CSE1142
Term Project
BlindPipes: a JavaFX Game

CSE1142 Computer Programming II,
Spring 2020

Anılcan Erciyes, 150119520

Fatih Said Duran, 150119029

DATE SUBMITTED: 08 / 05 / 20

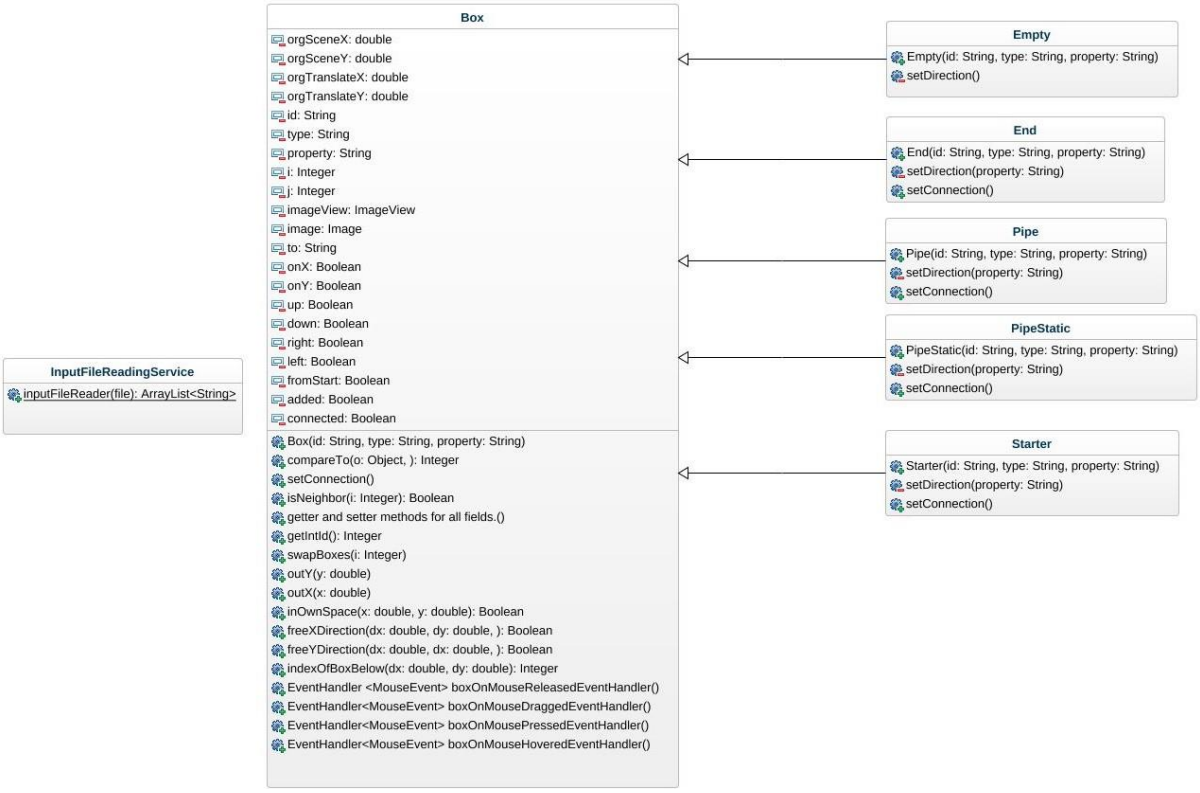
Problem Definition

In this project, we are expected to create a puzzle game which has five different levels, on a 4x4 playground made of boxes with various types. The characteristics of these nodes differ from each other: some of them are completely empty for free movement, whilst some others have a pipe inside them to roll the ball or some of them are even non-movable, in order to constitute a fixed location on the board. The main object for the player is to solve the puzzle by connecting all the boxes correctly, from the first sector to the last, and watch the ball rolling throughout the pipes until it reaches the "end" box.

In order to facilitate the project, before anything else, the gameplay functions have to be generated. These can be listed as the creation of the 4x4 array according to the input files, updating it both functionally and visually (using JavaFX framework) after each swapping operation (event-driven programming), checking whether if the current status that the user reached means level is completed or not and finally, designing the animation to be played. After a proper gameplay experience is estabilished, next objective is to manage the concordant flow of the programme by bridging different levels via a main menu or "next level" buttons. At that point, it should be stated that reaching the next level is impossible unless the current one is completed.

