



# BLACKJACK SIMULATOR

---

## **Team 4**

Robert Carswell

Jordan DeLaGarza

Jose Reyes

Patrick Smith

Samuel Hudgins

12 December 2023

## Table of Contents

<b>1.0 Overview .....</b>	<b>6</b>
<b>2.0 Project Plan .....</b>	<b>7</b>
2.1 Problem, Purpose, and Solution .....	7
2.2 Project Organizational Structure .....	8
2.3 Roles and Responsibilities .....	8
2.4 Scope Statement.....	10
2.4.1 Requirements Identification and Modification.....	10
2.4.2 Work Breakdown Structure .....	11
2.4.3 Sponsor Acceptance.....	11
2.4.4 Scope Control.....	12
2.4.5 Schedule Management Plan.....	12
2.5 Project Justification .....	12
2.5.1 Major Deliverables and Milestones .....	13
2.5.2 Major Known Risks .....	13
2.5.3 Project Assumptions and Constraints .....	14
2.5.4 Procurement of Personnel and Other Resources.....	15
2.6 Project Contacts .....	16
2.7 Communications Management .....	16
2.8 Risk Management .....	18
2.8.1 Risk Management Strategy.....	18
2.8.2 Risks Related to Requirements and Other Items .....	19
2.8.3 Risk Mitigation and Contingency Plans .....	20
2.9 Change Management Plan .....	22
2.9.1 Request Process .....	22
2.9.2 Generation.....	23
2.9.3 Evaluation.....	23
2.9.4 Authorization.....	24
2.9.5 Implementation .....	24
2.9.6 Change Control Board .....	27
2.10 Time and Cost Management .....	28
2.10.1 Time and Cost Management and Control .....	28
2.10.2 Estimating Project Costs.....	28
2.10.3 Tracking and Controlling Actual Project Costs .....	29
2.10.4 Analyzing Budgeted and Actual Project Costs .....	29
2.11 Quality Management Plan .....	30
2.11.1 Quality Control Process .....	30
2.11.2 Product Testing .....	30
2.12 Development Plan .....	31
2.12.1 Process Model.....	32

2.12.2 Technical Requirements .....	34
<b>3.0 Requirements Specification .....</b>	<b>35</b>
3.1 Purpose.....	35
3.2 Scope.....	36
3.3 Definitions, Acronyms, and Abbreviations .....	36
3.4 Organizations .....	37
3.5 Overall Description .....	37
3.5.1 Product Perspective.....	37
3.5.2 Product Functions.....	38
3.5.3 User Characteristics .....	38
3.5.4 Constraints .....	38
3.5.5 Assumptions and Dependencies.....	39
3.5.6 Apportioning of Requirements .....	39
3.6 Specific Requirements.....	39
3.7 Modeling Requirements.....	41
3.7.1 Use Case Diagram .....	42
3.7.2 Class Diagram (Preliminary) .....	48
3.8 Requirements and Features List.....	51
<b>4.0 User Guide.....</b>	<b>54</b>
<b>5.0 Test Plan and Results .....</b>	<b>54</b>
5.1 Objectives .....	54
5.2 Scope.....	54
5.2.1 Features to Test .....	54
5.3 Types of Testing.....	57
5.3.1 Black Box Testing.....	57
5.3.2 GUI Testing .....	57
5.3.3 White Box Testing .....	58
5.3.4 Non-Functional Test .....	58
5.3.5 User Acceptance Testing.....	59
5.3.6 Test Documentation.....	61
5.3.7 Defect Documentation .....	61
5.4 Entry and Exit Criteria for Testing.....	64
5.5 Environment Requirements for Testing .....	65
5.6 Testing Schedule and Testing Cycle .....	65
5.6.1 Item Transmittal Report .....	68
5.7 Test Metrics.....	69
5.7.1 Passed Test Cases Percentage.....	69
5.7.2 Failed Test Cases Percentage.....	69
5.7.3 Critical Defects Percentage .....	70
5.7.4 Test Count.....	70
5.8 Potential Risk .....	70

5.9 Security Considerations.....	70
5.10 Training Considerations.....	71
5.11 Roles and Responsibilities .....	71
<b>6.0 Design and Alternate Designs.....</b>	<b>72</b>
6.1 Purpose of the Software Design Specification .....	72
6.2 General Overview and Design Guidelines/Approach .....	73
6.2.1 Scope .....	73
6.2.2 Assumptions .....	73
6.2.3 Constraints .....	74
6.2.4 Design Precedents .....	75
6.2.4.1 GUI .....	75
6.2.4.2 Account Registration and Login .....	77
6.2.4.3 Tutorials .....	78
6.2.4.4 Displaying Optimal Blackjack Actions.....	79
6.2.4.5 Databases.....	80
6.3 System Design .....	82
6.3.1 Overview .....	82
6.3.2 Use-Cases.....	83
6.3.3 User Interface Design .....	83
6.3.3.1 UI Overview .....	83
6.3.3.2 Start Scene .....	85
6.3.3.3 Registration Scene.....	86
6.3.3.4 Login scene .....	87
6.3.3.5 Main menu .....	88
6.3.3.6 Tutorial.....	89
6.3.3.7 The Betting Scene .....	90
6.3.3.8 Blackjack Match.....	91
6.3.3.9 Stats Scene .....	92
6.3.3.10 Account Deletion Scene.....	93
6.4 Architecture and Design .....	93
6.4.1 JavaFX GUI Architecture .....	96
6.4.2 Security Architecture.....	100
6.4.3 SQLite JDBC Database Design .....	102
6.4.3.1 Required Database Features .....	104
6.4.3.2 Message Formation Process .....	105
6.4.3.3 SQLite Query Strings .....	107
6.4.4 Blackjack Gameplay Process .....	112
6.4.5 Odds Generation Process.....	114
6.5 Alternate Design Decisions .....	116
6.5.1 GUI .....	116
6.5.2 Database .....	116

6.5.3 Timed Events.....	117
6.5.4 Animations.....	117
<b>7.0 Development History .....</b>	<b>118</b>
7.1 Introduction .....	118
7.2 Version History.....	118
7.3 Current Version .....	123
<b>8.0 Summary .....</b>	<b>124</b>
8.1 Technical Decisions and Outcomes.....	124
8.2 The Actual and Delivery Schedule.....	125
8.3 Cost Burn .....	126
8.4 Gantt Metrix.....	126
<b>9.0 Conclusions .....</b>	<b>127</b>
Individual Lessons Learned:.....	128
<b>Appendix A - Project Charter .....</b>	<b>133</b>
<b>Appendix B - Project Team.....</b>	<b>134</b>
<b>Appendix C - Statement of Work.....</b>	<b>135</b>
<b>Appendix D – Schedule .....</b>	<b>137</b>
<b>Appendix E - Gantt Chart and EVM.....</b>	<b>139</b>
<b>Appendix F - Scope Baseline .....</b>	<b>141</b>
<b>Appendix G - Change Request Form.....</b>	<b>143</b>
<b>Appendix H - Document References.....</b>	<b>144</b>
<b>Appendix I - User’s Guide.....</b>	<b>145</b>
I.1 What is the Blackjack Simulator Game? .....	145
I.2 Minimum System Requirements.....	145
I.3 How to Create a User Account .....	145
I.4 How to Log Into The Game .....	147
I.5 Delete User Account .....	148
I.6 Game Play .....	149
I.6.1 Bank Balance .....	149
I.6.2 Betting .....	150
I.6.3 Cards dealt to the Player and Dealer .....	151
I.6.3.1 What is a HIT? .....	151
I.6.3.2 What is a Stand?.....	152
I.6.3.3 What is a Double?.....	152
I.6.3.4 What is a Split? .....	153
I.6.3.5 What is insurance? .....	154
I.6.3.6 What is a Push?.....	155
I.6.4 How do you Win or Lose?.....	156
I.6.4.1 How does a Player Win?.....	156
I.6.4.2 How does a Player Lose? .....	156
I.6.4.3 Win and Loss amounts. ....	156

I.7.0 View Stats .....	157
I.8.0 Tutorial .....	158
I.9.0 Additional Help .....	158
<b>Appendix J - Test Case Table .....</b>	<b>159</b>
<b>Appendix K - Final Test and Results .....</b>	<b>165</b>
<b>Appendix L - Document References .....</b>	<b>190</b>
<b>References .....</b>	<b>191</b>

## 1.0 Overview

Dynamic Gaming (DG) embarked on a project to develop an elegant and sophisticated blackjack gaming simulator for desktop computers. In response to the dominance of casinos, particularly evident in Las Vegas, the simulator seeks to provide players with a visually appealing digital environment that calculates their chances of winning while recording, calculating, and displaying their statistical play history. The Las Vegas tourism industry, with millions of visitors each year, serves as a promising market for DG's venture. The project is structured with a four-unit organizational framework, comprising the Project Manager (PM), Requirements Manager (RM) and User Interface (UI) / User Experience (UX), Test Director (TD), and Software Designer (SD). Each unit plays a distinct role, ensuring cohesive collaboration to achieve project success.

The PM, Robert Carswell, takes on the overarching leadership role, overseeing and implementing the strategic direction of the project. Jordan DeLaGarza leads the RM and UI/UX teams, which are in charge of gathering, evaluating, and prioritizing project requirements to ensure alignment with user needs and expectations. The TD, under Patrick Smith, oversees the testing process to guarantee software functionality and quality standards. Meanwhile, the SD, represented by Samuel Hudgins and Jose Reyes, translates project requirements into practical, efficient, and reliable software solutions.

The scope of the project involved developing a comprehensive blackjack simulator that offers users an engaging and educational experience. Key features include a user-friendly interface, an odds generator, and statistical tracking capabilities to enhance player decision-making during gameplay. The project was not without its challenges, and identified risks ranged from issues related to Random Number Generation (RNG) predictability to concerns about user input validation, the implementation of game rules, capacity bottlenecks, and data storage efficiency. Mitigation strategies were outlined to address these risks and ensure optimal project performance.

DG's project followed an 8-week schedule, emphasizing key development phases such as design, testing, and user experience. We adopted the agile Software Development Life Cycle (SDLC) model, supported by a Kanban board for efficient task management. Technical requirements included using Java, Eclipse IDE, GitHub, Trello, and other essential tools. The project's success was measured by adherence to timelines and budgets and by its ability to deliver a high-quality blackjack simulator that met user expectations and enhanced the overall gaming experience.

## **2.0 Project Plan**

Our team, Dynamic Gaming (DG), produced an elegant yet sophisticated blackjack gaming simulator for the desktop personal computer. This plan defines the project's scope, requirements, and milestones while identifying each stakeholder's role and responsibility. This plan also reports the project's known and perceived risks with a robust quality management and development plan.

### **2.1 Problem, Purpose, and Solution**

According to the Las Vegas Convention and Visitor Authority Research Center (2023), the Las Vegas, Nevada, tourism industry saw over 38 million visitors in 2022 and another 27 million through the end of August 2023. The Las Vegas Gaming Commission reported over 16 billion dollars in revenue through the first two quarters of 2023 (Velotta, 2023). However, it is no secret that the odds favor the casino; the DG blackjack simulator that turns the tide. This game simulator provides the player with an aesthetically pleasing digital environment that calculates their chance of winning while recording, calculating, and displaying their statistical play history.



## **2.2 Project Organizational Structure**

The organizational structure of this project consists of four distinct units: the Project Manager (PM), Requirements Manager (RM) / User Experience (UX), Test Director (TD), and Software Designer (SD).

## **2.3 Roles and Responsibilities**

The PM assumes the overarching leadership role in the organization, and each unit leader is accountable for managing their specific area while supporting a cohesive project environment. The UX gathers and analyzes user requirements and ensures the software design aligns with the user's needs and expectations. The TD oversees the testing process to ensure the software functions properly and meets all quality standards. The SD creates the technical design of the software and implements it according to the specifications provided by the RM and TD. Together, these four units work collaboratively to ensure the successful completion of the project.

### ***2.3.1 Project Manager — Robert Carswell***

The PM plays a vital role in a program design team, overseeing and implementing the program's strategic direction. The PM ensures the project is clearly defined, adequately funded, and on schedule and within budget. They proficiently handle stakeholders, guarantee quality assurance, oversee change management, and exercise financial control, fostering a winning culture. Moreover, the PM overcomes obstacles to ensure the successful completion of intricate projects in alignment with business goals, adhering to predetermined timelines and financial constraints.

### ***2.3.2 Requirements Manager / User Experience — Jordan DeLaGarza***

The RM and UX designer are crucial in the development team. The RM collects, analyzes, and records project requirements, prioritizing functional and non-functional characteristics. They manage changes to requirements efficiently, ensuring the project aligns with initial needs and objectives. The UX designer champions end-users, creating a user-friendly experience through thorough user research, wireframes, prototypes, interaction design, visual design, and usability testing. Their responsibilities include creating a visually appealing interface, ensuring user flow, and assessing the design's efficiency. Together, they ensure the final product meets the target audience's requirements and expectations, ensuring a user-friendly and visually appealing interface.

### ***2.3.3 Test Director — Patrick Smith***

The TD plays a pivotal role within a program design team, ensuring the quality and reliability of the program's outcomes. They oversee test planning, strategy development, and coordination of activities. They manage testing resources, assess risks, and collaborate with project members to align test cases with program requirements. They conduct thorough quality assessments to identify and address defects and issues, ensuring the program's components and deliverables meet quality standards, regulatory requirements, and user expectations, contributing to successful program implementation.

### ***2.3.4 Software Designer — Samuel Hudgins and Jose Reyes***

The SD is crucial to the program design team, translating project requirements into practical, efficient, and reliable software solutions. They design the software

architecture, make critical decisions about technologies, databases, and coding standards, and create detailed system design documentation. They collaborate closely with team members to ensure alignment between design and project goals, identify potential issues, conduct system testing, and refine the design to achieve optimal performance and maintainability. Their role contributes to the successful development of software systems that meet user needs and adhere to quality and scalability standards.

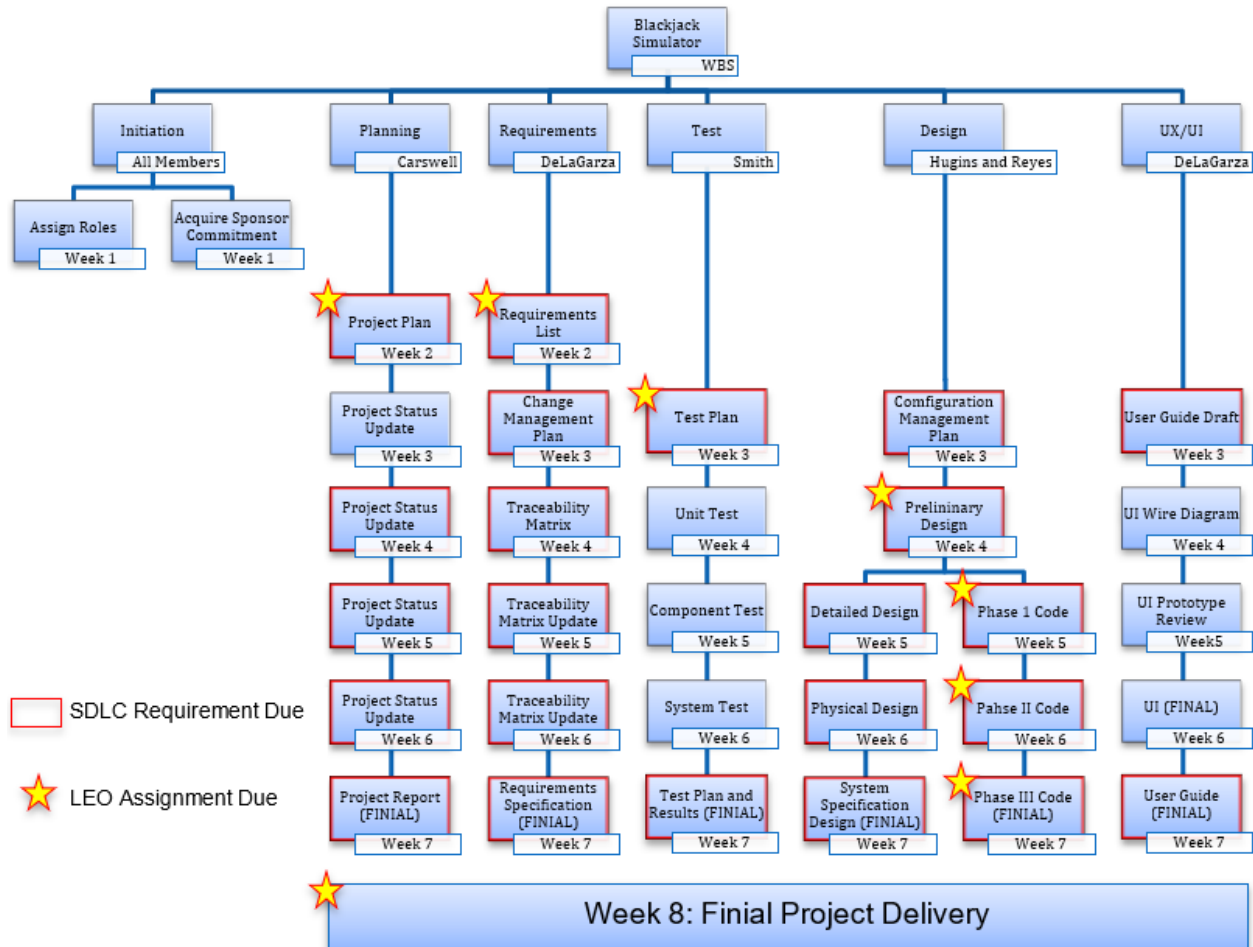
## **2.4 Scope Statement**

This project aims to develop a comprehensive blackjack simulator that allows users to play simulated games of blackjack and maintains a record of the player's statistical history. The primary objective is to create a user-friendly software application that provides an engaging and educational experience for players while offering a statistical history feature to track and analyze gameplay performance.

### **2.4.1 Requirements Identification and Modification**

All relevant stakeholders and the project sponsor have reviewed and approved the project scope statement and project requirements. Any changes to the project scope will follow the defined change control process.

## 2.4.2 Work Breakdown Structure



## 2.4.3 Sponsor Acceptance

The sponsor received a comprehensive presentation of the application's full capabilities upon its completion. This demonstration exhibited the functionality and array of features offered by the program and, at the same time, offered the sponsor the opportunity to witness how the application enhances their firm and fulfills their requirements. Furthermore, we resolved questions and concerns the sponsor had to guarantee their contentment with the application.

#### **2.4.4 Scope Control**

The primary focus was on the actions and efforts required to achieve the specified project requirements. We worked within the constraints associated with this project, particularly the need for the project to adhere to the requirements, predefined budget, and timeline. We also were aware of the potential limitations imposed by the hardware and software requirements of the target operating systems.

#### **2.4.5 Schedule Management Plan**

DG adhered to the 8-week schedule outlined in Appendix D that defines the key phases of the development process for the Blackjack Simulator, from project initiation to final acceptance, with a strong emphasis on design, development, testing, and user experience. Our schedule provides a structured framework for delivering a high-quality product with statistical tracking capabilities.

### **2.5 Project Justification**

The purpose of this project is to provide a casual, informative blackjack experience. Blackjack is one of the games that casinos offer and can involve losing real money. Some people want to learn how to play and enjoy casino games without the risks of real monetary impacts, so we designed that experience. Along with adding features to help players learn how to play blackjack, we provided features to help them learn how to win. For example, one of the features we implemented was an odds generator, which calculates the player's best course of action to win given their and the dealer's cards. We also implemented features that record and display data about the player's play history to help them make informed decisions during blackjack gameplay.

### 2.5.1 Major Deliverables and Milestones

Deliverable	Description	Due Date	Deadline Met?
Project Plan (PP)	The project scope and management plan	10/31/23	Yes
Project Design (PD)	All testing procedures and expectations	11/14/23	Yes
Test Plan (TP)	The project testing scope, expectations, and final intended product	11/7/23	Yes
Phase 1 Source (P1)	Basic GUI scenes, interactive elements, and blackjack gameplay.	11/21/23	Yes
Phase 2 Source (P2)	Implement additional blackjack actions, the user accounts database, and stats screen.	11/28/23	Yes
Phase 3 source (P3)	Implement odds generator, enhance the GUI with CSS and imagery, and add other outstanding features and fixes.	12/5/23	Yes
Final	Finished program with all documentation	12/12/23	Yes

*Note: See Appendix D for a detailed schedule.*

### 2.5.2 Major Known Risks

Random Number Generation (RNG): inadequate RNG can make the game predictable, resulting in repeating card shuffles and dealing patterns. Predictable RNG jeopardizes the simulator's integrity, which seeks to simulate the randomness of a real-world blackjack game. So, we ensured the set of cards played is disjointed from the set yet to be played.

User Input Verification: incorrect input validation can lead to bugs or crashes. Allowing a user to enter invalid bets, such as betting more money than they have in their bank or other activities, could interrupt the game flow or cause the software to crash.

Implementing Game Rules: incorrectly implementing blackjack rules results in an inaccurate simulator. Mistakes could include incorrectly processing split hands or payout ratios that distort the game. We, for instance, used the Las Vegas gaming standard for blackjack with a 3:2 payout ratio for natural blackjacks and a 2:1 payout ratio for insurance.

Capacity bottlenecks: Inefficient code can cause the simulator to lag, making it unsuitable for casual usage. Poorly tuned algorithms or data structures may cause performance bottlenecks. However, comprehensive testing allowed us to ensure the code was optimal and the user's device could handle the application.

Saving and retrieving data: inefficient or insecure data storage may result in performance problems or data loss. Data integrity becomes crucial if the simulator saves user data or game statistics in an external location, such as a file on the user's device or an online database. However, proper database usage and protocols allowed us to ensure the application transferred data efficiently and safely.

### **2.5.3 Project Assumptions and Constraints**

The allocated timeline, limited resources, and hardware and software requirements of the target operating systems required us to invoke assumptions and constraints to prioritize tasks and allocate resources efficiently. The project assumes that users understand their operating system and the physical and mental capability to navigate the digital environment. Concerning constraints, DG implemented data privacy

and security measures to protect user data, though we limited the required level of security. We employed effective communication and collaboration among team members to optimize the utilization of available resources while finding innovative solutions within the given limitations. Software testing and quality assurance measures also helped us ensure the software's functionality was consistent on multiple devices.

#### **2.5.4 Procurement of Personnel and Other Resources**

Collaborative projects require a team to combine their skill sets and resources to produce a successful product. Our goal for this software project was to design our interpretation of the game of blackjack that people can play on their personal computers. We wanted to design an entertaining video game, so we had to blend our skills and resources in developing software, designing graphic user interfaces, and creating art. Specifically speaking, these were the personnel and resources we needed to complete the project:

<b>Personnel</b>	<b>Resources</b>
<ul style="list-style-type: none"> <li>- Software designer</li> <li>- Graphics designer / Artist</li> <li>- Code tester</li> </ul>	<ul style="list-style-type: none"> <li>- Integrated development environment for developing our backend code</li> <li>- A Software application for prototyping and designing the user interface(s)</li> <li>- A software application for designing 2D assets</li> </ul>

If our time and resources permit us, we may also want to implement sounds and music into our game. Adding sounds could require a sound artist and access to software for designing sounds (e.g., a digital audio workstation) if we wish to develop our own sounds.



## 2.6 Project Contacts

The following table shows contact information for the individuals who played a significant role in developing this project and will be updated when needed.

Role	Name	Organization	E-mail Address
Project Sponsor	Mentzos, Terrence	UMGC	Terry.mentzos@faculty.umgc.edu
Project Manager	Carswell, Robert	Dyanmic Gaming, Inc.	Robert.carswell34@gmail.com
Test Director	Smith, Patrick	Dyanmic Gaming, Inc.	Bblackwave5@gmail.com
Software Design	Hudgins, Samuel	Dyanmic Gaming, Inc.	Samuel_h_713@msn.com
Software Design	Reyes, Jose	Dyanmic Gaming, Inc.	Jose.Reyesfabian@gmail.com
Requirements Manager / User Experience	DeLaGarza, Jordan	Dyanmic Gaming, Inc.	Jordandelagarza2001@gmail.com

## 2.7 Communications Management

Internal team communication ensures everyone is aligned with project goals and progress.

### Communication Methods:

**Emails:** For formal communication

**Shared Documentation:** Google documents

**Instant Messaging/Chat:** Discord: Quick exchanges for non-formal discussions.

**Phone Calls:** Especially for urgent matters or detailed discussions.

**Project Management Tools:** Trello: Task tracking, Github: Project code

### Communication Types:

**Meeting Locations:** Virtual meetings through Zoom for remote teams, and physical standups for co-located teams.

**Regular Team Meetings:** Teams: Weekly meetings to discuss project progress, challenges, and upcoming tasks.

**Standup Meetings:** Discord: Quick daily check-ins for brief updates and issue resolution.

**Ad-hoc Meetings:** As needed for critical decisions or problem-solving.

#### **Communication Aids:**

**Agendas:** Provided 24hr in advance by the meeting sponsor. It should ensure focus and productivity.

**Feedback and Retrospectives:** Regularly gather team feedback to improve communication processes.

Given the dynamic nature of projects, the communication plan is flexible and adaptable. It will be regularly reviewed and updated based on changing project needs or feedback from stakeholders and team members. This will establish clear communication channels, methods, and expectations, so that the project team can enhance collaboration, mitigate risks, and ensure a transparent and efficient flow of information throughout the project lifecycle.

<b>What</b>	<b>When</b>	<b>How</b>	<b>Responsible</b>	<b>Approval</b>	<b>Audience</b>
Project Kickoff	Project Start: 10/18/2023 Planning Date: 10/24/2023	Discord	Project Manager	Approved	All team members
Team Meetings	Weekly	Discord and Teams	Project Manager	Approved	All team members
Major Milestone Announcements	As completed	E-mail	Project Manager	Approved	All team members

## 2.8 Risk Management

Risk Management is the process of recognizing, evaluating, and making plans for recently discovered risks. It also assists in monitoring trigger conditions for backup plans, reevaluating current risks, and keeping track of the hazards that have been detected and those that are on the watchlist. To assess their efficacy during risk monitoring and control, the implementation of risk responses is also examined using a risk matrix.

Risk Level	Reasons
Low, Medium, High	Unanticipated intricacies in implementing gameplay mechanics
Low, Medium, High	Unexpected financial limitations
Low, Medium, High	Scope creep
Low, Medium, High	Disengagement and inattentiveness

### 2.8.1 Risk Management Strategy

Software development is a complex task that requires strategies for handling potentially disruptive events. As our team was developing the Blackjack Simulator, the risks that required a mitigation plan, further analysis, or other actions included:

- Unanticipated complexity in gameplay mechanic implementation.
- Unexpected need to use monetary resources.
- Loss of personnel involvement.
- Project scope creep resulting from external feature requests.
- Coding practices that lead to insecure components.
- Problematic probability, database management, and other algorithms.

DG monitored these risks daily, focusing on identifying early warning signs. We maintained a risk log to document identified risks, mitigation actions, and outcomes. Our project status reports included updates on risk management and mitigation activities.

### 2.8.2 Risks Related to Requirements and Other Items

We wanted our Blackjack Simulator application to satisfy the following general requirements, which had associated risks:

- **The software runs on the user's PC as an executable application.** To fulfill this requirement, we developed the application in an Integrated Development Environment (IDE) that can convert our software code into a file the user can easily run, like a .exe file. We agreed to use the Eclipse IDE to develop this project, which allowed us to export programs as runnable Java ARchive (JAR) files, so the risks involved in fulfilling this requirement were low.
- **The application replicates the visuals of a blackjack game through a well-crafted user interface.** Java has several frameworks that help developers create graphical user interfaces, including AWT, Swing, and JavaFX. We have had experience using these frameworks through prior courses at the University of Maryland Global Campus, but those courses restricted assisted/automated UI development. This previous restriction did not prevent us from developing the UI for our Blackjack Simulator. However, it could have easily hindered our efforts had we not located tools to assist us in UI development.
- **The application simulates the interactions and rules within blackjack through game mechanics.** Translating the rules and interactions of blackjack into software system mechanics required sufficient knowledge of the game and the

ability to identify interacting processes. Some of us had never played blackjack and needed to research how to, but considering the amount of online resources, this was not a significant issue for our team. Efficiently translating the rules and interactions into our system required designing understandable, reusable code, which took time to improve.

- **The application securely stores information about the player's game history.**

Our team members have experience designing databases through text files and structured query language. With a functioning Eclipse IDE, file-based databases are somewhat simple to implement but may eventually hinder application performance as they read/write more data. A database management system offers faster data access, security, and integrity. Our team decided to use a relational database management system, so we needed to research ways to implement it into our system.

There were challenges in implementing complex gameplay mechanics and statistical tracking features. Limited budget and personnel could have impacted the project's progress. There was a risk of uncontrolled scope expansion as stakeholders requested additional features. Protecting user statistics is vital, and any breach could lead to legal and reputational risks. Inadequate performance may lead to a poor user experience. DG team members reported all perceived risks to ensure we addressed all shortfalls.

### **2.8.3 Risk Mitigation and Contingency Plans**

DG performed the following activities to mitigate risks:

- Conducted regular code reviews and design consultations to address technical challenges promptly.

- Implemented efficient resource allocation and task prioritization to maximize available resources.
- Assessed changes for alignment with project goals and any significant scope changes that required sponsor approval.
- Maintained compliance with data protection regulations and standards throughout development.

We also conducted performance testing at different stages, allowing early detection and mitigation of issues. By implementing this risk management plan, we minimized the potential impact of risks on the project, ensuring a successful development process for the Blackjack Simulator software with player statistical history. Constant vigilance and proactive risk management were key to achieving our project objectives and delivering a high-quality product.

Table 3: Risk Register

Risk #	Date Identified	Risk Description	Likelihood of the risk occurring	Impact if the risk occurs	Severity (Rating based on impact and likelihood)	Owner (Person who will manage the risk)	Mitigating action
1	20231115	Software not being able to run as an executable application.	Medium	High	High	Project Manager	Doing extensive research into how to create a .jar file and testing it multiple times before releasing it to users.
2	20231106	Blackjack visuals not providing an enjoyable user experience.	Low	High	High	Project Manager	Cross referencing GUI's already used for Blackjack games online that are popular and implementing those ideas into our own project.

3	20231107	Blackjack game logic code not properly representing the rules of the game.	Low	High	High	Project Manager	Ensuring all coding logic is sound and represents the rules of the game  blackjack appropriately and testing it multiple times before releasing to users.
4	20231120	Not properly securing the user's data and game history within the database.	Medium	High	High	Project Manager	Doing research into database construction and encryption to ensure user's can create accounts where they can access their data later on.

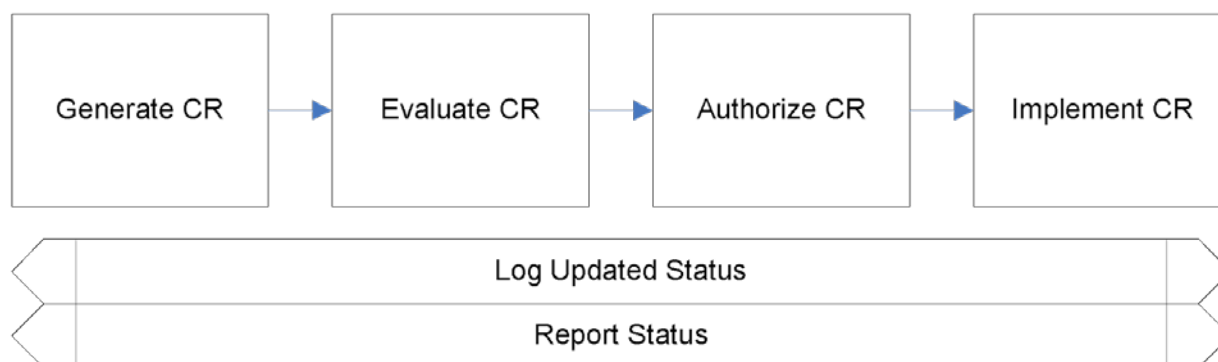
## 2.9 Change Management Plan

The plan outlines procedures to identify, assess, and implement changes while minimizing disruptions.

### 2.9.1 Request Process

Team leaders agreed to initiate change by creating a Change Request (CR), please refer to Appendix G. The project manager will assess and approve or disapprove the CR and send it back within 24 hours of receipt. Team members will keep CRs in our Kanban board for record-keeping. The project manager will notify all stakeholders of its disposition, and software designers will perform code updates using GitHub and the Eclipse IDE if approved. All stakeholders will ensure that all implemented code and changes adhere to the SDLC timeline, and team members will manage any discrepancies or modifications through group code audits. Team members will also showcase modifications made to the code in a Discord channel utilized to monitor inconsistencies and inform the group about their approval status. The project manager

will subsequently record any modifications made to the code to ensure that the entire team is aware of the project's trajectory.



*Change Management Diagram*

### **2.9.2 Generation**

Team leaders or project liaisons will initiate change by generating a Change Request (CR); please refer to Appendix A. Team leaders will post the CR package in the Google Documents folder with the proper file number based on previous submissions, i.e., (CR.request number. significance 1-3).

### **2.9.3 Evaluation**

Evaluate CR: The project manager will evaluate the CR and send all favorable impact assessments to all team leaders for review. The impact assessment will evaluate how the change affects the project's scope, schedule, budget, and quality. This evaluation will prioritize needs over wants to ensure our project meets all phase gates. This methodology will reduce the risk associated with any unnecessary deviations from the initial plan.

### **2.9.4 Authorization**

The digital change request will be distributed to all team leaders for acknowledgment before final disposition. The artifact will provide all stakeholders with



the ability to express concerns while allowing them to plan accordingly. This holistic approach will provide transparency for the final disposition while reducing risk. The project manager will record, report, and notify all team leaders of the final outcome after each team member has reviewed the CR.

### 2.9.5 Implementation

The approved change request will generate a work request that is sent to all relevant stakeholders. This will ensure all related documentation, plans, and processes reflect the change. All work requests will be associated with a CR number and a branch, providing all stakeholders with the ability to track and implement the changes required for their program activity. It prevents the redundancy associated with changes that fall within the scope of another change request. While ensuring all identified changes are tracked and logged.

The code updates will be incorporated into the project's main code via the GitHub repository and the Eclipse IDE after the project manager approves the CR. The other members of the team will then review and audit the code updates to ensure that there are no errors that arise when the program is executed.

Utilizing the Git download for Windows, team members need to clone the repository using the git clone command:

```
tsuna@DESKTOP-49FV48S MINGW64 ~  
$ git clone https://github.com/Khakipapi/CMSC495-Captstone.git  
Cloning into 'CMSC495-Captstone'...  
remote: Enumerating objects: 96, done.  
remote: Counting objects: 100% (96/96), done.  
remote: Compressing objects: 100% (92/92), done.  
remote: Total 96 (delta 5), reused 75 (delta 0), pack-reused 0  
Receiving objects: 100% (96/96), 4.89 MiB | 14.26 MiB/s, done.  
Resolving deltas: 100% (5/5), done.
```

They then need to change the directory to the respective path where that cloned repository is cloned:

```
tsuna@DESKTOP-49FV48S MINGW64 ~
$ cd CMSC495-Captstone/

tsuna@DESKTOP-49FV48S MINGW64 ~/CMSC495-Captstone (master)
$ ls
src/
```

The next step is to check to make sure everything within the cloned repository is up-to-date.

```
tsuna@DESKTOP-49FV48S MINGW64 ~/CMSC495-Captstone (master)
$ git pull
Already up to date.
```

The next step is to create a new branch of code approved via the CR and add it to the overall project:

```
tsuna@DESKTOP-49FV48S MINGW64 ~/CMSC495-Captstone (master)
$ git checkout -B logic/blackjacklogic
Switched to a new branch 'logic/blackjacklogic'
```

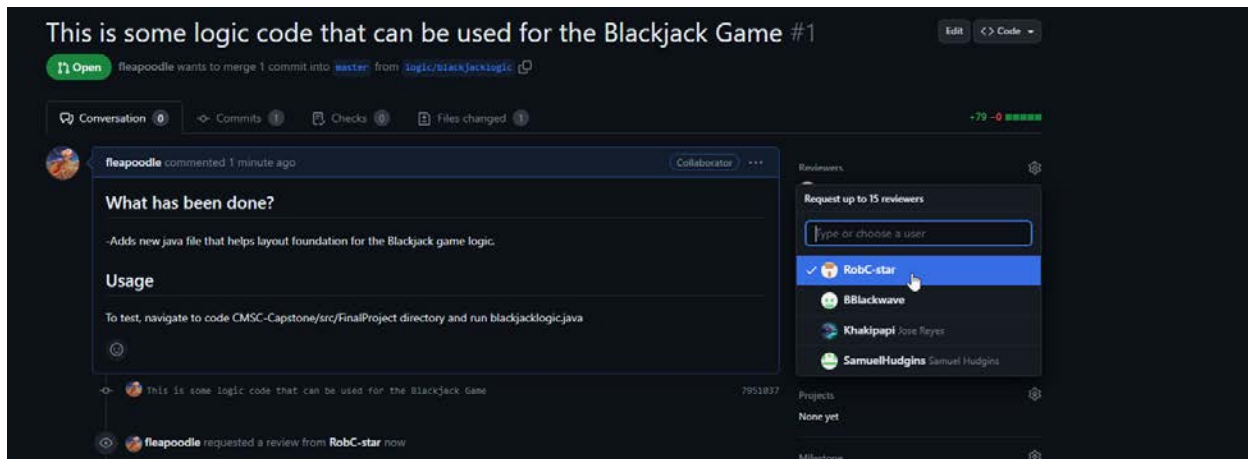
Once the branch is created, we have to create the code that we are going to upload via that branch. In this case, I am adding a simple blackjack logic code that other team members will review:

```
tsuna@DESKTOP-49FV48S MINGW64 ~/CMSC495-Captstone/src/FinalProject (logic/blackjacklogic)
$ git add blackjacklogic.java

tsuna@DESKTOP-49FV48S MINGW64 ~/CMSC495-Captstone/src/FinalProject (logic/blackjacklogic)
$ git status
On branch logic/blackjacklogic
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   blackjacklogic.java
```

Now, we can follow through with the commit using the git commit command, which includes a description of what we will upload:





## 2.9.6 Change Control Board

The Kanban board will host the Change Control Board (CCB), which will allow us to display each CR's status while providing all stakeholders with a means to communicate. The CCB will show the CR history to date and any requests for information that may arise during each event of the CR process. This will provide everyone with a real-time, transparent process.

