Clarifications and Simplifications
During the early development progress, specification ambiguities were clarified through group discussions. We initially talked about the game logic so our implementation would be simple and manageable for all of us to contribute to. Ensuring the core functionalities of the game were retained was vital if we were going to be able to understand how the game was meant to work. Additionally, we had to talk about how the game was going to be presented as a text-based user interface design to ensure it can be a clear and playable representation of what we were meant to be presenting.

What we discussed in the early stages:

- The grid/Image that was provided in the assignment handout was the same structure as what we were to be presented. Also in the game Cluedo, there are passages between each estate, however not represented/clarified in this implementation

- Implemented Door by establishing estate size and position on the board. From here we could determine walls that were inaccessible for players to move through (this also contributed to the grey walls). From this, we could determine the cell position of the doors and if the player is in front of the Door they will gain a prompt that will allow them to enter as well as once inside they would be able to exit either door of the estate.

- If there are only three players then we would need to make the 'fourth' character to be controlled by the 'Computer' and the others be able to be controlled by the text-based characters

- Decided to display character decisions, cards, refutable cards, and the estate's name the under the board. This allowed for our game to be played properly.

- When the player enters an estate, they are displayed the estate name, the choice to end the turn, the choice to exit the estate so they can use their remaining moves, as well as make a 'guess' which is only optional within an estate.

- An incorrect solve attempt would result in the player being removed. However, implemented the removed player in the refutation process.

CRC Cards and UML Diagrams

We planned how to organise important functions and designs for the game components (Board, Walls, Cells, Grey Squares, Positioning). We discussed how to add the game features using this structure. Our main aim was to ensure that adding new methods would be simple and wouldn't disrupt the existing setup, making debugging easier.

- Establishing the responsibilities and relationships of each class.
- Ensure all group members will conform, ensure and contribute to the desired plan
- Be able to identify a problem and add it to the cards so everyone is aware of what is needed.

Game Specifications

- Estate
    - Players entering one of the five estates will display the estate they are about to enter
    - Gives them a choice to end turn when in an estate if the guess is not made yet
    - The choice to exit the current estate from either door in the estate
    - Guess within the estate
- Solve Attempt/Guessing
    - Allows the player to make a final guess to solve the murderer
    - If incorrect, the player who attempted the final guess will be removed from the game
    - Still a part of the refutation process due to Cluedo rules
- Users playing game
    - If fewer than four users were playing the game, the game would not be able to play out with less than four users
    - If we had more time we would design a Computer-controlled player design so our product would be playable with only one user (design choice)
    - Create log to create playability for the user (design choice)

Design Contributions

We chose to collaborate in pairs while creating classes to ensure clear logic and communication among team members. Bernard del Puerto and I were in charge of the Board and Player classes as well as coordinating with Alex Wells and Phil Sparrow who handled the others. We merged our work, fixed issues, and refined methods to enhance functionality once base implementation was finished.

General Contributions

- CRC cards
- UML Diagrams
- Program Layout

Board Class

- I focused on coding check methods within the class that were linked to Character and Estate objects
- Character implementation included placement on board (drawPlayer()), navigation (checkDirections(), lookForChar()), and engagement with estates
- Estates were positioned on board (drawEstate(), drawAllEstatesAndDoors()) and door locations dependent on character (checkForDoorsEstate(), lookForChar()).
- drawAllEstatesAndDoors() included working with Phil and Alex as they designed Estate and Door classes.

- Added render method, displays informative logs displayed horizontally board showing estates, cards and refutation

Player Class

- Early on, we agreed game object, not the board, would implement this class, due to player information needed for certain functions
- Focused on designing coding methods using Card, Estate, Guess and movement objects
- Player movement utilized Enums, including setting initial positions (Enum characterPosition)
- Worked closely with Alex and Phil on object methods to ensure effective collaboration and coding

Game, Card, Estate, CharacterCard, WeaponCard, Moves, SolveAttempt Classes

- Design contributions
- Debug/Code contributions

Challenges and Improvement

Initial CRC and UML designs provided solid foundation for team. We made some logic adjustments in certain classes to ensure functionality. An example of this, Player class, our UML indicated the board containing players. However, we implemented it within the Player class. This decision ensured player information for functions in game class(rules) was working

- Simplify more intricate player class that handles methods instead of dispersing functions across classes

Setting up movement in and out of states

- We had issues with player movement early on when it came to movement in and out of estates. Where players would not be present on the board after choosing to exit from either door. We found that this was to do with the location method in the Player class and how it was associated with the board.

Collaboration and Communication

- When we merged classes, methods needed to be adjusted/changed to ensure functioning. This was due to lack of communication about implemented functions that are interconnected across classes as well as code being shared via multiple ZIP files. An example was methods involving Move and Player objects (e.g. performPlayerMove() utilized in Game class)
    - Utilizing resources like GitHub or GitLab. These would have facilitated everyone staying updated on each other's progress and allowed for suggestions or resolutions in case or disagreements for improvement.

Challenges and Improvement

Early Design

- CRC and UML design at early stages were well thought out and was a good basis for the group to work off. In some cases of the classes we had to change our logic due functionality. An example is in the Player class, the UML showed the game class will contain the board and the board will contain the players. Myself and Bernard decided it was best if the player instead was contained by game since it required player information for certain functionalities to work properly (e.g. rules)

    - More complex player class that handled rules of game, instead of functions and methods being spread across multiple

Setting up movement in and out of states

- We had issues with player movement early on when it came to movement in and out of estates. Where players would not be present on the board after choosing to exit from either door. We found that this was to do with the location method in the Player class and how it was associated with the board.

Refutation methods

- During later coding when we had combined all our code the main issue that was presented was our refutation methods. This involved our guessing methods, player classes and solving methods in general. Solving these issues mainly included the player class and its relation to the Game class.
    - More complex player class that would handle all its methods without multiple being spread between multiple.

Collaboration of methods

- When we combined all our classes as one, some of the methods that were created had to be redone or methods had to be added in order for our design to work properly. This was due to a lack of communication between everyone at the earlier stages of coding and understanding of what class would need what specific methods in order for the other ones to work. An example of this were methods in the Game class which required both Move and Player objects (e.g. performPlayerMove(Moves moveChoice, Player player, Scanner sc))
    - Improve by not fully separating code into different parts and planning on better method development at early stages.

Communication

- While group work in early and later stages was always satisfactory, communication and collaboration could have been better by using resources such as Github and Gitlab so everyone could keep up to date on what everyone was doing as well as make recommendations on if we disagreed or could improve on that specific area of design. We used zip files that we posted to our discord group chat which added complexity and error. Early identification of potential design flaws could have been improved from early stages and overall game implementation.
    - First time working in a group so we know what to do and what not to do now

UML Diagrams

- Writing up the UML, using Umple was a struggle based off what we were trying to design and our thought process. This proved to be an issue as we could not refer to it as an accurate representation of what everyone was doing during early coding stages and team members could get confused.

- Look to other software to improve efficiency and accuracy to what we are trying to provide