Rabbits and Foxes!

John Breton
Dani Hashweh
Samuel Gamelin
Abdalla El Nakla
Mohamed Radwan

1.0 INTRODUCTION

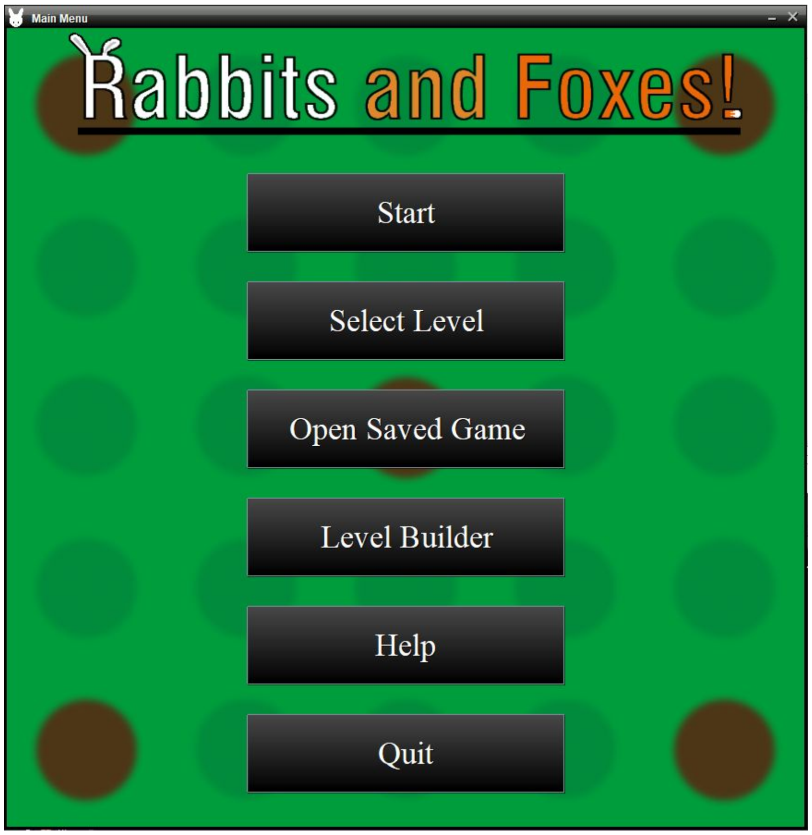
1.1 Team

The team is composed of 5 students studying Software Engineering at Carleton University. In no particular order, the names of these students are as follows:

- Mohamed Radwan
- Abdalla El Nakla
- John Breton
- Dani Hashweh
- Samuel Gamelin

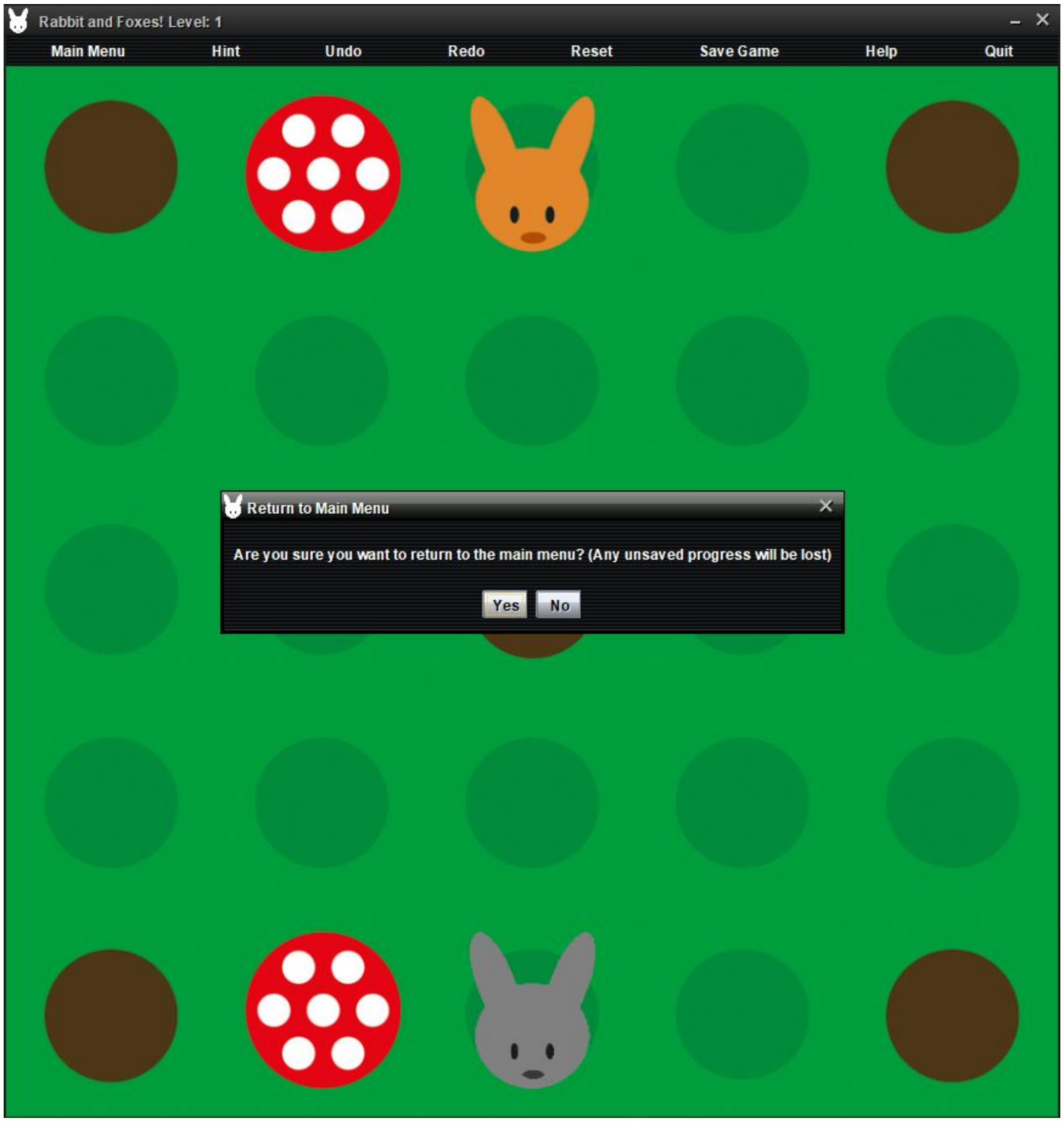
1.2 How to play Rabbit and Foxes!