## 2.10 Time and Cost Management

The development of Blackjack Simulator software with player statistical history was a project that required efficient cost management to ensure that it remained within budget and aligned with our financial goals. This cost management plan outlines key strategies, cost estimation methods, and financial controls.

### 2.10.1 Time and Cost Management and Control

We started with a well-defined initial budget, including resource costs, software development tools, and necessary licenses or permits. We accounted for software development tools, hardware, and infrastructure costs while considering initial purchase and ongoing maintenance expenses.

### 2.10.2 Estimating Project Costs

The costs associated with personnel, such as developers, designers, and testers, were estimated based on hourly rates and anticipated time commitments. For this project, we assumed all project members (Project Manager, Requirements Manager, Test Director, Software Designer, and User Experience Designer) received a salary of \$45.00 per hour. According to the Project Management Institute's Project Management Book of Knowledge (PMBOK), ROM estimates are -25 percent to +75 percent accurate. The following displays the cost analysis for this project using the following as a baseline. It is important to note I included profit in cost since it is unrealized at the time of planning, and runover can effectively reduce the amount however it also provides us with a 35% buffer in this case.

Item	Cost	ROM (-25%, 75%)
5 employees, \$45 per hour @ 40 hours a week for 8 weeks	\$72,000.00	
Insurance: 20% of personal cost	\$14,400.00	
<b>Total personal cost:</b>	<b>\$86,400.00</b>	<b>\$64,800 – \$151,200</b>
Rent, licenses, and equipment cost: \$5000 per month	\$10,000.00	
<b>Total Personal Cost and Equipment:</b>	<b>\$96,400.00</b>	<b>\$72,300 - \$168,700</b>
Profit: 35%	\$33,740.00	

Item	Cost	ROM (-25%, 75%)
<b>Total cost:</b>	<b>\$130,140.00</b>	<b>\$97,605 – \$227,745</b>

### 2.10.3 Tracking and Controlling Actual Project Costs

Each project team leader was responsible for cost management within their respective areas. The project manager monitored and controlled actual time and cost expenditures using an Earned Value Analysis (EVA). The financial budget report tracked the Planned Value (PV), Earned Value (EV), and Actual Cost (AC). At the same time, we maintained transparency in budget management, providing weekly updates to project stakeholders.

### 2.10.4 Analyzing Budgeted and Actual Project Costs

The EVA provided the data required to calculate the Cost Variance (CV), Schedule Variance (SV), Cost Performance Index (CPI), and Schedule Performance Index (SPI). We used the CV to measure costs and cost efficiency and the SV to measure schedule performance and schedule efficiency. At the same time, we used the CPI to identify whether we were under or over budget, while the SPI indicated whether the project was behind or ahead of schedule, as seen in Appendix E.

## 2.11 Quality Management Plan

The purpose of the quality management plan is to ensure the software meets or exceeds the customer's expectations. The planning approach was iterative and incremental and aligned with each development phase. This plan is a living document and will adapt to changes in customer requirements and feedback from the

development team. The quality team is responsible for documenting, passing test cases to the development team, providing user experience feedback, and securing sign-off from developers on all received information.

### **2.11.1 Quality Control Process**

The purpose of quality control is to analyze tests the test team conducts. The quality team will document any defects they discover and report them to the development team. The testing team performed tests early and often and reported defects to the development team after each development phase. The development team had the opportunity to ask for testing out of phase, and the quality and test teams would have provided a report within 24 hours.

### **2.11.2 Product Testing**

The QA team ensures team members address any backlogs of critical and major non-conformance issues identified through testing. The QA team addresses non-conformance in four different categories:

<b>Category</b>	<b>Definition</b>
Critical	Existential risk to customers
Major	Significant risk to the final product or service quality.
Minor	There is no significant risk to the final product or service quality.
Opportunity for improvement	Non-critical, non-systemic.

## **2.12 Development Plan**

DG followed a Software Development Life Cycle (SDLC) framework to help us develop, test, and release our software system. Our SDLC consisted of several week-long phases. During each phase, team members designed, reviewed, and completed project plans and deliverables.

Week	Phase	Deliverables
1	Project Initiation	
2	Project Planning and Requirements	<ol style="list-style-type: none"> <li>1. Project Plan</li> <li>2. Requirements List</li> <li>3. Development Plan</li> </ol>
3	Test Planning & User Guide	<ol style="list-style-type: none"> <li>1. Project Status Update</li> <li>2. Requirements</li> <li>3. Test Plan</li> <li>4. Configuration Management Plan</li> <li>5. User Guide Draft</li> </ol>
4	Design	<ol style="list-style-type: none"> <li>1. Project Status Update</li> <li>2. Requirements Traceability Matrix</li> <li>3. Unit Tests</li> <li>4. Preliminary Design</li> <li>5. UI/UX Review (Wire Diagrams)</li> </ol>
5	Construct	<ol style="list-style-type: none"> <li>1. Project Status Update</li> <li>2. Requirements Traceability Matrix</li> <li>3. Component Tests</li> <li>4. Detailed Design</li> <li>5. Phase I Code Prototype UI/UX Review</li> </ol>
6	Test	<ol style="list-style-type: none"> <li>1. Project Status Update</li> <li>2. Requirements Traceability Matrix</li> <li>3. System Tests</li> <li>4. Physical Design</li> <li>5. Phase II Code</li> <li>6. Final UI/UX Review</li> </ol>
7	Release	<ol style="list-style-type: none"> <li>1. Project Report</li> <li>2. Requirements Specification</li> <li>3. Test Plan and Results</li> <li>4. System Specification Design and Alternate Design</li> <li>5. Development History</li> <li>6. Phase III Code</li> </ol>
8	Final	<ol style="list-style-type: none"> <li>1. Overview, including a summary of individual contributions</li> </ol>

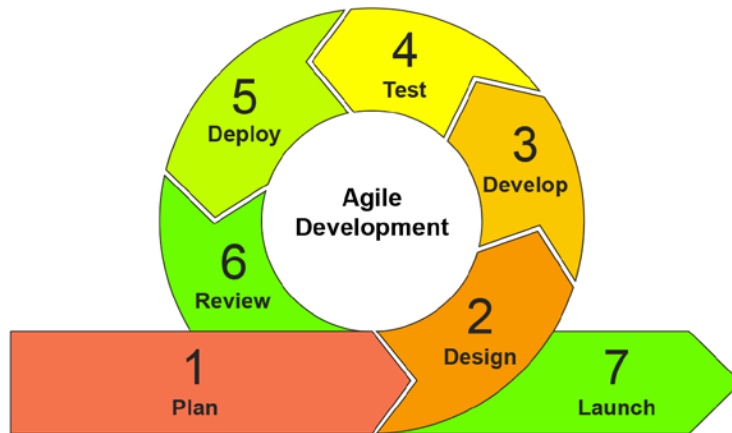


Week	Phase	Deliverables
		2. Project Plan 3. Requirements Specification 4. System Specification 5. User's Guide 6. Test Plan and Results 7. Main and alternate designs 8. Development History 9. Conclusion, including lessons learned, design strengths, limitations, and suggestions for future improvement 6. Submit code as a runnable JAR file

### 2.12.1 Process Model

Our team implemented the agile SDLC model throughout the project. The agile model is a popular framework for software development projects and involves frequent, incremental releases, increased team collaboration, iterative development, and customer involvement. It consists of splitting a project into several periods of intense activity. During each period, the team collaborates to select, design, implement, and test features, then deploys their in-progress project for review. Team members add features and feedback they couldn't implement during the current period into the next one. Over time, the team can quickly add value to the software system by integrating core features. The agile approach offers several benefits for our team.

The approach allowed us to self-organize and determine resource allocations for each initiative. Its emphasis on communication, teamwork, and collaboration increased our productivity and ability to complete deliverables. Its support of frequent reviews and tests decreased the chance of bugs and other issues causing problems within the development process, which led to higher satisfaction with our system.



*Agile development diagram*

We also used Kanban to implement the agile SDLC model. Taiichi Ohno, an industrial engineer, created the Kanban approach at Toyota Motor Corporation to increase manufacturing productivity (Mijacobs et al., n.d.). Kanban is a Japanese term for billboard or signboard; Kanban boards have a table structure and cards. The table columns have names for workflow states (e.g., to-do, in progress, reviewing, and done), and the cards correspond to each product backlog item. As the development team works on a work item, they progress its corresponding card through the Kanban board from the to-do column to the done column. Kanban boards allowed us to communicate the status of our project visually; they provided a way to view the work items we needed to work on, who was working on items, and the progress of items, all in real-time. They also allowed us to control the maximum number of items we worked on through work-in-progress limits, facilitating increased efforts on tasks rather than adding more tasks that divided our attention.

To-do	In progress (1/5)	Reviewing (2/5)	Done
<div>Add credits</div> <div>Add links to sources</div> <div>Add exit screen</div>	<div>Polish user interface</div>	<div>Optimize customer search algorithm</div> <div>Create reset input button</div>	<div>Create Hello Word App Screen</div>

*An Example of Kanban board*

Kanban boards can be physical or virtual. We used a Kanban board through the website Trello. Their interface provides base features, like adding, editing, and arranging table columns and cards, and other helpful features, including adding descriptions, labels, dates, and people to cards. Trello's Kanban board also provides features like using automation for cards and other events, adding buttons to the board or cards, and sending email reports about a board.

### 2.12.2 Technical Requirements

We utilized several software tools and processes while developing our software system:

Software Type	Software Tool
Programming Languages	Java MySQL CSS FXML (XML-based language)
Frameworks	JavaFX
Integrated Development Environments	Eclipse IDE

Software Type	Software Tool
(IDE)	
Configuration Management Software	GitHub
Project Task Management	Trello (Kanban board)
Documentation Storage	Google Drive
Communication	Discord
Other Software Tools and Processes	Draw.io Adobe Illustrator SceneBuilder DB Browser

### 3.0 Requirements Specification

This Software Requirements Document (SRD) records and describes the agreement between stakeholders and the Dynamic Gaming (DG) team regarding the required specification of the software product. It provides a clear and descriptive statement of user requirements that can be used as a reference in further software system development. This document is broken into several sections to separate the software requirements into easily referenced parts.

#### 3.1 Purpose

The purpose of this SRD is to provide a detailed description of the functionalities of the Blackjack Simulator application. This document covers the system's intended features and offers a preliminary view of the software application's User Interface (UI). The document will also cover hardware, software, and other technical dependencies.

#### 3.2 Scope

DG developed the Blackjack Simulator for users interested in playing the Blackjack casino game in a casual yet informative manner. Our application provides

users with information that helps them learn how to play blackjack and win. Our team designed the Blackjack Simulator to run on a personal computer as an executable application. Users can create an account, which allows our application to store their play history and statistics. Afterward, the user can view a tutorial on how to play blackjack, place bets and enter matches against a dealer, or view statistical data about their past games. The Blackjack Simulator also uses data about the player and dealer's cards and the rules, which helps determine the best course of action for the user.

### 3.3 Definitions, Acronyms, and Abbreviations

Term	Definition. Acronym, Abbreviation
Dynamic Gaming	DG - the name of our software development team.
Graphic User Interface	GUI
Personal Computer	PC
Software Requirements Document	SRD

### 3.4 Organizations

This requirements document contains multiple subsections. The first section includes explanations of the document's Purpose, Scope, and Organization. The first section also includes definitions, acronyms, and abbreviations for project-specific terminology we will use in the document. The second section of the document contains

the following five sections, which describe specific details of system uses and their corresponding actions:

- Product Perspective
- Product Functions
- User Characteristics
- Constraints
- Assumptions and Dependencies
- Apportioning of Requirements

The third section is an enumerated listing of the system's requirements. The fourth section includes diagrams that model the system. The final subsection provides a point of contact for anyone viewing this document.

### **3.5 Overall Description**

This section includes details about what we do/do not expect of the Blackjack Simulator, in addition to which cases are knowingly unsupported and assumptions that we used while developing the Blackjack Simulator.

#### **3.5.1 Product Perspective**

The Blackjack Simulator is an offline, single-player game application. It must be available to anyone using a Windows personal computer and provide the same functioning features across computers. There are no hardware or software requirements beyond these, no memory requirements, and no specific software packages we must or need not utilize.

#### **3.5.2 Product Functions**

The Blackjack Simulator provides several functions:

1. It allows users to create and log into the application.
2. It maintains data associated with the user:
  - a. Users have a username and password.
  - b. The application also stores and updates data about user play history, including winnings and losses.
    - i. The user can view this data within the main menu.
3. Provide a tutorial to help new users understand how to play blackjack.

### **3.5.3 User Characteristics**

A user refers to the individual who possesses a copy of our application. Our application expects the user to have an adequate understanding of computer operations, including operating system navigation, keyboard usage, and mouse usage. The user may or may not have experience playing blackjack games.

### **3.5.4 Constraints**

Although security is a concern for this system, the level of security is lower than that of a typical web application. Passwords will not require specific formatting, such as password length and the number of character types, so passwords can be as weak or strong as the user wants. The database stores passwords in encrypted text, but there is no need for a password recovery feature or a lockout system if a user has several invalid login attempts.

### **3.5.5 Assumptions and Dependencies**

Clients: We have assumed that all users can operate our system's basic functions, including but not limited to opening the application and using a mouse or keyboard.

Provider: We assumed the Blackjack Simulator would run on a compatible personal computer.

### **3.5.6 Appportioning of Requirements**

As stated previously, security is a minor concern of this project. As such, it is beyond the scope of this system to monitor, report, or prevent unauthorized login attempts or any other concern of this nature. Additionally, the system is not responsible for allowing users to edit their account details (such as their username or password). Our team may need to extend the system's functionalities for further system developments. One such example is adding support for multiple languages. Our application only supports English text, but adding support for more languages can help additional users navigate the application.

## **3.6 Specific Requirements**

1. Restrictions
  - a. User Side
    - i. Hardware
      - a. Personal Computers
2. Data Structures
  - a. A user has these attributes
    - i. Unique ID (Auto-increasing starting at 1)
    - ii. Username (must be unique)
    - iii. Password
    - iv. Winnings
    - v. Losses
  - b. A card deck has these attributes
    - i. A set of cards



- c. A card has these attributes
  - i. Type
  - ii. Value
  - iii. Image
- 3. System
  - a. Account Authentication
    - i. Account Creation
    - ii. Account Login
    - iii. Verify Login Information
    - iv. Display Login Error
    - v. Delete Account
  - b. Account Management
    - i. Update Bank Account
  - c. GUI Elements
    - i. Selectable GUI buttons
    - ii. Selectable GUI text fields
    - iii. Text labels
    - iv. Background imagery
    - v. Card imagery
  - d. GUI Screens
    - i. Start Screen
    - ii. Main Menu
      - a. Tutorial
      - b. Play game
        - a. Place bet
        - b. Blackjack game scene

- iii. View Stats
- iv. Delete account
- v. Exit game
- e. Blackjack Gameplay
  - i. Hit
  - ii. Stand
  - iii. Double
  - iv. Split
  - v. Insurance
  - vi. Even money

### **3.7 Modeling Requirements**

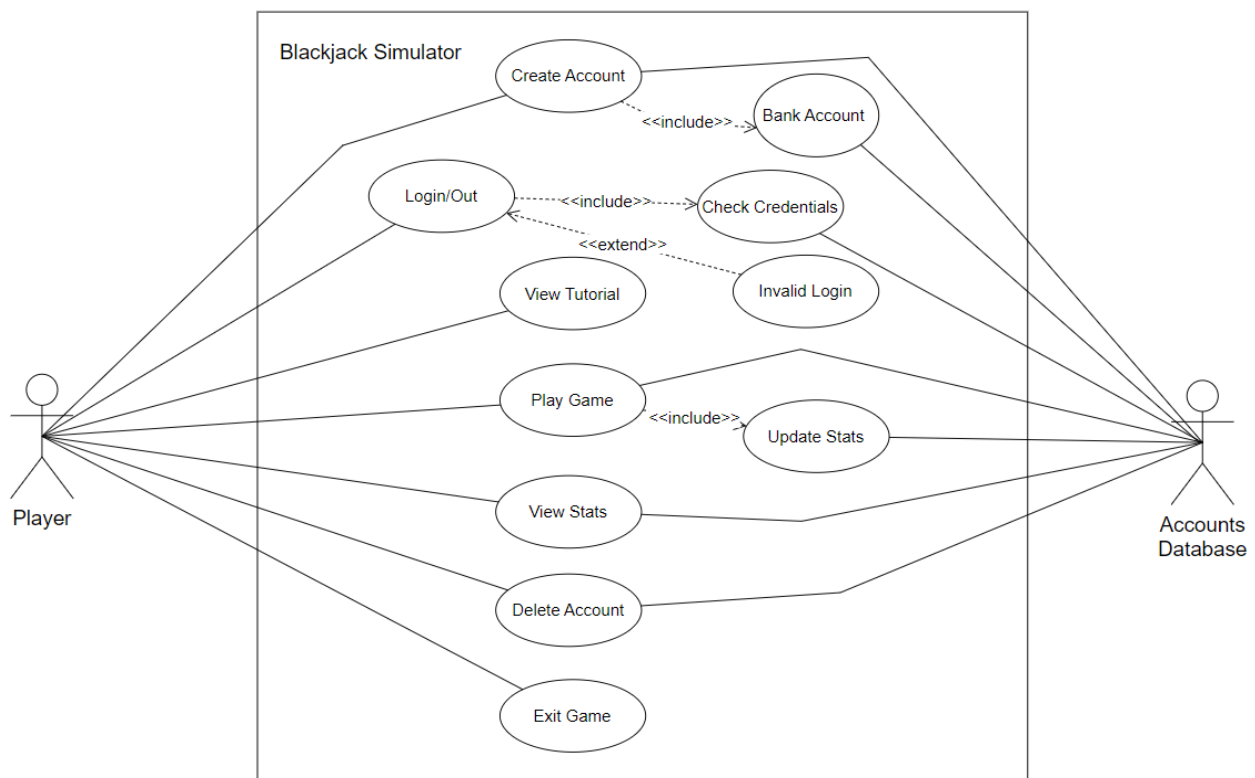
In software engineering, requirements modeling finds the conditions that a software system or application needs to satisfy in order to address the business issue. The purpose of programming models is to describe how a program will be run, whereas the parallel computer model is designed to represent the hardware in abstraction. Between algorithms and real software implementations, the programming model serves as a conduit.

#### **3.7.1 Use Case Diagram**

The purpose of the following use case diagram is to demonstrate how users interact with the Blackjack Simulator and map out the basic functionality of the system. The table below includes descriptions for elements within the diagram, as well as what is in the use case templates that follow.

Items	Description
Actors	Stick figures represent actors with a name underneath them. They represent elements that will be directly interacting with the system.
Use Cases	These are oval shapes that have their names in the center. These represent direct functionality within the system that we must implement.
Interactions	The lines that connect the actors with the different Use Cases. These show forms of direct interaction between the actor and some specific functionality.
Includes	Dotted lines labeled "<>" that connect two use cases and have an arrow pointing towards one. Use cases without an arrow call on the functionality of the use case with the arrow.
Extends	Dotted lines labeled "<>" that connect two use cases and have an arrow pointing towards one. They indicate that a use case without the arrow takes all of the functionality of the use case with the arrow while adding extra functionality.
The System Boundary	The boundary is a large rectangle that contains the Use Cases. Everything within the rectangle is what the system must implement.
Use Case Template	They describe the basic functionality and features of each use case. The templates are provided in the pages following the use case diagram.
Type	A field in the use case template that states whether or not the use case is directly interacted with by an actor (Primary) or not (Secondary), as well as whether or not it is essential to have a functioning system.

Items	Description
Cross References	A field in the use case templates that specifies the original requirement that a particular use case satisfies.
Use-Cases	This field in the use case templates indicates which other use cases must be executed before a particular use case.



Use Case: Create Account

Actors: Player, Accounts Database

Type: Primary, Essential

Description: Initiated when a user wants to create an account for tracking their progress.

Includes: Bank Account

Extends: None

Cross References: Required for 3.a.i

Use-cases: None

Use Case: Login/Logout

Actors: Player

Type: Primary, Essential

Description: Initiated when a user wants to log into an existing account or log out of their account.

Includes: Check Credentials

Extends: None

Cross References: Required for 3.a.ii and 3.a.iii

Use-cases: None

Use Case: View Tutorial

Actors: Player

Type: Primary

Description: Initiated when a user wants to view instructions for how to play Blackjack.

Includes: None

Extends: None

Cross References: Required for 3.d.ii.a

Use-cases: None

Use Case: Play Game

Actors: Player

Type: Primary

Description: Initiated when a user wants to play Blackjack.

Includes: Update Stats

Extends: None

Cross References: Required for 3.d.ii.b

Use-cases: None

Use Case: View Stats

Actors: Player, Accounts Database

Type: Primary

Description: Initiated when a user wants to view their past game stats.

Includes: None

Extends: None

Cross References: Required for 3.d.iii

Use-cases: None

Use Case: Delete Account

Actors: Player, Accounts Database

Type: Primary

Description: Initiated when a user wants to clear their account data, including past game stats.

Includes: None

Extends: None

Cross References: Required for 3.a.vi

Use-cases: None

Use Case: Exit Game

Actors: Player

Type: Primary

Description: Initiated when a user wants to exit the application.

Includes: None

Extends: None

Cross References: Required for 3.d.v

Use-cases: None

Use Case: Check Credentials

Actors: Accounts Database

Type: Secondary, Essential

Description: Initiated after a user attempts to log into an account.

Includes: None

Extends: None

Cross References: Required for 3.a.iv

Use-cases: Login/Out

Use Case: Invalid Login

Actors: None

Type: Secondary

Description: Initiated after a user provides login information that doesn't exist in the database.

Includes: None

Extends: Login/Out

Cross References: Required for 3.a.v

Use-cases: Login/Out

Use Case: Update Stats

Actors: Accounts Database

Type: Primary, Essential

Description: Initiated after a user wins or loses a blackjack match.

Includes: None

Extends: None

Cross References: Required for 3.b.i

Use-cases: Play Game

Use Case: Bank Account

Actors: Accounts Database

Type: Primary, Essential



Description: Initiated after a user creates an account. This gives the user a starting balance.

Includes: None

Extends: None

Cross References: Required for 3.b.i

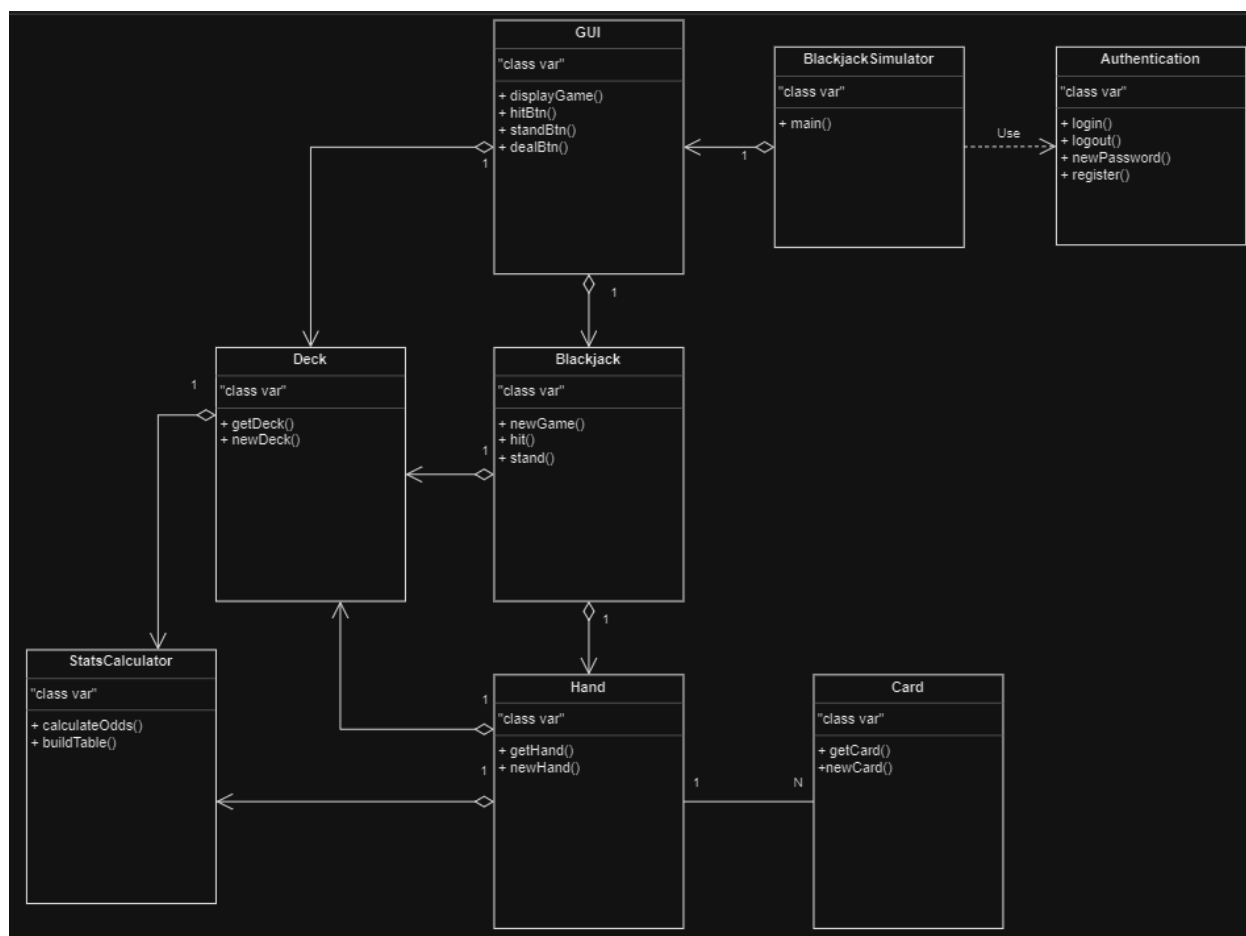
Use-cases: Create Account

### 3.7.2 Class Diagram (Preliminary)

The purpose of this diagram is to show how objects within the blackjack system will interact with each other in order to achieve the functionality required by the Use Case diagram. Below is a list of what you will see in the diagram itself as well as the class descriptions that follow.

Items	Description
Classes	Rectangles in the diagram that are split into three parts. The top section is the name of the class, the middle section is the list of variables that are stored in the class and the bottom section is the list of functions in the class. These rectangles represent objects within the system.
Variables	These have a name followed by a semicolon and then a type. The type denotes what kind of data can be stored in the variable
Functions	These have a name followed by a list of any variable that the function receives in-between the parenthesis "()". After that, there is a semicolon and any variables that the function may return, if none it will be void.

Items	Description
Generalizations	Shown using a line from one object to the other with an unfilled triangle on one end. The object without the triangle inherits the functionality and variables from the object that has the triangle pointing towards it.
Aggregations	Lines that have an unfilled diamond on one end. This means the object with the diamond contains the object(s) without the diamond. This may have numbers on the ends (multiplicities)
Associations	Lines connecting two classes that can have a name beside it, may point in one direction, and may have numbers at the ends (multiplicities). These designate some relationship between the objects. Arrows are simply there to assist you in recognizing which direction the name of the association is read.
Multiplicities	Numbers that may be on the ends of Aggregations and Associations. They state how many of the one object can be related to the other. The first number is the minimum and the second number is the maximum. An asterisk “*” means many, so “1..*” can be read as 1 to many. If no number exists it is assumed to be 1.



### 3.8 Requirements and Features List

ID	Feature	Description	Priority	Dev Estimate (in days) Baseline Scope (Must & Should)	Dev Estimate (in days) Baseline Scope (Other)
R001	Start GUI	The application displays a start screen and allows the user to enter the game.	Must	1	
R002	User account creation and storage	The user can create an account by entering their username and password. Their account information is then saved into the database.	Must	1	
R003	User account login	Allow the user to provide their login credentials and access their account.	Must	1	
R004	Encrypted User Data	Ensures user's data is secure and cannot be stolen.	Should	1	
R005	Saving user money	When the user earns or loses money, save their money value into their account data.	Must	1	
R006	User guest login	The user can skip the process of creating an account. Their money/progress will not be saved.	Considering		1
R007	Betting GUI	The application displays the screen where the user places a bet.	Must	1	
R008	Bet placement	The user can place a bet value.	Must	1	
R009	Game board imagery	The application uses images to represent the Blackjack board and cards.	Must	1	
R010	Game board GUI	The application displays a screen for the Blackjack game.	Must	1	
R011	Cards	The application contains representations of all cards in a deck of 52 cards.	Must	1	
R012	Dealer	The game includes a representation of a Dealer.	Must	1	
R013	Dealer hit on stand	The dealer hits if the player stands.	Must	1	
R014	Dealer hit	The dealer continues hitting until their	Must	1	

ID	Feature	Description	Priority	Dev Estimate (in days) Baseline Scope (Must & Should)	Dev Estimate (in days) Baseline Scope (Other)
	below 17	deck value exceeds 17.			
R015	Hitting	When the user hits, a card is removed from the combined deck of cards and given to the user.	Must	1	
R016	Standing	When the user stands, the Dealer's second card is revealed.	Must	1	
R017	Bet doubling	The user can double their bet after receiving their first two cards. This will cause the user to Hit and an immediate Stand.	Must	1	
R018	Splitting	The user can perform a split if they receive two cards of the same value. This splits the user's cards into two separate hands with equal bets, and each hand receives an additional card. Each hand is played separately.	Must	1	
R019	Insurance	If the dealer's face-up card is an ace, the user can purchase insurance. As a result, the user bets half their original bet (in addition to it) that the dealer does have Blackjack. If the dealer does, the insurance is paid 2 to 1, and the user's original bet is lost (so they break even for the hand). Otherwise, the user loses their insurance.	Must	1	
R020	Even Money	If the user has Blackjack and the dealer has an ace showing, the dealer offers the user even money for their Blackjack (instead of 3 to 2). If the user declines it and the dealer also has Blackjack, the user will have a push just like normal.	Must	1	
R021	Win condition (higher hand)	The user wins if their hand's total is higher than the dealer's.	Must	1	
R022	Win	The user wins if the dealer busts.	Must	1	

ID	Feature	Description	Priority	Dev Estimate (in days) Baseline Scope (Must & Should)	Dev Estimate (in days) Baseline Scope (Other)
	condition (dealer bust)				
R023	Lose condition (21+)	The user loses if their hand's total exceeds 21.	Must	1	
R024	Lose condition (lower hand)	The user loses if their hand's total is less than the dealer's.	Must	1	
R025	Push condition	The user receives their bet back if they and the dealer both have a hand total of 21.	Must	1	
R026	Game over GUI	When the user loses all of their money or closes the application, a game over screen is shown.	Must	1	
R027	Favorable odds	During gameplay, the GUI indicates whether the user should hit or stand.	Must	1	
R028	Total hand values	During gameplay, the GUI displays the total value of the player and dealer's hands.	Should	1	
R029	UI sound effects (GUI)	Menu UI elements generate sound feedback when selected.	Considering		1
R030	UI sound effects (Chips)	Chip UI elements generate sound feedback when selected.	Considering		1
R031	UI sound effects (Cards)	Card UI elements generate sound feedback when moving.	Considering		1

## 4.0 User Guide

The user guide provides a walkthrough for users to help them understand and use the application. Our user guide explains what our application is, what hardware and

software the user needs to operate it, and how to navigate through it once they open it. It includes guides for creating, logging into, and deleting an account, playing blackjack and understanding its rules, and viewing the tutorial and stats. For the complete user guide, please see Appendix I.

## 5.0 Test Plan and Results

### 5.1 Objectives

The testing objective is to ensure the proper functionality of the Blackjack Simulator and to ensure there are no critical or major defects in the application. Additionally, the tests will ensure all customer requirements are met and the application provides the desired user experience.

### 5.2 Scope

The test plan tests the Graphic User Interface (GUI) to determine if the functionality is simple and intuitive for users. It tests any inputs and outputs made through the GUI to ensure the information is displayed appropriately. Additionally, this plan tests all interfaces with databases to ensure data integrity is maintained.

#### 5.2.1 Features to Test

##### Application Features Tested

- Java UI
  - Tutorial
    - Tutorial Button
    - Tutorial Text
  - Login Page
    - Username
    - Password Field
    - Submit Button
  - Account Creation
    - User inputs
      - Username Field
      - **Confirm Username Field**
      - Password Field

- **Confirm Password Field**
  - Submit Button
- Delete Account
  - Username
  - Password
  - Delete Button
  - Confirm Delete
- Game Play User
  - Start Game Button
  - Stand Button
  - The user bets money (using fixed amounts)
  - Hit Button
  - Double Button
    - The user receives their first two cards; this will cause a hit and an immediate stand.
  - Spit Button
    - The user can split cards if they receive two cards with the same value, allowing the user to play two hands separately.
  - Insurance Button
  - Even Money
    - The user has a blackjack, and the dealer has an ACE showing; the dealer offers the user even money for their blackjack.
  - Win
    - The user has a higher hand.
    - Dealer Bust
  - Lose
    - User hand higher than 21
    - hand lower than the dealer
  - Push condition
    - The dealer and user have the same hand.
  - Game over
    - When the user loses all the money
  - Favorable Odds
    - GUI displays best option for winning
  - Exit Game Button
    - A message displays that the user has exited the game
- Game-Play Dealer
  - The dealer hits when the player stands.
  - The dealer hits until their deck value exceeds 17.
  - When a player hits, a card is removed from the deck.
  - When the user stands, the dealer's second card is revealed.
- Game-play imagery
  - Cards are easy to see and understand.



- The player and dealer's hands are represented.
- User input
  - Test users can not bet more than what is in the bank
    - The user is told that the amount is more than the bank amount.
- Database
  - Player Table
    - Username
    - Unique ID
    - Password
  - Bank Table
    - Balance
      - Earnings/Losses
  - Table Score Table
    - Unique I.D.
    - Score
    - Rank
  - Delete Account
    - Confirm profile deleted from player table.
    - Confirm the deletion of the profile from all tables.

## 5.3 Types of Testing

### 5.3.1 Black Box Testing

We conducted black-box tests on the Blackjack Simulator to ensure all functionality operates as expected. The team used a set of inputs to ensure the expected outputs were received. The team documented the functionality and performance of the application and reported any issues or suggestions to the project manager and development team. Black Box testing is performed without knowledge of a system's internals and can be carried out to evaluate the functionality, security, and performance (Michali, 2022). This is important to testing to ensure the system performs as expected from a user and development perspective.

### 5.3.2 GUI Testing

GUI testing refers to the validation of UI functions or features of an application that are visible to users, and they should comply with business requirements. GUI testing is also known as UI testing. That means “User Interface testing” (Ganguly, 2023)

The team’s GUI tests involved:

1. Testing the size, position, height, and width of the visual elements.
2. Verifying and testing whether any error messages are displayed or not.
3. Testing the navigation elements within the GUI to ensure they worked as desired.

The purpose of testing our application’s GUI is to ensure the application meets all the customer requirements and provides users with an intuitive experience.

### 5.3.3 White Box Testing

The database underwent a comprehensive evaluation, encompassing multiple dimensions of scrutiny. This examination included assessing the database’s internal structure to ascertain its alignment with the application's intended purpose and the validation of data accuracy, integrity, and consistency. The database houses essential user information, encompassing usernames, passwords, and bank balances, with an additional emphasis on verifying the encryption of stored passwords.

White box testing is a form of application testing that provides the tester with complete knowledge of the application being tested, including access to source code and design documents. This in-depth visibility makes it possible for white box testing to identify issues that are invisible to gray and black box testing (Michali, 2022). Access to the source code while conducting tests ensures the proper code is in place to secure

the database. The tester looks for code that provides salting and hashing for passwords and code that uses prepared statements when accessing the database to provide security against SQL injection.

#### **5.3.4 Non-Functional Test**

Non-functional requirements, or NFRs, are a set of specifications that describe the system's operation capabilities and constraints and attempt to improve its functionality. These requirements outline how well it will operate, including speed, security, reliability, data integrity, etc.

Although security is not a primary concern, since the project focuses on a standalone application, we checked its susceptibility to SQL injection attacks, as such attacks could damage the database and crash the application. Additionally, we ensured the application did not store any plain-text passwords in the database during testing, which would allow unauthorized access (e.g., username: admin, password: 123). If the application is due to go online in the future, it is structured to handle basic attacks.

Testers checked data integrity to ensure data remained consistent and accurate as the application processed data from user input and other internal processes into the database. Testers compared their input into the application's GUI against the database to ensure the application correctly updated tables.

#### **5.3.5 User Acceptance Testing**

The team conducted in-house testing to evaluate the overall application experience before advancing to the acceptance testing phase. During acceptance testing, the client assesses the product's functionality, and if it meets their expectations,

the project progresses to the final stages of production. The features below are included in acceptance testing.

Test Case ID	Function to be tested	Purpose
Acceptance Test #1	User Signup	Verify users can create a username and password.
Acceptance Test #2	Login	Verify user can log into the system and play the game.
Acceptance Test #3	Delete Account	Verify user can delete an account.
Acceptance Test #4	Tutorial Screen	Verify tutorial screen is accessible
Acceptance Test #5	Betting	Verify user can bet any amount in their bank account.
Acceptance Test #6	Stand Button	Verify user does not receive another card and the dealer shows their hand.
Acceptance Test #7	HIT	Verify user receives another card from the dealer.
Acceptance Test #8	Double Button	Verify user can double the bet.
Acceptance Test #9	Split Button	Verify user can split cards only when they are of the same value.
Acceptance Test #10	Insurance Button	Verify the user can take insurance when the dealer has an Ace face up.
Acceptance Test #11	Push Condition	Verify push is given when the user and dealer have the same value cards.

Test Case ID	Function to be tested	Purpose
Acceptance Test #12	Game Over	When a user loses all money, the game is reset to \$1000.00
Acceptance Test #13	Exit Game	The game closes once the user chooses to exit.
Acceptance Test #14	View Stats	Verified user can see winnings and losses as compared to other users.
Acceptance Test #15	Play as Guest	Verify users can play as a guest.

