****

**COMPUTER PROGRAMMING II, SPRİNG 2022**

**TERM PROJECT / PIPE-PUZZLE GAME**

NAME STUDENT NUMBER

Eren DUYUK 150120509

Eren ÇETİN 150120008

Submitted to: Mustafa AĞAOĞLU

Due Date: 09.05.2022

**1-) PROBLEM DEFINITION**

The aim of this project is to make a puzzle game using java and javafx framework. The purpose of the game is to get the ball from the starting point to the finish point.

There is a 4x4 game board in the game. Each cell on this board is one of the types " horizontal and vertical start", " horizontal and vertical end""horizontal and vertical pipe static", "horizontal and vertical pipe", "00,01,10,11 inclined pipe", "empty", "empty free". At the end of this section, information about cell types is given.

The game has 6 levels, each with different difficulty. The design of these levels is read from the text file as input. When you run the game, each level's button appears, you can start from the level you want, but you must successfully complete the level in order to move to the next level from the level you selected.

The game is played using the mouse. When you click on the movable cells and take them to the empty free cell and release it, the first cell we clicked goes there. This movement can only be a horizontal or vertical unit, not diagonal.

When you run the game, you will see the buttons for 6 different levels under the first "The Game" text. After choosing the level you want, the game board opens and by controlling the cells with the mouse, it is tried to complete the path that the ball will go from start to finish. Also, every move you make is written on the right side of the game board. In the game, if the player thinks that he has completed the level by combining it correctly, he checks the level by clicking the "nextPuzzle" button. If the player got it right, the ball moves and reaches the finish point, but if it's not right, the ball doesn't move and says "There must be a problem with your solution. Try again!" is written on the right side.

**Starter:** The starter is the cell where the ball is located at the beginning of the level. The ball starts moving from here. There can only be one on the starter game board. It cannot move. Starter can be horizontal or vertical.

**End:** End is the cell we want to deliver the ball. There can be only one end game board. It cannot move. End can be horizontal or vertical.

**Empty Free:** There are no pipes in this cell. Other movable cells can move here.

**Empty:** There are no pipes in this cell. Other movable cells cannot move here, but empty cell can move to empty free cell.

**Pipe:** The game board can contain more than one. There are two types, one is horizontal and the other is vertical. This cell, which can move, can move to empty free.

**Pipe Static:** There can be more than one on the game board. These cells, which cannot move, can be of two types, one horizontal and the other vertical.

**Curved Pipe:** Curved pipes have 4 different shapes, they are named 00, 01, 10, 11. There can be more than one on the game board. It can move to empty free cells.

**Curved Static Pipe:** Curved static pipes have 4 different shapes, they are named 00, 01, 10, 11. There can be more than one on the game board. It cannot move to empty free cells.

**2-) IMPLEMENTATION DETAILS**

**1-) InfoNode Class**

|  |  |
| --- | --- |
| InfoNode | |
| -  -  - | id : int  type: String  property : String |
| +  + | InfoNode(id : int, type : String, property : String)  Getter and setter methods for data field |

* The purpose of that class that keeps information about nodes which create pipe puzzle game.
* “id” variable whose type is integer is related to position of nodes in the pipe puzzle game. We start from left-top part as 1 and that number increases while going to right. After reaching the end of row, it continues from the most left of the new row to the end of it.
* “type” variable whose type is String is related to type of nodes. There are 5 types in our game. Starter, Pipe, Empty, PipeStatic, and End.
* “property” variable whose type is String is related to property of nodes. There are many properties in our game. There are two properties for Starter and End types. Horizontal and Vertical. There are six properties for Pipe and PipeStatic types, 00, 01, 10, 11, Horizontal and Vertical. Lastly, there are two properties for Empty. Free and none.
* We have InfoNode constructor. It has three parameters. They are id, type, and property. Its visibility is public.
* We have also getter and setter methods for id, type, and property.

