

Deliverable 3

SYST17796 - Software Development

Guided under - Professor Amandeep Sindhu

Date of Submission: Thursday,April 16, 2020

Group Members: Steven ( Group Leader), Jasmine

Daniel and Zawad

Content

Cover page 1

Content page 2

Project Background 3

Initial UML Class Diagram (Deliverable 2) 7

Updated UML Class Diagram 8

UML Class Diagrams Comparison 9

Use Cases and Mind Mapping Diagram 10

JUnit Test Code 12

Test Result Report 15

Project Background

As mentioned in Deliverable 1 the object of the game is to be the first player to get rid of all the cards in hand wins. Rules are pretty simple, cards have to match the top discard in color or number can be played into the pile or choose to play a wild card.

The sub-objectives of playing game and participating in activities is to learn problem-solving,strategy,trust and calculated risk taking.the rule of the game can have far reaching positive effects when they are applied to real life situations

We have introduced new variables to maintain a control in the game like,

Timer - which will be initialised to 20 seconds whenever a new move is about to be played. If the timer runs out before playing the move, the player will be penalised.

Count - with this the game will count the remaining cards in the deck and every player will have a card count variable. Whichever becomes 0 first the game will end.

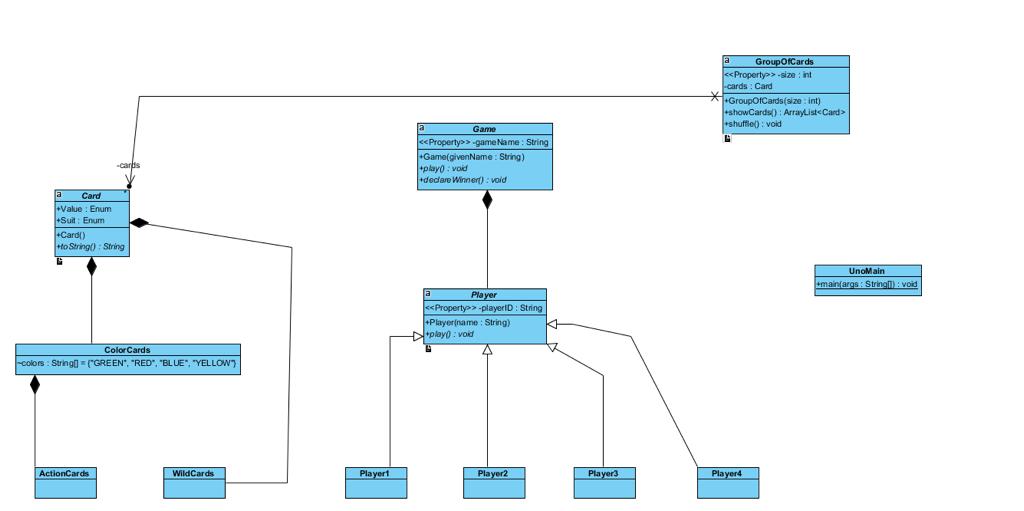
Keeping the OOD principles in mind, we tried to design and write an efficient code for our game. We created classes which are tightly cohesive and loosely coupled so that if a change needed to be done the code will not be affected. We extended from the base code to show the inheritance principle and specific methods are created to perform certain tasks to display delegation across our code.

In this document, we have added the UML Use cases(the list of task our game can do), Code Mine diagram(how the flow function of our game) and UML Class diagram of our updated code and showing the relationship among the classes.