Implementation Details

uml class diagram of the project:



Before anything else, our code reads the input file via a Scanner object and save all the information needed in an arraylist of words, splitted by the sign ",". Then, this information is read in a for loop, three by three. The strings coming from the information array are directly sent to the constructors of new "box" objects. Switch-case structure is used here, in order to create the exact right sub-type of the box.

For instance, in case of reading "1, Starter, Vertical" line from the input file, a new Starter object is created with the id of 1 and the property of "vertical", thanks to this style of implementation.

Our programme uses the "GridPane" functionality from JavaFX. Since the boxes are represented with arrays, we thought the best way to place our objects is to use this grid system.

Hitherto, the creation of the 4x4 array according to the input file is done behind the curtains. However, in order to run a game, a software has to work on both functional and visual sides. In order to provide the visuality, our Parent root has an "updateGrid" method. This method uses a 4-in-4 for loop and asks for information from each cell. Then, according to the latest information, it paves the boxes from the beginning, visually.

With the all the boxes of the level being set, the user can click and drag pipes and empty boxes to assemble them connected from the start to the end. The mouse events only happen when the selected boxes are empty boxes with none property or are non-static pipes. The ImageView nodes will translate with the cursor dragging it, just horizontally or just vertically one space. After the mouse is released from the node, the dragged box will swap with the free space under where the mouse is released or will go back to where it was if the box wasn't dropped anywhere new. With the 'swap' the dropped box and free box will swap their Id values and layout coordinates in the grid, with these values changed when the grid is updated, the boxes will be swapped in the grid. updateGrid method clears and re-creates the newly changed grid.

So far, the creation of the board and dealing with the swapping operations are covered. Nevertheless, in a game, it is vital to have a rather clean menu navigation and controls for the flow of the scenario. In order to achieve these, we created a main menu. At the beginning, only Level 1 is accessible, the upcoming levels can only be reached if the current one is successfully completed. setDisable method is used for the initial states, with "true" as arguments. In addition, inside a level, "Next Level" buttons are also available, which means the user don't have to go back to the main menu each and every time. Similarly, this button is disabled unless the current objective is done.

At the moment, our programme can create the board with the input file information, update it both functionally and visually after each swapping operation, can navigate the user throughout a clearly functioning menu system. The last two things that our code needs are the checking process for whether if the current level is completed and playing a ball animation.

After all the boxes and the grid are updated, we set the connections between the pipes to solve the level. Every pipe has direction boolean values that indicate which directions they have openings for connecting. Every pipe sets to connected or not connected whether all their openings face other pipes openings. The connecting process starts from the starter pipe to each and every pipe that is : connected from all sides and has a connection with a pipe that is

in connection reaching from the starter pipe. These pipes that have a connection from the start are added to pipes arraylist for building the path for the ball to go through. When the end gains a connection from start the level is finished and indicated with a boolean value levelFinished set to true. It should be notified that with every swap action and update of the grid, connections are reset and pipes solution is cleared and refilled, so when detaching already connected pipes the solution chain will break. When level is finished, the user may no longer make any changes to the grid as all the boxes will be mouse transparent.

Finally, the ball animation part comes. If this step is neatly scrutinized, it's clear that the animation should work together with the level completion check. A boolean called "animationresetter" is created in the code. Whenever the user clicks on the main menu button, the value for this resets to false. This made sure that in order to see the animation once again, one has to solve the puzzle from the beginning.

The visual side of the ballAnimation method uses the "polyline" class as a path. This shape is chosen since it is not automatically closed like a polygon.

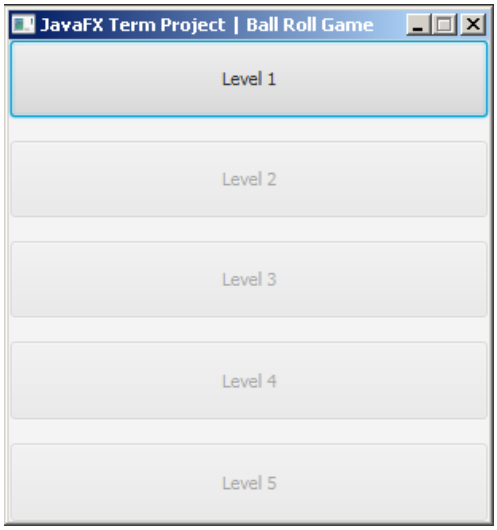
It was stated before that whenever a pipe is recognized as a one in the solution of the game, it is also added to the "pipes" arraylist. Here, this pipes arraylist (a.k.a the solution way) is traversed one by one and the coordinates of our polyline path take their values from the exact middle location of the pipes it encounters. When the pipe in the way is encountered from the arraylist, it's exact middle coordinates is sent to our polyline as the new target.
Under favour of this effectuation, from beginning to end, the polyline always follows the emerged path.

