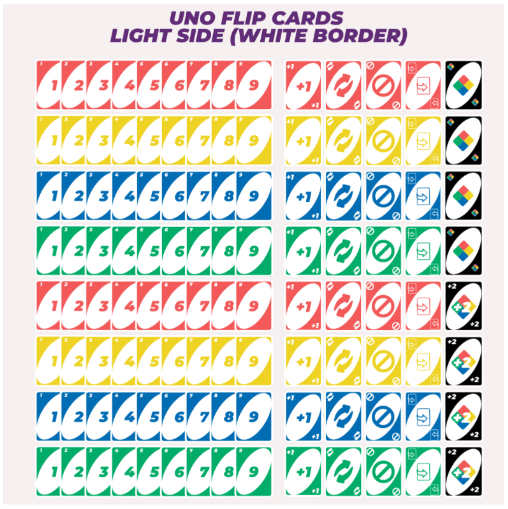
Documentation

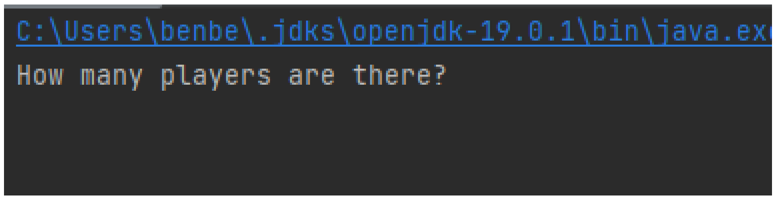
Deliverables

Milestone 2 is a GUI based of UNO FLIP. In this milestone our group has implemented all of the UNO FLIP light card rules, multiple players selection, player score and game winner checker with graphical user interface. The entire project is split into different classes according to the Model Control View design pattern.



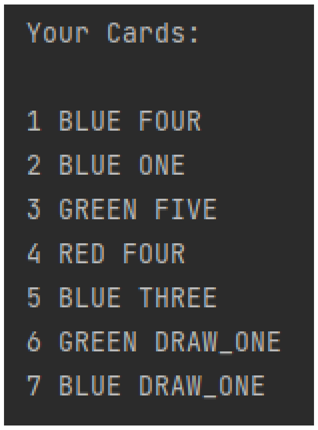
Complete User Manuals

1. Open the CMD
2. Java -jar UnoFlip.jar
3. Select the number of players by typing number 1-4



```
C:\Users\benbe\.jdks\openjdk-19.0.1\bin\java.ex
How many players are there?
```

4. After the selection of players. The game will give you a hand of cards. You can make a selection of cards by choosing the corresponding number.



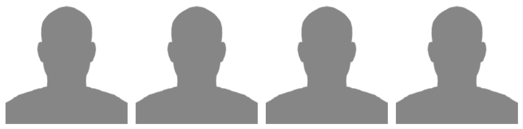
```
Your Cards:

1 BLUE FOUR
2 BLUE ONE
3 GREEN FIVE
4 RED FOUR
5 BLUE THREE
6 GREEN DRAW_ONE
7 BLUE DRAW_ONE
```

New feature added in milestone 2

- Card generator: Card can be greeted randomly by adjusting colour, rank, special card according to the UNO FLIP rule.
- Mapping each card into corresponding location in the folder to have a graphical view of the card.
-

- Get n cards: deck class can extract n number of random cards. Corresponding to their image location
- Add players of 2-4 with graphical view



- GUI view in console: user can now play and visualize the game with graphical user interface.
- Users can play multiple rounds in one Uno game, with score tracking between them.

Known Issues:

- When deck is empty the game finishes and exits instead of restarting
- Top card is unknown

