Blackjack

Final Report

Git Good

Jonathan Courter, Tyler Yasukochi, Parker Shore

Course: CPSC 224 - Software Development

I. Introduction and Project Description

The goal of this project was to create an interactive and visually engaging application that accurately replicates the core mechanics of the classic Blackjack card game, while incorporating user-friendly design elements and additional features.

The project began with the ambition of implementing a Blackjack game that included essential gameplay mechanics, such as managing a player's hand and calculating live value counts. Key features include a visually immersive experience with background images representing a casino table, a welcoming introductory screen, and a betting system. The betting system starts players with a balance of $1000, with strict limitations on betting amounts to ensure a structured gameplay experience. The player can also, of course, play the game of blackjack, which at a base level includes hitting to receive another card to their hand, standing, when you compare with the dealer to see who has a closer score to 21 without going over. We have implemented rules such that the player cannot keep hitting after exceeding 21, and the button disables actions that will not allow the player to cheat the system. We wanted to implement a system that would only allow the user to play after they bet, but it was cut short to time. We also wanted an end screen that would count all of the wins and balance changes that the player had while they played their rounds. This also got cut due to numerous bugs and time.

II. Team Members - Bios and Project Roles

Include an entry in a narrative form for each of your team members. The goal is to demonstrate the team's skills and project coverage. This is not just a pasted in resume, but a very short summary of your involvement in the project, and your technical interests. Include:

- Name
- Degree plan
- Project role - which aspects you're responsible for
- General areas of experience and technical interests

Jonathan Courter is a BA computer science major with interest in artificial intelligence, economics, data science, and music software development. His prior projects include, a full GUI version of the dice game Farkle, Small terminal based games, and an original Java game with custom drawn graphics. Jonathan's skills include C++, Java, and Python. For this project, I was responsible for coding a lot of the project logic, fixing bugs, making the main game panel, adding niceties like the background images, and much of the final report.

Parker is a current student of CS, and is planning to get an internship this summer and an IT job after school. He is also planning to minor in Applied Mathematics.
Using his previous experience in writing blackjack, Parker worked on some of the back end logic and planning for the project. He also worked on various other aspects such as testing, fixing bugs, GUI refinements and documentation.
He is experienced in making software projects, using programming to solve complex

mathematical equations, and program design. Parker has an interest in using computer science to make an impact on the real world.

TYLER'S HERE

III. Project Requirements

This section defines the key requirements for the Blackjack project, including the major features, constraints, and functionalities expected from the final design. The goal is to orient the reader to the quantified objectives that shaped the development of this project. These requirements were drawn directly from the initial project plan and reflect the core gameplay mechanics and user experience considerations necessary for a successful implementation.

The project encompasses several major features critical to delivering a functional and engaging Blackjack game. These include:

- **Cards**: A deck of cards must be implemented, shuffled, and distributed to the dealer and each player. Each card will have a value and suit, adhering to standard Blackjack rules.
- **Player**: Each player will have a hand containing their cards, a score representing the value of their hand, and a balance used for betting purposes.
- **Dealer**: The dealer, controlled by predefined rules, will have their own hand. The dealer's gameplay serves as the benchmark that players must try to beat.
- **Graphical User Interface (GUI)**: The game will be presented via a visually appealing and user-friendly interface. The GUI must clearly display all relevant game information, including the player's hand, the dealer's hand, the score, and the available actions.

Constraints and Implementation Goals

1. **Gameplay Mechanics**:
    - The game must adhere to traditional Blackjack rules, such as scoring, hitting, standing, and busting.
    - Players and the dealer follow a logical progression, with the dealer taking actions based on predefined rules.
2. **Scoring and Betting**:
    - Players start with a balance of $1000 for betting.
    - Betting amounts must be restricted based on the player's remaining balance.
3. **Technical Design**:
    - A modular structure ensures that classes, such as Player, Dealer, Card, Deck, and Game are well-defined and interact seamlessly.
    - A scalable architecture allows for potential future enhancements, such as adding more players or integrating multiplayer functionality.

User Interface Design

The initial user interface should include:

- A welcome screen to introduce players to the game.
- A main game view displaying the table, player hands, dealer hand, and current balances.
- Buttons for user actions like "Hit," "Stand," and "Bet."
- Feedback for gameplay events, such as when a player busts or wins.

Our Requirements are relatively simplistic, not unachievable, but we did run into some issues with the game logic especially with displayed cards and when and what a player can or cannot do during a turn. We wanted a basic blackjack game that felt playable and not entirely ugly as well. What we learned is that though our goals for the project were not super difficult, we still needed ample time to implement tests, and logic before working on a GUI. The team aspect of the project made things run a little less smoothly because we did not give it the respect it deserved. We had to rush to put together something that worked, and it still does not function 100% as intended. These are the kinds of things that cannot happen again and will not. Speaking for the group, we all learned that lesson over this project.

