CLUE

Final Report

Kernel Mustard
David, Sam, Waylan, and Charlie

Course: CPSC 224 - Software Development

Introduction and Project Description

For our final project, we developed a fully playable, GUI-based version of the board game *Clue* using Java and the Java Swing framework. We aimed to recreate the original board game's logic, flow, and core mechanics while presenting it through an intuitive and visually interactive interface. This involved implementing player movement, turn-based logic, card distribution, suggestions, accusations, and win/loss conditions in a digital format.

This report summarizes the development process, documents our engineering decisions, and provides insight into our project's architecture, testing strategy, and collaboration methods. It also serves as a technical reference for future improvements or iterations of the game.

We based the structure and rules of our implementation on Hasbro's official *Clue* board game. Elements such as characters, weapons, and rooms were directly adapted, and game flow elements like suggestions and accusations were faithfully reproduced. Any adaptations or simplifications were made to suit the scope of a semester project while retaining the original game's spirit.

Team Members - Bios and Project Roles

David Albelais | *Computer Science Major*

David is a computer science student interested in software engineering. His skills include C++, Java, and Assembly. For this project, he was relied upon to create the accusation and suggestion logic for the GUI. Additionally, with the accusation logic came the deletion of players and their respective tokens on the game board.

Samuel Allen | *Computer Science Major*

Sam is a computer science student interested in anything related to machine learning. His skills include C++, Python, Java, Assembly, and PyTorch. For this project, his responsibilities primarily included developing the game board, creating a basic multiplayer prototype, and performing miscellaneous work on the GUI.

Waylan Parsell | *Computer Science Major*

Waylan is a computer science student learning everything he can in the field, with an extended focus and personal learning of game design and development. For this project, he contributed both game files and GUI development, creating the Map, Player, and Cards, in addition to creating some of the opening game GUI features, such as the title screen, player set-up, and player movement system.

Charlie Serafin | *Computer Science Major*

Charlie wrote Part 2: Project Requirements and Part 4: Project Schedule in the group documentation. These sections established the functional objectives of our Clue game and mapped out the timeline we followed to meet key development milestones.

In terms of coding, he contributed approximately 300–400 raw lines. His primary development work focused on the following files:

- **GameOutcome.java** – Implemented logic for determining win and loss conditions and managing the end-of-game state transitions.
- **MainGame.java** – Wrote the game's initial setup and launch routines, handling startup flow and initializing key components.
- **PlayGame.java** – Developed the core game loop, including player turn sequencing, movement mechanics, and handling suggestions and accusations.
- **Suggestion.java** – Built the logic for handling in-room suggestions and triggering the disproving process among players.

Project Requirements

This section outlines the key functional requirements and interface expectations that shaped our implementation of Clue. These requirements were drawn directly from our project plan and were the foundation for our design and development.

Functional Requirements:

- **Game Setup**: At the start of each game, a secret solution (suspect, weapon, room) must be generated, and the remaining cards must be distributed randomly and evenly among the players.
- **Turn-Based System**: The game must support a structured, sequential turn order that allows players to roll dice, move, make suggestions, or accuse.
- **Suggestions and Refutations**: Players must be able to make a suggestion when inside a room, and the game should enforce the correct disproving sequence across other players.

- **Accusation and Win Conditions**: Players can make final accusations; the game ends if they are correct, or eliminates them if they are wrong.

- **Player Interface**: The GUI must communicate player status, cards, recent actions, and available moves.

User Interface Requirements:

- Top-down board map with rooms and character positions

- Log panel for game history (moves, suggestions, responses)

- Player info panel showing name, color, and cards (hidden from others)

- Interactive buttons for actions: Move, Suggest, Accuse, View Cards, End Turn

Solution Approach

Our solution involved designing an object-oriented architecture with clearly defined modules to manage the game state, user interactions, and backend logic.

Architecture Components:

- Player.java, Room.java, Board.java: Handle game entities and spatial navigation

- Card.java, Deck.java: Manage card creation, shuffling, and dealing

- PlayGame.java: Acts as the game controller, handling player actions, turn progression, suggestions, and accusations

- MainGame.java: Handles game initialization and GUI launch

- Suggestion.java, Accusation.java, GameOutcome.java: Manage gameplay mechanics and win/loss scenarios

GUI Design (Java Swing):
We built the GUI based on two interface mockups:

1. A traditional layout with board display, action log, and manual inputs
2. A refined layout with action buttons for a more modern and intuitive experience

UML/Sequence Diagrams:
We developed a UML class diagram to represent the main relationships in our system and two sequence diagrams to model user turn flow and full-game interactions.