Group 24 - Milestone 2

UML Diagrams

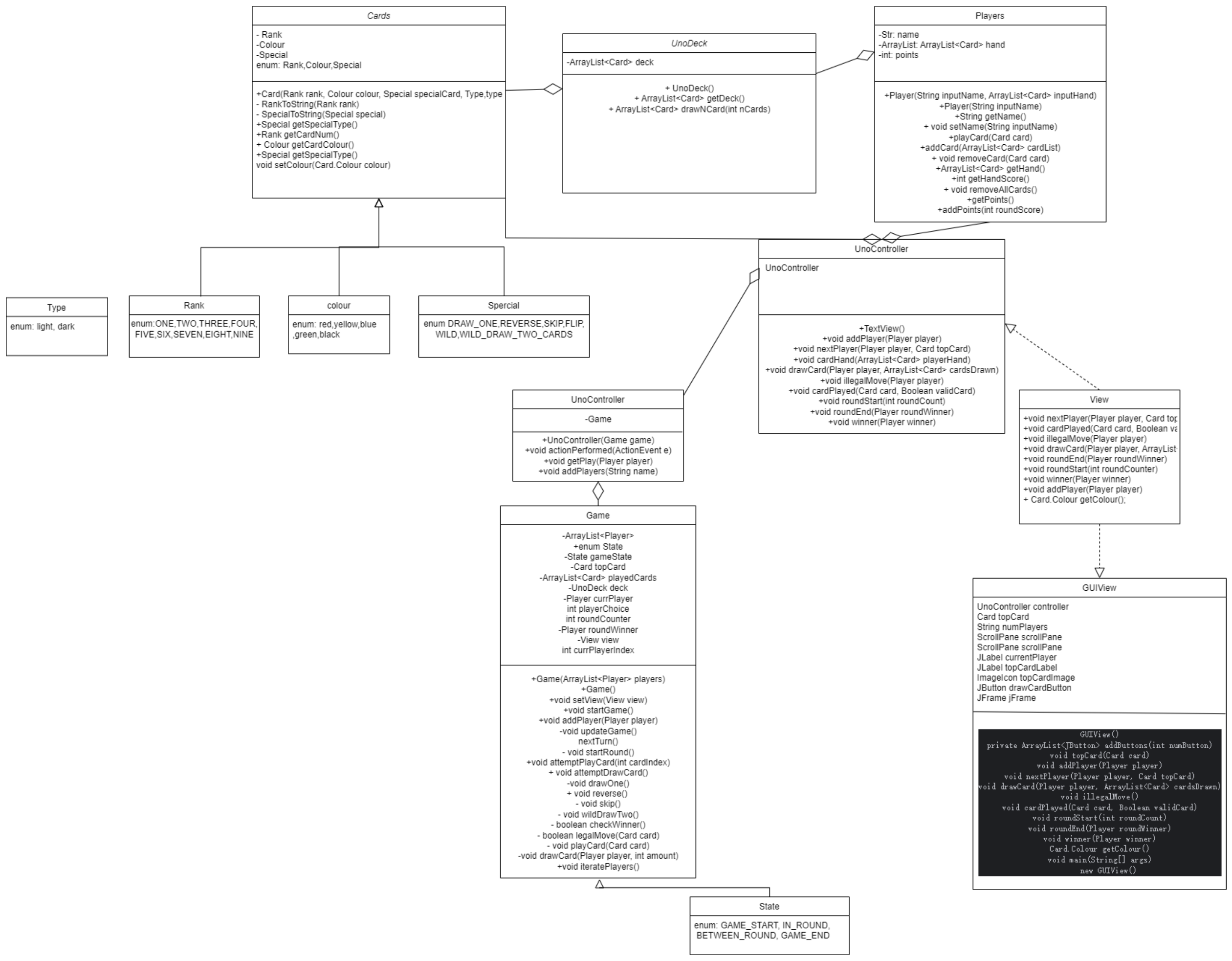
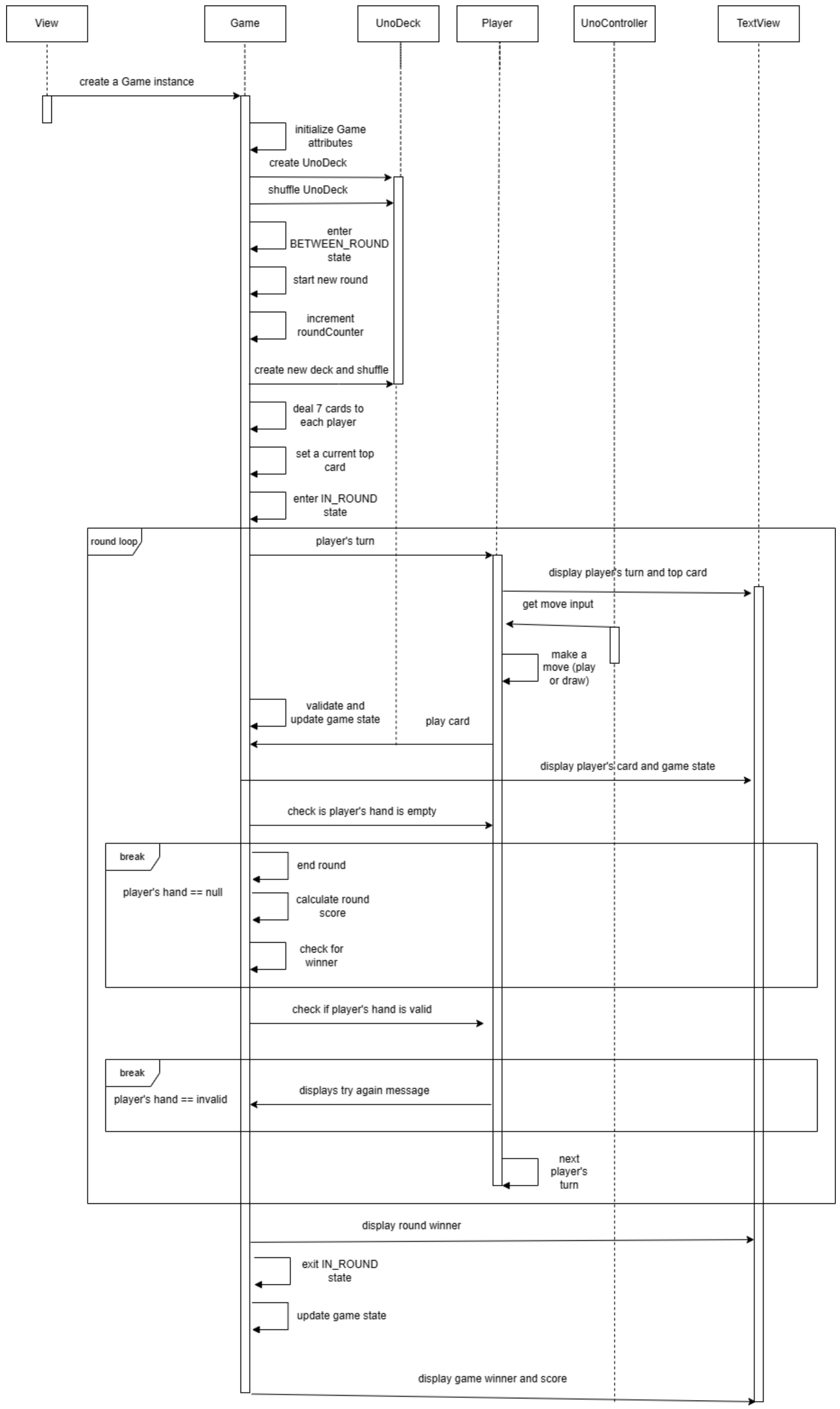

