Assumptions

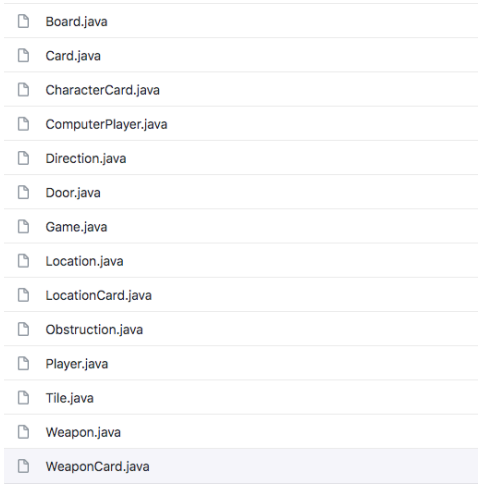
For game play we followed the instructions outlined by the brief.
Any ambiguities were supplemented by the 'Official rules'.

2.2 Board The "Murder Madness" board consists of five estates located on a 24x24 square area.	Assumed the grid had the same structure as the image in the handout (no passages). >The passages were in the original game but not the specification.
2.3 Characters There are four characters, one of which (randomly selected for each game play) is the murderer: • Lucilla • Bert • Maline • Percy Each player assumes the role of one of these characters. If there are less then four players, one character will not be controlled by a player.	Assumed if there are only three characters, the fourth will be played by the computer. >The text specified that one character would not be controlled by a human player.
The four grey areas are inaccessible to players and player cannot go through estate walls or use a square occupied by another character either. Estates do not have internal structure; they count as a single square.	Assumed that if a player is standing in front of a door, a player within a room cannot exit through that door. >A player cannot use a tile occupied by another player.
2. One player is selected to start at random.	Assume the player who is playing Lucilla starts each round. >This supports the testing process, allowing us to check if elements worked correctly.
When a guess is made, each player – using the order Lucilla, Bert, Malina, Percy, but starting one player after the player making the guess, and cycling around – attempts to refute the guess. A guess is refuted by producing a card that matches one of the suggested murder circumstances (such a card cannot be part of the solution and hence refutes the guess). A refutation card is only revealed to the player that made the guess and if a refutation can be made, it must be made. • Lucilla • Bert • Maline • Percy	Assumed that the character's name was Malina. >Malina is used more in the brief.
2.6 User Interface Implement an object-oriented program for playing the "Murder Madness" game. The game interface should be simple and must be text-based. Only text-based input and output is permitted, i.e., all input/output must occur via `System.in` and `System.out`. You may, of course, use the standard output to print a text character-based 2D presentation of the game board using text, location of characters, etc.	We decided to visually display the player's hand and 'seen' cards. >This was unnecessary but allowed for better game play.
squares. Diagonal movement is not allowed and no square can be used twice during one turn. When a player enters a estate, they do not need to use any remaining moves they have left. They may then hypothesise about the murder circumstances by making a *guess* which comprises the estate they are in, a character and a weapon.	When players enter an estate, they must immediately make a guess; they may not use additional moves. >Made coding easier.
A *solve attempt* comprises a character, a weapon, and an estate. If the solve attempt made by a player exactly matches the actual murder circumstances, the player wins, otherwise the player is excluded from making further guesses or solve attempts. This means the player cannot win the game anymore but will continue to refute guesses by others.	Incorrect assumption results in the player being removed from the game. Their cards are distributed to the other players. > Made coding easier.

CRC

We weren't taught how to use CRC cards until week 3, therefore we did not implement it in our initial planning process as we began coding early week 2 due to scheduling conflicts.
Instead, we created empty classes we needed for the game to function and collaboratively worked to comment on functions we believed necessary.
We found this to be an inefficient method of implementing the program, as we did not have a clear concept of how the game would operate.

After the initial coding process, CRC was implemented to identify missing methods and classes that ensured a clear overview of the responsibilities and relationships of each class.
> The aim of this was to reduce miscommunication; this ensured everyone had the same understanding of the game.
> This saved us time by making the code better encompass the specification, allowing for a more collaborative coding environment.
>After our initial coding process, using CRC cards revealed more classes and methods that we needed.



