Design Document

Group H2 - CMPT 370

Amanda Zimolag
Brendan Nykoluk
George Shi
Spencer O'Lain
Xinnan Xie
Zeshan Ahmad

1. High-Level Design

The high-level design of our project follows an event-driven architecture. We chose an event-driven architecture because the flow of the program is largely based on the program's interactions with the user and the server (events). Almost everything that happens in the program is triggered either by a message from the server or from a button press by the user. The diagram detailing this is figure 1 below. The play field is a complex element that has its own simulation style architecture. This is detailed in section 1.2.

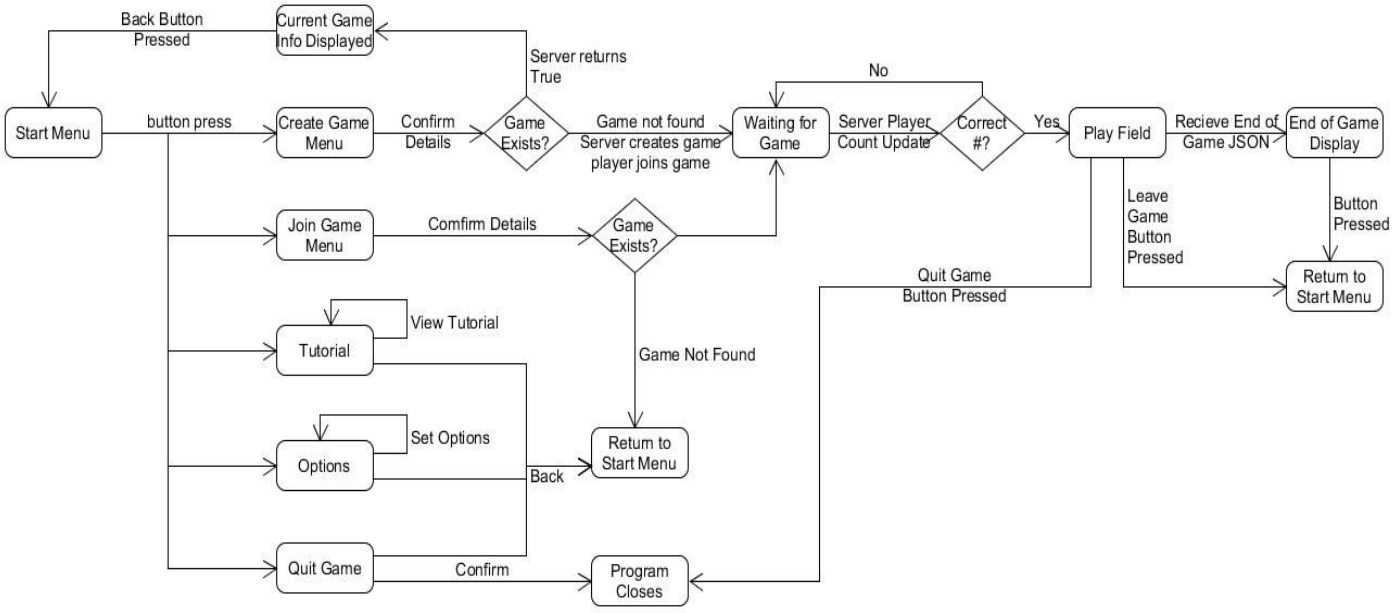


Figure 1 - Event based architecture for pre-game

1.1 Top Level – Event Based

Upon opening our program, the user will enter the start menu state. From this state the user will have five options available in the form of five buttons displayed. A mock-up of this menu is displayed below in figure 2. The options are

- quit game,
- set options,
- run the tutorial,
- join game, and
- create a game.

Figure 2 - Start Menu

At the time of release, the play button will do nothing, and is a placeholder for a solo-play feature to be implemented in the future.

If the user selects the quit game option by pressing the corresponding button, a confirmation popup will appear. If the user presses the confirm button, the program will close. If the user presses the back button, the program will return to the start menu state.

If the user presses the settings button, the program enters the settings state. From here the user will be able to alter the default turn length for creating a server, view help, and view developer information. The user will then be able to press the back button to save their changes and return to the start menu. A depiction of this screen is displayed in figure 3.

