OOD Team Project Final Report

Project Members:
Torin Stott
Ezekiel Maynard (**Team Leader**)
Carson Downs
Stephen Barnhart
Caleb Pace

Project Objective:
Create a program which contains and can run four different games, each utilizing a different design pattern.

Paradigm Unified Process:

Inception:

- Decide on a concept (game collection)
- Ensure concept is possible in the given time
- Exchange contact info for group members

Elaboration:

- Decide on patterns
- Decide on games to create, but leave room for change

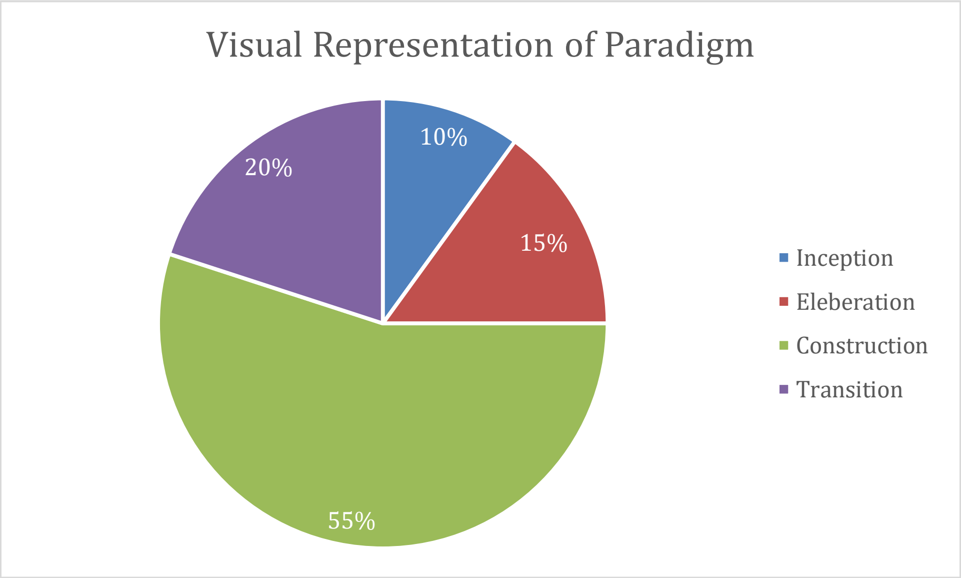
Construction:

- Develop games
- Integrate patterns in games
- Prototyping games
- Revision and creation of central code repository.

Transition:

- Test GUI ran games and other games as well
- Finalize Documentation and Submit

Paradigm Time Representation:

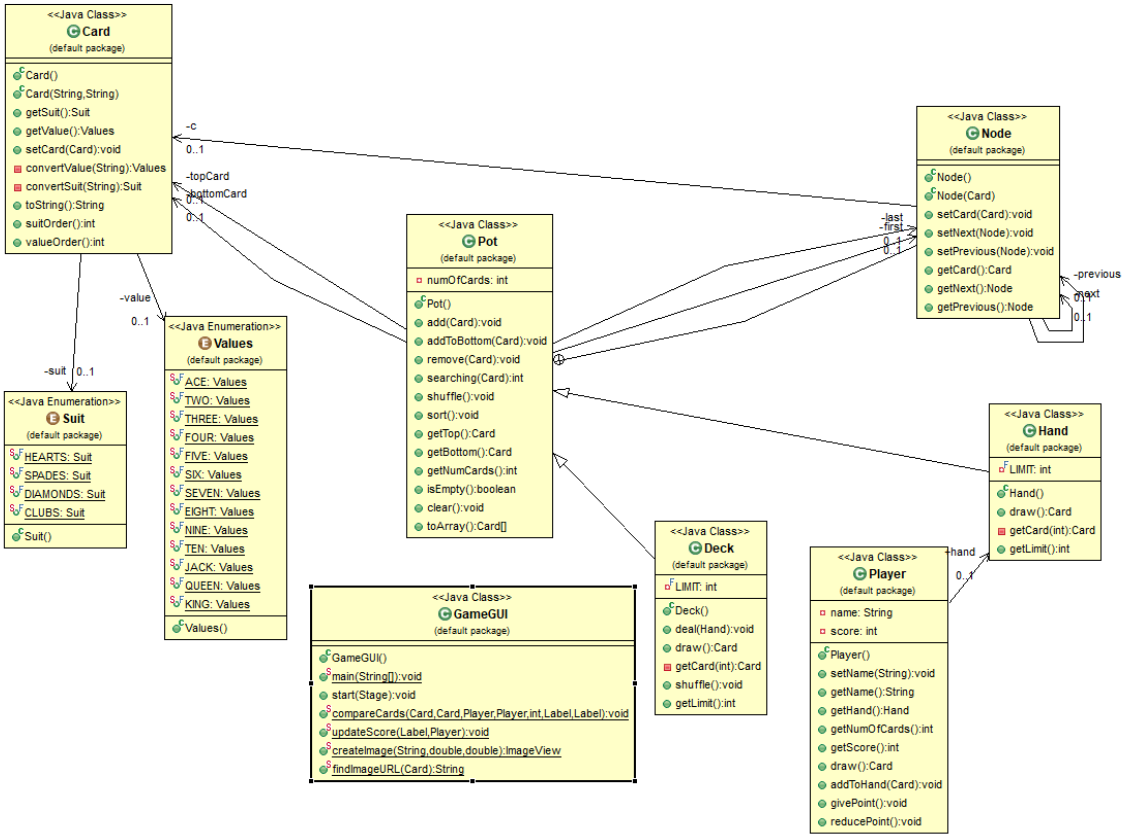


Design Patterns Usage:

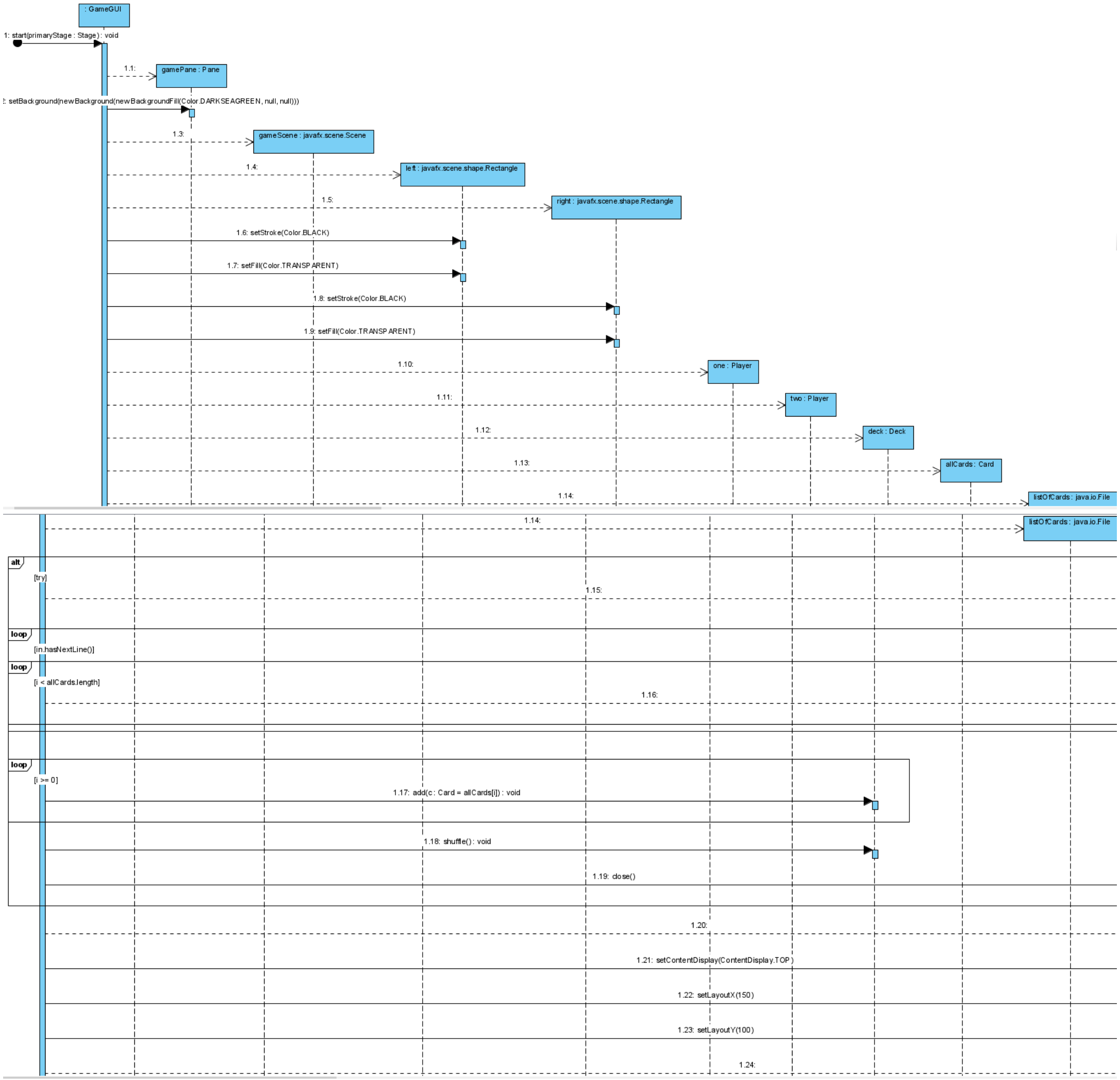
- **MVC** - Used to create a JavaFX based War card game. The MVC pattern is used in the interaction between the user and the JavaFX buttons in which the models are updated. Controller is the JavaFX buttons and the model is the GUI elements (ex: cards) that appear when those buttons are pushed.
- **State** - Used to create a hangman game. States transition based on the inputs and user actions until the win or lose state has been achieved.
- **Command** - Used to create a console-based tic-tac-toe game with undo functionality. Commands are executed for XCommand and OCommand. Undo function stores any previous command made. Actions are made until win, loss, or draw is achieved.
- **Iterator** - Used to create and display the game board. Updates board as game is played.
- **Decorator** - Used to organize everyone’s game to keep uniformity in the method they are called and run within the Central Repository.
- **Proxy** - Used to create a proxy that played the game of pong that the users can play using the W/S keys and UP/Down Arrow keys to operate the paddles.

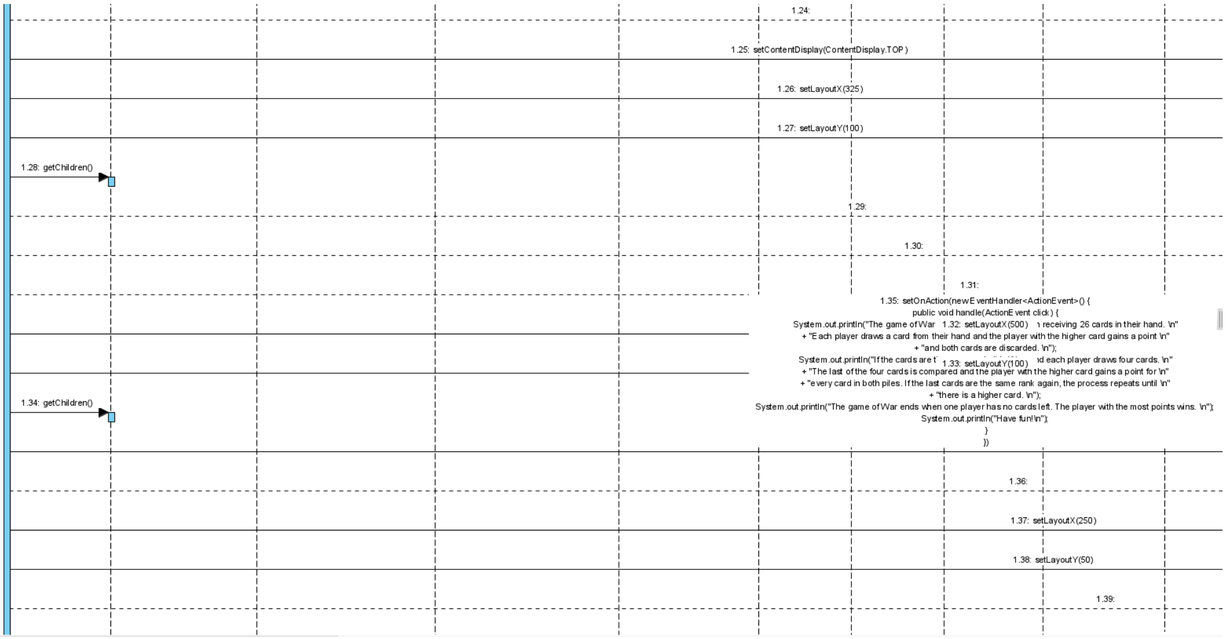
Pattern Diagrams:

1. MVC
   - UML:

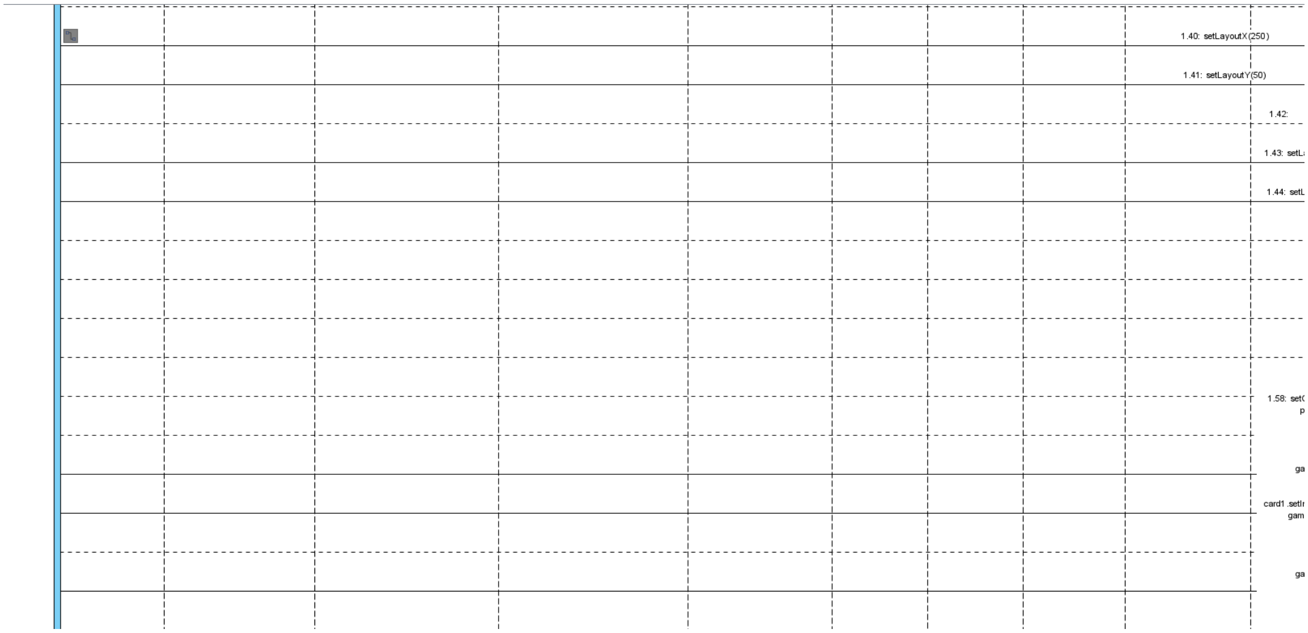


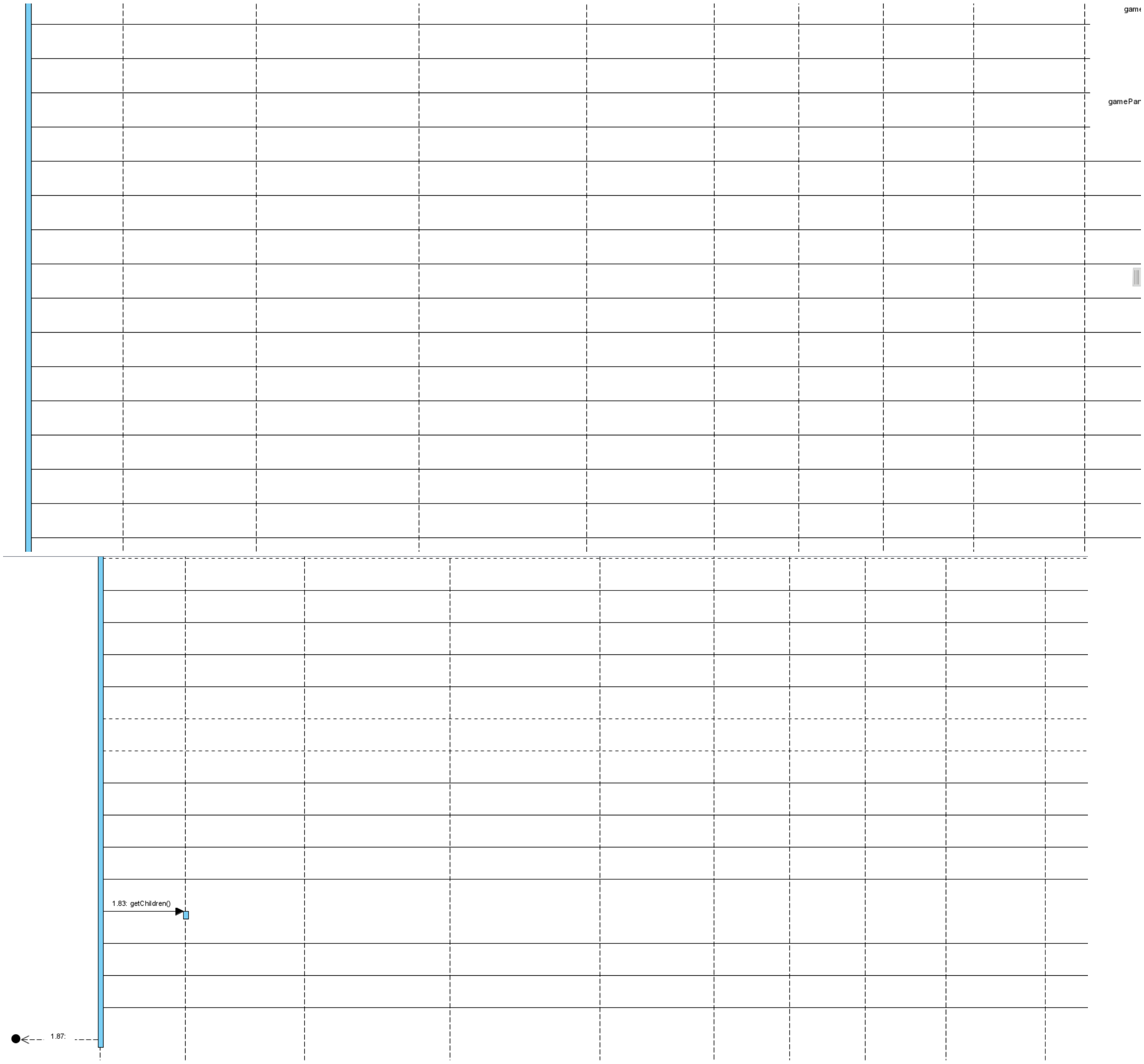
   - Sequence(Broken into parts):


: GameGUI
1: start(primaryStage : Stage) : void
1.1:
gamePane : Pane
2: setBackground(newBackground(newBackgroundFill(Color.DARKSEAGREEN, null, null)))
1.3:
gameScene : javafx.scene.Scene
1.4:
left : javafx.scene.shape.Rectangle
1.5:
right : javafx.scene.shape.Rectangle
1.6: setStroke(Color.BLACK)
1.7: setFill(Color.TRANSPARENT)
1.8: setStroke(Color.BLACK)
1.9: setFill(Color.TRANSPARENT)
1.10:
one : Player
1.11:
two : Player
1.12:
deck : Deck
1.13:
allCards : Card
1.14:
listOfCards : java.io.File
1.14:
listOfCards : java.io.File
alt
[try]
1.15:
loop
[in.hasNextLine()]
loop
[i < allCards.length]
1.16:
loop
[i >= 0]
1.17: add(c : Card = allCards[i]) : void
1.18: shuffle() : void
1.19: close()
1.20:
1.21: setContentDisplay(ContentDisplay.TOP)
1.22: setLayoutX(150)
1.23: setLayoutY(100)
1.24:

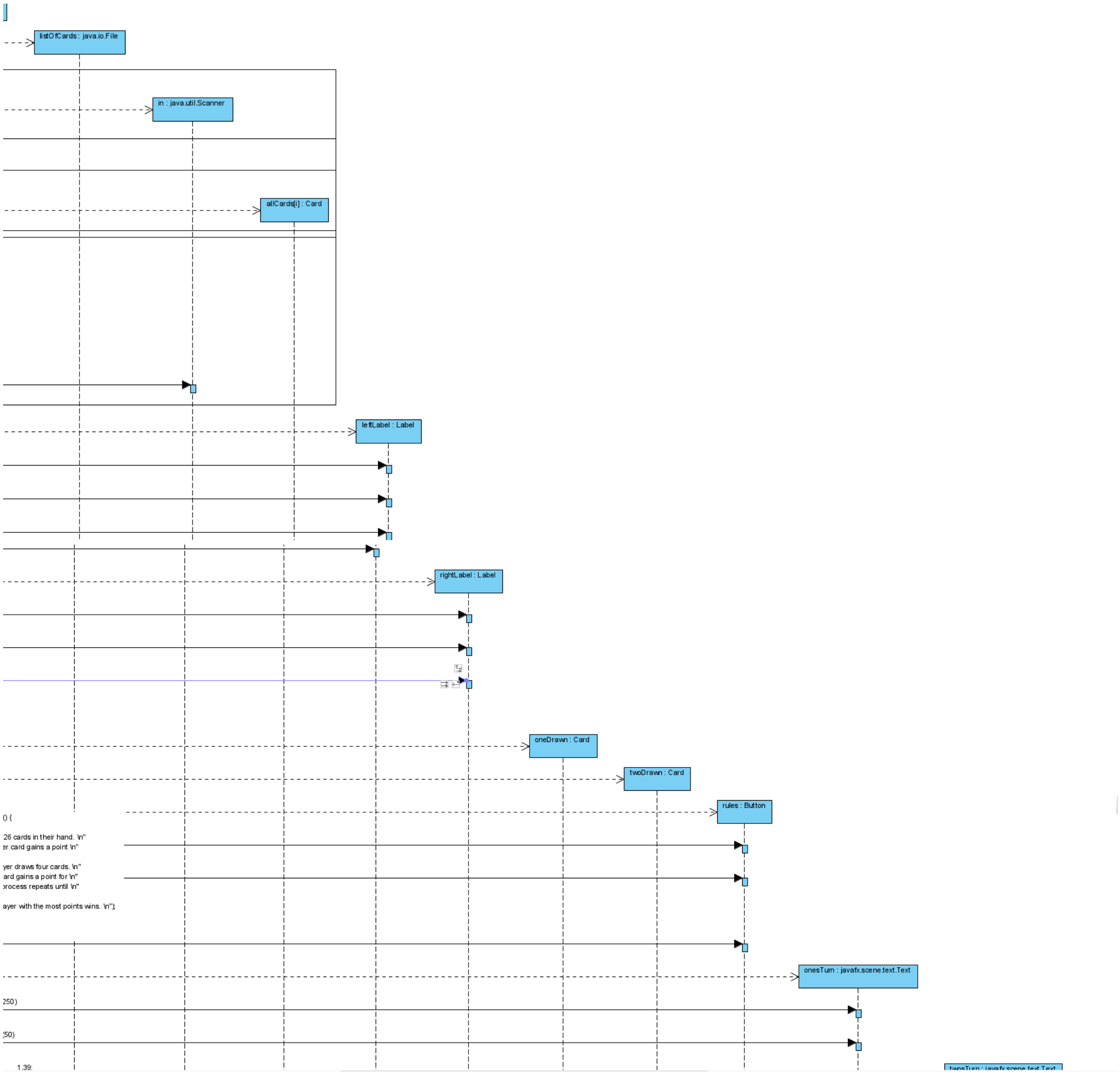


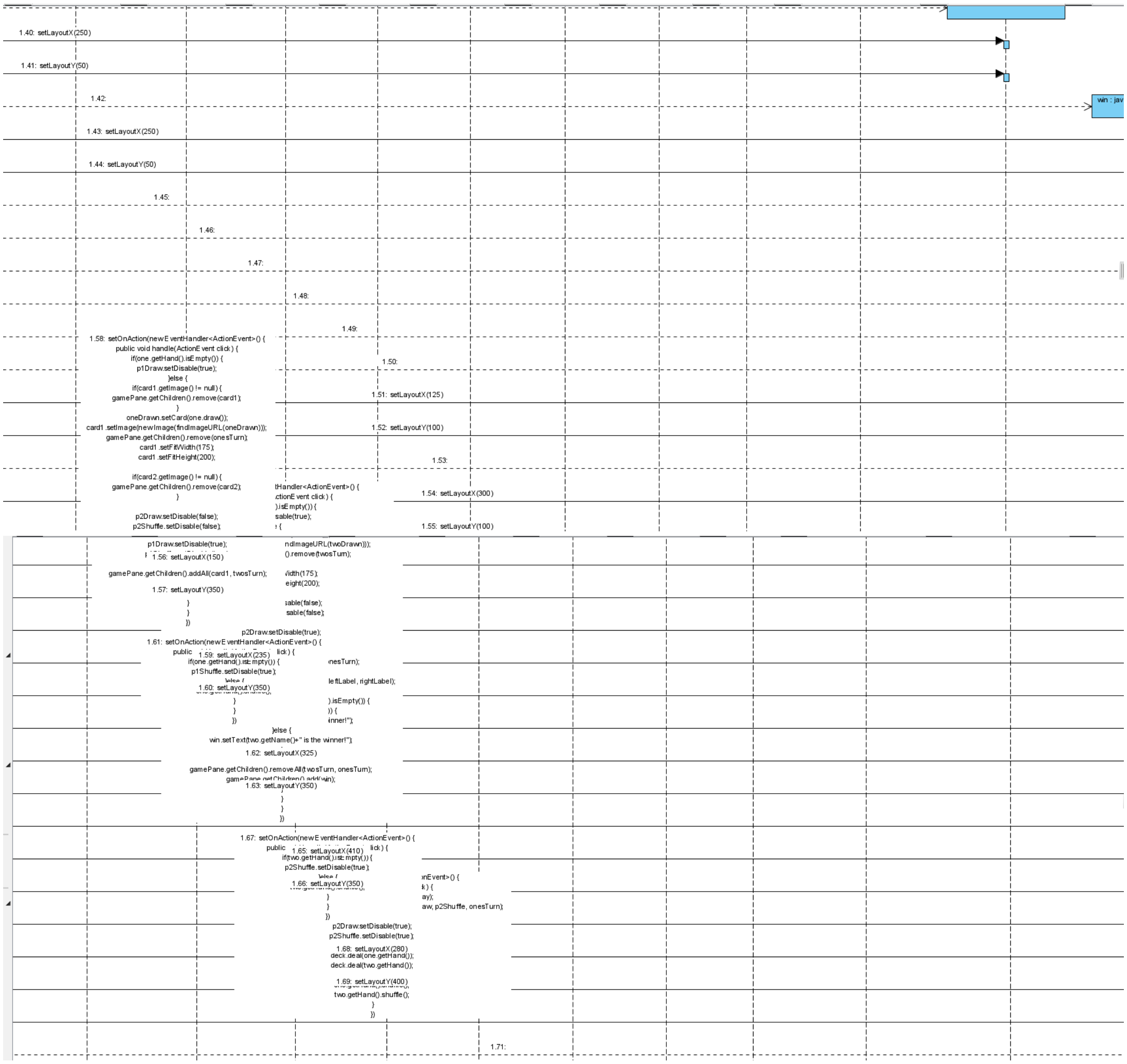
```
1.35: setOnAction(new EventHandler<ActionEvent>() {
public void handle(ActionEvent click) {
System.out.println("The game of War    1.32: setLayoutX(500)    n receiving 26 cards in their hand. \n"
+ "Each player draws a card from their hand and the player with the higher card gains a point \n"
+ "and both cards are discarded. \n");
System.out.println("If the cards are t    1.33: setLayoutY(100)    nd each player draws four cards. \n"
+ "The last of the four cards is compared and the player with the higher card gains a point for \n"
+ "every card in both piles. If the last cards are the same rank again, the process repeats until \n"
+ "there is a higher card. \n");
System.out.println("The game of War ends when one player has no cards left. The player with the most points wins. \n");
System.out.println("Have fun!\n");
}
})
```