Class: Card

Responsibility	Collaborator
Interface for card types	
Forces a toString method	

Class: CharacterCard

Responsibility	Collaborator
Knows the character name	
Knows if a character card is equivalent to another card	

Class: LocationCard

Responsibility	Collaborator
Knows the location name	
Knows if a location card is equivalent to another card	

Class: WeaponCard

Responsibility	Collaborator
Knows the weapon name	
Knows if a weapon card is equivalent to another card	

Class: Player

Responsibility	Collaborator
Set the starting tile of the player	Board
Know the name of character that player is playing	
Add card to hand	Card
Add cards seen to hand	Card
Knows current tile it is on	Tile
Know if card in its hand is equivalent to another card	Card
Superclass of computer player	
Set the tile of the character	Tile

Class: ComputerPlayer

Responsibility	Collaborator
Set the starting tile of the player	Board
Know the name of character that the player is playing	
Make a guess of 3 cards	Board

Class: Door

Responsibility	Collaborator
Knows location, direction and position of door	Location, Tile
Lets player enter room	Player, Location
Lets player leave room	Location, Player, Tile, Board

Class: Tile

Responsibility	Collaborator
Knows position of tile and if there is a player on the tile	Player
Set position of tile to a point	
Set player on tile	Player

Class: Location

Responsibility	Collaborator
Stores players	Player
Knows how much space is taken up by this location	
Knows the name of the location	
Add players into this location	Player
Remove players from this location	Player
Get the top left position of this location	
Get the bottom right position of this location	

Class: Weapon

Responsibility	Collaborator
Weapon will hold a Tile on board	Tile
Holds weapon name and symbol	

Class: Board

Responsibility	Collaborator
Holds tiles on the board	Tile
Holds the locations in the game	Location
Hold the doors	Door
Holds the obstructions	Obstructions
Generates tiles	Tile
Reset valid move spaces	Tile
Add Player	Player
Check if a move made by player is valid	Player, Board
Knows where players are on the board	Player, Location
Moves Player	Player, Direction(enum)
Creates visual representation of board	Player, Location
Generate new position given direction	

Class: Game

Responsibility	Collaborator
Holds evidence folder	Card
Holds all players in game	Player
Knows when end of game condition is met	Card
Create a new game object and populate the lists	Board, Player, ComputerPlayer, Card
Run the program	Board
Lets player input and make guess	Card, Player, ComputerPlayer
Let player input accusation and make accusation	Card, Player
Moves player around board in desired direction for sum of dice	Player, ComputerPlayer
Knows the number of human players participating - determines need for computer player	Player, ComputerPlayer
populate the card sets and deal out the cards to all the players	Card, all Card subclasses, Player

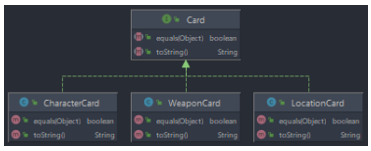
Class: Obstruction

Responsibility	Collaborator
Holds two points	
Knows if a point is an obstruction	

Logic

1. Cards Package

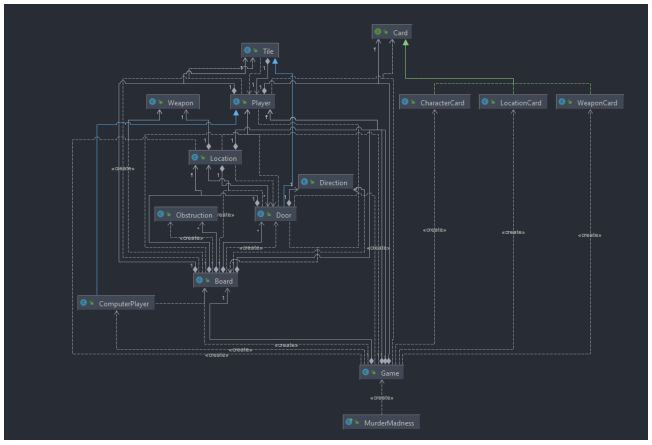
>Card interface allowed different types of cards to be stored in the same ArrayList; this allow for better functionality of code.
>Used as the main object within the game, held by a player and determines the end condition of the game.