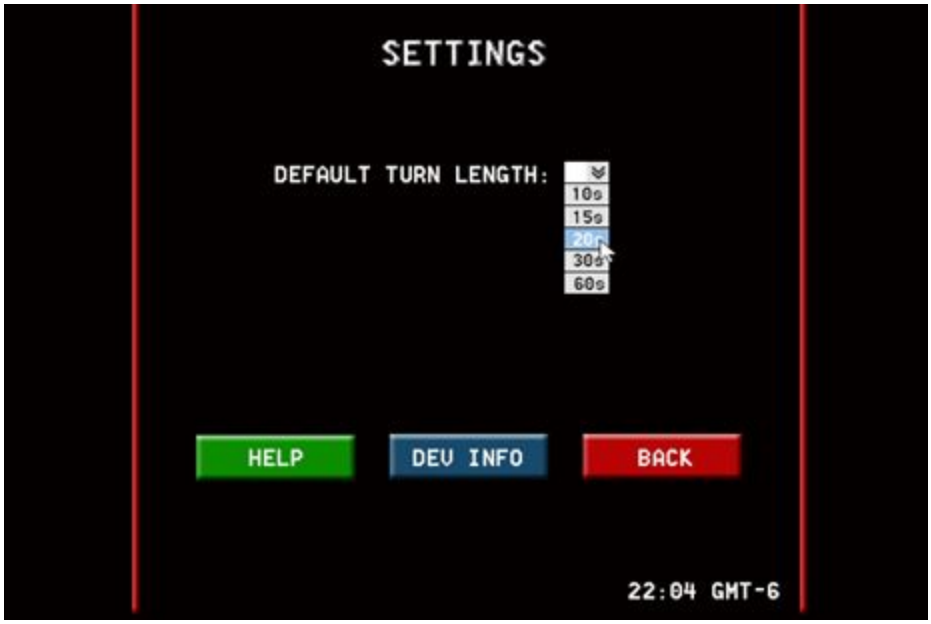


Figure 2 - The Settings Screen.

If the user presses the tutorial button, they will be taken to the first in a series of images that will guide them through the basics of Hanabi as well as our game screen interface. From here there will be two buttons: one to advance through the pictures and another to return to the start menu.

On pressing the join game button, the user will be prompted for the information of the server they are attempting to connect to. If this information corresponds to a valid server, the game will attempt to connect. On a successful connection, the program will be in the waiting state until the server informs the program that all the required players are present at which point the player is taken to the play field state. If the information does not match a game that currently exists, the user is taken back to the start menu. This sub-menu will be as it appears in figure 4.

Figure 3 - The Join Game Sub-menu

Figure 4 - The Create Server Menu.

The final option available to the user from the start menu is to create a server. On pressing this button, the user will be taken to a sub-menu like the one pictured in figure 4. From here the user can press the back button, returning them to the start menu state. The user's other option is to create a new server after specifying the number of players and timeout length. This will send a message to the network asking if the current user has a game created already. If the server detects a game already made by this user, it will cancel the create attempt. A pop-up will be displayed to the user informing them of the existing game's information, and a button press will return the user to the start menu. If the game doesn't already exist, the server creates a new game and adds the client to the game. The user then enters the same waiting loop that occurs when joining an existing game. Once the server indicates all players are present, the program moves into the play field state.

Once in the play field state, there are three ways that the state can be exited. The play field is pictured in figure 5 below. The first is by the user pressing the quit game button. This will exit the program. The next is by the user pressing the leave game button. This will disconnect the client from the server and return the state to the start menu. The final way for the client to leave the play field state is for the server to issue an end game message. Upon receiving this, the client will go into the end of game state where the user's score and a corresponding fireworks display will be shown. The user will then return to the start menu state by pressing a confirm button.

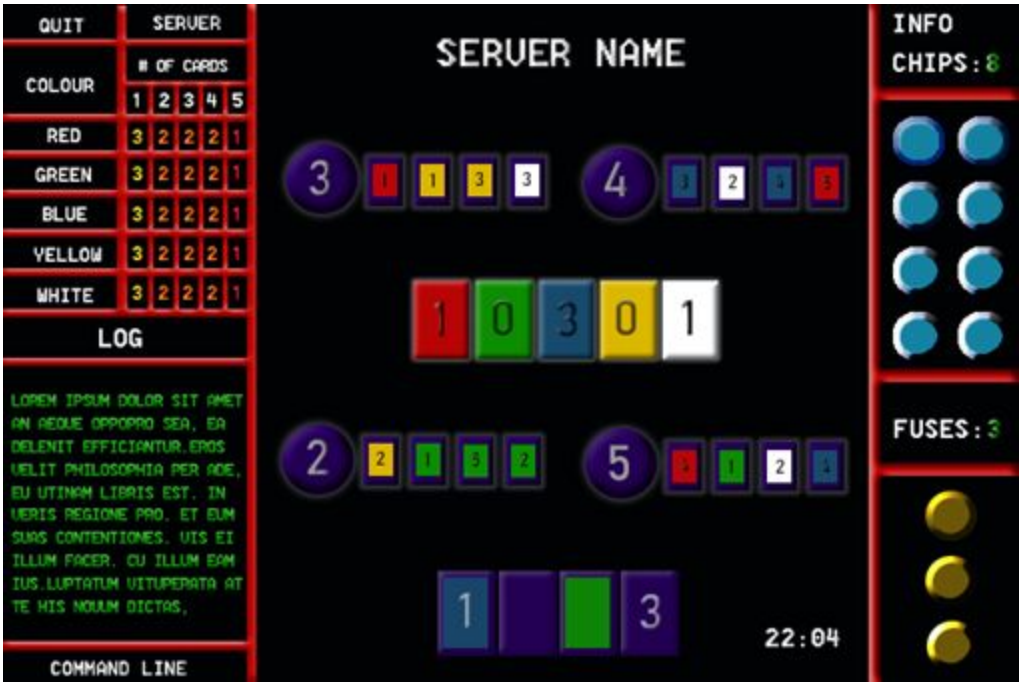


Figure 5 - Play Field Interface

1.2 Play Field

The play field of our program will follow the virtual machine architecture. It will be represented by a model-view-controller system that uses the publish-subscribe design pattern. We are using this architecture because it allows for a separation between the user interfaces and the logic. This makes testing the main system easier since it can be done without needing to have any of the user interface functional. This also makes it possible to update the user interface in the future without having to make any changes to the underlying logic. A simple depiction of our play field architecture can be seen below in figure 6.