(See **Appendix B** for diagrams.)

Test Plan

We employed a mix of manual and informal user testing to verify the game's correctness and usability.

Functional Testing:

- Played through full games to test turn order, user input handling, and win/loss conditions
- Validated GUI behavior under different player counts and scenarios

User Testing:

- Roommates and friends participated in play sessions and provided feedback on clarity, bugs, and game flow
- Observations improved button labeling, action prompts, and flow control

Project Implementation Description

We implemented all major subsystems of our initial architecture, resulting in a fully playable and interactive game. The current status of principal components:

- Core Game Logic
- Suggestion/Accusation Mechanics
- Turn Rotation & Player Management
- Card Distribution & Solution Generation
- GUI & Game Board Rendering
- End-Game Detection

We also implemented:

- Mouse- and button-driven user interaction
- Real-time updates to the action log
- Full synchronization between player actions and GUI state

(See **Appendix D** for screenshots and code snippets.)

Project Statistics

As a team, we wrote roughly 2250 lines of code.

Estimated Total Lines Written By:

- Sam: 1721
- Charlie: 166
- David: 120
- Waylan: 447

We only had 1 unit test running, but we primarily hand-tested our GUI.

Number of Commits:

- Sam: 12
- Charlie: 4
- David: 1
- Waylan: 8

You can find a screenshot of our GitHub commit history in Appendix E

LINK TO GITHUB REPO

Team Collaboration and Tools Used

Our team collaborated using a mix of digital tools and in-person meetings to manage progress, divide tasks, and solve integration challenges.

Communication Tools:

- **Discord**: Primary platform for daily messaging, voice calls, and planning discussions.
- **iMessage**: Used for quick updates, reminders, and coordinating in-person sessions.
- **Computer Science Study Room**: Our main workspace for live collaboration, pair programming, and debugging.

Development Tools:

- **GitHub**: Used for version control, with a structured workflow involving branches and pull requests to keep code changes organized and reviewable.
- **Git Project Board**: Helped track tasks, assign responsibilities, and monitor progress toward milestones.

Team Workflow:

- We met in person approximately 5 times during the project, with all members attending most sessions.
- Most code was written individually, but reviewed and integrated collectively.
- We used feature branches to work in parallel, then merged through pull requests, often with code reviews or group walkthroughs before approval.
- Debugging was a collaborative effort — we regularly solved merge conflicts and GUI inconsistencies.

Lessons Learned:

- Importance of clear Git practices — merging frequently and communicating about changes prevented major integration issues.
- Code review helped identify logic bugs and enforce consistent design patterns.
- Team software development requires flexible roles — everyone contributed across logic, GUI, and testing.

Future Work

While our Clue game implementation meets the core requirements and is fully playable, there are several features and improvements we would pursue if given more time or future development cycles.

Potential Feature Enhancements:

- **AI Players**: Implement computer-controlled players to allow single-player or mixed matches.
- **Save/Load Functionality**: Add serialization support to save and resume games.
- **Online Multiplayer**: Transition the game to support remote play over a network.
- **Improved Graphics**: Upgrade the GUI with enhanced visuals, animations, and sound effects for a more engaging user experience.
- **Settings Menu**: This menu allows users to configure game options such as the number of players, board themes, or game speed.
- **Hint System**: Add an optional hint or notepad system to help players track clues.

Codebase Improvements:

- Refactor core logic for improved modularity and unit test coverage.
- Add automated testing for GUI interactions and edge-case scenarios.
- Improve the separation between game logic and GUI for easier maintenance and future portability.

Glossary

- **GUI (Graphical User Interface)**: A visual interface that allows users to interact with a program through graphical elements like buttons, panels, and images instead of text-based commands.

- **UML (Unified Modeling Language)**: A standardized modeling language to visualize the design and relationships between software components in an object-oriented system.

- **Object-Oriented Programming (OOP)**: A programming paradigm around objects encapsulating data and behavior through classes and methods.

- **Unit Test**: A software test that checks the functionality of a specific section (unit) of code, typically an individual function or method.

- **Commit**: A snapshot of changes made to a codebase, recorded in version control (e.g., GitHub) for collaboration and rollback purposes.

- **Pull Request (PR)**: A method used in Git-based development to propose codebase changes, allowing team members to review and discuss before merging.

- **Branch**: An independent line of development in version control, often used to isolate new features or fixes from the main codebase until they're ready to be merged.

- **Serialization**: The process of converting an object into a format (like a file or byte stream) that can be saved and restored later.