#### *Acceptance Testing*

### **5.3.6 Test Documentation**

Each test was thoroughly documented, with outcomes classified as passed or failed. The test case documentation comprises a test number, test steps, input data, expected output, pass/fail result, and an impact rating denoted as critical, major, minor, or opportunity for improvement (OFI).

Test #	Input	Expected Output	Actual Output	Pass/Fail	Impact
1					

*Example of a Pass/Fail table for test documentation*

Please refer to appendix J for the test case table and appendix K for the final test cases and results.

### 5.3.7 Defect Documentation

Bugs found during testing will be tracked and routed to the project manager and the development team. See the user form below for tracking and the description of each column header. All column headers are as follows:

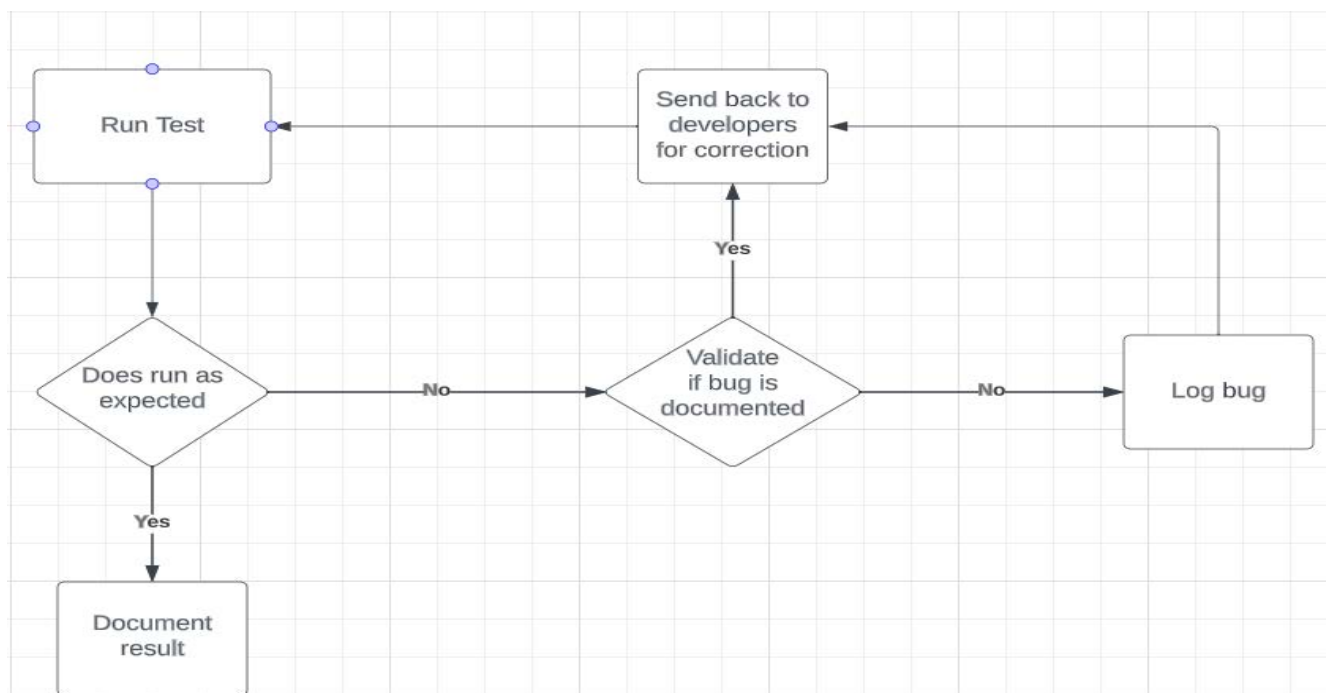
1. The defect ID is used for tracking purposes and will be constructed in the following format:
  - a. The first part will always be **bug\_**.
  - b. The next part will be the area where the bug was found (i.e., the login part of the program), and as a result, will be **bug\_login**.
  - c. The third portion is the incremental number **001,002, etc.**
2. A defect title is a name assigned by the person who found the bug.
3. Defect Description
  - a. Includes a detailed description highlighting the process and its expected and actual outcomes.
4. Steps to reproduce:
  - a. Describe a step-by-step sequence of events to reproduce the error.
    - i. Logged on to the main menu
    - ii. Selections show high scores.
  - b. **Expected Result:** Show a list of high scores.
  - c. **Actual Result:** The table returns with an error.
5. The severity of the bug:

- a. **Critical:** Existential risk to customers that causes a crash or severe data loss.
  - b. **Major:** Significant risk to the final product or service quality.
  - c. **Minor:** There is no significant risk to the final product or service quality.
6. Bug assignment priority:
- a. Urgent: Must be fixed immediately
  - b. Medium: Fix before the application deployment
  - c. Low: Not urgent and can be fixed at developers' convenience
7. Found By:
- a. The name and contact information of the individual who found the bug
8. Assigned To:
- a. The team responsible for fixing the bug

Defect ID (i.e. bug_login_0_1)	Defect Title (i.e. Username not found)	Defect Description	Steps to Reproduce Bug	Severity	Priority	Found By	Assigned To

*Bug tracking and documentation table*

The chart below shows the flow of bug documentation and correction. Any bugs caught during testing will flow back to the development team for correction. This pattern will continue until the bugs are corrected.



*Bug flow testing and documentation*

All defects will be logged and tracked on the document above and stored in the Google shared drive for all team members to view. Defects will be assigned to the appropriate personnel and checked during testing to ensure the issue is no longer present.

#### **5.4 Entry and Exit Criteria for Testing**

Before proceeding to the testing step, it is necessary to fulfill the entry criteria. Otherwise, the project team will need to assess any risk that may occur if they are not satisfied. The testing team documented all test cases as pass/fail and all bugs at a minimum before exiting the test phase. Each testing phase will have a new entry and exit cycle to give the test team the platform to assess the previous test cycle and test any critical or major bugs and corrections. At a minimum, there will be three testing cycles.



Entry Criteria	NOTE
Testing Environment Available	
Code unit Tested by developers	
Test scripts are developed and approved.	
Test Data Prepared	

*Entry Criteria for Testing*

Exit Criteria	NOTE
All test executions are complete.	
The application passes 95% of the tests.	Before Deployment
All Critical and Major Bugs Resolved at %100	Before Deployment
All remaining bugs will be documented and fixed in future releases	Before Deployment
All test cases documented as Pass/Fail	

*Exit Criteria for Each Phase and Final Phase*

## 5.5 Environment Requirements for Testing

The application runs on any standard Windows 10 operating system and has a lower level of security since it runs as a standalone application on the desktop.

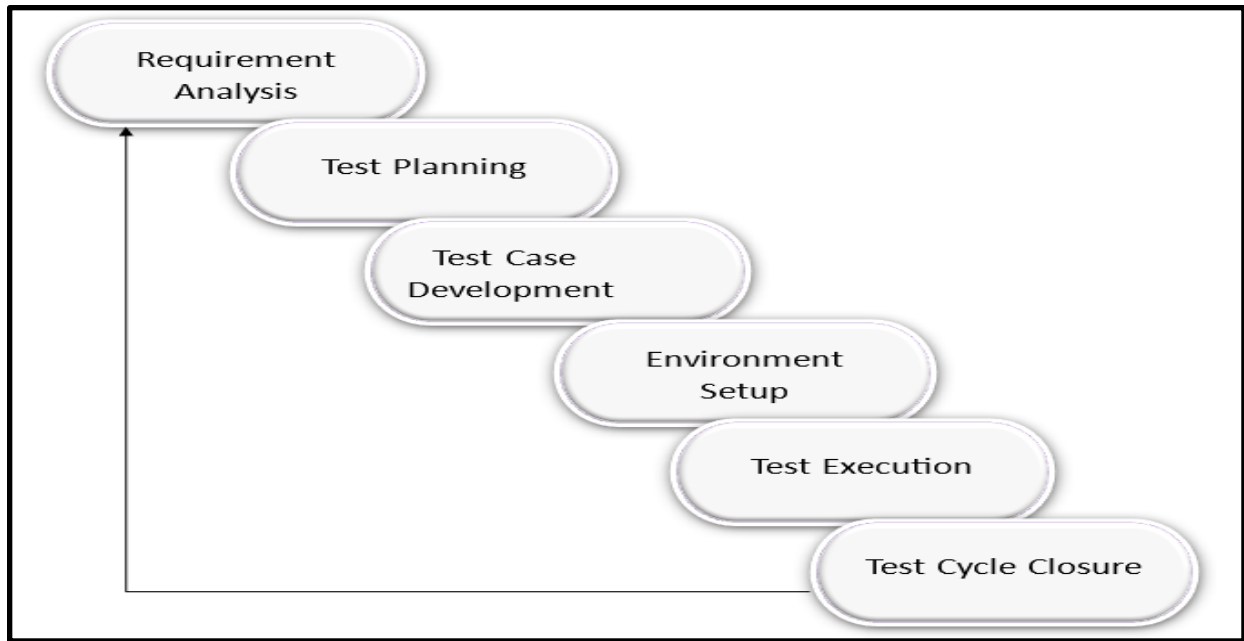
## 5.6 Testing Schedule and Testing Cycle

Outlined below is the test schedule for this project. The Test Director is responsible for implementing this schedule and ensuring that it meets the development team's needs. This schedule is a living document and could change as needed.

Task Name	Duration ⓪	Start	Finish	Predecesso...	Assigned To	Allocation %	% C	Oct				Nov				Dec												
								Oct 1	Oct 8	Oct 15	Oct 22	Oct 29	Nov 5	Nov 12	Nov 19	Nov 26	Dec 3	Dec 10	Dec 17	Dec 24								
Project Initiation and Requirements																												
Project Charter	10d	10/18/23	10/31/23		Carswell, Robert																							
Project Charter Revisions	10d	10/18/23	10/31/23		Carswell, Robert																							
Project Scope	10d	10/18/23	10/31/23		Carswell, Robert																							
Schedule (Gantt)	5d	10/25/23	10/31/23		Reyes, Jose																							
Cost	10d	10/18/23	10/31/23		Carswell, Robert																							
Development Plan	2d	11/26/23	11/27/23		Hudgins, Samuel																							
Project Plan	5d	10/25/23	10/31/23		Carswell, Robert																							
Requirement List	3d	10/27/23	10/31/23		DeLaGarza, Jordan																							
Project Test Planning and User Guide																												
Project Status	5d	11/01/23	11/07/23		Carswell, Robert																							
Risk Management update	5d	11/01/23	11/07/23		Carswell, Robert																							
Project Scope Revised	5d	11/01/23	11/07/23		Carswell, Robert																							
Schedule (Gantt) Revised	5d	11/01/23	11/07/23		Carswell, Robert																							
Management plan Revised	5d	11/01/23	11/07/23		DeLaGarza, Jordan																							
Test Plan	5d	11/01/23	11/07/23		Smith, Patrick																							
Schedule	5d	11/01/23	11/07/23		Smith, Patrick																							
Cost	5d	11/01/23	11/07/23		Smith, Patrick																							
Test Cases	5d	11/01/23	11/07/23		Smith, Patrick																							
Configuration Management Plan(Github)	5d	11/01/23	11/07/23		Reyes, Jose and Hudgi																							
User Guide (Draft)	5d	11/01/23	11/07/23		DeLaGarza, Jordan																							
Project Design																												
Project Status	5d	11/08/23	11/14/23		Carswell, Robert																							
Risk Management update	5d	11/08/23	11/14/23		Carswell, Robert																							
Project Scope Revised	5d	11/08/23	11/14/23		Carswell, Robert																							
Requirements Traceability Matrix	5d	11/08/23	11/14/23		DeLaGarza, Jordan																							
Preliminary Design	5d	11/08/23	11/14/23		Reyes, Jose and Hudgi																							
Wire Diagram	5d	11/08/23	11/14/23		DeLaGarza, Jordan																							
Project Construct																												
Project Status	5d	11/15/23	11/21/23		Carswell, Robert																							
Risk Management update	5d	11/15/23	11/21/23		Carswell, Robert																							
Project Scope Revised	5d	11/15/23	11/21/23		Carswell, Robert																							
Requirements Traceability Matrix	5d	11/15/23	11/21/23		DeLaGarza, Jordan																							
Component Test	5d	11/15/23	11/21/23		Smith, Patrick																							
Detailed Design	5d	11/15/23	11/21/23		Reyes, Jose and Hudgi																							
Phase I Code	5d	11/15/23	11/21/23		Reyes, Jose and Hudgi																							
Prototype	5d	11/15/23	11/21/23		DeLaGarza, Jordan																							
Phase I Test	3d	11/19/23	11/21/23		Smith,Patrick																							
Project Test																												
Project Status	5d	11/22/23	11/28/23		Carswell, Robert																							
Requirements Traceability Matrix	5d	11/22/23	11/28/23		DeLaGarza, Jordan																							

### Test Schedule

The testing schedule and cycle ensure all team members are familiar with previous and current requirements and what new phases will accomplish. Each cycle and out-of-cycle test proceeds through the following phases (Admin, 2016):



*Testing cycle diagram*

Within the requirement analysis phase, the test team comprehensively examines the project's requirements to understand what they must test for each cycle. Additionally, the team revisits previous cycles, identifying any tests needing reevaluation. This reexamination is prompted by the discovery of bugs in earlier cycles, necessitating a verification process to confirm that the successfully resolved identified issues.

After reviewing the requirements, the team starts the test planning phase. During this stage, the team defines the objective and scope of the testing phase, lists testing types for the phase, and defines the test environment for the specific phase (Admin, 2016). Once the team solidifies the plans, they prepare the test cases.

The team prepares test cases once the planning phase is complete. During this phase, the team writes detailed test cases (Admin, 2016). Additionally, during this phase, the team prepares any required test data. The team reviews the test cases

before moving on to the environmental setup. During the environmental setup, the team ensures all hardware and any additional required software are in place (Admin, 2016).

Once cases are prepared and the environment is set, the team can start test execution. The test team executes all prepared cases and documents them with a pass or fail using the document in Appendix K. The team will use the bug tracking and documentation table from section 5.3.7 if any bugs are found during testing. Once all testing is complete, the team closes out the current phase of the testing cycle.

In the last phase, the testing team makes certain all tests are complete and, if not, documents the reason why. Additionally, the testing team guarantees that the test execution report is complete and approved by the development team. Also, the testing team ensures the bug/defect report is complete, and all the bugs and defects have the appropriate priority assigned to provide direction to the development team.

The testing team may suspend tests for a few reasons, such as critical failure of a specific component necessary for other functions to operate. Additionally, the testing team may suspend tests if they discover broken environmental factors during test execution. Once issues are corrected, the test will resume from the point of suspension.

#### **5.6.1 Item Transmittal Report**

At the end of each development phase and before the Test Planning Cycle starts, the development team submits a Test Item Transmittal Report outlining the items released for testing. The report will be available in the team's Google Drive folder for access to testers. The transmittal report has the following information:

- a. **Transmittal Report identifier:** Is a unique identifier assigned to this transmittal report (e.g., TR0001)

- b. **Transmitted Item:** Provide reference to the item documentation and provide a test case identifier if available.
- c. **Status:** Will refer to the version of this test item.
- d. **Approval:** The name of the developer who has released this item for testing.

Transmittal Report identifier	Transmitted Items	Status	Approval

*Transmittal Report*

## 5.7 Test Metrics

The test team used the following metrics to test the progress of testing efforts. The test team chose each metric to measure the software's progress and the test team's testing efficiency. Below are the different metrics they used on each test report at the end of the testing cycle.

### 5.7.1 Passed Test Cases Percentage

This is the number of test cases that passed during the particular testing cycle and is calculated in the following manner (Hamilton, 2023).

- Passed Test Cases Percentage = (Number of Passed Tests/Total number of tests executed) X 100

### 5.7.2 Failed Test Cases Percentage

This is the number of cases that failed during the particular testing cycle and is calculated in the following manner (Hamilton, 2023).

- Failed Test Cases Percentage = (Number of Failed Tests/Total number of tests executed) X 100

### 5.7.3 Critical Defects Percentage

This metric calculates the percentage of defects in the software by dividing the number of total defects by the number of critical defects (Hamilton, 2023).

- Critical Defects Percentage = (Critical Defects/Total Defects Reported) X 100

### 5.7.4 Test Count

This metric calculates the number of tests run within a time period. This metric is calculated in the following manner (Hamilton, 2023).

- Tests run per time period = Number of tests run/Total time taken

## 5.8 Potential Risk

The following are potential risks involved in executing the test plan:

Risk	Mitigation
The listed features tested do not cover enough of the application.	Review all the features continuously to see if test coverage needs expansion.
All tests may not be completed during the testing cycle.	Only test critical components that failed on the last cycle.
Insufficient Resources	Automate some tests if the application becomes too large.

*Potential Risk table*

## 5.9 Security Considerations

Considering this is a standalone desktop application, no significant security concerns exist. However, to keep the integrity of the database and application, certain

features have been given to guests and users. Referring to NIST 800.53 AC-2 Account management, users can manage accounts for which they have passwords. Account management includes viewing their profile and deleting that account with the proper passwords.

To protect users on the system from profile hijacks, we will follow NIST 800.53 IA-5 and salt and hash passwords stored in the database, which helps protect user profiles from being manipulated by others and helps create a fun, competitive experience.

### 5.10 Training Considerations

Team members had varying software development experience going into this project, so we took the following actions to help mitigate training issues and leverage the team's full capabilities.

Training Issue	Mitigation
Testers not familiar with Java	Have training course to get Testers familiar with the basics
Testers not Familiar with Database testing.	Formulate step-by-step instructions for testing the database and what to look for.

*Training Considerations*

### 5.11 Roles and Responsibilities

The roles and responsibilities below outline the individual duties of the team members to ensure communication and cohesiveness before, during, and after each test cycle. The test plan is a living document, and as discovered, other tasks will be added to this list and assigned to the appropriate team.

Role	Responsibilities
------	------------------

Project Manager	1. For the overall success of the project.
Test Director	1. For developing test cases. 2. Produce a testing report after each phase. 3. Readjust the testing schedule as necessary. 4. Ensure all critical and major bugs are corrected prior to deployment. 5. Review all test cases 6. Prepare Test Data 7. Determine if the environment is ready.
Software Developers	1. Inform the test team when software is released for testing in each phase. 2. Communicating with the test team environment is ready for testing.

#### *Roles and Responsibilities*

## **6.0 Design and Alternate Designs**

This section describes the software system architecture and design of Blackjack Simulator and provides stakeholders, the development team, and users with an understanding of our application's current design and functionality.

### **6.1 Purpose of the Software Design Specification**

A product design specification defines a product or service's requirements, features, and functions ([www.linkedin.com](http://www.linkedin.com)). In software design, the Software Design Specification (SDS) details how the development team will translate agreed-upon requirements and features into a structured software system. It provides the development team with the underlying code and data structures they must integrate and connect to develop the final system. The SDS also supplies training material for new project members through its software system details and provides evidence that the



software developers are following through on their commitment to implementing the functionality described in the requirements specification (Zuci Systems).

## **6.2 General Overview and Design Guidelines/Approach**

This section describes the principles and processes we followed while designing and implementing the software system of Blackjack Simulator. Here, we discuss our system's scope, assumptions about it and its use, any constraints we and stakeholders placed on it, and the design precedents we followed while developing it.

### **6.2.1 Scope**

Our project produces a comprehensive blackjack experience that allows users to play simulated games of blackjack while maintaining a record of the player's statistical history. Our primary objective was to create a user-friendly software application that provides players with an engaging and educational experience while offering a statistical history feature to track and analyze gameplay performance. The scope of our project included:

- Developing a menu-driven Graphical User Interface (GUI).
- Translating the rules and interactions of blackjack into a software application.
- Presenting information to help the user learn how to play blackjack.
- Designing a system for calculating/displaying optimal courses of action during blackjack matches.
- Implementing the basis for account registration and login.
- Setting up and utilizing a database to store and manage user information.

### 6.2.2 Assumptions

We made several assumptions about the system and its users as we designed and developed the Blackjack Simulator application. Here, we refer to the system as the device and its operating system on which a user runs our application and the user as any individual who possesses a copy of our application. We assumed:

- The user's system was a personal computer (PC), such as a laptop or desktop computer, which operates on Windows 10 and supports a built-in or external display and mouse and keyboard input.
- The user's system would have the necessary software to run our application.
- The user would use a functioning built-in or external monitor with their PC and understand how to use their operating system, including navigating its GUI with a mouse and keyboard.
- The user will interact with our application with a mouse and keyboard.

### 6.2.3 Constraints

Like other software projects, we developed our software system while respecting stakeholders' deadlines. The software development phase for our project spanned only three weeks, so we observed several design constraints as we developed our project:

- Regarding device compatibility, we limited our software application to run on Windows 10 devices specifically.
- We also constrained our software application from running with or across a network connection, so it had to operate as a standalone desktop application.
  - Due to this constraint, the application's database had to exist on the user's PC.

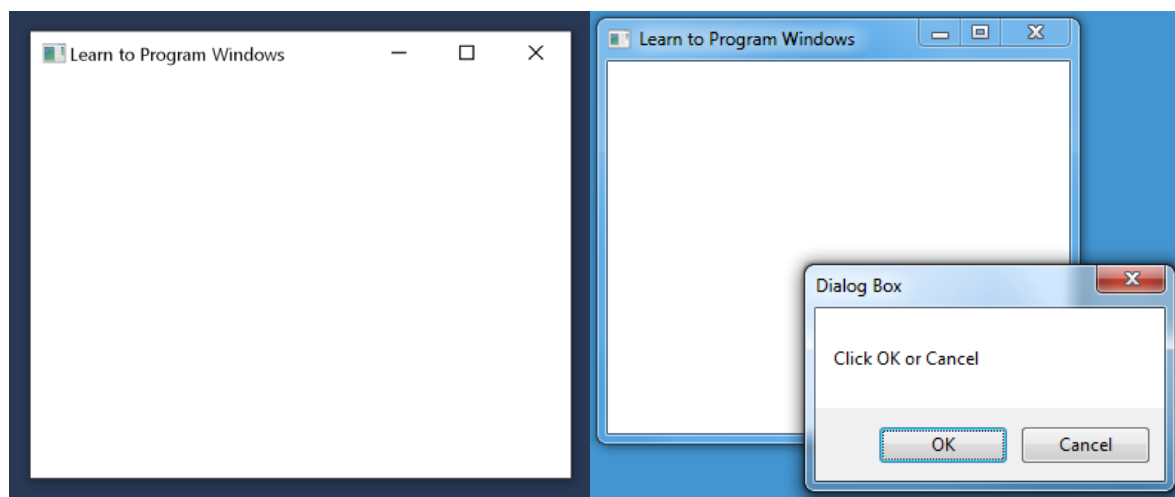
- Although security was a concern for our system, the level of security we required was lower than that of a typical web application.
  - Other than prohibiting blank passwords, our application would not include functionality for specifying password formats, such as password length and the number of character types, so users can create weak or strong passwords as they wish.
  - We also would not include a password recovery feature or a lockout system if a user has several invalid login attempts.

#### **6.2.4 Design Precedents**

Our team located, compared, and utilized software design precedents to facilitate our efforts in developing our software application. From a high-level perspective, we designed a blackjack video game with a GUI, a database for storing, managing, updating, and viewing user account information, a tutorial, and a system for displaying optimal courses of action during blackjack matches.

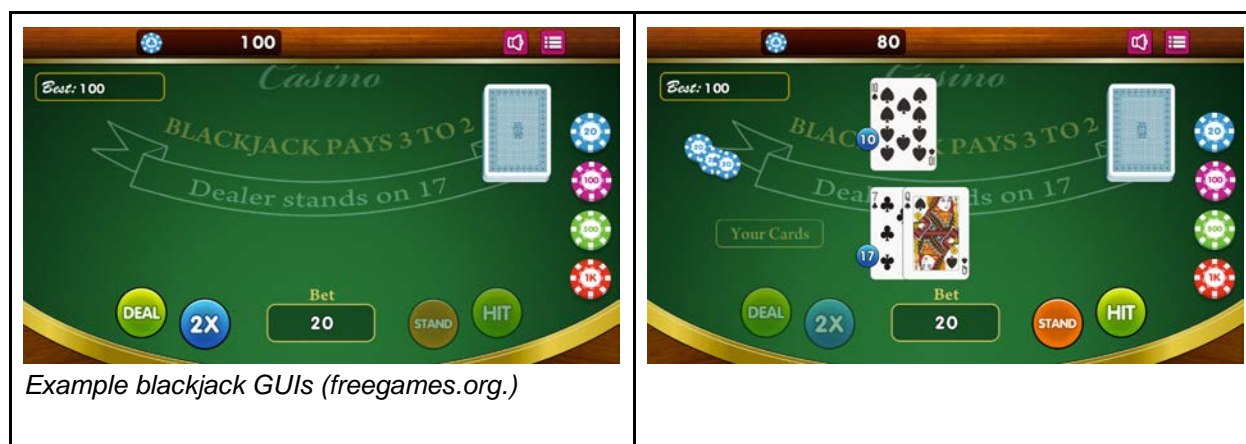
##### **6.2.4.1 GUI**

Most application GUIs reside in an application window consisting of a frame with a title bar, a minimize and maximize button, and other UI elements. Below are examples of Windows operating system application windows:



*Example GUI application window (QuinnRadich et al.)*

Our GUI needed to emulate other typical GUIs' visual cues and experiences. For instance, many applications that include GUIs also include buttons. These GUIs indicate whether the user can select a button and when the user selects a button through graphical coloring, animations, and possibly sounds. Our application also uses imagery associated with the blackjack experience, such as the game board, cards, and chips. Below are the example blackjack game GUIs we used to develop our GUI elements:



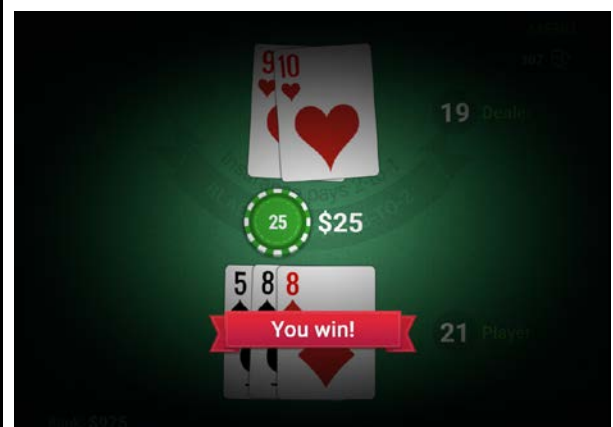
*Example blackjack GUIs (freegames.org.)*



Example blackjack GUIs (Relax Gaming)



Example blackjack GUIs (WP Company)



#### 6.2.4.2 Account Registration and Login

Account registration and login screens typically reside within an application's window, including text fields, selectable GUI elements to which the user can input keyboard data, and buttons for submitting information in fields. These screens usually

include fields for the user to enter personal data, including their name, email, and password. Account registration screens typically require more information from the user than login screens since the user provides defining information about the account the system will store for them. Login screens may only ask for the user's name/email and password since the user's information will presumably be in the system, so the application can use the user's provided information to verify it's in the system. Below are examples of the registration and login screens on the website Squarespace:

The image displays two side-by-side screenshots of the Squarespace website's account creation and login interface.

**Left Screenshot: Create Your Account**

- Title:** Create Your Account
- Fields:**
  - FIRST NAME: First Name
  - LAST NAME: Last Name
  - EMAIL ADDRESS: name@example.com
  - PASSWORD: Password (with an eye icon for toggling visibility)
- Text:** By creating an account, you agree to our [Terms of Service](#) and have read and understood the [Privacy Policy](#).
- Buttons:** CONTINUE (grey), BACK (orange)
- Footer:** Secure Login with reCAPTCHA subject to Google [Terms & Privacy](#)

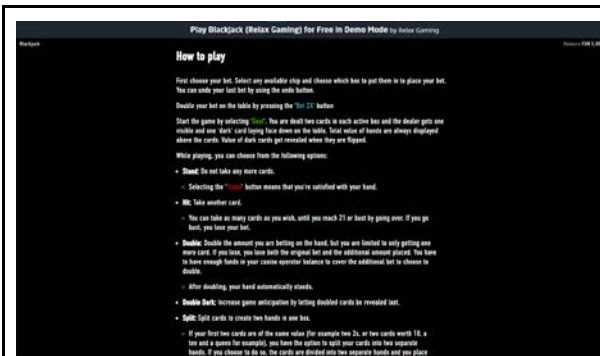
**Right Screenshot: Log into Squarespace**

- Title:** Log into Squarespace
- Fields:**
  - EMAIL ADDRESS: name@example.com
  - PASSWORD: Password (with an eye icon for toggling visibility)
- Buttons:** LOG IN (grey)
- OR** (text separator)
- Social Login Buttons:**
  - Continue with Google (Google logo)
  - Continue with Apple (Apple logo)
  - Continue with Facebook (Facebook logo)
- Text:** CAN'T LOG IN?
- Footer:** Secure Login with reCAPTCHA subject to Google [Terms & Privacy](#)

*Example GUI for account registration and login (Squarespace)*

### 6.2.4.3 Tutorials

Many software applications include tutorials to help users understand how to interact with and use their system. Tutorials can be entirely text-based with imagery, take the form of videos, include interactive steps, or combine these approaches. We needed our software application's tutorial to focus on how to play blackjack, so we analyzed how other designers conveyed blackjack tutorials. The following images show some examples of tutorials we used to design ours:



Example blackjack tutorial GUIs (Relax Gaming)



Example blackjack tutorial GUIs (WP Company)

- 5 When it's your turn, look at your card values and figure out your next move.
- 6 Once you're happy with your hand, stand to end your turn.
- 7 After every player has finished their turn, watch the dealer play out their hand.



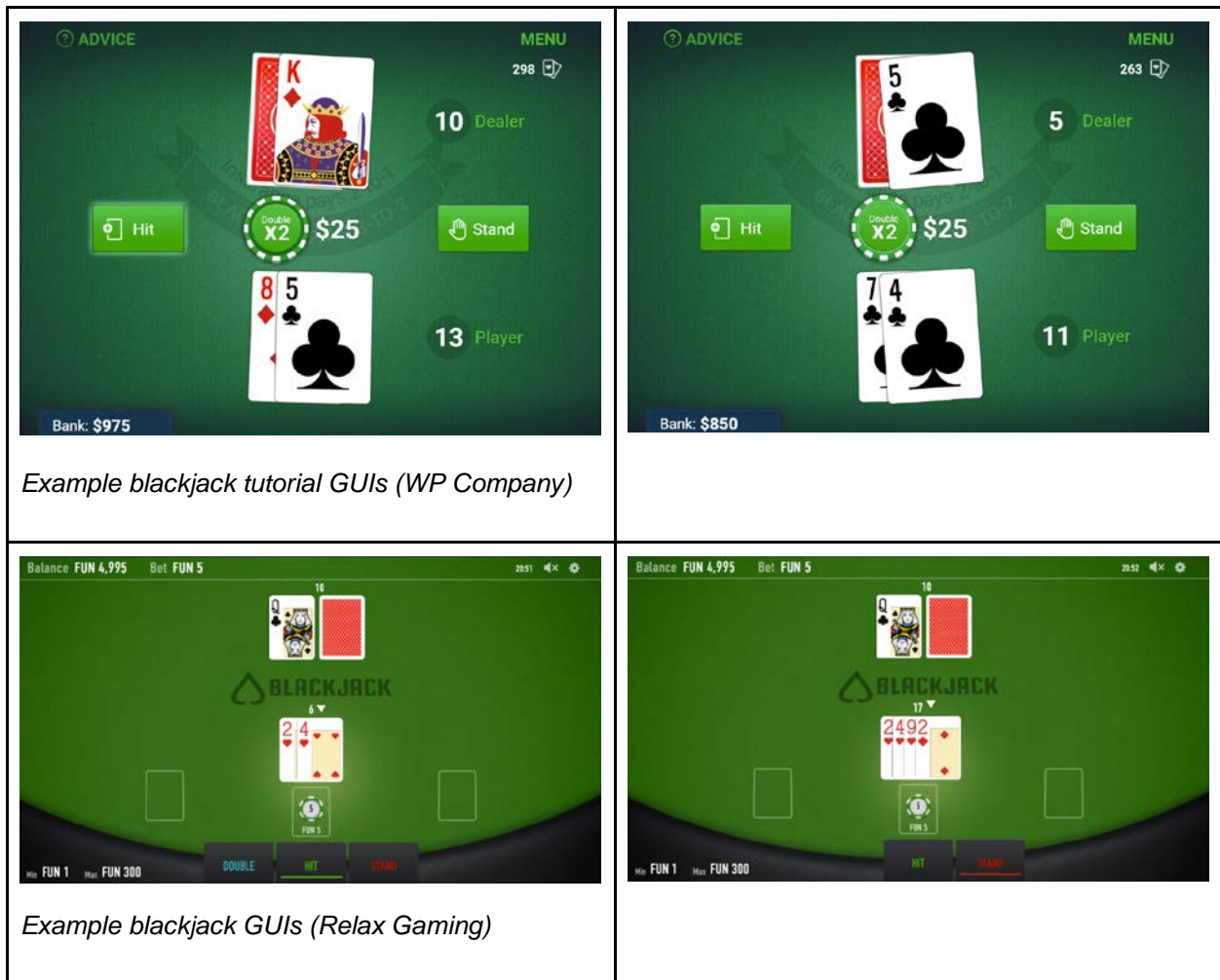
- 8 Take your winnings and repeat the steps above to play another round.

Example blackjack tutorial on website (Blake)

#### 6.2.4.4 Displaying Optimal Blackjack Actions

Providing suggestions to the user about how to proceed through a situation requires calculating or accessing preprocessed calculations of the situation's properties and conveying a choice to the user. As software developers, we had several methods of communicating these choices through the application's GUI, including coloring, animations, and sounds. The following images show how other blackjack games use visual and auditory cues to suggest actions for the user:





#### 6.2.4.5 Databases

A database allows software developers to manage and access a collection of data. The primary functions of databases include creating, reading, updating, and deleting data (Dickson). Databases organize data into tables consisting of rows and columns. Tables organize information about similar objects or entities, for example, a person or product. Rows are commonly referred to as records and contain information about objects, while columns are referred to as fields and represent the pieces of information for a record (Microsoft).



Relational databases contain tables whose information is related to other tables.

According to Microsoft, a good database design has the following traits (Microsoft):

- It divides information into subject-based tables to reduce redundant data.
- It provides the application with the information required to join the information in the tables as needed.
- It helps support and ensure the accuracy and integrity of information.
- It accommodates the data processing and reporting needs.

In designing a database, Microsoft mentions the following steps (Microsoft):

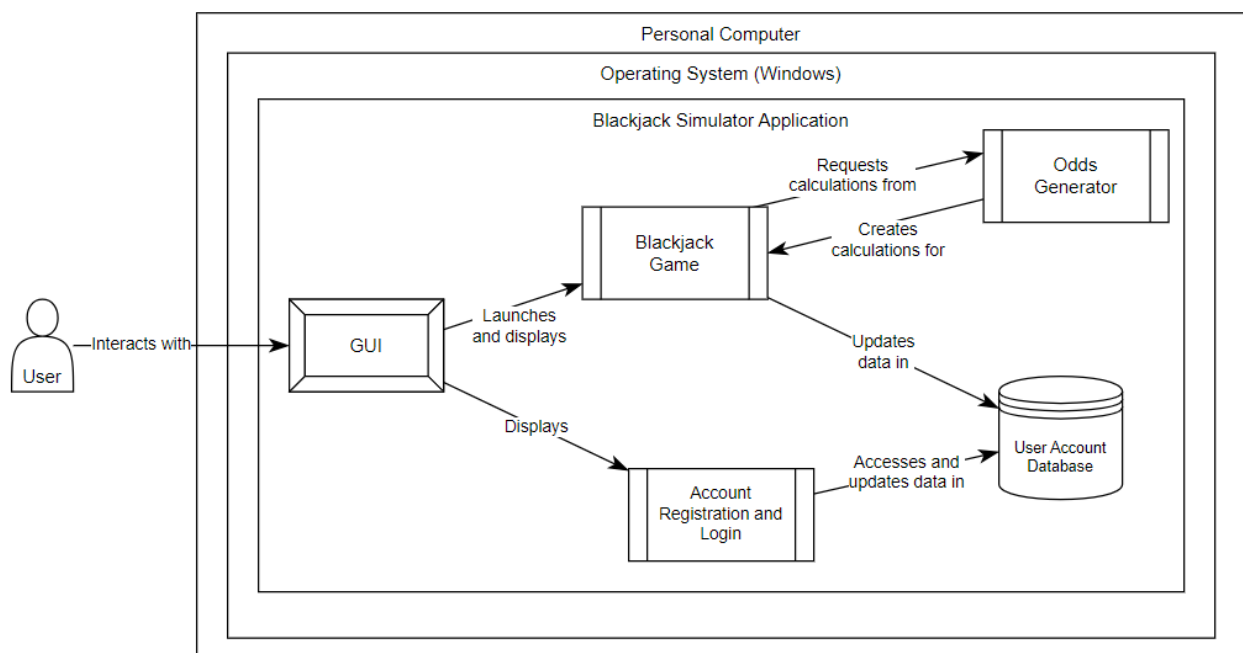
1. **Determine the purpose of your database** - This helps prepare you for the remaining steps.
2. **Find and organize the required information** - Gather all the information you might want to record in the database, such as entity names and IDs.
3. **Divide the information into tables** - Divide the items into major entities or subjects. Each subject then becomes a table.
4. **Turn information items into columns** - Decide which information to store in each table. Each item becomes a field and is displayed as a column in the table. For example, an Employee table might include fields such as Last Name and Hire Date. Ensure the column information does not include calculated data and that the data is in its smallest logical unit (i.e., create columns for first, middle, and last names rather than one column for full names).
5. **Specify primary keys** - Choose each table's primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID.

6. **Set up the table relationships** - Analyze each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. **Refine the design** - Analyze the design for errors. Create the tables and add a few records of sample data. See if the tables return the required results and adjust the design as needed.
8. **Apply the normalization rules** - Apply the data normalization rules to see if the tables are structured correctly. Make adjustments to the tables as needed.