IV. Solution Approach

This section describes the technical implementation and design of the Blackjack project, detailing the architecture, components, and functionality of the final product. It includes the UML diagrams, sequence diagrams, and descriptions of the key modules used to achieve the project's goals. These elements demonstrate how the initial design was realized (and not) in the finished product.

Architectural Overview

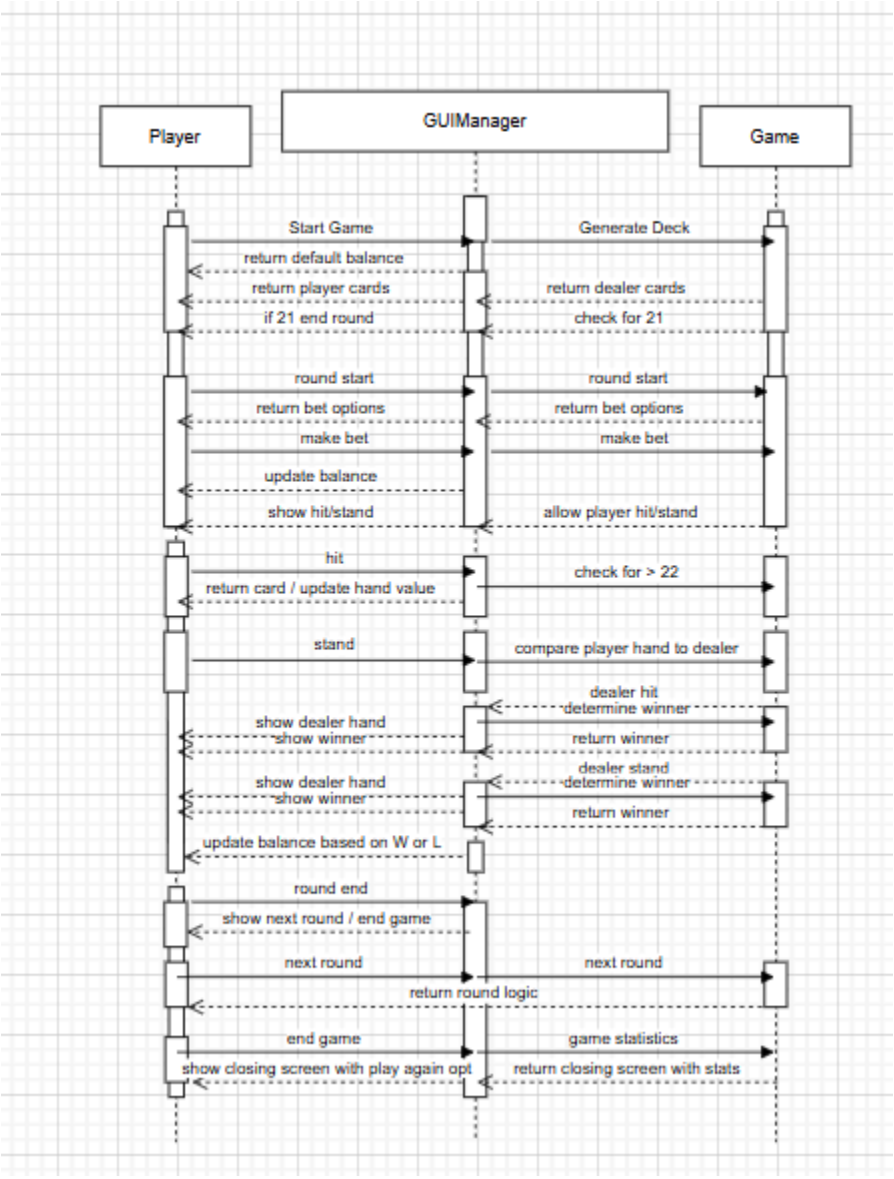
The project employs a modular design, dividing the system into distinct classes and components to facilitate code readability, maintainability, and scalability. The main components include:

- **Player Class**: A generic Player class serves as the base for the human player. Each player has attributes such as a name, and a balance for betting.
- **Dealer Class**: We were supposed to make a dealer class, but the implementation of it was much easier than needing a full class.
- **Card and Deck Classes**: Represent individual cards and the deck used in the game. The Deck class handles shuffling, drawing cards, and resetting when necessary.
- **GUI**: Provides the user interface for interacting with the game. It includes:
    - A **welcome screen** introducing the player to the game.
    - A **main game interface** displaying the table, player hands, dealer hand, and buttons for actions like "Hit," "Stand," and "Bet."
    - Dynamic updates, such as live score counts and feedback messages for gameplay events.

