**Design Decision Made  
Milestone 1**

By: Adam Prins 100879683

|  |  |
| --- | --- |
| **Design Decision** | **Why it Was Made** |
| Game | * Tile board[][]  This was designed to hold a collection of tiles that represent the game board. They instantiate the Tile interface so that the Tile type can easily be swapped out once we implement the GUI. It’s a 2-Dimensional array, which allows it to represent the row and column of the Tile. * Tile tileSelected This is the currently selected Tile. It is used when the player selects a Tile. This Tile and the new one will attempt to swap contents. See the method SelectTile() below for further detail. * Stack undoStack/redoStack Stacks that hold all the moves the player has made. Will be used later when undo and redo are implemented into the game. * ArrayList<bunny> buns An array of all the bunnies in the current puzzle. Allows for checking only this array, instead of iterating over the whole board to check the state of the game. * Game(int puzzleNum) Simple implementation that populates the initial board with a given puzzle number. * selectTile(Coord coord) This method attempts to select a tile. If there is no currently selected tile, this will select it if it has a Bunny or a Fox. If this is the same Tile as selected Tile, it deselects it. The last case is when the game will attempt to swap the contents of the Tiles, using trySwapPiece. This implementation is clunky for the text implementation of the game, but is designed for use of buttons in the GUI, where Tiles are selected sequentially. * setBoard This populates the board with an array of pieces, that will be created by static a static method in Puzzles. It has additional logic for placing the tail of the Fox, and will add any new Bunnies to the buns array. * trySwapPiece(Coord coord) This handles all of the tests for seeing if a move is valid. First, it asks the piece if this is a valid move for this piece type. Bunnies need to be inline in the same column or row, and be at least 2 Tiles away, since a bunny must jump over something. Foxes must be passed a tile that is in line with the head and the tail. Then it makes sure the destination is empty (or the same Fox piece that is trying to move).  Any failed call to swap will throw an exception, that will be caught by whatever runs the game. The error message of this exception can be used to print information about the attempt. Giving information to the player. This method is private since it should only be called from inside game, directly from selectTile. * swapPiece(Coord coord, Boolean changeStack) This will swap the Piece in the selected tile and the passed coordinate. The Boolean determines if the change is pushed to the undoStack or not. This is private, since only the game class can call it. It is called from trySwapPiece on it not finding any errors, and from the undo and redo functions. * endgame()  Determines the completion of the puzzle by iterating over the buns array, and determining if they are in the holes or not. * clearBoard() sets the selected tile to null, and removes the Pieces from all of the game tiles. This is called in setBoard before any of the new Pieces are allocated. * printPiece(Piece piece)  Prints to the console three charactures that represent the piece. * printGameBoard() prints the game board row by row. Calling printPiece for the contents of each Tile. * main Prints a welcome message. Initialized the opening puzzle. Sets up a character reader. Loops until the user compleats the last puzzle. |
| Tile Interface | * This Interface guarantees some methods that will be used by both TextTile and ButtonTile. Since ButtonTile needs to inherit from JButton, we decided to make Tile and interface that both classes can implement from. * It guarantees the following methods: getCoord(), getPiece(), setPiece(Piece piece), isEmpty, and removePiece() |
| TextTile | * This is the implementation of the tile for use in the text version of the game for Milestone 1. It instantiates the methods required in Tile. * Its Coord is public final since once a Tile is created, it will never change its X and Y values. |
| Coord | * Stores location data for all objects in the game. * Has public final X and Y values since individual coordinates will not be changed. To move a piece, you pass it the coordinates of its new position, likely pulled from the destination tile. * It cannot be instantiated with values that wouldn’t appear on the game board. * It is considered equal to another coord if they both have the same X and Y values. |
| Move | * Will be used to implement the undo and redo functions. It holds the destination coordinate as the coordNew, and the source of the move with coordOld. These are immutable, and so are public final. |
| Piece Interface | * Tiles have the ability to hold some Bunny, Mushroom, or Fox object, and will treat them all the same. Therefore they need to be extend a class, or implement an interface. |
| Mushroom | * Mushrooms are inert Pieces in JumpIn. They fail any test related to movement. |
| Bunny | * Bunny is allowed to move, and has two logical tests to make sure a move is valid. The new position must be inline with the current position, and the new position must be at least 2 Tiles away. This makes sure that there is the possibility that the bunny can jump over a piece. |
| Fox | * Fox is allowed to move, but has different restrictions. It is allowed to slide as long as the new position is in line with the Fox’s head and tail. * Fox is always controlled by the head position. Any selection of the fox will select the position of the head. This was done to reduce edge cases. Because of this, the Fox cannot try to move to row or column 4, as this will mean the tail is our of bounds of the board. |