Rabbits and Foxes is played via a GUI interface. Players are tasked with getting all of the rabbit pieces into brown holes. Rabbits can only jump over other pieces and must land in unoccupied holes. Upon launching the application, the user is greeted with the main menu, providing them the options to either start the game, access a level selection screen, open a saved game, access a level builder screen, request help, or quit the game.



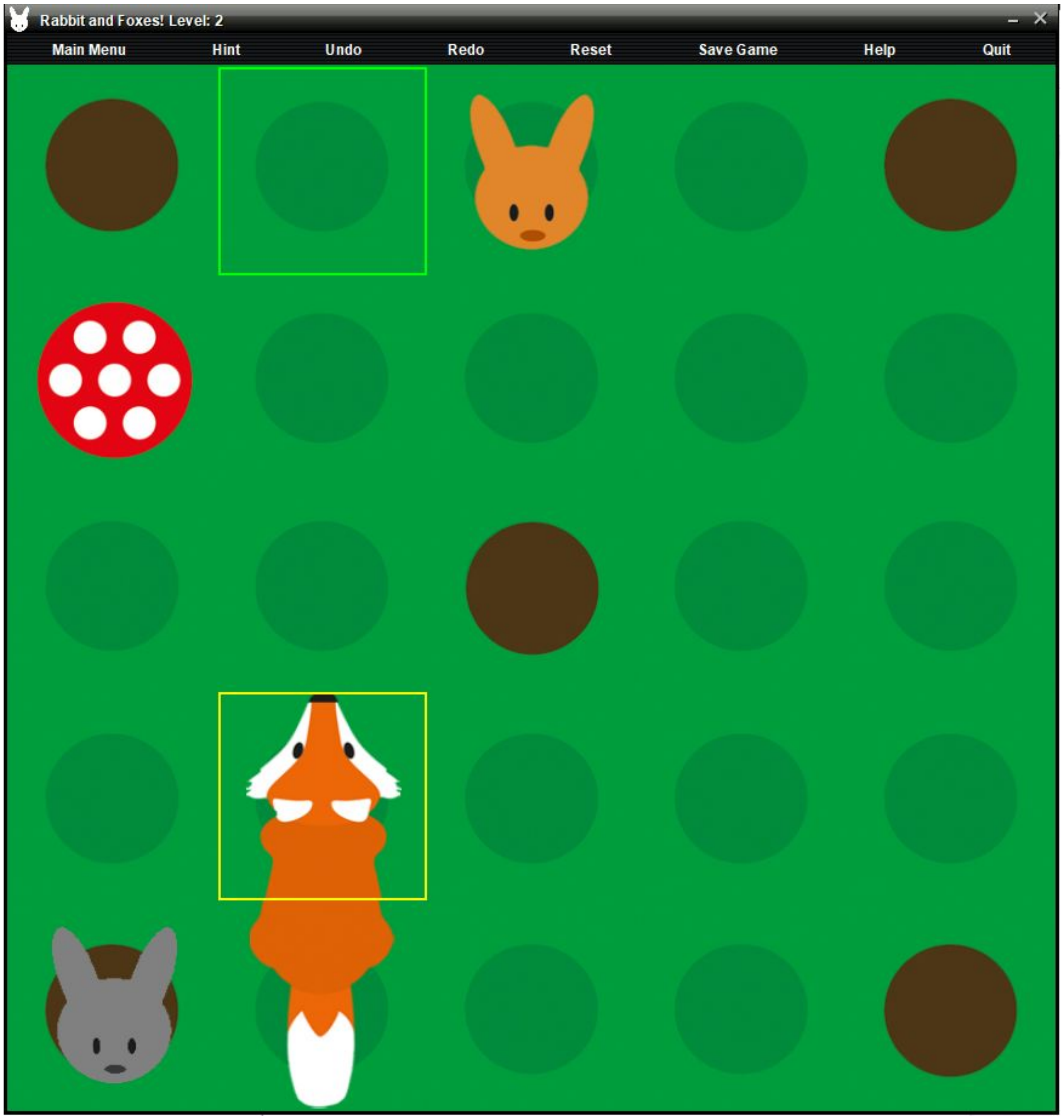
Date: December 2nd, 2019

Upon starting the game, the player has many options available to them. One of these options is to press either the r, u, or h keys in order to redo a move, undo a move, or request a hint, respectively. These functions may also be accessed by clicking their respective options on the top bar.

