Author: Aashna Narang

Design Decision

- JumpIn - Model
    - 2d array game board chosen because it perfectly resembles the real gameboard (5x5 square)
        - Limitations: fixed size, must know index to get a certain object,
        - These limitations aren't that impactful since it will always be a 5x5 grid and we won't need to get an object in this array without knowing the index often, so this was chosen
    - Play method - Milestone 1
        - Loops until there is a win. Continuously show possible animals to move, get the chosen animal, show the different options the user can legally move the animal, and move the animal there
        - We decided to show where the user can move to simplify the logic for checking whether or not the user made a legal move. Also reduced ambiguity for the user as it might not be clear where 0,0 and what a legal move is etc.
    - JumpIn keeps track of all JumpIn Listeners in an ArrayList —> whenever a move was made, call update function on listeners and they will update if needed
    - **Milestone 2 change -** the logic for playing the game has been split into various steps as the view needs to be updated when a piece has been selected and when it is moved.
    - **Milestone 3 change** - added logic to be able to undo and redo moves.
        - Use undoRedo functions, return a JumpInEvent. If undo, flip the points and process the command using the contents of the JumpInEvent. If redo, keep the points the way they are. Process command function already takes care of updating the view.
    - **Milestone 3 change -** fixed highlighting issues by unhighlighting the square after the move was validating but before officially moving the game piece.
    - **Milestone 3 change:** Added solver using depth first search method using recursion and backtracking as it seemed to be the easiest to implement.
- JumpInEvent
    - Object to store info to send to JumpInListeners when event created
    - Includes final location, the piece moved, holes on board, and source
    - **Milestone 2 change -** also include initial location since we don't know if movable animal object or view will be updated first, so the initial location in the animal may actually be the final location now.
    - **Milestone 3 change -** added a constructor to create an empty JumpInEvent object so it can be returned in undo/redo functions when there are no more JumpInEvents in the stack
- Undo / Redo - **Milestone 3**

    - A stack of JumpInEvents for undo and one for redo. There are methods to add JumpInEvents to the stack, when you undo a move, take off from the top of the stack and add it to the redo stack.
    - Decided to use stack because the way you insert and remove from a stack works perfectly for undoing and redoing moves. Made a stack of jumpinEvents because those are created once everything has been checked, it already has all the required info, can easily be used to pass to JumpInListeners
- JumpIn Listener
    - Interface that includes handleEvent method
    - Used so it's guaranteed that any listener will have this method implemented
- GameObject - Parent class
    - Include coordinate and name of object
    - Kept this because it can be used as parent class for rabbit and fox and be used to create mushroom objects
- Fox
    - Also include second coordinate since Fox takes up two spaces
        - Limitation: Hard to code the logic for moving the fox as there were 2 points to deal with as opposed to 1 point in the Rabbit class.
    - Through the implementation of helperDetermineOptions, I was able to deal with the fox's backward and forward efficiently by changing the point parameters.
    - Also by passing in lambda functions I was able to offset the location of the second point of the fox depending on the direction it was moving
- Rabbit
    - Separated fox and rabbit because they will both be jumpin listeners but mushrooms wont be and they will have their own unique handleEvent function
    - Rabbit also needs a status to check if it is in a hole or not, but fox doesn't
- Move Class
    - Created this to encapsulate all the information needed for the ___ to send to the jump in model to be able to move the object properly.
    - Includes all initial and final locations, chosen animal, and whether or not a move can be made
    - **Milestone 3 Change:** Changed the constructor of Move to take Point[] instead of multiple points to make it less coupled to whether a Fox was moving or rabbit was moving (same number of parameters).
- Parser
    - Asks user for input and sends the input to the jumpin model
    - **Milestone 2 change**: This class is not used for the GUI implementation, but was kept in case anyone wanted to play the textbased version
- JumpInView - View
    - Handles all GUI config and manipulation. Decided to put it in one class to follow the MVC pattern and to ensure that each class has their own unique purpose.
    - 2d array of GameButtons that resemble a game board put into grid array

        - Limitation: grid array doesn't allow you to pick specific location because it needs to look like a grid. This isn't a big deal for us because we need a grid
    - Decided to make it a jump in the listener so it can Handle event created by model and update the view accordingly
- JumpInController
    - Handles user input made from the mouse / clicking buttons
    - Implements handlers for moving an animal and selecting the undo and redo button
- MovableAnimal
    - Class which I used to give Rabbit and Fox the same methods and reduce copy paste code.
    - **Milestone 3 change:** Added getPosition() method and left subclasses to implement it depending on whether their location take up one space or 2 spaces
- GameButtons
    - Decided to use JButtons to make the GUI since it is easy to create mouse/action events and easy to add pictures to make a board that looks like the real game.
    - Can also change the background of certain buttons/spaces on the board and we wanted to highlight the legal moves to help the user
    - NOTE: WANT TO MAKE AN EASY AND HARD BUTTON IN FUTURE -> if click easy —> play game with highlighting legal moves and hard plays game without highlighting
- Board
    - Used in JumpInView to encapsulate the logic for initializing and designing buttons for each individual level
    - Reduce repeated code by creating functions to initialize each type of object and to set icons.

- Level Selector
    - Encapsulate code to initialize objects for each level in one class so each class doesn't have too many responsibilities
    - **Milestone 2 change**: Include getter functions to get the initial positions of each type of object to easily set the icons for each object in the GUI easily.
- Resources
    - Chose to create a static class to access each type of image icon to reduce repetition of initializing each one
    - Include resize functions for different images
    - Also, include a hashmap with rabbit images that map to the images of that rabbit in the hole and vice versa. **Did this so it is easier to change the icon when leaving or entering a hole.
        - Limitations: Need to put each pair of image icons into hashmap which is a bit of extra code, but this allows user to get the image by easily searching for the image they need, so this was chosen.

- Utility
    - Used it to store functions and variables that could be utilized in multiple classes
- Put all images under resource file for easy access
- Play
    - **Milestone 3 change:** Created play class to shorten code to start game, and efficiently update model view and controller when moving to the next level
-