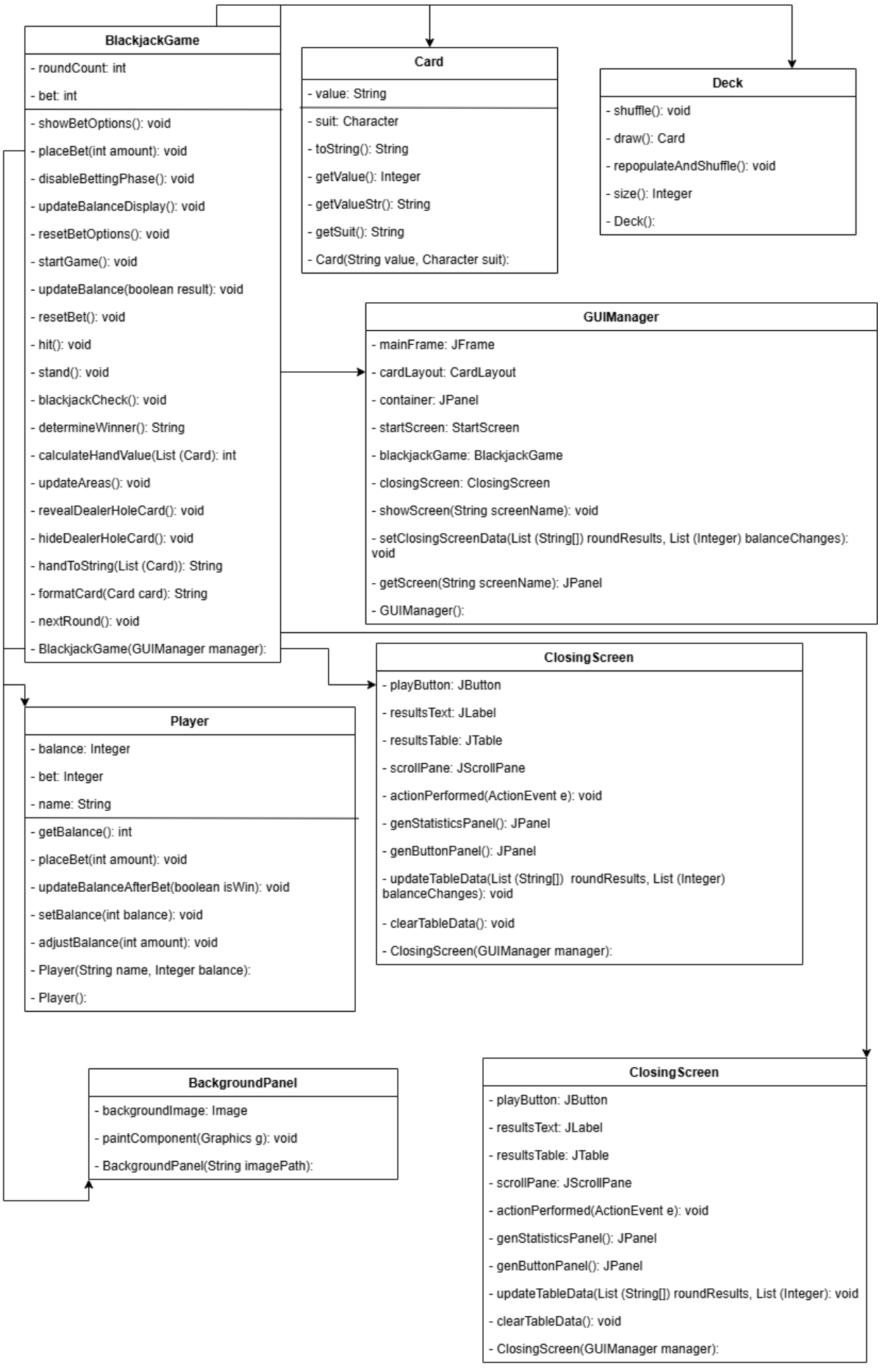
The game follows the standard flow of Blackjack:

1. Players place their bet using the GUI.
2. Each player is dealt two cards, and the dealer receives one visible card and one hidden card.
3. Players take turns choosing actions such as "Hit" to draw a card or "Stand" to end their turn.
4. Once the player finishes – pressing “stand,” the dealer plays according to predefined rules.
5. Scores are calculated, and balances are updated based on the outcome.