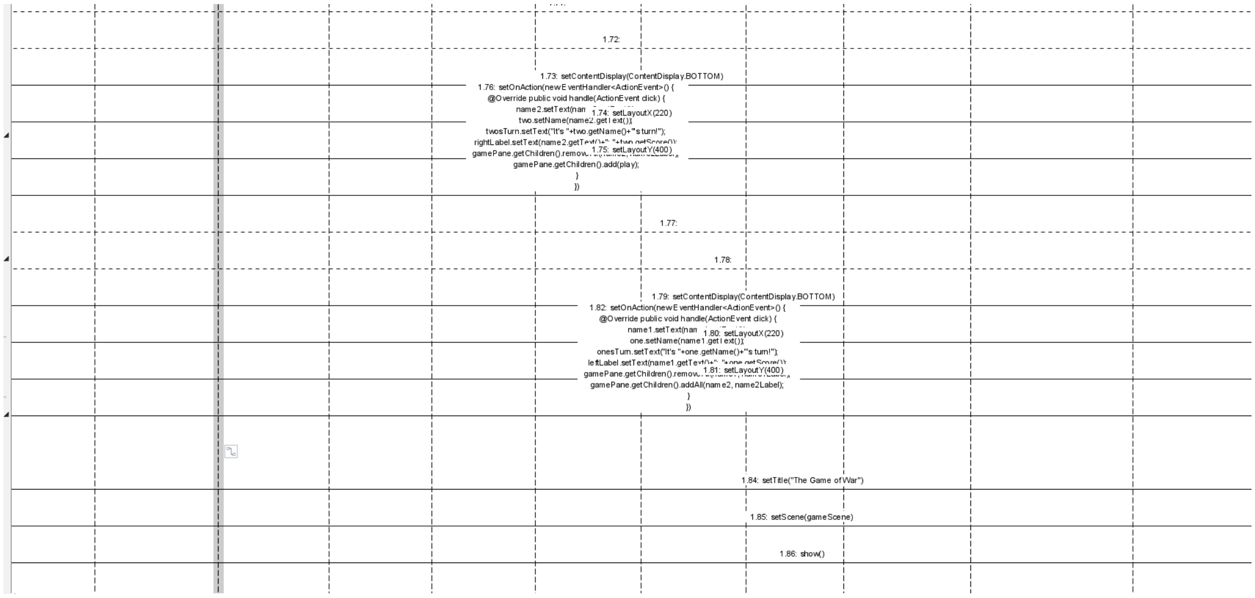




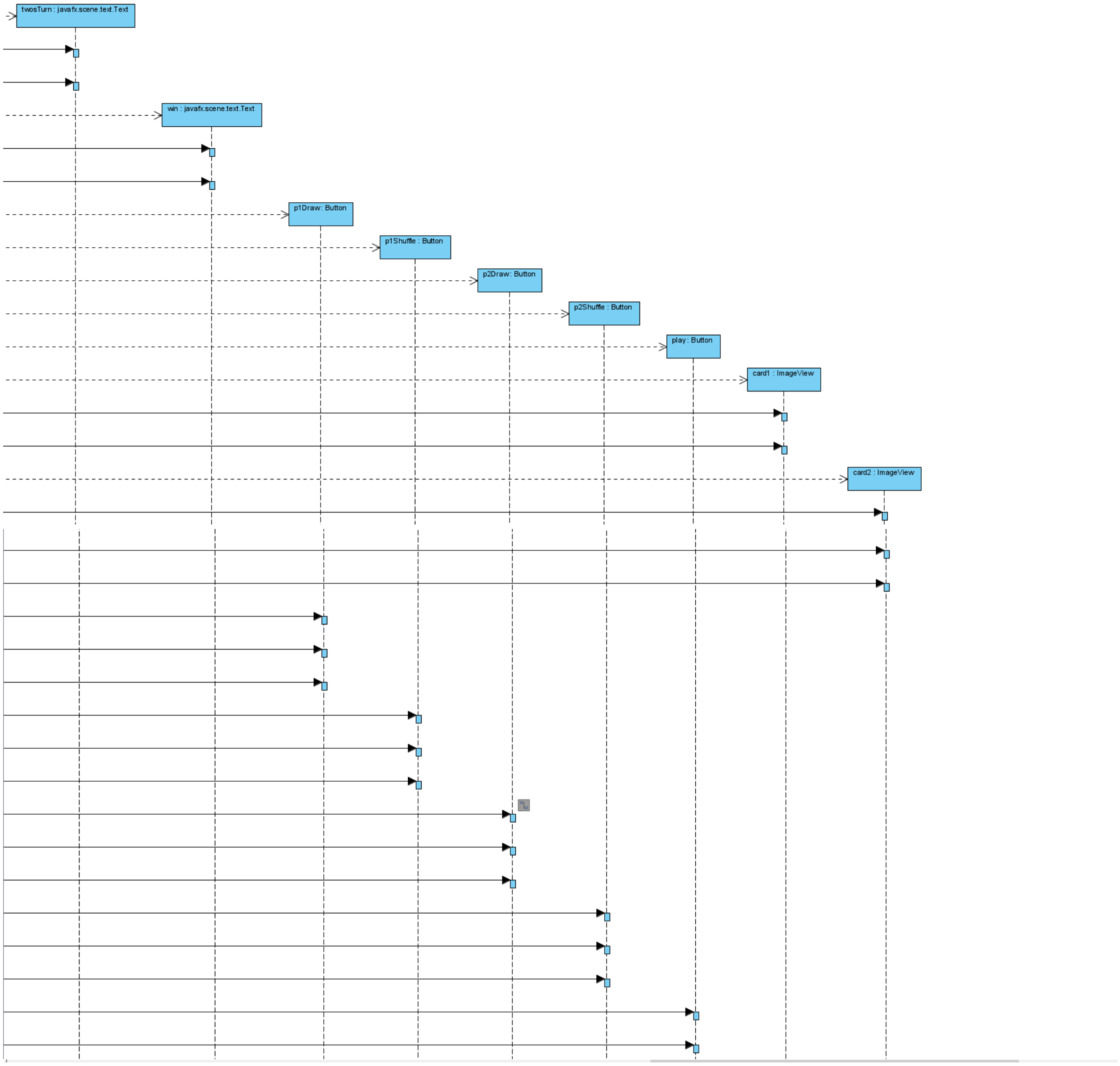
New Row:

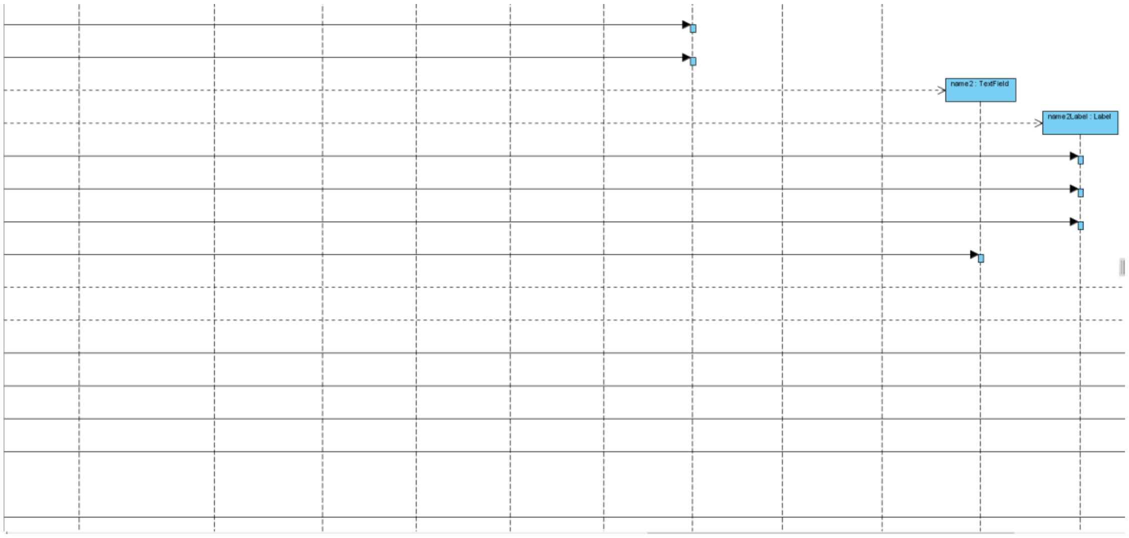

listOfCards : java.io.File
in : java.util.Scanner
allCards[i] : Card
leftLabel : Label
rightLabel : Label
oneDrawn : Card
twoDrawn : Card
rules : Button
() {
26 cards in their hand. \n"
er card gains a point \n"
yer draws four cards. \n"
ard gains a point for \n"
process repeats until \n"
ayer with the most points wins. \n");
onesTurn : javafx.scene.text.Text
250)
50)
1.39:
twosTurn : javafx.scene.text.Text


1.40: setLayoutX(250)
1.41: setLayoutY(50)
1.42:
win : jav
1.43: setLayoutX(250)
1.44: setLayoutY(50)
1.45:
1.46:
1.47:
1.48:
1.49:
1.58: setOnAction(new EventHandler<ActionEvent>() {
public void handle(ActionEvent click) {
if(one.getHand().isEmpty()) {
p1Draw.setDisable(true);
}else {
if(card1.getImage() != null) {
gamePane.getChildren().remove(card1);
}
oneDrawn.setCard(one.draw());
card1.setImage(new Image(findImageURL(oneDrawn)));
gamePane.getChildren().remove(onesTurn);
card1.setFitWidth(175);
card1.setFitHeight(200);
if(card2.getImage() != null) {
gamePane.getChildren().remove(card2);
}
p2Draw.setDisable(false);
p2Shuffle.setDisable(false);
p1Draw.setDisable(true);
1.56: setLayoutX(150)
gamePane.getChildren().addAll(card1, twosTurn);
1.57: setLayoutY(350)
}
}
})
1.50:
1.51: setLayoutX(125)
1.52: setLayoutY(100)
1.53:
tHandler<ActionEvent>() {
ctionEvent click) {
).isEmpty()) {
sable(true);
e {
ndImageURL(twoDrawn)));
().remove(twosTurn);
Vidth(175);
eight(200);
sable(false);
sable(false);
1.54: setLayoutX(300)
1.55: setLayoutY(100)
p2Draw.setDisable(true);
1.61: setOnAction(new EventHandler<ActionEvent>() {
public lick) {
1.59: setLayoutX(235)
if(one.getHand().isEmpty()) {
p1Shuffle.setDisable(true);
}else {
1.60: setLayoutY(350)
}
}
})
nesTurn);
leftLabel, rightLabel);
).isEmpty()) {
)) {
inner!");
}else {
win.setText(two.getName()+" is the winner!");
1.62: setLayoutX(325)
gamePane.getChildren().removeAll(twosTurn, onesTurn);
gamePane.getChildren().add(win);
1.63: setLayoutY(350)
}
}
})
1.67: setOnAction(new EventHandler<ActionEvent>() {
public lick) {
1.65: setLayoutX(410)
if(two.getHand().isEmpty()) {
p2Shuffle.setDisable(true);
}else {
1.66: setLayoutY(350)
}
}
})
onEvent>() {
k) {
ay);
aw, p2Shuffle, onesTurn);
p2Draw.setDisable(true);
p2Shuffle.setDisable(true);
1.68: setLayoutX(280)
deck.deal(one.getHand());
deck.deal(two.getHand());
1.69: setLayoutY(400)
two.getHand().shuffle();
}
})
1.71:

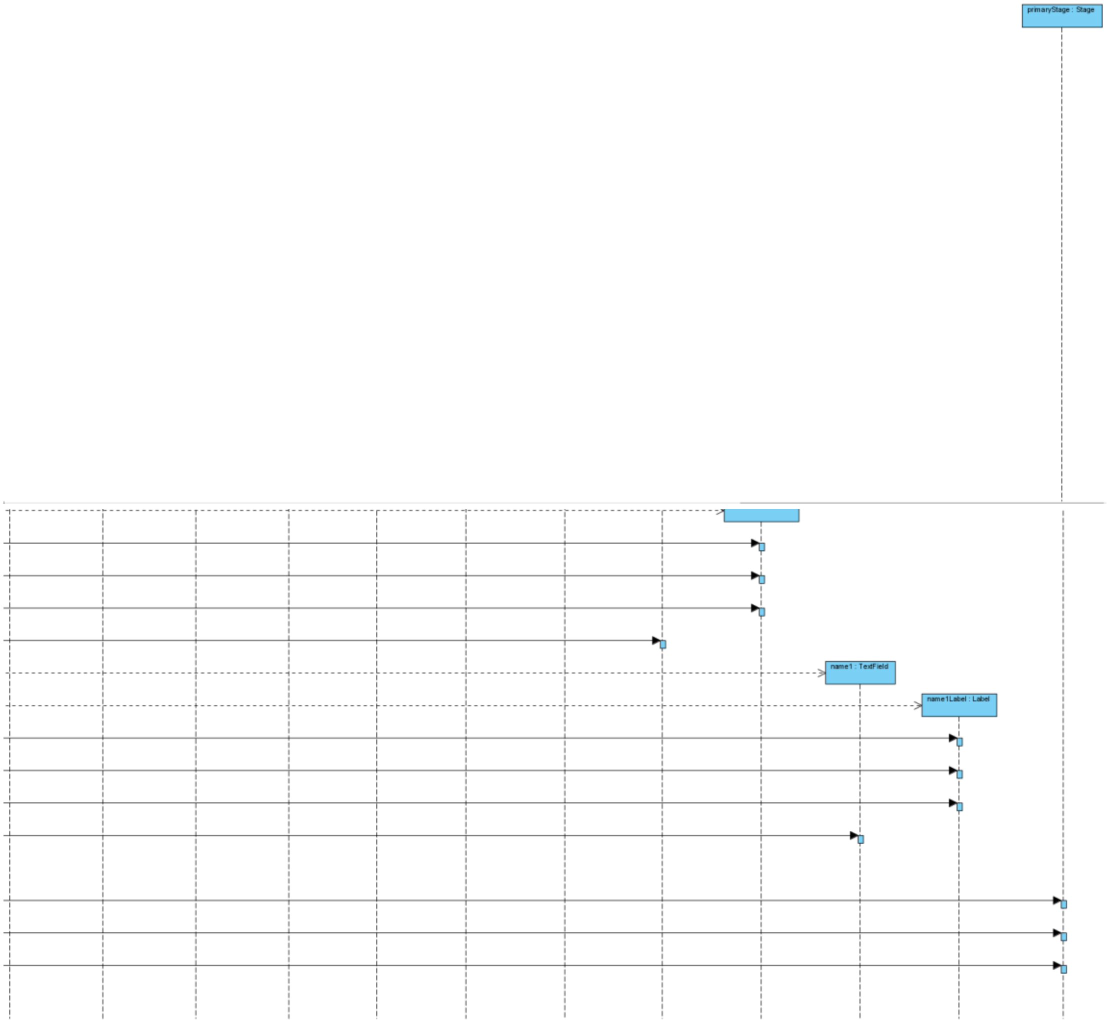

1.72:
1.73: setContentDisplay(ContentDisplay.BOTTOM)
1.76: setOnAction(newEventHandler<ActionEvent>() {
@Override public void handle(ActionEvent click) {
name2.setText(nam
1.74: setLayoutX(220)
two.setName(name2.getText());
twosTurn.setText("It's "+two.getName()+"'s turn!");
rightLabel.setText(name2.getText()+": "+two.getScore());
1.75: setLayoutY(400)
gamePane.getChildren().remove
gamePane.getChildren().add(play);
}
})
1.77:
1.78:
1.79: setContentDisplay(ContentDisplay.BOTTOM)
1.82: setOnAction(newEventHandler<ActionEvent>() {
@Override public void handle(ActionEvent click) {
name1.setText(nam
1.80: setLayoutX(220)
one.setName(name1.getText());
onesTurn.setText("It's "+one.getName()+"'s turn!");
leftLabel.setText(name1.getText()+": "+one.getScore());
1.81: setLayoutY(400)
gamePane.getChildren().remove
gamePane.getChildren().addAll(name2, name2Label);
}
})
1.84: setTitle("The Game of War")
1.85: setScene(gameScene)
1.86: show()

New Row:


twosTurn : javafx.scene.text.Text
win : javafx.scene.text.Text
p1Draw: Button
p1Shuffle : Button
p2Draw: Button
p2Shuffle : Button
play: Button
card1 : ImageView
card2 : ImageView

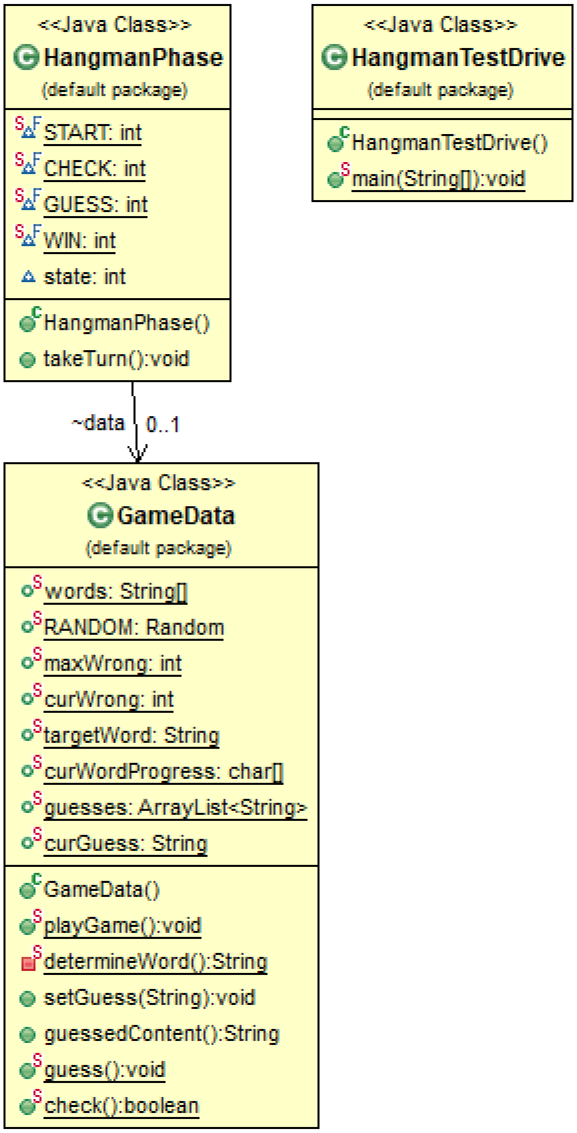

name2 : TextField
name2Label : Label

New Row:


primaryStage : Stage
name1 : TextField
name1Label : Label

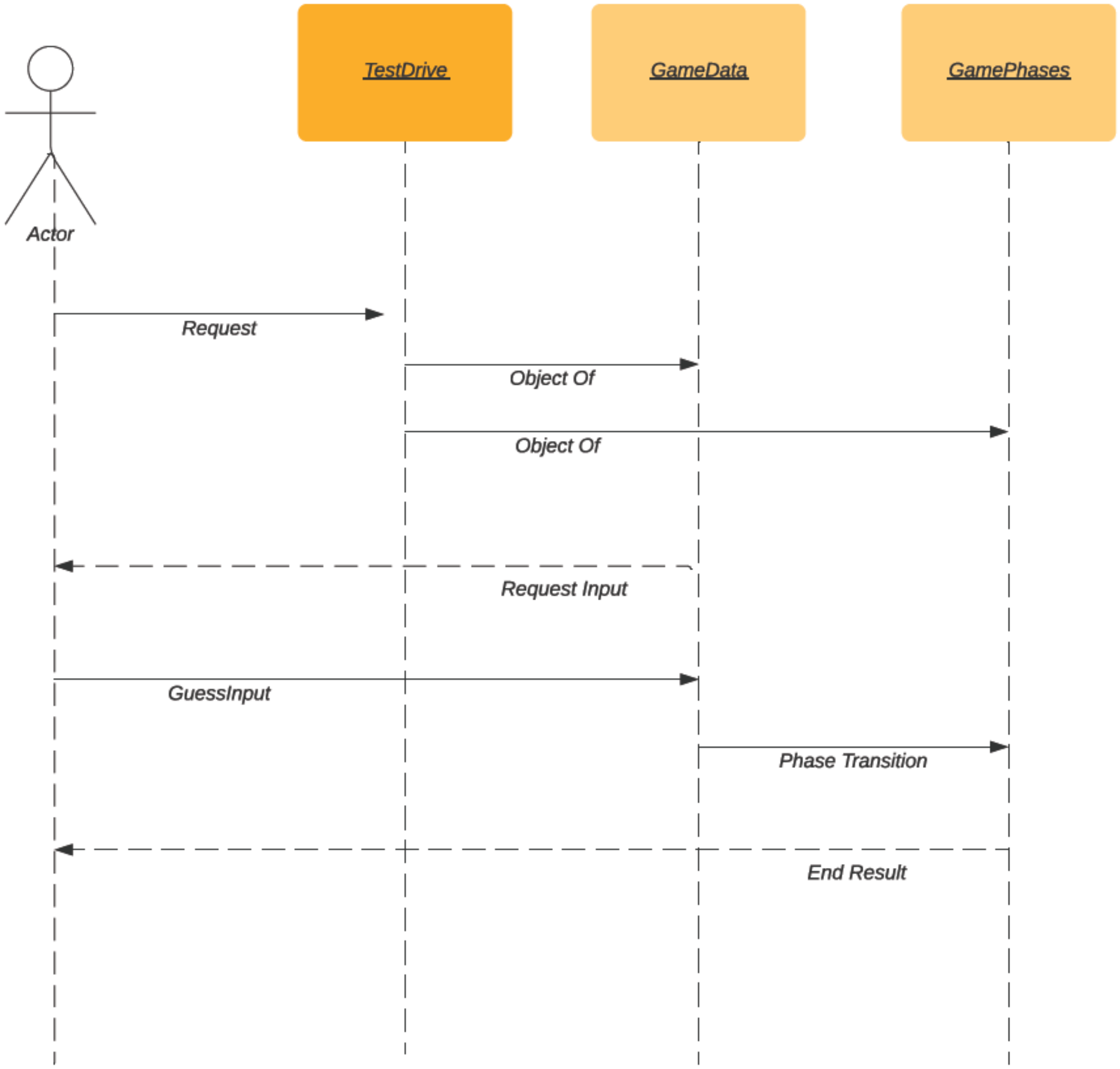
2. State

- UML:



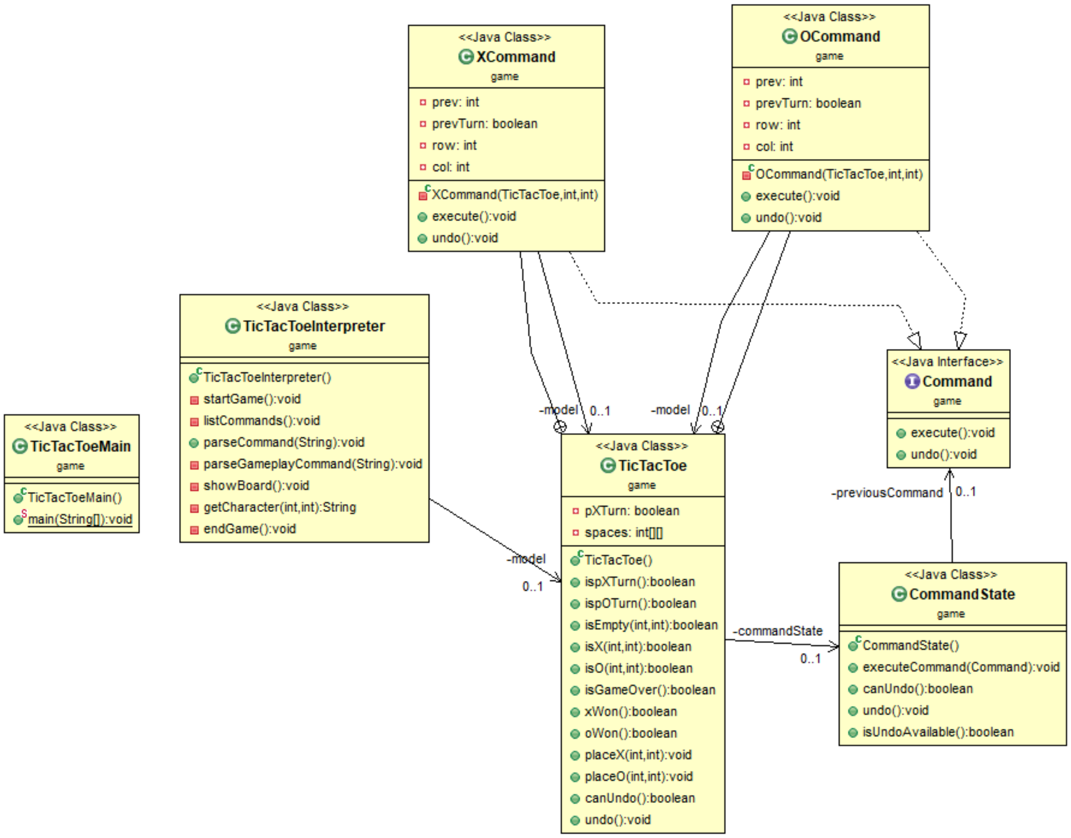
<<Java Class>>
HangmanPhase
(default package)
START: int
CHECK: int
GUESS: int
WIN: int
state: int
HangmanPhase()
takeTurn():void
<<Java Class>>
HangmanTestDrive
(default package)
HangmanTestDrive()
main(String[]):void
~data
0..1
<<Java Class>>
GameData
(default package)
words: String[]
RANDOM: Random
maxWrong: int
curWrong: int
targetWord: String
curWordProgress: char[]
guesses: ArrayList<String>
curGuess: String
GameData()
playGame():void
determineWord():String
setGuess(String):void
guessedContent():String
guess():void
check():boolean

- Sequence:

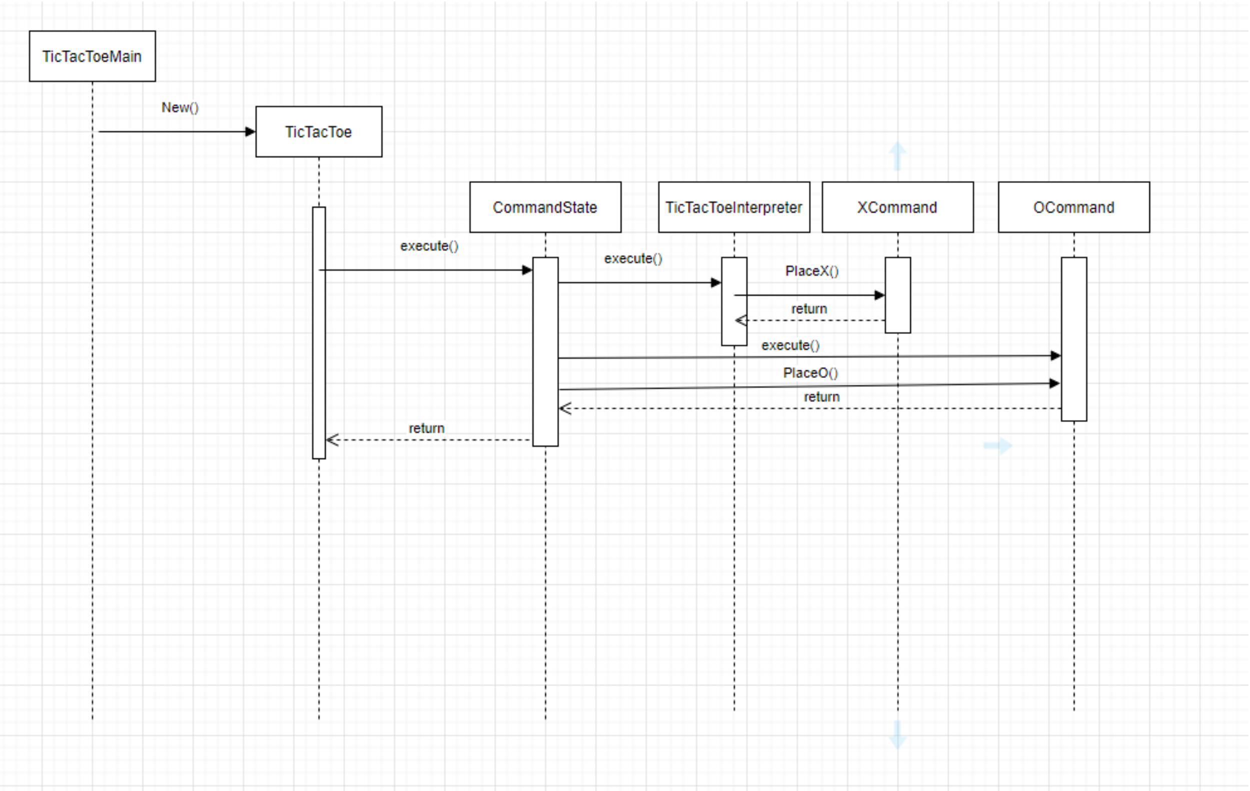


3. Command

UML:

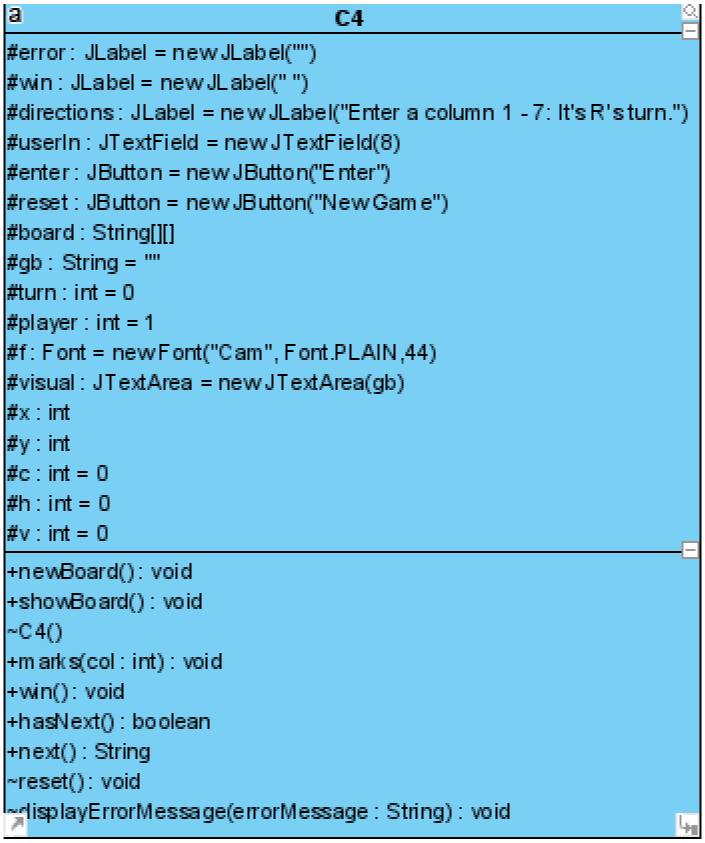

<<Java Class>>
XCommand
game
prev: int
prevTurn: boolean
row: int
col: int
XCommand(TicTacToe,int,int)
execute():void
undo():void
<<Java Class>>
OCommand
game
prev: int
prevTurn: boolean
row: int
col: int
OCommand(TicTacToe,int,int)
execute():void
undo():void
<<Java Class>>
TicTacToeInterpreter
game
TicTacToeInterpreter()
startGame():void
listCommands():void
parseCommand(String):void
parseGameplayCommand(String):void
showBoard():void
getCharacter(int,int):String
endGame():void
<<Java Interface>>
Command
game
execute():void
undo():void
<<Java Class>>
TicTacToeMain
game
TicTacToeMain()
main(String[]):void
-model
0..1
-model
0..1
<<Java Class>>
TicTacToe
game
pXTurn: boolean
spaces: int[][]
TicTacToe()
ispXTurn():boolean
ispOTurn():boolean
isEmpty(int,int):boolean
isX(int,int):boolean
isO(int,int):boolean
isGameOver():boolean
xWon():boolean
oWon():boolean
placeX(int,int):void
placeO(int,int):void
canUndo():boolean
undo():void
-previousCommand
0..1
-model
0..1
-commandState
0..1
<<Java Class>>
CommandState
game
CommandState()
executeCommand(Command):void
canUndo():boolean
undo():void
isUndoAvailable():boolean

Sequence


TicTacToeMain
New()
TicTacToe
CommandState
TicTacToeInterpreter
XCommand
OCommand
execute()
execute()
PlaceX()
return
execute()
PlaceO()
return
return

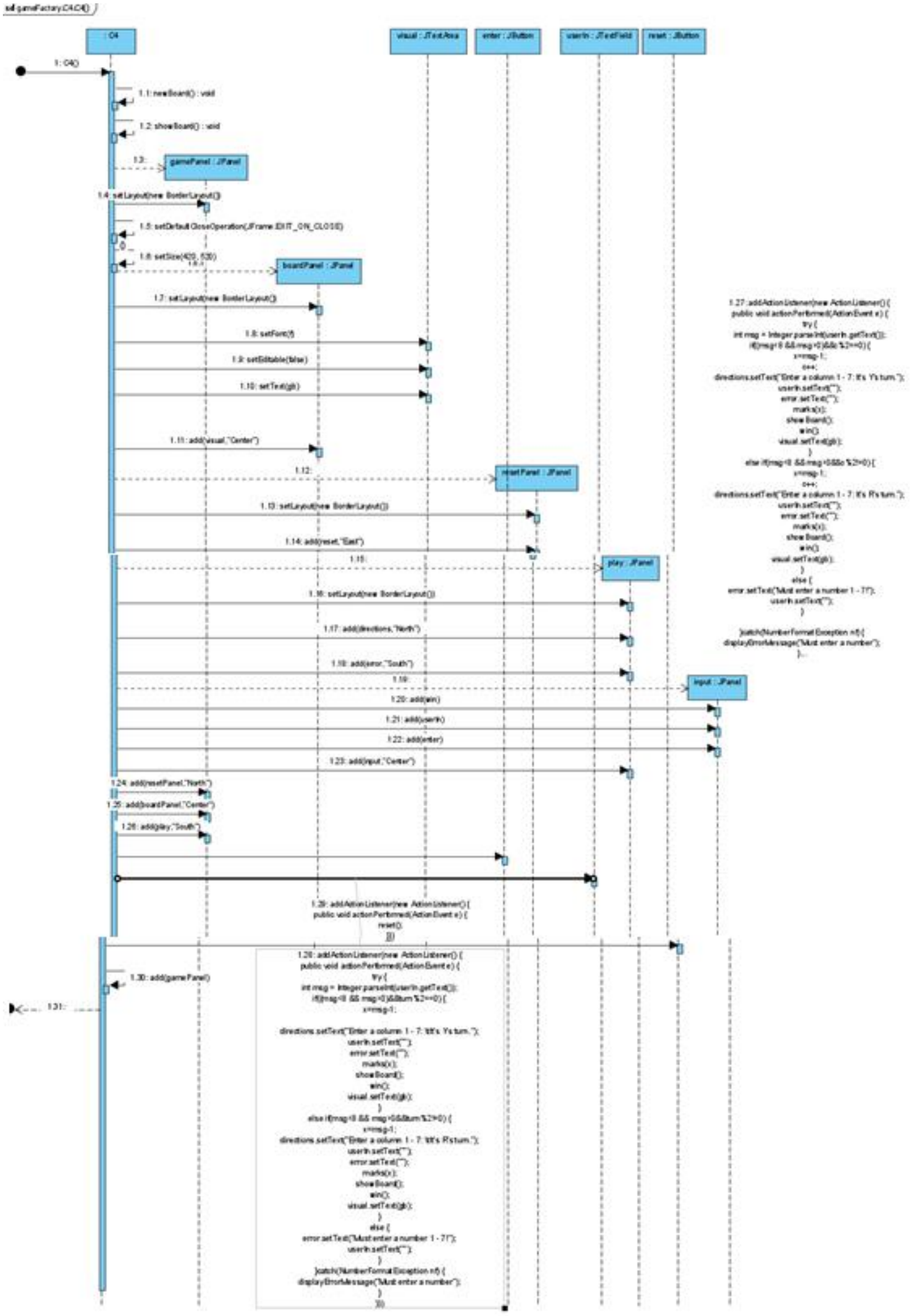
4. Iterator

UML:



```
C4
#error : JLabel = new JLabel("")
#win : JLabel = new JLabel(" ")
#directions : JLabel = new JLabel("Enter a column 1 - 7: It's R's turn.")
#userIn : JTextField = new JTextField(8)
#enter : JButton = new JButton("Enter")
#reset : JButton = new JButton("New Game")
#board : String[][]
#gb : String = ""
#turn : int = 0
#player : int = 1
#f : Font = new Font("Cam", Font.PLAIN,44)
#visual : JTextArea = new JTextArea(gb)
#x : int
#y : int
#c : int = 0
#h : int = 0
#v : int = 0
+newBoard() : void
+showBoard() : void
~C4()
+marks(col : int) : void
+win() : void
+hasNext() : boolean
+next() : String
~reset() : void
~displayErrorMessage(errorMessage : String) : void
```

Sequence Diagram:



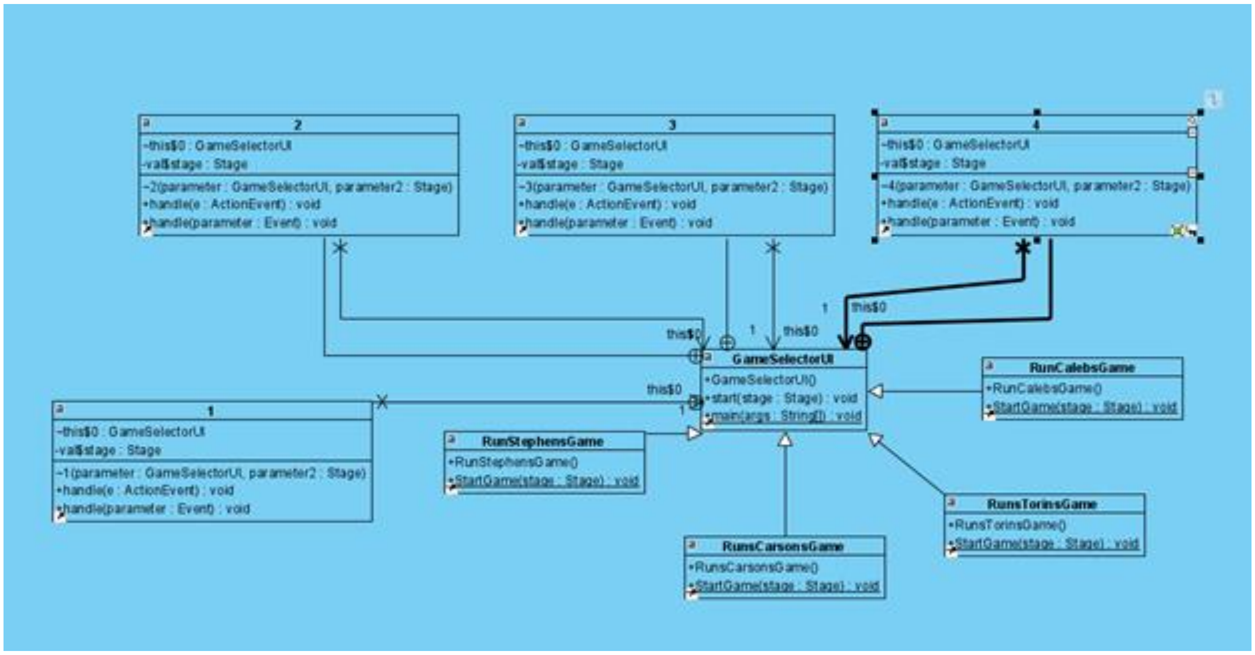
```
1.27: addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
try {
int msg = Integer.parseInt(userIn.getText());
if((msg<8 && msg>0)&&c %2==0) {
x=msg-1;
c++;
directions.setText("Enter a column 1 - 7: It's Y's turn.");
userIn.setText("");
error.setText("");
marks(x);
showBoard();
win();
visual.setText(gb);
}
else if(msg<8 && msg>0&&c %2!=0) {
x=msg-1;
c++;
directions.setText("Enter a column 1 - 7: It's R's turn.");
userIn.setText("");
error.setText("");
marks(x);
showBoard();
win();
visual.setText(gb);
}
else {
error.setText("Must enter a number 1 - 7!");
userIn.setText("");
}

}catch(NumberFormatException nf) {
displayErrorMessage("Must enter a number");
}...
```

```
1.29: addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
reset();
}})
```

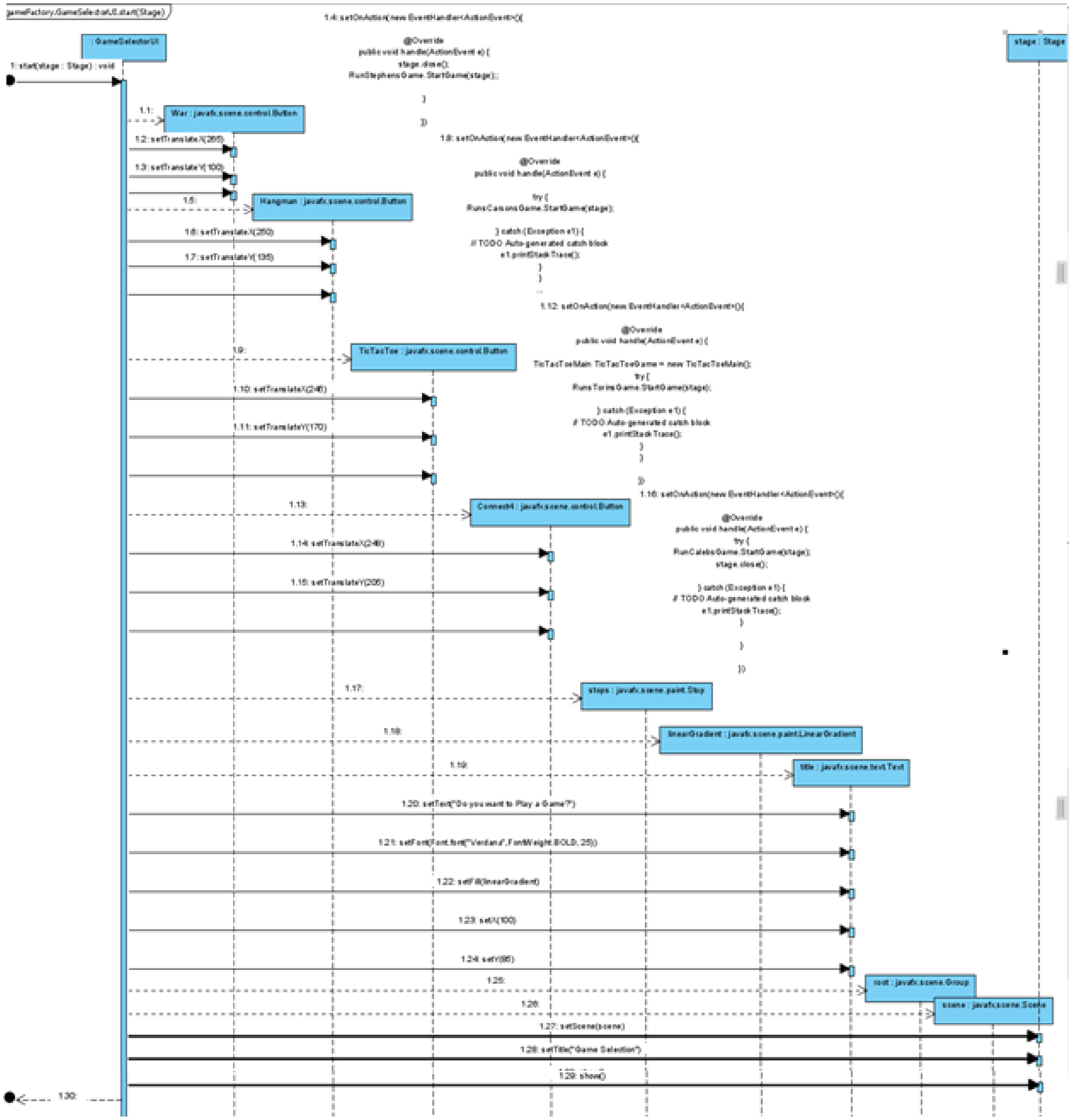
```
1.28: addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
try {
int msg = Integer.parseInt(userIn.getText());
if((msg<8 && msg>0)&&turn %2==0) {
x=msg-1;

directions.setText("Enter a column 1 - 7: \tIt's Y's turn.");
userIn.setText("");
error.setText("");
marks(x);
showBoard();
win();
visual.setText(gb);
}
else if(msg<8 && msg>0&&turn %2!=0) {
x=msg-1;
directions.setText("Enter a column 1 - 7: \tIt's R's turn.");
userIn.setText("");
error.setText("");
marks(x);
showBoard();
win();
visual.setText(gb);
}
else {
error.setText("Must enter a number 1 - 7!");
userIn.setText("");
}
}catch(NumberFormatException nf) {
displayErrorMessage("Must enter a number");
}
}});
```

5. Design

UML:


2
~this$0 : GameSelectorUI
-val$stage : Stage
~2(parameter : GameSelectorUI, parameter2 : Stage)
+handle(e : ActionEvent) : void
+handle(parameter : Event) : void
3
~this$0 : GameSelectorUI
-val$stage : Stage
~3(parameter : GameSelectorUI, parameter2 : Stage)
+handle(e : ActionEvent) : void
+handle(parameter : Event) : void
4
~this$0 : GameSelectorUI
-val$stage : Stage
~4(parameter : GameSelectorUI, parameter2 : Stage)
+handle(e : ActionEvent) : void
+handle(parameter : Event) : void
1
this$0
this$0
1
this$0
1
this$0
1
GameSelectorUI
+GameSelectorUI()
+start(stage : Stage) : void
+main(args : String[]) : void
1
~this$0 : GameSelectorUI
-val$stage : Stage
~1(parameter : GameSelectorUI, parameter2 : Stage)
+handle(e : ActionEvent) : void
+handle(parameter : Event) : void
RunStephensGame
+RunStephensGame()
+StartGame(stage : Stage) : void
RunCalebsGame
+RunCalebsGame()
+StartGame(stage : Stage) : void
RunsTorinsGame
+RunsTorinsGame()
+StartGame(stage : Stage) : void
RunsCarsonsGame
+RunsCarsonsGame()
+StartGame(stage : Stage) : void

Sequence Diagram:


gameFactory.GameSelectorUI.start(Stage)
: GameSelectorUI
stage : Stage
1: start(stage : Stage) : void
1.1:
War : javafx.scene.control.Button
1.2: setTranslateX(265)
1.3: setTranslateY(100)
1.5:
Hangman : javafx.scene.control.Button
1.6: setTranslateX(250)
1.7: setTranslateY(135)
1.9:
TicTacToe : javafx.scene.control.Button
1.10: setTranslateX(246)
1.11: setTranslateY(170)
1.13:
Connect4 : javafx.scene.control.Button
1.14: setTranslateX(248)
1.15: setTranslateY(205)
1.17:
stops : javafx.scene.paint.Stop
1.18:
linearGradient : javafx.scene.paint.LinearGradient
1.19:
title : javafx.scene.text.Text
1.20: setText("Do you want to Play a Game?")
1.21: setFont(Font.font("Verdana",FontWeight.BOLD, 25))
1.22: setFill(linearGradient)
1.23: setX(100)
1.24: setY(85)
1.25:
root : javafx.scene.Group
1.26:
scene : javafx.scene.Scene
1.27: setScene(scene)
1.28: setTitle("Game Selection")
1.29: show()
1.30:

```
1.4: setOnAction( new EventHandler<ActionEvent>(){

@Override
public void handle(ActionEvent e) {
stage.close();
RunStephensGame. StartGame(stage);;

}

});
```