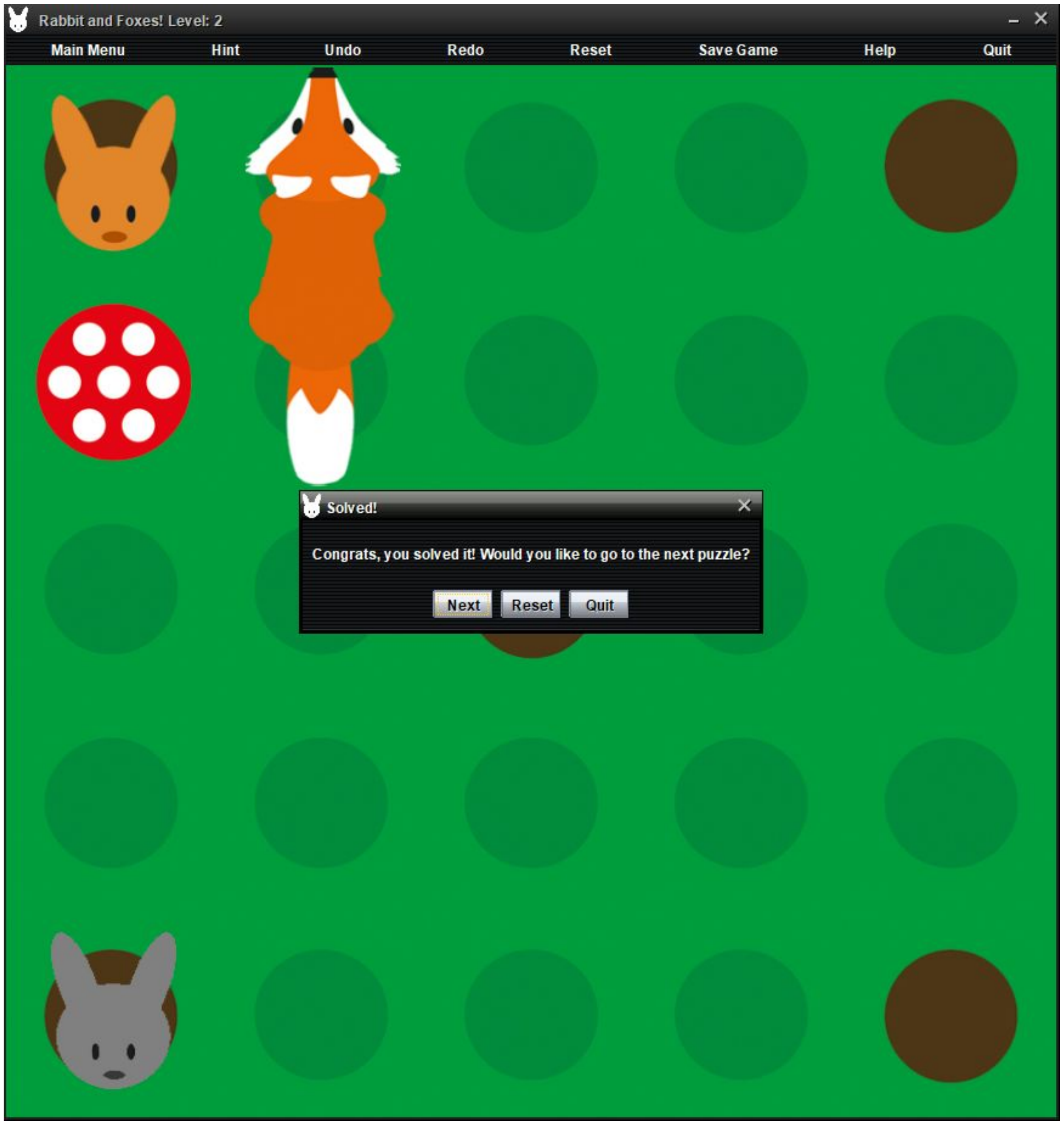


Are you sure you want to return to the main menu? (Any unsaved progress will be lost)

When the mouse hovers over a square, that square will have a border around it, indicating that it can be selected. Once the user selects a valid square, that square will be surrounded by a red border, indicating to the user their current selection for the initial position of the move. In order to clear that current selection, the user can hit the escape key, which will remove this red border. Once a second valid square is clicked on by the user (should it be invalid, an audio cue will play informing the player that the move was invalid), the move will be made and the initial red border will disappear. Another feature of the game is the hint system. Users can request a hint in order to assist them in winning the game. The user must simply press the ‘h’ key or click the hint button. Once the user has done either one of these, the next best move is highlighted, with the starting square being surrounded with a yellow border and the landing square surrounded with a green border, as shown below.