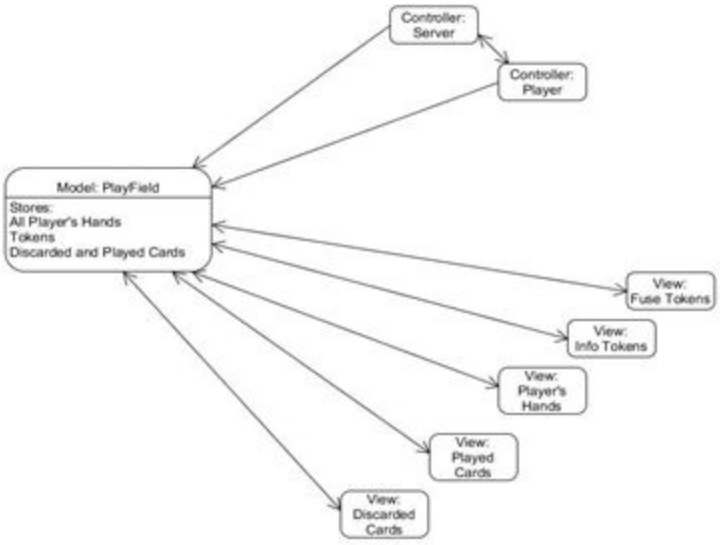


Figure 6 - Virtual Machine Architecture for Play Field

The system has five separate views. They will all be displayed simultaneously but are separate because many moves will only affect some of them. This will allow for the system to refresh only the necessary views on a change within the model.

The fuse and information token views will be a visual representation of the number of tokens remaining. The next views are the player's hands. Each player's hand will be a separate view so that only one needs to be updated on any given move. The user's hand will be a special version of the view capable of displaying cards with missing suit and rank. Next, the played cards view will show the stacks of successfully played cards towards the firework display. Finally, the discarded cards view will be a table showing all the cards that have been discarded by suit and rank.

The model for the playfield will be where all the data for the system is stored. This includes

- the cards in each player's hand,
- the fuse token count,
- the information token count,
- the cards that have been discarded, and
- the cards that have been successfully played.

It will also contain the functionality to update the required fields when a card is played, discarded, or a hint given by the user. It will also be able to update fields when another user makes a move and call the end of game function when the message is received.

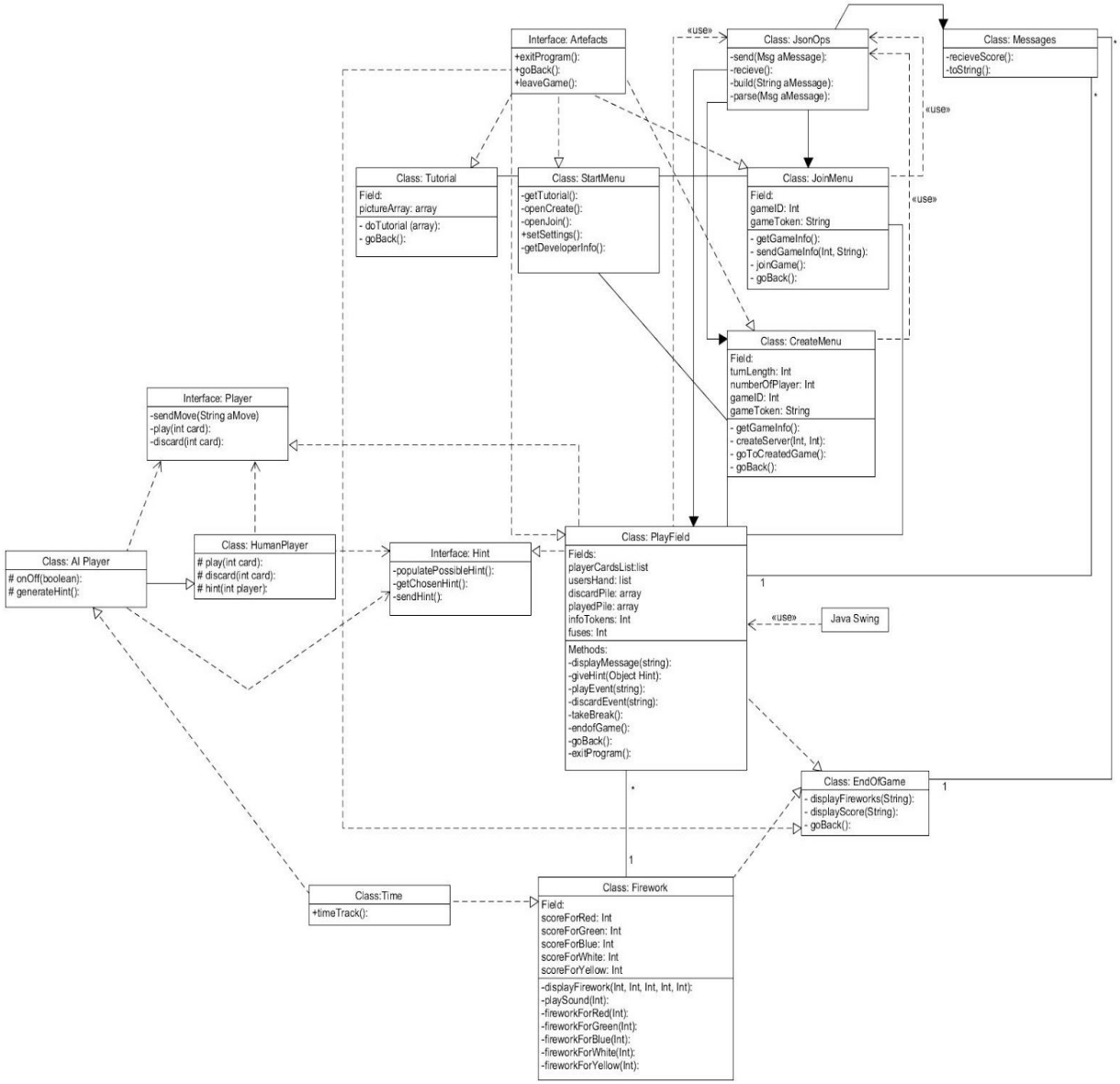
The system has two controllers: the player and the server. The player controller can either be operated by a human or AI. It is responsible for selecting the move for the user. The human player will do this through a series of button presses. The AI will select a move based on a decision-making tree. When the move is selected a message is sent to the model telling it the move