UML AND SEQUENCE DIAGRAM:



Tyler Yasukochi


BlackjackGame
- roundCount: int
- bet: int
- showBetOptions(): void
- placeBet(int amount): void
- disableBettingPhase(): void
- updateBalanceDisplay(): void
- resetBetOptions(): void
- startGame(): void
- updateBalance(boolean result): void
- resetBet(): void
- hit(): void
- stand(): void
- blackjackCheck(): void
- determineWinner(): String
- calculateHandValue(List (Card): int
- updateAreas(): void
- revealDealerHoleCard(): void
- hideDealerHoleCard(): void
- handToString(List (Card)): String
- formatCard(Card card): String
- nextRound(): void
- BlackjackGame(GUIManager manager):
Card
- value: String
- suit: Character
- toString(): String
- getValue(): Integer
- getValueStr(): String
- getSuit(): String
- Card(String value, Character suit):
Deck
- shuffle(): void
- draw(): Card
- repopulateAndShuffle(): void
- size(): Integer
- Deck():
GUIManager
- mainFrame: JFrame
- cardLayout: CardLayout
- container: JPanel
- startScreen: StartScreen
- blackjackGame: BlackjackGame
- closingScreen: ClosingScreen
- showScreen(String screenName): void
- setClosingScreenData(List (String[]) roundResults, List (Integer) balanceChanges): void
- getScreen(String screenName): JPanel
- GUIManager():
Player
- balance: Integer
- bet: Integer
- name: String
- getBalance(): int
- placeBet(int amount): void
- updateBalanceAfterBet(boolean isWin): void
- setBalance(int balance): void
- adjustBalance(int amount): void
- Player(String name, Integer balance):
- Player():
ClosingScreen
- playButton: JButton
- resultsText: JLabel
- resultsTable: JTable
- scrollPane: JScrollPane
- actionPerformed(ActionEvent e): void
- genStatisticsPanel(): JPanel
- genButtonPanel(): JPanel
- updateTableData(List (String[]) roundResults, List (Integer) balanceChanges): void
- clearTableData(): void
- ClosingScreen(GUIManager manager):
BackgroundPanel
- backgroundImage: Image
- paintComponent(Graphics g): void
- BackgroundPanel(String imagePath):
ClosingScreen
- playButton: JButton
- resultsText: JLabel
- resultsTable: JTable
- scrollPane: JScrollPane
- actionPerformed(ActionEvent e): void
- genStatisticsPanel(): JPanel
- genButtonPanel(): JPanel
- updateTableData(List (String[]) roundResults, List (Integer): void
- clearTableData(): void
- ClosingScreen(GUIManager manager):

Lift materials from your presentation and any other materials about your solution. This should demonstrate to the reader what your project included, at least according to the current design. Include your UML and sequence diagrams here.

BTW: If your UML object diagram has gotten too big, it's okay to break it into multiple pieces. You can even refer to objects defined in other UML diagrams if needed.

V. Test Plan

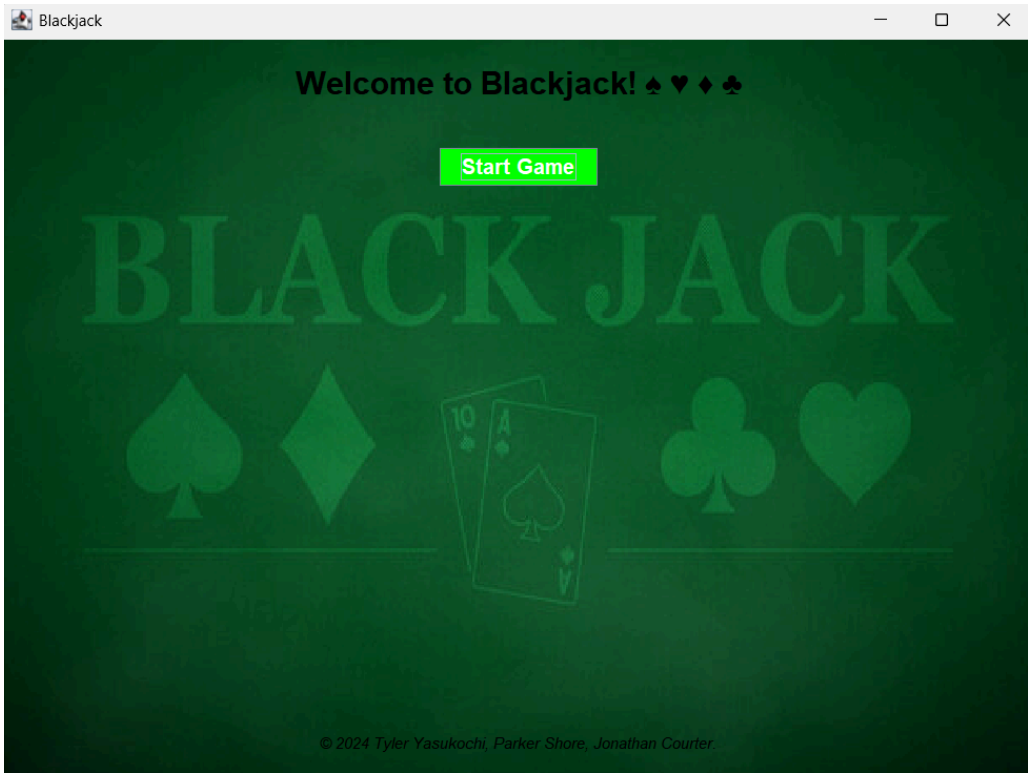
As we reflect on the project, it is clear that one of the biggest mistakes we made was not prioritizing testing from the start.We had people do user tests, such as my roommate and friends, but we had no unit tests by the end of the project. This was again, a very costly mistake, and avoidable if we had gotten started on the project right away. Despite our best intentions, we found ourselves scrambling to meet the deadline, and as a result, our test plan was severely neglected. In hindsight, we realized that this is a critical error. Without robust testing, we were unable to catch bugs before it would take hours to fix them. This just led to a more stressful and chaotic development process. As a result of starting late, we thought we could scrape together a project without testing, but this was clearly very wrong. As the project progressed, we encountered numerous issues that could have been caught and dealt with earlier. We were forced to rely on manual user testing, which was slow and hard to find bugs and edge cases until they showed up halfway through coding the project. If we could do it again, we would have firstly started on time, but next, prioritize testing and a strong foundation for what we want the program to look like organizationally. We will carry this lesson with us into future projects.