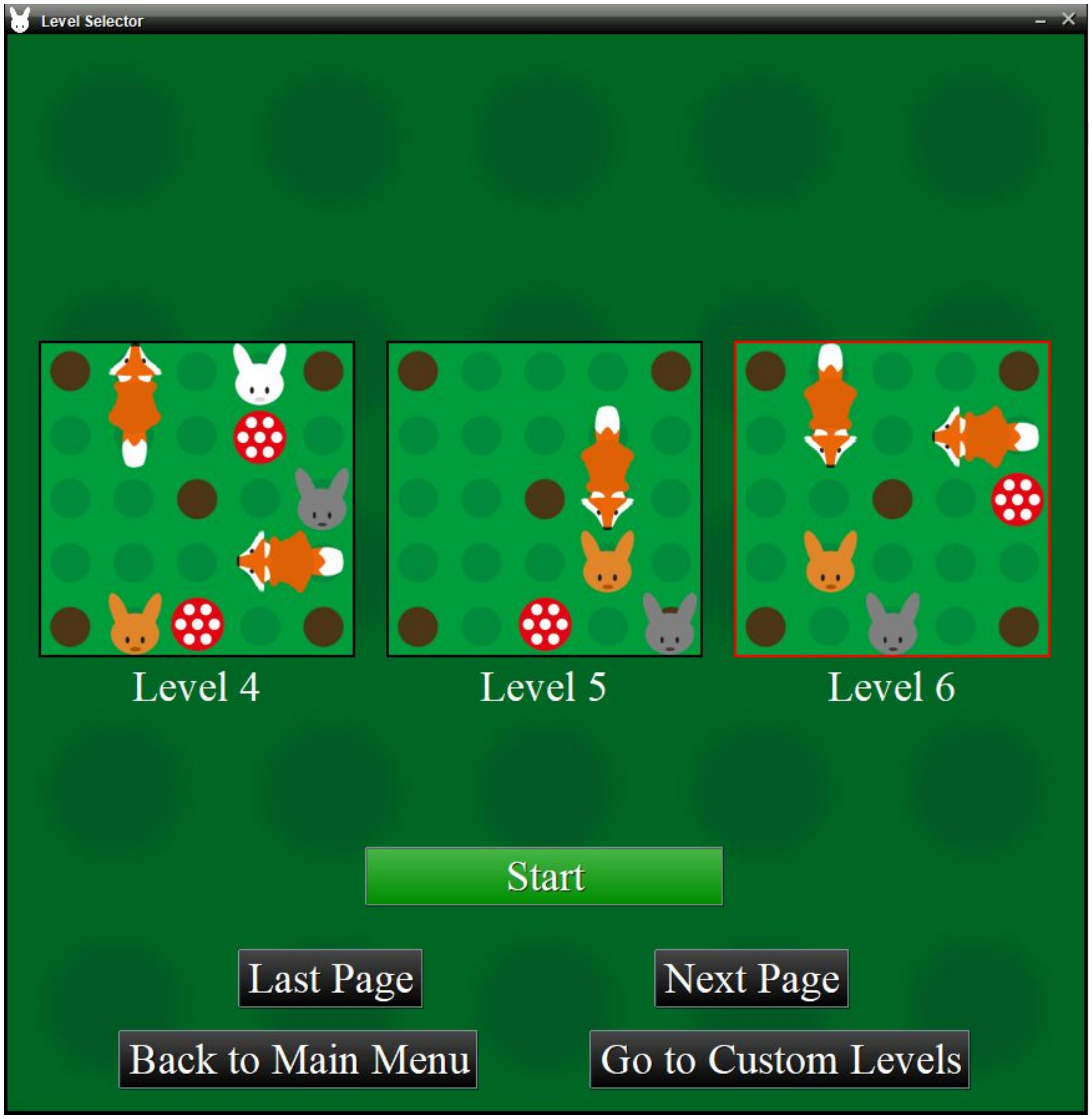
Once a level is won, the user will have the option to either move on to the next level, reset the current level, or quit the game.



Congrats, you solved it! Would you like to go to the next puzzle?

Another feature includes both the undo and redo buttons. If you make a move, you can click undo to undo the move. If the user would like to redo that move, just click redo. If the user undoes a move, then moves any piece there will be no move to redo.

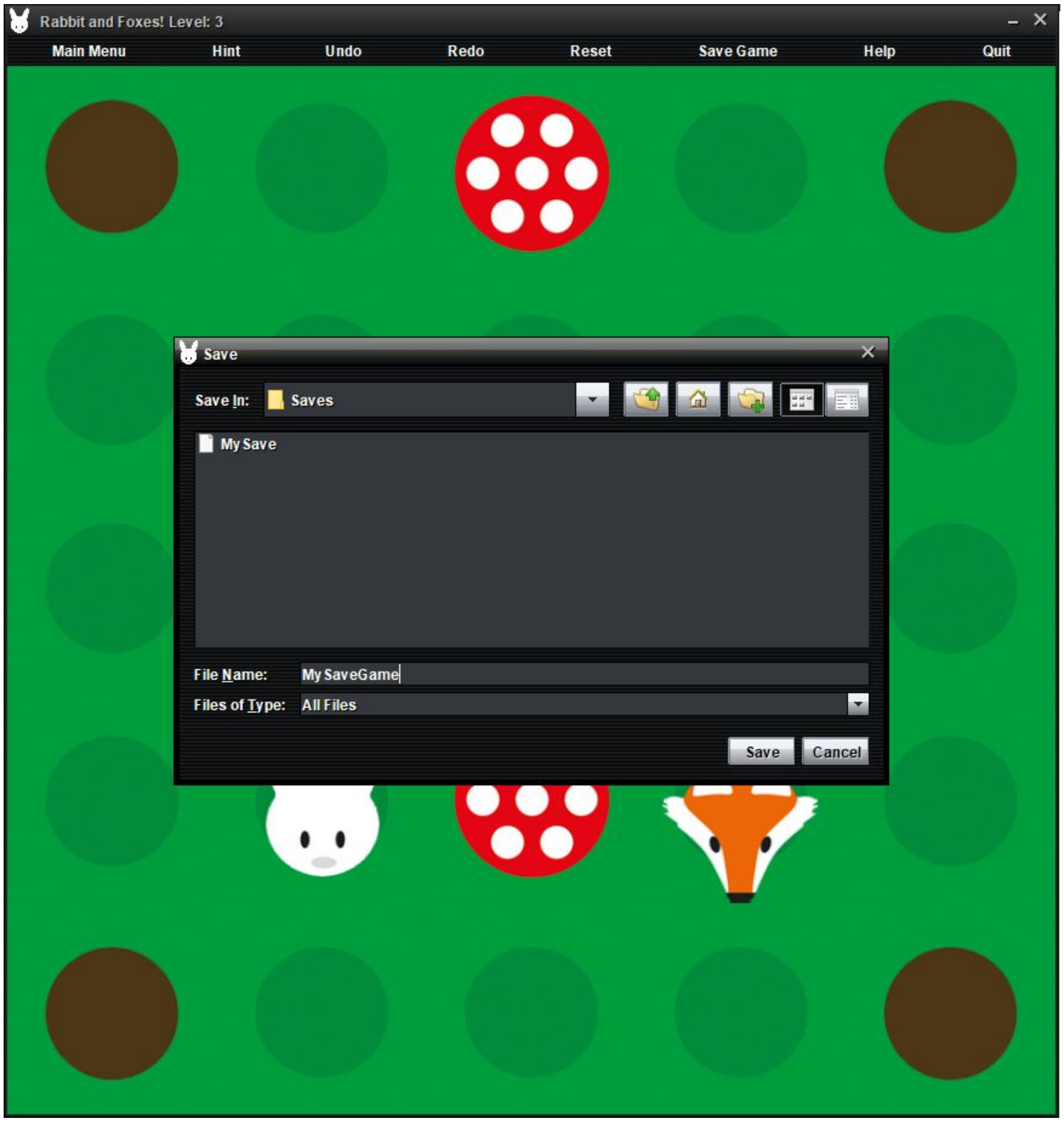
The user also has the ability to select the level with which they want to start the game by selecting the “Select Level” option from the main menu, as demonstrated below:



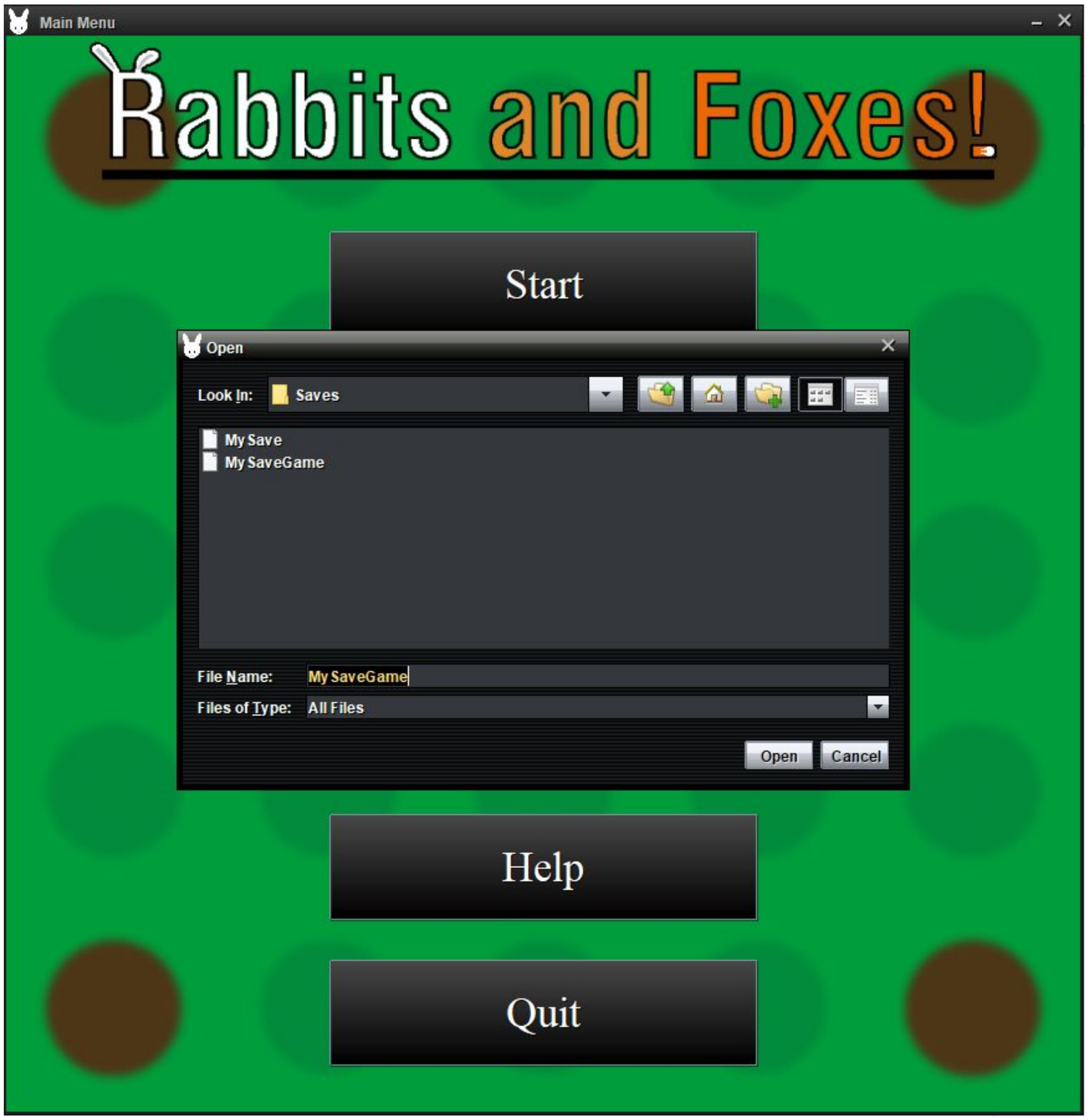
The user can also access any custom built levels via this screen. They also have the ability to delete any custom made levels if they so choose.

At this time, the game currently features 20 levels that gradually increase in difficulty. Once all of the levels are completed, the player has the option to either quit the game or return to the main menu.

