JavaFX Maven Texas Hold'em Game

Programming Documentation

Torgrim Thorsen
Programming 2 - NTNU in Gjøvik

March 17, 2023

Contents

1 Introduction

This document provides a detailed overview of the development of a JavaFX Texas Hold'em game. The game features a computer opponent, sound effects and music and a dark mode/light mode option. The main objective of this project was to familiarize oneself with the development of a well-structured and efficient, but simple, game using JavaFX. The project ended up in the end becoming a large-scale showcase of how JavaFX can be used to implement various features in Java game design including: 3D objects and animation, sound effects and music and a computer opponent.

2 Development

2.1 Project Structure and Naming Conventions

The project is organized using Maven and follows a clear and concise structure, with proper naming conventions for all classes, methods, and variables See Figure 1. The project structure follows the Model-View-Controller (MVC) architecture to ensure a separation of concerns and make it easy to navigate the project and understand the codebase. Proper naming conventions were implemented to ensure the codebase is easily readable and understandable.

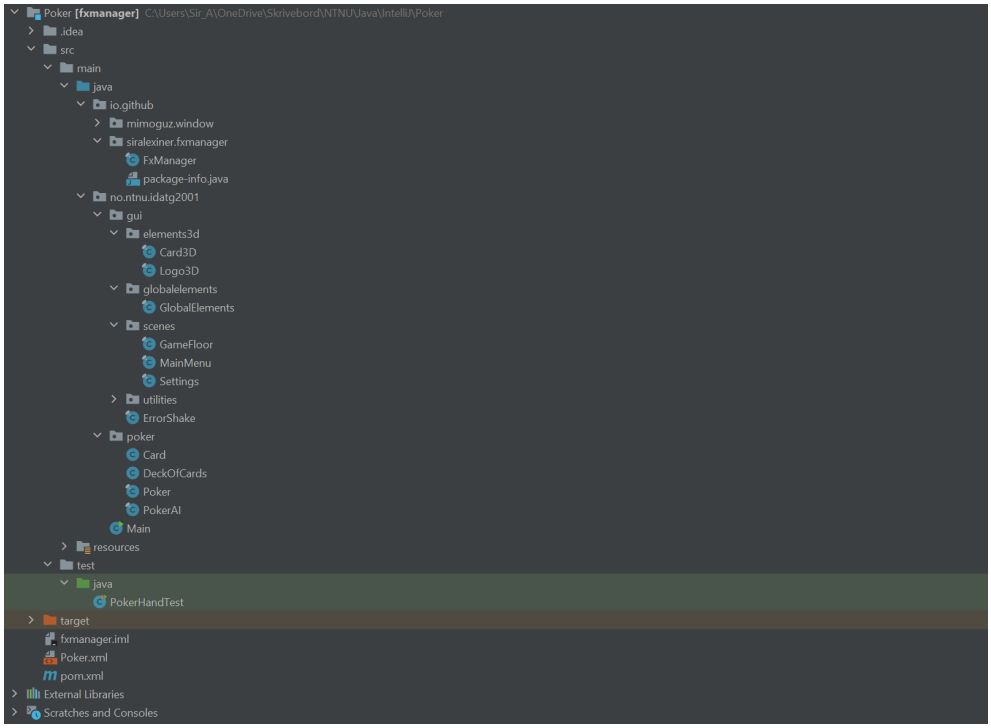


Figure 1: The images shows the projects structure at one point in development.

2.2 CheckStyle and SonarLint

The project has been developed with a focus on maintaining the lowest possible number of CheckStyle and SonarLint warnings, with no warnings, and zero errors. CheckStyle and SonarLint are static analysis tools used to detect coding issues and improve code quality. The project was tested and debugged to eliminate all errors and warnings to ensure the code is efficient and free from issues and bugs. Trough out the entire development phase of the project it has utilized the Google CheckStyle, and will continue to do so.

2.3 Streams and Arrays

Streams and arrays have been implemented in the project, in accordance with project requirements, to improve the overall efficiency and performance of the game. Streams have been implemented for "for" loops, and some arrays have been optimized for performance to reduce the complexity of the game.

