**Software Design Specification**

**Hanabi Client G3**

Authored by:

Joel Siemens <jms676>

Olivier Lacoste-Bouchet <oll096>

Troy Belsher <twb431>

Kole Barber <klb731>

Destin Astrope <dja392>

* Justify and describe architecture
* Describe classes, methods (functions) and fields (variables)
* UML diagrams

## Architecture

### Overview

The program will be made with using several different architectures, resulting in a Hybrid architecture. The first is a Simulation architecture based on Model View Controller (MVC) approach. The controller and the interaction model will have an event driven architecture. The AI will have a pipes and filters architecture.

### Model View Controller Architecture

A model view controller architecture was chosen for how well it can be adapted to the Hanabi program. We are using view controller to separate visual code and game state code from AI code and controller code working behind the scenes. The view will display the info stored in the model to the user. The model will contain the information about the game, including what cards are in each hand and what hints have been given about those cards, as well as the remaining time tokens and fuse tokens, the current firework stacks, the remaining turn time, and the discard pile. The view will use this information to build the UI, as well as respond to the player when they are hovering over cards during their turn to decide what to do.

### Event System: Controller and Interaction Model

While in game, the controller is will be based around two states, Active and Inactive which are triggered by server events. The messages from the server will trigger operations on the controller and model, based on the message. The messages from the server can trigger a state from Active to Inactive or from Inactive to Active, where active is a state where the user can play.

The interaction model has several states that are changed by the user’s input. The states are based on the controller’s active and inactive states. They include several Active state substates: Pay, Discard, Inform Colour and Inform Number. These states will change how the mouse interacts with the on screen card objects.

### Pipes and Filters: AI controller

The AI will be have data flowing through pipes and filters. These will be computed in the AI controller. The filters will filter out moves based on a weighing system, which will then choose the best option based on the weighing system.

### Other architectures

There several other architectures that we considered, although we did not use these. A dataflow architecture doesn't make a lot of sense in this situation because each thing our program does doesn't require several methods to be chained together using each others' outputs. It might be possible to use such an architecture for our AI, with various options for moves feeding into one deciding function that tries to choose the best from among them.

A call and return architecture isn't really necessary for this program because we don't need to wait for the results of an action to know what that action has done to the game state (unless it is our own turn, since we don't know the cards in our own hand), and the server does not need input from our client to progress the game. The server might operate under a style of call and return, since the game cannot progress until the person whose turn it is responds, but that isn't within the purview of our project.

A data-centered system might make sense if our game was operating as both server and client, since we could tell each client to trigger its turn when a turn marker got to the client's player number for that particular game, and it would be easy to go looking for what information was needed. It would also, however, necessitate some level of honesty since a player's own hand would also be on the blackboard, and they would just have to be trusted not to look at it. Because the server is set up at arm's length for this project, and would not look at the blackboard, there is no compelling reason to use this architecture.