In addition to this, the user also has the ability to save their progress as a separate file, which they can later reload (using the “Open Saved Game” option from the main menu screen) in order to resume their progress, as demonstrated below:



Date: December 2nd, 2019

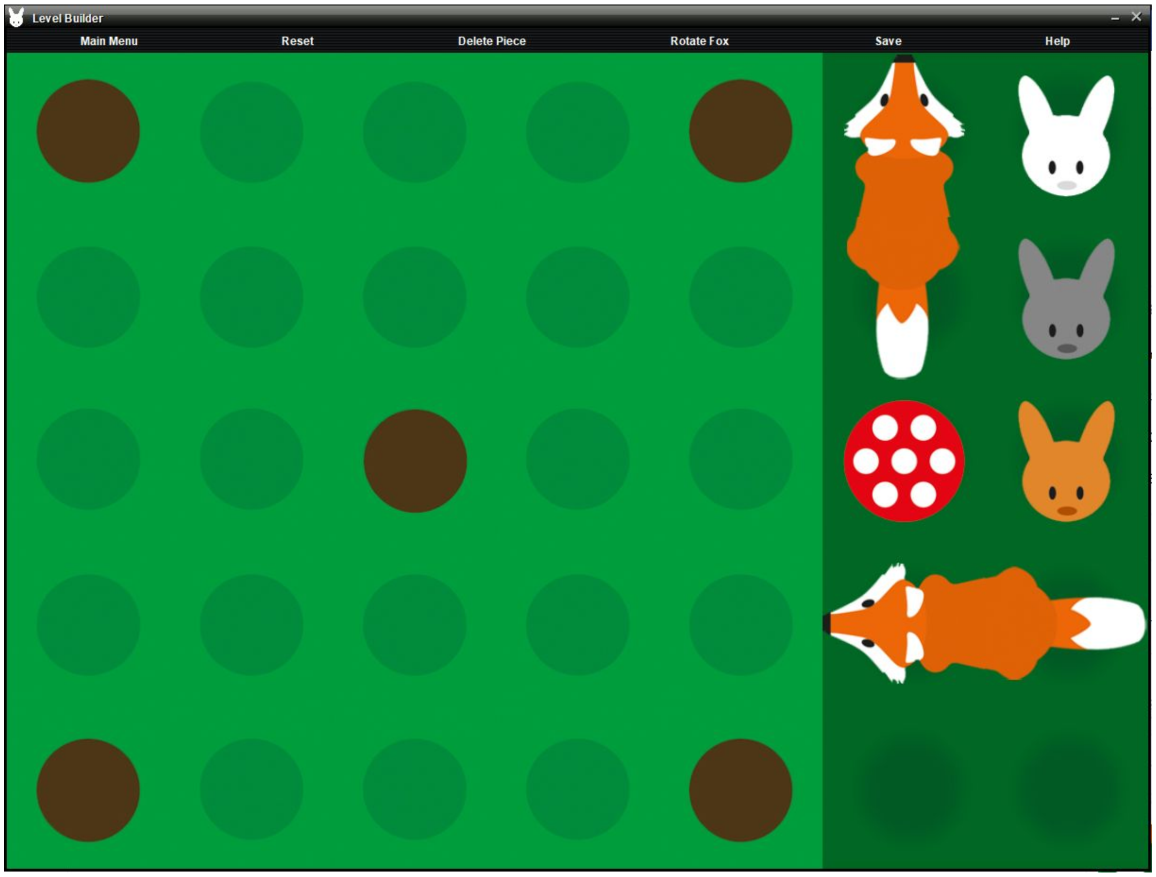


My Save
My SaveGame

File Name: My SaveGame
Files of Type: All Files

Date: December 2nd, 2019

Another important feature of the game is the level builder. It allows the user to build their own custom levels, which they can save and then play by selecting them under customer levels in the previously mentioned level selector. A view of the level builder is given below:



When the user saves their newly-created level, should it be unsolvable, the user will be notified of this. In addition, if the name given by the user for their level is one that has already been used for previous levels, or contains numbers, then the user will be notified of this and prompted for another name.

2.0 DEVELOPMENT CYCLE

2.1 Contribution

At the beginning of the milestone, we have divided the work evenly among the group. During our two weekly standup meetings, the team would go over everyone's completed tasks and if there any issues they have been facing in which they require the team's assistance. This ensured that the team was following our schedule and deadlines that we have set throughout the week.

As a team, we thrive to learn more about technology. Therefore, we would all contribute as much as possible and explain our decisions so that everyone understands and is on the same page.

After small updates, the team meets for a code review. This is done to ensure all team members understand all the new concepts that have been implemented. The team also takes that time to refactor the code in order to improve the code being pushed to consumers.

2.2 Methodologies

Development began in the first week as the team had three 2 hour meetings to discuss how the UML will be designed. There were many drafts of the UML but as a team, we decided the current version is the best designed and contains the best software design practices. Tasks for the project were also discussed and assigned as "Issues" on GitHub.

The team used a scrum and agile type of project management. This ensured that the team worked in the most efficient way possible. The scrum ensured the team members were never lost or stuck on a problem for a long period of time, this also ensured that the team was always up-to-date on the other team members' tasks. Using GitHub for agile development ensured that all commits could be reviewed and changed by other team members if needed.

In the second week, the team began working on their tasks assigned to GitHub. Throughout this week, the team began building the model for Rabbits and Foxes. The fully playable text-based implementation was also completed this week.

As for the coding, the team discussed their suggestions to other team members if they felt that their class was needing any adjustments or clean-ups. Even though team members were assigned specific “classes”, there were times where the member would have to go into another class to implement another method that was needed. For the future, tasks will be assigned to multiple team members as this would be easier to manage work among the group. The UML was updated throughout as the needs of the program evolved.

As the next milestone approached, the primary focus shifted to ensure that all components required were up to date and complete. Once the first milestone was completed, development began immediately on milestone 2. This included creating a complete GUI representation of the game. The team once again adopted the scrum and agile methodology to ensure that work was done efficiently.

This pattern has continued throughout the development of milestones 3 and 4.

3.0 PRODUCT DESIGN

3.1 Design Pattern

This project follows the MVC (model-view-controller) pattern. For milestone 1, the classes and UML design reflect the “model” of said design pattern. As a team, we have implemented the underlying logic to have a text-based implementation of the game. For milestone 2, a complete GUI-based representation of the game was completed, which meant incorporating the “view” and “controller” classes to complete the MVC pattern. For milestone 3, few classes were added to the project, save for those associated with the Solver. These classes did not fall under the MVC pattern and served more as utility classes, which is where the name of their package is derived from. For milestone 4, the GUI component of the game was broken down into multiple classes, so as to cleanly separate different screens of the game. Many more utility classes and methods were added, specifically those related to saving and loading user-created levels and user-saved level files.

