Design Decisions

JumpInModel:

JumpInModel was chosen to be the infrastructure of the game. JumpInModel is also responsible for printing out the string representation of the current board and the legend for the pieces. It is also responsible for determining the current game's state and returning an invalid move. Whenever the model is changed it automatically notifies the views.

JumpInView: The view uses the Java Swing library and displays all the content of the components. The view subscribes to the model and listens when the model is changed. Once it is changed, the view is notified and updates the buttons on the grid layout right away to reflect the model's changes. The View is the GUI which is also the only thing the user sees.

JumpInController: JumpInController was chosen to contain all the logic behind moving the pieces and playing the game since it is where the events occur between the user and the view. The controller is used to effectively communicate between the model and the view. The controller also updates the model when an event has occurred and the model then notifies the view. The controller handles action events happens with the buttons. The controller also checks if the moves for the foxHead and foxtail are valid as well as the rabbit and moves and selects them upon notice.

Board: The Board class contains a 2 dimensional Array of spaces to keep track of all the pieces currently on the board and their locations respectively. This class is responsible for determining and incrementing how many holes are filled on the board, and providing that information to JumpInModel so that JumpInModel can determine when the game is done. This class is also responsible for updating the game board to reflect the moves the user did

Space: Space is the parent class of all the individual pieces providing the methods to get the location of the pieces and set a new location for the pieces. Space also keeps track of which row and column each piece is in.

MoveableSpace: The Interface Moveable Space is implemented by Rabbit and FoxPart, the two pieces which the player can move, so that determining if the player has selected a valid piece to move is easier. The only method contained in moveableSpace is move which both moveable pieces must contain.

Mushroom: The mushroom class is simply an immobile obstacle for the rabbit to jump over, and thus once initialized doesn't need any further methods aside those inherited from Space Hole The hole class contains a Boolean 'isFilled' which is uses to determine if a rabbit can move into the hole or not. The Hole object has methods to determine whether the hole is filled or not and to fill the hole when needed, and inherits its locational methods from Space

EmptySpace: The EmptySpace class is a placeholder object for the locations on the board with no piece and after initialization acts solely as a destination for a move.

Rabbit: The Rabbit class contains the move function which is used to set the new position of the rabbit while updating the board state is done in Board and all the logic and determining whether a move is valid is done in JumpInModel

FoxPart: The Fox class is the most complicated piece as each fox takes up two spaces. In order to keep track of this, each FoxPart contains a second fox part such that the two parts stay together when moving. Each fox part also has two booleans, isVertical which is used in determining valid moves for the fox and its orientation and isHead which determines which of the two FoxParts is the head and which is the tail. The moveBoth method is used in order to make moving the fox simpler by moving both pieces at the same time instead of one followed by the other, however the movement of individual fox parts can also be done via the move method inherited by the interface

Written By: Benjamin Ransom

Revised by: Rafid Dewan (November 4, 2019)