Cards
- Rank
-Colour
-Special
enum: Rank,Colour,Special
+Card(Rank rank, Colour colour, Special specialCard, Type,type
- RankToString(Rank rank)
- SpecialToString(Special special)
+Special getSpecialType()
+Rank getCardNum()
+ Colour getCardColour()
+Special getSpecialType()
void setColour(Card.Colour colour)
UnoDeck
-ArrayList<Card> deck
+ UnoDeck()
+ ArrayList<Card> getDeck()
+ ArrayList<Card> drawNCard(int nCards)
Players
-Str: name
-ArrayList: ArrayList<Card> hand
-int: points
+Player(String inputName, ArrayList<Card> inputHand)
+Player(String inputName)
+String getName()
+ void setName(String inputName)
+playCard(Card card)
+addCard(ArrayList<Card> cardList)
+ void removeCard(Card card)
+ArrayList<Card> getHand()
+int getHandScore()
+ void removeAllCards()
+getPoints()
+addPoints(int roundScore)
Type
enum: light, dark
Rank
enum:ONE,TWO,THREE,FOUR,
FIVE,SIX,SEVEN,EIGHT,NINE
colour
enum: red,yellow,blue
,green,black
Spercial
enum DRAW_ONE,REVERSE,SKIP,FLIP,
WILD,WILD_DRAW_TWO_CARDS
UnoController
UnoController
+TextView()
+void addPlayer(Player player)
+void nextPlayer(Player player, Card topCard)
+void cardHand(ArrayList<Card> playerHand)
+void drawCard(Player player, ArrayList<Card> cardsDrawn)
+void illegalMove(Player player)
+void cardPlayed(Card card, Boolean validCard)
+void roundStart(int roundCount)
+void roundEnd(Player roundWinner)
+void winner(Player winner)
UnoController
-Game
+UnoController(Game game)
+void actionPerformed(ActionEvent e)
+void getPlay(Player player)
+void addPlayers(String name)
View
+void nextPlayer(Player player, Card top
+void cardPlayed(Card card, Boolean va
+void illegalMove(Player player)
+void drawCard(Player player, ArrayList
+void roundEnd(Player roundWinner)
+void roundStart(int roundCounter)
+void winner(Player winner)
+void addPlayer(Player player)
+ Card.Colour getColour();
Game
-ArrayList<Player>
+enum State
-State gameState
-Card topCard
-ArrayList<Card> playedCards
-UnoDeck deck
-Player currPlayer
int playerChoice
int roundCounter
-Player roundWinner
-View view
int currPlayerIndex
+Game(ArrayList<Player> players)
+Game()
+void setView(View view)
+void startGame()
+void addPlayer(Player player)
-void updateGame()
nextTurn()
- void startRound()
+void attemptPlayCard(int cardIndex)
+ void attemptDrawCard()
-void drawOne()
+ void reverse()
- void skip()
- void wildDrawTwo()
- boolean checkWinner()
- boolean legalMove(Card card)
- void playCard(Card card)
-void drawCard(Player player, int amount)
+void iteratePlayers()
GUIView
UnoController controller
Card topCard
String numPlayers
ScrollPane scrollPane
ScrollPane scrollPane
JLabel currentPlayer
JLabel topCardLabel
ImageIcon topCardImage
JButton drawCardButton
JFrame jFrame
GUIView()
private ArrayList<JButton> addButtons(int numButton)
void topCard(Card card)
void addPlayer(Player player)
void nextPlayer(Player player, Card topCard)
void drawCard(Player player, ArrayList<Card> cardsDrawn)
void illegalMove()
void cardPlayed(Card card, Boolean validCard)
void roundStart(int roundCount)
void roundEnd(Player roundWinner)
void winner(Player winner)
Card.Colour getColour()
void main(String[] args)
new GUIView()
State
enum: GAME_START, IN_ROUND,
BETWEEN_ROUND, GAME_END