References

- Hasbro. *Clue Game Official Rules*.

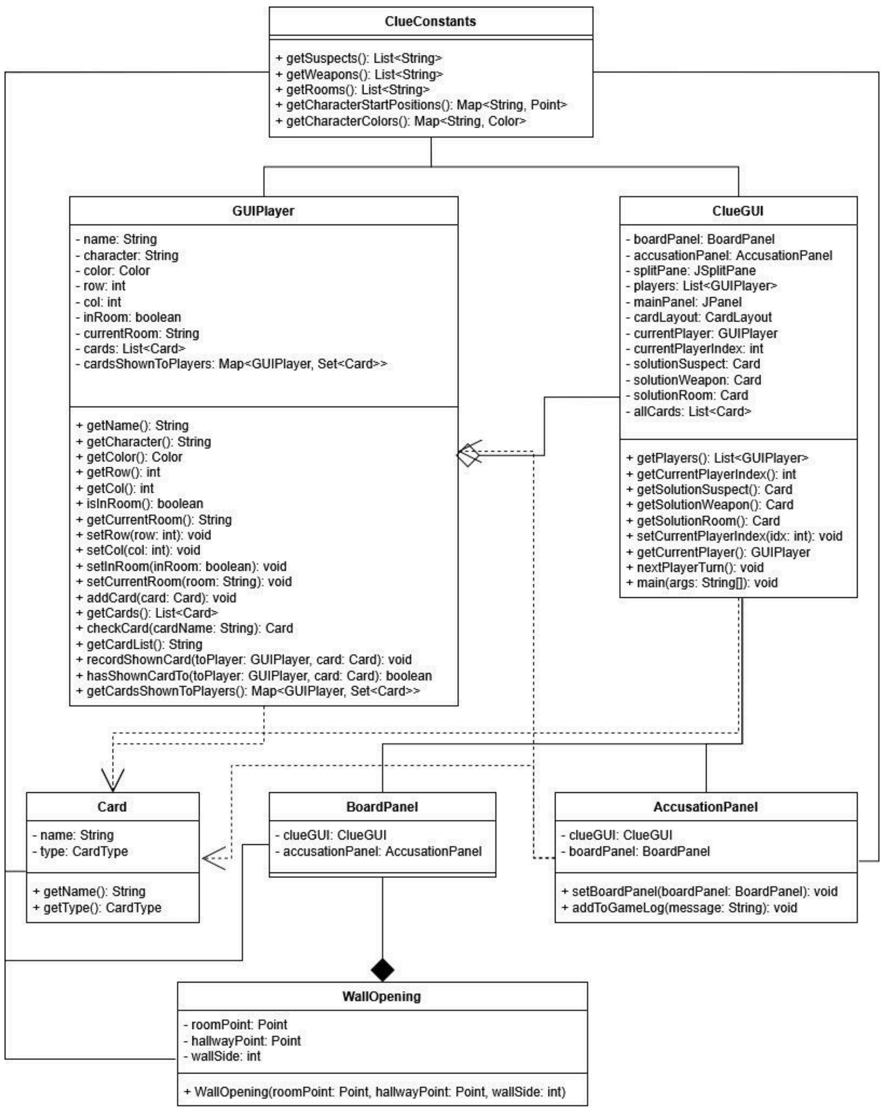
Appendices

- **Appendix A – Clue Game Rules**
  Summary or link to official Clue board game rules for reference.

