of a HanabiGame, Log, Action, Token, FireworksPile, DiscardPile, Player, Hand, and Card. HanabiGame is the main class in the model and handles the overall game state, aggregating the various parts of a game and ensuring consistent changes among those parts. A Log, meanwhile, deals with storing all the actions made during a game in the form of Actions, which contain information about a single move made by a Player.

The main primitive entity in the model is the Card, which has a rank, a colour, and flags for whether they are known to a given Player. Each Player has a Hand of cards, and cards can be added to and removed from specific positions in a hand. The DiscardPile and FireworksPiles are other card containers that store cards that have been discarded throughout a game and cards that have been successfully played and are part of built-up fireworks. The model’s other primitive entity is the Token, which deals with tracking fuse and information tokens as they are gained and lost during a game.

From the Controller package the HanabiController depends on the HanabiGame for the games state. In addition, the AIController temporarily uses the Card class to determined the best course of action.

**View Description**

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.

*3.3.3 JSONParser*

Description: The JSONParser will be a public class that is responsible for receiving the messages from the server and making the information available.

Methods:

* public String makeJSON(): String data type that takes in parameters of a String list for names and values. This method will make the JSON messages.
* public String[\*] parseJSON(): String list data type and parses through the data from the server.

*3.3.4 ServerComm*

Description: ServerComm is a public class that is responsible for communicating the JSON messages to the of what has and needs to happen for the game client to continue.

Methods:

* void sendCreate(numPlayers: int, timeout: int, nsid: String, force: bool): Sends the number of players, nsid and the boolean value to start the game to the server.

-   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 sendJoin(id: int, token: String, nsid: String): Sends the id, token and nsid data to the server.

-   int id: the game’s id.

-   String token: the game’s secret token.

-   String nsid: the player’s nsid.

* void sendPlay(handIndex: int): Sends the play action taken by the player to the server.
  + int handIndex: the index of the card being played on the current turn.
* void sendDiscard(handIndex: int): Sends the discard action taken by the player to the server.
  + int handIndex: the index of the card being discarded on the current turn.
* void sendInfo(playerIndex: int, property: String): Sends the information from the player to the server.

-   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 receiveMessage(controller: HanabiController): Receives the messages from the server and directs it to the HanabiController class.
  + HanabiController controller: the main controller class that takes user inputs and tell model to change accordingly

**4. Conclusion**

Overall, this design document describes the high and low level architecture used by the Client. The high level architecture describes the major components and their organization in our case as model-view-controller. Whereas, the low level architecture describes the detailed design of the classes and objects that compose the system and interface. (ADD/EDIT)