2. Logic Package

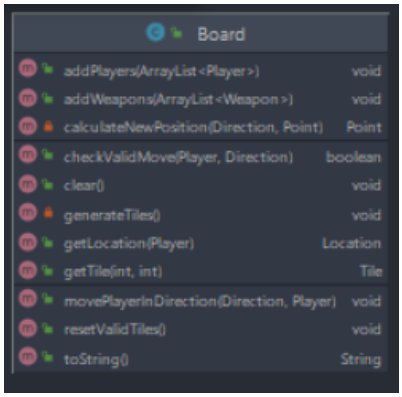
Game

>The Game class is made up of Player, ComputerPlayer, Card, Location.
>The Game class handles setup and acts to load the board and populate the other classes with data (inside the constructor) through a variety of helper setup methods.
>Game class has main cycling logic. The main run method consists of a loop, if a player is not in a room the player is able to choose between the action ‘Move’ or ‘Accuse’ else they also have the option to ‘Guess’.
>This run() method then calls upon helper methods based on the actions selected by players. The method then redraws the board to visually represent the action.
>It then sets the next active player and ensures there are still active human players in the game.



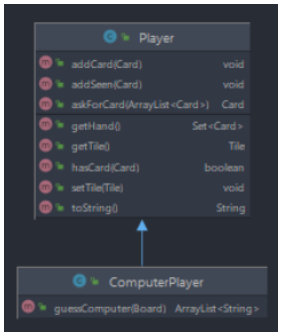
Board

>The board is made up of an array of tiles, locations, doors and obstructions.
>Used as a visual representation of the game to support efficient gameplay.



3. Players Package

>This package holds the players used within the game, allows for a clear visualisation of player's functionality.
>ComputerPlayer extends Player allowing for it to use player's methods.



Code and Design Contribution
We decided to use the 'CodeWithMe' function in IntelliJ for the majority of our project, this was to support collaborative work between classes and method to allow team members to assist in and alter methods as required.

Design contributions indicated by RED
Code contribution indicated by BLUE
Combination indicated by GREEN

General Contributions
-CRC Cards
-Familiarising the team with Github
-Program layout (Created basic classes)

Players Package
This package was my assigned section of the project. The code, commenting and design in this section was completed by me.

Player
> Fields, Constructors, Getters, Setters.

ComputerPlayer
>Fields, Guess method.

Cards Package
Helped with commenting, design and coding.

CharacterCard
>Fields and Constructor.

LocationCard
>Fields and Constructor.

WeaponCard
>Fields and Constructor.

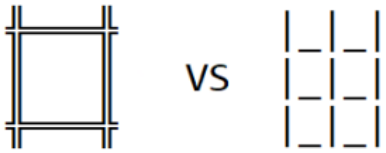
Logic Package
Helped with overall conceptual design of Game and Board, but made direct contributions to listed methods. Also assisted in converting all code from Java 16 (using lambda) to Java 11(using old-style).

Tile
>Fields, Constructors.

Location
>Fields, Constructors, Getters, Setters.

Game
Helped design any methods involving the Player package
E.g
>getNumberOfPlayers()
>setNextPlayer()
>playerSetup(int amountOfPlayers)

Challenges and Improvements
>Initially, we had top and bottom border surrounding a cell. We had to remove this border as players had to scroll around the screen to see the entire board which was inappropriate. On the left, is the original design this is only one cell in a 5x5 character space. On the right, is the final design.



> We had conflicts in the team about the ComputerPlayer implementation. Certain team members wanted a more complex ComputerPlayer, others wanted a more simplified player. To make coding the Game class easier, we agreed on the simplified ComputerPlayer. This significantly reduced my ability to expand the ComputerPlayer class.

-We would make a more complex ComputerPlayer in future.

> We had a lack of planning. This resulted in an unfair distribution of the project as we did not understand the amount of coding required for each section. Additionally, we had expected to collaborate more to distribute work using the 'CodeWithMe' function on IntelliJ but found that this was not efficient on different networks (delay in code updates).

-We would plan using UML or CRC before beginning the project to better distribute work, learn how to use Github so we are not dependent on the 'CodeWithMe' function and better align our schedules.