The game is started by the Game class, which begins asks the user for the number of players. It then gets all the players to choose a player to play as, and sets their initial position to the start position of the character. It also sets up the cards of the game, by choosing one weapon, one room and one character, and making them the murder evidence. It then dishes out the remaining cards evenly to the players.

The Player class is what models a human player of the game. When the game class calls TakeTurn() on a player, the dice roll is simulated. The game then uses a modified A\* algorithm to find the number of moves it will take to reach each room, and then asks the user to choose a room to travel to based on this information. This search uses the PathfindFrame class as the ‘fringe’ node for the algorithm. If the player reaches a room before its moves run out, they are prompted to make a new guess, which creates a Guess object containing the character, weapon and room they guessed. The game then chooses a card (starting from the left player), and if one of the guessed cards is found in that players hand, it reveals the card to the player who guessed it. Once a player’s turn has ended, play moves onto the next player and they take their turn. This continues until the game is over.

The board is made up of a 2D array of BoardTile’s, one for each square on the board grid. There are two classes that extend BoardTile, Door and Room. A room on the board is represented by setting all of the spaces in the 2D array to be the same ‘Room’ object. On the square directly next to the door on the board, there is a ‘Door’ object in the 2D array. This object contains a link to the room it is next to, so when a player wishes to move into a room, they must travel through the Door to get there.

Objects that get placed on the board (character pieces and weapons) are modeled using the GameObject abstract class. Character and Weapon classes both extend this class. Each weapon has a room field, which tells the game where the weapon is located. Each character has an X and Y position on the board.

Text output is controlled by the TextCluedoUI class (which implements the CludeoUI interface), which prints to System.out. The reason for using a class like this was to make it easy to modify the program in the future, particularly when adding the GUI. The rest of the programs implementation will be largely unchanged when using different methods of UI, as we will just need to create a different UI implementation. The CluedoTextUI asks the user for input, and then returns an object that matches this input, depending on what the output asked the user for.