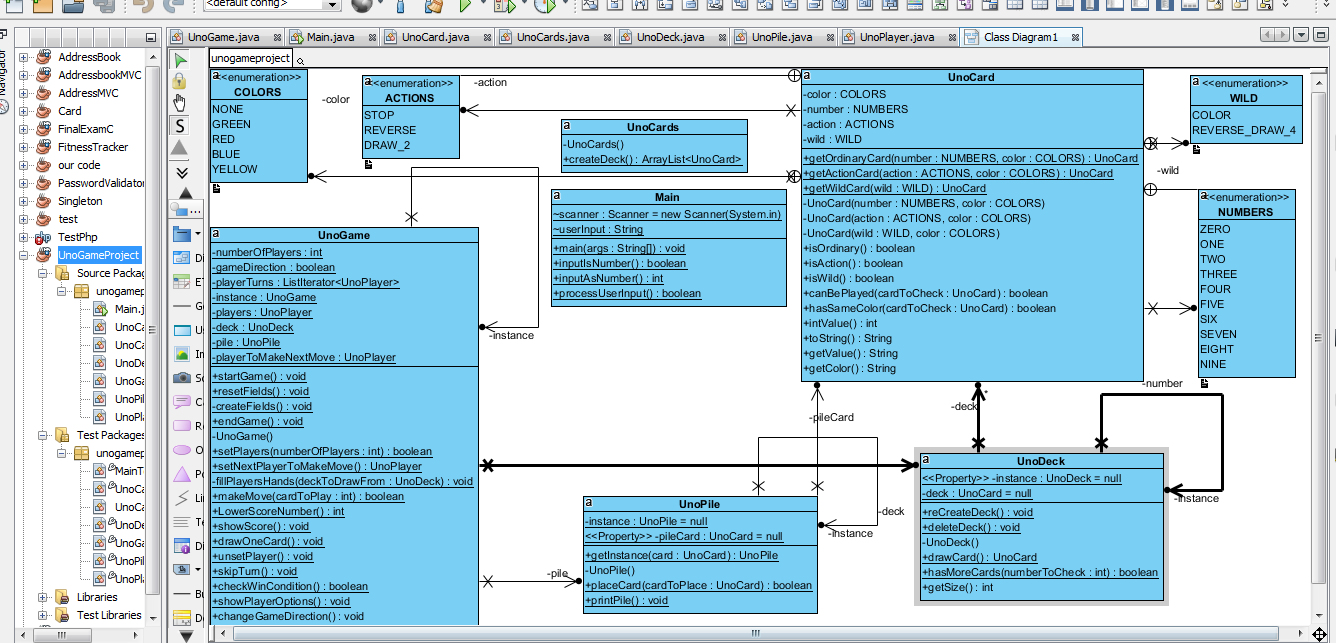
Our repository link :

https://github.com/skebila/SoftDev-Deliverable/tree/master/Deliverable%203

Initial UML Class Diagram( Deliverable2)



Updated UML Class Diagram



UML Class Diagram Comparison

In terms of OOP, OOD, Design pattern

In our initial UML class diagram, there were some errors in creating the our classes hence resulting in a poorly designed UML class. The classes had wrong relationships among them and there were some redundant classes which were not needed to be created. Initially our design, we thought there will be an abstract Card class extending from it will be our UnoCard class(which will be the parent to sub classes of WildCards and Action Cards showing inheritance) relation. Similarly, we created a player class and extended four subclasses of Player(instead of creating instances of Player class we extended it which was incorrect). Clearly the diagram wrongly shows it has no relation with UnoCard class.

We have revised our UML class diagram from our previous submission. In our updated UML diagram it clearly shows the clear meaningful relations between our classes.

Starting from the UnoCard class, instead of creating subclasses like in the previous approach, we used ENUMERATION type for different Uno Cards naming them Ordinary Cards, Action Cards and Wild cards, avoiding going through the hassle of doing string comparison input mismatch.

There are “composition” relationships between UnoCard class and other classes except for UnoCards(which just creates a deck) as without an Uno card the components (game, player, deck or the pile would not exist). Similarly UnoGame class will go out of existence. The UnoPlayer class is dependent on the UnoPile class and UnoDeck depends on the UnoCards class and finally UnoCards is dependent on the UnoCard class..