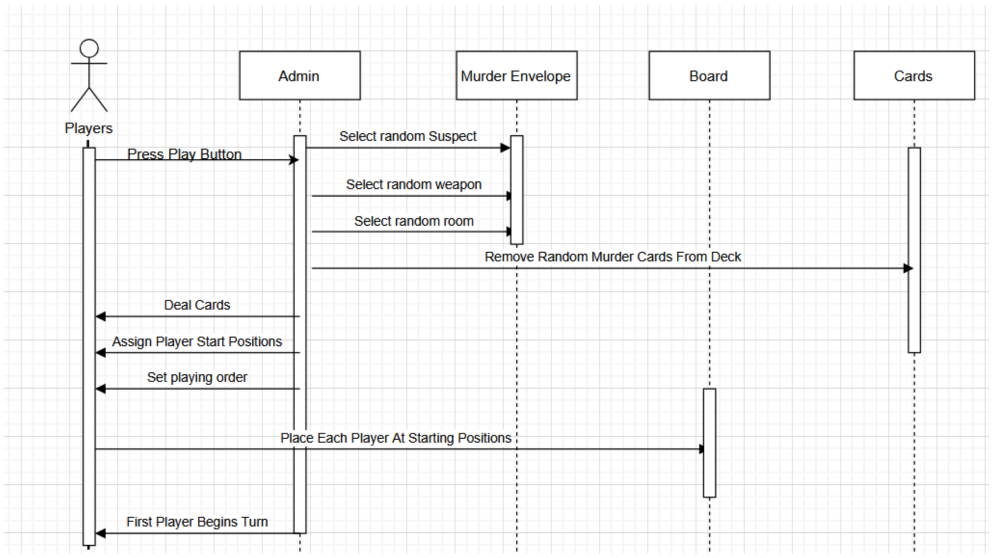
  00045_en-us_clue-game.pdf

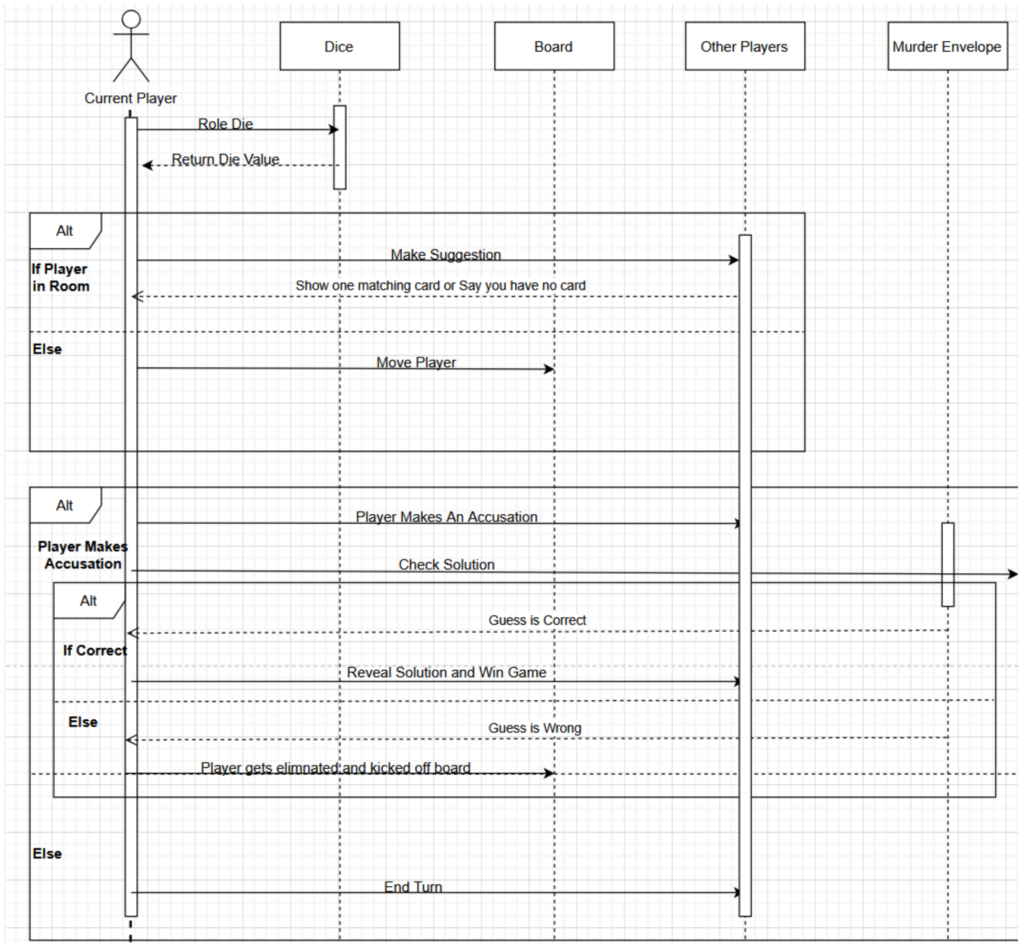
- **Appendix B – UML and Sequence Diagrams**
  Screenshots or diagrams showing class relationships and interaction flows.