2.4 Enums

Enums are a powerful feature of Java that allow developers to define a set of named constants. In this project, enums were implemented late, If implemented earlier it could have made the code much more organized and easier to read earlier. Enums could also have been have been used to define the various poker hands in the game, as an example.

2.5 CSS Styling

JavaFX supports CSS styling, and the developer of this project has prior knowledge in CSS. However, the CSS styling of the game is minimal this is in large part because of how dark mode/light mode was implemented, See Section 2.6.1. Having utilized more CSS could have improved the overall appearance and user experience of the game further, which could be an area for improvement in future iterations of the game.

2.6 Features

2.6.1 Dark Mode/Light Mode

The game features a dark mode/light mode option, allowing users to choose the appearance of the game according to their preferences, a feature not native to JavaFX. Achieved though code developed for this project, it also allows the window's title bar to change color based on the selected mode. Without the developed code the color of the titlebar would always be white, and with no easy way to change it.

The dark mode/light mode feature is developed using two GitHub dependencies: a JavaFX CSS library: AtlantaFX [1], and an OS theme detection library: JSystemThemeDetector [2]. The user can switch between the modes at any time using the in-game settings, and the game's appearance will be automatically updated. The user can also select the game to mirror the OS theme, if supported. See Section 3.3 for how this feature can be improved in the future.

2.6.2 3D Objects and Animation

As mentioned in the Introduction the game also features 3D objects and animation, with "Global" illumination and reflections. The use of 3D objects and animation enhances the visual appeal of the game and provides an immersive experience for the user.

The development of 3D Objects in code, with animations, provided a challenging aspect. And the addition of reflections and a global light-source to the "scene" proves a harder aspect. Going forward a better understanding of JavaFX's tools and resources for their 3D-GUI is going to be vital.

2.6.3 Sound Effects and Music

Sound effects and music have been added to enhance the overall immersion of the game. The game includes sound effects for various game events, such as card shuffling and dealing, as well as background music. The sound effects and music were carefully selected to complement the game's theme and overall appearance, and provide an enjoyable gaming experience.
The game assets that where not created for this project is used under the following licences:

- CC-BY-SA 3.0 (Creative Commons)
- GPL 3.0 & 2.0 (General Public Lience)
- CC0 (No Rights Reserved)

From OpenGameArt.org [3]

2.6.4 Computer Opponent

A functioning computer opponent has been developed to provide a single-player gaming experience. The computer opponent uses static decimal numbers to determine when to raise, check, and fold, utilizing a working developed logic, See Appendix A. These Upper and Lower limits where determined in development by creating a Excel spreadsheet for the maximum value for each poker hand based on the highest value of that hand, See Figure 2. The computer opponent is designed to provide a challenging opponent for the player and keep the game interesting.

Going forward further development on the opponent is required to continue to provide a challenging and fun experience to the players, as well as working on implementing additional opponents.

Highest Card	High Card	One Pair	Two Pairs	Three of a Kind	Straight	Flush	Full House	Four of a Kind	Straight Flush
Two	0,00	21,20		101,20				261,20	
Three	1,00	23,20	63,20	103,20			223,20	263,20	
Four	2,00	25,20	66,20	105,20			226,20	265,20	
Five	3,00	27,20	69,20	107,20			229,20	267,20	
Six	4,00	29,20	72,20	109,20	149,20	189,20	232,20	269,20	309,20
Seven	5,00	31,20	75,20	111,20	151,20	191,20	235,20	271,20	311,20
Eight	6,00	33,20	78,20	113,20	153,20	193,20	238,20	273,20	313,20
Nine	7,00	35,20	81,20	115,20	155,20	195,20	241,20	275,20	315,20
Ten	8,00	37,20	84,20	117,20	157,20	197,20	244,20	277,20	317,20
Jack	9,00	39,20	87,20	119,20	159,20	199,20	247,20	279,20	319,20
Queen	10,00	41,20	90,20	121,20	161,20	201,20	250,20	281,20	321,20
King	11,00	43,20	93,20	123,20	163,20	203,20	253,20	283,20	323,20
Ace	12,00	45,20	96,20	125,20	165,20	205,20	256,20	285,20	325,20

AI Highest Card:	Ace	12

Figure 2: The images shows an Excel file with a projection of numbers used in determining the Upper and Lower limits of the opponent.