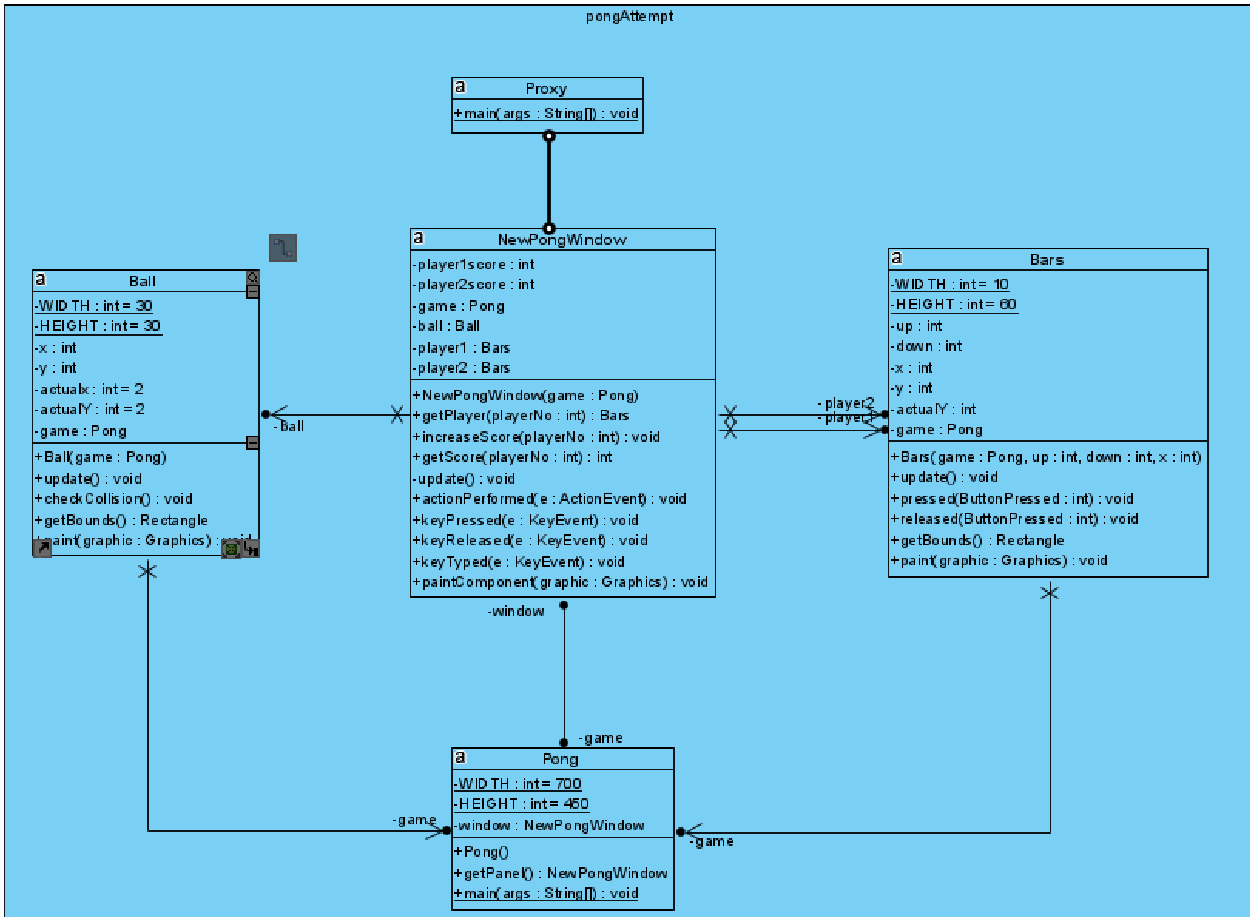
```
1.8: setOnAction( new EventHandler<ActionEvent>(){

@Override
public void handle(ActionEvent e) {

try {
RunsCarsonsGame.StartGame(stage);

} catch (Exception e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
}
...
```

```
1.12: setOnAction(new EventHandler<ActionEvent>(){

@Override
public void handle(ActionEvent e) {

TicTacToeMain TicTacToeGame = new TicTacToeMain();
try {
RunsTorinsGame.StartGame(stage);

} catch (Exception e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
}

});
```

```
1.16: setOnAction(new EventHandler<ActionEvent>(){

@Override
public void handle(ActionEvent e) {
try {
RunCalebsGame.StartGame(stage);
stage.close();

} catch (Exception e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}

}

});
```

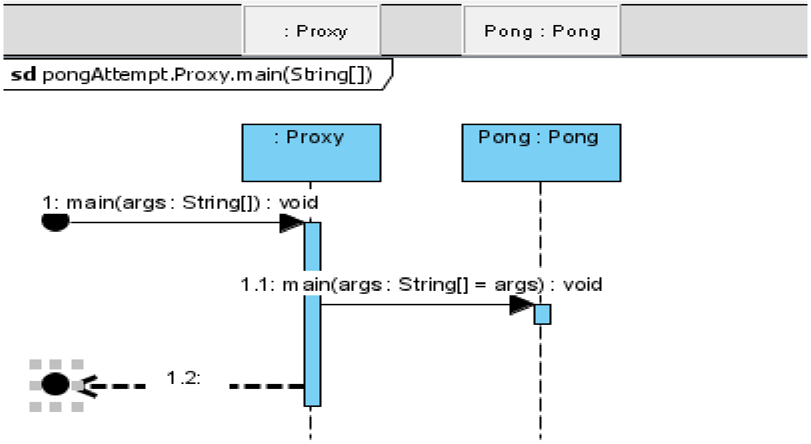
6.Proxy

UML:



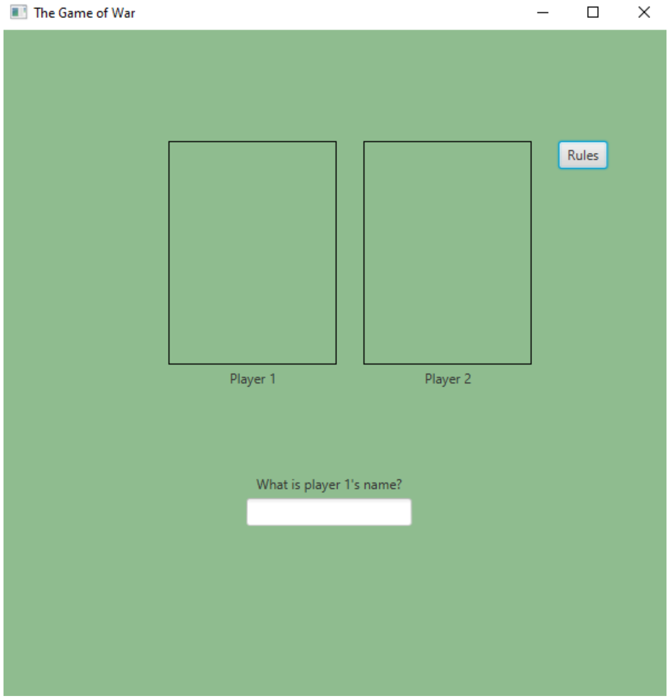
Sequence Diagram:

Proxy:

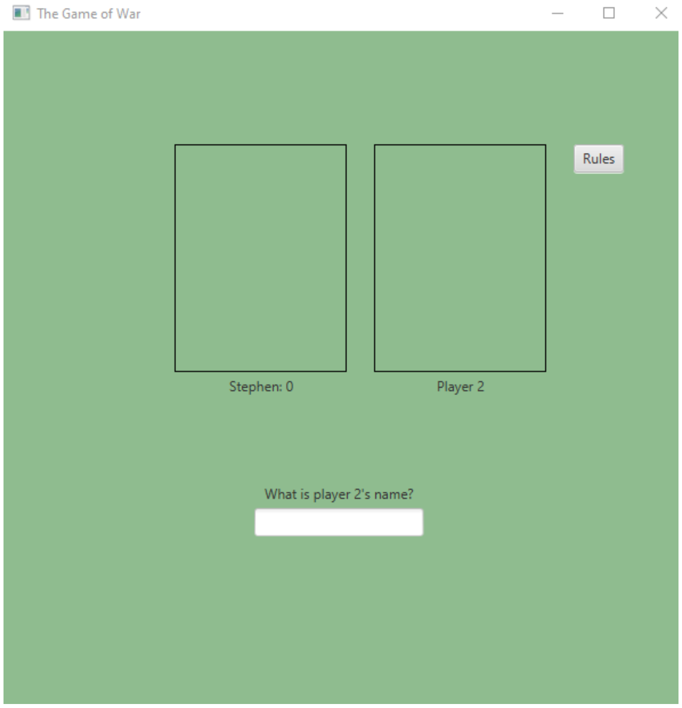

: Proxy
Pong : Pong
sd pongAttempt.Proxy.main(String[])
: Proxy
Pong : Pong
1: main(args : String[]) : void
1.1: main(args : String[] = args) : void
1.2:

Testing(MVC Pattern/War Card Game):
The war card game has been tested with two players, where both of them are human controlled. The game successfully adds up points and declares a winner when the game has concluded.

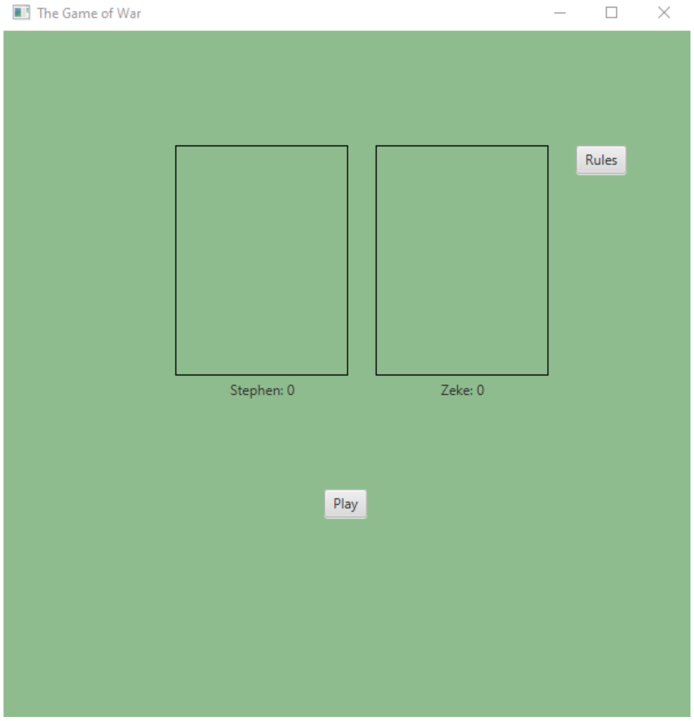
Game initializes with a prompt saying to enter player 1's name:



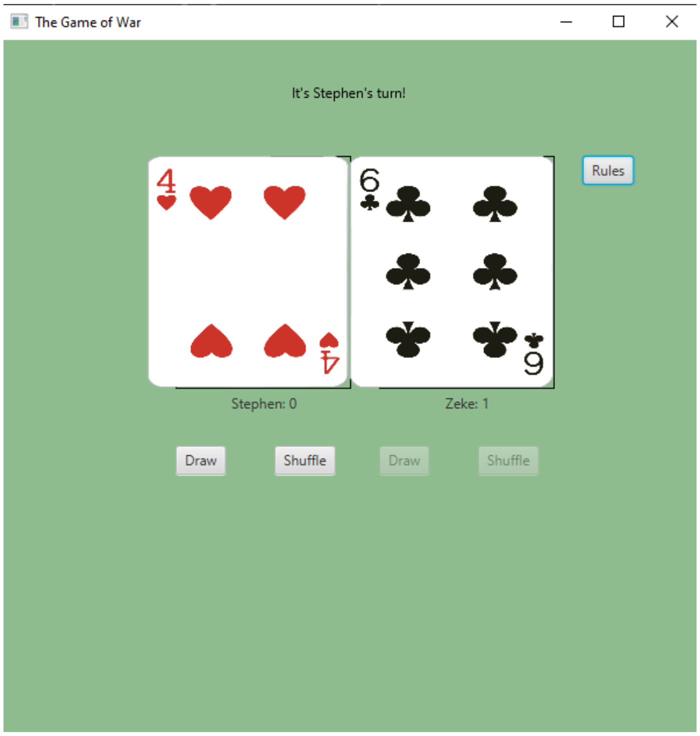
After player's 1 name is entered a prompt appears to enter player's 2 name:



After both name are entered a play button appears:



After the play button is pressed four buttons appear and each player hits the button draw once at a time. If a player's card is higher than the other that player gets a point.

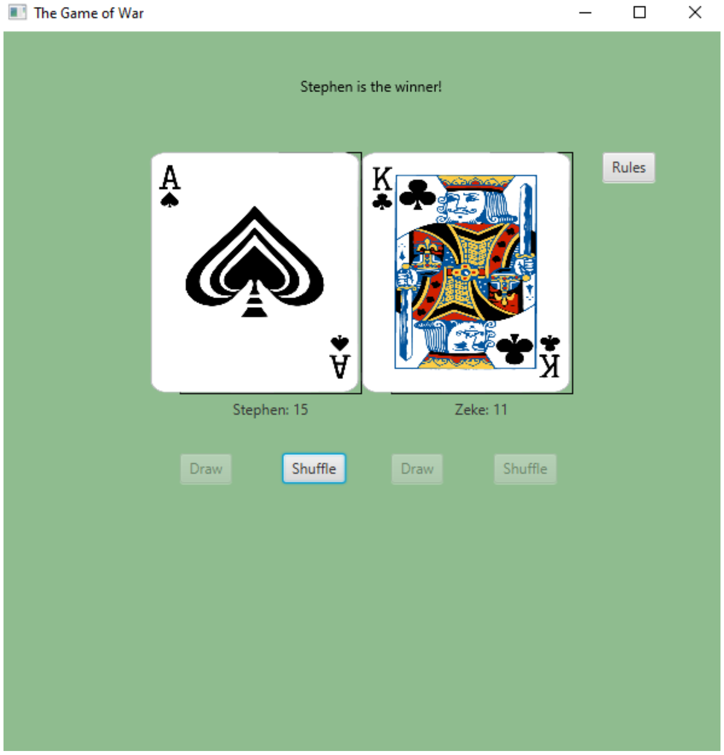


It's Stephen's turn!

Stephen: 0

Zeke: 1

When a players deck runs out (26 plays total) the other player wins and a string appears announcing the winner



Stephen is the winner!

Stephen: 15

Zeke: 11

Use Case MVC Pattern/War Card Game: Player vs. Player

Primary Actors: Player 1, Player 2

Preconditions: A deck of 52 cards dealt evenly between 2 players

Stakeholders and Their Interests:

- Player 1: Wants to play their cards in their hand one a time where that card is compared against player 2's card.
- Player 2: Wants to play their cards in their hand one a time where that card is compared against player 1's card.

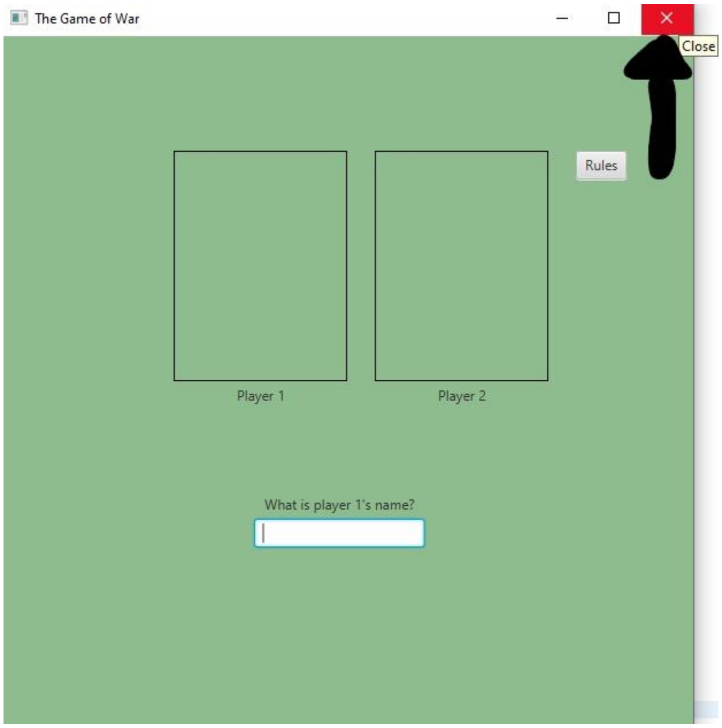
Main Success Scenario:

- Player 1 and Player 2 each play their hands until empty.
- Points are assigned based on the outcome of each play
- After 26 plays the player with the most points wins and a message appears declaring the winner.

Scenario Variation:

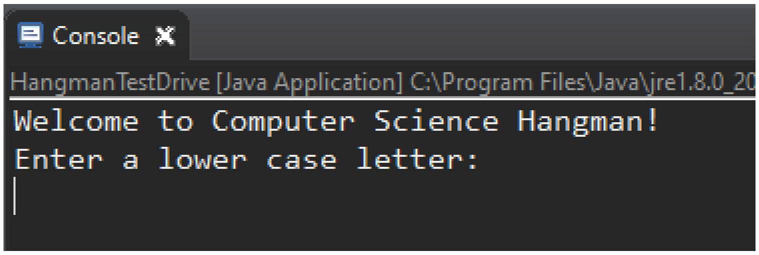
- The game is closed before its completed:
  - The game exits and is completely restarted when started again.

Scenario Operational Example:



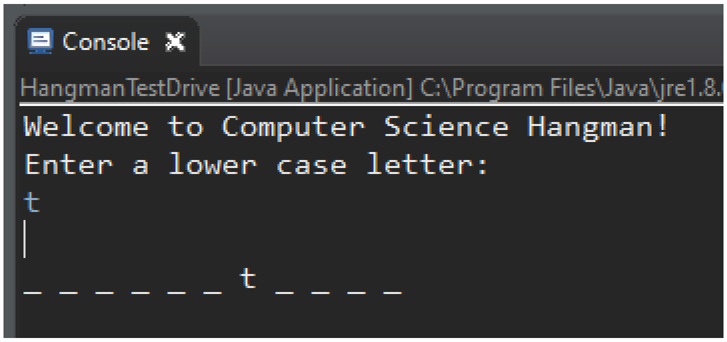
Example run/State Diagram game use case:

1. Player starts the hangman program



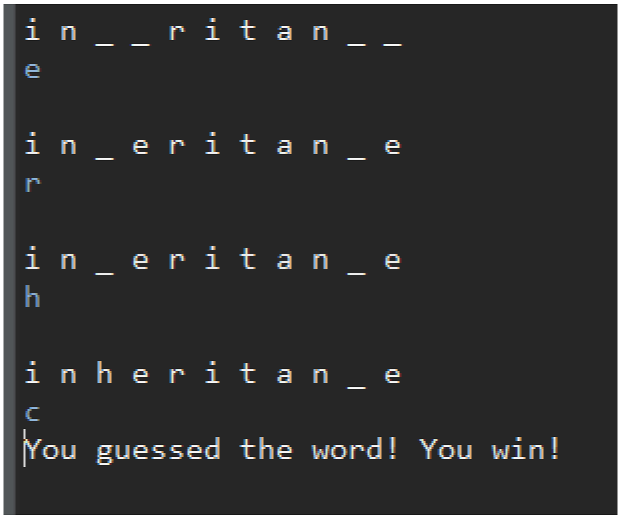
```
HangmanTestDrive [Java Application] C:\Program Files\Java\jre1.8.0_20
Welcome to Computer Science Hangman!
Enter a lower case letter:
```

2. Player enters a letter to guess
3. Player is presented with output. Either a correct guess now shows in the output, or the incorrect guess is tallied towards the total.



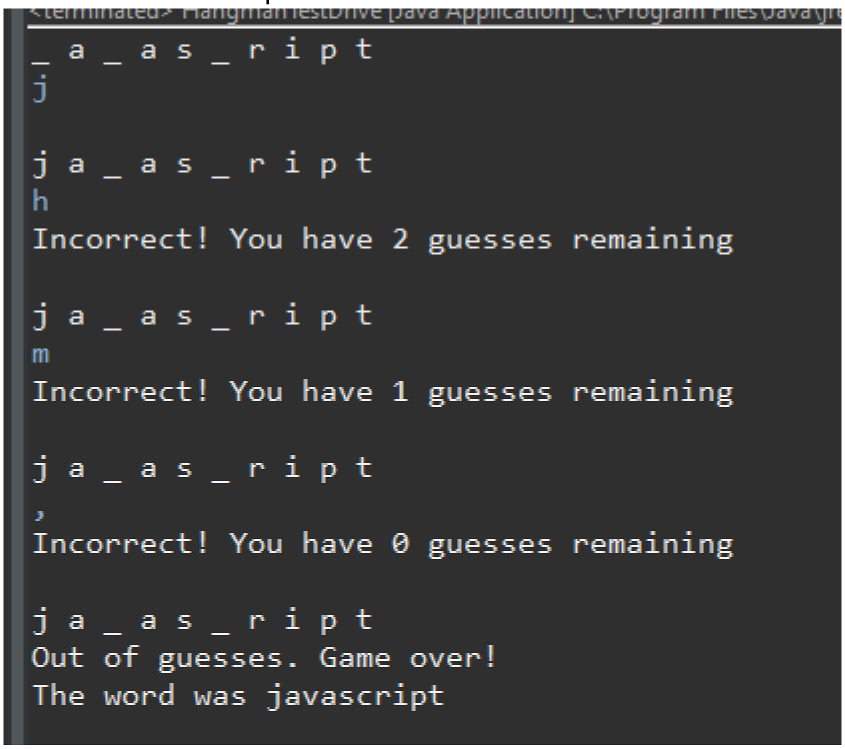
```
HangmanTestDrive [Java Application] C:\Program Files\Java\jre1.8.
Welcome to Computer Science Hangman!
Enter a lower case letter:
t

_ _ _ _ _ _ t _ _ _ _
```

4. Player either correctly guesses the word



```
i n _ _ r i t a n _ _
e

i n _ e r i t a n _ e
r

i n _ e r i t a n _ e
h

i n h e r i t a n _ e
c
You guessed the word! You win!
```

Or runs out of attempts

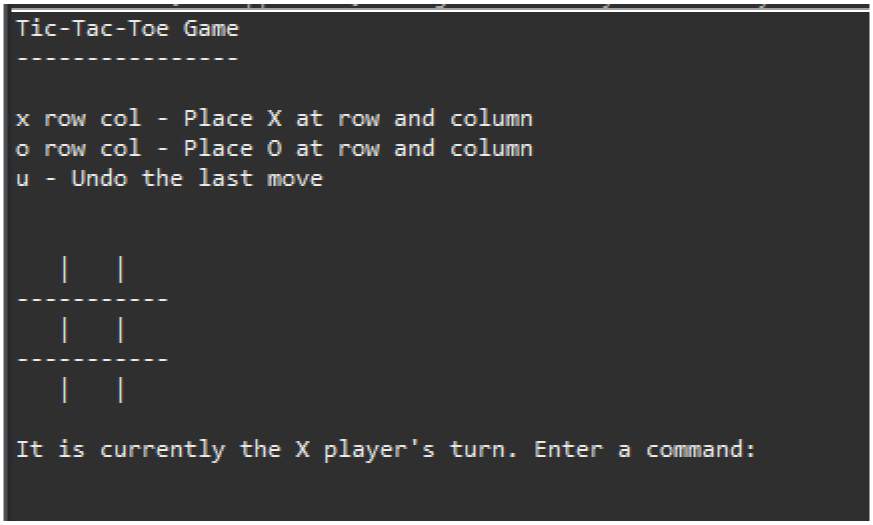


```
<terminated> HangmanTestDrive [Java Application] C:\Program Files\Java\jre
_ a _ a s _ r i p t
j

j a _ a s _ r i p t
h
Incorrect! You have 2 guesses remaining

j a _ a s _ r i p t
m
Incorrect! You have 1 guesses remaining

j a _ a s _ r i p t
,
Incorrect! You have 0 guesses remaining

j a _ a s _ r i p t
Out of guesses. Game over!
The word was javascript
```

TicTacToe Usage Case#1

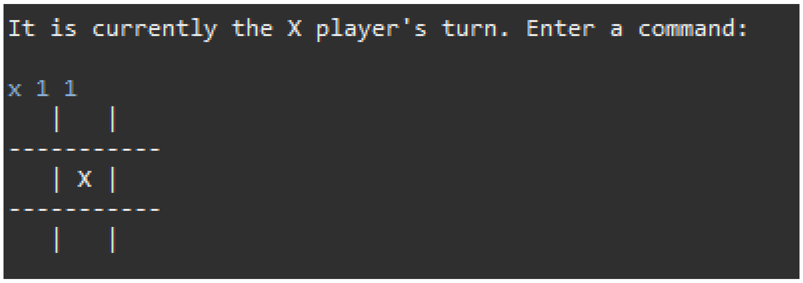
Two Friends want to play TicTacToe with each other.