UML:


ClueConstants
+ getSuspects(): List<String>
+ getWeapons(): List<String>
+ getRooms(): List<String>
+ getCharacterStartPositions(): Map<String, Point>
+ getCharacterColors(): Map<String, Color>
GUIPlayer
- name: String
- character: String
- color: Color
- row: int
- col: int
- inRoom: boolean
- currentRoom: String
- cards: List<Card>
- cardsShownToPlayers: Map<GUIPlayer, Set<Card>>
+ getName(): String
+ getCharacter(): String
+ getColor(): Color
+ getRow(): int
+ getCol(): int
+ isInRoom(): boolean
+ getCurrentRoom(): String
+ setRow(row: int): void
+ setCol(col: int): void
+ setInRoom(inRoom: boolean): void
+ setCurrentRoom(room: String): void
+ addCard(card: Card): void
+ getCards(): List<Card>
+ checkCard(cardName: String): Card
+ getCardList(): String
+ recordShownCard(toPlayer: GUIPlayer, card: Card): void
+ hasShownCardTo(toPlayer: GUIPlayer, card: Card): boolean
+ getCardsShownToPlayers(): Map<GUIPlayer, Set<Card>>
ClueGUI
- boardPanel: BoardPanel
- accusationPanel: AccusationPanel
- splitPane: JSplitPane
- players: List<GUIPlayer>
- mainPanel: JPanel
- cardLayout: CardLayout
- currentPlayer: GUIPlayer
- currentPlayerIndex: int
- solutionSuspect: Card
- solutionWeapon: Card
- solutionRoom: Card
- allCards: List<Card>
+ getPlayers(): List<GUIPlayer>
+ getCurrentPlayerIndex(): int
+ getSolutionSuspect(): Card
+ getSolutionWeapon(): Card
+ getSolutionRoom(): Card
+ setCurrentPlayerIndex(idx: int): void
+ getCurrentPlayer(): GUIPlayer
+ nextPlayerTurn(): void
+ main(args: String[]): void
Card
- name: String
- type: CardType
+ getName(): String
+ getType(): CardType
BoardPanel
- clueGUI: ClueGUI
- accusationPanel: AccusationPanel
AccusationPanel
- clueGUI: ClueGUI
- boardPanel: BoardPanel
+ setBoardPanel(boardPanel: BoardPanel): void
+ addToGameLog(message: String): void
WallOpening
- roomPoint: Point
- hallwayPoint: Point
- wallSide: int
+ WallOpening(roomPoint: Point, hallwayPoint: Point, wallSide: int)

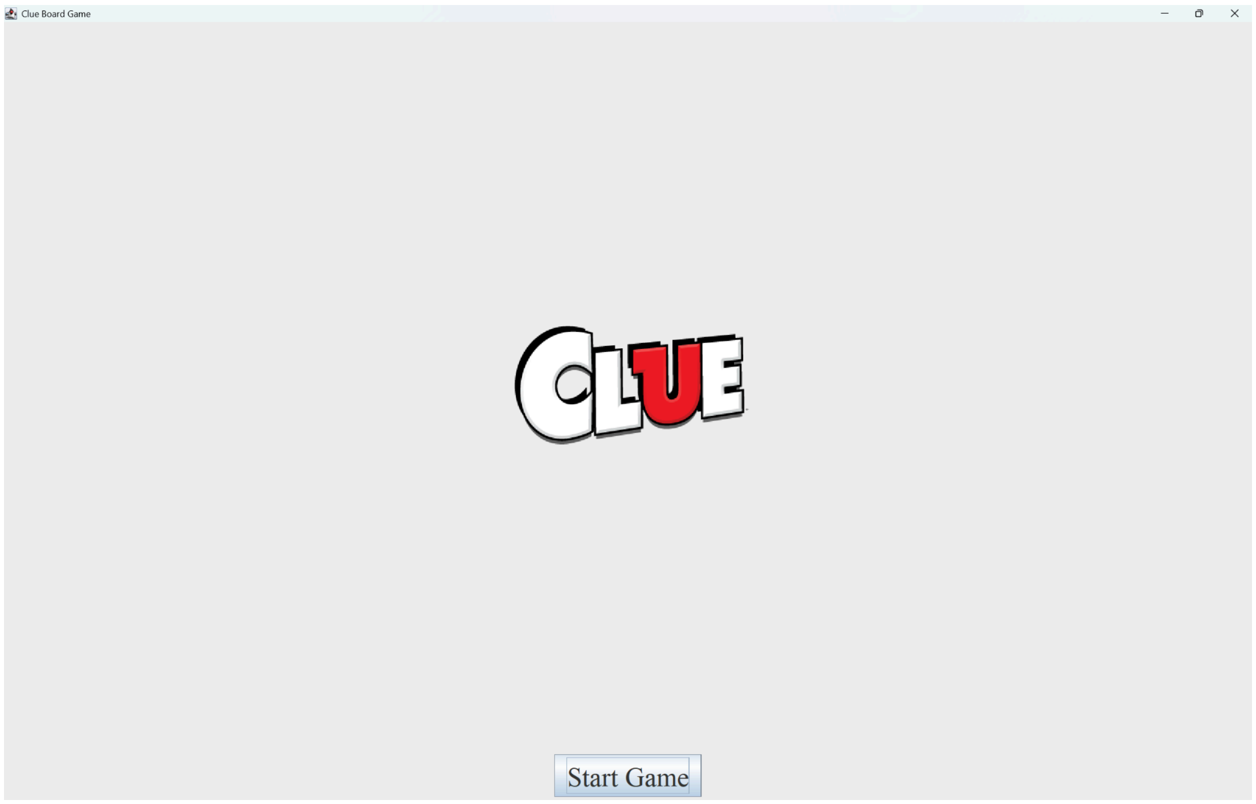
Sequence Diagrams:



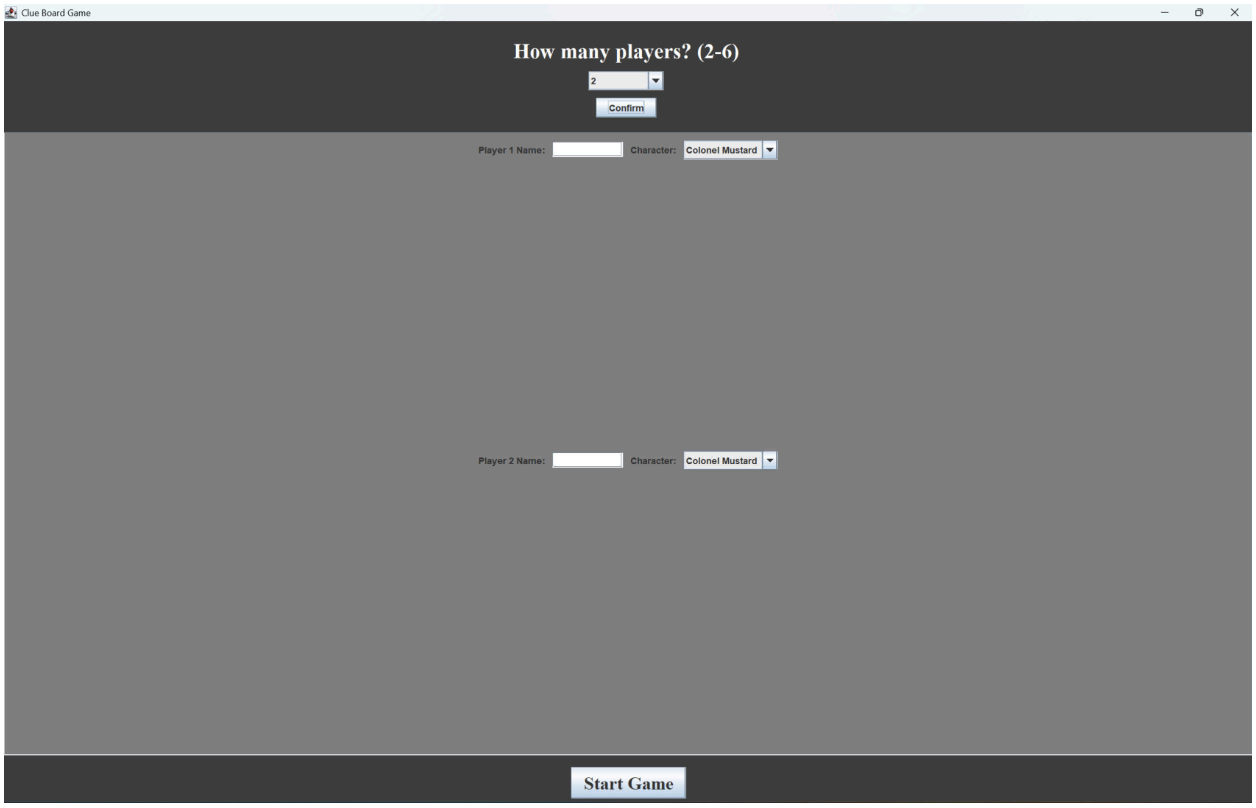


- **Appendix C – GUI Screenshots**
  Images of the running interface, including gameplay moments and features.