Table 1: The table shows how scores are calculated

Highest Card	Value	Highest on Hand	Value	Player Hand Rank	Value	Calc	Score
Jack	9	Five	3	Two Pairs	60	$60+9\times2+3+0.3(1)$	81.3
Queen	10	Four	2	One Pair	20	$20+10\times2+0.2(2)$	40.2
King	11	Three	1	Straight Flush	300	$300+11\times2+0.1(3)$	322.1
Ace	12	Two	0	Flush	180	$180+12\times2+0.0(4)$	204.0

2.6.5 Calculation Example

Table 1 shows how the scores are calculated, take note that in the case of a pair in Jack on the table matched with a pair in Five on hand: Equation (1), we need to calculate the value of the second lower pair, this also applies for the hand: Full House. If the game was not a single player and supported four players, player 3 with a Straight Flush would win this example. [1]

[1] In this example we ignore the possibility of 4 players having these hands against each other for demonstration purposes

2.7 Testing

2.7.1 Matlab Script

A Matlab script was written to simulate around 228 thousand win-lose conditions of the game using a math function for assigning the poker hands values, See Table 1. The script [2.7.1] ensures that there are an equal number of losses and wins, and an even number of ties, providing a fair and balanced game.
The final output of the script is a text file that contains every combination of poker hands between two players and the outcome from checking them against each other. Using this file alongside development allowed for a reference to which we could refer if the game did not adhere to the rules of poker, allowing us to quickly tweak the code so it went back to adhering to them.

Going forward the Matlab script could be developed further to confirm that every possible bout is valid, and implement ignorance towards impossible hand combinations. Using this we could rework the Java code to use the same logic that the reworked Matlab script would utilize.

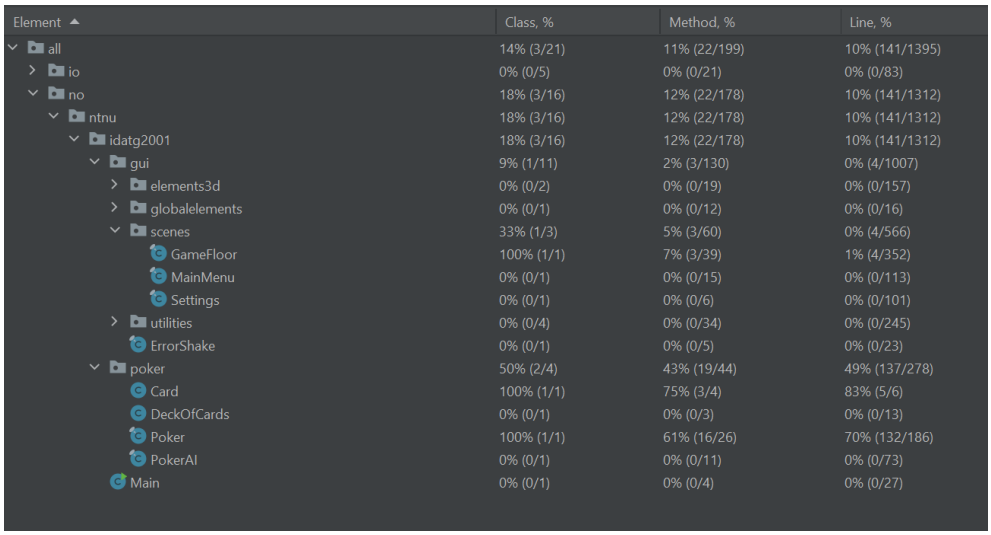
Code listing 1: MatLab excerpt from file

```
% Load the player and computer data
player_data = zeros(13, 8);
computer_data = zeros(13, 8);
for k = 0:7
    for l = 0:12
        if isequaln(k,1) || isequaln(k,5)
            player_data(l+1, k+1) = 20.0 + (40.0 * k) + (l * 2.0) + x_values(i) * 10 + x_values(i);
            computer_data(l+1, k+1) = 20.0 + (40.0 * k) + (l * 2.0) + y_values(j) * 10 + y_values(j);
        else
           player_data(l+1, k+1) = 20.0 + (40.0 * k) + (l * 2.0) + x_values(i);
           computer_data(l+1, k+1) = 20.0 + (40.0 * k) + (l * 2.0) + y_values(j);
        end
    end
end

% Compare the player and computer data
poker_matrix = cell(size(player_data, 1), size(player_data, 2), size(computer_data, 1));
for k = 1:size(player_data, 2)
    for l = 1:size(player_data, 1)
        for m = 1:size(computer_data, 1)
            if player_data(l,k) < computer_data(m,k)
                poker_matrix{l,k,m} = [pokerHandToString(k), ' in ', cardRankToString(l), ' # ', num2str(player_data(l,k)), ' Against ',
                pokerHandToString(k), ' in ', cardRankToString(m), ' & ', num2str(computer_data(m,k)), ' | Computer Wins'];
            elseif isequaln(player_data(l,k), computer_data(m,k))
                poker_matrix{l,k,m} = [pokerHandToString(k), ' in ', cardRankToString(l), ' # ', num2str(player_data(l,k)), ' Against ',
                pokerHandToString(k), ' in ', cardRankToString(m), ' & ',
                num2str(computer_data(m,k)), ' | Tie'];
            elseif player_data(l,k) > computer_data(m,k)
                poker_matrix{l,k,m} = [pokerHandToString(k), ' in ', cardRankToString(l), ' # ', num2str(player_data(l,k)), ' Against ',
                pokerHandToString(k), ' in ', cardRankToString(m), ' & ',
                num2str(computer_data(m,k)), ' | Player Wins'];
            end
        end
    end
end
```

