Design Choice Explanation

**CommandWords:**

A final array of Strings is used to store the valid commands that the program will recognize. An array is used for this because the number of valid commands will not change hence the data structure does not need to have malleable size. Two ArrayLists is used to store the rabbit and fox game piece colours.

An ArrayList is used because there is a variety of challenges, which all have a different amount of each game piece. This results in every game having a different amount of rabbit and fox game piece colours, hence an ArrayList is chosen as it has a malleable size.

**Game:**

A HashMap is used to store all the fox and rabbit game pieces, which are the game pieces that the user controls. A HashMap is used for easy and direct access to each game piece specified by the application user, which is more efficient than using a data structure that requires iteration every time you want a specific object. A user command is taken as a String, which is then split into a String array by parseCommand(). Within that String array, the Strings at index 1 and index 2 are added together with a space in between the words, which is then used as the key to access the specified game piece within the HashMap.

The parseCommand() method takes in a command String and checks if the command is a legal command. It first splits the command string into a String array. The command is legal if the command String was split into a String array that has a length of 4. Depending on the first word String of the String array, four courses of action can be taken. If the first word String is “move”, it combines the word Strings at indexes 1 and 2 with a space in between and sends it along with the word String to the game Board to process which game piece is being moved and what direction it moves in. If the first word String is “rules”, calls the printRules() method, which prints the rules of the game. If the first word String is “commands”, it calls the printCommands() method, which prints the command words available, and examples on how to use it. If the first word String is “quit”, it sets the quitGame instance variable to true, which is used to end the application.

The play() method uses a scanner to read the user commands. It first calls the createBoard() method, which initializes all the game pieces required for the specific challenge noted by the integer parameter passed to createBoard(). The printRules() and printCommands() are then called to show the rules and the available commands. A nested do-while loop is used to run the game. The inner do-while loop runs until the user inputs a legal command. The outer do-while loop prints a text-based version of the board with the updated board before every move and runs until the win condition is satisfied or the user decides to quit.

**Board:**

The board() method is used to create the 5 by 5 board by using the board variable (of type slot). The addPiece() method is used to add a piece onto the board. Both of these methods are used by the createBoard() method in the game class to create a board based on which challenge the user wants to play.

The move() method in this class allows the user to make a move on the board. It uses an if statement to check whether the slot given is a rabbit or a fox. If it is a rabbit, there are 4 possible cases (up/down/right/left). Inside each case an if statement is used to check whether the rabbit can jump in the specified direction and if it can jump it also updates the rabbit’s new position. If the slot is a fox, there are also 4 possible cases (up/down/right/left). For the up/down cases an if statement is used to check that the fox is indeed vertical(return’s true) which means that it can move up and down. The same test is performed for right/left except the program checks if the fox is horizontal (return’s true if this is the case). There are 2 methods that are used inside the move function: canHop() and canSlide(). The canHop() method uses a switch and case to check if the rabbit can hop over a slot. It returns the number of slots that the rabbit can jump over. The canSlide method uses an if statement to check if the fox can slide in the specified direction and returns true if the slot is empty (fox can move) or false if the slot if occupied (which means that the fox can not move).

The checkwin() method checks if the game has been won by checking all the holes and incrementing the count if a rabbit is in there. It returns true if count is equal to the number of rabbits in the game.

**Slot:**

The slot(method) is used to create a slot on the board by taking in the x and y coordinates. A slot represents one unit on the board. The board will be created using slots. It is also assigned a default name of SS. This class also has a setName() method. This method will be used to set the name of each object that is being placed in a slot (whether it is a mushroom, rabbit, or fox) or if the slot is a hole (HH). This naming system will help the user to identify which parts of the board contain a slot, an object, or a hole.

**Hole:**

This class inherits from Slot. An Array list is used to store the game piece type. This allows the hole to still have its labelling (HH) on the board but also allows the user to move game pieces into the hole without changing the actual hole. The hasGamePiece() and hasRabbit() methods are used to check whether the hole already has a piece inside. For the hasGamePiece() method it returns false if it is empty and true if there is an object in the list. For the hasRabbit() method it first uses one if statement to check if the list is empty or not and if the list is not empty it then uses another if statement to check whether the hole contains an instance of a rabbit. This will be important when the program needs to check whether all the rabbits are in holes.

**Mushroom:**

This class inherits from Slot. The Mushroom () method is used to create a mushroom on the board by taking in the position (x and y coordinates). It also sets the name of the mushroom as MM which is how it will be identified on the board. This will help the to identify whether the rabbit can jump over a specific piece or not.

**Rabbit:**

This class inherits from Slot. The rabbit () method is used to create a rabbit on the board by taking in the position (x and y coordinates). It also sets the name of the specific rabbit since some rabbits have different colors.

**Fox:**

This class inherits from Slot. The fox () method creates a fox on the board by taking in the position of the fox. There are 2 parts to the fox. The first part of the fox is represented using xPos and yPos and the second part is represented using xPos2 and yPos2. The fox method also uses an if statement to check whether the fox is being placed vertically. The if statement returns true if the y positions are equal and false otherwise which would mean that the fox is being placed horizontally. This will help to determine which direction the fox can be moved in. The method getVertical() which returns a boolean type is used to get the result (whether isVertical equals true or false).