VI. Project Implementation Description

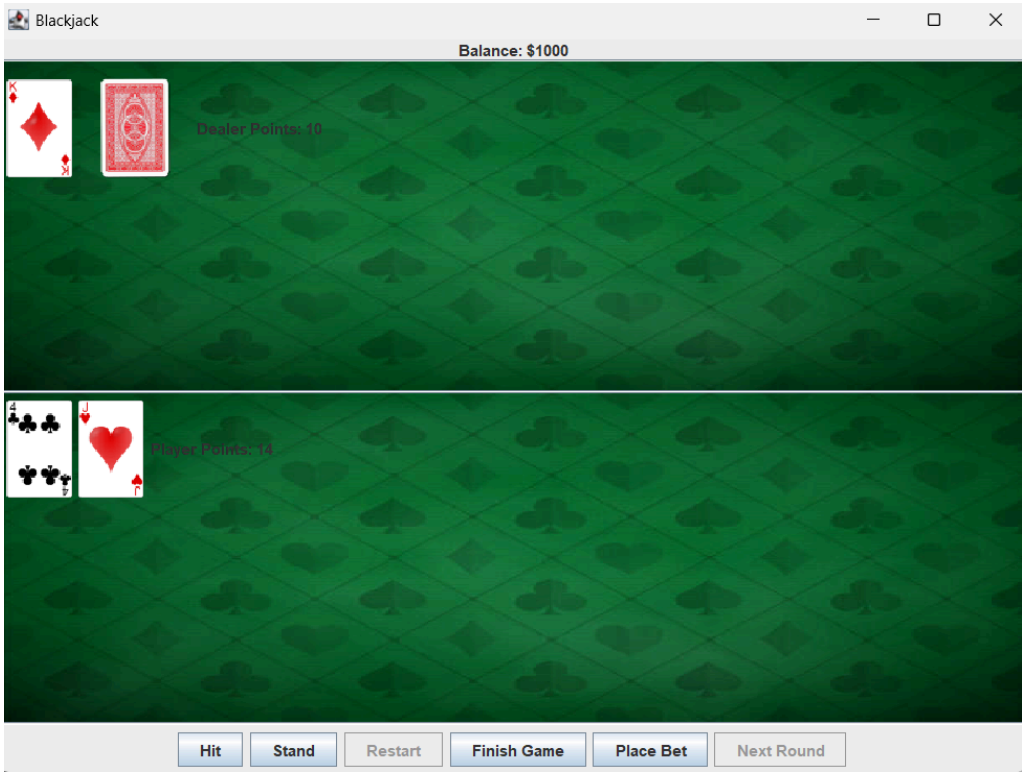
Our team did not get as far as we had hoped. We implemented all of the game features we wanted, however, the program is plagued with bugs due to our lack of testing early on in the project. Our current progress is as follows:
Game Logic: ~75-80% complete
GUI ~ 80% I would have liked to add more "prettiness"
Card/Deck - 100%
Player - 100%

Images of key moments:

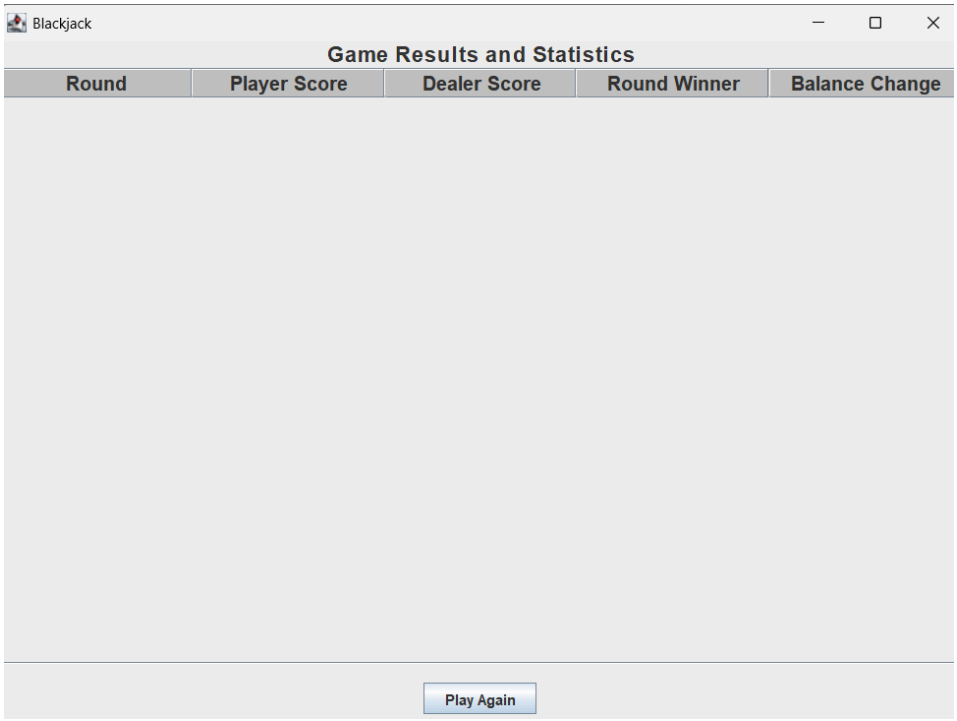
Intro Screen:



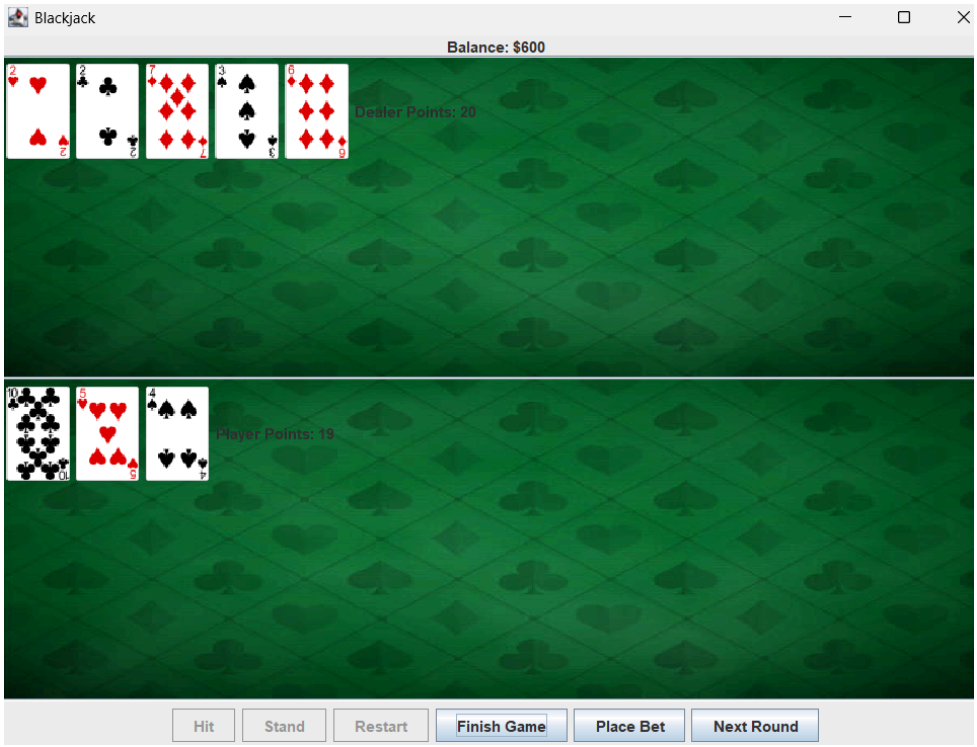
Main Game Panels:



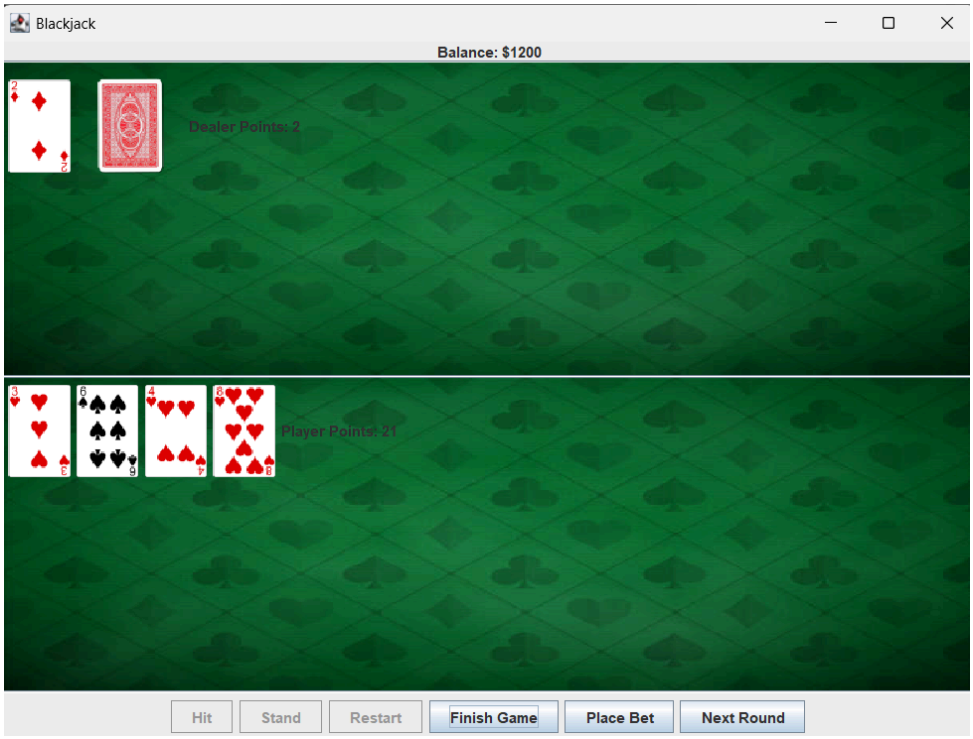
End Screen:



Loss:



Win:

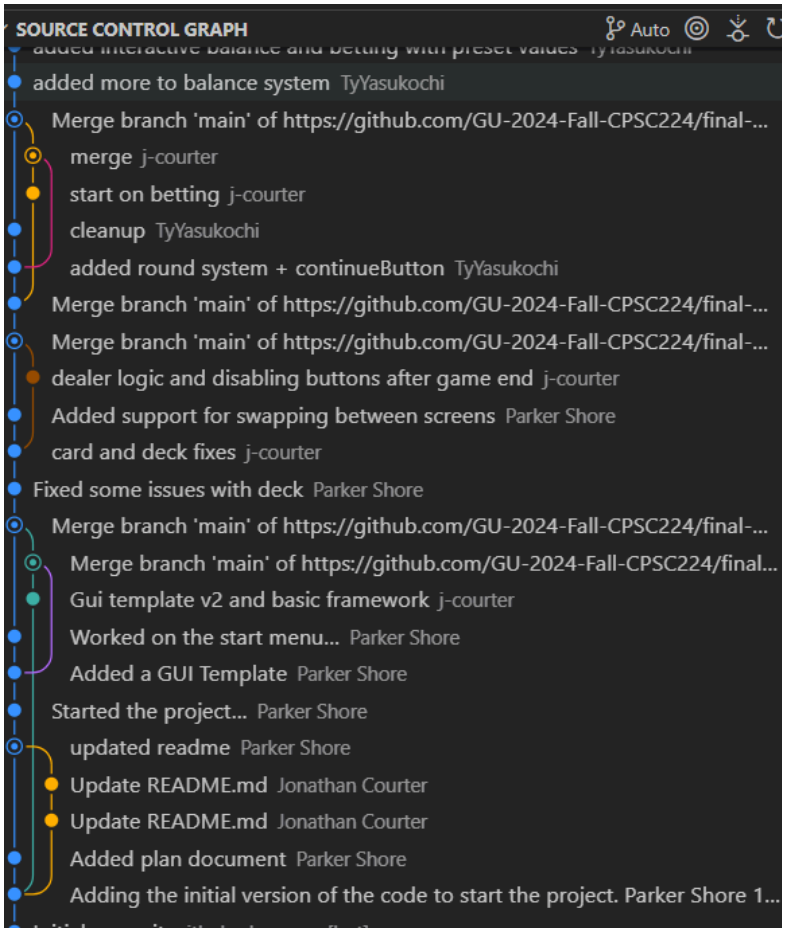


Our team fully implemented everything except some of the logic around when players can bet, the end screen functionality, and otherwise minor visual tweaks. The reason I estimated so low for game logic, is because without testing, we had no idea how to force players to bet before play, without disrupting the rest of the game. Things got out of control quickly and became less object oriented.