2.7.2 Junit Coverage

Junit is a testing framework used to perform unit testing on Java applications. Within our project The non-GUI methods have a 100% coverage. This ensures that the "engine" of the game performs exactly as expected and without or with minimal issues.



Element	Class, %	Method, %	Line, %
all	14% (3/21)	11% (22/199)	10% (141/1395)
io	0% (0/5)	0% (0/21)	0% (0/83)
no	18% (3/16)	12% (22/178)	10% (141/1312)
ntnu	18% (3/16)	12% (22/178)	10% (141/1312)
idatg2001	18% (3/16)	12% (22/178)	10% (141/1312)
gui	9% (1/11)	2% (3/130)	0% (4/1007)
elements3d	0% (0/2)	0% (0/19)	0% (0/157)
globalelements	0% (0/1)	0% (0/12)	0% (0/16)
scenes	33% (1/3)	5% (3/60)	0% (4/566)
GameFloor	100% (1/1)	7% (3/39)	1% (4/352)
MainMenu	0% (0/1)	0% (0/15)	0% (0/113)
Settings	0% (0/1)	0% (0/6)	0% (0/101)
utilities	0% (0/4)	0% (0/34)	0% (0/245)
ErrorShake	0% (0/1)	0% (0/5)	0% (0/23)
poker	50% (2/4)	43% (19/44)	49% (137/278)
Card	100% (1/1)	75% (3/4)	83% (5/6)
DeckOfCards	0% (0/1)	0% (0/3)	0% (0/13)
Poker	100% (1/1)	61% (16/26)	70% (132/186)
PokerAI	0% (0/1)	0% (0/11)	0% (0/73)
Main	0% (0/1)	0% (0/4)	0% (0/27)

Figure 3: The images shows the projects coverage at one point in development.

Code listing 2: Java Testing excerpt from file

```
  @Test
  void isRoyalStraightFlush() {
    Card cards = {new Card("Ace", "Spades"), new Card("King", "Spades")};
    ArrayList<Card> board = new ArrayList<>();
    board.add(new Card("Queen", "Spades"));
    board.add(new Card("Jack", "Spades"));
    board.add(new Card("Ten", "Spades"));
    board.add(new Card("Ten", "Hearts"));
    board.add(new Card("Ten", "Clubs"));
    GameFloor.setBoard(board);
    String opponentHand = Poker.getHand(cards);
    Assertions.assertEquals("Royal␣Straight␣Flush", opponentHand);
  }
  @Test
  void isNotRoyalStraightFlush() {
    Card cards = {new Card("Ten", "Spades"), new Card("Jack", "Spades")};
    ArrayList<Card> board = new ArrayList<>();
    board.add(new Card("Queen", "Spades"));
    board.add(new Card("King", "Spades"));
    board.add(new Card("Ace", "Clubs"));
    board.add(new Card("Ten", "Hearts"));
    board.add(new Card("Ten", "Diamonds"));
    GameFloor.setBoard(board);
    String opponentHand = Poker.getHand(cards);
    Assertions.assertEquals("Straight", opponentHand);
  }
}
```