Figure 1. The Updated UML Diagram UNO FLIP.

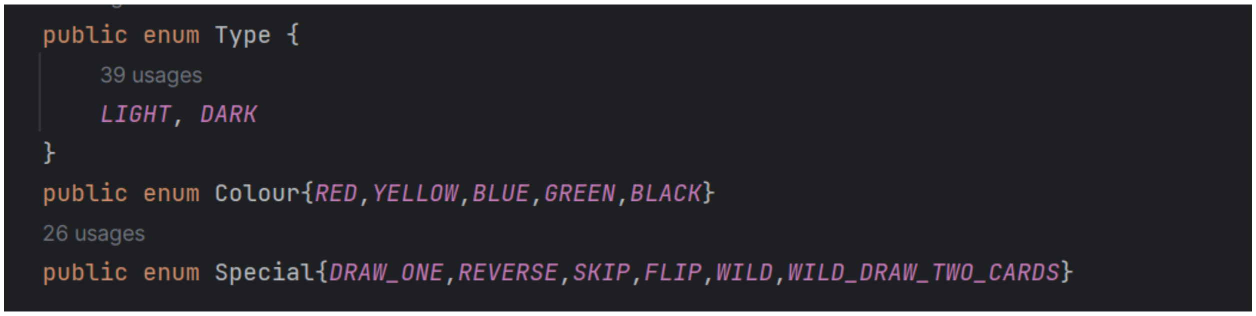
Sequence Diagrams:

Group 24 - Milestone 2



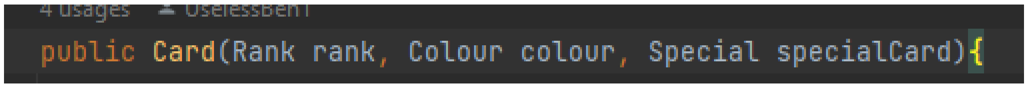
Data structures

Enum: The cards are split into 3 different characteristics(parameters). The 3 characteristics are created by 3 lists of enums. Order in Enum does not matter here and represents the characteristic for each card the best.



```
public enum Type {
    39 usages
    LIGHT, DARK
}
public enum Colour{RED,YELLOW,BLUE,GREEN,BLACK}
26 usages
public enum Special{DRAW_ONE,REVERSE,SKIP,FLIP,WILD,WILD_DRAW_TWO_CARDS}
```