1. User is prompted with commands for X and O



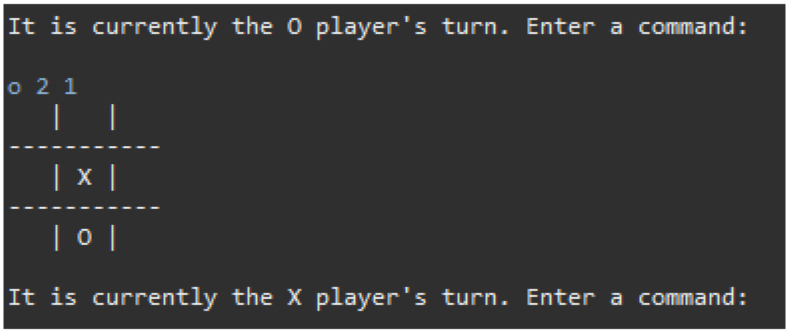
```
Tic-Tac-Toe Game
----------------

x row col - Place X at row and column
o row col - Place O at row and column
u - Undo the last move


   |   |
-----------
   |   |
-----------
   |   |

It is currently the X player's turn. Enter a command:
```

2. User enters where to place X



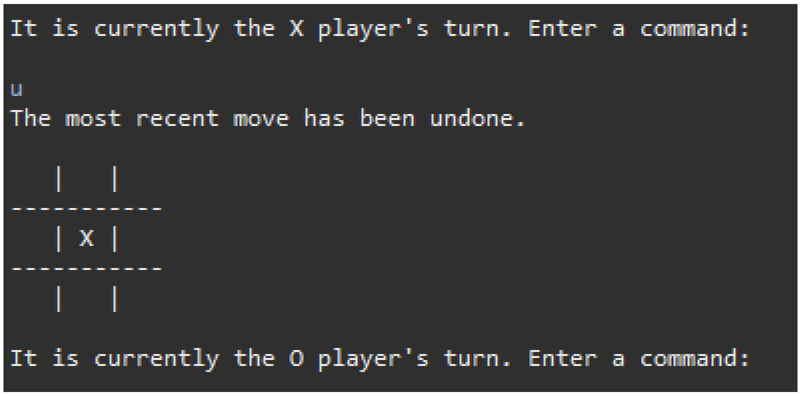
```
It is currently the X player's turn. Enter a command:

x 1 1
   |   |
-----------
   | X |
-----------
   |   |
```

3. User enters where to place O



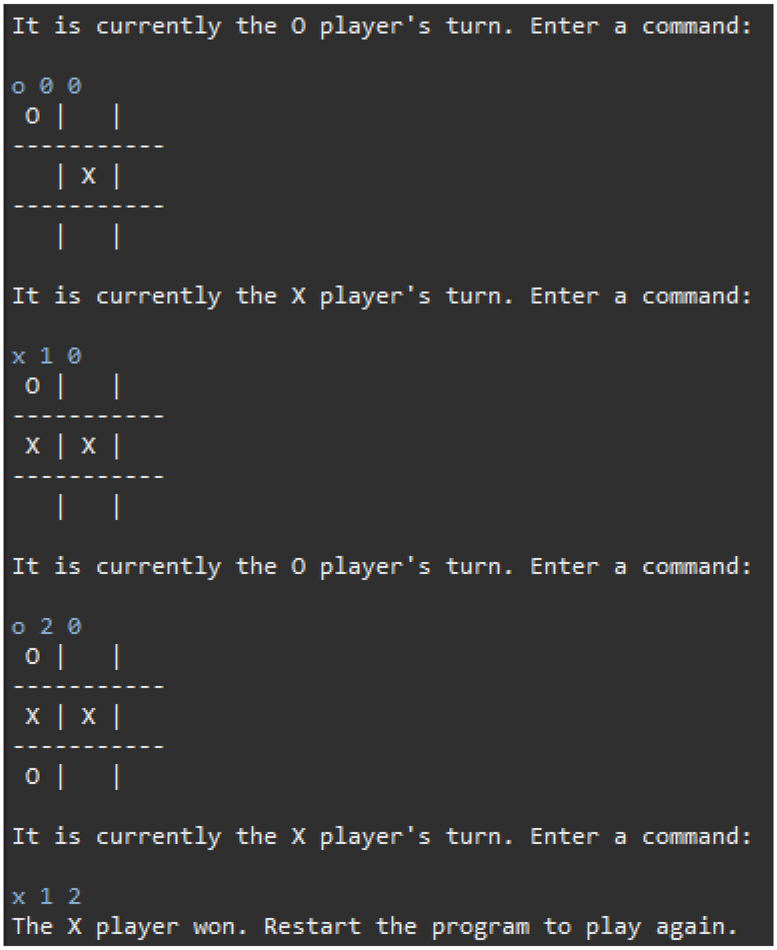
```
It is currently the O player's turn. Enter a command:

o 2 1
   |   |
-----------
   | X |
-----------
   | O |

It is currently the X player's turn. Enter a command:
```

4. At any time if the user makes a mistake they can press “u” for undo. The undo feature is handled using the command pattern.



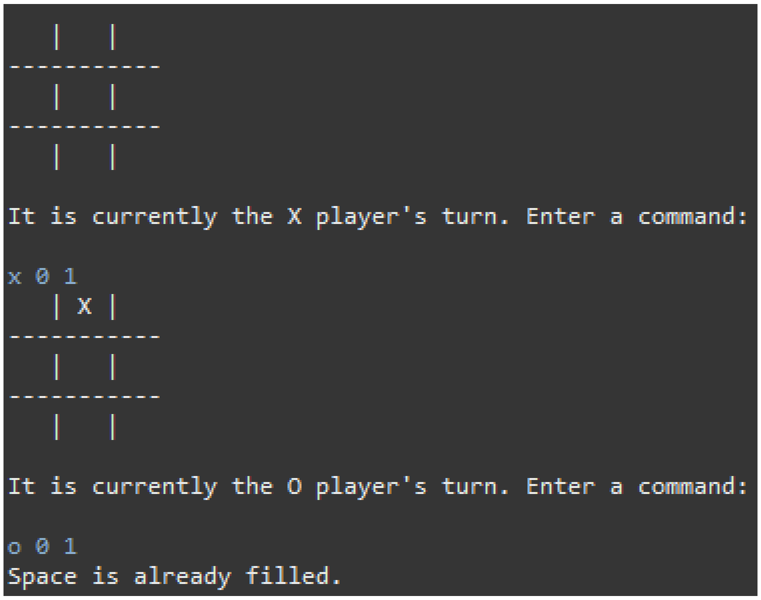
```
It is currently the X player's turn. Enter a command:

u
The most recent move has been undone.

   |   |
-----------
   | X |
-----------
   |   |

It is currently the O player's turn. Enter a command:
```

5. Game continues until a player wins, or the game is tied.



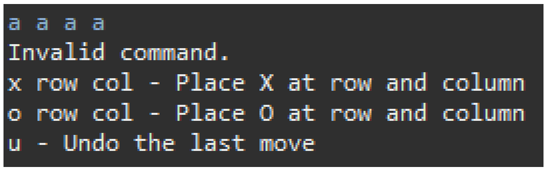
```
It is currently the O player's turn. Enter a command:

o 0 0
 O |   |
-----------
   | X |
-----------
   |   |

It is currently the X player's turn. Enter a command:

x 1 0
 O |   |
-----------
 X | X |
-----------
   |   |

It is currently the O player's turn. Enter a command:

o 2 0
 O |   |
-----------
 X | X |
-----------
 O |   |

It is currently the X player's turn. Enter a command:

x 1 2
The X player won. Restart the program to play again.
```

6. If at any point a user enters invalid input (Figure 1) or tries to occupy a square that already has an X or O (Figure 2) they will be prompted with the command list again. Both cases are depicted below.



```
   |   |
-----------
   |   |
-----------
   |   |

It is currently the X player's turn. Enter a command:

x 0 1
   | X |
-----------
   |   |
-----------
   |   |

It is currently the O player's turn. Enter a command:

o 0 1
Space is already filled.
```

Figure 1



```
a a a a
Invalid command.
x row col - Place X at row and column
o row col - Place O at row and column
u - Undo the last move
```

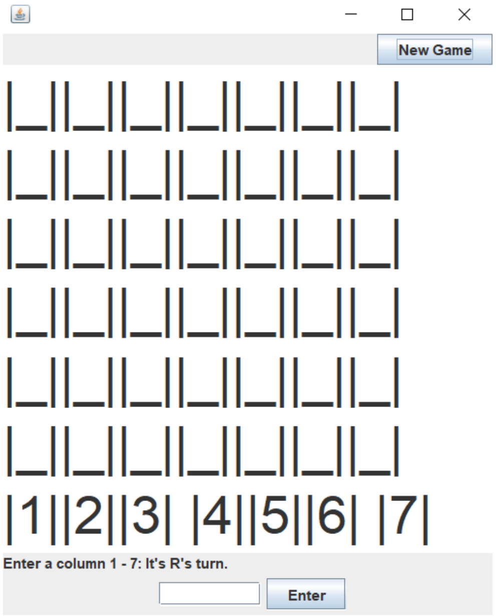
Figure 2

Connect 4 Use Case-

Directions:
Enter a number (1 - 7) into the text box and press enter to place a tile into that column.
Player 1 will go first and use the R tiles.
Player 2 goes second and uses the Y tiles.
Turns will alternate after every valid move.
You can click the New Game button to start a new game at any point.
THERE IS NO UNDO BUTTON. ALL MOVES ARE FINAL.

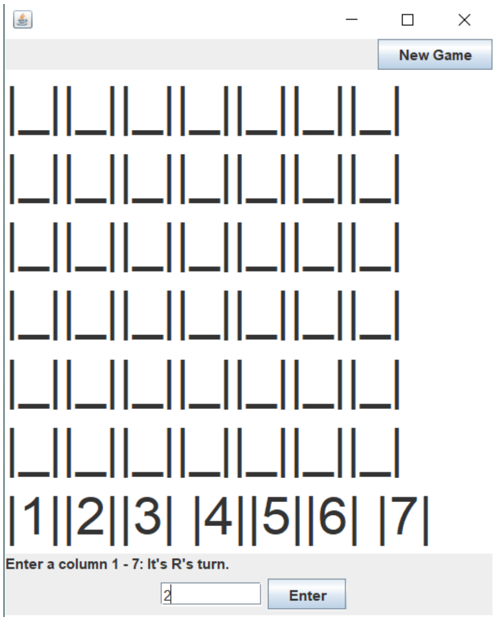
Objective:
Be the first player to get 4 tilles in-a-row. This can be done horizontally, vertically or diagonally.

The start of the game

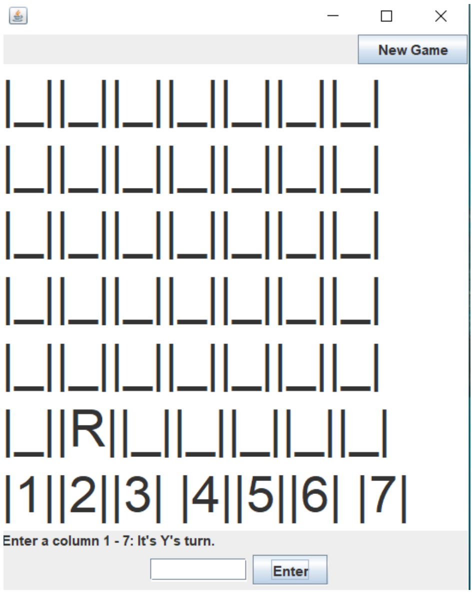


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
```

Player 1 enters a 2 and then an R tile fills the empty part of the column

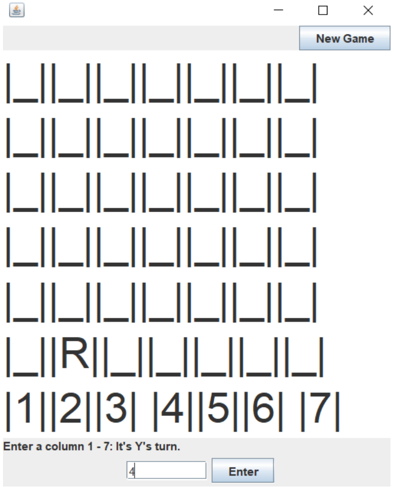


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
2
```

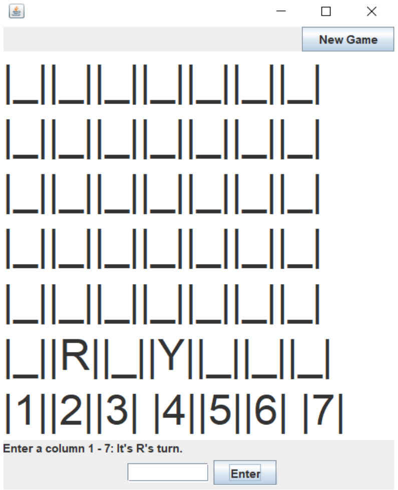


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||R||_||_||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's Y's turn.
```

Then player two enters a 4 and that column get a Y added to it

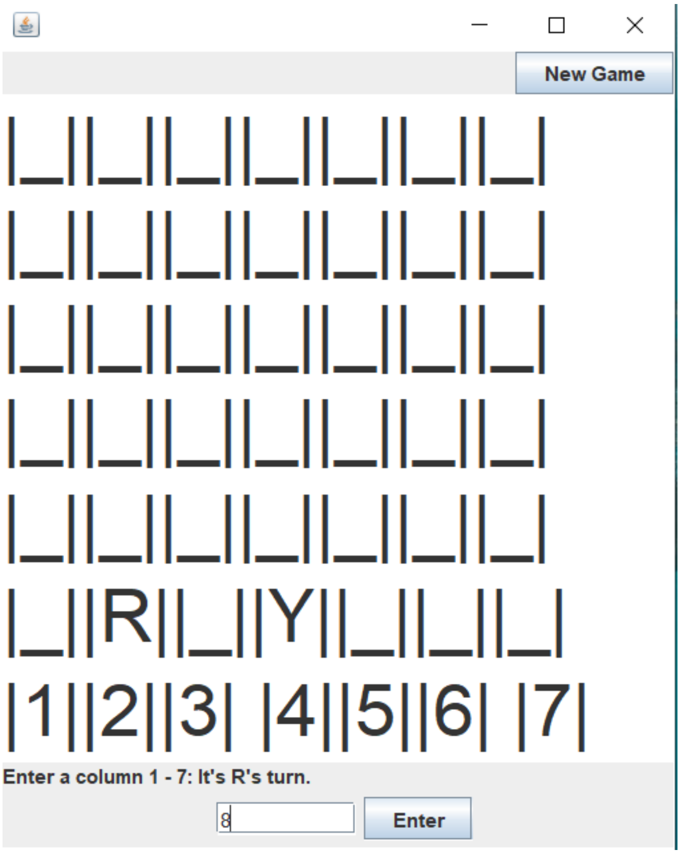


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||R||_||_||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's Y's turn.
4
```

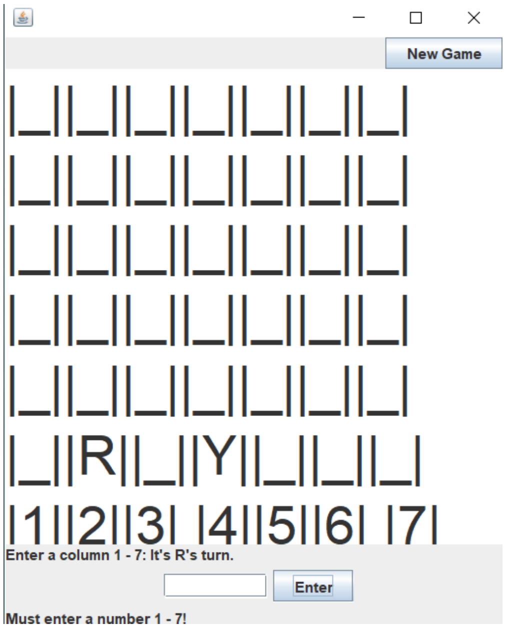


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
```

If the player enters a number that isn’t 1 through 7 they receive an error message

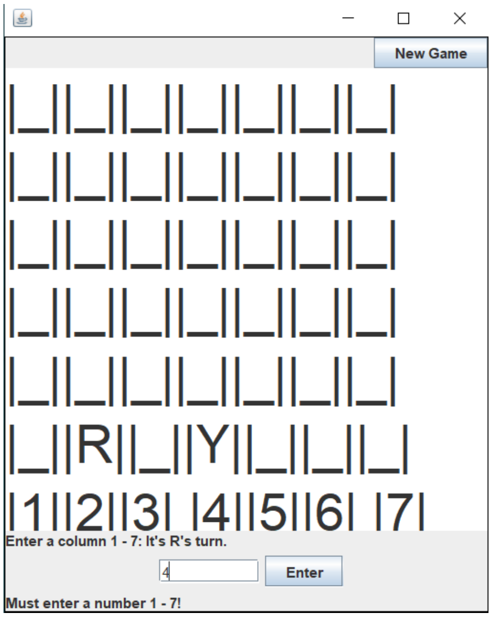


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
8
```

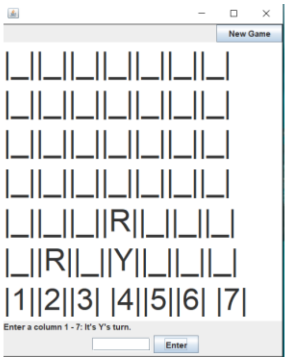


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
Must enter a number 1 - 7!
```

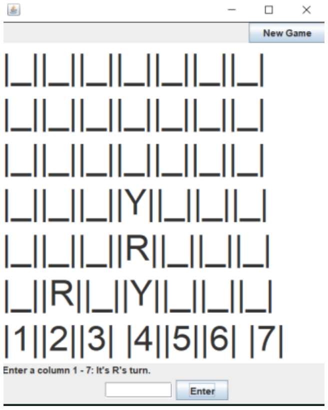
Play then continues as players make valid moves



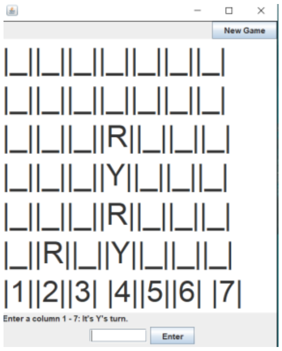
```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
4
Must enter a number 1 - 7!
```



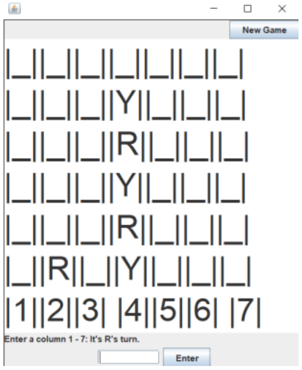
```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's Y's turn.
```



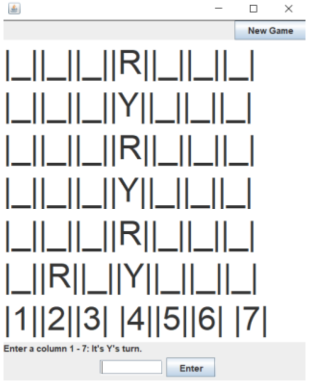
```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
```



```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's Y's turn.
```

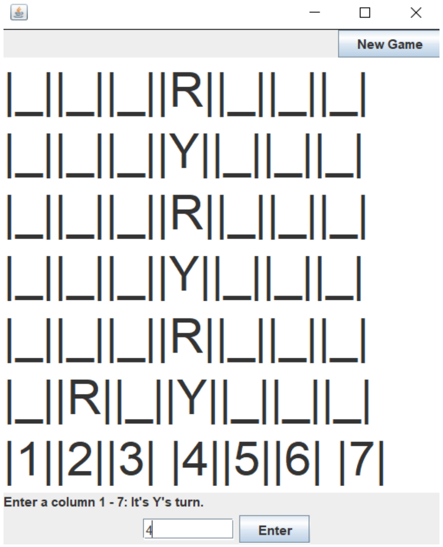


```
|_||_||_||_||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's R's turn.
```

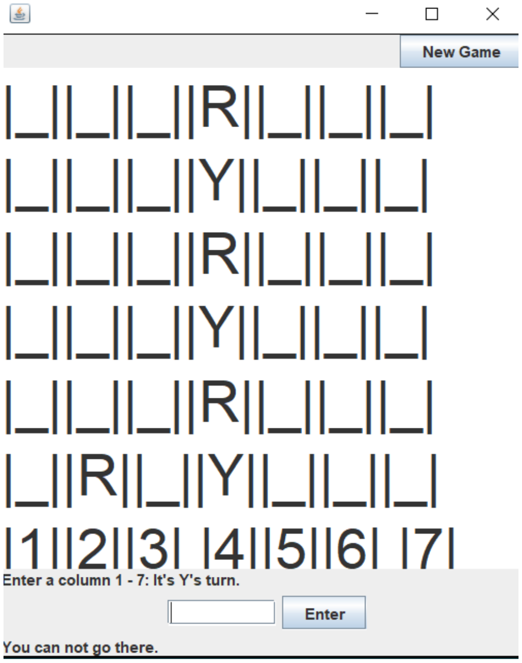


```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's Y's turn.
```

If a player enters into a column that is already full they get an error message.



```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
Enter a column 1 - 7: It's Y's turn.
4
```

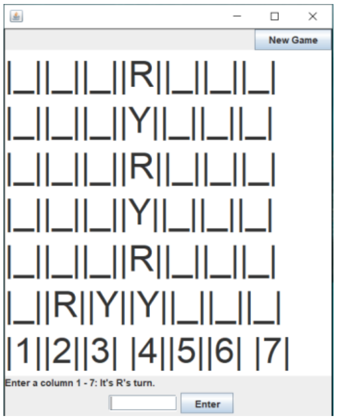


```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|1||2||3| |4||5||6| |7|
```

Enter a column 1 - 7: It's Y's turn.

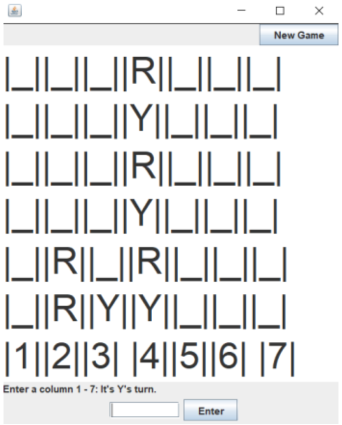
You can not go there.

Play continues until another player enter something incorrect or there is a victor



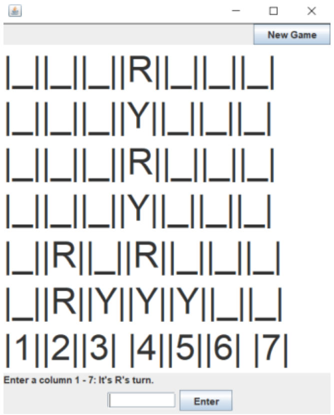
```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||Y||Y||_||_||_|
|1||2||3| |4||5||6| |7|
```

Enter a column 1 - 7: It's R's turn.



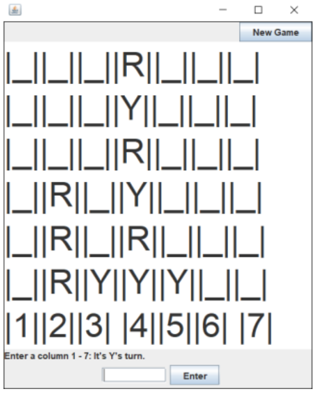
```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||R||_||R||_||_||_|
|_||R||Y||Y||_||_||_|
|1||2||3| |4||5||6| |7|
```

Enter a column 1 - 7: It's Y's turn.



```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||R||_||R||_||_||_|
|_||R||Y||Y||Y||_||_|
|1||2||3| |4||5||6| |7|
```

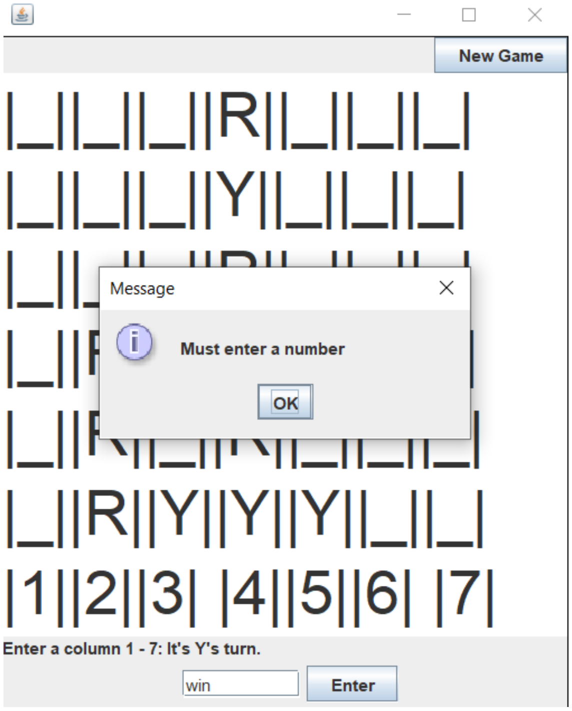
Enter a column 1 - 7: It's R's turn.