## 6.3 System Design

### 6.3.1 Overview

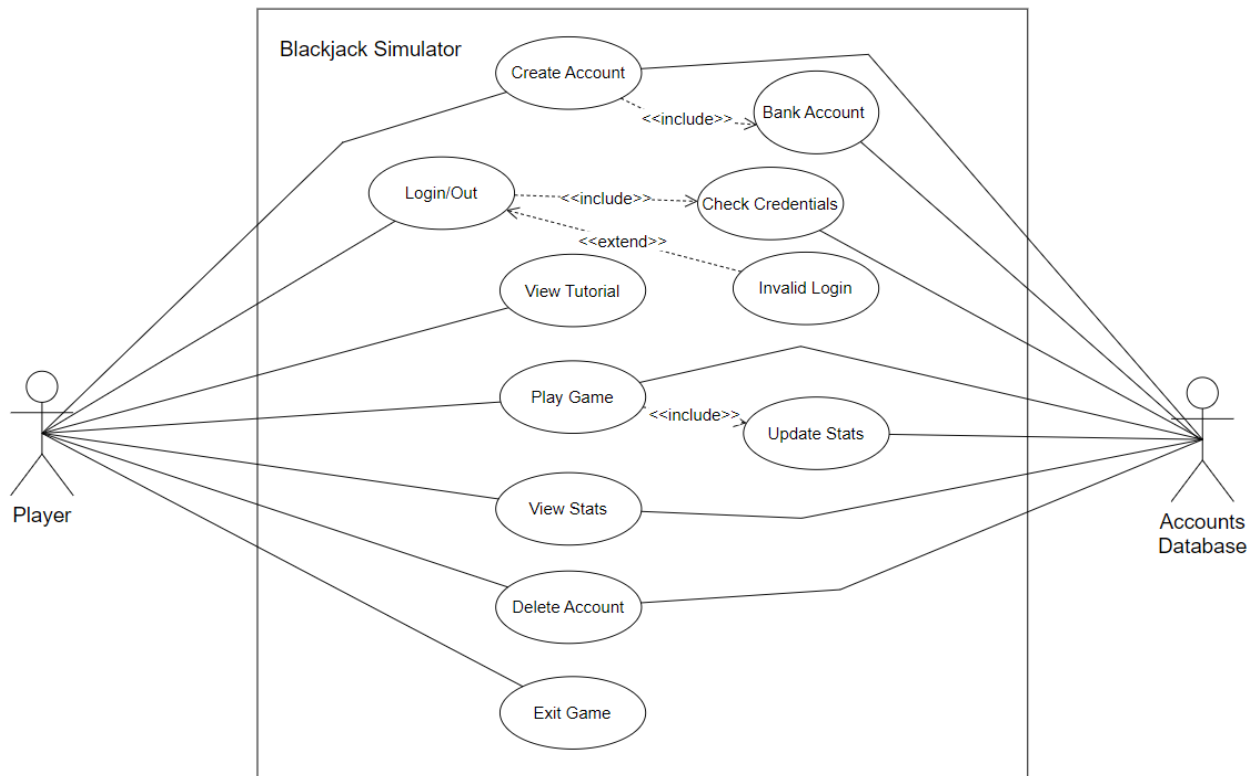
Our software application consists of hardware and software components. The following diagram shows a high-level view of how they interact with each other:



*High-level system components interaction*

### 6.3.2 Use-Cases

The following use-case diagram demonstrates how users interact with the Blackjack Simulator and provides the system's basic functionality.



### 6.3.3 User Interface Design

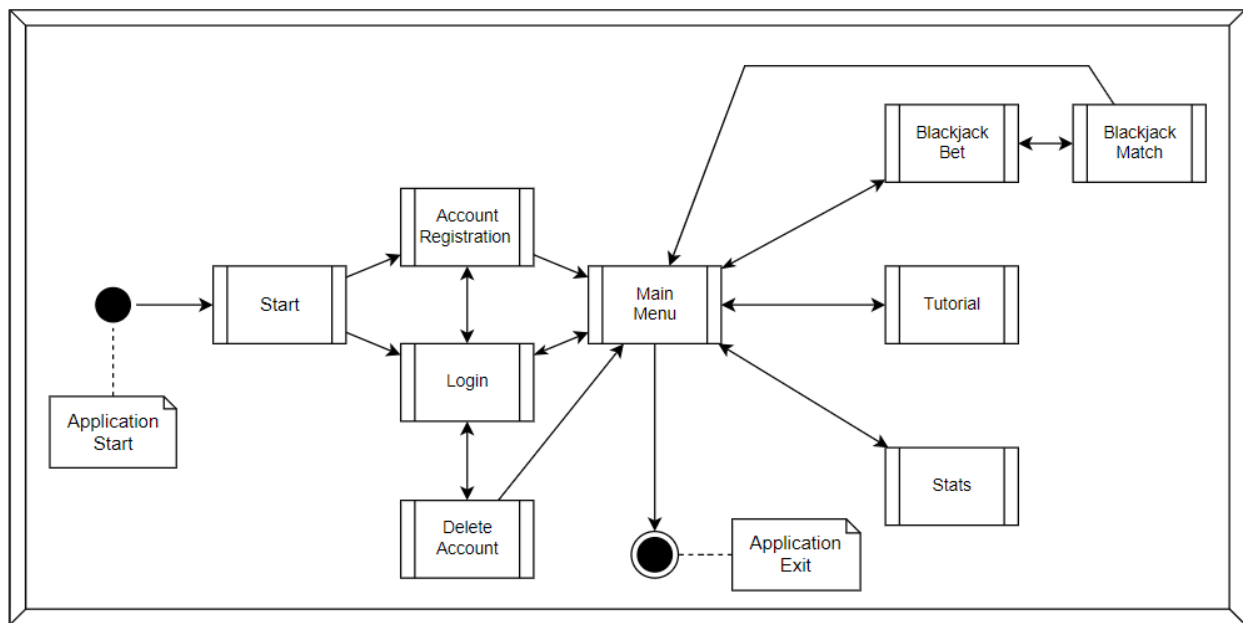
#### 6.3.3.1 UI Overview

The Blackjack Simulator's GUI includes several scenes:

- Start scene
- Registration scene
- Login scene
- Account deletion scene
- Main menu
- Tutorial

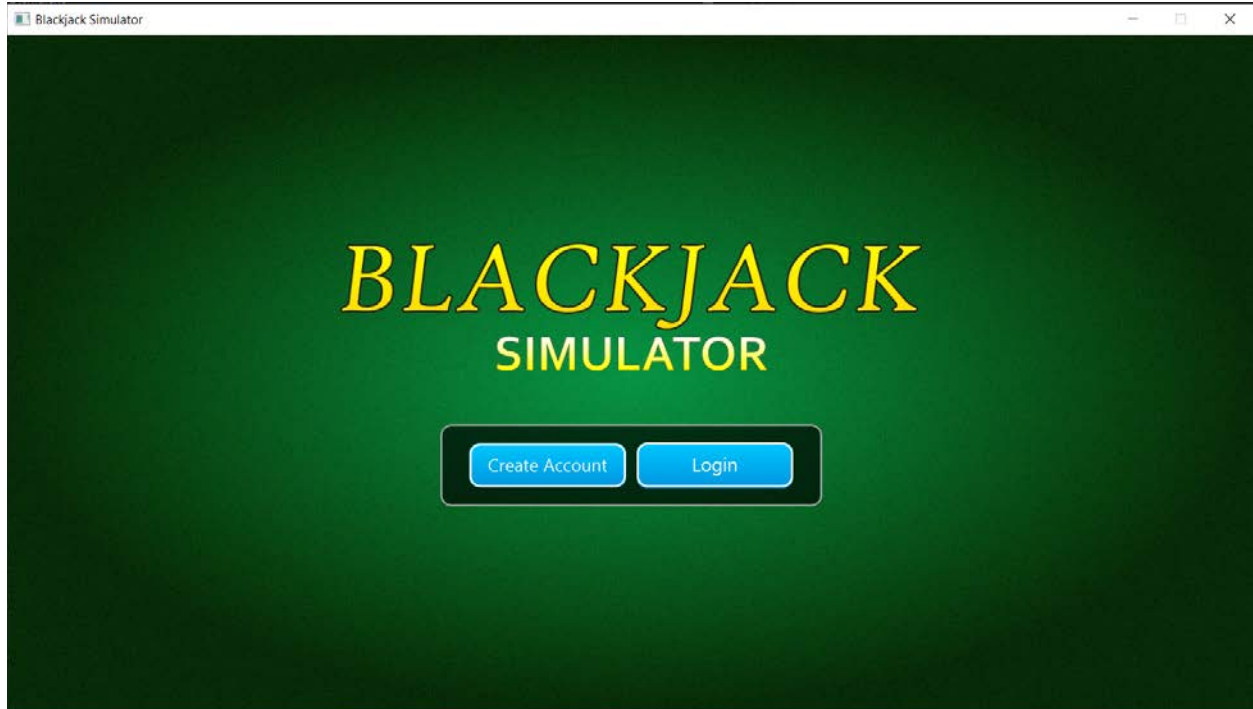
- Stats scene
- The blackjack betting scene
- The blackjack match

The following diagram shows the GUI scenes and how the user progresses through them. The arrows indicate screen state transitions due to button events, which trigger GUI controller class events to switch to new scenes:



*User interface scene transition flow*

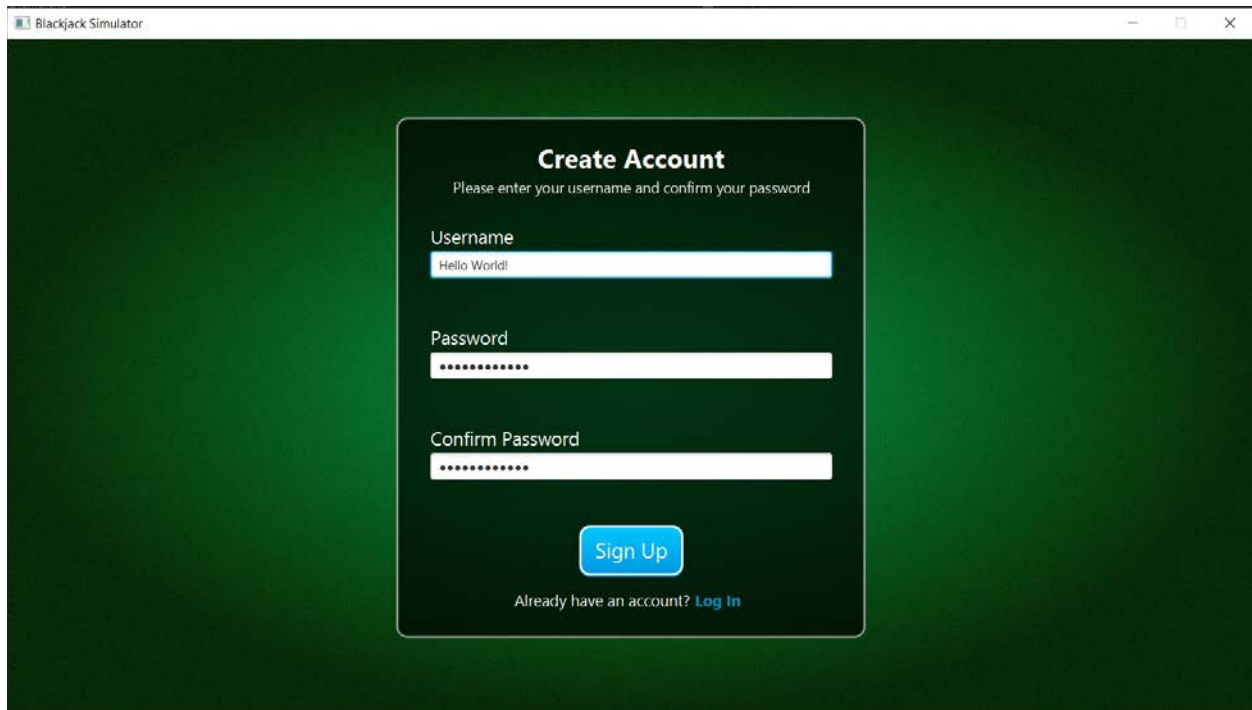
### 6.3.3.2 Start Scene



*Start scene*

The start scene is the first one the user interacts with in the Blackjack Simulator. It displays the application's title and allows the user to register an account or log into one.

### 6.3.3.3 Registration Scene

A screenshot of a web application window titled "Blackjack Simulator". The main content area has a dark green background. In the center, there is a white rectangular form titled "Create Account". Below the title, it says "Please enter your username and confirm your password". There are three input fields: "Username" with the text "Hello World!", "Password" with masked characters "\*\*\*\*\*", and "Confirm Password" also with masked characters "\*\*\*\*\*". Below these fields is a blue "Sign Up" button. At the bottom of the form, it says "Already have an account? [Log in](#)".

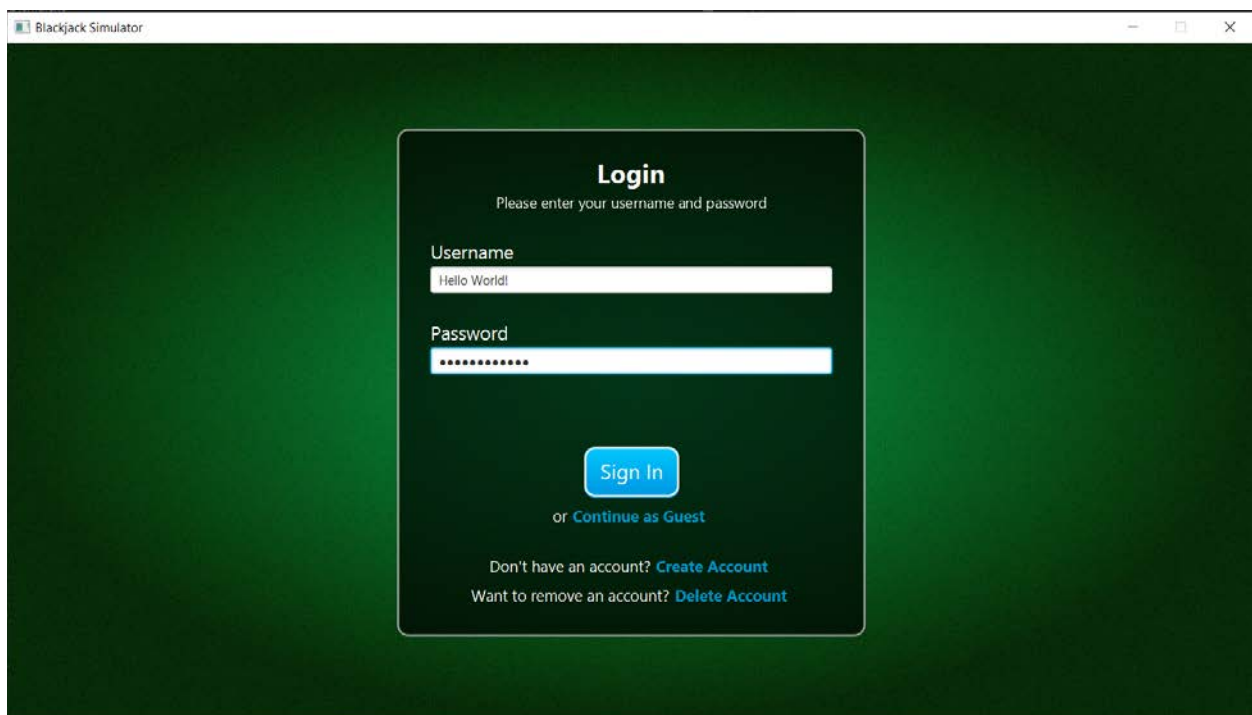
*Registration scene*

The registration scene allows the user to register an account within Blackjack Simulator's database or access the login scene to log into an existing account. The registration scene contains three text fields: one for the user to enter their username, another to enter their password, and a third for reentering their password to confirm it. The user can then select the submit button to register their account. The application's backend then performs the following checks:

1. verify the username field is not empty.
2. verify the username contains at least three characters.
3. verify the password field is not empty.
4. verify the two password fields match.
5. confirm that the user's name is not in the database.

If all checks succeed, the application encrypts the user's password, adds their username and encrypted password to the database, and takes the user to the main menu. The application also gives users a starting bank account to place bets during the game. If any previous checks fail, the application informs the user how to correct the issue. Alternatively, the user can select a link to log into an existing account or select a link to return to the menu if they previously logged into an account or signed in as a guest.

#### 6.3.3.4 Login scene



*Login scene*

The login scene allows a user to log into an existing account within the database or log in as a guest. It has two text fields, one for the username and password and a button for submitting login information. Once the user submits their username and password, the application's backend first verifies that the username and password fields

are not empty, then searches the database for the user's provided username. If the application finds the username, it decrypts the encrypted password it's associated with and compares the decryption with the user's provided password. If they match, the application logs the user in and takes them to the main menu. Otherwise, the application informs the user the username or password needs to be corrected. This scene also provides options if the user has not set up an account, wants to delete an existing one, or return to the main menu if they previously logged in.

### 6.3.3.5 Main menu

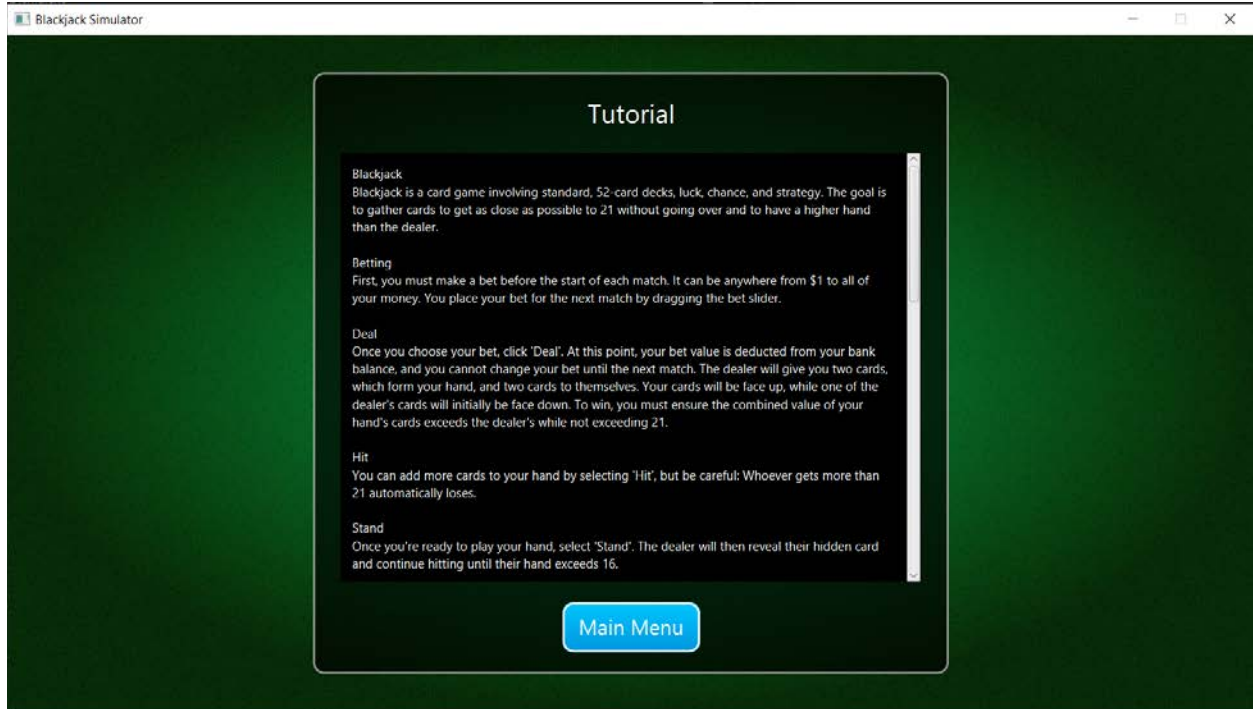


*Main menu scene*

The main menu acts as a GUI hub and allows the user to access the other scenes. It contains GUI buttons that take users to the login, tutorial, stats, and betting scenes and has a button for exiting the application. It also displays the user's username and current balance.



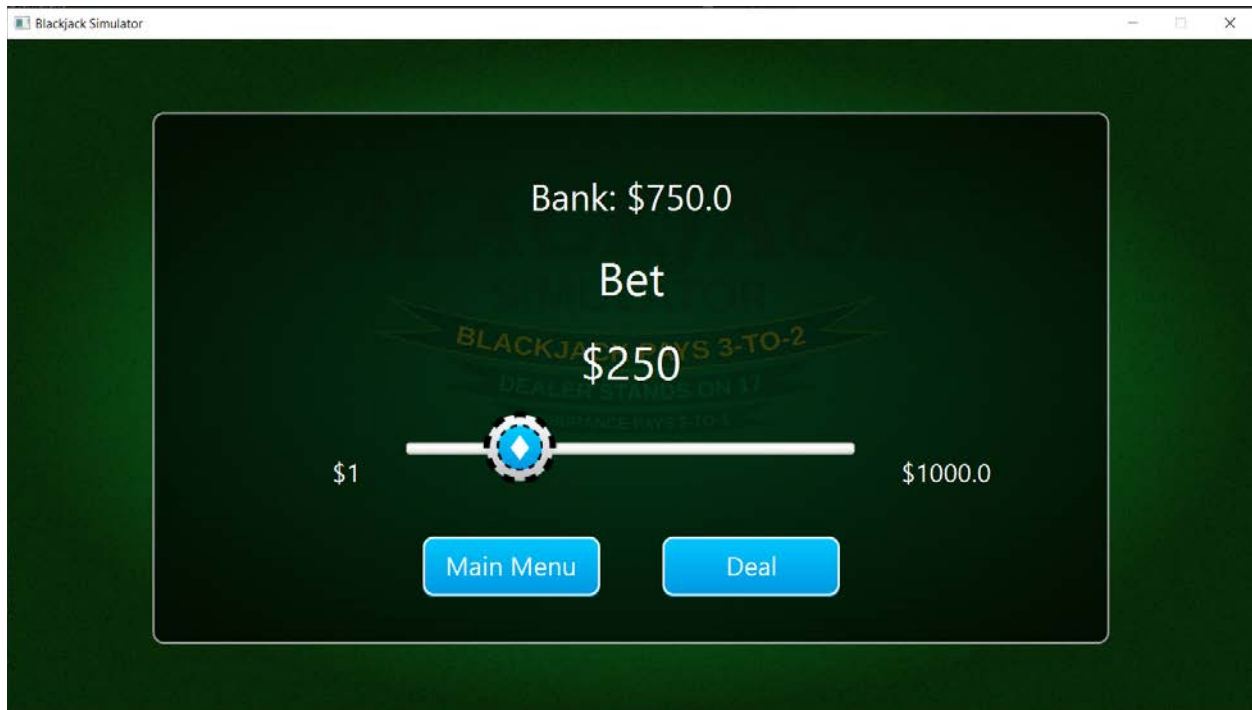
### 6.3.3.6 Tutorial



*Tutorial scene*

The tutorial scene shows the user how to play blackjack. It includes an introduction to explain what blackjack is, how to place a bet, and the various actions available to the user during the blackjack match, including hitting, standing, and doubling. Once the user finishes viewing this scene, they can return to the main menu.

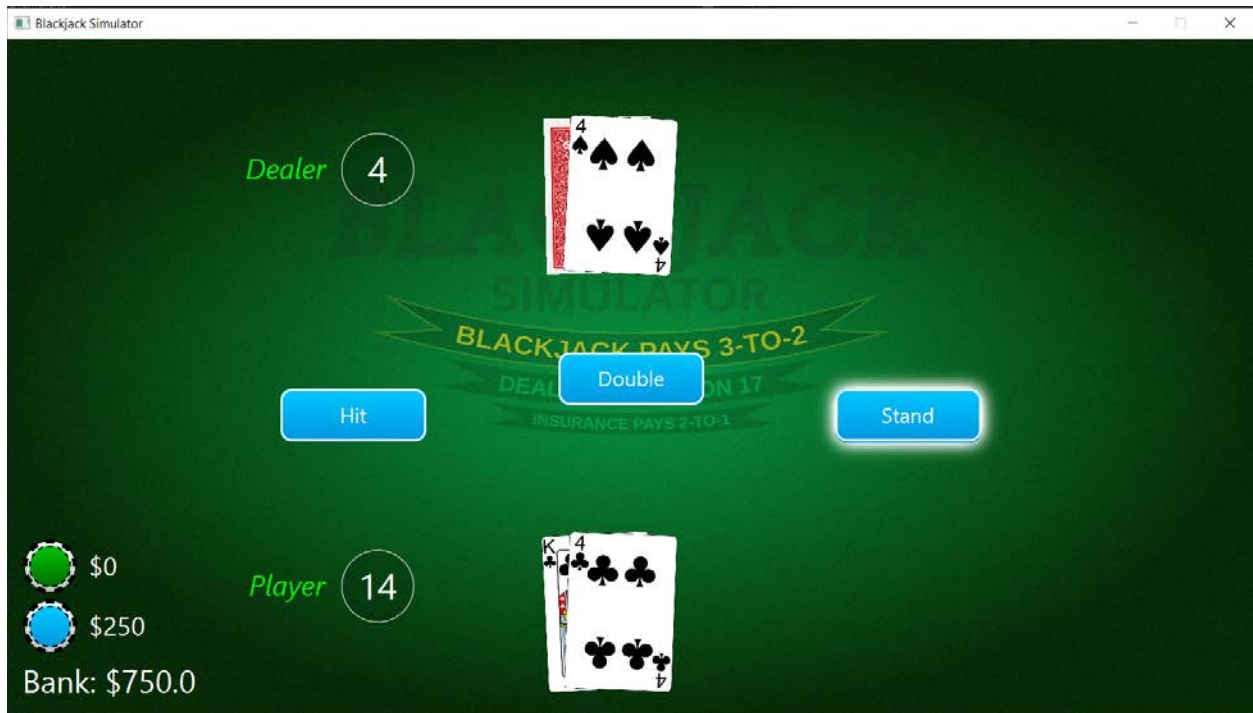
### 6.3.3.7 The Betting Scene



*Betting scene*

The betting scene allows the user to select a bet amount before entering the blackjack match. This scene shows the user how much money is in their account and provides a slider that allows the user to adjust their bet amount. The slider's minimum value is \$1, and the maximum is the user's bank account value, so if the user sets the slider to its maximum position, they will bet all their money on their starting hand in the blackjack match. Once the user is satisfied with their bet, they can choose the option to place their deal and enter the match. If a user enters this scene with an insufficient balance, the application resets their account stats, gives them a starting balance, and notifies them of the process. Alternatively, the user can return to the main menu from this scene.

### 6.3.3.8 Blackjack Match



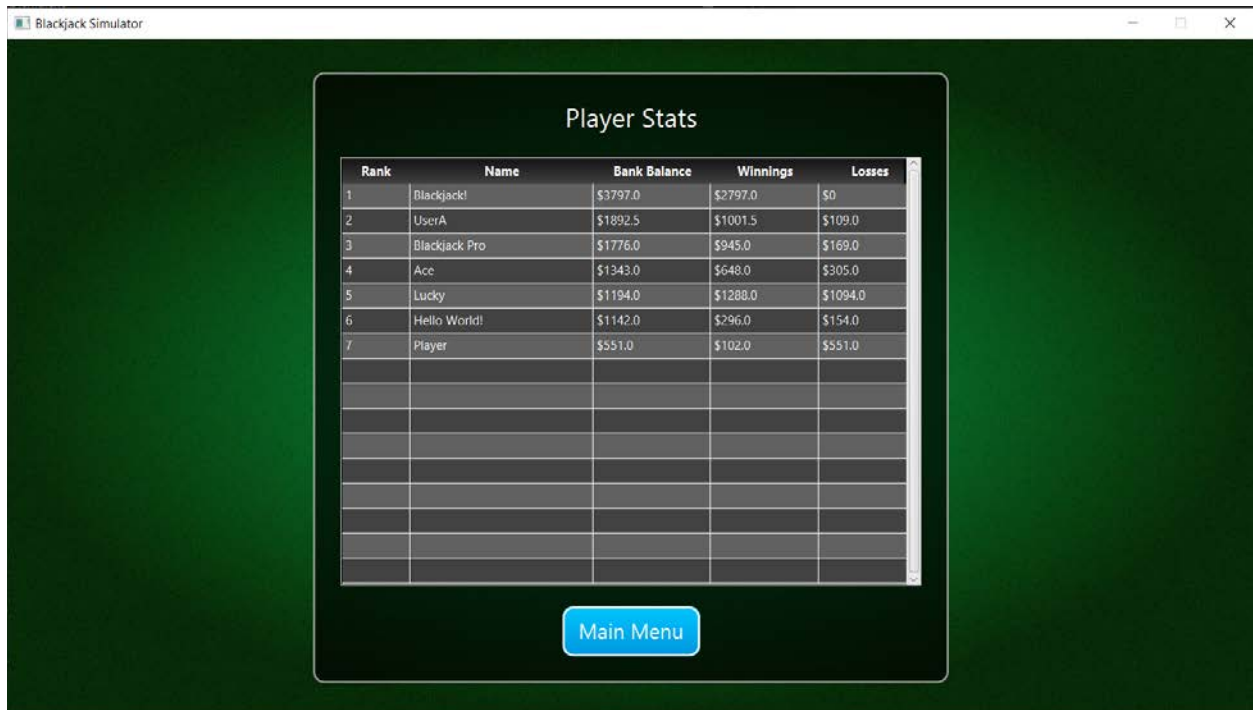
*Blackjack match scene*

This scene is where the blackjack gameplay takes place. During blackjack matches, the user plays against a computer-controlled dealer. The blackjack match's GUI displays various objects:

- the blackjack gameboard, including information about payouts and when the dealer hits,
- the user's and dealer's deck of cards and their total values,
- Buttons representing actions the user can perform,
- and the user's bank balance, bet amount, and insurance bet amount.

Once the match ends, it provides options to return to the betting scene or the main menu.

### 6.3.3.9 Stats Scene

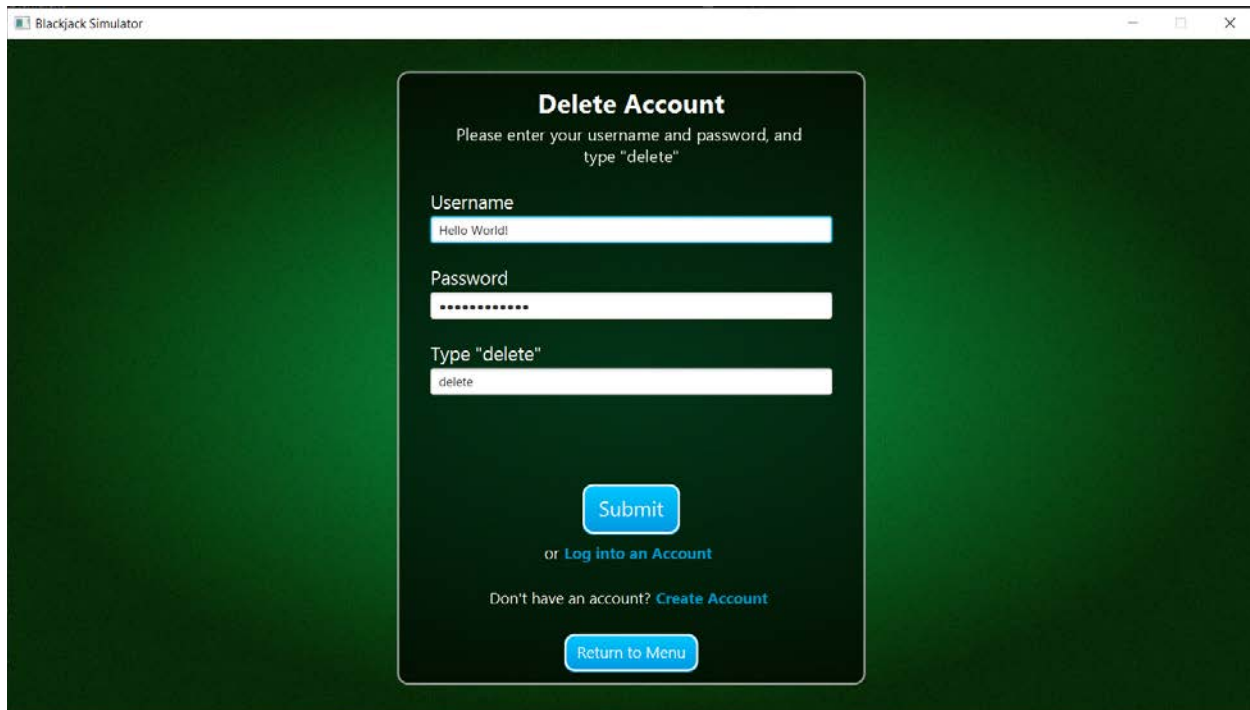
The screenshot shows a window titled "Blackjack Simulator" with a dark green background. In the center is a "Player Stats" panel. It contains a table with 5 columns: Rank, Name, Bank Balance, Winnings, and Losses. The table lists 7 players. Below the table is a blue "Main Menu" button.

Rank	Name	Bank Balance	Winnings	Losses
1	Blackjack!	\$3797.0	\$2797.0	\$0
2	UserA	\$1892.5	\$1001.5	\$109.0
3	Blackjack Pro	\$1776.0	\$945.0	\$169.0
4	Ace	\$1343.0	\$648.0	\$305.0
5	Lucky	\$1194.0	\$1288.0	\$1094.0
6	Hello World!	\$1142.0	\$296.0	\$154.0
7	Player	\$551.0	\$102.0	\$551.0

*Stats scene*

The statistics scene displays information about all the users who registered accounts in the database. It shows each user's rank, username, balance, and wins and losses. It also sorts the users by rank among other users based on their balance. Once the user finishes viewing this scene, they can return to the main menu.

### 6.3.3.10 Account Deletion Scene



**Delete Account**

Please enter your username and password, and type "delete"

Username  
Hello World!

Password  
\*\*\*\*\*

Type "delete"  
delete

Submit

or Log into an Account

Don't have an account? [Create Account](#)

Return to Menu

#### *Deletion scene*

This scene allows the user to delete their account from the database. It includes a username and password text field so the user can provide information about the account they want the system to remove. It also has a text field so the user can confirm they wish to delete the account by typing "delete" into it. Deleting an account removes all of its data from the database. After the user deletes the account, the application will return them to the account registration scene. The user can also log into an account or return to the main menu if they previously logged in.

## 6.4 Architecture and Design

The Blackjack Simulator's architecture consists of the user's operating system (Windows 10) and the application. The application's front end relies on 12 FXML files, 60+ PNG files, and three Cascading Style Sheet (CSS) files. The backend relies on 32

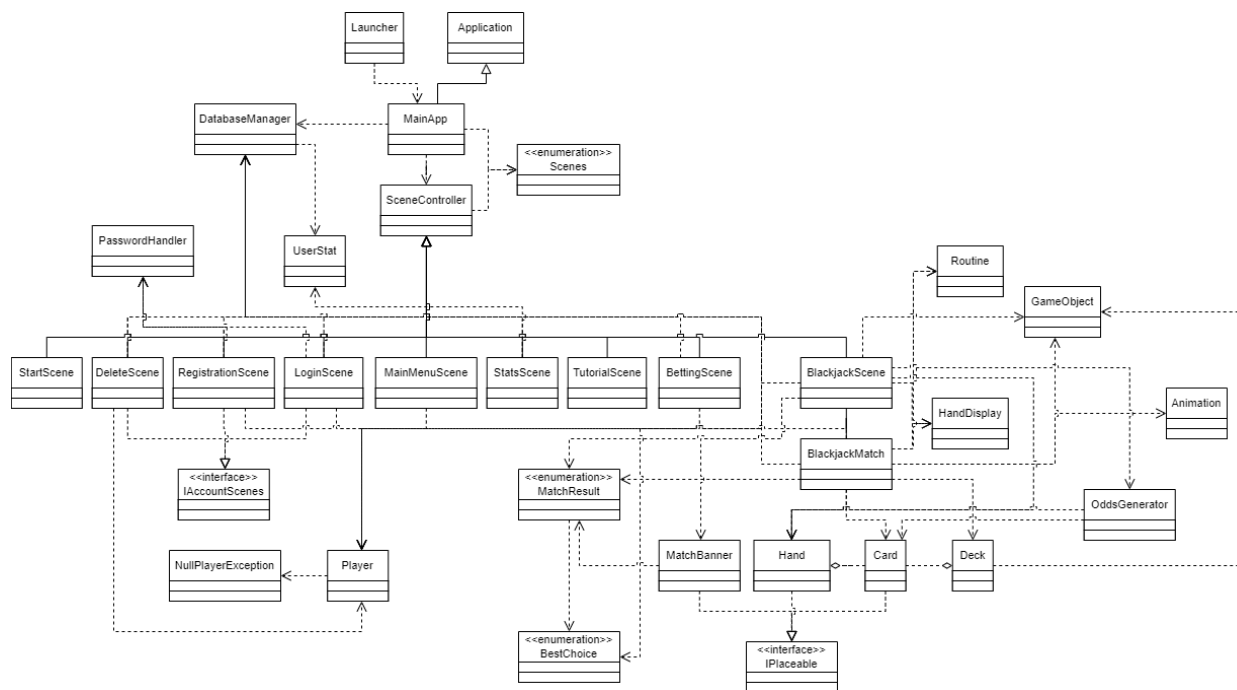
class files, including enums and interfaces, three Comma Separated Values (CSV) files, and the database file the application creates. The application consists of the JAR runnable file and the database file it creates and manages. The JAR file is approximately 42 MB, and the database file is at least approximately 24 KB. Including the frontend, backend, internal program dependencies and files, and JAR and database, the project is approximately 230 MB in size.

Our application's front end uses the JavaFX Software Development Kit (SDK). The JavaFX SDK is an open-source platform that provides libraries and software for developing Java applications that run on desktop, mobile, and embedded systems (Gluon). Our application uses the JavaFX SDK to generate GUI scenes and handle GUI interactions from the user. We also used JavaFX's SceneBuilder tool to facilitate GUI creation, which is an application that allowed us to build application GUIs rapidly through drag-and-drop interface functionality.