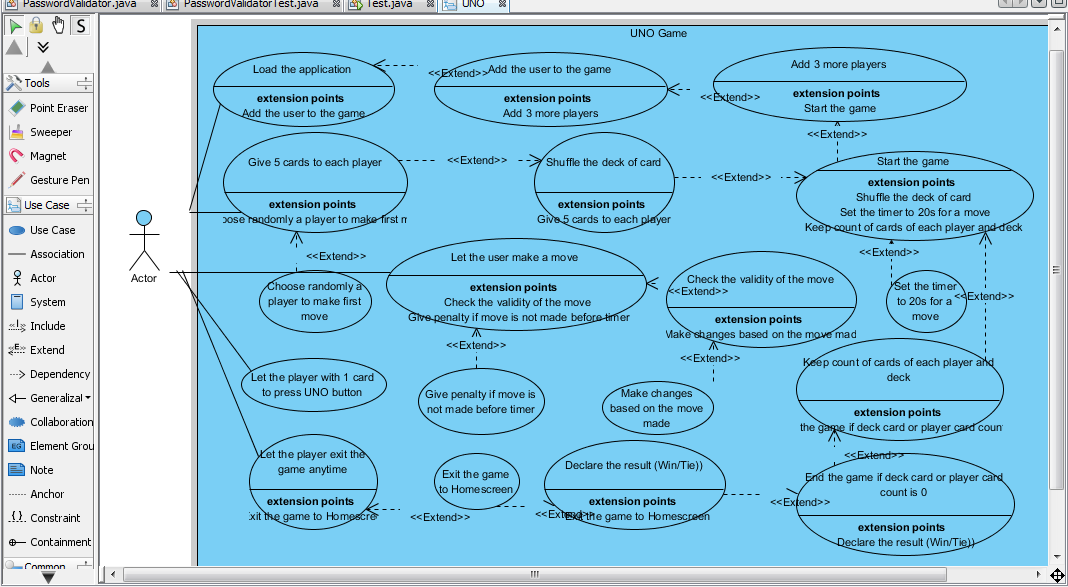
We have implemented Singleton Pattern to create only one instance of the UnoGame, as simultaneously two games can not be played. We also have factory classes for objects instantiation to have a control in a more centralised manner.

By following the MVC pattern our code is quite loosely coupled but with high cohesion.

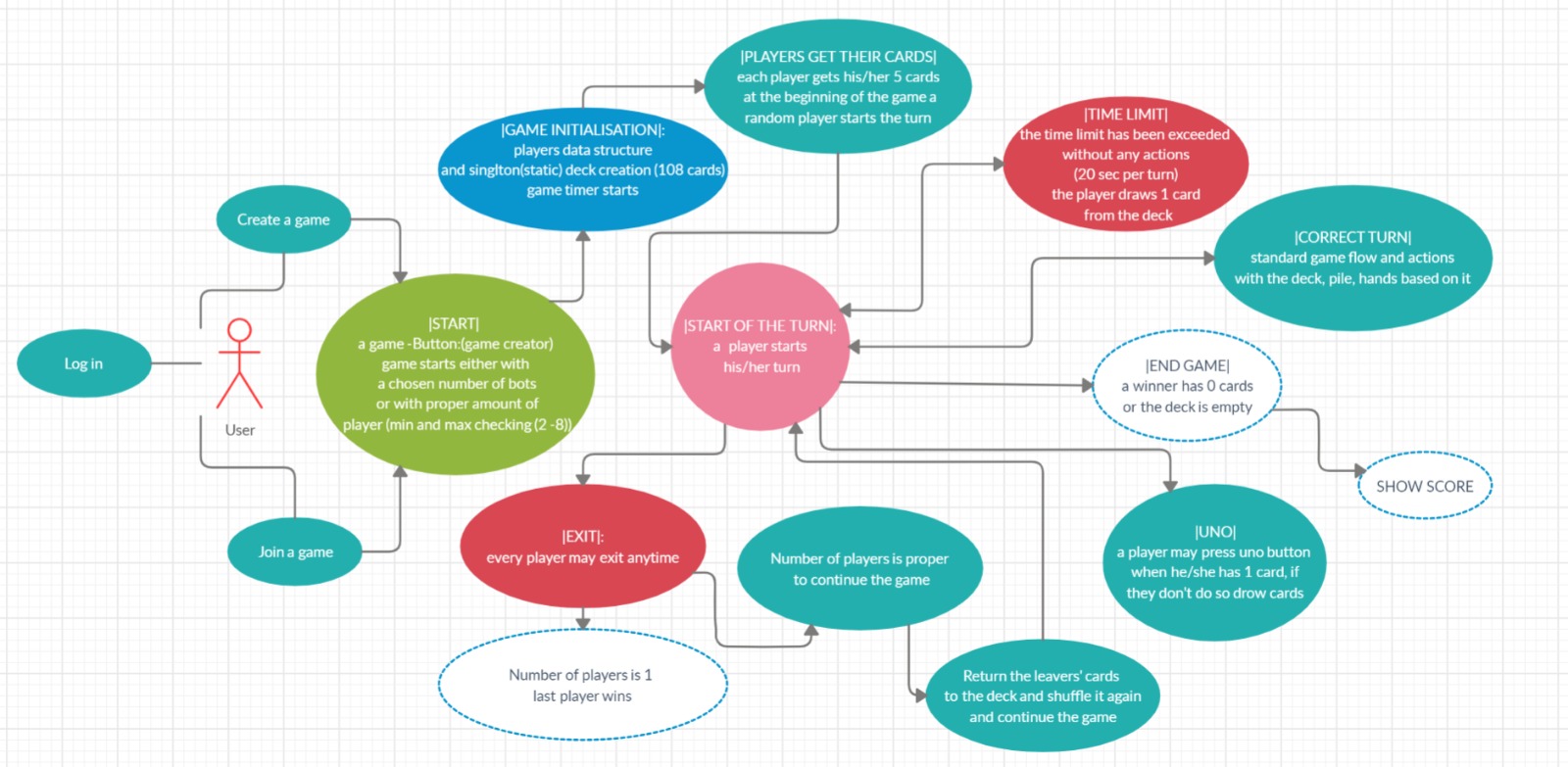
Repo link to updated UML Class Diagram:

<https://github.com/skebila/SoftDev-Deliverable/tree/master/Deliverable%203/Images>

UML USE CASE



Code Mind Mapping Diagram



List the features or Use Cases our code does which will be tested by the JUnit testing

1.

2.

JUnit Test codes and Scripts

Link to the repository :

https://github.com/skebila/SoftDev-Deliverable/tree/master/Deliverable%203/Code/JUnit%20Test%20code%20and%20script

//

Copy paste the codes

//

**Test Result Report**

The JUnit Test were carried out and tabulated in the below table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test# | Requirement | Use case | Test method (Class name : method name) | Type | Status date |
| 1 | User enters q to quit or enters nothing to skip turn | Quit game/Skip turn | MainTest.testProcessUserInput | Good | Pass (17th April 2019) |
| 2 | User enters q to quit or enters nothing to skip turn | Quit game/Skip turn | MainTest.testProcessUserInput | Bad | Pass (17th April 2019) |
| 3 | User enters q to quit or enters nothing to skip turn | Quit game/Skip turn | MainTest.testProcessUserInput | Boundary | Pass (17th April 2019) |
| 4 | User has to enter number for players | Checks number entry format | MainTest.testInputAsNumber | Good | Pass (17th April 2019) |
| 5 | User has to enter number for players | Checks number entry format | MainTest.testInputAsNumber | Bad | Pass (17th April 2019) |
| 6 | User has to enter number for players | Checks number entry format | MainTest.testInputAsNumber | Boundary | Pass (17th April 2019) |
| 7 | User has to enter number for players | Checks integer format entry | MainTest.testInputIsNumber | Good | Pass (17th April 2019) |
| 8 | User has to enter number for players | Checks integer format entry | MainTest.testInputIsNumber | Bad | Pass (17th April 2019) |
| 9 | User has to enter number for players | Checks integer format entry | MainTest.testInputIsNumber | Boundary | Pass (17th April 2019) |
| 10 | User has to enter number for players between 2 -10 | Checks the number of player | UnoPlayerTest.testGetPlayerNumber | Good | Pass (17th April 2019) |
| 11 | User has to enter number for players between 2 -10 | Checks the number of player | UnoPlayerTest.testGetPlayerNumber | Bad | Fail (17th April 2019) |
| 12 | User has to enter number for players between 2 -10 | Checks the number of player | UnoPlayerTest.testGetPlayerNumber | Boundary | Pass (17th April 2019) |
| 13 | User enters the game gets an empty card pile | Creates and empty hand | UnoPlayerTest.testGetHand |  | Pass (17th April 2019) |
| 14 | User hand size of UnoCards | Checks for number of card in hand | UnoPlayerTest.testHasCardsGood | Good | Pass (17th April 2019) |
| 15 | User hand size of UnoCards more than 0 | Checks for number of card in hand | UnoPlayerTest.testHasCardsBoundary | Boundary | Pass (17th April 2019) |
| 16 | User hand size of UnoCards exactly 0 | Checks for hand empty | UnoPlayerTest.testHasCardsEmpty | Empty | Pass (17th April 2019) |
| 17 | The system creates the ordinary card | Creates ordinary card and returns it | UnoCardTest.testGetOrdinaryCardNotNull | Not Empty | Pass (17th April 2019) |
| 18 | The system creates the actions card | Creates action card and returns it | UnoCardTest.testGetActionCardNotNull | Not Empty | Pass (17th April 2019) |
| 19 | The system creates the wild card | Creates wild card and returns it | UnoCardTest.testGetWildCardNotNull | Not Empty | Pass (17th April 2019) |
| 20 | The system checks for ordinary card | Checks for Ordinary card | UnoCardTest.testIsOrdinary | TRUE | Pass (17th April 2019) |
| 21 | The system checks for ordinary card | Checks for Ordinary card | UnoCardTest.testIsNotOrdinary | FALSE | Pass (17th April 2019) |
| 22 | The system checks for action card | Checks for Action card | UnoCardTest.testIsAction | TRUE | Pass (17th April 2019) |