3.2 Design

3.2.1 UML

See the pictures included in the submission.

3.2.2 Class Design Decisions

- GameController - Mohamed Radwan, Samuel Gamelin, Dani Hashweh, Abdalla El Nakla
    - This class is used to connect the View and the Model. This class takes in user input from the view then registers the click. Once the user makes a valid click(clicked on a movable piece) the click is then added to a list. The list is then used to make a move on the board when the user clicks another valid click.
    - This class is also responsible for resetting the board. This is done through the creation of a new board object and then returning it to the view. This is done so that the view and controller will have the same board.
    - The undo/redo methods are included within this class. This is done by using two stacks, one for undo and one for redo. Once the buttons undo is clicked, it calls the function and checks if the stack is empty. If it is empty, it will return false as there are no moves to

undo. If it was not empty, it will pop the most recent move, and add it into the redo stack. Then it will call board.move with the pieces with popped move reversed to “undo” the move. For redo it is similar, except board.move will not pass in a popped reverse move . It will just use the regular popped in move.

- GameView - Dani Hashweh, John Breton, Samuel Gamelin, Abdalla El Nakla, Mohamed Radwan
    - This class is the view of the entire application. In this class, the main menu GUI is created and so is the GUI of the game.
    - The main menu is a JFrame which is setup with a box layout, the game is a JFrame with the grid layout setup, and the level selector form is a JFrame with the border layout.
    - In the main menu the user can start the game at level 1, or can select the level they want to play.
    - In the help dialog, the user can also select if they want the “Possible Moves” feature on or off.
    - The level is shown at as the title of the jframe once the game starts.
    - All the button handlers are in this GameView class.
    - When it comes to setting up the game, the GameView class looks at a board object and takes the setup of the board that was set in the Board class, and represents it in GUI form.
    - Once a piece is pressed and “moved” the view will notify the controller about this move, and the controller will deal with this.
    - To ensure that the GUI is consistent across all platforms, the setLookAndFeel() method was used. The display is also consistent across all computers, and the game frame resolution is scaled to the user’s computer resolution.
    - GameView makes use of Clips for audio processing. Currently, there are two audio clips implemented: one that plays once an invalid move is attempted by the user and one that plays when the user wins a level.
    - JButtons are registered as MouseListeners to determine where the cursor is currently located. This was done to draw borders around the JButtons as the cursor hovers over them, as well as drawing a selected border to indicate to the user which JButton is currently selected for the move.
    - Keybindings were included to access the JMenuItems, as well as to clear the currently selected button.

    - Once the puzzle is solved, a prompt is forced on the user, removing their ability to interact with the game. This removes the need to disable the JButtons.
    - The GameView class listens to the Board for any changes and will update the GUI accordingly once any change occurs.
    - Clicks that result in Moves are registered to the GameController, which in turn will inform the Model (which is the Board), which the GameView will then use to update the GameView accordingly. This is the standard MVC design pattern.
- Resources - Samuel Gamelin, John Breton
    - This final, uninstantiable class was designed to provide a way for the view-related classes to be able to easily access resources (such as pictures of the board and pieces and audio clips).
    - Additionally, Resources makes use of the Gson library to read in levels for the player to progress through as well as custom levels that the user created with the level builder. This dependency is specified in the pom.xml file.
    - Facilities in this class to add and remove user levels are also provided.
- BoardListener - Samuel Gamelin
    - This interface outlines the behaviour that a board listener should have, namely one method to handle updating its representation whenever the state of the board changes.
- Board - Samuel Gamelin, Dani Hashweh, Abdalla El Nakla, John Breton
    - The Board class provides a way to manage all the tiles that comprise the game. A two-dimensional array was chosen as the structure to maintain all tiles, as this structure inherently includes a position on the board (i.e. the 2D array can be indexed using an x-y based coordinate system).
    - Board also maintains a list of listeners that are notified whenever the state of the board changes (i.e. a move is made successfully)
    - Board also has the responsibility of determining if it is in a winning state and to provide utility methods for determining whether a certain position is occupied and to be able to remove and set a piece at any position.
    - New to milestone 3, Board now has a static factory method, along with an improved toString method. Board delegates the work associated with generating an appropriate String to the Tile class,

which in turn delegates to the Piece occupying each Tile. This produces a String that can be saved to a JSONObject. This allows for level configurations to be saved, so that the player has multiple levels to progress through. The static factory method takes in this produced Board String to recreate a new Board that can be played by the user.

- Tile - Dani Hashweh
    - The Tile class was chosen to represent the tiles the pieces will be placed on - furthermore, the Board class will delegate from this class to keep track of piece positions and tile colours.
    - Board will use the Tile class like the following. For example, when a user wants to move a Rabbit, the Board will use the isOccupied() method in Tile to check if the following move is valid. If the move is valid, the Board class will use the placePiece() method to ensure that the new position is set as occupied.
- Piece - Samuel Gamelin, Abdalla El Nakla
    - The Piece class was chosen to be an abstract class as all the pieces of the game share common attributes and functionalities.
    - For example, every piece has a piece type and has a method to retrieve that piece type.
    - However, methods such as determining if a certain move (from one position to another) is valid (which is a method whose implementation will not be the same for every piece type) was made abstract.
- Fox - Abdalla El Nakla, Samuel Gamelin, John Breton
    - Fox extends the abstract class Piece.
    - Fox has a direction (either left, right, up, or down) indicating which way it is facing. Fox also has a type (either head or tail) identify which part of the fox it represents)
    - Foxes have also gained the ability to keep track of their otherHalf. This was done to refactor the Fox class to make it easier to follow. The logic for validatePath remains obtuse, however, this will be worked on further in future milestones.
    - The use of a private Constructor is rare, however in this case, it allows for quick setup of Fox Pieces. Once a level editor is implemented, this will allow for quick fox placement on the Board, as all that needs to be specified is the location of the head and the direction in which the Fox is facing.