Our application's backend runs on the Java programming language and uses Java to handle the declaration and implementation of classes, data types, and method processes. The backend handles and influences JavaFX GUI events, utilizes password encryption/decryption, accesses CSV files, and uses the Java Database Connectivity (JDBC) API. The JDBC API allows Java-based applications to access various data sources, including flat files, spreadsheets, and relational databases (Java JDBC API). The JDBC API requires a driver to mediate connections between the JDBC API and a database, so we used the SQLite JDBC driver, which provides a library for creating and accessing SQLite database files. Our application uses the JDBC API and SQLite JDBC

driver to create and manage the database containing user information, including login details and play history.

The following diagram shows a high-level view of the main Java classes our application's backend uses:

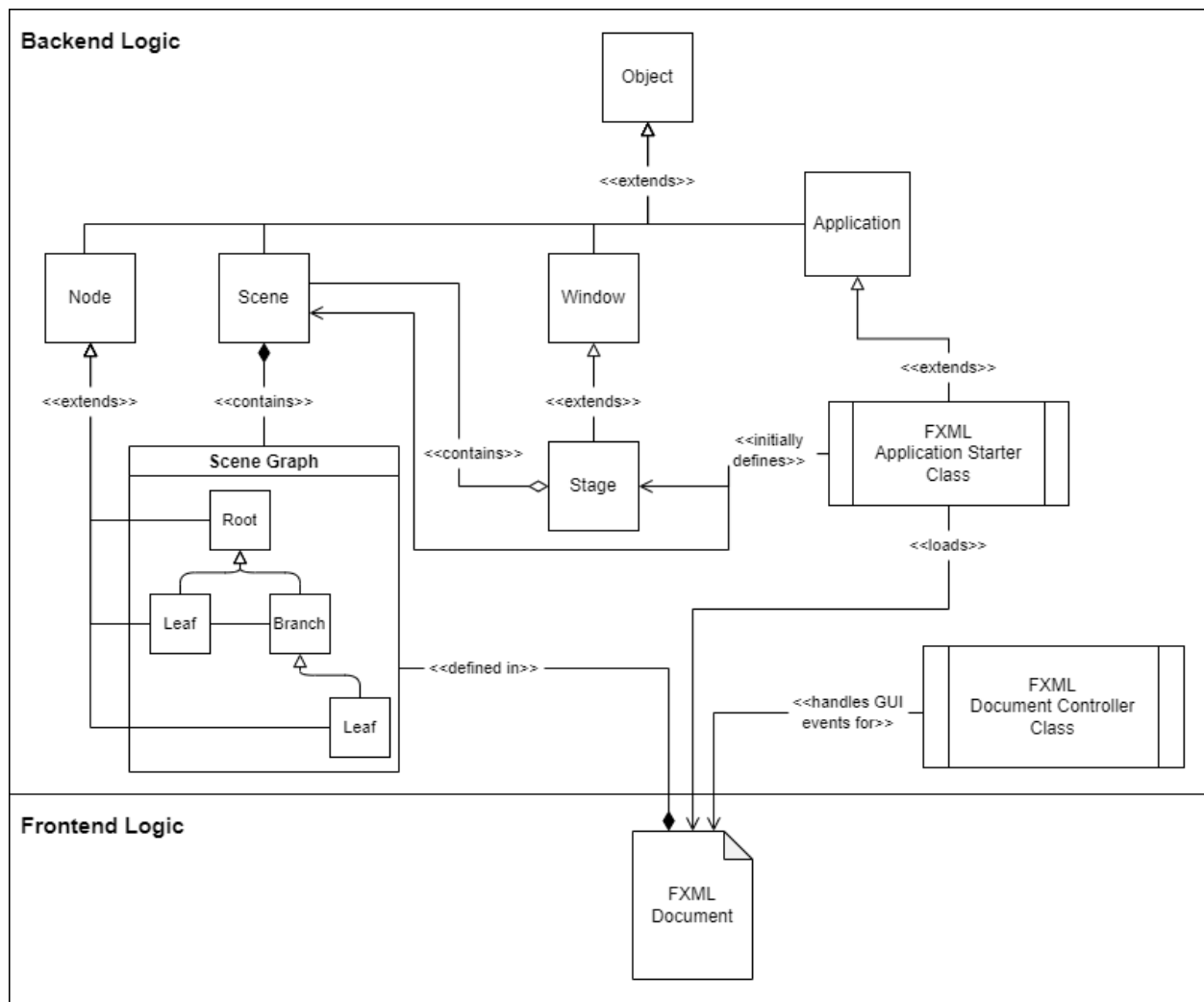


*High-level class diagram*

Among these classes, the `MainApp` class extends JavaFX's `Application` class and acts as a starting point for our application. The `StartScene`, `RegistrationScene`, `LoginScene`, `DeleteScene`, `MainMenuScene`, `TutorialScene`, `StatsScene`, `BettingScene`, and `BlackjackScene` classes contain logic for handling the application's scenes. The `PasswordHandler` class provides methods for encrypting and decrypting Strings of passwords. The `DatabaseManager` class is responsible for creating and managing the application's database. The blackjack gameplay relies on several classes, including the `BlackjackScene`, `BlackjackMatch`, `Deck`, `Hand`, `Card`, and `OddsGenerator` classes.

### 6.4.1 JavaFX GUI Architecture

The diagram below illustrates the basic structure of JavaFX applications. Oracle also discusses this structure and provides JavaFX tutorials on its website (3 Hello World, javafx style).



*JavaFX backend and frontend components*

A JavaFX application defines the GUI container with a stage and a scene. JavaFX's Stage class extends JavaFX's Window class, which extends Java's Object class, and acts as the top-level JavaFX container. It works similarly to a JFrame in



Java's Swing framework. The Scene class also extends Java's Object class and serves as the container for all content.

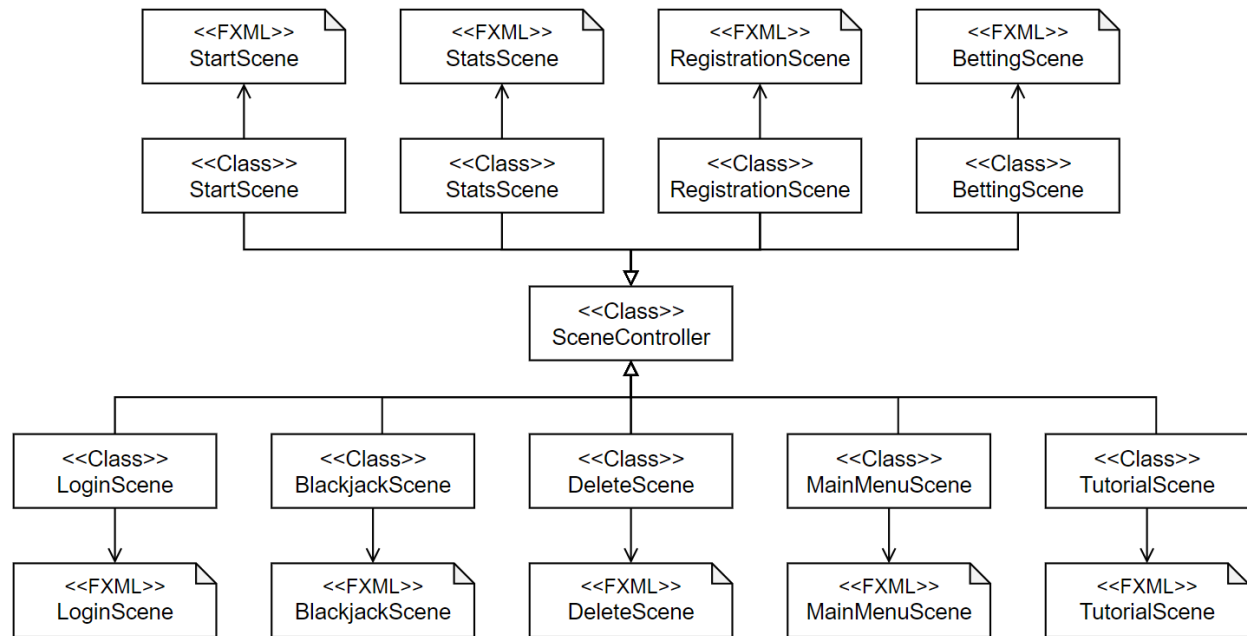
In JavaFX, a scene graph contains the contents of the scene. It's a hierarchical graph structure of the scene's nodes. Scene graphs include a root node, zero or more leaf nodes, and branch nodes. A node can be a variety of entities, including GUI elements that control the layout of others or controls that provide user interactivity.

The main class for JavaFX applications extends JavaFX's Application class and includes Java's main() method and the Application class's start() method. The main class serves as the entry point for the JavaFX application.

An FXML file uses an XML-based language to define the structure of a GUI scene (3 Hello World). The main class and other classes can use JavaFX's FXMLLoader class to load FXML source files and return their resulting object graph.

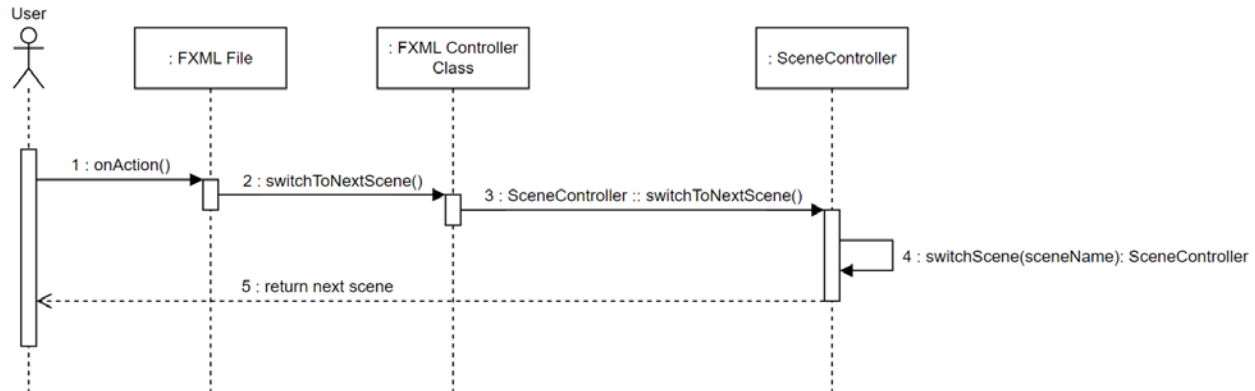
An FXML controller class is any class that handles the user's interactions with FXML GUI scene elements. It, for example, holds the logic for handling events that occur when a user selects a button.

Among these classes, nine are controller classes for managing FXML files: StartScene, RegistrationScene, LoginScene, DeleteScene, MainMenuScene, TutorialScene, StatsScene, BettingScene, and BlackjackScene. The controller classes extend the SceneController class and override its methods. Each controller class contains logic for an FXML file, and its FXML file has an attribute that defines its controller class.



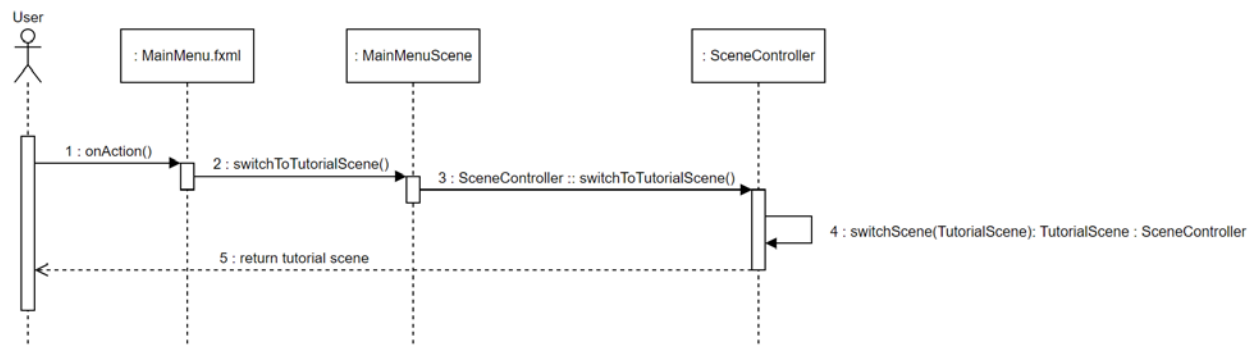
*Relationships of controller classes and FXML files*

Each scene contains buttons, and each button performs an event when the user presses it. The FXML file of a button defines its `onAction` attribute, which defines the event the button invokes when a user presses it. Since an FXML file has an attribute that defines its controller class, the file can access its controller's methods, so a button's `onAction` attribute can determine which method of the file's controller to invoke. All scenes have buttons that load other scenes when pressed by the user. The following diagram shows the backend events that take place after a user presses a button to go to the next scene:



*General scene transition sequence diagram*

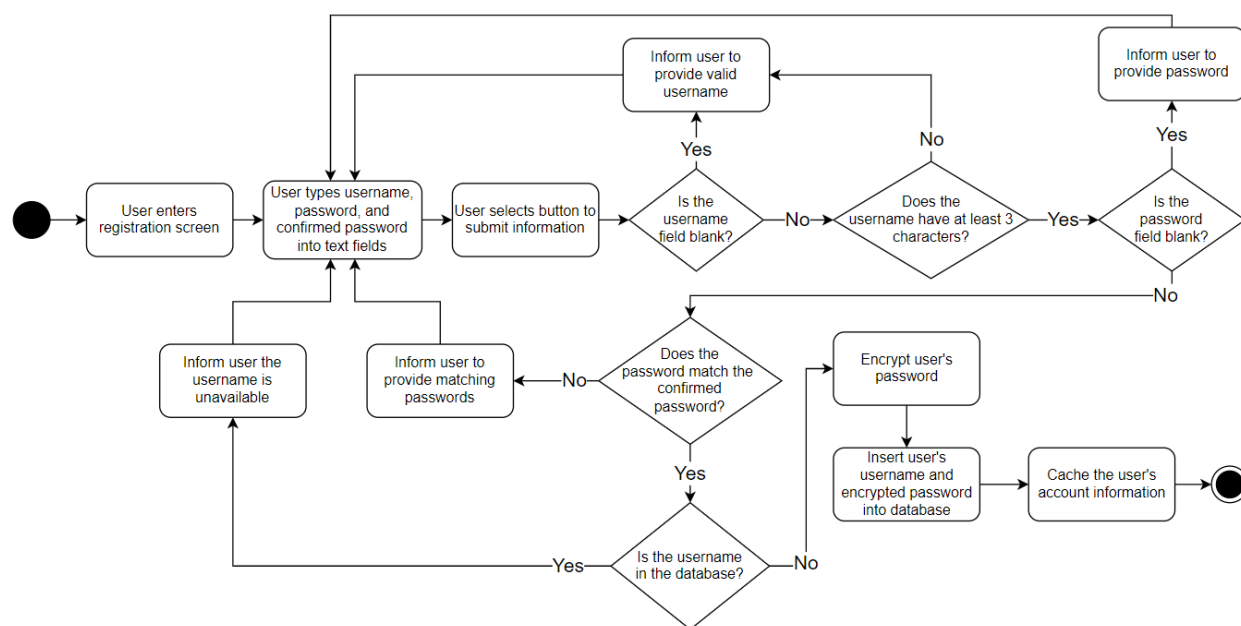
When the user presses one of these buttons, it invokes the method defined in its onAction attribute. This is a method in the button's FXML file's controller class that switches to the next scene. The controller (which extends the SceneController class) calls the base implementation of the corresponding method to switch to the next scene. Internally, SceneController then calls the method to switch scenes, which loads the next scene, where the user can select another button to repeat the process. The following diagram shows an example of a user on the main menu scene selecting the button to go to the tutorial scene:



*Main menu to tutorial scene sequence example*

### 6.4.2 Security Architecture

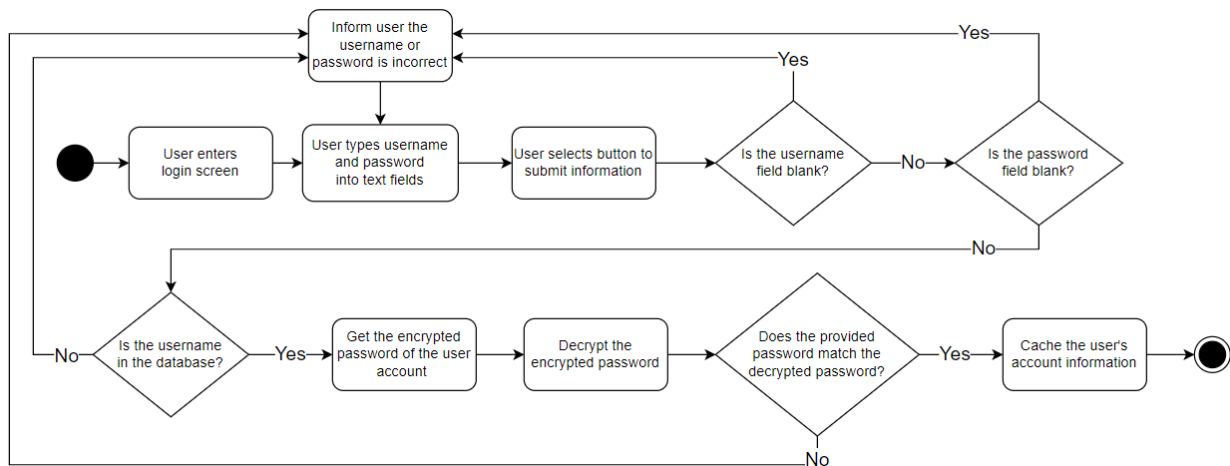
The Blackjack Simulator's database does not store passwords as plain text and relies on encryption and decryption processes. When users register an account, the application encrypts their password and stores the encrypted text in the database. During the encryption process, the application takes the plain text of the password, a secret key, and a salt value. Then, it uses the PBKDF2 key derivation function with an SHA-256 hash function and AES encryption to encrypt the given text.



#### *User Account Registration*

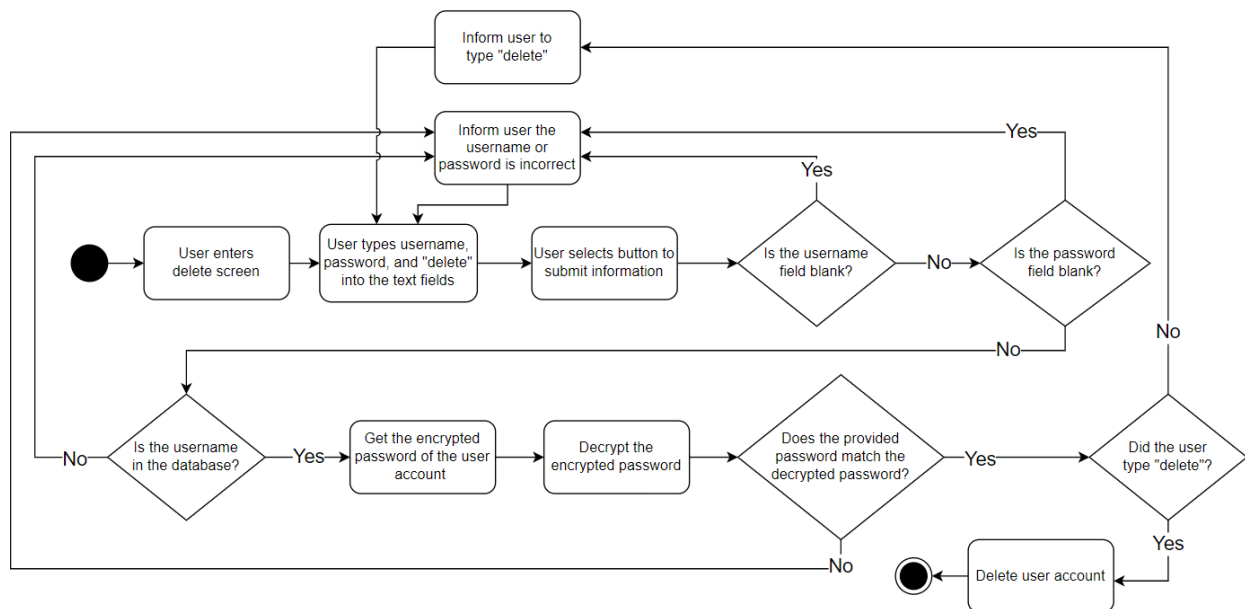
When users log in to their accounts, the application uses their input to retrieve and decrypt a password and then compares it with the submitted password. During decryption, the application takes the encrypted text of the password, the secret key, and the salt value. Then, it uses PBKDF2 with SHA-256 to derive the secret key, which it

then uses for AES decryption to return the plain text of the encrypted password.



### *Current User Account Login*

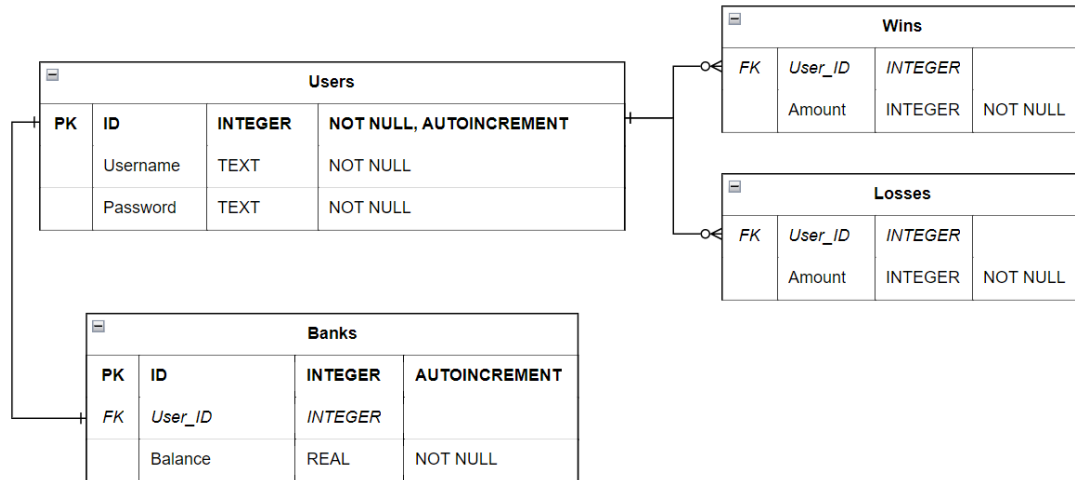
Similarly to logging into an account, when users want to delete their account, the application uses their input to retrieve and decrypt their password and compares it with the submitted password.



### *User Account Deletion*

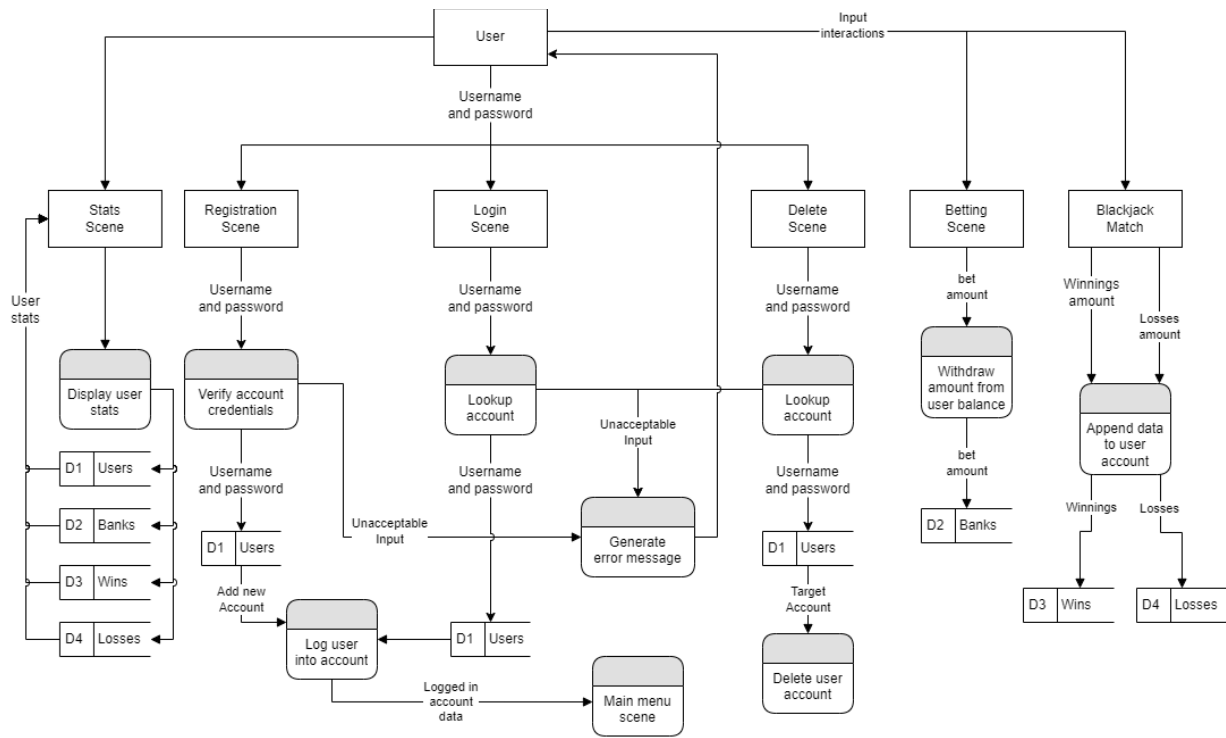
### 6.4.3 SQLite JDBC Database Design

The primary function of the Blackjack Simulator database is to provide functionality for storing and updating statistical information about users' blackjack sessions. The schema below shows the entities, attributes, and relationships used for managing and updating data:



*User accounts database schema*

The application's GUI provides limited user interactions with the database; the user only indirectly interacts with the database when registering, logging into an account, deleting an account, or viewing player stats. When a user logs into an account and plays blackjack matches, the application updates the user's account by appending data for wins and losses. The below diagram illustrates how data flows into/out of the user accounts database.



### Application data flow

The application's backend deploys and manages the database on the user's device. The SQLite JDBC functionality allows the backend to translate String data into various executable SQL queries. However, the backend logic will only contain the functions necessary to handle the required database features.

### 6.4.3.1 Required Database Features

The table below lists the database features our database will need to satisfy our requirements. These features expand upon our requirements and perform processes related to the above database schema. Each feature has an associated ID we will use later to show which SQL process performs the required feature.

Required Feature	FID
Creating the database tables.	F001
Checking if a database table exists.	F002
Inserting a new user account into the database with a username and password.	F003
Checking if a user account exists using a username.	F004
Retrieving a user account's ID using a username.	F005
Retrieving a user account's password using a username.	F006
Inserting a new bank relation for a user account using a user account ID and set bank balance.	F007
Retrieving a user account's related bank balance using a user account ID.	F008
Updating a user account's related bank balance using a user account ID and a balance amount.	F009
Inserting a new wins relation for a user account with a set win amount.	F010
Inserting a new loss relation for a user account with a set loss amount.	F011
Displaying data of all user accounts using their usernames, balance, and summed wins and losses, sorted in descending order by their balance.	F012
Deleting a user account using a username.	F013
Deleting all win relations for a user account using a user account ID.	F014
Deleting all loss relations for a user account using a user account ID.	F015



### 6.4.3.2 Message Formation Process

We needed to use various data types to access the database and perform SQL operations. The main datatypes we used that the SQLite JDBC API provides are DriverManager, Connection, Statement, PreparedStatement, and ResultSet. We also needed Java's String datatype to send SQL statements to the database.

Datatype	Purpose
DriverManager	Provides the services required for managing a JDBC driver.
Connection	Provides a connection to a specific database and allows us to send and execute SQL queries and retrieve query results.
String	Forms the literal SQL statement text that a Statement or PreparedStatement uses to execute queries.
Statement	Used for executing a static SQL statement and returning its results.
PreparedStatement	Used for executing a dynamic, precompiled SQL statement and returning its results. Prepared statements can pass parameterized data into SQL String queries.
ResultSet	Provides the resulting data table generated by executing a statement that queries the database.

Before using the database, we must set up a connection with it by passing its URL as a string to the DriverManager's method for retrieving a database. The following pseudocode demonstrates this process:

```

Connection connection;

func establishDBConnection returns void() {
    String databaseURL = "Add path to database here";
    connection = DriverManager.getConnection(databaseURL);
}

```

Once the connection is established, we can send SQL queries to the database. SQL queries require a String that stores the literal text of the query, a Statement/PreparedStatement object, and the connection to the database. The following pseudocode shows how to execute a simpler SQL query on the database. This query could, for example, create a table in the database:

```
func doSQLProcess returns void() {  
    String sqlQuery = "Add SQL here";  
    Statement stmt = connection.createStatement();  
    stmt.executeSQL(sqlQuery);  
}
```

Some processes require us to send SQL queries to the database to retrieve a data set. We can achieve this by passing a String of the SQL query into a Statement as before, then using a ResultSet to store the data returned from executing the query:

```
func doSQLProcess returns ResultSet() {  
    String sqlQuery = "Add SQL here";  
    Statement stmt = connection.createStatement();  
    ResultSet rs = stmt.executeQuery(sql);  
    Return rs;  
}
```

For less defined processes requiring input from the user or other processes, we can use PreparedStatement instead of a Statement. A PreparedStatement allows us to use various setter methods to pass data into the SQL query once we place question marks within the SQL string. Each question mark refers to a location in the String where a setter method can provide a value. Question marks use a 1s-based indexing (i.e., the first question mark has an index of 1, the second an index of 2, and so on). The following pseudocode demonstrates how to pass data into a PreparedStatement:

```
func doSQLProcess returns void(String value1, Int value2, Double value3) {
```

```

String sqlQuery = "Add SQL (?, ?, ?)";
PreparedStatement pstmt = connection.prepareStatement(sqlQuery);
pstmt.setString(1, value1);    // Set value of first question mark
pstmt.setInt(2, value2);      // Set value of second question mark
pstmt.setDouble(3, value3);   // Set value of third question mark
pstmt.executeUpdate();
}

```

Building on the previous examples, a PreparedStatement also allows us to retrieve a ResultSet from an SQL query. The following pseudocode demonstrates how to pass data into a PreparedStatement, then retrieve its ResultSet:

```

func doSQLProcess returns ResultSet(String value1, Int value2, Double value3) {
    String sqlQuery = "Add SQL (?, ?, ?)";
    PreparedStatement pstmt = connection.prepareStatement(sqlQuery);
    pstmt.setString(1, value1);
    pstmt.setInt(2, value2);
    pstmt.setDouble(3, value3);
    ResultSet rs = pstmt.executeQuery();
    Return rs;
}

```

#### 6.4.3.3 SQLite Query Strings

The following sections contain the text/values we needed to store in Strings to fulfill the required database features. In Java, the following Strings require backslashes to escape inner quotation characters, but we have omitted them to increase the clarity of the Strings we used.

## Creating the database tables

Creating the Users table (F001):

```
"CREATE TABLE "Users" (  
    "ID"          INTEGER NOT NULL,  
    "Username"    TEXT NOT NULL,  
    "Password"    TEXT NOT NULL,  
    PRIMARY KEY("ID" AUTOINCREMENT)  
);"
```

Creating the Banks table (F001):

```
"CREATE TABLE "Banks" (  
    "ID"          INTEGER,  
    "User_ID"     INTEGER,  
    "Balance"     INTEGER,  
    FOREIGN KEY("User_ID") REFERENCES "Users"("ID") ON DELETE  
    CASCADE,  
    PRIMARY KEY("ID" AUTOINCREMENT)  
);"
```

Creating the Wins table (F001):

```
"CREATE TABLE "Wins" (  
    "User_ID"     INTEGER,  
    "Amount"      INTEGER NOT NULL,  
    FOREIGN KEY("User_ID") REFERENCES "Users"("ID") ON DELETE  
    CASCADE  
);"
```

Creating the Losses table (F001):

```
"CREATE TABLE "Losses" (  
    "User_ID"     INTEGER,  
    "Amount"      INTEGER NOT NULL,  
    FOREIGN KEY("User_ID") REFERENCES "Users"("ID") ON DELETE  
    CASCADE  
);"
```

Checking if a given table exists in the database using its table name (F002):

```
"SELECT name FROM sqlite_master WHERE type="table" AND name=" +  
tableName"
```

## **Sending SQL queries to the database**

Inserting a new user account into the database with a username and password (F003):

```
"INSERT INTO Users (Username, Password) VALUES (?, ?);"
```

Checking if a user account exists using a username (F004):

```
"SELECT EXISTS (SELECT 1 FROM Users WHERE Username = ?);"
```

Retrieving a user account's ID using a username (F005):

```
"SELECT ID FROM Users WHERE Username = ?"
```

Retrieving a user account's password using a username (F006):

```
"SELECT Password FROM Users WHERE Username = ?"
```

Inserting a new bank relation for a user account using a user account ID and set bank balance (F007):

```
"INSERT INTO Banks (User_ID, Balance) VALUES (" + userID + ", " + amount + ")"
```

Retrieving a user account's related bank balance using a user account ID (F008):

```
"SELECT Balance FROM BANKS WHERE User_ID = " + userID
```

Updating a user account's related bank balance using a user account ID and a balance amount (F009):

```
"UPDATE Banks SET Balance = " + amount + " WHERE User_ID = " + userID
```

Inserting a new wins relation for a user account with a set win amount (F0010):

```
"INSERT INTO Wins (User_ID, Amount) VALUES (" + userID + ", ?)"
```

Inserting a new losses relation for a user account with a set loss amount (F011):

```
"INSERT INTO Losses (User_ID, Amount) VALUES (" + userID + ", ?)"
```

Displaying data of all user accounts using their usernames, balance, and summed wins and losses, sorted in descending order by their balance (F012):

```
"SELECT
    ROW_NUMBER() OVER(ORDER BY Balance DESC) AS Rank,
    Username,
    Banks.Balance,
    (SELECT ifnull(sum(Amount), 0) FROM Wins WHERE Wins.User_ID =
Users.ID) AS winnings,
    (SELECT ifnull(sum(Amount), 0) FROM Losses WHERE Losses.User_ID =
Users.ID) AS loss
FROM
    Users
    LEFT JOIN Banks ON Banks.User_ID = Users.ID"
```

Deleting a user account using a username (F013):

```
"DELETE FROM Users WHERE Username = ?"
```

Deleting all wins relation for a user account using a user account ID (F014):

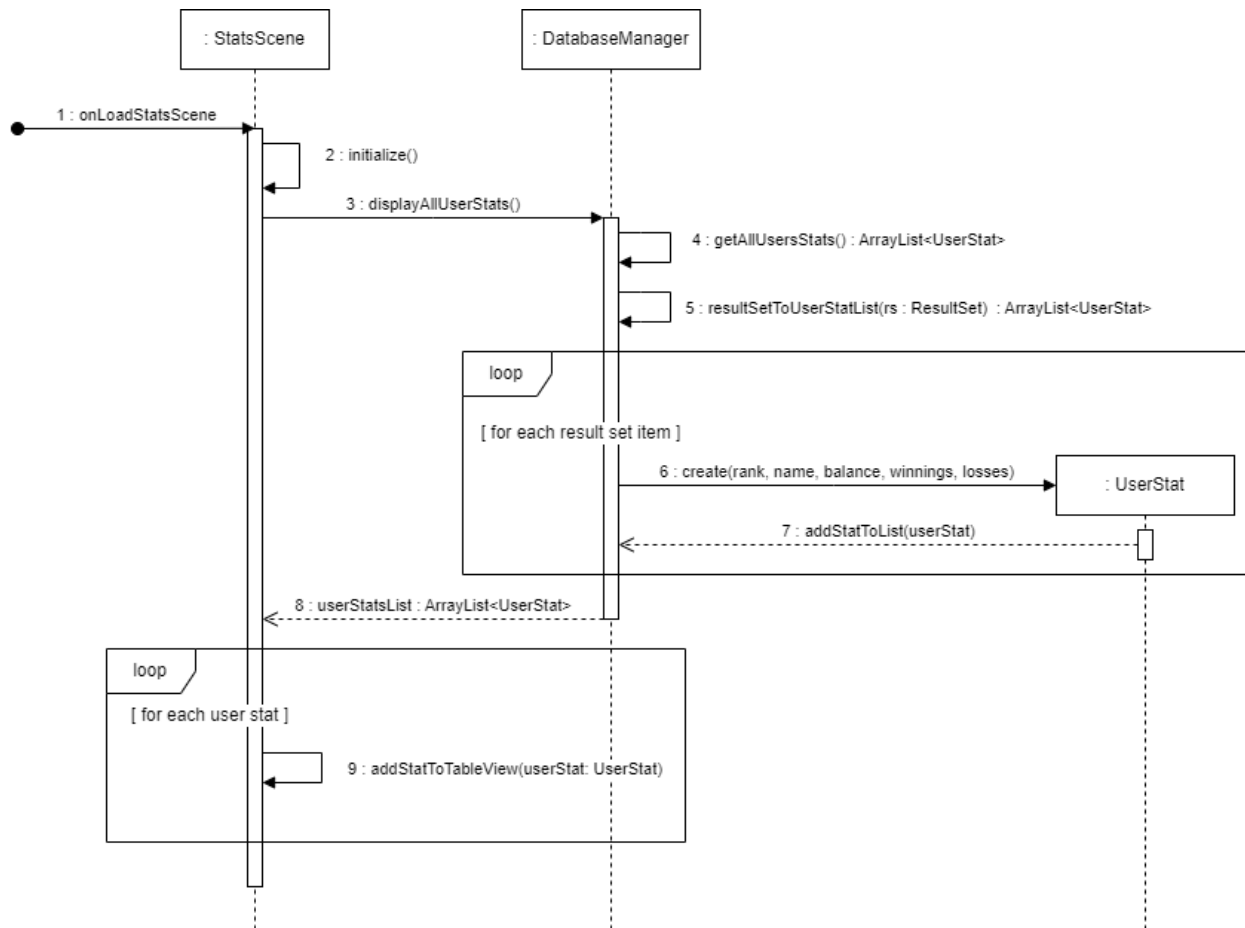
```
"DELETE FROM Wins WHERE User_ID = " + userID
```

Deleting all losses relation for a user account using a user account ID (F015):

```
"DELETE FROM Losses WHERE User_ID = " + userID
```

### Displaying all user stats

The application's process to display all user stats is among the more complicated database processes. The following sequence diagram illustrates the steps the application performs to display data for all users on the stats screen.