| 23 | The system checks for action card | Checks for Action card | UnoCardTest.testIsNotAction | FALSE | Pass (17th April 2019) |
| 24 | The system checks for wild card | Checks for Wild card | UnoCardTest.testIsWild | TRUE | Pass (17th April 2019) |
| 25 | The system checks for wild card | Checks for Wild card | UnoCardTest.testIsNotWild | FALSE | Pass (17th April 2019) |
| 26 | The System checks whether card can be played | Checks for validity | UnoCardTest.testCanBePlayed | TRUE | Pass (17th April 2019) |
| 27 | The System checks whether card can be played | Checks for validity | UnoCardTest.testCanNotBePlayed | FALSE | Fail (17th April 2019) |
| 28 | The System retrieves the color of a card | Gets the Color of the card | UnoCardTest.testGetColor | Good | Pass (17th April 2019) |
| 29 | The System retrieves the error message | Gets the no color | UnoCardTest.testGetColorError | Error | Fail (17th April 2019) |
| 30 | The system checks the same color card | Checks the color of card | UnoCardTest.testhasSameColor | TRUE | Pass (17th April 2019) |
| 31 | The system checks the same color card | Checks the color of card | UnoCardTest.testhasNotSameColor | FALSE | Pass (17th April 2019) |
| 32 | The system retrieves the number value on the card | Gets the number | UnoCardTest.testIntValueGood | Good | Pass (17th April 2019) |
| 33 | The system retrieves -1 for non ordinary card | Gets the number | UnoCardTest.testIntValueBad | Bad | Fail (17th April 2019) |
| 34 | The System prints out the String representation of card | Gets the representation | UnoCardTest.testToString |  | Pass (17th April 2019) |
| 35 | The System prints out the Value of the card | Gets the value | UnoCardTest.testGetValueGood | Good | Pass (17th April 2019) |
| 36 | The System prints out the Error for non card | Gets the value | UnoCardTest.testGetValueBad | Bad | Fail (17th April 2019) |
| 37 | The System creates a deck of Uno Cards | Creates a deck | UnoCardsTest.testcreateDeck |  | Pass (17th April 2019) |
| 38 | The system creates a deck object | Creates a deck object | UnoDeckTest.testGetInstance |  | Pass (17th April 2019) |
| 39 | The System recreates the deck before the game | creates a new deck | UnoDeckTest.testRecreatedDeck |  | Pass (17th April 2019) |
| 40 | The System deletes a deck at the end of the game | deletes a deck | UnoDeckTest.testDeleteDeck |  | Pass (17th April 2019) |
| 41 | The system checks if player/deck has more cards | checks for more cards | UnoDeckTest.testHasMoreCards |  | Pass (17th April 2019) |
| 42 | The system allows user to remove a card from the deck to give it to player | Removes a card gives to player | UnoDeckTest.testDrawCard |  | Pass (17th April 2019) |
| 43 | The system checks whether deck is full | checks for a full deck | UnoDeckTest.testGetSize |  | Pass (17th April 2019) |
| 44 | The System creates a pile of unocards been played | creates UnoPile object | UnoPileTest.testGetInstance |  | Pass (17th April 2019) |
| 45 | The system checks card user played with the top of deck and allows to play if true | checks and allows player to play card | UnoPileTest.testPlaceCardGood | Good | Pass (17th April 2019) |
| 46 | The system checks card user played with the top of deck and prevents to play if false | checks and don’t allow to play card | UnoPileTest.testPlaceCardBad | Bad | Pass (17th April 2019) |
| 47 | The system prints the pile with the top card | prints pile | UnoPileTest.testPrintPileGood | Good | Pass (17th April 2019) |
| 48 | The sytem checks if pile empty, prints error message | prints message | UnoPileTest.testPrintPileBad | Bad | Pass (17th April 2019) |
| 49 | If the deck is not empty, system allows to draw card from deck | Draws a card from deck | UnoPlayerTest.testDrawCardsGood | Good | Pass (17th April 2019) |
| 50 | If the deck is not empty, system allows to draw card from deck | Draws all 108 cards from deck | UnoPlayerTest.testDrawCardsBoundary | Boundary | Pass (17th April 2019) |