being attempted. The player will then send a message to the server with the move being attempted. If the move is successful, the model goes through with the move given by updating the necessary fields. If the move is invalid, the model ignores the player and waits for a new move. The player is also informed by the server in this case to choose a new move.

The other controller is the server. The server has three main duties. The first is to inform the player when it is their turn by prompting the player for a move. This is described above in the player controller description. The second duty of the server is to inform the model of the moves of the other users. The final duty of the server is to send the end game message to the playfield. This allows it to initiate the end game sequence.

2. Class Level design.

The following section highlights the classes we chose to use for the implementation of our client. We have also provided a class diagram which will show the way each class interacts with each other.


Interface: Artefacts
+exitProgram():
+goBack():
+leaveGame():
«use»
Class: JsonOps
-send(Msg aMessage):
-recieve():
-build(String aMessage):
-parse(Msg aMessage):
Class: Messages
-recieveScore():
-toString():
«use»
Class: Tutorial
Field:
pictureArray: array
- doTutorial (array):
- goBack():
Class: StartMenu
-getTutorial():
-openCreate():
-openJoin():
+setSettings():
-getDeveloperInfo():
Class: JoinMenu
Field:
gameID: Int
gameToken: String
- getGameInfo():
- sendGameInfo(Int, String):
- joinGame():
- goBack():
«use»
Class: CreateMenu
Field:
turnLength: Int
numberOfPlayer: Int
gameID: Int
gameToken: String
- getGameInfo():
- createServer(Int, Int):
- goToCreatedGame():
- goBack():
Interface: Player
-sendMove(String aMove)
-play(int card):
-discard(int card):
Class: AI Player
onOff(boolean):
generateHint():
Class: HumanPlayer
play(int card):
discard(int card):
hint(int player):
Interface: Hint
-populatePossibleHint():
-getChosenHint():
-sendHint():
Class: PlayField
Fields:
playerCardsList:list
usersHand: list
discardPile: array
playedPile: array
infoTokens: Int
fuses: Int
Methods:
-displayMessage(string):
-giveHint(Object Hint):
-playEvent(string):
-discardEvent(string):
-takeBreak():
-endofGame():
-goBack():
-exitProgram():
«use»
Java Swing
1
*
Class: EndOfGame
- displayFireworks(String):
- displayScore(String):
- goBack():
1
1
Class:Time
+timeTrack():
Class: Firework
Field:
scoreForRed: Int
scoreForGreen: Int
scoreForBlue: Int
scoreForWhite: Int
scoreForYellow: Int
-displayFirework(Int, Int, Int, Int, Int):
-playSound(Int):
-fireworkForRed(Int):
-fireworkForGreen(Int):
-fireworkForBlue(Int):
-fireworkForWhite(Int):
-fireworkForYellow(Int):

CreateMenu

The CreateMenu class will allow the user to create a new game by entering some information and it will move the player to the play field related to that server. These actions will be the sole purpose of this class. When the user clicks the “Create Server” button, the options will be displayed to the user in the form of boxes. These options include, the length of each players turn and the number of players who will be in the game. The CreateMenu have 4 methods and 4 fields.

Fields:

turnLength: This field will store the length of the players turn as an integer.

numberofPlayer: This field will store the number of players for the game to be created.

gameID: This field will store the Game Identification Number as an integer.

gameToken: This field will store the secret token for the server as a string.

Methods:

createServer(int, int): This method will read the information that the user chose and send it to the JsonOps.

Parameter: 2 Integers; one is the length of a turn and the other is the number of players.

Pre-condition: The user is in the start menu.

Post-condition: The details of the game to be created are sent to JsonOps.

getGameInfo():This method will receive a message sent from JsonOps which contains the game ID and the secret token.

Parameter: None.

Pre-condition: When the JsonOps receive the message from sever.

Post-condition: This method will modify the data base on the message that it received.

goToCreateGame(): This method sends the user to the Playfield to wait for other players to join.

Parameter: None.

Pre-condition: The user has information to connect to a valid server.

Post-condition: The program moves to the play field.

goBack(): This method returns the user to the Start Menu.

Parameters: None.

Pre-conditions: The user is in the create game menu.

Post-conditions: The user is returned to the start menu.