*The sequence of displaying user stats*

This process begins when the user enters the application's stats scene. The scene calls an internal method 'initialize()' to initialize the scene, which then calls the 'displayAllUserStats()' method to get and display the user stats from the DatabaseManager. The 'displayAllUserStats()' method then invokes the database's 'getAllUsersStats()' method. This method returns an ArrayList of UserStats, which it obtains by passing a ResultSet to the 'resultSetToUserStatList()' method. The 'resultSetToUserStatList()' method iterates through the provided ResultSet, creates a UserStat instance for each result item, and returns the list of UserStat instances. Once the processes within the DatabaseManager's 'getAllUsersStats()' method conclude, it

returns the list of UserStat instances that the 'displayAllUserStats()' method iterates through, adding each UserStat instance to the stat scene's view.

#### **6.4.4 Blackjack Gameplay Process**

After the player enters the blackjack match scene from the betting scene, the application hands cards to the player and dealer. Afterward, the application displays several options to the player as GUI buttons. Each button performs the same behavior a typical blackjack game provides to players:

- Hit – Gives the player another card.
- Stand – Finishes the player's current hand.
- Double – Doubles the bet on the player's current hand.
- Split – Splits the player's current hand into two separate hands.
- Insurance – Places a side bet on the player's current hand.

As the player selects these buttons throughout the match, the application determines which of them, minus the Stand button, the player can press. At the start of this process, the BlackjackMatch class calls its checkPlayerHand() method, which calls its boolean methods canHit(), canDouble(), canSplit(), canAcceptInsurance(), and canAcceptEvenMoney(). The application then passes the boolean results of each method through the BlackjackScene class's showMatchOptions() method. This method then uses the parameter input of the results to set the visibility of the corresponding buttons based on the boolean results. If the resulting value is true, the application displays the button. Otherwise, the application hides the button. The Insurance button invokes the same behavior if the player can accept regular insurance or even money





which takes the player and dealer's hands as arguments. This class builds three tables from three CSV files that contain blackjack strategy data, then uses the player and dealer's hands to determine whether the player should hit, stand, double their hand's bet, or split their hand. The strategy tables are derived from the Basic Strategy chart provided by the Blackjack Apprenticeship (Blackjack Apprenticeship). The OddsGenerator uses three CSV files of tables: hard totals, soft totals, and pair splitting:

Hard Totals										
	2	3	4	5	6	7	8	9	10	A
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
11	D	D	D	D	D	D	D	D	D	D
10	D	D	D	D	D	H	D	D	H	H
9	H	D	D	D	D	H	H	H	H	H

Soft Totals										
	2	3	4	5	6	7	8	9	10	A
20	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	D	S	S	S	S
18	D	D	D	D	D	S	S	H	H	H
17	H	D	D	D	D	H	H	H	H	H
16	H	H	D	D	D	H	H	H	H	H
15	H	H	D	D	D	H	H	H	H	H
14	H	H	H	D	D	H	H	H	H	H
13	H	H	H	D	D	H	H	H	H	H

Pair Splitting										
	2	3	4	5	6	7	8	9	10	A
11	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
10	N	N	N	N	N	N	N	N	N	N
9	Y	Y	Y	Y	Y	N	Y	Y	N	N
8	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
7	Y	Y	Y	Y	Y	Y	N	N	N	N
6	Y	Y	Y	Y	Y	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N
4	N	N	Y	Y	N	N	N	N	N	N
3	Y	Y	Y	Y	Y	Y	N	N	N	N
2	Y	Y	Y	Y	Y	Y	N	N	N	N
1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

*Basic strategy CSV tables*

The columns of the above tables represent the value of the dealer's face-up card. The rows in the hard and soft totals table represent the sum of the cards in the player's hand, though the OddsGenerator class uses the hard totals table when the player's hand has no Ace cards and the soft totals table when their hand has Ace cards. The rows in the pair splitting table represent the value of the two splittable cards in the player's hand. When the BlackjackMatch class passes the player and dealer's hand through OddsGenerator class's `getBestChoice()` method, the method uses dealer's up-card and the traits of the player's hand to find the value at the intersection of the column and row of a table:

### Hard Totals

	2	3	4	5	6	7	8	9	10	A
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
11	D	D	D	D	D	D	D	D	D	D
10	D	D	D	D	D	H	D	D	H	H
9	H	D	D	D	D	H	H	H	H	H

*Basic strategy example*

In this example, the dealer's up-card has a value of seven, and the player's cards, e.g., 5 and 7, sum to 12. Within the `getBestChoice()` method, the method sets an `int` variable, `index`, to the position of the upcard's value (5), then determines which table to use based on the player's hand. Since the player's hand has no Ace cards, the method chooses the hard totals table. Afterward, the method uses a variable for an `ArrayList` of `Strings`, `options`, to store the `ArrayList` at the index of the player's hand's sum (4). Then, the method uses a `String` variable, `choice`, to store the value in the `options ArrayList` at the position of the value `index` (H). Finally, the method compares the value of `choice` with a set of `Strings` to return a `BestChoice` enum representing the choice. "H" corresponds to the enum `Hit`, so the method returns that enum and highlights the Hit button in the GUI.

## **6.5 Alternate Design Decisions**

While we were determining how we would design the application's front and back end, there were several major design decisions we decided not to employ. If we had utilized these design choices, they would have cost valuable time and effort, increasing the chance of an unsuccessful project. These design areas included our application's GUI, database, animations, and timed processes.

### **6.5.1 GUI**

Before learning JavaFX and SceneBuilder, we considered designing our application's GUI around the Java Swing and AWT frameworks. We had experience using these frameworks to design GUIs for other projects, but most of our experience involved manually creating GUI. We considered the potential cost in time to implement and test a manually generated GUI and opted to search for frameworks that offer some level of assisted/automated GUI development.

### **6.5.2 Database**

Without the SQLite JDBC implementation, we would have considered designing our own file-based database management system. A file-based database may have worked for our application and, with our Java experience, would have required minimal effort to set up. Early in the project, when we were considering designing a web-based application, we may have considered a simple Google Sheets file to act as a database. When we switched to designing a standalone desktop application, an embedded file-based database may have worked for our application since it would reside on the user's PC, and so could provide a simple system for storing data. However, we eventually decided to search for a more robust database system due to potential issues in data inconsistency and redundancy. A relational database management system alleviates

these issues and performs queries faster and more efficiently than a file-based system. We looked for relational database management solutions we could embed within our application and favored the SQLite JDBC implementation due to its simple setup and online resources.

### **6.5.3 Timed Events**

JavaFX's Timeline class facilitated the creation of timed/delayed events. Considering the speed at which most computers operate, these events were necessary to allow users to comprehend the various blackjack gameplay processes, such as adding cards into the player and dealer's hands. We could have used Java's Thread class and Runnable interface without the Timeline class to add the timing element into events. However, we may have needed to experiment to acquire the functionality the Timeline class facilitates, such as calling events at the end of other timed events and creating sets of timed events.

### **6.5.4 Animations**

JavaFX's Timeline class also facilitated the generation of animations for various objects. Coding the animations was simple, and the Timeline class's methods provided our needed capabilities. Without the Timeline class, we could have created animations with the Thread class and Runnable interface. Most of our animations were simple enough that the Thread class and Runnable interface would have been sufficient. However, without the knowledge that animating nodes by translations was convenient for creating animations, we would have needed to figure that out on our own, which could have increased the risks of being unable to implement animations.

## 7.0 Development History

The development history provides information on the additions and adjustments we made to our project during its lifecycle. It focuses on our development of the software system's code, including the additions and changes we made and when we performed them, and then discusses the current state of our project.

### 7.1 Introduction

The development history focuses on the development of the software application itself, which occurred during the construction, testing, and release phases (11/15/23 - 12/5/23). In this section, we will use a numbering scheme to identify version changes, consisting of three numbers separated by periods:

- The first number represents significant changes, such as system component additions or removal.
- The second number represents minor changes, such as sub-system component additions, removal, or changes.
- The third number represents any patches, such as fixes to bugs or security vulnerabilities and other application backend adjustments.

### 7.2 Version History

Version	Description
V-0.0.0 (11/16/23)	<b>Project setup</b> - Setup for basic scenes, scene switching, and passing data between scenes.
V-0.1.0 (11/16/23)	<b>Window resizes</b> - Increased the size of the application window for scenes.
V-0.2.0 (11/17/23)	<b>Setup for betting and blackjack gameplay</b>

Version	Description
	<ul style="list-style-type: none"> <li>- Set up a basic blackjack match. Users can hit or stand. The dealer hits while their hand is below 17. The game indicates who won, if a push happened, and if someone had blackjack or busted.</li> <li>- Added simple guest account functionality that allows users to earn and lose money.</li> </ul>
V-0.3.0 (11/18/23)	<b>Doubling bets &amp; GUI adjustments</b> <ul style="list-style-type: none"> <li>- Added functionality for doubling a bet during the blackjack match. Users can bet if they have not performed a hit and have enough money.</li> <li>- Added a label to display the user's bank account balance in the betting scene.</li> <li>- Adjusted the betting and blackjack GUI to include dollar signs where monetary values appear.</li> <li>- The user's balance adjusts when entering the blackjack scene.</li> </ul>
V-0.3.1 (11/18/23)	<b>Hand instantiation</b> <ul style="list-style-type: none"> <li>- Modified how the application managed the user or dealer's cards to handle their display.</li> <li>- The application now instantiates representations of hands using FXML files to increase their reusability and to facilitate the planned functionality of splitting hands.</li> </ul>
V-0.3.2 (11/19/23)	<b>Card instantiation, hand adjustments, GameObject, and IPlaceable</b> <ul style="list-style-type: none"> <li>- Added a class for instantiating hands and cards.</li> <li>- Modified the cards to manage their data and display.</li> <li>- The deck can now instantiate cards using the FXML files to increase their reusability and to facilitate the planned addition of card and hand animations.</li> <li>- Removed card images from the FXML file of hands, as the FXML file for cards now handles them.</li> <li>- Added an interface for handling placeable objects, like hands or cards.</li> </ul>
V-0.4.0 (11/20/23)	<b>Light GUI polish</b> <ul style="list-style-type: none"> <li>- Implemented a route to the Delete scene from the Login scene.</li> <li>- Added background and layout elements to Start, Registration, Login, Main Menu, Tutorial, Stat, and Delete scenes.</li> </ul>
V-0.4.1 (11/21/23)	<b>Blackjack scene changes</b>

Version	Description
	<ul style="list-style-type: none"> <li>- Separate the logic for the blackjack GUI and gameplay into two classes to increase cohesion.</li> <li>- Fixed a possible bug that prevented users from earning the natural blackjack payout if the dealer didn't bust.</li> <li>- Fixed a possible bug that caused the application to adjust the user's balance twice once the game ended.</li> <li>- Added a class for cases where a process attempts to get the user's object instance before the application sets it.</li> </ul>
(11/21/23)	Initial Blackjack Simulator build release.
V-1.0.0 (11/23/23)	<b>Splitting, insurance, and even money</b> <ul style="list-style-type: none"> <li>- Added GUI elements and code functionality to allow players to split hands (for a maximum of three additional hands), and accept insurance and even money if the dealer has an ace showing and the player hasn't performed other actions.</li> <li>- The GUI indicates if the player lost their insurance bet.</li> <li>- Added a class, gradle.build, and module-info files to enable more compact application deployment.</li> <li>- Changed the datatype for cards in the hand class to use an ArrayList type.</li> <li>- Adjusted the timing of blackjack and bust callouts to occur when the application compares hands.</li> </ul>
V-2.0.0 (11/27/23)	<b>User account creation, login, updates, and deletion</b> <ul style="list-style-type: none"> <li>- Implemented the embedded JDBC SQLite database and functionality for users to create, log into, and delete accounts stored within it. The application also records logged-in users' winnings and losses within the database.</li> <li>- Added password encryption/decryption class for handling passwords.</li> <li>- When a user loses all their money, the application removes their winnings and losses history and resets their balance so they can continue playing. A message also indicates this process occurred once a match concludes or if the player attempts to bet without money.</li> <li>- Updated the stats screen to display all users' usernames, balances, winnings, and losses, and display users in descending order by balance.</li> <li>- Added GUI elements to the main menu to display a logged-in user's name and balance.</li> </ul>



Version	Description
	<ul style="list-style-type: none"> <li>- Updated tutorial screen to include information about splitting and taking using insurance.</li> <li>- Adjusted the bet slider so the user can use keyboard inputs to adjust it in increments of 1 (was 10).</li> <li>- Added a delay to when the application hits unfinished split hands so the player can see them more clearly.</li> <li>- Fixed an issue where the hand status display (for when the hand had blackjack/busted) didn't update when the application compared additional split hands.</li> <li>- Fixed an issue where users should split when their hand had more than two cards.</li> <li>- Converted money-related datatypes to doubles/real values for instances where the player bets an odd amount of money, which fixed an issue where the application incorrectly rounded wins and losses.</li> </ul>
V-2.0.1 (11/27/23)	<p>Comment additions</p> <ul style="list-style-type: none"> <li>- Added comments and documentation comments to classes and methods throughout the program to explain code intentions and processes.</li> <li>- Adjusted the syntax of some statements.</li> <li>- Removed some commented/unused code.</li> <li>- Removed the runnable JAR file from .gitignore so it gets pushed to the repository.</li> </ul>
(11/28/23)	Phase 2 Blackjack Simulator build release.
V-3.0.0 (12/2/23)	<p><b>Odds generation, animations, and other adjustments/fixes</b></p> <ul style="list-style-type: none"> <li>- Implemented an odds generator process for informing the player about the best choices to perform during a blackjack match. It factors in the player and dealer's hands and uses strategy tables (consisting of hard and soft totals and pair splitting) to determine whether the player should hit, stand, double, or split. The application highlights the button corresponding to the best choice m.</li> <li>- Added animations to show cards moving to the player or dealer's hand and when the dealer reveals their card. Hands use animations for split hands that move across the scene.</li> <li>- Disabled the player's button options while cards and hands animate to prevent actions from detrimentally impacting routines.</li> </ul>

Version	Description
	<ul style="list-style-type: none"> <li>- The application distinguishes natural blackjacks against hands whose cards equal 21 but have more than two cards.</li> <li>- If the player's current hand gets a natural blackjack, 21, or busts, their hand will immediately finish, and the application will get the next unfinished split hand or play the player and dealer's hands.</li> <li>- If the player gets a natural blackjack and the dealer has an Ace showing, the player can either accept even money or stand. Otherwise, the game continues.</li> <li>- Adjusted bet doubling so the player can double their current hand's bet if it has exactly two cards and hasn't exceeded a value of 20.</li> <li>- The player can use insurance or even money if their current hand has two cards. If the player uses either option, the dealer will now check their card only after the player finishes all their hands.</li> <li>- Updated tutorial to reflect adjusted gameplay.</li> <li>- Implemented performance optimizations for card creation to decrease the time to start blackjack matches.</li> <li>- Simplified the code used when initially dispersing cards to the player and dealer and for playing the other player's hands.</li> <li>- Adjusted the backend code to use a single insurance button and switched the button's text if the player can accept even money or only insurance.</li> <li>- Added code to module-info file to allow clients to access the constructor for the class that displays hand values.</li> <li>- Adjusted internal conditional checks to reuse the newly implemented checks within the hand class.</li> <li>- Reduced the internal code used for instantiating hands and cards.</li> <li>- Reduced the code used when a player selects the option to stand.</li> <li>- Reduced the code in the controller class for the Delete scene.</li> <li>- Removed unused code from the deck class.</li> <li>- Fixed an issue where the player could bust on their last unfinished split hand and prevent the dealer from hitting.</li> <li>- Fixed an issue where the application didn't correctly adjust the value of Ace cards if it evaluated them before the hand value surpassed 21.</li> <li>- Fixed an issue where the player couldn't split Ace cards due to varying values.</li> <li>- Fixed an issue where the player could receive insurance payments for hands they did not request insurance on.</li> </ul>

Version	Description
V-3.1.0 (12/4/23)	<b>CSS and imagery additions</b> <ul style="list-style-type: none"> <li>- Implemented CSS and imagery to various GUI scene elements, including buttons, text links, and GUI bordering.</li> <li>- Added match banner GUI elements for blackjack matches to show when the player or dealer's hands lost, won, busted, had blackjack, or pushed.</li> <li>- Added background blackjack board imagery that shows information about the payout of blackjack, insurance, and when the dealer stops hitting.</li> <li>- Added option to allow a player to return to the main menu if they logged in as a guest or player to avoid the need to re-log in beforehand.</li> <li>- Fixed an issue where the player could spam buttons during matches.</li> </ul>
V-3.1.1 (12/5/23)	<b>Blackjack rounding adjustment</b> Fixed an issue where the application incorrectly rounded blackjack wins to integers.
V-3.1.2 (12/5/23)	<b>JAR update</b> Updated the JAR file for the rounding fix and added the .bat file to the .gitignore so that application updates would push it to the repository.
V-3.1.3 (12/5/23)	<b>Insurance adjustments</b> <ul style="list-style-type: none"> <li>- Added corrections to how the application recorded winnings and losses related to insurance payouts.</li> <li>- Fixed an issue where the player could receive insurance if their hand busted after they accepted insurance.</li> </ul>
(12/5/23)	Final Blackjack Simulator build release.

### 7.3 Current Version

The Blackjack Simulator application is currently working under version 3.1.3. In this version, the software application gives users various choices during the blackjack gameplay, utilizes a database and password handlers for managing accounts securely, and uses CSS, animations, and imagery to enhance the GUI appearance. Since we

have finished implementing the requirements and features we planned to, no further changes are needed. Any further updates we apply to our application will lead to a major version change.

## **8.0 Summary**

In summation, we will discuss the project's technical decisions, actual schedule, cost burn, and delivery schedule.

### **8.1 Technical Decisions and Outcomes**

In developing the Blackjack Simulator, key technical decisions were instrumental in shaping the project's architecture and functionality. Opting for JavaFX for the graphical user interface (GUI) provided a platform-independent solution with rapid development capabilities through tools like SceneBuilder. Security measures were prioritized, employing encryption techniques such as PBKDF2, SHA-256, and AES for password protection. The choice of SQLite as the database engine, coupled with a well-designed relational database schema, facilitated efficient data access and manipulation through Java Database Connectivity (JDBC). The application also integrated blackjack strategy data using CSV-based tables and OddsGenerator, enhancing the authenticity of gameplay decisions.

Using JavaFX's Timeline class for timed events and animations improved the user experience and streamlined the implementation of complex processes. The overall emphasis on leveraging existing technologies, such as established encryption methods and the SQLite database engine, underscored the project's commitment to efficiency and security. Additionally, considerations of alternative design choices, such as opting for JavaFX over Java Swing and AWT, reflected a strategic approach to technology

selection. The technical decisions made throughout the project underscored the importance of adopting proven tools and methodologies to achieve a robust and user-friendly application.

## **8.2 The Actual and Delivery Schedule**

The planned delivery schedule was established based on initial project planning, considering tasks, dependencies, resource availability, and other factors. As the project progressed, tasks were executed according to the original plan, and actual delivery dates were recorded at each phase or milestone. The outcome reflects how closely the project adhered to the initial plan, indicating whether tasks were completed on time and if adjustments were made to meet changing project requirements.

Several insights were gained from comparing the planned and actual delivery schedules. The accuracy of the initial planning was assessed by evaluating variations between the planned and actual schedules. Identification of patterns in delays or accelerations highlighted potential bottlenecks or areas of efficiency within the project. The adaptability of the project team in responding to changes and unforeseen obstacles was assessed, providing insights into the team's resilience. The experience with the actual delivery schedule contributes to continuous improvement in project management practices, allowing teams to refine planning processes, improve resource allocation, and enhance overall project execution in future projects. Reflecting on the actual delivery schedule enhances the team's ability to plan effectively, respond to challenges, and achieve successful outcomes in subsequent projects.

### **8.3 Cost Burn**

Earned Value Management (EVM) is a project management approach that integrates scope, schedule, and resource measurements to objectively assess project performance and control. Throughout the project, EVM metrics such as Planned Value (PV), Earned Value (EV), and Actual Cost (AC) were regularly calculated and monitored, providing a comprehensive view of performance in terms of schedule and cost.

The outcome of employing EVM was a detailed analysis of project performance, offering insights into schedule adherence, budget compliance, and overall performance expectations. The approach contributed to a well-informed understanding of the project's health and progress. Valuable lessons learned from EVM included the importance of integrating schedule and cost metrics for a holistic project view. It emphasized the necessity of accurate and timely data to ensure the reliability of EVM metrics and highlighted the value of proactive management based on objective performance indicators. See Appendix E.

### **8.4 Gantt Metrix**

The Gantt Chart, intended as a visual representation of the project schedule, effectively served its purpose throughout the project lifecycle. It provided a comprehensive view of tasks, dependencies, and timelines, aiding both the team and stakeholders in understanding the project's sequence and durations. Regular updates to the Gantt Chart accommodated changes and ensured it remained a reliable project management tool. The outcome was a well-managed project closely aligned with the planned schedule. The Gantt Chart played a vital role in tracking progress, identifying critical paths, and efficiently managing resources. Its visual representation facilitated effective communication with stakeholders, fostering better understanding and expectation management. The Gantt Chart also proved valuable as a historical record, enabling post-project analysis for insights into successful practices and areas for improvement in future

projects. Overall, the Gantt Chart significantly contributed to enhanced project management practices and successful project delivery. See Appendix E.

## **9.0 Conclusions**

While developing the Blackjack Simulator software application, we learned several valuable lessons that have positive implications for future projects. One prominent lesson revolves around the importance of clear and detailed design specifications. Ambiguities or oversights in the early stages of design can lead to inefficiencies and challenges during development. Therefore, meticulous attention to detail in the system specification, including precise requirements and well-defined design guidelines, is crucial for streamlining development and ensuring seamless implementation.

The design strengths of the Blackjack Simulator lie in its user-centric approach, which provides an engaging and educational experience for players. The comprehensive scope, encompassing GUI design, account management, tutorials, and statistical tracking, contributes to a well-rounded application. The adherence to design precedents for GUI and account functionalities ensures familiarity for users, facilitating a more intuitive interaction with the application. Using JavaFX for the front end and JDBC for database connectivity demonstrates a robust technological foundation, enhancing the application's responsiveness and reliability.

However, we identified certain limitations during the design and development phases. The constrained development timeline, spanning only three weeks, imposed challenges in accommodating all desired features and refining the application thoroughly. The decision to limit the software to Windows devices and operate as a standalone desktop application without network connectivity may limit accessibility for

users preferring alternative platforms or seeking online multiplayer functionality. Also, the application might not be as strong when better security is needed because the security measures are not as strong, such as not having specific password format guidelines or recovery features.

To improve the Blackjack Simulator for future iterations, we can expand its compatibility to other operating systems, allowing a broader user base to access the application. The integration of network capabilities could enhance the gaming experience by introducing multiplayer functionalities and fostering user interaction beyond a standalone environment. Addressing security concerns by implementing more sophisticated measures, like password complexity requirements and recovery features, would fortify the application against potential vulnerabilities. Furthermore, a more extended development timeline could facilitate the inclusion of additional features and thorough testing, ensuring a more polished and refined final product. Overall, these suggestions aim to enhance the versatility, security, and overall user experience of the Blackjack Simulator in future iterations.

### **Individual Lessons Learned:**

#### **Jordan DeLaGarza:**

The topics within this course that I found the most interesting and applicable would have been the documentation required for making a program, the communication required with teammates, and the different ways to utilize code collaboration. When looking back at the role documentation played in mapping out exactly what was expected from the program and how that would be achieved was a new and interesting part of programming I had never prioritized to that extent before. It all really helped



establish a clear and concise image of what was expected from the program and how it should be tested moving forward which in the long run most likely saved lots of time with having to come up with some of those things on the fly. I think if I decide to choose a future in software development those documentation skills will definitely pay off in the long run with understanding the bigger picture of the SDLC cycle. Communication was also a huge thing that was of major importance to my understanding of working within the new programming team dynamic. This was highlighted when it came to meeting deadlines and outlining what needed to be done to move forward to the next phase and properly prepare ourselves to be on the right track for finishing the project on time. Our team was very effective with this by always keeping each other up to date on the current status of the milestones we were working on and constantly reviewing each other's work to ensure it was being submitted with no errors. This type of communication is not only something to apply to my future career but also all aspects of my life. The last thing I think I found the most interesting would be the way we were able to share and review code. We used GitHub when it came to the development of our project. This method made it extremely easy to make documentation based on the most recent code because it was constantly being updated to a main source where all team members could access it. It wasn't something I had really thought of prior to this course because I always worked exclusively on projects myself within the Eclipse IDE or Visual Studio Code. This new method made it extremely easy to monitor and track new features and changes that were being implemented to the code in the day to day development. I think this would be very practical to any future job when it comes to code development because it most likely reflects how things are done at most companies instead of just

constantly sending people new code every single time. Overall all the topics I have learned throughout this course have developed me both as a programmer and as a person.

**Samuel Hudgins:**

The time we spent planning the system's design helped me to design its structure in code quickly. The planning phases also helped me discover frameworks, such as JavaFX and SQLite, that significantly reduced the complexity and time required to implement GUI scenes and the database.

The hand-splitting feature was the most challenging aspect of this program, though it could have been more difficult. I originally embedded the hands and cards within the application's game scene and activated/deactivated specific card images in each hand when they received cards. Although this setup worked, the hand-splitting functionality would have been problematic, as we would have had to create and adjust separate panes for cards manually. I eventually decided to treat the hands less as individual sets of images and more as prefabricated entities. This setup allowed us to instantiate hands similarly to instantiating class instances, adjust all hand instances when modifying the base's setup, and move the hand itself to change the placement of its cards. This setup also required less code and complexity than keeping track of individual card panes and the list of images within them and swapping the placement of cards around.

The animations were also challenging to implement. I used JavaFX's 'Timeline' class since it provided a straightforward process for creating animated events, though I had difficulty getting hands and cards to animate to specific positions. JavaFX's

documentation on how animations apply to scene nodes' translation values helped, but I still had issues getting objects to animate from one position to another. I eventually adjusted how the placeable objects internally calculated positions to position themselves against the center of the scene rather than from the initial upper left-hand corner. This adjustment helped me understand the objects' placement and animate them to the correct locations.

**Patrick Smith:**

Engaging in the Software Development Life Cycle (SDLC) process has been a fascinating journey that highlighted the importance of collaboration in achieving successful outcomes. Successfully navigating the complexities of the SDLC demands a synergy of technical expertise and project management proficiency. Working collaboratively within a group setting has been particularly enlightening, revealing the unique capabilities and limitations of each individual. Through this collaborative effort, I've come to appreciate the exchange of knowledge among team members, recognizing that collaboration enhances the capabilities of individuals and fortifies the team as a whole.

The benefits of working together as a team are extensive. Collaborative efforts lead to enhanced knowledge, improved capabilities, and increased efficiency. The pooling of diverse perspectives and ideas within a group setting results in more innovative solutions and creative problem-solving. Moreover, the camaraderie and mutual support cultivated in a collaborative team environment contribute to a positive work atmosphere, fostering motivation and boosting overall productivity.

Fostering a supportive and inclusive team environment has been equally crucial. This positive team culture encourages the open sharing of ideas, respects diverse perspectives, and promotes collaboration among team members. In this inclusive setting, everyone feels valued and heard, leading to increased creativity and innovation. Additionally, the establishment of strong relationships and trust among team members enhances overall team cohesion and productivity.

As I continue to build on my technical expertise and project management skills, the collaborative experiences gained from working in a group setting will undoubtedly contribute significantly to my future success. Beyond expanding my knowledge, these experiences have shaped me into a well-rounded individual ready to tackle any challenge that comes my way.

**Robert Carswell:**

Mastering the intricacies of managing a software project, collaborating within a team, and utilizing project management tools has been an enlightening experience. These skills are not only fundamental for current endeavors in software development but hold lasting importance for future work in various industries where project management is a key component. Understanding project management tools, whether it's Microsoft Project for development operations, version control tools like Git, or integrated development environments (IDEs) like Eclipse, is crucial for their effective application. Proficiency in these tools ensures a smooth workflow and successful project completion. The ability to navigate their capabilities and limitations is paramount for professionals aiming to excel in project management, contributing to streamlined processes, improved communication, and enhanced productivity. So, investing time and effort into learning about these tools gave me the ability to streamline processes,

eliminate bottlenecks, and meet deadlines more efficiently, ultimately leading to increased productivity and client satisfaction. At the same time allowing us to deliver the high-quality results that foster success.

## Appendix A - Project Charter

Project Name: Blackjack Simulator
Date: October 28th, 2023
Project Manager: Robert Carswell
Project Sponsor: Terrence Mentzos
Requested Completion Date: October 31st, 2023
<p>Project Justification:</p> <p>The purpose of this project is to provide a casual yet informative blackjack experience. Blackjack is one of the games that casinos offer and can involve losing real money. Some people want to learn how to play and enjoy casino games without the risks of real monetary impacts, so we want to design that experience. Along with adding features to help players learn how to play blackjack, we want to provide statistical information to help players win while recording and displaying data about the player's play history to help them win.</p>

Project Overview
<ol style="list-style-type: none"> <li>1. Project Plan</li> <li>2. Test Plan</li> <li>3. Project Design</li> <li>4. Phase 1 Source: classes</li> <li>5. Phase 2 Source: methods</li> <li>6. Phase 3 Source: user experience and modules</li> <li>7. Final deliverable</li> </ol>

Approvals			
Title	Name	Signature	Date
Project Sponsor	Terrence Mentzos		
Project Manager	Robert Carswell		

## Appendix B - Project Team

Project Name: Blackjack Simulator
Date: October 28th, 2023
Project Manager: Robert Carswell

Role	Name	Contact
Project Manager (PM)	Robert Carswell	Robert.carswell34@gmail.com
Requirements Manager/Technical Writer (RM/TE)	Jordan DeLaGarza	Jordandelagarza2001@gmail.com
Software Designer (SD)	Jose Reyes	Jose.Reyesfabian@gmail.com
Software Designer (SD)	Samuel Hudgins	Samuel_h_713@msn.com
Test Director	Patrick Smith	Bblackwave5@gmail.com

## **Appendix C - Statement of Work**

### **Scope**

This project aims to develop a comprehensive simulation of a blackjack game experience while maintaining a record of the player's statistical history. The primary objective is to create a user-friendly software application that provides an engaging and educational experience for players while offering a statistical history feature to track and analyze gameplay performance.

### **Location of Work**

This project does not require in-person collaboration, so we will complete this project and meet up virtually using personal devices.

### **Timeline**

The work timeline is between September 18, 2023, and December 12, 2023, and we will deliver our final project by December 12, 2023.

### **Acceptance Criteria**

The acceptance criteria, as agreed upon with the client, are as follows:

1. Given that the user has no experience in blackjack, the application will use an interactive user interface and graphics to allow the user to practice playing.
2. During gameplay, given that the user has a set of cards, the application will present statistical information about their odds of winning given the dealer's cards.
3. If the user wins or loses a match, the application will record and update their total monetary wins and losses.

### **Software and Hardware Requirements**



In terms of hardware, the client will need a Windows personal computer to run the software application. The client will also need software on their computer that enables them to run a JAR (.jar) file.

**Appendix D – Schedule**

Week	Topics	Lead(s)	Assignment(s) due at end of week
1 10/18/23 - 10/24/23	<ul style="list-style-type: none"> <li>- Meet with team members</li> <li>- Pick a topic</li> <li>- Select development environment and tools</li> </ul>	Everyone	N/A
2 10/25/23 - 10/31/23	<ul style="list-style-type: none"> <li>- Describe project's purpose</li> <li>- Identify project scope, costs, and schedule</li> <li>- Identify system specifications</li> <li>- Outline Milestones</li> <li>- Delegate responsibility</li> </ul>	<ul style="list-style-type: none"> <li>- Project Manager</li> <li>- Requirements Manager / Technical Writer</li> </ul>	Project Plan
3 11/1/23 - 11/7/23	<ul style="list-style-type: none"> <li>- Report project status</li> <li>- Determine requirement change management plan</li> <li>- Create test plan</li> <li>- Develop configuration management plan</li> <li>- Design draft for user's guide</li> </ul>	Test Director	<ul style="list-style-type: none"> <li>- Test Plan</li> <li>- Peer Review 1</li> </ul>
4 11/8/23 - 11/14/23	<ul style="list-style-type: none"> <li>- Report project status</li> <li>- Update requirements traceability matrix</li> <li>- Perform unit tests</li> <li>- Create preliminary software application design</li> <li>- Design user interface diagrams</li> </ul>	Software Designer User Experience / Training Manager	Project Design
5 11/15/23 - 11/21/23	<ul style="list-style-type: none"> <li>- Report project status</li> <li>- Update requirements traceability matrix</li> <li>- Perform component tests</li> <li>- Create detailed software application design</li> <li>- Develop software code</li> <li>- Design prototype user interface</li> </ul>	Everyone	<ul style="list-style-type: none"> <li>- Phase 1 Source</li> <li>- Peer Review 2</li> </ul>
6 11/22/23 - 11/28/23	<ul style="list-style-type: none"> <li>- Report project status</li> <li>- Update requirements traceability matrix</li> <li>- Perform system tests</li> <li>- Create physical software application design</li> <li>- Develop software code</li> </ul>	Everyone	Phase 2 Source

Week	Topics	Lead(s)	Assignment(s) due at end of week
	- Design final user interface		
7 11/29/23 - 12/5/23	<ul style="list-style-type: none"> <li>- Develop project report</li> <li>- Create requirements specification</li> <li>- Display test plan and results</li> <li>- Indicate system specification design and alternate designs</li> <li>- Develop software code</li> <li>- Design user's guide</li> </ul>	Everyone	Phase 3 Source
8 12/6/23 - 12/12/23	Compile all Topics into a single document	Everyone	<ul style="list-style-type: none"> <li>- Peer Review 3</li> <li>- Final</li> </ul>



## Blackjack Simulator

### Earned Value Analysis Report

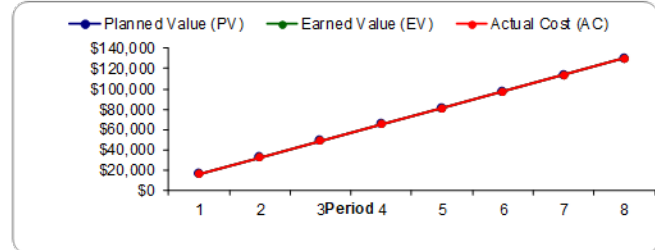
Dynamic Gaming

Prepared By: Robert Carswell

Date: 12 December 2023

For Period: <sup>30/11</sup> Week 8**Summary:**

CV and SV are within bounds to sustain an optimal CPI and SPI.

**Planned Value (PV) or Budgeted Cost of Work Scheduled (BCWS)**

WBS	Task Name	TBC	1	2	3	4	5	6	7	8	9	10	11	12
1.1.1	Assign Roles	8134	8134											
1.1.2	Acquire Sponsor Commitment	8134	8134											
1.2.1	Project Plan	8134		8134										
1.2.2	Requirements List	8134		8134										
1.3.1	Change Management Plan	4067			4067									
1.3.2	Test Plan	4067			4067									
1.3.3	Configuration Management Plan	4067			4067									
1.3.4	User Guide Draft	4067			4067									
1.4.1	Project Status Update	5423				5423								
1.4.2	Traceability Matrix	5423				5423								
1.4.3	Preliminary Design	5423				5423								
1.5.1	Project Status Update	4067					4067							
1.5.2	Traceability Matrix Update	4067					4067							
1.5.3	Detailed Design	4067					4067							
1.5.4	Phase 1 Code	4067					4067							
1.6.1	Project Status Update	4067						4067						
1.6.2	Traceability Matrix Update	4067						4067						
1.6.3	Physical Design	4067						4067						
1.6.4	Phase II Code	4067						4067						
1.7.1	Project Report (Final)	2711							2711					
1.7.2	Requirements Specification (Final)	2711							2711					
1.7.3	Test Plan and Results (Final)	2711							2711					
1.7.4	System Specification Design (Final)	2711							2711					
1.7.5	Phase III Code (Final)	2711							2711					
1.7.6	User Guide (Final)	2711							2711					
1.8	Final Project Delivery	16268								16268				

Insert new rows above this one

<b>Total Budgeted Cost</b>	<b>130140</b>	16268	16268	16268	16268	16268	16268	16268	16268	16268				
<b>Cumulative Planned Value (PV)</b>		16268	32535	48803	65070	81338	97605	113873	130140					

**Actual Cost and Earned Value**

<b>Cumulative Actual Cost (AC)</b>	16268	32535	48803	65070	81338	97605	113873	130140						
<b>Cumulative Earned Value (EV)</b>	16267.5	32535	48802.5	65070	81337.5	97605	113872.5	130140						

**Project Performance Metrics**

<b>Cost Variance (CV = EV - AC)</b>	0	0	0	0	0	0	0	0	0					
<b>Schedule Variance (SV = EV - PV)</b>	0	0	0	0	0	0	0	0	0	0				
<b>Cost Performance Index (CPI = EV/AC)</b>	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00				
<b>Schedule Performance Index (SPI = EV/PV)</b>	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00				
<b>Estimated Cost at Completion (EAC)</b>	130140	130140	130140	130140	130140	130140	130140	130140	130140					

## Appendix F - Scope Baseline

Scope Baseline: Blackjack Gameplay	
Item	System Requirement Notes
1a	Can the user place a bet?
1b	Can the user perform a Hit?
1c	Can the user perform a Stand?
1d	Can the user double their bet?
1e	Can the user perform a Split?
1f	Can the user accept Insurance?
1g	Can the user accept even money if they have blackjack and the dealer has an ace showing?
1h	Can the dealer perform a Hit if the player performs a Stand?
1i	Can the dealer continue to Hit if their hand value is less than 17?
1j	Can the user win if their hand's total exceeds the dealer's?
1k	Can the user win if the dealer busts?
1l	Can the user lose if their hand's total exceeds 21?
1m	Can the user lose if their hand's total is lower than the dealer's?
1n	Can the user tie (push) with the dealer if their hands are 21?
1o	Can the user win money if they beat the dealer?
1p	Can the user lose money if the dealer beats them?
2a	Does the application display a start screen?
2b	Does the application display a bet screen?
2c	Does the application display the screen where the blackjack match takes place?
2d	Does the application display a game over screen?
2e	Does the application show a representation of the dealer?
2f	Does the application show representations of cards?
2g	Does the application indicate the total value of the player's cards?
2h	Does the application indicate the total value of the dealer's cards?
2i	Does the application display a user interface element that indicates whether the user should Hit or Stand?
2j	Can the player access a screen that allows them to view data about their past games?
3a	Can the user register an account by providing a username and password?
3b	Can the user log into a registered account with their username and password?
3c	Does the application store the user account data?