**2-) BackendGame Class**

|  |  |
| --- | --- |
| BackendGame | |
| -  - | backendGame : InfoNode[][]  nextMoveOfBall : ArrayList<String> |
| +  +  +  +  +  +  + | BackendGame()  createBackend(fileName : String) : void  isMoveable(x0: int, y0: int, x1 : int, y1 : int) : boolean  moveInfoCell(x0: int, y0: int, x1 : int, y1 : int) : void  checkSolution() : boolean  isLegal(x0: int, y0: int, x1 : int, y1 : int) : boolean  Getter and setter methods for data field |

* The “backendGame” variable is a two dimensional array. It consists of InfoNode class’s objects. The “nextMoveOfBall” variable is an ArrayList which consists of String objects. That ArrayList is designed for animation of ball.
* We have a BackendGame constructor. In the constructor, we create backendGame to consist of 4 rows and 4 columns because our pipe-puzzle game has a 4x4 size. We also create nextMoveOfBall arrayList to add or remove elements in other methods.
* We have createBackend method. It has a String type parameter. We have three local variables for that method. They are line which has a String type, x which is related to row, and y which is related for column whose type is integer. We use BufferedReader to read datas from text files line by line and line which is local variable changes according to that. We create InfoNode objects from each line from text files, except blank lines and according to x and y values, we place them into the backendGame array. We increase the y value by one after each placing, If the value of y is equal and bigger than 4, we increase x value by one and set the y value with 0. Thıs way, we create backend of our game.
* We have the isMovable method. It has four parameters and the return type of it is boolean. Its parameters are x0, y0, x1, y1 and all of them are integer type. In our game, a cell can move 1 cell distance, vertically or horizontally, and cannot move diagonally. Because of that, this method checks current position of the cell(x0,y0) and new position of the cell(x1,y1) and returns true or false according to whether these coordinates are appropriate to move or not.
* We have the moveInfoCell method. It has four parameters and the return type of it is void. Its parameters are x0, y0, x1, and y1 and all of them are integer type. In that method, we change contents between (x0,y0) in the backendGame two dimensional array and (x1,y1) in the backendGame two dimensional array.
* The checkSolution method, whose parameter is empty and returns boolean type, checks whether the player has connected the pipes correctly. Returns true if the player connected the pipes correctly Returns false if the player connected the pipes incorrectly.

The first thing this method does is to find the starter cell by going through all the cells on the game board. The position of the stater cell on the game board is to assign the values x0 and y0, whose values were originally 0. This is also the position where the ball begins play. Then, the next cell where the ball will go is determined according to whether the starter cell is vertical or horizontal. If it is horizontal, it means the ball will go to the cell on the left, and the value of x1 is set to x0, the value of y1 is set to the value of y0-1. If the starter cell is vertical, it means the ball will go to the cell below and the value of x1 is set to x0+1, the value of y1 is set to the value of y0. In addition to all these, information on which direction the ball will go is added to the nextMoveOfBall list. If the starter cell is vertical, it is added as "down", if it is horizontal, it is added as "left".

After these operations, the position of x1 and y1 values ​​in the while loop is checked. If the cell is end, the method returns true and the loop ends. If the pipe or pipe is a static cell, the new values ​​of x1 and y1 are assigned according to the shape of the pipe and the direction the ball comes from, and x0 and y0 are assigned the previous value of x1 and y1. Another possibility is that if the location of the cell in the new x1 and y1 values ​​does not satisfy the conditions, the method returns and the loop ends.

For example, suppose the cell at the position at the initial value of x1 and y1 is a curved pipe 10. If x1 and x0 are equal, the ball is coming from the left and its next move is to go to the cell below. If x1 is less than x0, the ball is coming from below and its next move is to go left. Let's assume x1 and x0 are equal. The process will be to assign the position of the next cell the ball will go to to the values ​​x1 and y1. In the case we assume, x0 and y0 values ​​are assigned to x1 and y1 at first, then the new x1 value becomes x1+1 as the ball goes down and the y1 value does not change. Also "down" is added to the nextMoveOfBall list. The cycle continues like this. If the location of the cell in the new x1 and y1 values ​​does not satisfy the conditions, the method returns and the loop ends. The method returns true if the position of the cell with the new x1 and y1 value reaches the end cell correctly.