```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|_||R||_||R||_||_||_|
|_||R||Y||Y||Y||_||_|
|1||2||3| |4||5||6| |7|
```

Enter a column 1 - 7: It's Y's turn.

If a player enters something that isn't a number they get a popup error.



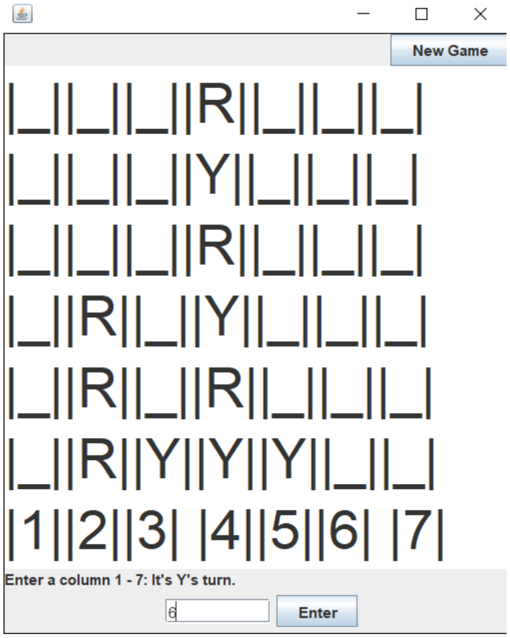
```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_
|_||F
|_||R
|_||R||Y||Y||Y||_||_|
|1||2||3| |4||5||6| |7|
```

Must enter a number

Enter a column 1 - 7: It's Y's turn.

win

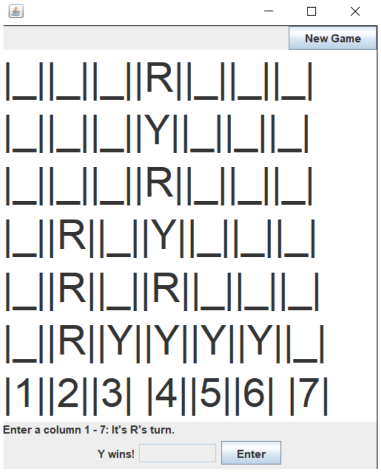
Then gameplay resumes once the error is closed and corrected



```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|_||R||_||R||_||_||_|
|_||R||Y||Y||Y||_||_|
|1||2||3| |4||5||6| |7|
```

Enter a column 1 - 7: It's Y's turn.

6

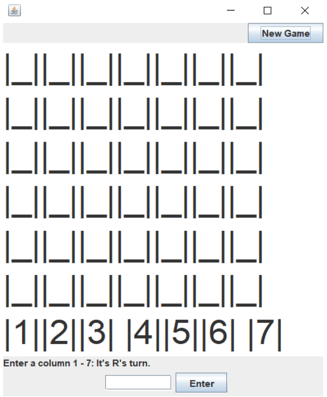


```
|_||_||_||R||_||_||_|
|_||_||_||Y||_||_||_|
|_||_||_||R||_||_||_|
|_||R||_||Y||_||_||_|
|_||R||_||R||_||_||_|
|_||R||Y||Y||Y||Y||_|
|1||2||3| |4||5||6| |7|
```

Enter a column 1 - 7: It's R's turn.

Y wins!

Once a player has gotten 4 tiles in a row they win and the board won't allow any more tiles to be played unless they click the New Game button. The game will then return to the start.

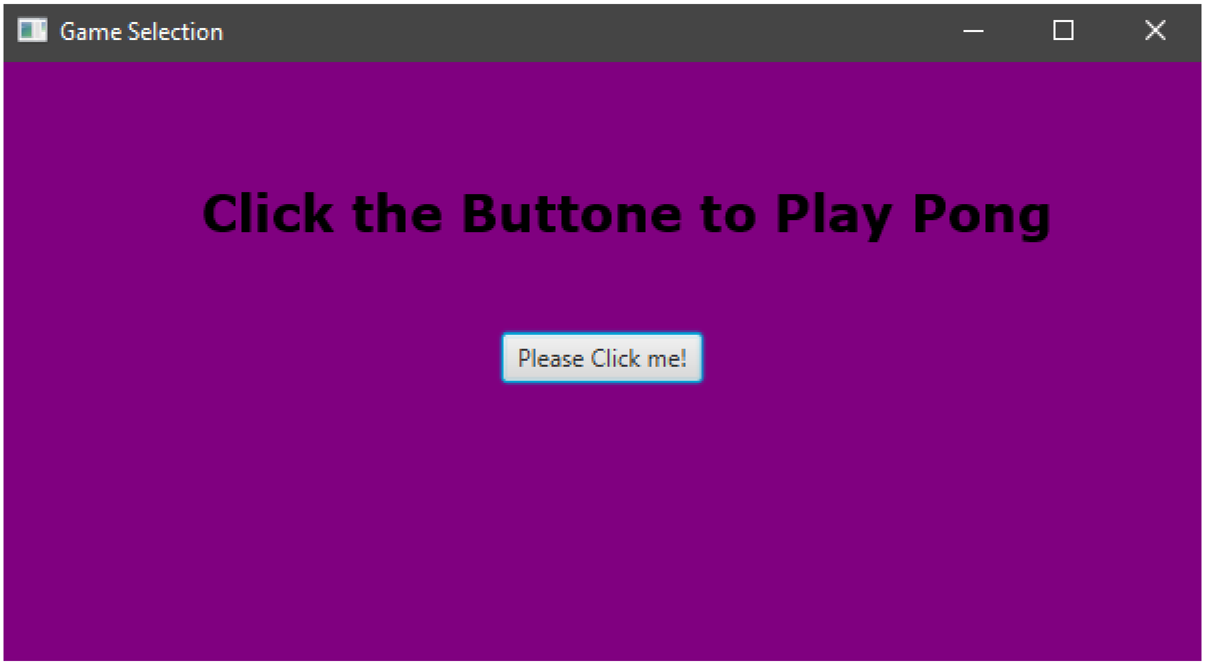


```
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|_||_||_||_||_||_||_|
|1||2||3| |4||5||6| |7|
```

Enter a column 1 - 7: It's R's turn.

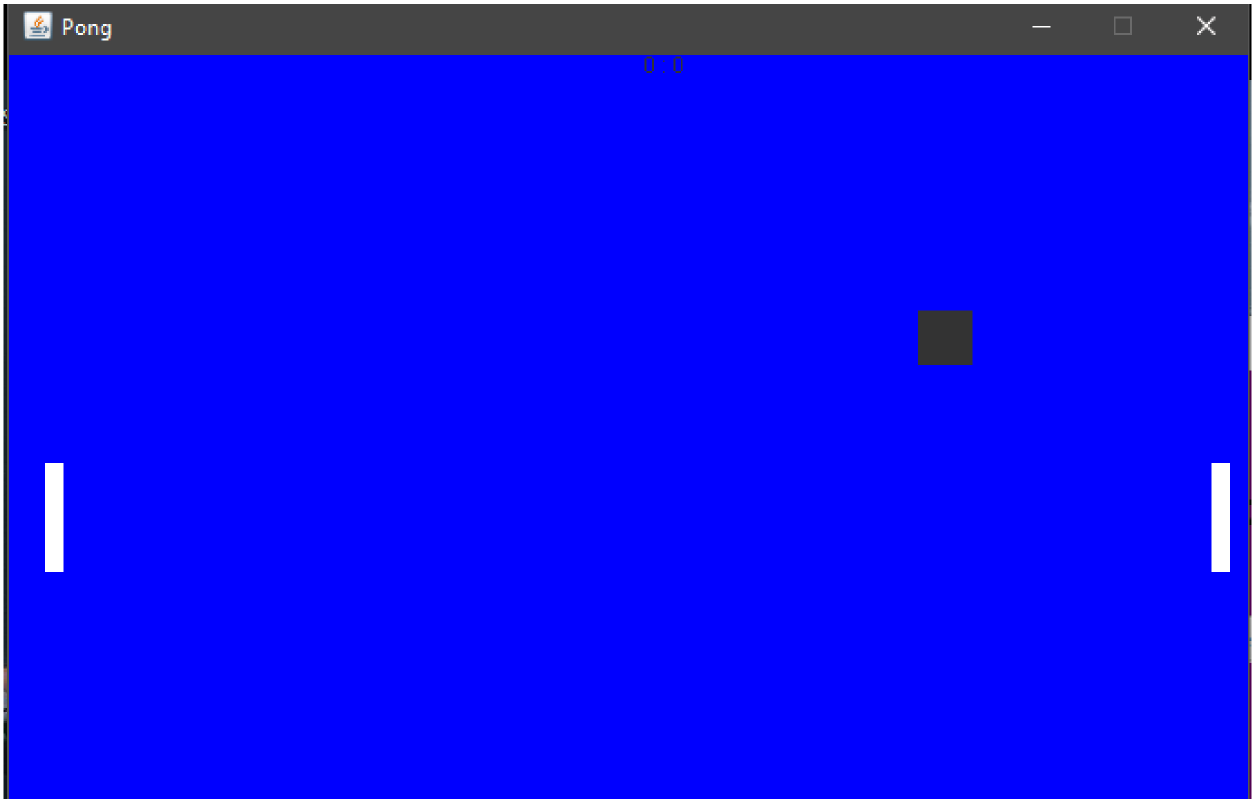
<u>Pong Use Case-</u>

The Proxy class opens up and prompts the user to click the button to be redirected to pong.

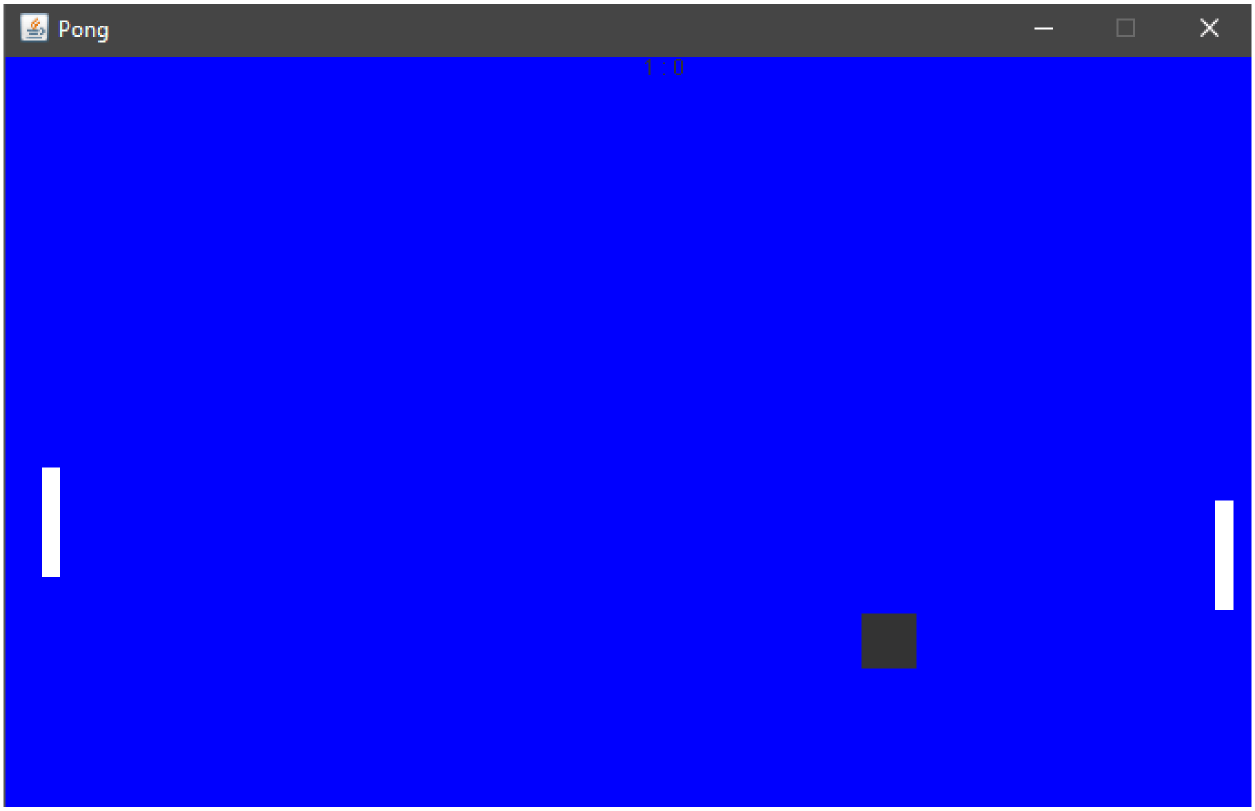


Click the Buttone to Play Pong

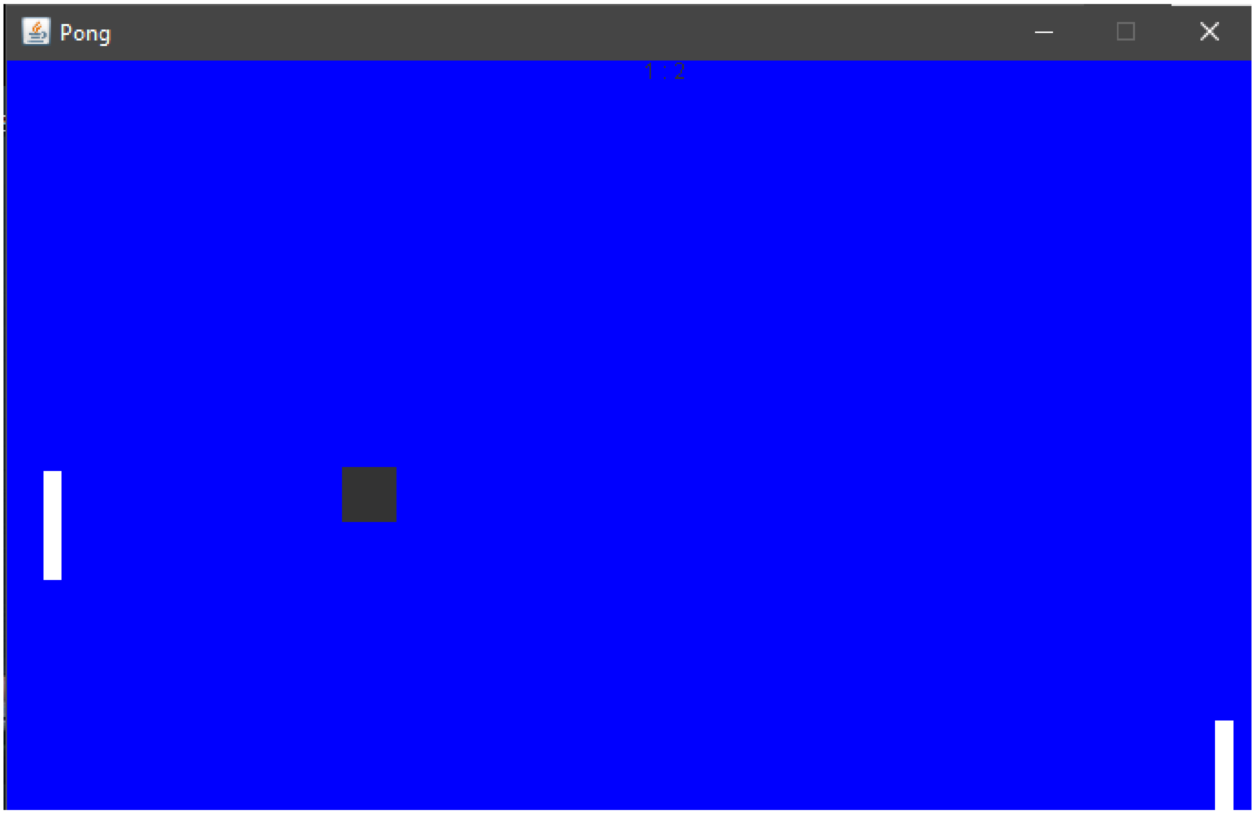
Two friends want to play pong, Player 1 is on the left and is controlled with the W key for up, and the S key for Down. Player 2 is on the right and is controlled with the UP Arrow Key for up and the Down Arrow Key for down.



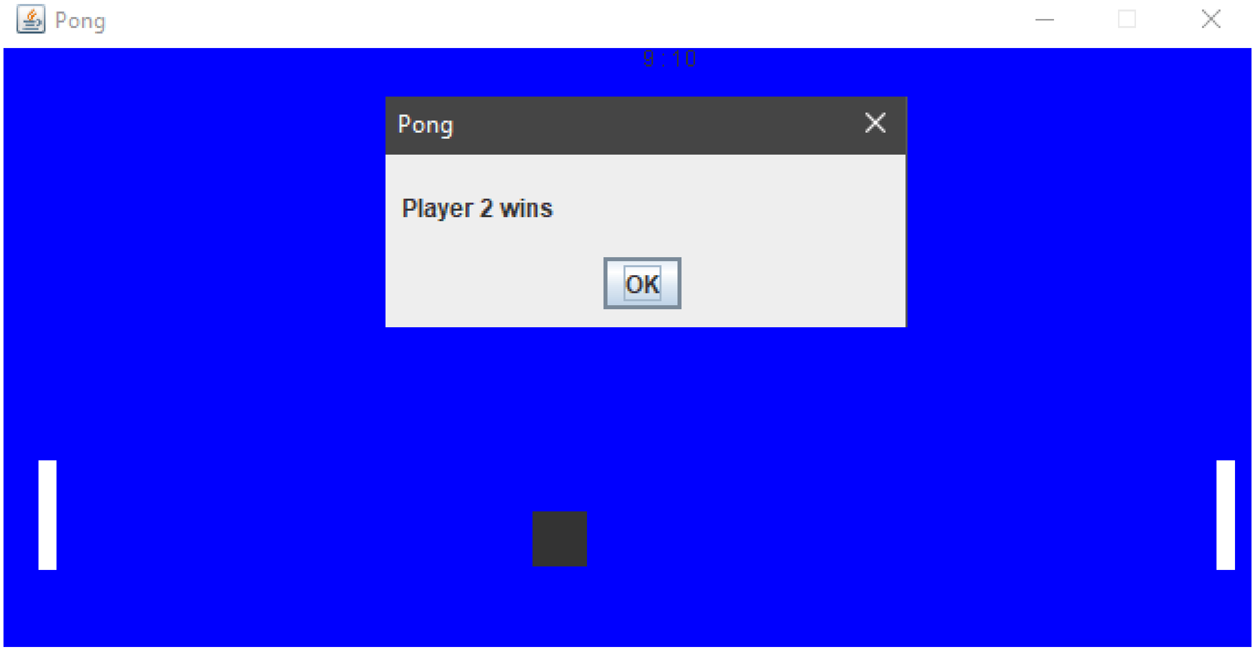
The players move the paddles on their side of the screen to prevent the “ball” from getting past them. If the ball passes a player it is a point for that opposing player.Player 1 Scores:



Player 2 Scores:



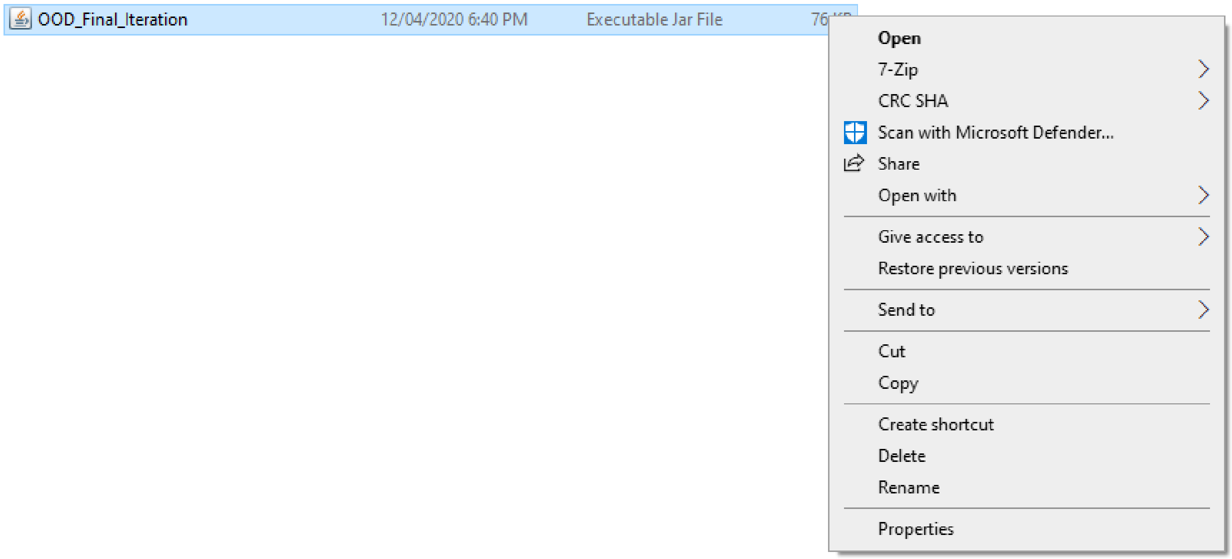
The game is continued to be played until a player reaches 10 points, the game then ends with the first to reach 10 points declared the winner.



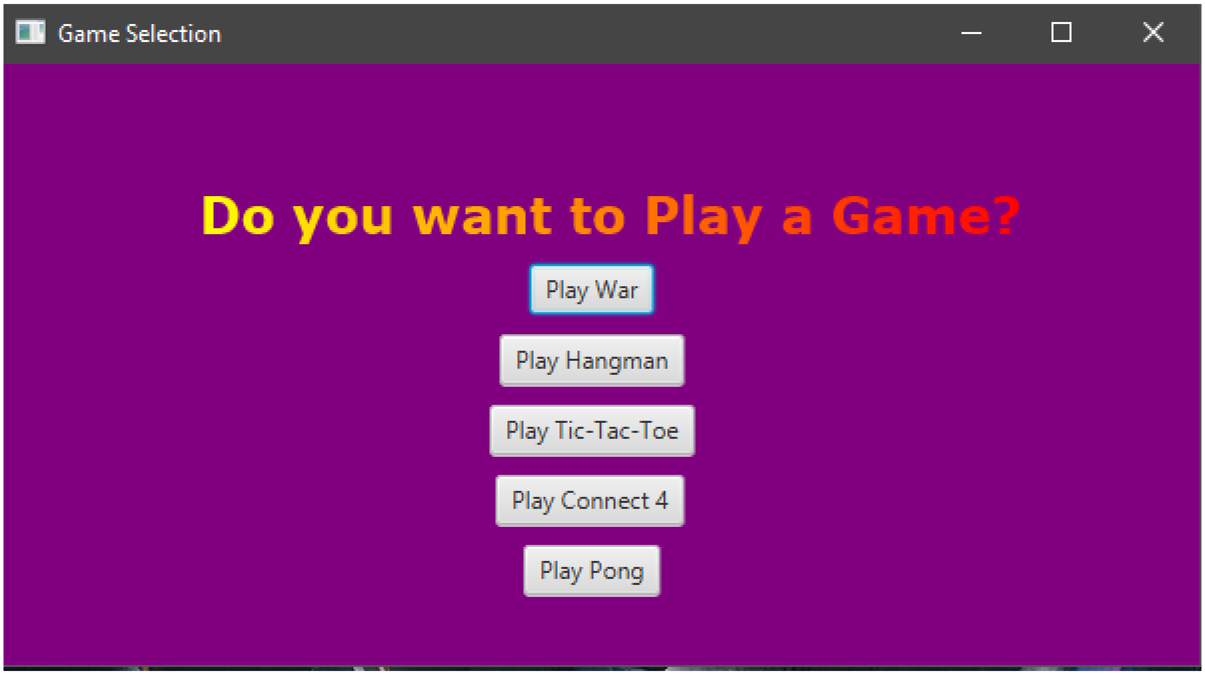
Player 2 wins

<u>Game Selector Use Case-</u>

User opens the Executable OOD_Final_Iteration.jar file



Two local users cannot decide what to play so they are given options in the forms of Buttons



Each Button will run a different java file as stated on the buttons. Each game is already defined with their own use cases above.

(Stephen)

Source Code(MVC Pattern/War Card Game):

Deck.java

```
package gameFactory;
//Class that deals with the deck(dealing, shuffling, etc)
public class Deck extends Pot{
    private final int LIMIT = 52;

    //method that deals the deck
    public void deal(Hand h) {
        int count = 0;

        while(count < h.getLimit()) {
            h.add(draw());
            count++;
        }
    }

    //method that deals with drawing from the deck
    public Card draw() {
        int rng = (int)(Math.random() * getNumCards());
        Card drawn = getCard(rng);

        remove(drawn);
        return drawn;
    }

    //method that gets a card from the deck
    private Card getCard(int x) {
        Card[] cards = toArray();
        return cards[x];
    }

    //shuffles the deck
    public void shuffle() {
        Card[] cards = toArray();
        int times = 3;

        while(times > 0) {
            for(int i = 0; i < cards.length; i++) {
                int rng = (int)(Math.random() * cards.length);
                Card temp = cards[i];
                cards[i] = cards[rng];
                cards[rng] = temp;
            }
            times--;
        }
        clear();
        for(int i = 0; i < cards.length; i++) {
            add(cards[i]);
        }
    }

    //get method for the variable LIMIT
    public int getLimit() {
        return LIMIT;
    }
}
```

Card.Java

```
package gameFactory;
import java.util.Arrays;
import java.util.List;

//Class that deals with the card objects
public class Card {

    //Global Variables that store the value and suit of the card
    private Suit suit;
    private Values value;

    //Constructor with no arguments that sets the value and suit of a card object to null
    public Card() {
        suit = null;
        value = null;
    }

    //Constructor with two arguments
    public Card(String s, String v) {
        this.suit = convertSuit(s);
        this.value = convertValue(v);
    }

    //Gets the suit of the card object
    public Suit getSuit() {
```

```
    return suit;
}

//Gets the value of the card object
public Values getValue() {
    return value;
}

//Sets both the value and suit of the card object
public void setCard(Card c) {
    suit = c.getSuit();
    value = c.getValue();
}

//Converts a string argument into a value for a card object
private Values convertValue(String value) {
    List<Values> a = Arrays.asList(Values.values());
    for(Values v : a) {
        if(v.toString().equals(value)) {
            return v;
        }
    }

    return null;
}

//Converts a string argument into a suit for a card object
private Suit convertSuit(String suit) {
    List<Suit> a = Arrays.asList(Suit.values());
    for(Suit s : a) {
        if(s.toString().equals(suit)) {
            return s;
        }
    }

    return null;
}

//returns a string that decribes the card object
public String toString() {
    return getValue() + " OF " + getSuit();
}

//returns an integer that describes the order of the selected suit
```

```
public int suitOrder() {
    switch(suit) {
    case CLUBS:
            return 4;
    case DIAMONDS:
            return 3;
    case HEARTS:
            return 2;
    case SPADES:
            return 1;
    default:
            return 0;
    }
}

//returns an integer that describes the order of the selected value
public int valueOrder() {
    switch (value) {
    case TWO:
            return 2;
    case THREE:
            return 3;
    case FOUR:
            return 4;
    case FIVE:
            return 5;
    case SIX:
            return 6;
    case SEVEN:
            return 7;
    case EIGHT:
            return 8;
    case NINE:
            return 9;
    case TEN:
            return 10;
    case JACK:
            return 11;
    case QUEEN:
            return 12;
    case KING:
            return 13;
    case ACE:
            return 14;
```

```
             default:
                   return 0;
             }
   }
}
```

Hand.java

```
package gameFactory;
//Class that deals with the players hands
public class Hand extends Pot{
   private final int LIMIT = 26;

   //method that draws a card from the players hand at random
   public Card draw() {
          int rng = (int)(Math.random() * getNumCards());
          Card drawn = getCard(rng);

          remove(drawn);
          return drawn;
   }

   //gets a card from an array
   private Card getCard(int i) {
          Card[] cards = toArray();
          return cards[i];
   }