Scope Baseline: Blackjack Gameplay	
Item	System Requirement Notes
3d	Is the user's password encrypted?
3e	Does the application store the user's money in their user account?
3f	Does the application update the user's stored money when they win a blackjack match?
3g	Does the application update the user's stored money when they lose a blackjack match?
3h	Does the application store data about the player's winnings?
3i	Does the application update the player's winnings if they win a blackjack match?
3j	Does the application store data about the player's losses?
3k	Does the application update the player's losses if they lose a blackjack match?

## Appendix G - Change Request Form

SUBMITTER - GENERAL INFORMATION				
CR#				
Submitter Name				
Brief Description of Request				
Date Submitted				
Date Required				
Priority	Low	Medium	High	Mandatory
Reason for Change				
Other Artifacts Impacted				
Assumptions and Notes				
Attachments or References	Yes	No		
	Link:			

INITIAL ANALYSIS		
Hour Impact		
Duration Impact		
Schedule Impact		
Comments		
Recommendations		

CHANGE CONTROL BOARD - DECISION				
Decision	Approved	Approved w/Conditions	Rejected	More Info
Decision Date				
Decision Explanation				
Conditions				



## Appendix H - Document References

The following table summarizes the documents referenced in this document.

Document Name	Version	Description	Location
Project Plan	1.0.0	Provides the project's purpose, scope, and members and outlines the process for achieving project goals.	<a href="https://docs.google.com/document/d/1odORs2VYeSzzONOs4kiUUVkEe_SbkZGnlb3zQBoQYk/edit?usp=sharing">https://docs.google.com/document/d/1odORs2VYeSzzONOs4kiUUVkEe_SbkZGnlb3zQBoQYk/edit?usp=sharing</a>
Software Requirements Document	1.0.0	Describes the project's requirements and discusses assumptions and constraints around the project.	<a href="https://docs.google.com/document/d/1A0Qx97-KMaU-qclehVgPfwOo6OEN52UJ-euaOkfEWNwM/edit?usp=sharing">https://docs.google.com/document/d/1A0Qx97-KMaU-qclehVgPfwOo6OEN52UJ-euaOkfEWNwM/edit?usp=sharing</a>
User Guide	1.0.0	Outlines how users interact with the Blackjack Simulator application.	<a href="https://docs.google.com/document/d/1fu6sEsR55B-ALG5OBPPILaAQZZ2J-QCd/edit">https://docs.google.com/document/d/1fu6sEsR55B-ALG5OBPPILaAQZZ2J-QCd/edit</a>
Requirements/Features List	1.0.0	Lists and describes the requirements the project must fulfill.	<a href="https://docs.google.com/spreadsheets/d/1kL9UFM9xcEq8ZRJMzHJPWiFj9jpnRVC0eQDL3NYsTs8/edit#gid=0">https://docs.google.com/spreadsheets/d/1kL9UFM9xcEq8ZRJMzHJPWiFj9jpnRVC0eQDL3NYsTs8/edit#gid=0</a>

## **Appendix I - User's Guide**

### **I.1 What is the Blackjack Simulator Game?**

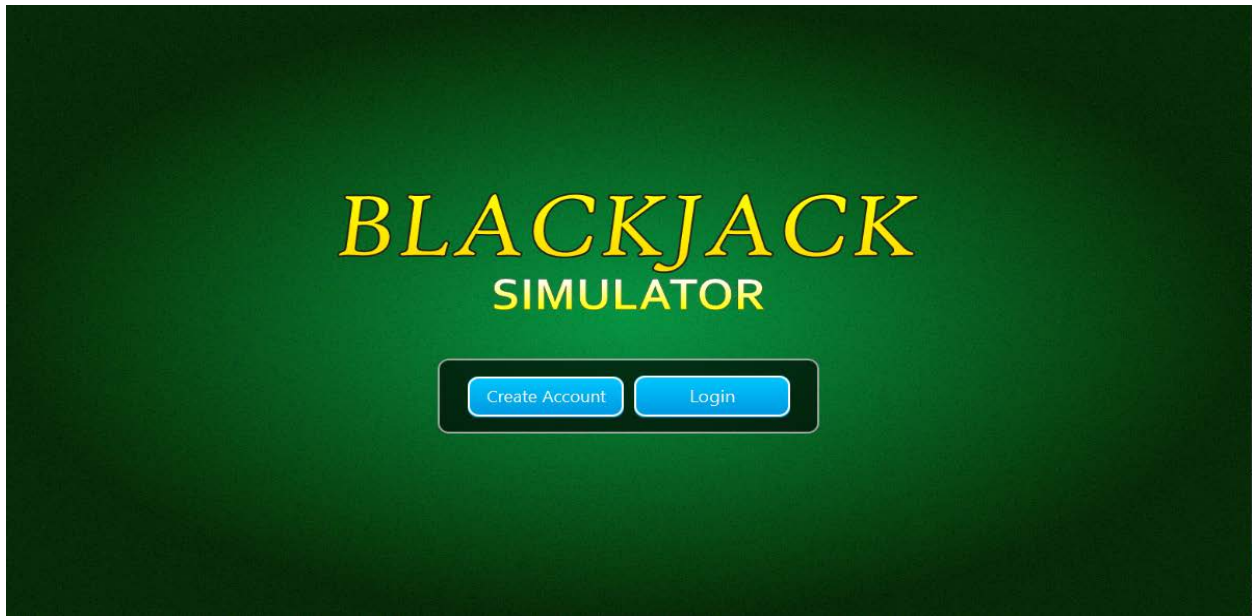
The Blackjack Simulator game was created by Dynamic Gaming to educate users who want to learn blackjack in a fun and safe environment. The game is loaded with features that help the user understand the basics of blackjack and how to read the game to achieve the best results in a real-world environment. Many of our users have no intention of using the knowledge gained from our game in the real world. No worries, this game also provides plenty of competition by allowing multiple players to use the application to keep track of individual success and rank each user. This game is Player vs CPU, but after each player logs into the system, their scores are kept locally, allowing each player to be ranked on their local machine.

### **I.2 Minimum System Requirements**

1. Requires Windows 10.
2. Java Version 8.

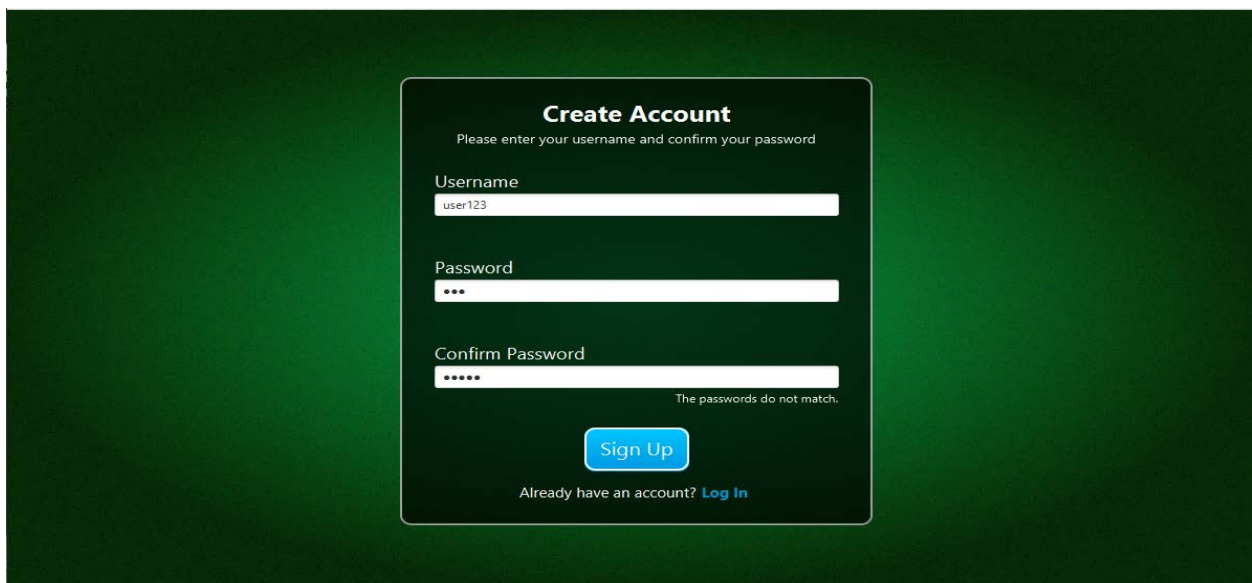
### **I.3 How to Create a User Account**

To create a user account, select the link "Create User Account" on the **Login Screen**. At this point, the user will be directed to the "Create User Login page". Enter the username and password and select Confirm to create a user account.



*Image 1: Startup Screen*

Once the user is directed to the “**Create Account**” screen they can enter their Username and password. If matching passwords are not entered, the message “**The Passwords do not Match**” will be displayed on the screen. If the passwords match the user will be directed to the “Main Menu”.



*Image 2: Passwords do not match*

## I.4 How to Log Into The Game

On the “**Login Screen**,” the user can type their **Username** and **Password** to gain access to the system. The user will be prompted with a message if the username or password is incorrect; otherwise, the user will be directed to the main menu screen.

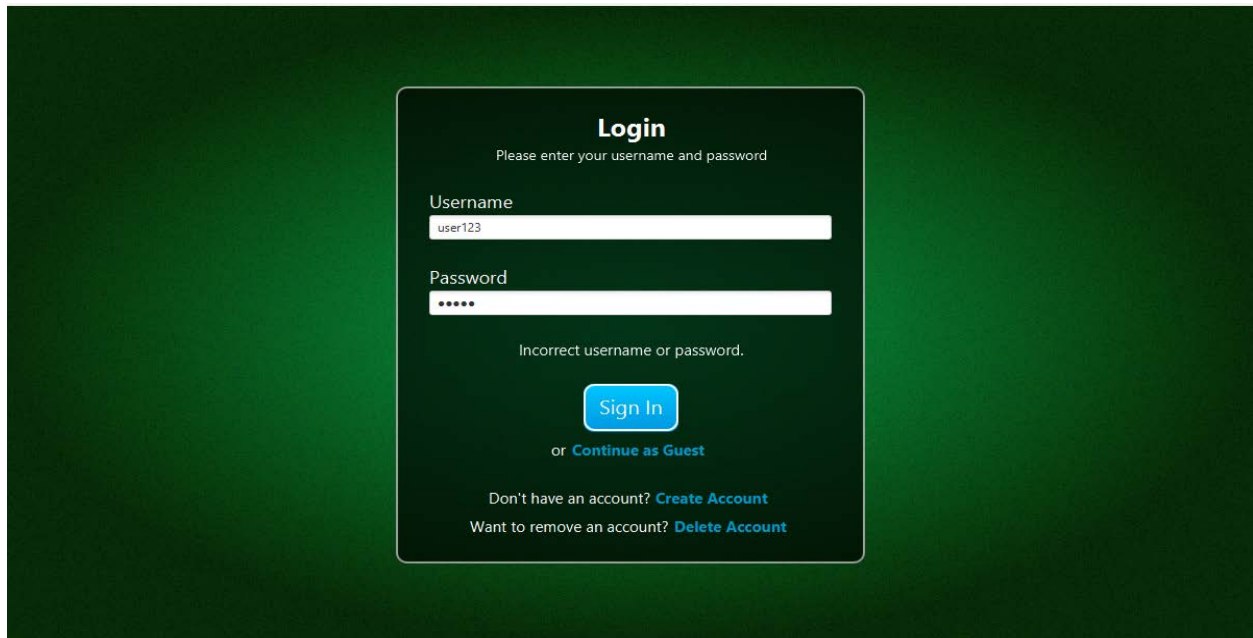


Image 3: *Login Screen with incorrect login information*

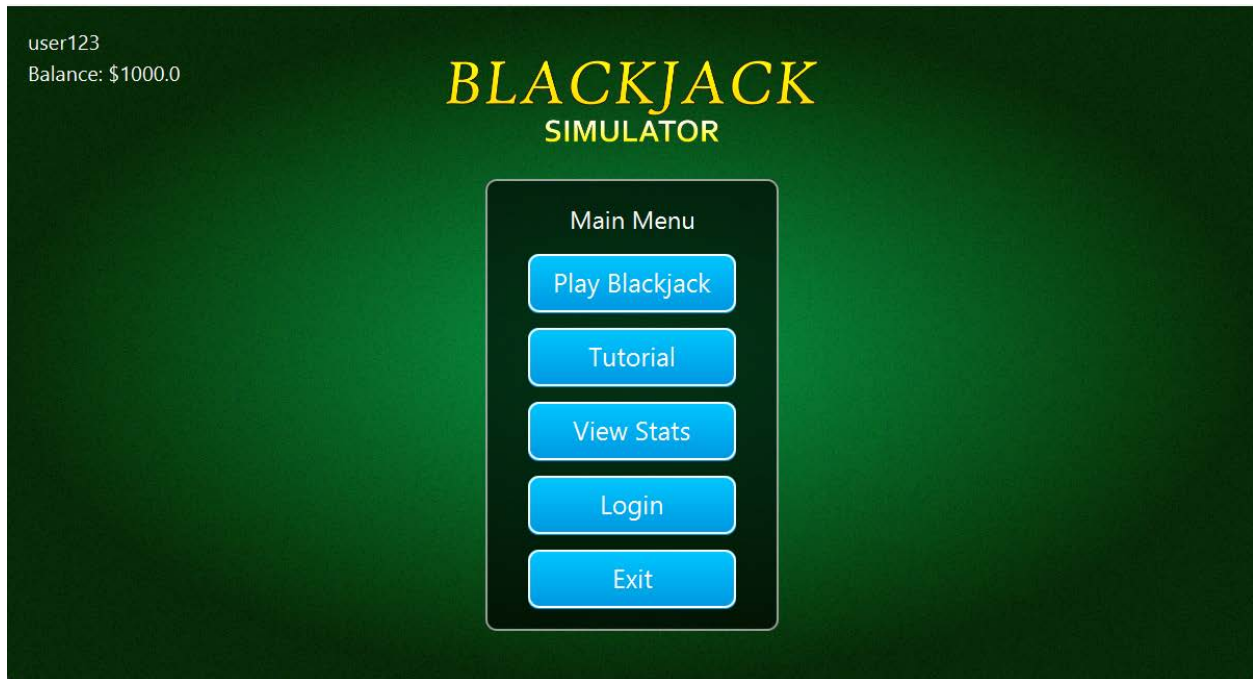
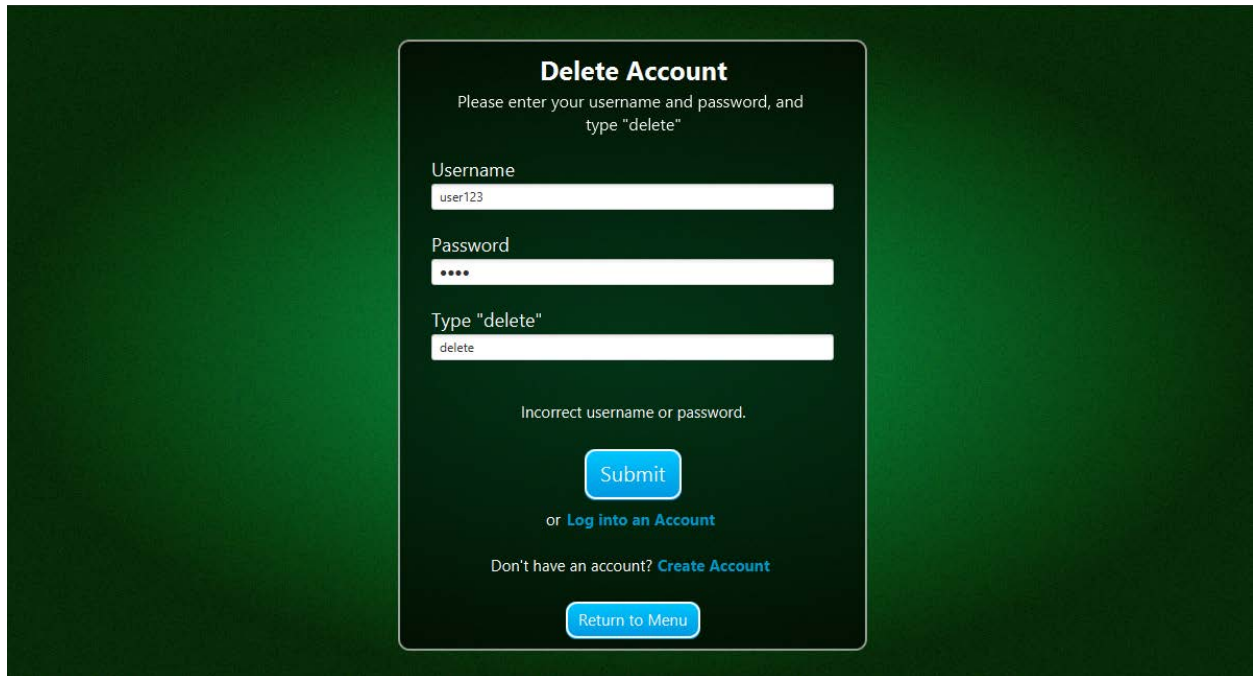


Image 4: Main Menu after successful login

The user can also play as a guest. When the user plays as a guest no data will be saved to the database.

### I.5 Delete User Account

Once a user has determined they no longer want to use a specific profile, they do have the option to delete an account. The user must select the link “**Delete Account**” on the login screen. On this screen, the user must enter their username and password and type “**delete**”. Once they select the submit button, the entire user profile will be removed from the database.



The image shows a 'Delete Account' form on a dark green background. The form is a light gray rounded rectangle. At the top, it says 'Delete Account' in bold. Below that, it says 'Please enter your username and password, and type "delete"'. There are three input fields: 'Username' with 'user123', 'Password' with four dots, and 'Type "delete"' with 'delete'. Below the fields, there is a red error message: 'Incorrect username or password.' Below the error message is a blue 'Submit' button. Below the button, it says 'or [Log into an Account](#)'. At the bottom, it says 'Don't have an account? [Create Account](#)'. At the very bottom is a blue 'Return to Menu' button.

**Image 5:** *Wrong username or password entered*

## I.6 Game Play

### I.6.1 Bank Balance

Once the user has created a profile, the user will start with a balance of \$1000.00. Any earnings and losses will be reflected in the user's bank account balance and carried over when the user logs into the system for future play.

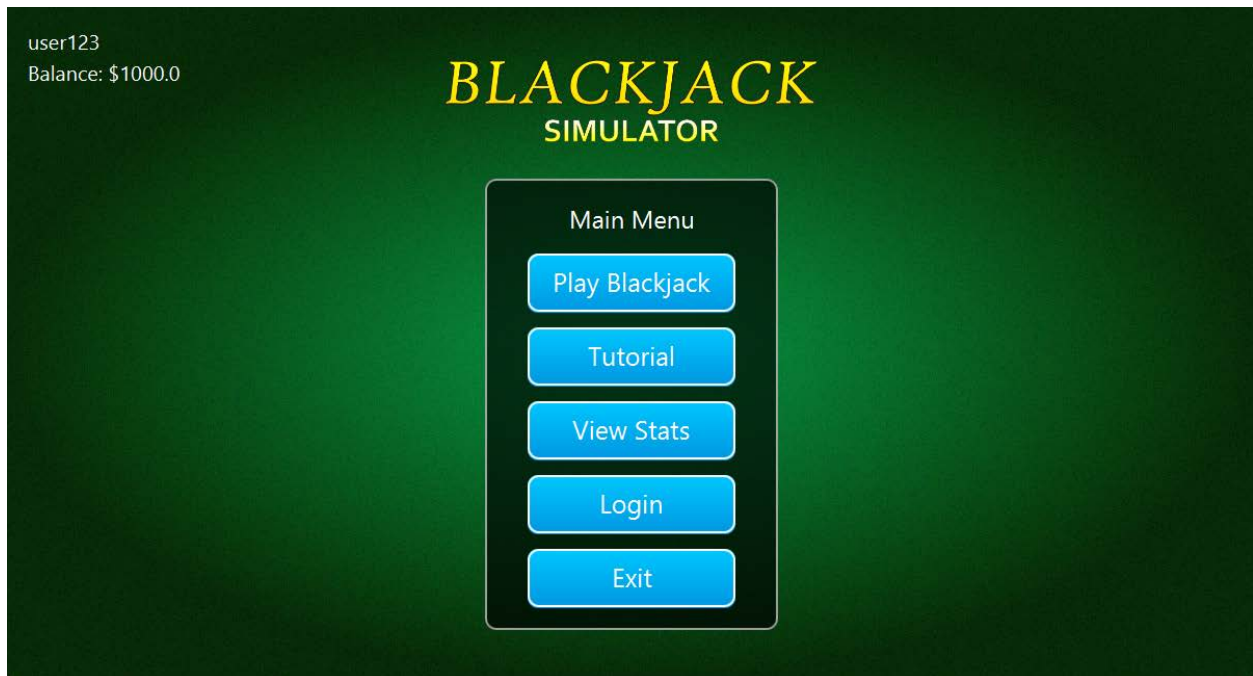


Image 4: Main Menu after successful login with bank balance in upper left corner.

### I.6.2 Betting

Once the user selects “**Play Blackjack**” from the main menu, they will be directed to the betting screen. At this point, the user can use the slider to bet any amount up to the maximum amount.

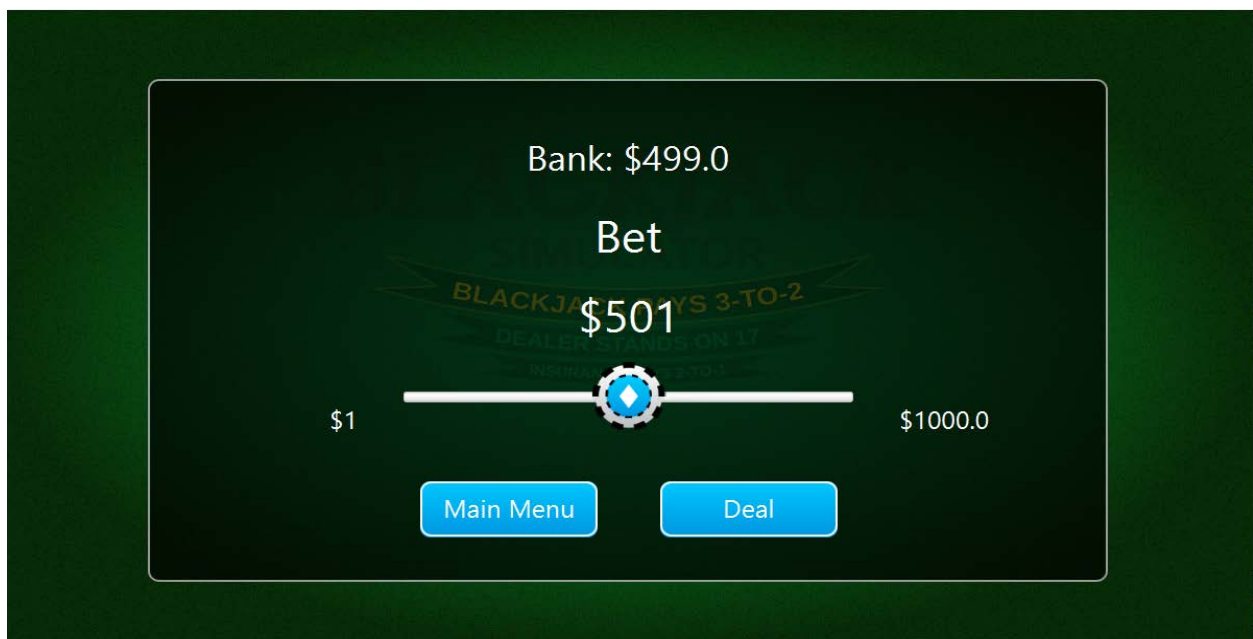


Image 5: Betting Screen



### I.6.3 Cards dealt to the Player and Dealer

The dealer will automatically deal two cards to the player and themselves. The player will be responsible for asking for another card, known as a HIT, or deciding not to take a card, known as a STAND. The dealer, however, will take a HIT until they have at least enough cards valued at 17.

#### I.6.3.1 What is a HIT?

HIT is a term used to tell the dealer you want another card. Once you select HIT, the dealer will give you another card, and your new total card value will be displayed on the screen.



Image 6: Hit button for another card 1st button to the left.

#### I.6.3.2 What is a Stand?



A STAND refers to a position the player takes when they decide they do not want another card. At this point, the dealer may take a HIT (the dealer must take a HIT if they are below 17).



Image 6: Stand button 2nd button on the right.

### I.6.3.3 What is a Double?

You can Double your bet by selecting the DOUBLE BUTTON once the dealer hands out the initial cards and before hitting or standing. Once you double, you will automatically take another card and stand. When you double your bet, you must have enough money in the bank that is equal to the original bet.

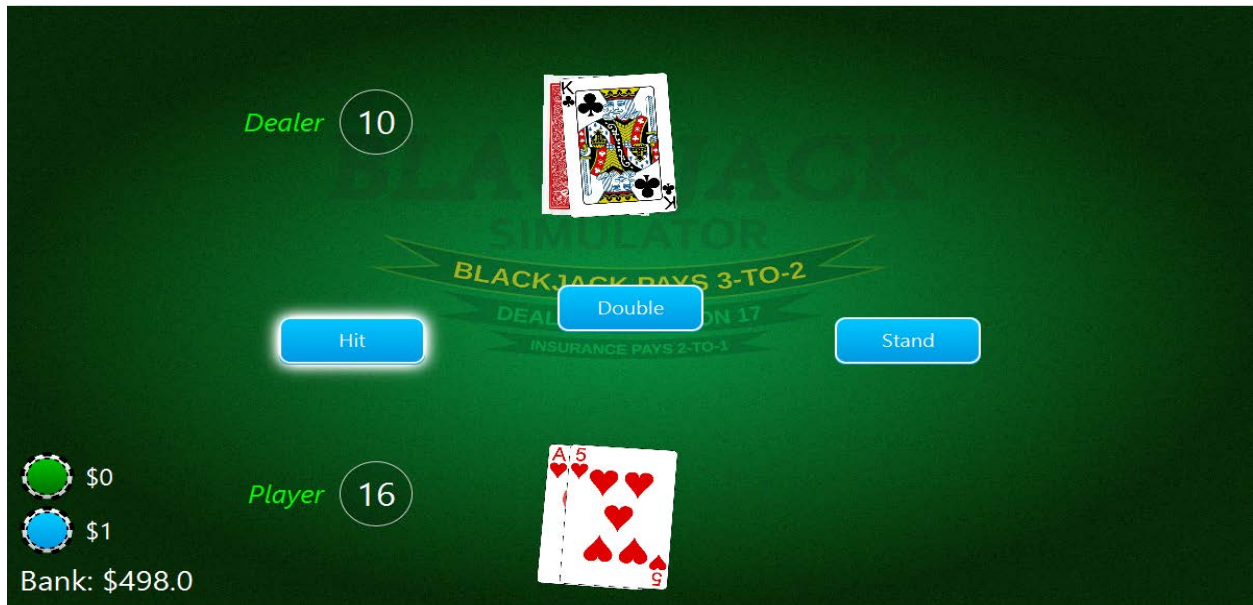


Image 6: Double button 2nd button from the left.

#### I.6.3.4 What is a Split?

A split occurs when you have two cards of the same value. You can split those cards and play additional hands at once. You will play the first hand, take a stand, and then play the remaining hands. Once you take a stand on the other hand, both hands will be matched against the dealer's hand as separate hands. You can win both hands, lose one hand and win the next, or lose both hands. This is risky but can pay off big.



Image 7: Split button 1st button from the left.

#### I.6.3.5 What is insurance?

If the dealer has an Ace face up, this gives you the opportunity to protect yourself in the event the dealer has a natural blackjack (natural blackjack is a 21 on the first two cards). If you select insurance, you will put up half the amount you bet. For example, if you bet \$500 on this hand, you will put up \$250 in insurance. If the dealer does not have a natural blackjack, you lose the insurance money. If the dealer has a natural blackjack, you get your money back and do not lose any money.



Image 8: Insurance button 3rd button from the left.

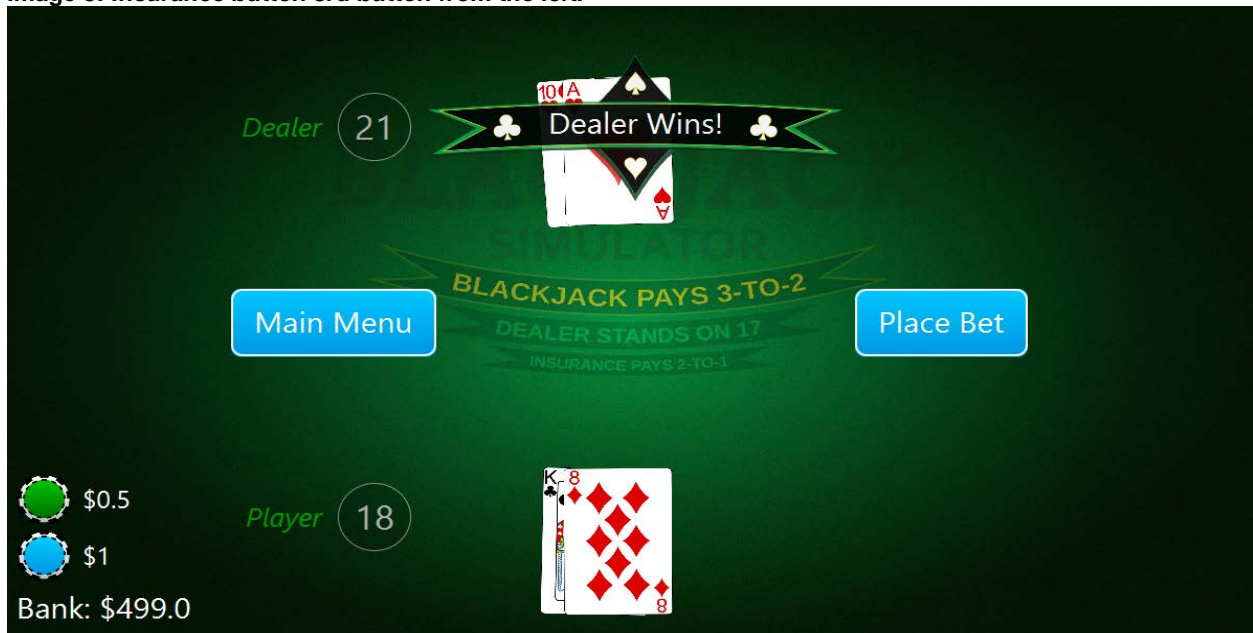


Image 8: Dealer wins player bank goes back to original amount

### I.6.3.6 What is a Push?

Push refers to a situation when both the dealer and the player have the same card value. When this happens, neither the dealer nor the player wins, and as a result, the player does not win or lose any money.

### **I.6.4 How do you Win or Lose?**

There are a few conditions to win or lose the game.

#### **I.6.4.1 How does a Player Win?**

1. The Player has a hand of 21, and the Dealer does not.
2. The Player has a higher hand than the Dealer.
3. The Dealer Bust and the Player does not.

#### **I.6.4.2 How does a Player Lose?**

1. The Player Bust if their hand is greater than 21.
2. The Dealer has a higher hand than the Player.
3. The Dealer has a hand of 21, and the Player does not.

#### **I.6.4.3 Win and Loss amounts.**

1. If a player wins the hand, they receive double their original bet. For example, if you bet \$500.00 and win, you will receive your \$500.00 bet back, plus an additional \$500.00 for a total of \$1000.00, so your bank account will increase by \$500.00.
  - a.  $\$500.00 \text{ bet} + \$500.00 \text{ won} = \$1000.00$
2. If a player gets a natural blackjack (which is when the player's first two cards equal 21), they will earn 1.5 times what they bet.
  - a.  $\$500.00 \text{ bet} + 750.00 \text{ won} = \$1250.00.$
3. If the player doubles their bet, they will place an additional amount equal to their original bet. In other words, if the player's original bet is \$500.00, they will bet an additional \$500.00 when they double, for a total of \$1000.00. If the player wins the hand, they will have \$2000 in the bank.



## I.8.0 Tutorial

In-game instructions can be seen by selecting the “**Tutorial**” option in the “**Main Menu**”. This gives the player information on how the game is played and the rules of the game.

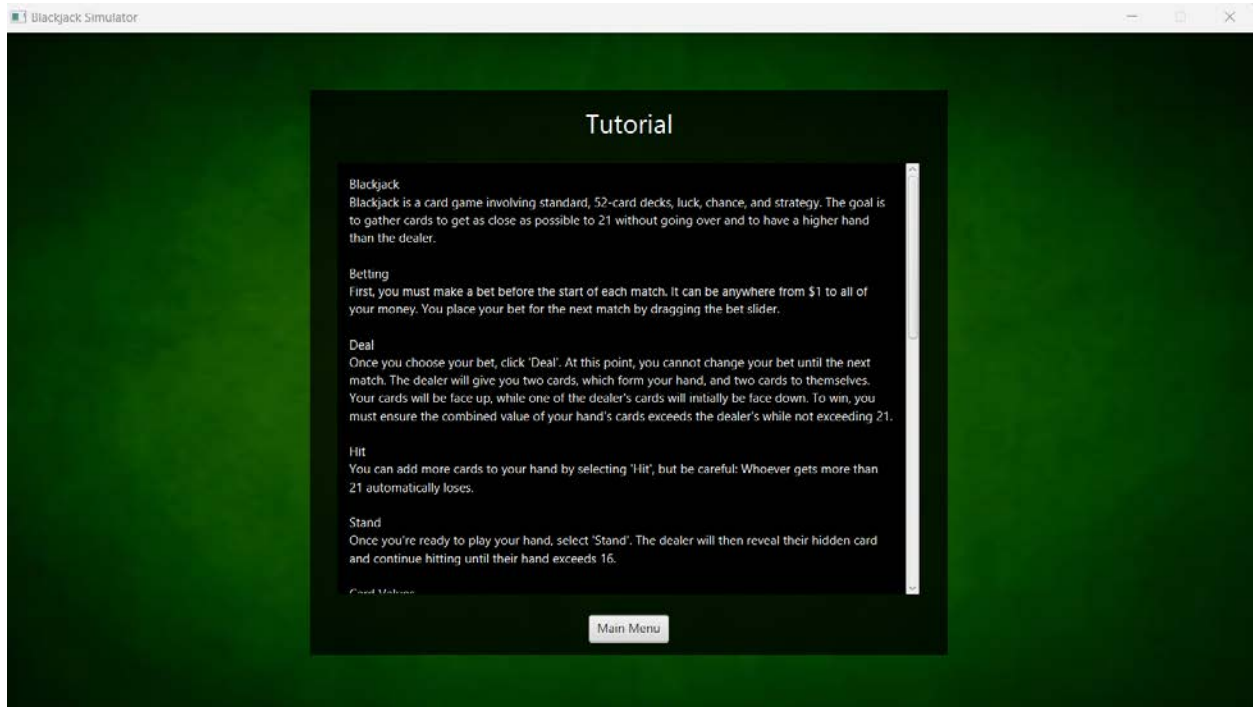


Image 10: *Player Stats*

## I.9.0 Additional Help

For more tips on blackjack, you can search online for resources that can give you more sophisticated techniques. The additional techniques applied to the Blackjack Simulator will give you additional insight into how to navigate different strategies.



## Appendix J - Test Case Table

Test #	Test Description	Input	Expected Output
1	The application displays a start screen and allows the user to enter the game.	The user starts the application.	After the application finishes loading, it displays a start screen. The user can then select a GUI element to enter the game.
2	The user can create an account by entering their username and password.	<p>On the screen for setting up an account, the user enters:</p> <ol style="list-style-type: none"> <li>1) A unique username (the username must not be in the database)</li> <li>2) Any password</li> <li>3) The same password from Step 2</li> </ol> <p>After they enter these fields, they select the 'Submit' button.</p>	After the user submits their data, the application creates a new record within the database containing the username and password the user provided. The application should also add an initial amount of money to the user's bank account.
3	Allow the user to provide their login credentials and access their account.	<p>On the login screen, the user enters:</p> <ol style="list-style-type: none"> <li>1) Their username</li> <li>2) Their password</li> </ol> <p>After they enter these fields, they select the 'Submit' button.</p>	After the user submits their account credentials, the application retrieves them from the database and displays the main menu for the application.
4	Ensures user's data is secure and cannot be stolen.	<p>On the screen for setting up an account, the user enters:</p> <ol style="list-style-type: none"> <li>1) A unique username (the username must not be in the database)</li> <li>2) Any password</li> <li>3) The same password from Step 2</li> </ol> <p>After they enter these fields, they select the</p>	After the user submits their data, the application creates a new record within the database containing the username and password the user provided. The application should encrypt the user's password so that viewers cannot reenter the user's information.