Combining all of these logic, now the JavaFX game is created. It can update itself, check the conditions for an endgame, navigate the game flow and the user's choices, with proper graphics and animations.

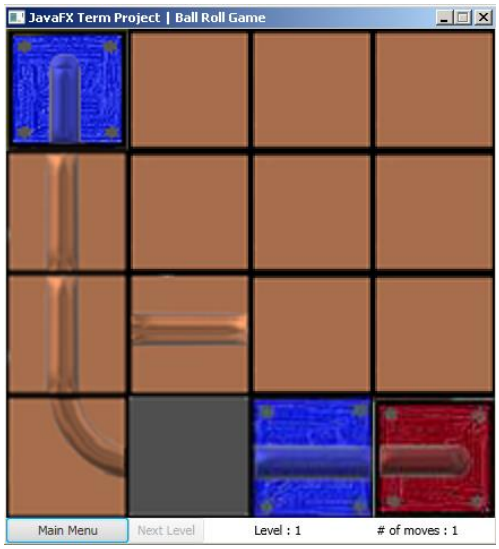
Test Cases

When the application is opened, a main menu welcomes the user.

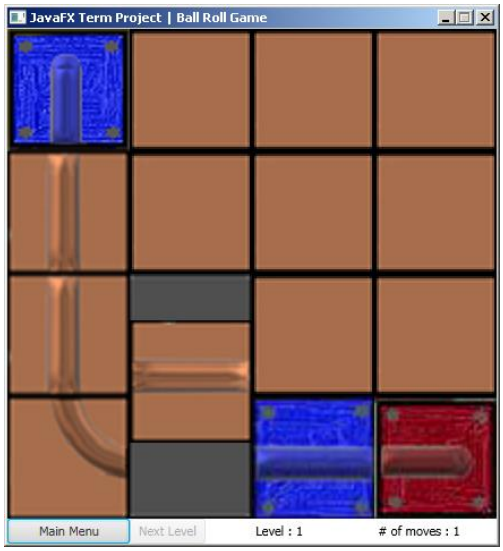
Here, only the first level is available.



When “Level 1” is clicked, the gameboard is generated correctly and ready to play. Also, informative strings show the current level and the number of moves:

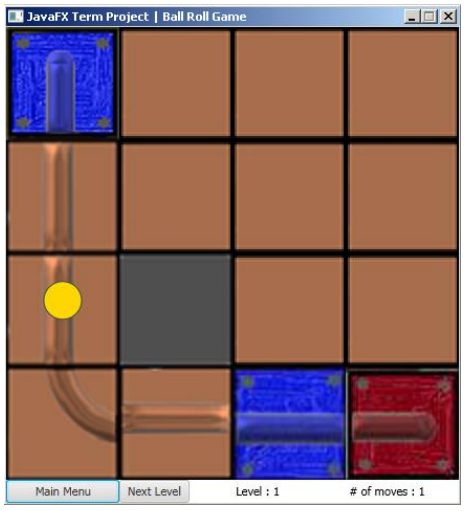


Here, the player can start the swapping operations. An exemplary moment can be seen below:

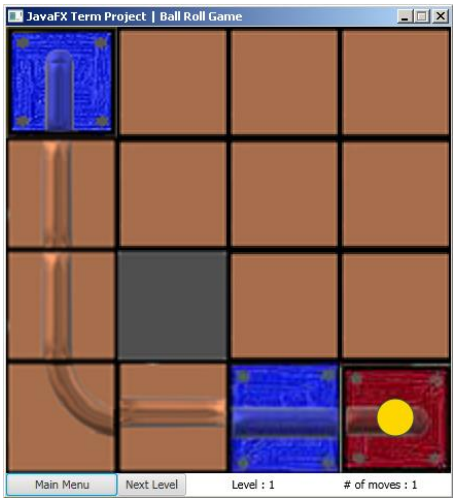


It should be declared that as an *additional feature*, **we added a smooth visual effect** when swapping the nodes, instead of harsh movement of the default dragging operation.