* We have the isLegal method. It has four parameters and the return type of it is boolean. Its parameters are x0, y0, x1, y1 and all of them are integer types. We check the type of (x0,y0) in the backendGame array. In some cases, we check both (x0,y0) and (x1,y1) in the backendGame array with the getType() method in InfoNode class. According to check operation, the method returns true or false.
* We have also getter and setter methods for backendGame and nextMoveOfBall.

**3-) Game Class**

|  |  |
| --- | --- |
| Game | |
| -  -  -  -  -  -  -  -  -  -  -  -  -  - | rectangle : Rectangle[][]  boardPane : GridPane  backendGame : BackendGame  x0 : int  y0 : int  level : int  moveCounts : int  buttons : ArrayList<Button>  levelDesign : String[]  menuScene : Scene  gameScene : Scene  showMoveCount : Text  situation : Text  ball : Circle |
| +  +  +  +  +  +  +  +  + | start(primaryStage : Stage) : void  createGridPane() : void  setDraggable(rect : Rectangle) : void  setFree(rect : Rectangle) : void  exchangeRectangle(x0: int, y0: int, x1 : int, y1 : int) : void  setBall(level : String) : void  displayAnimation() : PathTransition  setPathOfBall() : Path  setMoveCount(moveCount : int) : void |

* We have start method. The return type of it is void. It has one parameter and it is object of Stage class. In this method, we create buttons ArrayList, levelDesign array which is consist of text file of levels, ball which is Circle object, backendGame which is BackendGame object, boardPane which is GridPane object, rectangle which is a two dimensional array of size 4x4, showMoveCount which is Text object, and situation which is Text object. We have nested loop to access each element in the rectangle array and create rectangle object for each element because all objects are null in array at the beginning. For menuScene which user see first on screen, we declare and create different six buttons and they represent levels. Then, we add all buttons to the buttons arraylist. We also declare and create another button for gameScene whose name is nextPuzzle. For that button, showMoveCount, and situation, we declare and create Vbox object with a name of vbox to keep them vertically on gameScene. We declare and create Pane object with a name of pane. We add boardPane and ball to the pane. We create and declare Hbox object with a name of hbox. We add pane and vbox to the hbox. “hbox” is for gameScene. We set on action event on each buttons and we arrange gameScene and stage according to that. For menuScene, we declare and create BorderPane object with a name of borderPane and we add buttons arraylist and title to the borderPane. Lastly, we set menuScene as borderPane and invoke show method.
* We have createGridPane method. Return type of it is void and it doesn’t have parameter. We have nested loop in that method. That loop allows us to access each element in the backendGame array and take type and property of each element in the array. We have two local variables. They are type and property. Their type is String and they store type and property each element in the array. After storing them, we fill inside rectangle objects in rectangle array with consistent images according to storing values. We have another nested loop in that method to determine which rectangles are draggable or places to drop draggable rectangles for the pipe-puzzle game.
* We have setDraggable method. Return type of it is void and it has one parameter. It is Rectangle object. In that method, we set on drag detected event on object which pass to the that method. In the event, we use getSource() method to find source object. We cast it into Rectangle class and assign it to the temp which is local variable with Rectangle type. We use nested loop to find temp in rectangle array and take coordinates of it with x0 and y0. Lastly, we use startFullDrag() and consume methods to work that event properly.
* We have setFree method. Return type of it is void and it has one parameter. It is Rectangle object. In that method, we set on mouse drag released event on object which passes to the that method. In the event, we use getSource() method to find source object. We cast it into Rectangle class and assign it to the temp which is local variable with Rectangle type. We use nested loop to find temp in rectangle array and take coordinates of it with x1 and y1 which are local variables with integer type. With x0,y0,x1, and y1 values, we call the isLegal, isMoveable, moveInfoCell, and exchangeRectangle methods.
* We have exchangeRectangle method. Return type of it is void. It has four parameters. They are x0,y0,x1,y1 and they are all integer type. In that method, we arrange position of rectangles on boardPane. (x0,y0) represents coordinate of rectangle which move and (x1,y1) represents coordinate of where that rectangle drop. Lastly, we set moveCount value to be one more and set showMoveCount which is Text object according to that.
* We have setBall method. Return type of it is void. It has one parameter and its type is String. The purpose of that method is that find an InfoNode object which type is Starter on backendGame array and arrange x and y coordinates of ball according to pane and position of InfoNode whose type is Starter on backendGame array.
* We have displayAnimation method. Return type of it is PathTransition object and it doesn’t have parameter. In this method, we create a PathTransition object and set duration, cycle count, node, and path of that object. We set path of that object according to return path object of setPathOfBall method
* This method, which returns an empty Path data type, creates the path that the ball should follow during its animation. In order to create this path, it is necessary to know the cells where the ball will go, for this, the nextMoveOfBall list created in the backendGame class is used. From the first element to the last element of this list, it is visited in order and the direction the ball will go is determined by the values in it. For example, let's say "down", "right" and "up" in the list. In the for loop, the first "down" value is read and an element is added to the pathOfBall of the Path type defined at the beginning of the method to go down 150 pixels. Then the "right" value is read and the element is added to the pathOfBall so that it moves 150 pixels to the right. Finally, "up" is read and 150 pixels up is added to the pathOfBall. As a result pathOfBall is set to down right up and the method returns this pathOfBall'.
* We have setter method for moveCount.