Interactions:

StartMenu: The start menu calls CreateMenu when the user presses the Create Game button.

PlayField: The PlayField is where the user goes when they sucessfully join a game.

JsonOps: CreateMenu uses JsonOps to communicate with the server about creating new games.

Tutorial

The Tutorial class contains the methods and fields that will provide the user with a sequence of that describes the way to play the game in our client.

Fields:

array pictureArray: An array of pictures that will be cycled through by the user. These pictures will be examples on navigating the graphical user interface and the rules of the game.

Methods:

doTutorial(array pictures): The method doTutorial will display each of the pictures in the array.

Preconditions: The user must be in the Tutorial menu.

There should be a valid array of pictures.

Post-conditions: None.

Return: The method will return a 1 on success and a 0 on failure.

goBack(): The method goBack will be responsible for allowing the user to return to the start menu.

Preconditions: The user must be in the Tutorial menu.

Post-conditions: The user should be taken to the start menu.

Return: The method will return a 1 on success and a 0 on failure.

Interactions:

StartMenu: The Tutorial will be accessed only through the interactions and methods of the start menu.

Artefacts: The artifacts interface will provide the Tutorial class with the methods to navigate to the previous menu.

JoinMenu

The JoinMenu class will be responsible for providing the methods which will allow the user to join a game by entering the game ID and game token which will allow the user to join a game with the client.

Fields

int gameID**:** The gameID will be the field responsible for containing the ID of the game that is hosted on the server that the user wants to connect to.

int gameToken**:** The gameToken will be used by the client to access the server that is described by the gameID of the server that will be joined.

Methods

getGameInfo(): getGameInfo will allow the user to enter in the game token and game ID of the server they would like to join.

Preconditions: The server that the user wants to connect to must be created.

The user should enter a valid form of gameID.

Post-conditions: The fields gameID and gameToken will store the game ID and game token provided by the user of the desired server.

Return: The method will return a 1 on success and a 0 on failure.

sendGameInfo (int gameID, String token): The method sendGameInfo will send the game ID and game token to the server that the user wants to connect to.

Preconditions: The user must be in the Tutorial menu.

Post-conditions: The user should be taken to the game lobby.

Return: The method will return a 1 on a successful connection and 0 on failure.

StartMenu

The StartMenu class displays the Start Menu for the user when the program starts up.. There are three cases where this class will be used. The first one is the when the user starts the program. The second case where it will be used is when a game is finished. The user will be sent back to the StartMenu. The last case where it will be used is when user click “back” button in others menu. In this case, user will send back to the StartMenu.

Methods:

getTutorial():This method will send the user to the Tutorial Menu.

Parameter: None.

Pre-condition: The user is in the Start Menu.

Post-condition: The Start Menu closes and the Tutorial opens.

openCreate(): This method will send the user to the Create Menu.

Parameter: None.

Pre-condition: The user is in the Start Menu.

Post-condition: The Start Menu closes and the Create Game Menu opens.

openJoin(): This method will send the user to the Join Menu.

Parameter: None.

Pre-condition: The user is in the Start Menu.

Post-condition: The Start Menu closes and the Join Game Menu opens.

setSetting(): This method will send the user to the Settings Menu.

Parameter: None.

Pre-condition: The user is in the Start Menu.

Post-condition: The Start Menu closes and the Settings Menu opens.

getDeveloperinfo(): This method will display the Developer information on the screen.

Parameter: None.

Pre-condition: The user is in the start menu.

Post-condition: The developer information will be displayed.

Interactions:

Tutorial: The user can move to the tutorial class from the Start Menu.

CreateMenu: The user can move to the Create Game menu from the Start Menu.

JoinMenu: The user can move to the Join Game menu from the Start Menu.

Artefacts: The StartMenu class implement the Artefacts class, which it allows StartMenu to use the back () and exit () functions.

Artefacts

Artefacts is an interface which will provide the methods for the user to go back to the previous menus or quit the game. It is connected with the menu classes and some of the display classes as well. Therefore, the user can quit the program or go back to previous page in the menus which impliment the interface. The Artefacts interface will have 3 methods and no fields.

Method:

exitProgram(): This method will give the user the ability to exit the program.

Parameter: None.

Pre-condition: None.

Post-condition: The program will close.

goBack(): This method will allow the user to go back to the previous page.

Parameter: None.

Pre-condition: None.

Post-condition: The user will leave the current page and go back to the previous page.

leaveGame(): This method will allow the user to quit the current game and return to the start menu.

Parameter: None.

Pre-condition: The user is connected to a server..

Post-condition: The user will be sent back to the Start Menu.

Classes that Implement Artefacts:

JoinMenu

CreateMenu.

StartMenu

Tutorial

EndOfGame

Messages Class

The messages class exists to convert complicated information into a simple string that can be displayed to the user. There are two cases where this will be used. The first is when the server returns a message about an invalid move or a hint being given to the user. In this case, the JSON packet will be converted into a sentence for the user to read. This sentence will then be sent as a string to the PlayField to be displayed. The other case where it will be used is at the end of the game. The messages class will take in the final score and prepare a message based on that score. That message will then be sent to the EndOfGame class to be displayed. The Messages class has no fields and two methods.

Methods:

recieveScore(): This method will ask the PlayField class for the final score of the game. This is used to generate the end of game message.

Parameters: None.

Preconditions: A game has ended.

