Design Choice Explanation

Authors: Sudarsana Sandeep, Omar Elberougy

**Board:**

The board() method is used to create the 5 by 5 board and the method takes in a challenge level as a parameter. The addPiecesToBoard() method is used to add pieces onto the board. 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. The move() method uses the coordinates of the object to be moved as well as the coordinates of the destination, to find out the object to be moved and asks that object (Rabbit/Fox) to move to that position. The method checks whether the object is a fox or rabbit and delegates the actual movement to that object.

**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. The setPos() method sets the initial value of a game piece or slot object itself. Slot also has getter methods to get the (x, y) coordinate of a piece on the board.

**Hole:**

This class inherits from Slot. An Array list is used to store the game piece type within the hole. With the use of the Array list we can then determine if a hole contains a rabbit or not. 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. The getGamePiece() returns the piece inside the hole and removeGamePiece() method removes the piece from the hole.

**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). 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. The canHop() method checks if a rabbit can perform the hop movement using recursion. Rabbit has a move() method which actually performs the move operation on rabbits, so the board class just calls this method. Whereas in milestone 1 the board was performing the move for all game pieces. We have done this to increase cohesion between the classes because in milestone 1 some classes were not doing anything other than constructing the object and board was doing all the movement.

**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). The method setPos() sets the position of the fox on the board. The getTailX() method gets the location of the tail piece of the first fox and the getTailY() method gets the location of the tail of the second fox. The canSlide() method checks if the fox can perform the slide that was requested, this check is done using recursion. The move() method actually performs the move that was requested for the fox object. The move() method was added in the fox class to increase cohesion between classes, because in milestone 1 the board was performing the move operation for foxes.

**View:**

The View class contains the JFrame (implemented with ActionListener) and the Board, so it’s basically the view and controller of the MVC. Creating an instance of View starts another method (initialize) which initializes the GUI. The GUI has a CardLayout, with the startMenu() being the first card, followed by the levelSelect() as the second card, and the startLevel() as the third card. The ActionListener is used to determine which JButton is pressed (Start game, select level, etc.) and which object on the board is pressed (Rabbit, Fox, Hole, etc.) which are also JButtons. Clicking the command JButtons (Start,..) makes the corresponding action, while clicking an object JButton (Rabbit,..) returns that object’s position with respect to the board. Clicking a moveable object stores the objects coordinates, while clicking normal objects (Hole, Mushroom, empty Slot) stores the coordinates in different variables. This is done because we want different coordinates for moveable objects and destination objects. The actionPerformed() method is handling what happens when a button is pressed. After a game is won or the player wishes to play another game, the current Card holding the display of the board is removed and another one is added once a new game is started.