**3-) QUESTIONS**

1. **Which parts are complete/incomplete in your Project?**

There is no incomplete part in our project, all of them are working without any errors.

1. **What are the difficulties you have encountered during the implementation?**

The first problem we encountered during the project was moving the cells with the mouse because we had no experience before. At first some cells were moving, but when we tried to move some of them, we were getting too many errors and the program stopped working.

Another problem we encountered was related to the animation of the ball. Creating the path for the ball to follow was not as easy as we expected because we had to determine the next direction the ball should go and combine them.

1. **What are the additional functionalities of your project added by your team?**

We added an extra menu to the game, in this menu, each level has its own button under "The Game". Thus, the player can start from the level they want when starting the game.

Since the design of the given cells did not match our animation, we used another cell design. In this way, the ball moves more realistically in the cells. Also, "GOAL!" in the end cell. and the direction arrow in the starter cell makes it easier for the player to play the game.

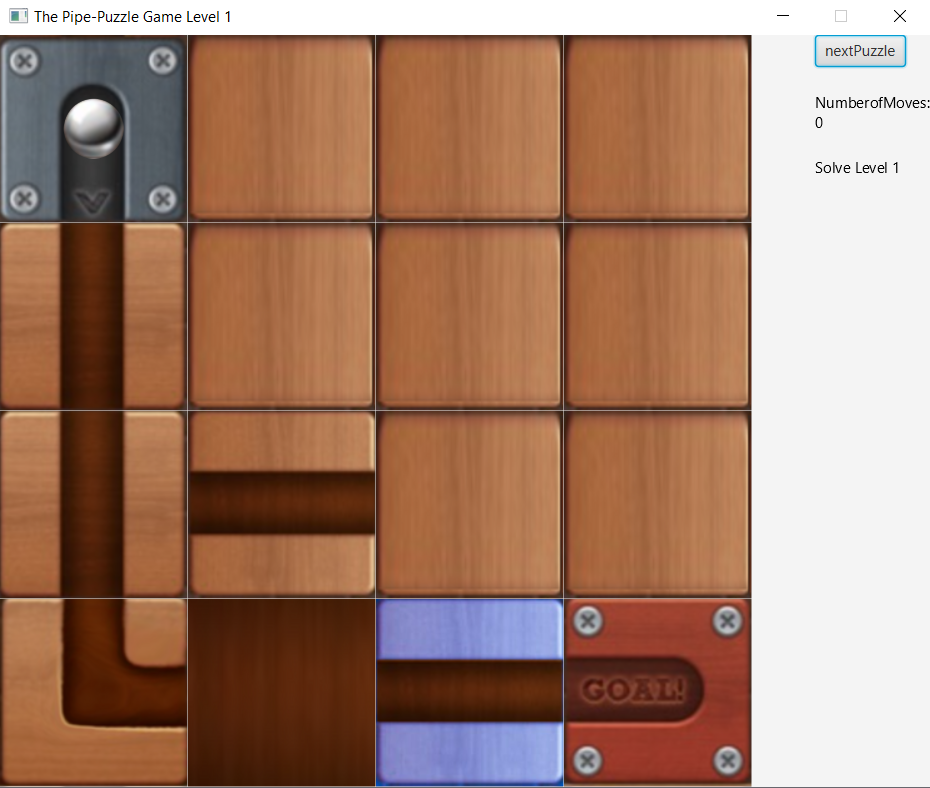
**4-) TEST CASES**

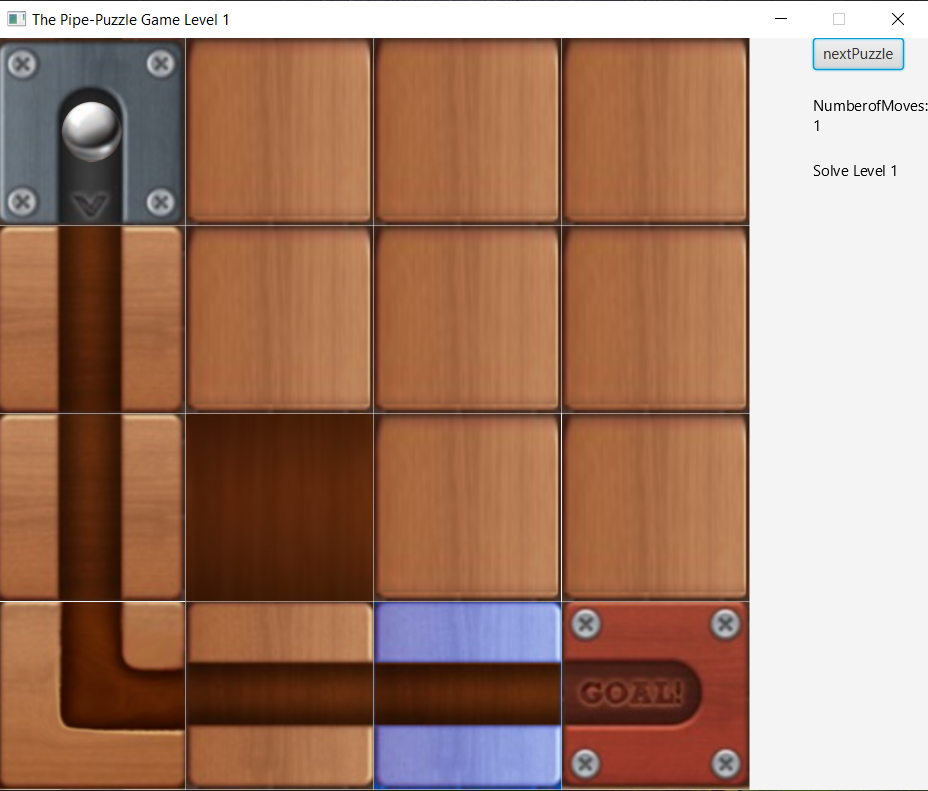
**1-) Menu**

****

The game has a simple menu. You can see the buttons for 6 different levels under the first "The Pipe-Puzzle Game" text. From which level the player wants to start, he/she can press the button of that level.