Return: A string containing the results of the game.

toString(): This method will take either a JSON packet or no argument. If the argument is a JSON packet, the relevant information will be taken and converted to a string. If there is no argument, receiveScore will be called and a message generated based on the final score achieved.

Parameters: A json to be converted to a string, or nothing.

Preconditions: None.

Return: A string containing information to be displayed by the user.

Interactions:

JsonOps: JsonOps sends packets that need to be displayed to the user to Messages.
PlayField: Playfield displays messages generated by Messages.
EndOfGame: EndOfGame displays messages generated by Messages.

PlayField

We will be using the model view controller architecture for the purposes of acting on, storing and displaying the information in the game. PlayField will be responsible for storing the game information, acting as our model.

Fields

list playerCardsList: This list will contain a string tuple (suit, number) of the hands of all the players besides the user.

list usersHand: This list will contain string tuples that represents the cards in the players hand. Starts out as empty strings for each card, changing when the player is given hints about his hand.

int fuses: The fuse tokens will be represented by an integer. Starting at 3 and the game ending once they have reached 0.

int infoTokens: The information tokens will be represented by an integer. Starting with a value of 8.

array discardPile: This array will contain all the cards which were discarded during the game.

array playedPile: This array will contain 5 other stacks of cards. These cards will represent the play pile of cards for each color with the most recently played card at the top.

Methods

displayMessage (string aMessage): displayMessage will create a message object with its string argument. This message object will display the relevant message to the user.

Preconditions: The user is connected to the server.

An event was triggered which required a message to be displayed.

Post-conditions: The message is created to be displayed to the user.

Return: 1 on a successful display or 0 on failure.

giveHint (Object Hint): giveHint will use the Hint object to update the players hand who will be

Preconditions: A valid hint was given to a player.

The player who is hinting is doing so on his/her turn.

There are more than 0 info tokens available on the play field.

The player must be connected to the server.

Post-conditions: The player will receive the hint and the information that was hinted at is revealed in his hand. The turn will then change.

If the hint has failed prompt the player for another move.

Return: 1 on a successful hint and 0 on failure

playEvent (string): Given a string which represents the action of playing a card, playEvent will change the played pile if the card was the next card in one of the card stacks. Otherwise the fuse tokens will be triggered.

Preconditions: It must be the players turn to play the card.

The player must be connected to the server.

Post-conditions: If the card is the next number of the stack of its suit, the play pile will be updated. The player replaces the played card with a new card in their hand.

If the card breaks the sequence of its suit, the fuse counter is decremented and the discard pile is updated. The player replaces the played card with a new card in their hand.

If the moves was not accepted by the server prompt the user for another move.

Return: 1 if the card is played with either Post-condition. 0 if the move was not accepted by the server.

discardEvent (string): Given a string which represents the action of discarding a card from the users hand, discardEvent will discard the card replenishing a information token and updating the players hand with a new card as well as the discard pile with the played card.

Preconditions: It is the players turn to discard a card.

There are less than 8 information tokens on the Field.

The player must be connected to the server.

Post-conditions: The discard pile is updated with the card that was discarded and an information token counter is incremented by 1.

If the user cannot discard a card prompt the user for another move.

Return: 1 if the discard was successful and 0 if it failed.

takeBreak (): Once the user decides to toggle on the take a break mode, the users move are then decided by the AI implemented by the client.

Preconditions: The player is connected to the server.

Post-conditions: The player has his moves decided for him by the AI we implement.

Return: 1 if the change was successful and 0 otherwise.

endofGame (): When the game reaches a terminal state with the score, fuse tokens or the server sends a end of game message, endofGame will run to start the process of ending the game where the Fireworks and score will be displayed.

Preconditions: The game has ended.

Post-conditions: The display is changed to the end of game screen where the score and fire works will be displayed.

Return: 1 if the routine ran without failures and 0 if there was a failure when it displays the score or the fireworks.

goBack(): goBack will allow the user to leave the game and end up in the start menu.

Preconditions: The player must be connected to the server.

Post-conditions: The user will be leaving the game.

The user is taken to the start menu.

The game will end due to the player disconnecting.

Return: 1 if the menu change was successful and 0 if it was not.

exitProgram (): exitProgram will allow the user to quit the client.

Preconditions: The user must be connected to the server.

Post-conditions: The user will be disconnected from the server.

The game the user was in will be terminated.

The user will quit the client and return to the OS.

Return: 1 if the client quits without errors and 0 if there was an error detected.

Interactions:

Player: The interface will provide the methods for the players to interact with the Playfield values to progress the game. Example playing a card will update the play pile, fuse tokens, etc.

Hint: The Hint interface will provide the methods to the Human and AI player classes to update the hands of other players with information they specify with their hints.

JsonOps: The provides quite a bit of game information to the play field. It will tell us if a move is valid so we can continue to play it and it will send us end of game messages in the event a player leaves the game or plays too many illegal moves.

Messages: Message objects will be created to display the messages that represent events that occur due to the values of the fields found in the PlayField. For example, information tokens and the restrictions that come with certain number of tokens on the board.

Firework: Once the end of game has been reached, The Firework class will need the values from the PlayField class to display the correct firework for each suit.

EndOfGame: Once the end of game has been reached, the EndOfGame class will calculate the final score which will be displayed to the users in the game. This final score will be calculated with the fields in PlayField.