VI(a). Project Statistics

As of now, our project has a total of around 1037 lines of code in the project files. Parker wrote around 1309 lines in total over 10 commits according to Git. Jonathan wrote 1045 in 11 commits, and Tyler wrote 396 in 9 commits. We had a total of 6 merges. As of now, we still do not have any unit testing. This was a major issue too. Because we started coding so late, we did not have enough time to write unit tests before the actual program code, so that hurt us in the long run. This is a common theme with our project. I really wish we had gotten started right away. If we had started earlier, with time to write tests, our code would have been much more complete and run without as many bugs to squash. We did not create any media ourselves, but we used two images for the welcome screen and the main table panel. We also had an image for each of the cards in the deck, which totals to 54 total images. There was no music, however, I wanted to add it, it just got cut to time.

Required: Include a screenshot of your team's git repository commit tree.



https://github.com/GU-2024-Fall-CPSC224/final-game-project-git-good

VII. Team Collaboration and Tools Used

Give a short summary of how your team collaborated.

Our group used mostly messages to communicate. We made a discord server to send some files and images in the planning stage, and then did not use it since. In total, we met around once a week in person to talk about the project and code. Looking back, this was a huge mistake. I think our communication and lack of knowing what was happening and where we wanted to take the project led to a lot of the problems. If we had gotten even one more meeting in person per week, I think we would have been closer to a better final product. We coded individually as well as group sessions. Most of the commits were individual though. Typically someone would commit something, and then something else would not work, which was a huge problem for us as far as getting progress in the project. With better structuring around who was doing what, we could have saved a lot of time in terms of assigning people to code certain classes. Because I (Jonathan) did a lot of the work on the main game class, we had problems in deciphering how to implement everything else because only I knew how it worked. This I think is what hurt us the most as a group. Because we started so late, which was another huge mistake, we could not recover from a poorly designed project plan. I think if we reviewed my code as a group after making the base structure, it would have been a little easier to manage, as I am by

no means an excellent programmer. We used git, but mostly just coded on main, we had some branching, as well as either one or two merges, but they went smoothly for the most part. I have learned a lot in the process of this project. This was my first big group coding project and it went much slower than I had anticipated. Many things went wrong such as not being able to recover from some poor early decisions, and the code quickly became a mess to decipher. As of now, I think most of us would agree that we know it works, but could not track exactly how.

VIII. Future Work

If given more time, we would have liked to improve the Blackjack game, both in functionality and user experience. These future implementations aim to expand gameplay possibilities, add complexity, and further refine the technical aspects of the project. Before any of this, we would like to fix the current state of the project.

Multiplayer Functionality

One significant enhancement would be introducing multiplayer capabilities, allowing multiple players to compete in the same game. This could include:

- **Local Multiplayer**: Multiple players could take turns on the same device, each managing their own hand and balance.

Enhanced Betting System

To create a more engaging gambling experience, the betting system could be expanded with features such as:

- **Side Bets**: Players could place bets on outcomes like the first card being an Ace or getting a pair in the first two cards.
- **User Defined Bets:** Players would select the amount they would like to bet on a slider or by typing in how much they would like to bet to a popup when selecting the “bet” button.

Deck Customization and Variants

To diversify gameplay, we could introduce the ability to play Blackjack variants or customize the rules:

- **Multiple Decks**: Options to play with multiple decks, changing the probabilities of drawing certain cards.
- **Rule Variants**: Adding gameplay variations like allowing users to double down, or split if they draw the same value card.

Statistical Analysis Tools

A statistical analysis feature would allow players to review their gameplay and improve their strategies. Features could include:

- Win/loss ratios.
- Average bet amounts.
- Insights into how often specific actions (like hitting on certain scores) resulted in wins or losses.

Immersive Graphics and Animations

We could further enhance the GUI with:

IX. **Card Animations**: Smooth animations for dealing and flipping cards.
X. **Sound Effects**: Background music and sound effects for card movements
XI. **Themed Tables**: Players could select table designs or themes for a personalized experience.

XII. Glossary

Blackjack: A card game where the goal is to achieve a hand value of 21 or as close to it as possible without exceeding it. Players compete against a dealer following predefined rules for hitting, standing, and scoring.

Bust: A situation in Blackjack where a player's hand value exceeds 21, resulting in an automatic loss for that round.

Hit: An action in Blackjack where the player requests another card to be added to their hand.

XIII. References

XIV. Appendices

As needed, copy over your appendices for the various sections. You can have as many appendices as required. Normally, they're numbered with letters:
Appendix A
Appendix B
…
Appendix *n*