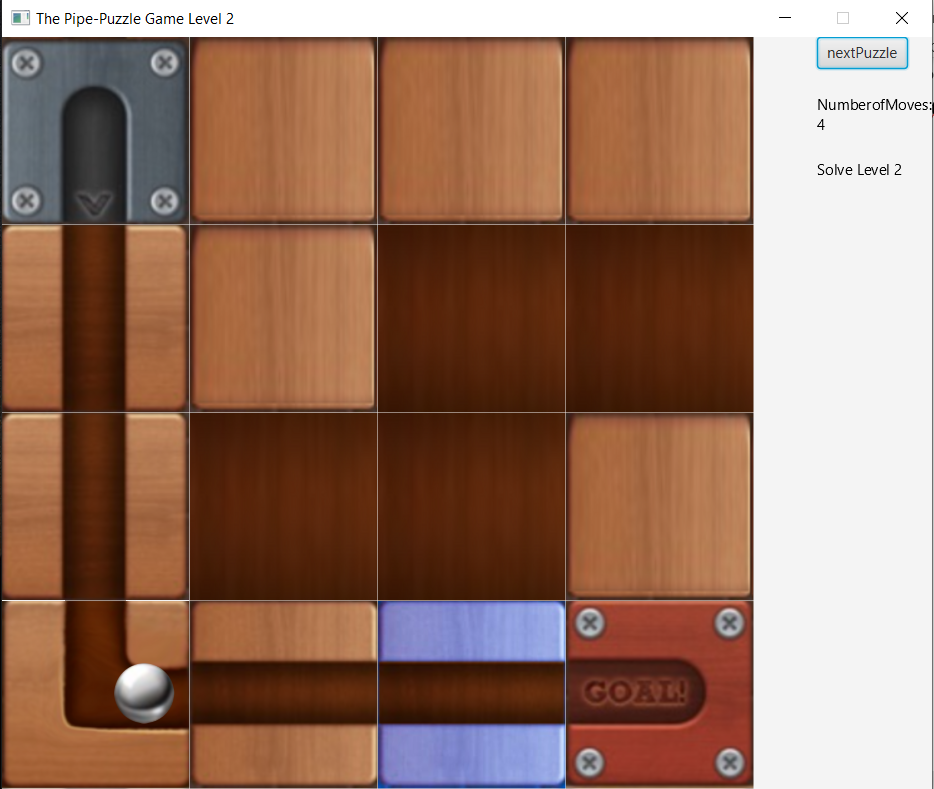
**2-) Game Board**

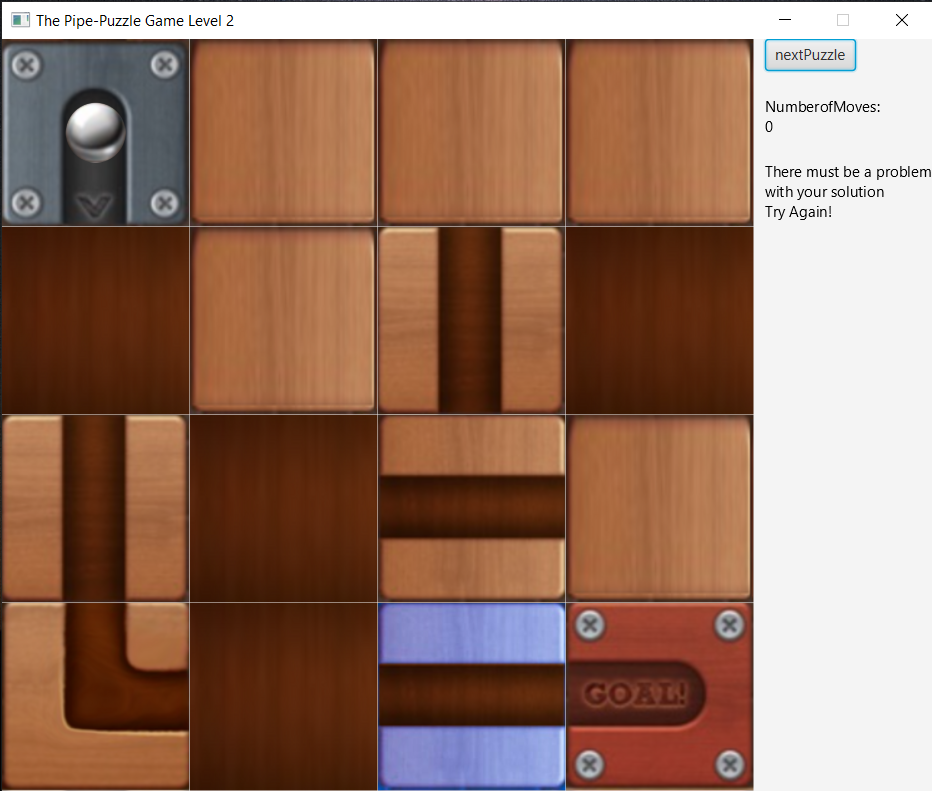
When you click on “Play Level 1”, the game board appears on the screen and cells can be