ArrayList: The collection of cards which is in UnoDeck class. The arraylist contains the collection of 112 Cards.



```
4 usages   UselessBen1
public Card(Rank rank, Colour colour, Special specialCard){
```

Detailed Description of the Design

1. Controller now has action listener that gets action command
2. GUIview to have player to interact with the game with Junit buttons. Gui view is now the executable of the game.
3. The controller is the called to get the amount of players and their names, and adds those to the game
4. Controller now interact with the view of the game for player interaction. Controller get information from model.
5. Card class now map image location with each card object.
6. On a player's turn, view.nextPlayer is called, which shows their hand, then calls the controller to get input.
7. That input is then passed back to the game model via game.setPlayerChoice()
8. The model then checks if this input is valid. If it is, it displays it and allows it to pass through. If it is not, it calls view.illegalInput(), and continues to loop until a valid move is made.
9. Once a player's hand is empty, the state changes to BETWEEN_ROUND, the score is tabulated and displayed via more view stuff, and updateGame() is called once more.

10. This continues until a winner has been declared.

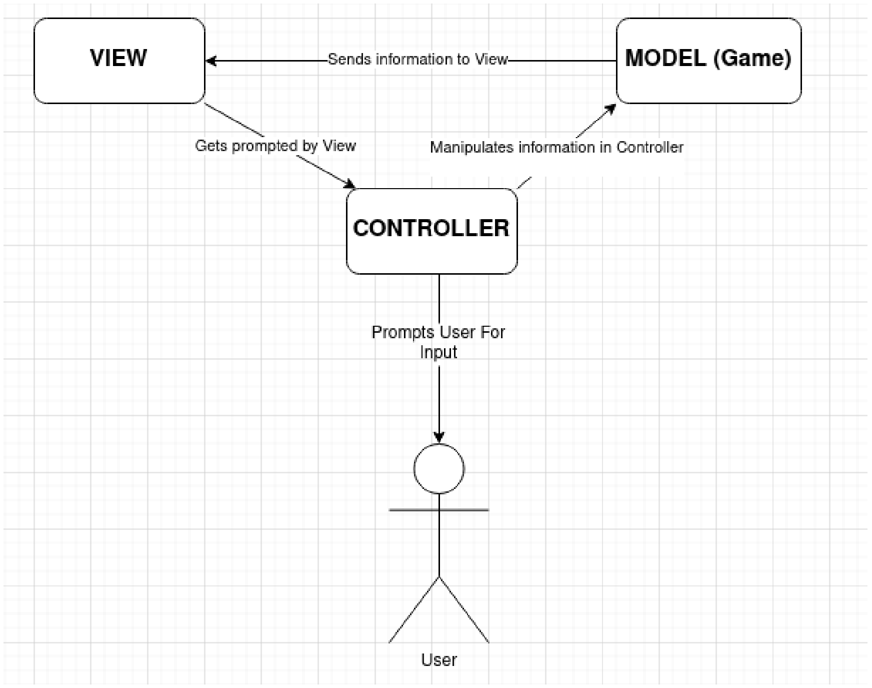
Type of Pattern Used

Our group has decided to follow the model–view–controller (MVC) pattern. The MVC pattern is very useful to this project since
MVC stands for Model View Controller. Each part of the MVC has its unique job and is separate. Jobs in Model have nothing to do with jobs in Version and Controller, and jobs in Version have nothing to do with Model and Controller.

The model : Card class to generate cards with colour, special and rank. UnoDeck class to generate the entire deck of cards and draw N cards. Player class to set player score and cards.

The controller : playRound() method in Game where it asks data from the Uno deck, after each round it is sent to the player's view.

View : GUI view class shows user the graphical view of the game and allow user to sselect how many players by clicking. The TextView class also has methods to show Card in hand, card has been drawn, if the move is legal, card that has been played, when round start and when round finished. Finally the score of the winner.



Test cases

CardTest.java
GameTest.java
PlayerTest.java
UnoControllerTest.java
UnoDeck.java

Authors

Ben Li (101113284)
Joshua Noronha (101194076)
Nicolaus Derikx (101150157)
Sahil Agrawal (101132393)
Divya Vithiyatharan (101196047)