Hint

The Hint class is an interface which will provide methods for the player classes to choose a hint and send it to the JsonOps class. This class has three methods and no fields.

Method:

populatePossibleHint (): This method will calculate the possible hints that the user can give to the target player, not allowing any invalid hints.

Parameter: None

Pre-condition: The user is in the hint sub-menu.

Post-condition: This method will return a list of hints that user can select.

getChosenHint ():This method will get the hint that the player has chosen.

Parameter: None

Pre-condition: When user clicks on a valid hint for another player.

Post-condition: The hint is stored and ready to be sent to JsonOps.

SendHint():This method will send a message containing the chosen hint to the JsonOps class.

Parameter: None

Pre-condition: A valid hint is selected.

Post-condition: A message containing the clue that the user picked will be sent to the JsonOps class.

Interactions:

HumanPlayer: The HumanPlayer class will use the Hint class as an interface. It uses the methods in the Hint class to create the hint that the player may send.

AIPlayer: The AIPlayer uses the Hint class as an interface. It uses the methods in the Hint class to create the hint that the player can send.

Player

Player is an interface class which provides methods for the player to select their move. This class has three methods and no fields.

Method:

sendMove(String aMove): This method will take the move that the user selected and send it to the PlayField.

Parameter: String, which represents the move type.

Pre-condition: The user has selected a valid move.

Post-condition: The move is sent to the PlayField.

play (int card): This method will play the target card selected by the player.

Parameter: An integer that represents the card's position in the player's hand.

Pre-condition: It is the user's turn to make a move.

Post-condition: The card to be played is sent to the PlayField.

discard (int card): This method will discard the target card that player picked.

Parameter: An integer that represents the card's position in the player's hand.

Pre-condition: It is the user's turn to make a move.

Post-condition: The discarded card is sent to the PlayField.

Interactions:

HumanPlayer: The HumanPlayer implements the Player interface. It uses methods in Player to allow the user to make a move.

AIPlayer: The AIPlayer implements the Player interface. It uses methods in Player to allow the artificial intelligence to make a move.

Playfield: The playfield can realize the moves of classes that implement Player.

HumanPlayer:

The HumanPlayer class is what is used to collect the moves made by a human user. It provides three different moves for the user to pick. This class has three method and no fields.

Methods:

play (int card): This method will play the target card selected by the user.

Parameter: Integer that represent the card's position in the player's hand.

Pre-condition: It is the user's turn to make a move.

Post-condition: The card to be played is sent to the PlayField.

discard (int card): This method will discard the target card that player picked.

Parameter: Integer that represent the card's position in the player's hand.

Pre-condition: It is the user's turn to make a move.

Post-condition: The card to be discarded is sent to the PlayField.

hint (int player): This method will send the hint to the playfield.

Parameter: Integer that represent which player the hint will be sent to.

Pre-condition: It is the user's turn to make a move.

Post-condition: The chosen hint type and the player it is being sent to are sent to the PlayField class.

Interactions:

Player: The HumanPlayer implements the Player interface. It uses methods from Player to perform the play, discard, and hint actions.

AiPlayer: The HumanPlayer class is the super class of the AI Player class.

Ai Player:

The AI Player class will have the same functionality as the player class with the ability to play, discard and give a hint. To generate which move to use it will utilize a class to generate a decision tree with the possible moves to make and a heuristic to weight the various choices. As the heuristic it will utilize a modified formula based in part on the system used to “count cards” in the card game blackjack. Cards that are played or discarded will be tracked and there will be a current count for each color of cards. Based on which card is played or discarded it will continually adjust the current count for each color stack. Based off of this and the optimal times to play and discard a mini max algorithm will come up with an “optimal” play. The AI Player class will then send the move to “Playfield” and “JsonOPS” which will then do any further actions that need to take place. The AIPlayer class will also have a simple Boolean method to turn it on or off, true or false for the bathroom break feature.

Methods:

onOff(Boolean turnOn): Toggles the activation of the AI player.
Parameters: turnOn is a Boolean that is true if the AI is to be turned on.
Preconditions: None.
Postconditions: The AI’s state is that of turnOn.

generate(): Generates a hint to be sent to another player.
Parameters: None
Preconditions: The AI Player is asked to generate a hint.
Return: A string containing a hint.

Interactions:

Player: AIPlayer implements the Player interface. Some of the player methods are used to make moves.
HumanPlayer: HumanPlayer is the superclass of AIPlayer.

EndOfGame

The EndOfGame class is responsible calculating the score and deciding on the display of fireworks for each suit.

Methods

displayFireworks (String aFirework): The method will take in a string which represents the type and

colour of the fireworks. Allowing the method to start the fireworks show.

Preconditions: The game has ended.

The score has been calculated for each suit.

Post-conditions: The fireworks show will start.

Return: 1 on success and 0 on failure.

displayScore (String aScore): The displayScore method will responsible for displaying the score in the

final end game screen.

Preconditions: The game has ended.

The score has been calculated for each suit.

Post-conditions: A final score will be displayed to the user.

Return: 1 on success and 0 on failure.

goBack (): After the fireworks show the game has ended and the file GoBack will return the user to the

startMenu.

Preconditions: The game has ended.

The fireworks show and final score has been displayed.

Post-conditions: The user has been taken back to the start menu.

Return: 1 on success and 0 on failure.