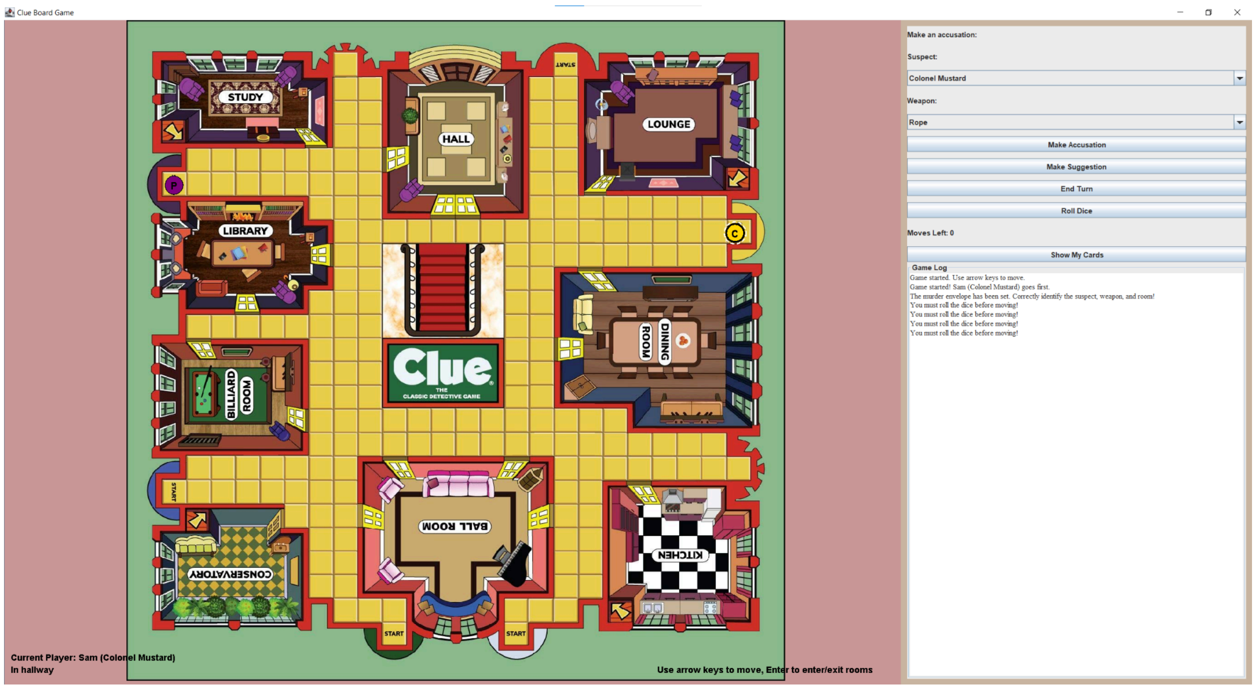
Start Screen:



Player Setup:



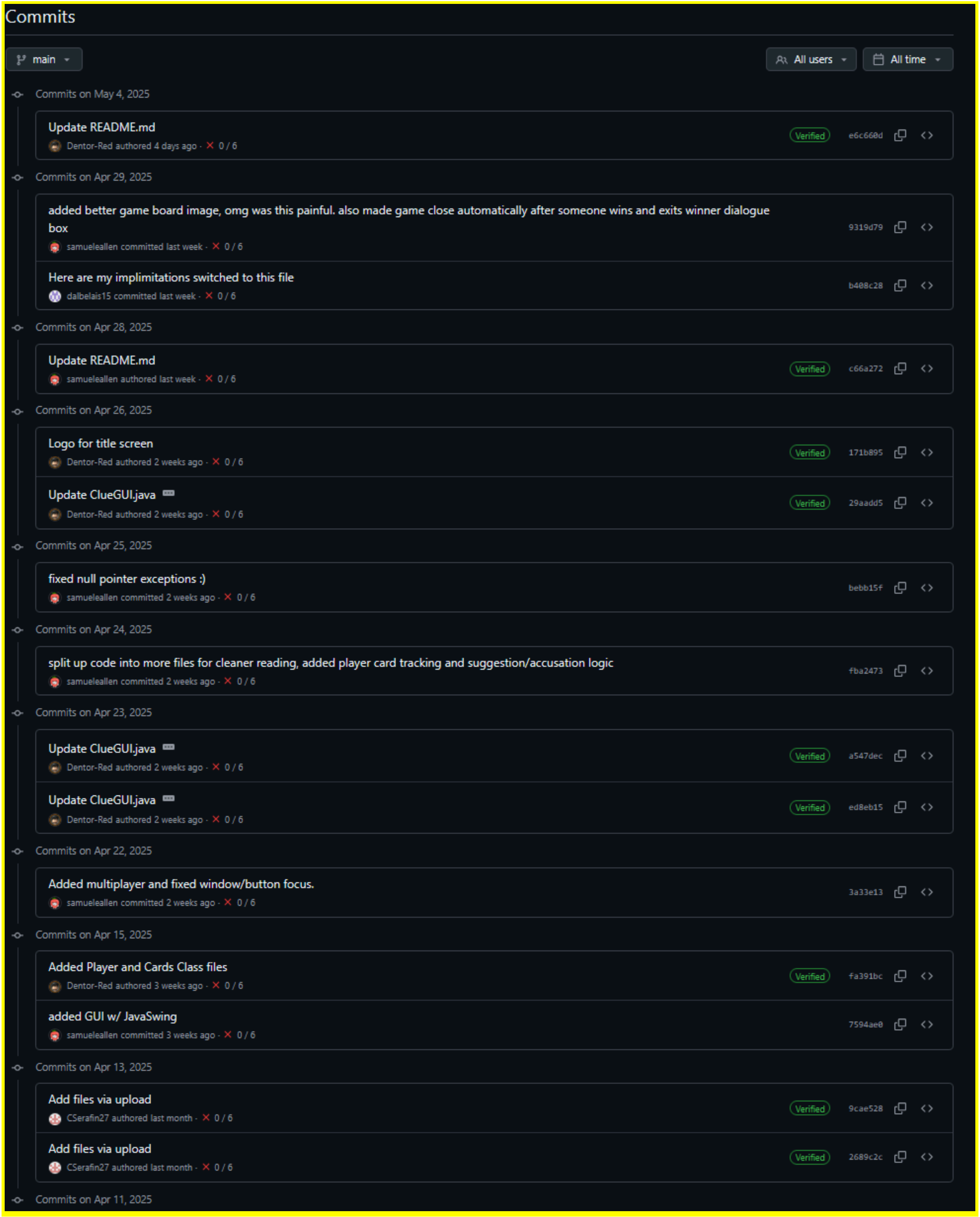
Board Gameplay:



Current Player: Sam (Colonel Mustard)
In hallway

```
Game started. Use arrow keys to move.
Game started! Sam (Colonel Mustard) goes first.
The murder envelope has been set. Correctly identify the suspect, weapon, and room!
You must roll the dice before moving!
You must roll the dice before moving!
You must roll the dice before moving!
You must roll the dice before moving!
```

- **Appendix D – GitHub Repository Activity**
  Screenshot of the commit history, branches, and pull requests.



Commits

main | All users | All time

Commits on May 4, 2025

- Update README.md — Dentor-Red authored 4 days ago · ✕ 0 / 6 — Verified — e6c660d

Commits on Apr 29, 2025

- added better game board image, omg was this painful. also made game close automatically after someone wins and exits winner dialogue box — samueleallen committed last week · ✕ 0 / 6 — 9319d79
- Here are my implimitations switched to this file — dalbelais15 committed last week · ✕ 0 / 6 — b408c28

Commits on Apr 28, 2025

- Update README.md — samueleallen authored last week · ✕ 0 / 6 — Verified — c66a272

Commits on Apr 26, 2025

- Logo for title screen — Dentor-Red authored 2 weeks ago · ✕ 0 / 6 — Verified — 171b895
- Update ClueGUI.java — Dentor-Red authored 2 weeks ago · ✕ 0 / 6 — Verified — 29aadd5

Commits on Apr 25, 2025

- fixed null pointer exceptions :) — samueleallen committed 2 weeks ago · ✕ 0 / 6 — bebb15f

Commits on Apr 24, 2025

- split up code into more files for cleaner reading, added player card tracking and suggestion/accusation logic — samueleallen committed 2 weeks ago · ✕ 0 / 6 — fba2473

Commits on Apr 23, 2025

- Update ClueGUI.java — Dentor-Red authored 2 weeks ago · ✕ 0 / 6 — Verified — a547dec
- Update ClueGUI.java — Dentor-Red authored 2 weeks ago · ✕ 0 / 6 — Verified — ed8eb15

Commits on Apr 22, 2025

- Added multiplayer and fixed window/button focus. — samueleallen committed 2 weeks ago · ✕ 0 / 6 — 3a33e13

Commits on Apr 15, 2025

- Added Player and Cards Class files — Dentor-Red authored 3 weeks ago · ✕ 0 / 6 — Verified — fa391bc
- added GUI w/ JavaSwing — samueleallen committed 3 weeks ago · ✕ 0 / 6 — 7594ae0

Commits on Apr 13, 2025

- Add files via upload — CSerafin27 authored last month · ✕ 0 / 6 — Verified — 9cae528
- Add files via upload — CSerafin27 authored last month · ✕ 0 / 6 — Verified — 2689c2c

Commits on Apr 11, 2025