3 Restrictions and Future Plans

3.1 Deviations

As mentioned in the introduction, the project deviated from the given task of creating a simple, at bare minimum, text based GUI card game. It all started with the developer dabbling in JavaFX and exploring its capabilities, which lead to expanding on each new exciting feature JavaFX offered, despite the deviation, the project still manages to meet the projects requirements, and showcase the developer's knowledge and skill in programming and software development.

One downside of this deviation is that the Git history of the project is lacking, which could have been helpful in tracking the progress of the project and identifying any issues or bugs that were encountered and resolved, or went undetected.

3.2 OS Support

Currently, the game is designed for Windows deployment and may not run on other operating systems. This is a limitation that could potentially reduce the game's reach and popularity. In future iterations of the game, adding support for different operating systems should be explored.

3.3 Future Features

The developer plans to continue working on the game to add more features and improve its overall functionality. Some of the planned future features include: multiplayer support, additional opponents, improved opponents, fixing the issue with screen resolution and the game's size, currently forces 1920x1080 Fullscreen, adding different OS support, resolving issue with change listener to autoupdate the light/dark mode when changed on the OS level and so much more. In addition, the developer is considering a possible release on gaming platforms such as Steam or Itch.io to reach a wider audience.

4 Conclusion

In conclusion, the JavaFX Texas Hold'em game developed in this project is a well-structured and efficient project that showcases the developer's knowledge and skills in software development. The game includes a number of exciting features, such as a functioning computer opponent, sound effects and music, and a dark mode/light mode option. With the addition of planned future features and improvements, this game has the potential to be a popular and engaging release if released onto a platform.

References

[1] AtlantaFX. 'Mkpaz.' (Jul. 2022), [Online]. Available: `https://github.com/mkpaz/atlantafx`.

[2] JSystemThemeDetector. 'Daniel gyoerffy.' (Oct. 2022), [Online]. Available: `https://github.com/Dansoftowner/jSystemThemeDetector`.

[3] OpenGameArt.org. 'Bart kelsey.' (Apr. 2009), [Online]. Available: `https://opengameart.org`.

A Additional Material

Code listing 3: Java AI Logic excerpt from file

```
private static int getBetAction(double handStrength) {
  if (handStrength < 20) {
    return calculateAction(handStrength, 3, 8);
  } else if (handStrength < 60) {
    return calculateAction(handStrength, 27.20, 37.20);
  } else if (handStrength < 100) {
    return calculateAction(handStrength, 72.20, 84.20);
  } else if (handStrength < 140) {
    return calculateAction(handStrength, 107.20, 117.20);
  } else if (handStrength < 180) {
    return calculateAction(handStrength, 153.20, 159.20);
  } else if (handStrength < 220) {
    return calculateAction(handStrength, 193.20, 199.20);
  } else if (handStrength < 260) {
    return calculateAction(handStrength, 232.20, 244.20);
  } else if (handStrength < 300) {
    return calculateAction(handStrength, 267.20, 277.20);
  } else if (handStrength < 1000) {
    return calculateAction(handStrength, 313.20, 319.20);
  } else { return 10000; }
}

private static int calculateAction(double handStrength, double x, double y) {
  if (handStrength <= y) {
    return foldOrCall(handStrength, x);
  } else {
    if (Math.max(y, handStrength) != handStrength) {
      return callCheck(); // Call : Check
    } else { return raise(bet * 2); // Raise }
  }
}

private static int foldOrCall(double handStrength, double x) {
  if (handStrength <= x) {
    int z = rand.nextInt(0, 4);
    if (z == 0) {
      fold(); // Fold
      return 0;
    } else { return callCheck(); // Call : Check }
  } else { return callCheck(); // Call : Check }
}
```