Interactions

Artifacts: The artifact interface will provide the EndOfGame class with the method to go back to the start menu after the score and fireworks have been displayed.

Messages: Messages will receive the score from PlayField and allow EndOfGame to access it for score display.

PlayField: The EndOfGame class will need the playPile information from the PlayField class to calculate and display the score.

Firework: Firework class will be responsible for providing the firework graphic for EndOfGame to display.

Fireworks:

The fireworks class will be utilized to build our fireworks show. The class will mainly deal with graphics and setting up a complete show to utilize in our end of game sequence of events. The class will be given a string comprised of 5 other strings which will represent the top 5 cards in our played card fields in our playfield class. The strings will follow the format of (color number) and the whole string will follow the format of (color number, color number, color number, color number, color number) for example (g2,r3,b4,w1,r2). We have 5 types of fireworks and each can be any of the five colors. Once the show is compiled the graphics and sounds will be outputted by the endOfGame class.

Fields:

Int ScoreForRed: The maximum rank that player successfully played in the red play pile.
Int ScoreForGreen: The maximum rank that player successfully played in the green play pile.
Int ScoreForBlue: The maximum rank that player successfully played in the blue play pile.
Int ScoreForWhite: The maximum rank that player successfully played in the white play pile.
Int ScoreForYellow: The maximum rank that player successfully played in the yellow play pile.

Methods:

displayFirework(Int, Int, Int, Int, Int): the display method will take the final score for the five colors and display the corresponding fireworks.
Parameter: The final score for each color as integers.

Precondition: A game ends.
Postcondition: The firework are displayed on the screen.

playSound(Int): The playSound method will play the sound of the firework when the firework is playing.
Parameter: Integer is the level of the firework sound.
Precondition: A firework is being displayed.
Postcondition: The firework sound will play.

fireworkForRed(Int): This method will make the red color firework depending on the input score.
Parameter: An integer containing the final score of the red play pile.
Precondition: The displayFirework method is called.
Postcondition: The red firework will display.

fireworkForGreen(Int): This method will make the green color firework depend on the input score.
Parameter: An integer containing the final score of the green play pile.
Precondition: The displayFirework method is called
Postcondition: The green firework will display.

fireworkForBlue(Int): This method will make the blue color firework depend on the input score.
Parameter: An integer containing the final score of the blue play pile.
Precondition: The displayFirework method is called
Postcondition: The blue firework will display.

fireworkForWhite(Int): This method will make the white color firework depending on the input score.
Parameter: An integer containing the final score of the white play pile.
Precondition: The displayFirework method is called.
Postcondition: The white firework will display.

fireworkForYellow(Int): This method will make the yellow color firework depending on the input score.
Parameter: An integer containing the final score of the yellow play pile.
Precondition: The displayFirework method is called.
Postcondition: The yellow firework will display.

Interactions

EndOfGame: The fireworks will be displayed by EndOfGame.
Time: Time is used to choreograph the firework display.

Time Class:

The time class will implement a trackTime method, to do exactly that. The class will utilize some of java's built in time functions to get the real time and track the time as the program runs. It will record and track various time intervals to initiate various events that have to do with the AIPlayer class and the fireworks class. For the ai player class it will monitor the time that is being taken to make the ai players move so it does not go over the specified turn length. It will also be utilized to implement the search cut-off in the ai player class. As well in the fireworks class it will in a sense choreograph the fireworks show. It will initiate and end various sounds and animations and we will utilize java's built in time functions to initiate certain firework animations and sounds to make a whole show. It will be used to make sure that the show is set off in an orderly sequence which and also ensure the graphics and sounds pair up as the show progresses to completion.

Methods:

timeTrack (): the timeTrack method will track the real time as the program runs.

Precondition: None.

Postcondition: None.

Return: A time interval as an integer.

Interactions:

AI Player: AI Player use time class to monitor time to make a decision within the given turn length.
Firework: Time is used to choreograph the firework display.

JsonOps Class

The JsonOps class is how our project will communicate with the server. Its purpose is to receive JSON packets, parse them for important information, build new JSON packets, and send JSON packets back to the server. It is connected every other class that needs to send or receive information from the server. The JsonOps class has four methods and no fields.

Method:

send(JSON aMessage): The send method takes a generated JSON packet and sends it to the server.

Parameters: aMessage is a Json packet to be sent to the server.

Preconditions: None.

Postconditions: The server receives a Json packet.

receive(): The receive method collects the most recent JSON packet from the server.

Parameters: None.

Preconditions: The server sends a Json packet to the program.

Return: A string containing the information from the Json packet.

build(): The build method will take a series of information (depending on the JSON to be created) and will make it into a JSON packet to be sent to the server.

Parameters: Information relevant to the Json being created.

Preconditions: None.

Return: A Json message to be sent to the server.

parse(JSON aMessage): The parse method will take a JSON packet, figure out where it needs to go, and convert the JSON message into information appropriate for the receiver.

Parameters: A Json Packet

Preconditions: None

Postconditions: Information from aMessage is sent to the class that requires it.

Interactions:

CreateMenu: CreateMenu sends and receives messages from the server through JsonOps.

JoinMenu: JoinMenu sends and receives messages from the server through JsonOps.

Player: The classes that implement player send and receive messages from the server through JsonOps.

GameField: GameField receives messages from the server through JsonOps.