Since the move above solves the first level, the roll should be rolling after this:

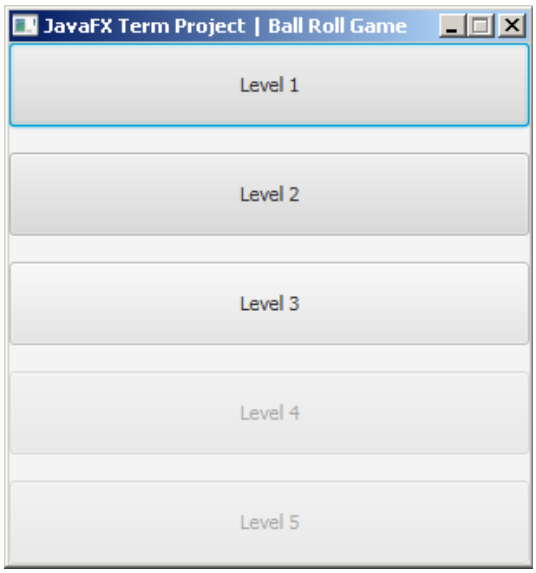


And level one will be completed after the animation.

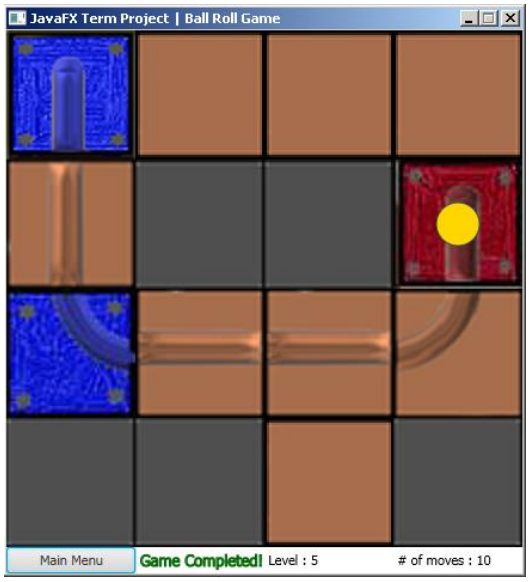


It should be noticed that when the ball reaches to the end node and the level is successfully completed, “Next Level” button becomes available for the click. However, the user may choose to insist on main menu for the navigation, so also in the main menu, buttons for the upcoming level is opened after the previous level is completed.

An examplary moment for this can be seen below. This picture captures the moment when user finishes first two levels and clicks on the main menu button:

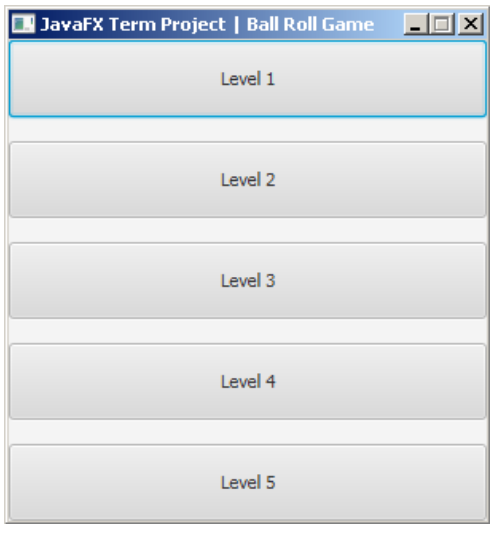


Finally, after assuming that the user has completed all the levels and made the exact final move, which solves the ultimate (fifth) level:



A green **"Game Completed!"** text is now visible. This can be considered as an additional feature for applying more interactive connection with the user.

As a final screenshot, it is appropriate to show the condition of the main menu, after the game is hundred percent completed. All levels are now unlocked:



After this, the game is finished.

Final noteworthy aspect: if the user wants to play a level more than one time and clicks back to something which he already finished, that level starts from the initial point as if it's the

first time being displayed. Nevertheless, unlocking other levels is still a valid charter and isn’t affected by this operation. Under this favour, one can play the same level as many times as he pleases but at the same time, completing it for solely one time is enough for opening the next level.

These explanations conclude the test cases.