   //get method for the limit variable
   public int getLimit() {
          return LIMIT;
   }
}
```

Player.java

```
package gameFactory;
public class Player {
   //global variables
   private String name;
   private int score;
   public Hand hand;

   //Constructor that has no arguments
   public Player() {
          name = null;
```

```
        hand = new Hand();
        score = 0;
    }

    //set methods
    public void setName(String name) {
        this.name = name;
    }

    //Get methods
    public String getName() {
        return name;
    }
    public Hand getHand() {
        return hand;
    }
    public int getNumOfCards() {
        return hand.getNumCards();
    }
    public int getScore() {
        return score;
    }

    //draws a card from players hand
    public Card draw() {
        return hand.draw();
    }

    //adds a card to the players hand
    public void addToHand(Card card) {
        hand.add(card);
    }

    //gives player a point
    public void givePoint() {
        score++;
    }

    //reduces a player points
    public void reducePoint() {
        score--;
    }
}
```

Pot.java

```
package gameFactory;
//Class that deals with the pot(discard pile)
public class Pot {
    //Global Variables
    private Card topCard;
    private Card bottomCard;

    private Node first;
    private Node last;

    private int numOfCards;

    //Constructor with no arguments that sets the global variable to null
    public Pot() {
            topCard = null;
            bottomCard = null;

            first = null;
            last = null;

            numOfCards = 0;
    }

    //methods that adds a card to the pot
    public void add(Card c) {
            Node newNode = new Node(c);

            //If the card is the first card in the pot
            if(first == null) {
                    first = newNode;
                    last = first;
                    topCard = newNode.getCard();
                    bottomCard = newNode.getCard();
            }else {
                    first.setPrevious(newNode);
                    newNode.setNext(first);
                    first = newNode;
                    topCard = newNode.getCard();
            }
            numOfCards++;
    }

    //method that adds a card to the bottom of the pot
```

```
public void addToBottom(Card c) {
    Node newNode = new Node(c);

    Node temp = last;
    last.setNext(newNode);
    last = newNode;
    last.setPrevious(temp);
    bottomCard = c;

    numOfCards++;
}

//removes a card form the pot
public void remove(Card c) {
    //Exception Dealing
    if(isEmpty()) {
        throw new NullPointerException();
    }else if(searching(c) == -1) {
        throw new IndexOutOfBoundsException();
    }

    int place = searching(c);

    if(place == 0) {
        if(numOfCards == 1) {
            first = null;
            last = null;
            topCard = null;
            bottomCard = null;
            numOfCards = 0;
        }else {
            first = first.getNext();
            first.setPrevious(null);
            topCard = first.getCard();
            numOfCards--;
        }
    }else if(place == (numOfCards - 1)) {
        last = last.getPrevious();
        last.setNext(null);
        bottomCard = last.getCard();
        numOfCards--;
    }else {
        int i = 0;
        Node nextNode = first;
```

```
                for(Node currentNode = first; nextNode != null; currentNode = nextNode) {
                    nextNode = currentNode.getNext();

                    if(i == place) {
                        currentNode.getPrevious().setNext(nextNode);
                        currentNode.getNext().setPrevious(currentNode.getPrevious());
                        numOfCards--;
                        break;
                    }
                    i++;
                }
        }
}

//method that searches for a card in the pot
public int searching(Card c) {
        Card[] cards = toArray();
        int index = -1;

        for(int i = 0; i < cards.length; i++) {
                if(cards[i].equals(c)) {
                    index = i;
                }
        }

        return index;
}

//shuffles the pot
public void shuffle() {
        Card[] cards = toArray();

        for(int i = 0; i < cards.length; i++) {
                int rng = (int)(Math.random() * cards.length);

                Card temp = cards[i];
                cards[i] = cards[rng];
                cards[rng] = temp;
        }

        clear();
        for(int i = 0; i < cards.length; i++) {
                add(cards[i]);
        }
    }

    //sorts the cards by value and suit
    public void sort() {
        int lowest = 1;
        int count = 0;

        Suit[] suits = {Suit.CLUBS, Suit.HEARTS, Suit.DIAMONDS, Suit.SPADES};
        Card[] cards = toArray(), array = new Card[numOfCards];

        for(int i = 0; i < suits.length; i++) {
            for(int j = lowest; j <= 14; j++) {
                for(int k = 0; k < cards.length; k++) {
                    if(cards[k].getSuit().equals(suits[i]) && cards[k].valueOrder() == j)
{
                        array[count] = cards[k];
                        count++;
                    }
                }
            }
        }

        clear();
        for(int i = array.length - 1; i >= 0; i--) {
            add(array[i]);
        }
    }

    //get methods
    public Card getTop() {
        return topCard;
    }
    public Card getBottom() {
        return bottomCard;
    }
    public int getNumCards() {
        return numOfCards;
    }

    //Checks if the pot is empty
    public boolean isEmpty() {
        return numOfCards == 0;
    }
```

```
//Clears the pot
public void clear() {
        if(isEmpty()) {
                throw new NullPointerException();
        }

        while(!isEmpty()) {
                remove(topCard);
        }
}

//method that returns and array of cards
public Card[] toArray() {
        Card[] cards = new Card[getNumCards()];

        int i = 0;
        Node n = first;

        while(n != null && i < getNumCards()) {
                Card c = n.getCard();
                n = n.getNext();
                cards[i] = c;
                i++;
        }

        return cards;
}

//Class that deals with the nodes in the pot(player 1 and player 2 pots)
private class Node{
        //global variables
        private Card c;
        private Node next;
        private Node previous;

        //Constructor with no arguments that sets the global variable to null
        public Node() {
                c = null;
                next = null;
                previous = null;
        }

        //Constructor with a card argument that sets the card global variable
        public Node(Card c) {
            this.c = c;
        }

        //set methods
        public void setCard(Card c) {
            this.c = c;
        }
        public void setNext(Node next) {
            this.next = next;
        }
        public void setPrevious(Node previous) {
            this.previous = previous;
        }

        //Get methods
        public Card getCard() {
            return c;
        }
        public Node getNext() {
            return next;
        }
        public Node getPrevious() {
            return previous;
        }
    }
}
```

Suit.java

```
package gameFactory;
//Enum that stores all four types of suits in a deck of cards
public enum Suit {
    HEARTS, SPADES, DIAMONDS, CLUBS
}
```

Values.java

```
package gameFactory;
//Enum that stores all 13 types of values in a deck of cards
public enum Values {
    ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING
}
```

GameGUI.java

```
package gameFactory;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.*;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;

//Game Engine and deals with the GUI aspects of the game
public class GameGUI extends Application{
    public static void main(String[] args) {
        Application.launch(args);
    }

    //Deals with creating the stage(For the MVC Design pattern this acts as the model
function)
    public void start(Stage primaryStage) throws Exception{
        Pane gamePane = new Pane();
        gamePane.setBackground(new Background(new
BackgroundFill(Color.LIGHTGRAY, null, null)));
        Scene gameScene = new Scene(gamePane, 600, 600);

        //Deals with the card outline boxes
        Rectangle left = new Rectangle(150,100,150,200);
        Rectangle right = new Rectangle(325,100,150,200);
        left.setStroke(Color.BLACK);
        left.setFill(Color.TRANSPARENT);
        right.setStroke(Color.BLACK);
        right.setFill(Color.TRANSPARENT);

        //Reads the information for the cards from a file named 'List of 52 Cards'
        Player one = new Player();
        Player two = new Player();
```

```
Deck deck = new Deck();
Card[] allCards = new Card[52];
File listOfCards = new File("List of 52 Cards");
try {
    Scanner in = new Scanner(listOfCards);

    while(in.hasNextLine()) {
        for(int i = 0; i < allCards.length; i++) {
            allCards[i] = new Card(in.next(), in.next());
        }
    }

    for(int i = allCards.length - 1; i >= 0; i--) {
        deck.add(allCards[i]);
    }

    //shuffles the deck
    deck.shuffle();
    in.close();
}catch(FileNotFoundException e) {
    System.out.println("File Not Found");
}


//Deals with the player 1 label
Label leftLabel = new Label("Player 1", left);
leftLabel.setContentDisplay(ContentDisplay.TOP);
leftLabel.setLayoutX(150);
leftLabel.setLayoutY(100);

//Deals with the player 2 label
Label rightLabel = new Label("Player 2", right);
rightLabel.setContentDisplay(ContentDisplay.TOP);
rightLabel.setLayoutX(325);
rightLabel.setLayoutY(100);

//adds the player 1 and 2 labels
gamePane.getChildren().addAll(left, leftLabel, right, rightLabel);

//gets the cards that are played and assigns them to card objects
Card oneDrawn = new Card();
Card twoDrawn = new Card();

//deals with the labels that indicates which players turn it is
```

```
Text onesTurn = new Text();
onesTurn.setLayoutX(250);
onesTurn.setLayoutY(50);
Text twosTurn = new Text();
twosTurn.setLayoutX(250);
twosTurn.setLayoutY(50);

//deals with the winning text
Text win = new Text();
win.setLayoutX(250);
win.setLayoutY(50);

//Deals with the draw and play buttons for each player
Button p1Draw = new Button("Draw");
Button p1Shuffle = new Button("Shuffle");
Button p2Draw = new Button("Draw");
Button p2Shuffle = new Button("Shuffle");

//deals with the play button
Button play = new Button("Play");

//Sets up the imageView for each of the players cards played
ImageView card1 = new ImageView();
card1.setLayoutX(125);
card1.setLayoutY(100);
ImageView card2 = new ImageView();
card2.setLayoutX(300);
card2.setLayoutY(100);


//deals with the drawing cards for player one
p1Draw.setLayoutX(150);
p1Draw.setLayoutY(350);
p1Draw.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent click) {
                if(one.getHand().isEmpty()) {
                        p1Draw.setDisable(true);
                }else {
                        if(card1.getImage() != null) {
                                gamePane.getChildren().remove(card1);
                        }
                        oneDrawn.setCard(one.draw());
                        card1.setImage(new Image(findImageURL(oneDrawn)));
                        gamePane.getChildren().remove(onesTurn);
```

```
            card1.setFitWidth(175);
            card1.setFitHeight(200);

            if(card2.getImage() != null) {
                gamePane.getChildren().remove(card2);
            }

            p2Draw.setDisable(false);
            p2Shuffle.setDisable(false);

            p1Draw.setDisable(true);
            p1Shuffle.setDisable(true);

            gamePane.getChildren().addAll(card1, twosTurn);


        }
    }
});


//Shuffles players 1 deck
p1Shuffle.setLayoutX(235);
p1Shuffle.setLayoutY(350);
p1Shuffle.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent click) {
        if(one.getHand().isEmpty()) {
            p1Shuffle.setDisable(true);
        }else {
            one.getHand().shuffle();
        }
    }
});

//deals with the drawing cards for player two
p2Draw.setLayoutX(325);
p2Draw.setLayoutY(350);
p2Draw.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent click) {
        if(two.getHand().isEmpty()) {
            p2Draw.setDisable(true);
        }else {
            twoDrawn.setCard(two.draw());
            card2.setImage(new Image(findImageURL(twoDrawn)));
```

```
                    gamePane.getChildren().remove(twosTurn);

                    card2.setFitWidth(175);
                    card2.setFitHeight(200);

                    p1Draw.setDisable(false);
                    p1Shuffle.setDisable(false);

                    p2Draw.setDisable(true);
                    p2Shuffle.setDisable(true);

                    gamePane.getChildren().addAll(card2,onesTurn);

                    compareCards(oneDrawn, twoDrawn, one, two, 1,
leftLabel, rightLabel);

                    if(one.getHand().isEmpty() || two.getHand().isEmpty()) {
                        if(one.getScore()>two.getScore()) {
                            win.setText(one.getName()+" is the
winner!");
                        }else {
                            win.setText(two.getName()+" is the
winner!");
                        }

                        gamePane.getChildren().removeAll(twosTurn,
onesTurn);
                        gamePane.getChildren().add(win);
                    }
                }
            }
        });

        //Shuffles players 2 deck
        p2Shuffle.setLayoutX(410);
        p2Shuffle.setLayoutY(350);
        p2Shuffle.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent click) {
                if(two.getHand().isEmpty()) {
                    p2Shuffle.setDisable(true);
                }else {
                    two.getHand().shuffle();
                }
            }
        });




        play.setLayoutX(280);
        play.setLayoutY(400);
        play.setOnAction(new EventHandler<ActionEvent>() {
             public void handle(ActionEvent click) {
                    gamePane.getChildren().remove(play);
                    gamePane.getChildren().addAll(p1Draw, p1Shuffle, p2Draw,
p2Shuffle, onesTurn);

                    p2Draw.setDisable(true);
                    p2Shuffle.setDisable(true);

                    deck.deal(one.getHand());
                    deck.deal(two.getHand());

                    one.getHand().shuffle();
                    two.getHand().shuffle();
            }
        });


        //deals with getting the players 2 name and and the text field that gets that name
        TextField name2 = new TextField();
        Label name2Label = new Label("What is player 2's name?", name2);
        name2Label.setContentDisplay(ContentDisplay.BOTTOM);
        name2Label.setLayoutX(220);
        name2Label.setLayoutY(400);
        name2.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent click) {
                    name2.setText(name2.getText());
                    two.setName(name2.getText());
                    twosTurn.setText("It's "+two.getName()+"'s turn!");
                    rightLabel.setText(name2.getText()+": "+two.getScore());
                    gamePane.getChildren().removeAll(name2, name2Label);
                    gamePane.getChildren().add(play);
            }
        });

        //deals with getting the players 1 name and and the text field that gets that name
        TextField name1 = new TextField();
        Label name1Label = new Label("What is player 1's name?", name1);
        name1Label.setContentDisplay(ContentDisplay.BOTTOM);
        name1Label.setLayoutX(220);
        name1Label.setLayoutY(400);
        name1.setOnAction(new EventHandler<ActionEvent>() {
            @Override public void handle(ActionEvent click) {
                name1.setText(name1.getText());
                one.setName(name1.getText());
                onesTurn.setText("It's "+one.getName()+"'s turn!");
                leftLabel.setText(name1.getText()+": "+one.getScore());
                gamePane.getChildren().removeAll(name1, name1Label);
                gamePane.getChildren().addAll(name2, name2Label);
            }
        });

        //adds the player 1 name and its label
        gamePane.getChildren().addAll(name1, name1Label);

        //sets up and shows the stage
        primaryStage.setTitle("The Game of War");
        primaryStage.setScene(gameScene);
        primaryStage.show();
    }

    //This deals with all the game logic(For MVC Design it represents the controller function)
    public static void compareCards(Card a, Card b, Player p1, Player p2, int war, Label l1,
Label l2) {
        //variable for the war event
        int warScore = war;

        //if player 1's card is higher
        if(a.valueOrder()>b.valueOrder()) {
            for(int i=0;i<warScore;i++) {
                p1.givePoint();
            }
            updateScore(l1, p1);
        //if players two card is higher
        }else if(b.valueOrder()>a.valueOrder()) {
            for(int i=0;i<warScore;i++) {
                p2.givePoint();
            }
            updateScore(l2, p2);
        //if the two cards are the same and a war event happens(which is dealt with here)
        }else if(a.valueOrder()==b.valueOrder()) {
            int i=0;
            Card c = new Card();

            //draws four cards from each player hand and compares the fourth card to
decide who wins the war(recursive function)
            while(!p1.getHand().isEmpty() && i<=3) {
                c.setCard(p1.draw());
                i++;
            }
            int j=0;
            Card d = new Card();
            while(!p2.getHand().isEmpty() && j<=3) {
                d.setCard(p2.draw());
                j++;
            }
            warScore++;
            compareCards(c, d, p1, p2, warScore, l1, l2);
        }
    }

    //updates the players score and its label
    public static void updateScore(Label l, Player p) {
        l.setText(p.getName() + ": " + p.getScore());
    }

    public static ImageView createImage(String url, double x, double y) {
        Image image = new Image(url);

        ImageView image2 = new ImageView(image);

        image2.setX(x);
        image2.setY(y);

        return image2;
    }

    //Gets the corresponding cards images using a switch statement. Cards images are
stored in the source folder called Images.
    public static String findImageURL(Card card) {
        switch(card.getValue()) {
        case TWO:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "2C.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "2H.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "2D.png";
            }else {
                return "2S.png";
            }
        case THREE:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "3C.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "3H.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "3D.png";
            }else {
                return "3S.png";
            }
        case FOUR:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "4C.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "4H.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "4D.png";
            }else {
                return "4S.png";
            }
        case FIVE:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "5C.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "5H.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "5D.png";
            }else {
                return "5S.png";
            }
        case SIX:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "6C.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "6H.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "6D.png";
            }else {
```

```
        return "6S.png";
    }
case SEVEN:
    if(card.getSuit().equals(Suit.CLUBS)) {
        return "7C.png";
    }else if(card.getSuit().equals(Suit.HEARTS)) {
        return "7H.png";
    }else if(card.getSuit().equals(Suit.DIAMONDS)) {
        return "7D.png";
    }else {
        return "7S.png";
    }
case EIGHT:
    if(card.getSuit().equals(Suit.CLUBS)) {
        return "8C.png";
    }else if(card.getSuit().equals(Suit.HEARTS)) {
        return "8H.png";
    }else if(card.getSuit().equals(Suit.DIAMONDS)) {
        return "8D.png";
    }else {
        return "8S.png";
    }
case NINE:
    if(card.getSuit().equals(Suit.CLUBS)) {
        return "9C.png";
    }else if(card.getSuit().equals(Suit.HEARTS)) {
        return "9H.png";
    }else if(card.getSuit().equals(Suit.DIAMONDS)) {
        return "9D.png";
    }else {
        return "9S.png";
    }
case TEN:
    if(card.getSuit().equals(Suit.CLUBS)) {
        return "10C.png";
    }else if(card.getSuit().equals(Suit.HEARTS)) {
        return "10H.png";
    }else if(card.getSuit().equals(Suit.DIAMONDS)) {
        return "10D.png";
    }else {
        return "10S.png";
    }
case JACK:
    if(card.getSuit().equals(Suit.CLUBS)) {
```

```
                return "JC.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "JH.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "JD.png";
            }else {
                return "JS.png";
            }
        case QUEEN:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "QC.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "QH.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "QD.png";
            }else {
                return "QS.png";
            }
        case KING:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "KC.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "KH.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "KD.png";
            }else {
                return "KS.png";
            }
        case ACE:
            if(card.getSuit().equals(Suit.CLUBS)) {
                return "AC.png";
            }else if(card.getSuit().equals(Suit.HEARTS)) {
                return "AH.png";
            }else if(card.getSuit().equals(Suit.DIAMONDS)) {
                return "AD.png";
            }else {
                return "AS.png";
            }
        default:
            return null;
        }
    }
}
```

(Torin)

Code for TicTacToe using Command Pattern

Command.java:

```
package gameFactory;

public interface Command {
	public void execute();
	public void undo();
}
```

CommandState.java:

```
package gameFactory;

public class CommandState {
	private Command previousCommand;
	public CommandState() {}
	public void executeCommand(Command aCommand) {
		aCommand.execute();
		previousCommand = aCommand;
	}
	public boolean canUndo() {
		return previousCommand != null;
	}
	public void undo() {
			previousCommand.undo();
			previousCommand = null;
	}
	public boolean isUndoAvailable() {
		if(previousCommand != null) {
			return true;
		}
		else {
			return false;
		}
	}
}
```

TicTacToe.java:

```
package gameFactory;

public class TicTacToe {
	private boolean pXTurn;
	private int[][] spaces;
	private CommandState commandState;
	//Constructor for model
```

```
    public TicTacToe() {
            spaces = new int[3][3];
            pXTurn = true;
            commandState = new CommandState();
    }
    //Helper methods
    public boolean ispXTurn() {
            return pXTurn;
    }
    public boolean ispOTurn() {
            return !pXTurn;
    }
    public boolean isEmpty(int r, int c) {
            return spaces[r][c] == 0;
    }
    public boolean isX(int r, int c) {
            return spaces[r][c] == 1;
    }
    public boolean isO(int r, int c) {
            return spaces[r][c] == 2;
    }
    public boolean isGameOver() {
            if(xWon() || oWon()) {
                    return true;
            }
            for(int row = 0; row < 3; row++) {
                    for(int col = 0; col < 3; col++) {
                            if(spaces[row][col] == 0) return false;
                    }
            }
            //Tie game
            return true;
    }
    public boolean xWon() {
            //Check rows
if(spaces[0][0] == 1 && spaces[0][1] == 1 && spaces[0][2] == 1) return true;
if(spaces[1][0] == 1 && spaces[1][1] == 1 && spaces[1][2] == 1) return true;
if(spaces[2][0] == 1 && spaces[2][1] == 1 && spaces[2][2] == 1) return true;
//Check columns
if(spaces[0][0] == 1 && spaces[1][0] == 1 && spaces[2][0] == 1) return true;
if(spaces[0][1] == 1 && spaces[1][1] == 1 && spaces[2][1] == 1) return true;
if(spaces[0][2] == 1 && spaces[1][2] == 1 && spaces[2][2] == 1) return true;
//Check diagonals
if(spaces[0][0] == 1 && spaces[1][1] == 1 && spaces[2][2] == 1) return true;
if(spaces[0][2] == 1 && spaces[1][1] == 1 && spaces[2][0] == 1) return true;

return false;
   }
   public boolean oWon() {
             //Check rows
if(spaces[0][0] == 2 && spaces[0][1] == 2 && spaces[0][2] == 2) return true;
if(spaces[1][0] == 2 && spaces[1][1] == 2 && spaces[1][2] == 2) return true;
if(spaces[2][0] == 2 && spaces[2][1] == 2 && spaces[2][2] == 2) return true;
//Check columns
if(spaces[0][0] == 2 && spaces[1][0] == 2 && spaces[2][0] == 2) return true;
if(spaces[0][1] == 2 && spaces[1][1] == 2 && spaces[2][1] == 2) return true;
if(spaces[0][2] == 2 && spaces[1][2] == 2 && spaces[2][2] == 2) return true;
//Check diagonals
if(spaces[0][0] == 2 && spaces[1][1] == 2 && spaces[2][2] == 2) return true;
if(spaces[0][2] == 2 && spaces[1][1] == 2 && spaces[2][0] == 2) return true;
//Otherwise, there is no line
return false;
   }
   public void placeX(int r, int c) {
             commandState.executeCommand(new XCommand(this, r, c));
   }
   public void placeO(int r, int c) {
             commandState.executeCommand(new OCommand(this, r, c));
   }
   //XCOMMAND CLASS
   public class XCommand implements Command{
             private TicTacToe model;
             private int prev;
             private boolean prevTurn;
             private int row;
             private int col;

             private XCommand(TicTacToe model, int row, int col) {
                       this.model = model;
                       this.col = col;
                       this.row = row;
                       //copy previous value
                       this.prev = model.spaces[row][col];
                       this.prevTurn = model.pXTurn;
             }

             @Override
             public void execute() {
```

```
        model.spaces[row][col] = 1;
        model.pXTurn = false;
    }

    @Override
    public void undo() {
        model.spaces[row][col] = prev;
        model.pXTurn = prevTurn;
    }
}
//OCOMMAND CLASS
public class OCommand implements Command{
    private TicTacToe model;
    private int prev;
    private boolean prevTurn;
    private int row;
    private int col;

    private OCommand(TicTacToe model, int row, int col) {
        this.model = model;
        this.col = col;
        this.row = row;
        //copy previous value
        this.prev = model.spaces[row][col];
        this.prevTurn = model.pXTurn;
    }
    @Override
    public void execute() {
        model.spaces[row][col] = 2;
        model.pXTurn = true;
    }

    @Override
    public void undo() {
        model.spaces[row][col] = prev;
        model.pXTurn = prevTurn;
    }

}
//UNDO
public boolean canUndo() {
    return commandState.isUndoAvailable();
}
public void undo() {
        commandState.undo();
    }

}
```

<u>TicTacToeInterpreter.java</u>:

```
package gameFactory;

public class TicTacToeInterpreter {
    private TicTacToe model;

    public TicTacToeInterpreter() {
        model = new TicTacToe();
        startGame();
    }

    private void startGame() {
        System.out.println("Tic-Tac-Toe Game");
        System.out.println("----------------");
        System.out.println();
        listCommands();
        System.out.println();
        showBoard();
    }

    private void listCommands() {
        System.out.println("x row col - Place X at row and column");
        System.out.println("o row col - Place O at row and column");
        System.out.println("u - Undo the last move");
        System.out.println();
    }
    public void parseCommand(String command) throws Exception {
            parseGameplayCommand(command);
    }
    private void parseGameplayCommand(String command) throws Exception {
        char userInput = command.charAt(0);
        String[] tokens = command.split(" ");

        switch(userInput) {
        case 'x':
            if(!model.ispXTurn()) {
                System.out.println("It isn't the X player's turn.");
                System.out.println();
                return;
            }
            try {
                int row = Integer.parseInt(tokens[1]);
                int col = Integer.parseInt(tokens[2]);
                if(!model.isEmpty(row, col)) {
                    System.out.println("Space is already filled.");
                    System.out.println();
                    return;
                }
                model.placeX(row, col);
                if(model.isGameOver()) {
                    endGame();
                } else {
                    showBoard();
                }
            } catch (Exception e) {
                System.out.println("Invalid command input.");
                System.out.println();
            }
            break;
        case 'o':
            if(!model.ispOTurn()) {
                System.out.println("It isn't the O player's turn.");
                System.out.println();
                return;
            }
            try {
                int row = Integer.parseInt(tokens[1]);
                int col = Integer.parseInt(tokens[2]);
                if(!model.isEmpty(row, col)) {
                    System.out.println("Space is already filled.");
                    System.out.println();
                    return;
                }
                model.placeO(row, col);
                showBoard();
            } catch (Exception e) {
                System.out.println("Invalid command input.");
                System.out.println();
            }
            break;
        case 'u':
            if(model.canUndo()) {
                model.undo();
                System.out.println("The most recent move has been undone.");
                System.out.println();
                showBoard();
            } else {
                System.out.println("Can't undo any more moves.");
                System.out.println();
            }
            break;
        default:
            System.out.println("Invalid command.");
            listCommands();
            System.out.println();
        }
    }


    private void showBoard() {
        System.out.printf( " %s | %s | %s%n", getCharacter(0, 0), getCharacter(0, 1),
getCharacter(0, 2));
        System.out.println("-----------");
        System.out.printf( " %s | %s | %s%n", getCharacter(1, 0), getCharacter(1, 1),
getCharacter(1, 2));
        System.out.println("-----------");
        System.out.printf( " %s | %s | %s%n", getCharacter(2, 0), getCharacter(2, 1),
getCharacter(2, 2));
        System.out.println();


        if(model.ispXTurn()) {
            System.out.println("It is currently the X player's turn. Enter a command:");
            System.out.println();
        } else {
            System.out.println("It is currently the O player's turn. Enter a command:");
            System.out.println();
        }
    }

    private String getCharacter(int row, int col) {
        if(model.isEmpty(row, col)) {
            return " ";
        } else if(model.isX(row, col)) {
            return "X";
        }
        return "O";
    }


    private void endGame() {
        if(model.xWon()) {
            System.out.println("The X player won. Restart the program to play
again.");
            System.out.println();
        } else if(model.oWon()) {
            System.out.println("The O player won. Restart the program to play
again.");
            System.out.println();
        } else {
            System.out.println("Tie Game. Restart the program to play again.");
            System.out.println();
        }
        System.exit(0);
    }



}
```

<u>TicTacToeMain.java</u>:

```
package gameFactory;

import java.util.Scanner;

public class TicTacToeMain {

    public static void main(String[] args) throws Exception {
        TicTacToeInterpreter interpreter = new TicTacToeInterpreter();
        Scanner input = new Scanner(System.in);
        String str;
      while ((str = input.nextLine()) != null) {
        interpreter.parseCommand(str);
        }
    }

}
```

```
    }
```

(Carson)

Code for Hangman using State Pattern:

HangmanPhase.java:

```
package gameFactory;
public class HangmanPhase {
    GameData data = new GameData();
    final static int START = 0;
    final static int CHECK = 1;
    final static int GUESS = 2;
    final static int WIN = 3;

    int state = WIN;


    public HangmanPhase() {
        state = START;
    }

    public void takeTurn() {
        if(state == START) {
            GameData.playGame();
            state = GUESS;
        }
        else if(state == GUESS) {
            GameData.guess();
            state = CHECK;
            takeTurn();
        }
        else if(state == CHECK) {
            if(GameData.check()) {
                state = WIN;
                takeTurn();
            }
            else {
                state = GUESS;
            }
        }
        else if(state == WIN) {
            System.out.println("You guessed the word! You win!");
            System.exit(0);
        }
    }
}
```

GameData.java:

```
package gameFactory;
import java.util.ArrayList;
import java.util.Random;

public class GameData {

    // Constructor
    public GameData() {

    }
    // Possible words to guess
    public static String[] words = {
            "java", "python", "compiler", "architecture","queue",
            "software", "graph", "array", "stack", "heap",
            "exception", "javascript", "interface", "abstract", "inheritance"
    };

    public static Random RANDOM = new Random();
    public static int maxWrong = 8;
    public static int curWrong = 0;
    public static String targetWord;
    public static char[] curWordProgress;
    public static ArrayList<String> guesses = new ArrayList<String>();
    public static String curGuess = "";

    public static void playGame() {
        System.out.println("Welcome to Computer Science Hangman!");
        targetWord = determineWord();
        curWordProgress = new char[targetWord.length()];

        for (int i = 0; i < curWordProgress.length; i++) {
            curWordProgress[i] = '_';
        }
    }

    private static String determineWord() {
        return words[RANDOM.nextInt(words.length)];
    }

    // Sets curGuess to the letter passed in, so it can then be used
    // in the guess function
    public void setGuess(String str) {
        if (str.length() > 1) {
            curGuess = str.substring(0, 1);
        } else
            curGuess = str;
    }

    public String guessedContent() {
        StringBuilder builder = new StringBuilder();

        for (int i = 0; i < curWordProgress.length; i++) {
             builder.append(curWordProgress[i]);

             if (i < curWordProgress.length - 1)
                 builder.append(" ");
         }
        return builder.toString();
    }

    public static void guess() {
        if (!guesses.contains(curGuess)) {
            if (targetWord.contains(curGuess)) {
                int index = targetWord.indexOf(curGuess);

              while (index >= 0) {
                int curWordProgress[index] = curGuess.charAt(0);
                index = targetWord.indexOf(curGuess, index + 1);
              }
            }
            else {
                curWrong++;
                System.out.println("Incorrect! You have " + (maxWrong -
curWrong) + " guesses remaining");
            }

            guesses.add(curGuess);
     }
    }

    public static boolean check() {
        return targetWord.contentEquals(new String(curWordProgress));
    }
}
```

HangmanTestDrive.java:

```
package gameFactory;
import java.util.Scanner;

public class HangmanTestDrive {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        HangmanPhase game = new HangmanPhase();
        GameData data = new GameData();

        game.takeTurn();
        System.out.println("Enter a lower case letter: ");

        while(GameData.curWrong < GameData.maxWrong) {
            data.setGuess(input.next());
            game.takeTurn();

          System.out.println("\n" + data.guessedContent());
        }

        System.out.println("Out of guesses. Game over!");
        System.out.println("The word was " + GameData.targetWord);
    }

}
```

(Caleb)

Code for Connect 4 Game

<u>C4 Java Class</u>

```
package gameFactory;
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Iterator;
import javax.swing.*;
public class C4 extends JFrame implements Iterator<Object>{
        protected JLabel error= new JLabel("");
        protected JLabel win= new JLabel(" ");
        protected JLabel directions= new JLabel("Enter a column 1 - 7: It's R's turn.");
        protected JTextField userIn = new JTextField(8);
        protected JButton enter = new JButton("Enter");
        protected JButton reset = new JButton("New Game");
        protected String[][] board;
        protected String gb="";
        protected int turn=0;
        protected int player=1;
        protected Font f =new Font("Cam", Font.PLAIN,44);
        protected JTextArea visual =new JTextArea(gb);
        protected int x,y;
        protected int c=0;
        protected int h=0;
        protected int v=0;

        //creates blank board
        public void newBoard() {
                board =new String[6][7];
                for(int x=0; x<board.length; x++) {
                        for(int y=0; y<board[0].length; y++) {
                                board[x][y]="|_|";
                        }
                }
        }
        //Set gb to the current board for display
                public void showBoard() {
                        gb=next();
                        gb+="|1||2||3| |4||5||6| |7|";
                        h=0;
                }
                //Sets layout and activates buttons
                C4(){
                        newBoard();
                        showBoard();
                        JPanel gamePanel = new JPanel();
                        gamePanel.setLayout(new BorderLayout());
                        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                        this.setSize(420, 520);
                        JPanel boardPanel =new JPanel();
                        boardPanel.setLayout(new BorderLayout());
                        visual.setFont(f);
                        visual.setEditable(false);
                        visual.setText(gb);
                        boardPanel.add(visual,"Center");
                        JPanel resetPanel = new JPanel();
                        resetPanel.setLayout(new BorderLayout());
                        resetPanel.add(reset,"East");
                        JPanel play= new JPanel();
                        play.setLayout(new BorderLayout());
                        play.add(directions,"North");
                        play.add(error,"South");
                        JPanel input=new JPanel();
                        input.add(win);
                        input.add(userIn);
                        input.add(enter);
                        play.add(input,"Center");
                        gamePanel.add(resetPanel,"North");
                        gamePanel.add(boardPanel,"Center");
                        gamePanel.add(play,"South");
                        enter.addActionListener(new ActionListener() {
                 public void actionPerformed(ActionEvent e) {
                         try {
                      int msg = Integer.parseInt(userIn.getText());
                      if((msg<8 && msg>0)&&c%2==0) {
                        x=msg-1;
                        c++;
                        directions.setText("Enter a column 1 - 7: It's Y's turn.");
                        userIn.setText("");
                        error.setText("");
                        marks(x);
                        showBoard();
                        win();
                      visual.setText(gb);
                    }
                    else if(msg<8 && msg>0&&c%2!=0) {
                      x=msg-1;
                      c++;
                      directions.setText("Enter a column 1 - 7: It's R's turn.");
                      userIn.setText("");
                      error.setText("");
                      marks(x);
                      showBoard();
                      win();
                      visual.setText(gb);
                    }
                    else {
                      error.setText("Must enter a number 1 - 7!");
                      userIn.setText("");
                    }

                       }catch(NumberFormatException nf) {
                              displayErrorMessage("Must enter a number");
                       }
                }});
                      userIn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                       try {
                    int msg = Integer.parseInt(userIn.getText());
                    if((msg<8 && msg>0)&&turn%2==0) {
                      x=msg-1;

                      directions.setText("Enter a column 1 - 7: \tIt's Y's turn.");
                      userIn.setText("");
                      error.setText("");
                      marks(x);
                      showBoard();
                      win();
                      visual.setText(gb);
                    }
                    else if(msg<8 && msg>0&&turn%2!=0) {
                      x=msg-1;
                      directions.setText("Enter a column 1 - 7: \tIt's R's turn.");
                      userIn.setText("");
                      error.setText("");
                      marks(x);
                      showBoard();
```

```
        win();
        visual.setText(gb);
       }
       else {
        error.setText("Must enter a number 1 - 7!");
        userIn.setText("");
       }
         }catch(NumberFormatException nf) {
             displayErrorMessage("Must enter a number");
         }
    }});
        reset.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
       reset();
   }});
        this.add(gamePanel);
    }
    //puts pieces on board
public void marks(int col) {
    if(turn%2==0) {
        for(int x=5; x>=0; x--) {
            if(board[x][col].equalsIgnoreCase("|_|")) {
                player=2;
                board[x][col]="|R|";
                turn++;
                gb="";
                error.setText("");
                break;
            }
            if(x==0) {
                error.setText("You can not go there.");
                directions.setText("Enter a column 1 - 7: \tIt's R's turn.");
            }
        }
    }
    else {
        for(int x=5; x>=0; x--) {
            if(board[x][col].equalsIgnoreCase("|_|")) {
                player=1;
                board[x][col]="|Y|";
                turn++;
                gb="";
                error.setText("");
                break;
            }
            if(x==0) {
                error.setText("You can not go there.");
                directions.setText("Enter a column 1 - 7: \tIt's Y's turn.");
            }
        }
    }
}

public void win() {
    //check for verical wins
    String red ="|R||R||R||R|";
    String yellow="|Y||Y||Y||Y|";
    String check="";
    for(int y=0; y<7; y++) {
        check="";
        for(int x=5; x>=0; x--) {
            check+=board[x][y];
            if(check.contains(red)) {
                win.setText("R wins!");
                userIn.setEditable(false);
            }
            if(check.contains(yellow)) {
                win.setText("Y wins!");
                userIn.setEditable(false);
            }

        }
    }
    //checks for horizontal victories
    for(int x=5; x>=0; x--) {
        check="";
        for(int y=0; y<board[0].length; y++) {
            check+=board[x][y];
            if(check.contains(red)) {
                win.setText("R wins!");
                userIn.setEditable(false);
            }
            if(check.contains(yellow)) {
                win.setText("Y wins!");
                userIn.setEditable(false);
            }
        }
    }
```

```
        //Checks for an up to the right diagonal victory
        for(int x=3; x<=5; x++) {
            for(int y=0; y<4; y++) {
                if(board[x][y].equals(board[x-1][y+1])&&board[x][y].equals(board[x-
2][y+2])
                        &&board[x][y].equals(board[x-
3][y+3])&&board[x][y].equals("|R|")) {
                    win.setText("R wins!");
                    userIn.setEditable(false);
                }
                if(board[x][y].equals(board[x-1][y+1])&&board[x][y].equals(board[x-
2][y+2])
                        &&board[x][y].equals(board[x-
3][y+3])&&board[x][y].equals("|Y|")) {
                    win.setText("Y wins!");
                    userIn.setEditable(false);
                }
            }

        }
        //Checks for down to the right diagonal wins
        for(int x=0; x<=2; x++) {
            for(int y=0; y<4; y++) {

    if(board[x][y].equals(board[x+1][y+1])&&board[x][y].equals(board[x+2][y+2])

    &&board[x][y].equals(board[x+3][y+3])&&board[x][y].equals("|R|")) {
                    win.setText("R wins!");
                    userIn.setEditable(false);
                }

    if(board[x][y].equals(board[x+1][y+1])&&board[x][y].equals(board[x+2][y+2])

    &&board[x][y].equals(board[x+3][y+3])&&board[x][y].equals("|Y|")) {
                    win.setText("Y wins!");
                    userIn.setEditable(false);
                }
            }
        }
        //When nobody wins
        if(turn==42) {
            System.out.println("It's a draw!");
            userIn.setEditable(false);
        }
```

```
	}

	public boolean hasNext() {
		if(h<6 && v<7)
			return true;
		return false;
	}

	public String next() {
		String s="";
		while(hasNext()) {
		s+=board[h][v];
		v++;
		if(v==7) {
			v=0;
			h++;
			s+="\n";
		}
		}
		return s;
	}
	//resets the game
	void reset() {
		x=0;
		player=1;
		c=0;
		turn=0;
		newBoard();
		showBoard();
		directions.setText("Enter a column 1 - 7: It's R's turn");
		visual.setText(gb);
		userIn.setEditable(true);
		error.setText("");
		win.setText("");
	}
	//pop up window when a number isn't entered
	void displayErrorMessage(String errorMessage){
		JOptionPane.showMessageDialog(this, errorMessage);
	}

}
```

(Ezekiel)
Game Selector GUI using Decorator Class
GameSelectorUI.java

```
package gameFactory;
import java.io.IOException;
import java.io.InputStream;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.paint.Color;
import javafx.scene.paint.CycleMethod;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;


public class GameSelectorUI extends Application {
  public void start(Stage stage) {

    //Button that Starts Stephen's game of War that uses the MVC Design Pattern
    Button War= new Button("Play War");
    War.setTranslateX(265);
    War.setTranslateY(100);
    War.setOnAction(new EventHandler<ActionEvent>(){

                @Override
                public void handle(ActionEvent e) {
                        stage.close();
                        RunStephensGame.StartGame(stage);;

                }

    });

    //Button that starts Carson's game of Hangman that uses the state Design Pattern
    //TextArea gametext = new TextArea();
```

```
Button Hangman= new Button("Play Hangman");
Hangman.setTranslateX(250);
Hangman.setTranslateY(135);
Hangman.setOnAction(new EventHandler<ActionEvent>(){

            @Override
            public void handle(ActionEvent e) {

                    try {
                            RunsCarsonsGame.StartGame(stage);

                    } catch (Exception e1) {
                            // TODO Auto-generated catch block
                            e1.printStackTrace();
                    }
            }

  });

//Button that starts Torin's game of TicTacToe Using the Command Pattern
Button TicTacToe= new Button("Play Tic-Tac-Toe");
TicTacToe.setTranslateX(245);
TicTacToe.setTranslateY(170);
TicTacToe.setOnAction(new EventHandler<ActionEvent>(){

            @Override
            public void handle(ActionEvent e) {

                    TicTacToeMain TicTacToeGame = new TicTacToeMain();
                    try {
                            RunsTorinsGame.StartGame(stage);

                    } catch (Exception e1) {
                            // TODO Auto-generated catch block
                            e1.printStackTrace();
                    }
            }

  });
//Button that runs Caleb's Game of using the Iterator Pattern
Button Connect4= new Button("Play Connect 4");
Connect4.setTranslateX(248);
Connect4.setTranslateY(205);
Connect4.setOnAction(new EventHandler<ActionEvent>(){
        @Override
        public void handle(ActionEvent e) {
            try {
                RunCalebsGame.StartGame(stage);
                stage.close();

                } catch (Exception e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }

        }

  });
  //Runs Ezekiel's or my Game of Pong using the Proxy Pattern
  Button Pong= new Button("Play Pong");
  Pong.setTranslateX(262);
  Pong.setTranslateY(240);
  Pong.setOnAction(new EventHandler<ActionEvent>(){

        @Override
        public void handle(ActionEvent e) {
            try {
                RunsEzekielsGame.StartGame(stage);
                stage.close();

                } catch (Exception e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }

        }

  });



  //Linear Gradient For Title
  Stop[] stops = new Stop[] {
        new Stop(0, Color.YELLOW), new Stop(1, Color.RED)
  };
  LinearGradient linearGradient = new LinearGradient(0,0,1,0, true,
CycleMethod.NO_CYCLE, stops);
```

```
    //Title of Game Selector in Scene
    Text title = new Text();
    title.setText("Do you want to Play a Game?");
    title.setFont(Font.font("Verdana",FontWeight.BOLD, 25));
    title.setFill(linearGradient);
    title.setX(100);
    title.setY(85);
   //Instantiating the group class
    Group root = new Group(War, Hangman, TicTacToe, Connect4, Pong, title);
    //Instantiating the Scene class
    Scene scene = new Scene(root, 600, 300, Color.PURPLE);
    //Setting the scene to the Stage
    stage.setScene(scene);
    //Setting Title to the stage
    stage.setTitle("Game Selection");
   //Displaying the contents of the stage
    stage.show();
  }
  public static void main(String args[]){
    launch(args);
  }
}
```

RunCalebsGame.java class

```
package gameFactory;

import javafx.stage.Stage;

public class RunCalebsGame extends GameSelectorUI {
    public static void StartGame(Stage stage) {
        C4 Connect = new C4();
        try {
            Connect.setVisible(true);
        }catch (Exception e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
}
```

RunsEzekielsGame.java class

```
package gameFactory;

import javafx.stage.Stage;

public class RunsEzekielsGame extends GameSelectorUI {
	public static void StartGame(Stage stage) {
		try {
			Proxy.main(null);
		}catch (Exception e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
	}
}
```

RunsCarsonsGame.java class

```
package gameFactory;

import javafx.stage.Stage;

public class RunsCarsonsGame extends GameSelectorUI{
	public static void StartGame(Stage stage) {
		HangmanTestDrive Hangman = new HangmanTestDrive();
		try {
			Hangman.main(null);
			stage.hide();
		}catch (Exception e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		/*Attempt to get Carson's game to run as a batch s instead of in the console
		 *
		 * Process proc = Runtime.getRuntime().exec("java -jar Hangman.jar");
		InputStream in = proc.getInputStream();
		InputStream err = proc.getErrorStream();
		*/
	}
}
```

RunStephensGame.java class

```
package gameFactory;

import javafx.stage.Stage;

public class RunStephensGame extends GameSelectorUI {
        public static void StartGame(Stage stage) {
                GameGUI WarGame = new GameGUI();
                try {
                        WarGame.start(stage);
                }catch (Exception e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                }
        }
}
```

RunsTorinsGame.java class

```
package gameFactory;

import javafx.stage.Stage;

public class RunsTorinsGame extends GameSelectorUI{
        public static void StartGame(Stage stage) {
                TicTacToeMain TicTacToeGame = new TicTacToeMain();
                try {
                        TicTacToeGame.main(null);
                        stage.hide();
                }catch (Exception e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                }
                /* Attempt to get Torin's game to run as a batch file in PowerShell instead of the
console
                 *
                 * Process proc = Runtime.getRuntime().exec("java -jar TicTacToe.jar");
                InputStream in = proc.getInputStream();
                InputStream err = proc.getErrorStream();
                */
        }
}
```

(Ezekiel)

Pong code using Proxy Design Pattern

Proxy.java class

```
package gameFactory;

public class Proxy {
public static void main(String [] args) {
        Pong.main(args);
}
}
```

Pong.java class

```
package gameFactory;

import java.awt.Color;
import javax.swing.JFrame;

public class Pong extends JFrame {
   private final static int WIDTH = 700, HEIGHT = 450;
   private NewPongWindow window;

   public Pong() {
      setSize(WIDTH, HEIGHT);
      setTitle("Pong");
      setBackground(Color.BLUE);
      setResizable(false);
      setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      window = new NewPongWindow(this);
      add(window);
   }

   public NewPongWindow getPanel() {
      return window;
   }

   public static void main(String[] args) {
      new Pong();
   }
}
```

NewPongWindow.java class

```
package gameFactory;
```

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.swing.JPanel;
import javax.swing.Timer;

public class NewPongWindow extends JPanel implements ActionListener, KeyListener {
   private Pong game;
   private Ball ball;
   private Bars player1, player2;
   private int player1score, player2score;

   public NewPongWindow(Pong game) {
      setBackground(Color.BLUE);
      this.game = game;
      ball = new Ball(game);
      player1 = new Bars(game, KeyEvent.VK_UP, KeyEvent.VK_DOWN, game.getWidth() -
36);
      player2 = new Bars(game, KeyEvent.VK_W, KeyEvent.VK_S, 20);
      Timer timer = new Timer(5, this);
      timer.start();
      addKeyListener(this);
      setFocusable(true);
   }

   public Bars getPlayer(int playerNo) {
      if (playerNo == 1)
         return player1;
      else
         return player2;
   }

   public void increaseScore(int playerNo) {
      if (playerNo == 1)
         player1score++;
      else
         player2score++;
   }
```

```
    public int getScore(int playerNo) {
        if (playerNo == 1)
            return player1score;
        else
            return player2score;
    }

    private void update() {
        ball.update();
        player1.update();
        player2.update();
    }

    public void actionPerformed(ActionEvent e) {
        update();
        repaint();
    }

    //Checkers for key inputs and updates
    public void keyPressed(KeyEvent e) {
        player1.pressed(e.getKeyCode());
        player2.pressed(e.getKeyCode());
    }

    public void keyReleased(KeyEvent e) {
        player1.released(e.getKeyCode());
        player2.released(e.getKeyCode());
    }

    public void keyTyped(KeyEvent e) {
        ;
    }

    @Override
    public void paintComponent(Graphics graphic) {
        super.paintComponent(graphic);
        graphic.drawString(game.getPanel().getScore(1) + " : " + game.getPanel().getScore(2),
game.getWidth() / 2, 10);
        ball.paint(graphic);
        player1.paint(graphic);
        player2.paint(graphic);
    }
}
```

Bars.java class

```
package gameFactory;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;

public class Bars {
    private static final int WIDTH = 10, HEIGHT = 60;
    private Pong game;
    private int up, down;
    private int x;
    private int y, actualY;

    public Bars(Pong game, int up, int down, int x) {
        this.game = game;
        this.x = x;
        y = game.getHeight() / 2;
        this.up = up;
        this.down = down;
    }

    public void update() {
        if (y > 0 && y < game.getHeight() - HEIGHT - 29)
            y += actualY;
        else if (y == 0)
            y++;
        else if (y == game.getHeight() - HEIGHT - 29)
            y--;
    }

    //Movement of the paddles
    public void pressed(int ButtonPressed) {
        if (ButtonPressed == up)
            actualY = -2;
        else if (ButtonPressed == down)
            actualY = 2;
    }

    //Stopping movement when the button is released
    public void released(int ButtonPressed) {
        if (ButtonPressed == up || ButtonPressed == down)
            actualY = 0;
    }
    public Rectangle getBounds() {
        return new Rectangle(x, y, WIDTH, HEIGHT);
    }

    public void paint(Graphics graphic) {
        graphic.fillRect(x, y, WIDTH, HEIGHT);
        graphic.setColor(Color.WHITE);
    }
}
```

Ball.java class

```
package gameFactory;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;

import javax.swing.JOptionPane;

public class Ball {
    private static final int WIDTH = 30, HEIGHT = 30;
    private Pong game;
    private int x, y, actualx = 2, actualY = 2;


    public Ball(Pong game) {
        this.game = game;
        x = game.getWidth() / 2;
        y = game.getHeight() / 2;
    }

    //Updates Score with position of the ball, First to 10 Wins
    public void update() {
        x += actualx;
        y += actualY;
        if (x < 0) {
            //increases player 1 score when the ball passes player 2
            game.getPanel().increaseScore(1);
            x = game.getWidth() / 2;
            actualx = -actualx;
        }
        else if (x > game.getWidth() - WIDTH - 7) {
            //increases player 2 score when the ball passes player 1
            game.getPanel().increaseScore(2);
            x = game.getWidth() / 2;
            actualx = -actualx;
        }
        else if (y < 0 || y > game.getHeight() - HEIGHT - 29)
            actualY = -actualY;
        if (game.getPanel().getScore(1) == 10) {
            JOptionPane.showMessageDialog(null, "Player 1 wins", "Pong",
JOptionPane.PLAIN_MESSAGE);
            System.exit(0);
        }
        else if (game.getPanel().getScore(2) == 10) {
            JOptionPane.showMessageDialog(null, "Player 2 wins", "Pong",
JOptionPane.PLAIN_MESSAGE);
            System.exit(0);

    } checkCollision();
    }

    //BouncesBall
    public void checkCollision() {
        if (game.getPanel().getPlayer(1).getBounds().intersects(getBounds()) ||
game.getPanel().getPlayer(2).getBounds().intersects(getBounds()))
            actualx = -actualx;
    }

    public Rectangle getBounds() {
        return new Rectangle(x, y, WIDTH, HEIGHT);
    }

    //Colors and Fills Ball
    public void paint(Graphics graphic) {
            graphic.fillRect(x, y, WIDTH, HEIGHT);
            graphic.setColor(Color.WHITE);
    }
}
```