    - Additionally, since there can be up to two foxes on a Board, a boolean field “id” was added to differentiate between the foxes. This was critical in determining how to move foxes on the board. For example, if a fox was beside another fox, without having this id the wrong fox piece may have been grabbed, causing a number of errors. Ensuring we are moving the fox pieces of the same id fixed this issue.
    - The move method (inherited from Piece) validates the provided move and moves the appropriate Fox head/tail pieces if possible.
    - New to milestone 3, Fox now has a static factory method. This was done to build foxes based on stored Strings located in a JSONObject. This made it possible to save level configurations, which in turn allowed for multiple levels to be created and progressed through.
- Mushroom - Abdalla El Nakla, Samuel Gamelin
    - Mushrooms extends the abstract class Piece.
    - This is an obstacle that has no movement, thus its inherited method move returns false.
- Rabbit - Abdalla El Nakla, Samuel Gamelin, Mohamed Radwan, John Breton
    - Rabbit extends the abstract Class Piece.
    - Unlike fox, It can move either both vertically or horizontally, while fox can only move in the direction it is initialized in.
    - Rabbit uses the enum class RabbitColour to represent its colour this helps the game distinguish between the two colours present for rabbits (Brown and white).
    - New to milestone 3, Rabbit now contains a static factory method to construct a rabbit based on stored Strings located in a JSONObject. This made it possible to save level configurations while allowing for appropriate delegation to the Piece subclasses to determine how they should be built based on their toString methods.
- Move - Mohamed Radwan, Samuel Gamelin
    - The Move class neatly packages and represents a move made in the game, with starting and ending positions, along with additional information such as the direction of movement and travel distance (in tile units).
    - This avoids repetition across classes where moves are commonly used and allow logic relating to moves to remain inside this class.

    - As Move objects are intended to be one-time use throwaway objects, the class is declared as final (it cannot be extended) and its fields are also declared as final (the instance variables of a move cannot be reassigned).
- Solver - Mohamed Radwan, Samuel Gamelin
    - The Solver class is responsible for providing the next best move given a Board object. To accomplish this, it uses three static utility methods. The first one, which is a public method, returns the next best move given a Board object. This is the method that is used by the GameController in the context of the game.
    - The second method is a private method that is used to “flatten” the list of winning moves: it removes any unwanted nodes containing repeated fox moves in succession.
    - The third method is a private method that is responsible for performing a breadth-first search on a root node and returning the list of moves that make up the winning path, should there be a winning path. If that is not the case, an empty list is returned.
    - The Solver class also keeps track of the most recently generated list of winning moves, to more efficiently provide the user with the next best move should they remain on the winning path.
- Node - Mohamed Radwan, Samuel Gamelin
    - The Node class represents a node in the game tree/graph of the Rabbits and Foxes game. A node contains a board (that is, the entire state of the board), and has methods to obtain all of its children along and to determine the required Move object to move to another node. These methods prove useful when performing a search on a graph of nodes.
- GUIUtilities - Samuel Gamelin, Dani Hashweh, John Breton, Mohamed Radwan
    - This utility class provides functionality that is used by the GUI-based classes.
    - These functionalities include predefined GUI components, constants for display size and calculated font sizes, and methods to configure JFrames and other GUI components, binding keystrokes to components, and displaying pop-up dialogs.
- LevelBuilder - Abdalla El Nakla, Mohamed Radwan
    - This UI component is responsible for helping the user create a playable level. Using GUI, the user will be able to place pieces on the board and save/load them.

    - Once the user saves a level, the LevelBuilder checks if the level is solvable using the Solver class. If there is a solution for the level then the level will be saved as a custom level.
    - The LevelBuilder restricts the user to only 3 mushrooms, 3 rabbits, and 2 foxes as per the requirements of the game.
- LevelSelector - John Breton, Dani Hashweh
    - This class is responsible for helping the user select a level they want to play.
    - The user has the choice of selecting from the 20 default levels included with the game, or selecting one of the custom levels they have created using the LevelBuilder.
    - If no custom levels are detected, the user will be prompted once the “Go to Custom Levels” button is clicked.
    - Additionally, if the user has custom levels but decides that they want to delete one of their levels, this class adds that functionality via a delete level button.
    - Users are not able to delete any default levels.
    - Should the user delete their last custom level, they will be returned to the default level selection screen. Should they click the “Go to Custom Levels” button, they will be prompted to create custom levels.
    - This class displays a preview of what the level looks like on the game board, along with the name of the level. Users have the ability to name custom levels, which helps differentiate the levels further.
    - Unfortunately, if a user gives a custom level a name that is too long, the text will be cut off in the JTextPane. This was done to ensure that the formatting of the various components remains consistent, regardless of the text included in each JTextPane.
    - Once a level is selected and the user clicks the start button, the frame is disposed and an instance of GameView with the selected level is created.
- MainMenu - Samuel Gamelin, Dani Hashweh, John Breton
    - This class represents the entry point of the game, greeting the user with a set of options.
    - Whenever one of the options from “Start”, “Select Level”, or “Level Builder” are selected, the currently running MainMenu frame is disposed and an instance of the class corresponding to the selected option is dispatched for instantiation on the AWT event

dispatching thread. This also applies to the “Open Saved Game” option only after the user has selected a valid save file.

- The “Help” and “Quit” options bring up help and confirmation dialogs, respectively.

3.2.3 Sequence Diagram

See the picture included in the submission.