| 51 | If the deck is empty, system do not allow to draw card | Cant draw from empty | UnoPlayerTest.testDrawCardsBad | Bad | Pass (17th April 2019) |
| 52 | If player/user hand is empty, system prints message | Prints empty hand | UnoPlayerTest.testPaintPlayerHandEmpty |  | Pass (17th April 2019) |
| 53 | If player/user hand not empty, prints the entire hand | Prints the player hand | UnoPlayerTest.testPaintPlayerHandFull |  | Pass (17th April 2019) |
| 54 | System checks the card choice, lets the player play | Lets play the card | UnoPlayerTest.testPlayCardGood | Good | Pass (17th April 2019) |
| 55 | System checks for an empty hand then a player can not play | Stops player to play | UnoPlayerTest.testPlayCardGood | Bad | Pass (17th April 2019) |
| 56 | System creates the game and resets fields | Starts the game | UnoGameTest.testStartGame |  | Pass (17th April 2019) |
| 57 | System checks if there is an instance of UnoGame, it will return that object | Creates the object of UnoGame | UnoGameTest.testGetInstance |  | Pass (17th April 2019) |
| 58 | System resets the fields sets the object to null | Resets the game | UnoGameTest.testResetFields |  | Pass (17th April 2019) |
| 59 | System creates the game and sets field at the start of the game with 108 cards on deck | Creates the game | UnoGameTest.testCreateFields |  | Pass (17th April 2019) |
| 60 | System ends the game and resets the fields | Ends the game | UnoGameTest.testEndGame |  | Pass (17th April 2019) |
| 61 | Checks the # of player if within range sets the players | Sets the players | UnoGameTest.testSetPlayersGood | Good | Pass (17th April 2019) |
| 62 | Checks the # of player if within range sets the players | Sets the players | UnoGameTest.testSetPlayersBoundary | Boundary | Pass (17th April 2019) |
| 63 | Checks the # of player for the game if less than 0 or more than 10 it sends a error message | Sends error message, for wrong number of players | UnoGameTest.testSetPlayersBad | Bad | Pass (17th April 2019) |
| 64 | Checks for the direction and returns the UnoPlayer whose turn is next | Finds out whose turn is next | UnoGameTest.testSetNextPlayerToMakeMove |  | Pass (17th April 2019) |
| 65 | System starts the game, creates the deck, fills player hands | Fills player hand with 7 cards | UnoGameTest.testFillsPlayerHand |  | Pass (17th April 2019) |

Test Manual Script

In the UnoGame class, there were few methods which we could not

run test on these are :

1) Public static Boolean makeMove (int cardPlay)

This method takes in a parameter from the player, which card to play. The system checks whether it is a valid choice or not then allows the player to play the card and it adds to the pile of cards

2) Public static Boolean drawOneCard()

This method do not take any parameter, it just calls the drawCards method of the UnoPlayer class and the makes the decision upon checking whether it is viable action or not like if the deck is empty it will not allow the player to draw the card and ends the game.

3) Public static void unsetPlayer()

This method also do not take any parameter, it is called when the player quits. The game then removes the player from the list

4) Public static void skipTurn()

This method is called when a player plays the Stop action card. It skips the next player to the next.

5) Public static Boolean checkWinCondition()

This method checks the three conditions, whether the deck is empty then stops the game or a player has run out of cards then declare the player as the winner.

6) Public showPlayerOptions()

This method prints out the cards in hand and on the pile using the printPile method so that user can understand which card to play next.

7) Public static Boolean changeGameDirection()

This method is called when a player plays a Reverse action card, which changes the direction from anticlockwise to clockwise.