Test #	Test Description	Input	Expected Output
		'Submit' button.	
5	When the user earns or loses money, the application saves their money value into their account data.	The user earns or loses money during a match with the dealer.	The application should access the user's account and increment their wins or losses by the amount they won or lost, respectively.
6	The user can skip the process of creating an account and their money/progress will not be saved.	The user earns or loses money during a match with the dealer.	The application should not attempt to access any user account to update its wins or losses.
7	The application displays the screen where the user places a bet.	In the main menu, the user selects the option to play blackjack.	Before starting the match against the dealer, the application should display the screen where the user can place their bet.
8	The user can place a bet value.	On the screen where the user places their bet, they select a bet amount, then press the 'Start Match' button.	Once the user selects the button to start the match, the application should store the user's bet amount and display its value during the match.
9	The application uses images to represent the blackjack board and cards.	On the screen where the user places their bet, they select a bet amount, then press the 'Start Match' button.	During the blackjack match, the application should display a GUI representing the board and cards.
10	During gameplay, the GUI displays the total value of the player and dealer's hands.	The user enters the blackjack match, and then the application disperses the two cards to the user and dealer.	After the application disperses the cards to the user and dealer, the application should calculate the total hand value of their cards and

Test #	Test Description	Input	Expected Output
			display them in the GUI.
11	The application contains representations of all cards in a deck of 52 cards.	On the screen where the user places their bet, they select a bet amount, then press the 'Start Match' button.	During the blackjack match, the application should display one image for each card in a standard deck of 52 cards. Each image should also increment the user or dealer's hand consistently.
12	The game includes a representation of a dealer.	On the screen where the user places their bet, they select a bet amount, then press the 'Start Match' button.	During the blackjack match, the application should display some representation of where the dealer is and where the user is.
13	When the user stands, the dealer's second card is revealed.	During the blackjack match, the user selects the option to stand.	After the user stands, the dealer should reveal their second card.
14	The dealer hits if the player stands.	During the blackjack match, the user selects the option to stand.	After the user stands, the dealer should reveal their second card and hit or stand based on their hand's total value.
15	The dealer continues hitting until their deck value is 17 or more.	During the blackjack match, the user selects the option to stand.	After the user stands, the dealer should reveal their second card and continuously hit while their hand is less than 17.
16	When the user hits, a card is removed from the combined deck of cards and given to the user.	During the blackjack match, the user selects the option to hit.	After the user hits, the application should remove one random card from the combined deck of cards, display the card in the user's hand, and adjust their hand's total based on the card's

Test #	Test Description	Input	Expected Output
			value.
17	The user can double their bet after receiving their first two cards.	During the blackjack match, the user selects the option to double their bet.	After doubling their bet, the application should set the user's bet value to itself multiplied by two. Then, the application should immediately process a hit followed by a stand for the user.
18	The user can perform a split if they receive two cards of the same value.	During the blackjack match, the user has two cards of the same value and selects the option to split.	After choosing the option to split, the application should split the user's cards into two hands with equal bets, and each hand should receive an additional card.
19	If the dealer's face-up card is an ace, the user can purchase insurance.	During the blackjack match, the dealer receives an ace, and the user selects the option to take insurance.	After the user accepts insurance, the application adds half of their bet to their original amount. If the dealer has blackjack, the insurance is paid 2 to 1, the user's original bet is lost, and the user gains or loses no money, so the application does not adjust their account. Otherwise, the user loses their insurance.
20	If the user has blackjack and the dealer has an ace showing, the dealer offers the user even	During the blackjack match: 1) the user receives a natural blackjack, 2) the dealer's face-up card is an ace, 3) the user selects the	After the user selects the option for even money, if the dealer also has blackjack, the application should pay the user their

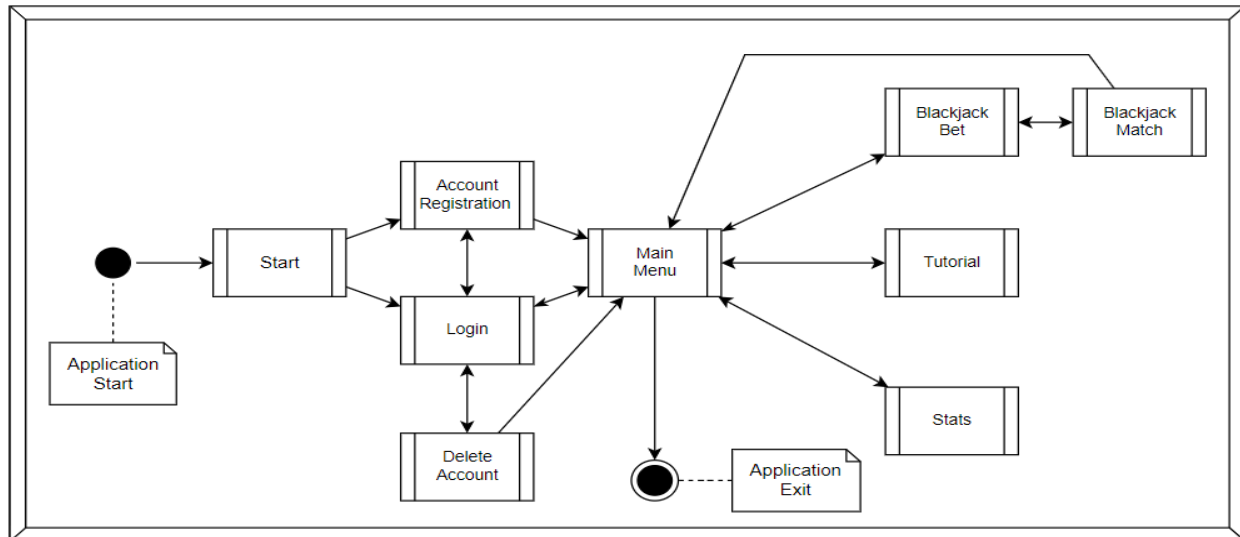
Test #	Test Description	Input	Expected Output
	money for their blackjack (instead of the 1.5x payout).	option to take even money.	<p>bet amount and increase their bank account balance by the amount.</p> <p>If the user declines it and the dealer does not have blackjack, the user will receive the 1.5x payout for their bet, and the application should increase their bank account balance by the amount.</p> <p>If the user declines it and the dealer also has blackjack, the user will have a push just like normal, so the application will not adjust the user's bank account.</p>
21	The user wins if their hand's total exceeds the dealer's.	During the blackjack match: 1) the user's hand totals to 18 or higher, 2) the user stands, 3) the dealer continues to hit until their hand totals to 17.	Since the user's hand is greater than the dealer's, the application should reward the user their bet amount and increase their bank account balance by the amount.
22	The user wins if the dealer busts.	During the blackjack match: 1) the user's hand totals to 18 or higher, 2) the user stands, 3) the dealer continues to hit, and then their hand exceeds 21.	Since the dealer's hand is greater than 21, the application should reward the user with their bet amount and increase their bank account balance by the amount.
23	The user loses if	During the blackjack match,	The user's hand will

Test #	Test Description	Input	Expected Output
	their hand's total exceeds 21.	the user continuously selects the option to hit.	eventually exceed 21, and the application should remove the user's bet amount from their bank account.
24	The user loses if their hand's total is less than the dealer's.	During the blackjack match, the user stands, and the dealer continues to hit until their hand is greater than the user's and less than 22.	Since the dealer's hand is greater than the user's, the application should remove the user's bet amount from their bank account balance.
25	The user receives their bet back if they and the dealer both have a hand total of 21.	During the blackjack match, the user stands when their hand is between 17 and 20, and the dealer continues to hit until their hand's total matches the user's.	Since the user and dealer's hands are equal, the application should not adjust the user's bank account balance.
26	When the user loses all of their money, or closes the application, a game over screen is shown.	During the blackjack match, the user bets all their money and loses to the dealer.	After the user loses all their money, the application should inform them they lost all their money and set their bank account balance to zero.
27	During gameplay, the GUI indicates whether the user should hit or stand.	The user enters the blackjack match, and then the application disperses the two cards to the user and dealer.	After the application disperses the cards to the user and dealer, the application should calculate the best course of action for the user and indicate which choice the user should take by highlighting GUI elements.

## Appendix K - Final Test and Results

### Final Tests and Results

The report below contains all the applicable tests for each phase, specific tests that trace back to the client's original requirements, and acceptance tests. Additionally, a diagram representing the connection between the interfaces is provided.



The diagram shows GUI scene interactions.

### Phase III Test

Test #	Input	Expected Output	Actual Output	Test Status
Startup #1	Select application from desktop	Startup screen, with options to enter the program.	Startup screen, with options to enter the program.	Pass
Betting Scene #1	Bet \$500	Change in Bet amount to \$500 and Change to Bank amount to \$500	The Bet Changed to \$500 and the Bank Changed to \$500	Pass
Betting Scene #2	Bet amount and lose.	The bank subtracts money from the bank	The bank subtracts money from the bank amount by	Pass

Test #	Input	Expected Output	Actual Output	Test Status
		amount by money loss in the Betting Scene.	money loss in the Betting Scene.	
Betting Scene #3	Bet amount and win.	Double the amounts and add to the principal amount in the bank account.	Double the amounts and add to the principal amount in the bank account.	Pass
Betting Scene #4	Select Main Menu	Scene switch to the Main menu.	Scene switch to the Main menu.	Pass
Betting Scene #5	Select the Betting scene after returning from the Main Menu	The Bank account has updated the balance after returning to the scene from the Main Menu.	The Bank account has updated the balance after returning to the scene from the Main Menu.	Pass
Blackjack Scene #1	Double button selected.	The amount bet doubles.	Amount doubled in bet amount.	Pass
Blackjack Scene #2	Hit button selected.	The dealer gives the player another card.	The dealer gives the player another card.	Pass
Blackjack Scene #3	Stand Button selected.	The dealer deals self-additional cards and shows.	The dealer deals self-additional cards and shows.	Pass
Blackjack Scene #3	The player takes a stand and the dealer has the same hand as the player.	The screen displays Push.	The screen displays Push.	Pass
Blackjack Scene #4	The player selects Stand and the dealer's hand is higher without a bust.	The screen displays Dealer Wins on the screen.	The screen displays Dealer Wins on the screen.	Pass
Blackjack Scene #5	The player selects stand and the player has a higher hand.	Scene displays You win.	The screen displays You Win.	Pass

Test #	Input	Expected Output	Actual Output	Test Status
Blackjack Scene #6	The player is higher than 21 and dealer is higher than 21.	Scene displays Dealer Wins.	The Scene displays Dealer wins.	Pass
Blackjack Scene #7	Select Main Menu after the hand is completed.	Scene switch to the main menu.	Scene switch to the main menu.	Pass
Blackjack Scene #8	Dealer deals player natural blackjack	Scene displays player wins, player is awarded 1.5 times the amount bet.	Scene displays player wins, player is awarded 1.5 times the amount bet.	Pass
Blackjack Scene #9	The dealer's face card is an Ace.	The scene displays the insurance button for the player.	The scene displays the insurance button for the player.	Pass
Blackjack Scene #10	The dealer does not have black jack when insurance is used.	Player loses insurance money.	Player loses insurance money.	Pass
Blackjack Scene #11	Player dealt two cards with the same face value.	The scene displays a split button.	The scene displays a split button.	Pass
BlackJack Scene #12	The Player selects the split button	The scene splits the player's hand and hits the original hand.	The scene splits the player's hand and hits the original hand	Pass
Blackjack Scene #13	The Player selects the split button	The scene splits hands and displays new card values.	The scene splits hands and displays new card values.	Pass
Blackjack Scene #14	The player does not have money equal to the bet amount in the account when two cards are equal on initial deal.	The scene does not display the split button.	The scene does not display the split button.	Pass
Blackjack Scene #15	The dealer card values are below 17.	The dealer hits until hand exceeds 17.	The dealer hits until hand exceeds 17.	Pass



Test #	Input	Expected Output	Actual Output	Test Status
Blackjack Scene #16	Dealer bust and player is below 21	Screen displays you win.	Screen Displays you win.	Pass
Blackjack Scene #17	Player loses all money.	A message telling player they lost or equivalent.	A message telling player they lost or equivalent.	Pass
Blackjack Scene #18	Display telling the user should hit or stand.	Button light up telling user weather they should hit or stand.	Button light up telling user weather they should hit or stand.	Pass
Blackjack Scene #19	Total value of hand shown.	Total value of hand shown for dealer and player.	Total value of hand shown for dealer and player.	Pass
Tutorial #1	Select Tutorial from Main Menu.	Scene switch to Tutorial screen.	Scene switch to Tutorial screen.	Pass
Tutorial #2	Scroll up and down the tutorial to read text.	Text box scrolls up or down.	Test box scrolls up or down.	Pass
Tutorial #3	Select Main Menu	Scene switch to Main Menu Scene	Scene Switch to Main Menu Scene.	Pass
View Stats #1	Select View Stats from the Main Menu Button.	Scene with to View Stats Scene.	Scene with to View Stats Scene.	Pass
View Stats #2	Select Main Menu Button.	Scene switch to Main Menu Scene.	Scene switch to Main Menu Scene.	Pass
View Stats #3	Player loses all money.	Game resets player balance to \$1000.00	Game resets player balance to \$1000.00	Pass
View Stats #4	Player Bank Balance is below other players,	Player losses current ranking.	Player losses current ranking.	Pass
Login #1	Select Login from the Main Menu Screen.	Scene switch to Login Scene.	Scene switch to Login Scene.	Pass

Test #	Input	Expected Output	Actual Output	Test Status
Login #2	Select Play as a guest button.	Scene switches directly to Main Menu.	Scene switches directly to Main Menu.	Pass
Login #3	Select Create Account Button	Scene switches directly to the Registration page	Scene switches directly to the Registration page.	Pass
Login #4	Select Delete Account Button	Scene switches directly to the delete account page.	Scene switches directly to the delete account page.	Pass
Login #5	Login with user name and password.	Scene switches to main menu.	Scene Switches to main menu.	Pass
Delete Account #1	Select Submit Button.	Scene switches directly to Main Menu.	Scene switches directly to Main Menu.	Pass
Delete Account #2	Type correct username and password Type "delete"	Account removed from database	Account removed from database	Pass
Delete Account #3	Type incorrect username and correct password. Type "delete"	Screen displays incorrect username or password.	Screen displays incorrect username or password.	Pass
Delete Account #4	Type correct username and incorrect password. Type "delete"	Screen displays incorrect username or password.	Screen displays incorrect username or password	Pass
Delete Account #5	Type correct username and correct password. Type " "	Screen displays "Please enter delete"	Screen displays "Please enter delete"	Pass
Create Account #1	Enter username Enter Password Confirm Password	Scene directs to the main menu.	Scene directs to the main menu/	Pass

Test #	Input	Expected Output	Actual Output	Test Status
Create Account #2	Enter username Enter mis-matching passwords	Scene displays passwords do not match	Scene displays passwords do not match	Pass
Create Account #3	Enter username already in database	Screen displays username is currently in use.	Screen displays username is currently in use.	Pass
Play Guest #1	Select Play as Guest.	Scene directs to the Main Menu.	Scene directs to the Main Menu.	Pass
Play Guest #2	Select Play as Guest.	Scene displays Logged in as Guest with \$1000.00	Scene displays Logged in as Guest with \$1000.00	Pass
Play Guest #3	Player bets all funds.	Scene routes player to main menu.	Scene routes player to main menu.	Pass
Database #1	Users Table username and password (Entered in Create account)	Username in table with Hashed password.	Username in table with Hashed password.	Pass
Database #2	Wins amounts won. from blackjack scene.	Win Table updates with win amount.	Win Table updates with win amount.	Pass
Database #3	Loss amounts from the blackjack scene.	Loss Table updates with Loss amount.	Loss Table updates with Loss amount.	Pass
Database #4	Available balance from blackjack scene.	Bank balance passed to the Banks Table upon exit.	Bank balance passed to the Banks Table upon exit.	Pass
Database #5	Delete the user in from the delete scene.	Users are removed from all tables.	Users are removed from all tables.	Pass
Database #6	Query users from the Login screen.	Username and Password not correct	Incorrect username or password.	Pass

Test #	Input	Expected Output	Actual Output	Test Status
Database #7	Drop user tables from the Login Screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #8	Drop user tables from the delete screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #9	Query users from the delete screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #10	Drop Banks tables from the login screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #11	Drop Banks tables from the delete screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #12	Drop Losses tables from the delete screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #13	Drop Losses tables from the login screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #14	Drop Wins tables from the login screen.	Incorrect username or password.	Incorrect username or password.	Pass
Database #15	Drop Wins tables from the delete screen.	Incorrect username or password.	Incorrect username or password.	Pass

### Acceptance TEST

Test #	Function	Input	Expected Output	Actual Output	Test Status
Acceptance Test #1	User Signup	Username: Noah Password: 12345	User is directed to Main Menu	User is directed to Main Menu	Pass

Test #	Function	Input	Expected Output	Actual Output	Test Status
Acceptance Test #2	Login	Username: Noah Password: 12345	User is directed to Main Menu	User is directed to Main Menu	Pass
Acceptance Test #3	Delete Account	Username: Noah Password: 12345	User is directed to delete screen	User is directed to delete screen	Pass
Acceptance Test #4	Tutorial Screen	Select Tutorial from the main menu.	Tutorial is displayed.	Tutorial is displayed.	Pass
Acceptance Test #5	Betting	Move slider to \$500.00	Amount bet is displayed.	Amount bet is displayed.	Pass
Acceptance Test #6	Stand Button	User selects Stand/	Dealer shows his hand.	Dealer shows his hand.	Pass
Acceptance Test #7	HIT	Users select HIT.	Dealer gives an additional card.	Dealer gives an additional card.	Pass
Acceptance Test #8	Double Button	User selects the Double Button.	User bets double and takes an additional card and then stands.	User bets double and takes an additional card and then stands.	Pass
Acceptance Test #9	Split Button	User selects a split button.	User plays additional hands.	User plays additional hands	Pass
Acceptance Test #10	Insurance Button	User selects the insurance button.	User bets half the bet amount.	User bets half the bet amount.	Pass
Acceptance Test #11	Push Condition	User and dealer has same hand.	Push condition is displayed.	Push condition is displayed.	Pass
Acceptance Test #12	Game Over	User runs out of money.	Game over user account reset to \$1000.00	Game over user account	Pass

Test #	Function	Input	Expected Output	Actual Output	Test Status
				reset to \$1000.00	
Acceptance Test #13	Exit Game	User selects Exit from the main menu.	Application closes.	Application closes.	Pass
Acceptance Test #14	View Stats	User selects view stats from the main menu.	Stats board is displayed.	Stats board is displayed.	Pass
Acceptance Test #15	Play as Guest	User selects Guest.	User is directed to the main menu.	User is directed to the main menu.	Pass

### Requirements Matrix

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
R001	The application displays a start screen and allows the user to enter the game.	T001	starts initialize(): void and allows user to access either the registration functions or the login functions	Completed	Pass
R002	The user can create an account by entering their username and password. Their account information is then saved into the	T002	Accesses database manager from start screen and run functions DatabaseManger(),	Completed	Pass

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
	database.		insertUser(String userName, String password), and getUser(String userName)		
R003	Allow the user to provide their login credentials and access their account.	T003	Pulls from DatabaseManager() and runs functions getUser(String userName): ResultSet, userExist(String userName): boolean	Completed	Pass
R004	Ensures user's data is secure and cannot be stolen.	T004	Pulls from PasswordEncryptDecrypt and runs the functions encrypt(String strToEncrypt): String, encrypt(String strToEncrypt, String secretKey, String salt): String, decrypt(String strToDecrypt): String, decrypt(String strToDecrypt):	Completed	Pass

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			String, and decrypt(String strToDecrypt, String secretKey, String salt): String		
R005	When the user earns or loses money, save their money value into their account data.	T005	Pulls from the DatabaseManager() and runs insertLoss/Win ForUser(String userName, int amount)	Completed	Pass
R006	The user can skip the process for creating an account. Their money/progress will not be saved.	T006	Clicks start instead of logging in where there data wont be saved but it will take them to BlackjackMatch java file	Completed	Pass
R007	The application displays the screen where the user places a bet.	T007	From SceneController the Blackjack match is started.	Completed	Pass
R008	The user can place a bet value.	T008	From the BlackjackMatch initializes variables cardDeck, playerHands, dealerHand,	Completed	Pass



Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			playerBet, dealerHandValueLabel, playerHandValueLabels		
R009	The application uses images to represent the Blackjack board and cards.	T009	Uses image files stored within the program that are pulled to be used in conjunction with the game also uses getImagePath(): String with Card java file	Completed	Pass
R010	The application displays a screen for the Blackjack game.	T010	Pulls from the BlackjackMatchGUI() function	Completed	Pass
R011	The application contains representations of all cards in a deck of 52 cards.	T011	Uses array cardDeck: Deck that contains all types of the 52 card deck	Completed	Pass
R012	The game includes a representation of a Dealer.	T012	Pulls from BlackjackMatch java file and uses dealerHand: Hand and dealerHandValueLabel: Label	Completed	Pass
R013	The dealer hits if the	T013	Pulls from	Completed	Pass

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
	player stands.		BlackjackMatch java file and uses onStandButton Pressed() void		
R014	The dealer continues hitting until their deck value exceeds 17.	T014	Limit set to dealerHand: Hand with a max value of 17	Completed	Pass
R015	When the user hits, a card is removed from the combined deck of cards and given to the user.	T015	Pulls out of cardDeck's total of 52 cards and disseminates them to the dealer and user from getCard(): Card from Deck java file	Completed	Pass
R016	When the user stands, the Dealer's second card is revealed.	T016	Within BlackjackMatch if user uses onStandButton Pressed(): void function dealer reveals second card	Completed	Pass
R017	The user can double their bet after receiving their first two cards. This will cause the user to Hit and an immediate Stand.	T017	When user clicks the button onBetDoubleButtonPressed(): void function is ran and	Completed	Pass

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			takes their bet and doubles it		
R018	<p>The user can perform a split if they receive two cards of the same value.</p> <p>This splits the user's cards into two separate hands with equal bets, and each hand receives an additional card. Each hand is played separately.</p>	T018	User executes onSplitButtonPressed(): void where they can now use two separate hands for the Blackjack game	Completed	Pass
R019	<p>If the dealer's face up card is an ace, the user can purchase insurance.</p> <p>As a result, the user bets half their original bet (in addition to it) that the dealer does have Blackjack.</p> <p>If the dealer does, the insurance is paid 2 to 1 and the user's original bet is lost (so they break even for the hand).</p> <p>Otherwise, the user loses their insurance.</p>	T019	User executes onInsuranceButtonPressed(): void within the BlackjackMatch.java file	Completed	Pass
R020	If the user has Blackjack and the dealer has an ace showing, the dealer offers the user even	T020	User executes onEvenMoneyButtonPressed(): void within the	Completed	Pass

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
	money for their Blackjack (instead of 3 to 2). If the user declines it and the dealer also has Blackjack, the user will have a push just like normal.		BlackjackMatch java file		
R021	The user wins if their hand's total is higher than the dealer's.	T021	User wins the hand and is their money is directed to WinTableCals with variables User_ID and WinAmount	Completed	Pass
R022	The user wins if the dealer busts.	T022	User wins the hand and is their money is directed to WinTableCals with variables User_ID and WinAmount	Completed	Pass
R023	The user loses if their hand's total exceeds 21.	T023	User loses the hand and is their money is directed to LossesTableCals with variables User_ID and LossAmount	Completed	Pass
R024	The user loses if their hand's total is less than the dealer's.	T024	User loses the hand and is their money is	Completed	Pass

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			directed to LossesTableControls with variables User_ID and LossAmount		
R025	The user receives their bet back if they and the dealer both have a hand total of 21.	T025		Completed	Pass
R026	When the user loses all of their money, or closes the application, a game over screen is shown.	T026	Game over screen is displayed once user loses after losing all their money or clicking closeGameButton() function.	Completed	Pass
R027	During gameplay, the GUI indicates whether the user should hit or stand.	T027	Uses BestChoice Java file that recommends to user what they should do within the game	Completed	Pass
R028	During gameplay, the GUI displays the total value of the player and dealer's hands.	T028	Pulls from dealerHandValueLabels: Label and playerHandValueLabels: Label to display total	Completed	Pass

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			values		
R029	Menu UI elements generate sound feedback when selected.	T029	Looks at SceneController and whatever menu user selects pulls from audio files stored within the program to play the sound	In Progress	
R030	Chip UI elements generate sound feedback when selected.	T030	Pulls from audio files for chips when user interacts with them or are moving	In Progress	
R031	Card UI elements generate sound feedback when moving.	T031	Pulls from audio files for cards when user interacts with them or are moving	In Progress	
R032	UI sound effects (GUI)	T032	Menu UI elements generate sound feedback when selected.	In Progress	
R033	UI sound effects (Chips)	T033	Chip UI elements generate sound feedback when selected.	In Progress	

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
R034	UI sound effects (Cards)	T034	Card UI elements generate sound feedback when moving.	In Progress	
R035	Odds Calculator	T035	Utilizes the getOddsCalculation function that calculates the best choice for the user.	Completed	Passed
R036	Password Handler	T036	Uses encrypt function to encrypt key and allow secure user passwords runs decrypt function when user is trying to log in.	Completed	Passed
R037	Launcher	T037	Provides a launcher for JavaFX application, must call class's main method that starts application.	Completed	Passed
R038	Null Player Exception	T038	Throws runtime exceptions in the event a process	Completed	Passed

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			attempts to get the instance before its set.		
R039	Registration Scene	T039	Allows user to register account and checks if user's credentials are valid after registering if they meet the requirements through credentialsValid function.	Completed	Passed
R040	Routine which allows methods for handling time events.	T040	Specifies when certain events are activated when being called from other functions through doAfter(), doRepeatOnCondition, Action, and Conditional.	Completed	Passed
R041	Tutorial	T041	Opens a TutorialScene that teaches the user how to play blackjack	Completed	Passed
R042	User Statistics	T042	Displays user stats on a java	Completed	Passed



Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			scene showing rank, name, bank balance, winnings, and losses. Displays these specific to the user that has logged in based off database.		
R043	Player class	T043	Starts player out with 1000 dollars in their balance to gamble with and initialize all of the users information.	Completed	Passed
R044	Login Scene	T044	onLoginButton Pressed function will check user credentials to ensure that their data is displayed to them appropriately.	Completed	Passed
R045	Scene Controller	T045	Scene Manager that controls what scene is opened depending on what user	Completed	Passed

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			selects START, LOGIN, REGISTRATION, MAIN_MENU, TUTORIAL, BETTING, BLACKJACK, STATS, or DELETE.		
R046	Images	T046	Where all images for cards are stored for functions that pull these images to display to user within the application	Completed	Passed
R047	CSS files	T047	Uses in conjunction with Java files to enable the bet slider, stats table, and styles for betting and blackjack to be displayed to the user.	Completed	Passed
R048	FXML files	T048	XML based user interface markup language used	Completed	Passed

Requirement ID	Requirement Description	Test Case ID	Test Description	Test Design Status	Test Status
			with JavaFX to display the BettingScene, BlackjackScene, Card, DeleteScene, Hand, LoginScene, MainMenuScene, RegistrationScene, StartScene, and Tutorial scene when called from the program.		
R049	Deck	T049	Creates two arrays cardValues and cardSuits that assigns these values that are used for the blackjack game.	Completed	Passed
R050	Game Object	T050	Pulls the FXML files to be displayed within the java file that allows user to see the objects displayed.	Completed	Passed

### Traceability Matrix with Test Cases

Requirement ID	Feature	Description	Test Case ID	Test Result
R001	Start GUI	The application displays a start screen and allows the user to enter the game	Startup #1	Pass
R002	User account creation and storage	The user can create an account by entering their username and password. Their account information is then saved into the database.	Create Account#1	Pass
			Database#1	Pass
R003	User account login	Allow the user to provide their login credentials and access their account.	Login#5	Pass
R004	Encrypted User Data	Ensures user's data is secure and cannot be stolen.	Database#1	Pass
R005	Saving user money	When the user earns or loses money, save their money value into their account data.	Database#2	Pass
R006	User guest login	The user can skip the process for creating an account. Their money/progress will not be saved.	PlayGuest#1	Pass
R007	Betting GUI	The application displays the screen where the user places a bet.	BettingScene #1	Pass
R008	Bet placement	The user can place a bet value.	BettingScene #1	Pass
R009	Game board imagery	The application uses images to represent the Blackjack board and cards.	Blackjack Scene#1	Pass
R010	Game board GUI	The application displays a screen for the Blackjack game.	Blackjack Scene#1	Pass
R011	Cards	The application contains representations of all cards in a deck of 52 cards.	Blackjack Scene#1	Pass

Requirement ID	Feature	Description	Test Case ID	Test Result
R012	Dealer	The game includes a representation of a Dealer.	Blackjack Scene#1	Pass
R013	Dealer hit on stand	The dealer hits if the player stands.	Blackjack Scene#3	Pass
R014	Dealer hit below 17	The dealer continues hitting until their deck value exceeds 17.	Blackjack Scene#15	Pass
R015	Hitting	When the user hits, a card is removed from the combined deck of cards and given to the user.	Blackjack Scene#2	Pass
R016	Standing	When the user stands, the Dealer's second card is revealed.	Blackjack Scene#3	Pass
R017	Bet doubling	The user can double their bet after receiving their first two cards. This will cause the user to Hit and an immediate Stand.	Blackjack Scene#1	Pass
R018	Splitting	The user can perform a split if they receive two cards of the same value. This splits the user's cards into two separate hands with equal bets, and each hand receives an additional card. Each hand is played separately.	BlackJack Scene#12	Pass
R019	Insurance	If the dealer's face up card is an ace, the user can purchase insurance. As a result, the user bets half their original bet (in addition to it) that the dealer does have Blackjack. If the dealer does, the insurance is paid 2 to 1 and the user's original bet is lost (so they break even for the hand). Otherwise, the user loses their	Blackjack Scene#10	Pass

Requirement ID	Feature	Description	Test Case ID	Test Result
		insurance.		
R020	Even Money	If the user has Blackjack and the dealer has an ace showing, the dealer offers the user even money for their Blackjack (instead of 3 to 2). If the user declines it and the dealer also has Blackjack, the user will have a push just like normal.		Pass
R021	Win condition (higher hand)	The user wins if their hand's total is higher than the dealer's.	Blackjack Scene#5	Pass
R022	Win condition (dealer bust)	The user wins if the dealer busts	Blackjack Scene#16	Pass
R023	Lose condition (21+)	The user loses if their hand's total exceeds 21.	Blackjack Scene#6	Pass
R024	Lose condition (lower hand)	The user loses if their hand's total is less than the dealer's.	Blackjack Scene#4	Pass
R025	Push condition	The user receives their bet back if they and the dealer both have a hand total of 21.	Blackjack Scene#3	Pass
R026	Game over GUI	When the user loses all of their money, or closes the application, a game over screen is shown.	Blackjack Scene#17	Pass
R027	Favorable odds	During gameplay, the GUI indicates whether the user should hit or stand.	BlackJack Scene#18	Pass
R028	Total hand values	During gameplay, the GUI displays the total value of the player and dealer's hands.	Blackjack Scene#19	Pass

## Appendix L - Document References

The following table summarizes the documents referenced in this document.

Document Name	Version	Description	Location
Project Plan	1.0.0	Provides the project's purpose, scope, and members and outlines the process for achieving project goals.	<a href="https://docs.google.com/document/d/1odORs2VYeSzzONOs4kjUUVkEeSbkZGnlb3zQBoQYk/edit?usp=sharing">https://docs.google.com/document/d/1odORs2VYeSzzONOs4kjUUVkEeSbkZGnlb3zQBoQYk/edit?usp=sharing</a>
Software Requirements Document	1.0.0	Describes the project's requirements and discusses assumptions and constraints around the project.	<a href="https://docs.google.com/document/d/1A0Qx97-KMaU-qclehVgPfwOo6OEN52UJeuaOkfEWNwM/edit?usp=sharing">https://docs.google.com/document/d/1A0Qx97-KMaU-qclehVgPfwOo6OEN52UJeuaOkfEWNwM/edit?usp=sharing</a>
User Guide	1.0.0	Outlines how users interact with the Blackjack Simulator application.	<a href="https://docs.google.com/document/d/1fu6sEsR55B-ALG5OBPPILaAQZZ2J-QCd/edit">https://docs.google.com/document/d/1fu6sEsR55B-ALG5OBPPILaAQZZ2J-QCd/edit</a>
Requirements Features List	1.0.0	Lists and describes the requirements the project must fulfill.	<a href="https://docs.google.com/spreadsheets/d/1sEX85DoVhcjCtCEIZWCoOsai6x2O_9XQhbM9HWluKCw/edit#gid=0">https://docs.google.com/spreadsheets/d/1sEX85DoVhcjCtCEIZWCoOsai6x2O_9XQhbM9HWluKCw/edit#gid=0</a> <a href="https://docs.google.com/spreadsheets/d/1kL9UFM9xcEq8ZRJMzHJPWiFi9jpnRVC0eQDL3NYsTs8/edit#gid=0">https://docs.google.com/spreadsheets/d/1kL9UFM9xcEq8ZRJMzHJPWiFi9jpnRVC0eQDL3NYsTs8/edit#gid=0</a>

## References

- Admin, S. (2016, December 18). *Software Testing Life Cycle STLC | Software Testing Class*. Software Testing Class. <https://www.softwaretestingclass.com/software-testing-life-cycle-stlc/>
- Blake, M. (2021, July 30). *How to play blackjack*. Blackjack Rules - How to Play Blackjack (Beginners Guide). <https://www.onlinegambling.com/blackjack/rules/>
- Blackjack Apprenticeship. (n.d.). Blackjack strategy charts - how to play Perfect blackjack. Blackjack Apprenticeship. <https://www.blackjackapprenticeship.com/blackjack-strategy-charts/>
- Dickson, C. (2020, July 20). *Designing a Relational Database and Creating an Entity Relationship Diagram*. Medium. <https://towardsdatascience.com/designing-a-relational-database-and-creating-an-entity-relationship-diagram-89c1c19320b2>
- freegames.org. (n.d.). *Play blackjack: 100% free online game*. FreeGames.org. <https://freegames.org/blackjack/#>
- Gluon. (n.d.). JavaFX. <https://openjfx.io/>
- Hamilton, T. (2023, October 14). *Software Testing Metrics: What is, Types & Example*. Guru99. <https://www.guru99.com/software-testing-metrics-complete-tutorial.html>
- Las Vegas Convention and Visitors Authority Research Center. (2023). LVCVA Executive Summary. LVCVA. <https://www.lvcva.com/research/visitor-statistics/>
- Microsoft. (n.d.). *Database design basics*. Microsoft Support. <https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5>



Mijacobs, alexbuckgit, v-thepet, & EdKaim. (n.d.). *What is Kanban? - Azure DevOps*.

Microsoft Learn. <https://learn.microsoft.com/en-us/devops/plan/what-is-kanban#kanban-boards>

Oracle. (2014). *3 Hello World, javafx style*. 3 Hello World, JavaFX Style (Release 8).

[https://docs.oracle.com/javase/8/javafx/get-started-tutorial/hello\\_world.htm](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/hello_world.htm)

Oracle. (2014). *6 using FXML to create a user interface*. 6 Using FXML to Create a User

Interface (Release 8). [https://docs.oracle.com/javase/8/javafx/get-started-tutorial/fxml\\_tutorial.htm](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/fxml_tutorial.htm)

Oracle. (2023). Java JDBC API.

<https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>

Project Management Institute. (2017). *A Guide to the Project Management Body of*

*Knowledge (PMBOK® Guide)—Sixth Edition: Vol. Sixth edition*. Project

Management Institute. [https://eds-s-ebscohost-](https://eds-s-ebscohost-com.ezproxy.umgc.edu/eds/ebookviewer/ebook/bmxlYmtfXzE1OTUzMjBfX0FO0)

[com.ezproxy.umgc.edu/eds/ebookviewer/ebook/bmxlYmtfXzE1OTUzMjBfX0FO0](https://eds-s-ebscohost-com.ezproxy.umgc.edu/eds/ebookviewer/ebook/bmxlYmtfXzE1OTUzMjBfX0FO0?sid=f8216582-e29d-4b67-980a-f59c511f73be@redis&vid=2&format=EB&rid=5)  
[?sid=f8216582-e29d-4b67-980a-f59c511f73be@redis&vid=2&format=EB&rid=5](https://eds-s-ebscohost-com.ezproxy.umgc.edu/eds/ebookviewer/ebook/bmxlYmtfXzE1OTUzMjBfX0FO0?sid=f8216582-e29d-4b67-980a-f59c511f73be@redis&vid=2&format=EB&rid=5)

QuinnRadich, drewbatgit, DCtheGeek, mijacobs, & msatranjr. (2021, April 27). *What Is a Window?*. What Is a Window - Win32 apps | Microsoft Learn.

<https://learn.microsoft.com/en-us/windows/win32/learnwin32/what-is-a-window->

Relax Gaming. (n.d.). *Blackjack (Relax Gaming)*. Play free blackjack (relax gaming)

game. <https://casino.guru/blackjack-relax-gaming-play-free#playGame>

Squarespace. (n.d.). *Website builder - create a website in minutes*. squarespace.com.

<https://www.squarespace.com/>

Velotta, R. N. (2023, September 3). *Gaming industry continues to amaze with record July revenue*. Las Vegas Review-Journal.

<https://www.reviewjournal.com/business/business-columns/inside-gaming/gaming-industry-continues-to-amaze-with-record-july-revenue-2898626/>

WP Company. (n.d.). *BlackJack*. The Washington Post.

<https://games.washingtonpost.com/games/blackjack>

www.linkedin.com. (2023, September 13). *How can you write a product design specification that works?*. How to Write a Product Design Specification.

<https://www.linkedin.com/advice/1/how-can-you-write-product-design-specification-works>

Zuci Systems. (n.d.). *Software Design Specification Template*. A Software Design

Specification Template. [https://www.zucisystems.com/wp-content/uploads/2020/08/Zuci\\_Software-Design-Specification-Template.pdf](https://www.zucisystems.com/wp-content/uploads/2020/08/Zuci_Software-Design-Specification-Template.pdf)

Michali. (2022, February 23). *What is Black Box Testing?* Check Point Software.

[https://www.checkpoint.com/cyber-hub/cyber-security/what-is-penetration-testing/what-is-black-box-](https://www.checkpoint.com/cyber-hub/cyber-security/what-is-penetration-testing/what-is-black-box-testing/#:~:text=Black%20box%20testing%2C%20a%20form,other%20aspects%20of%20an%20application.)

[testing/#:~:text=Black%20box%20testing%2C%20a%20form,other%20aspects%20of%20an%20application.](https://www.checkpoint.com/cyber-hub/cyber-security/what-is-penetration-testing/#:~:text=Black%20box%20testing%2C%20a%20form,other%20aspects%20of%20an%20application.)

Ganguly, S. (2023, February 15). *What is GUI Testing? (Types & Best Practices)*

*BrowserStack*. Browser Stack <https://www.browserstack.com/guide/gui-testing>