moved according to the rules of the game. On the right, we can see the number of moves we made at that level.

****

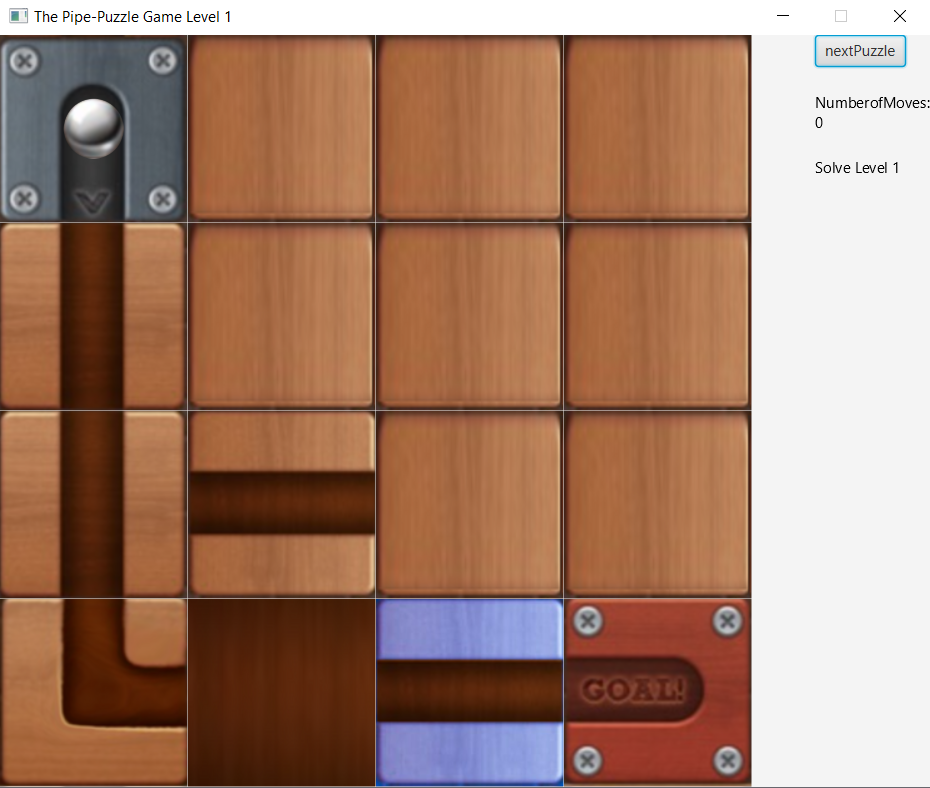
**3-) Press nextPuzzle Button**

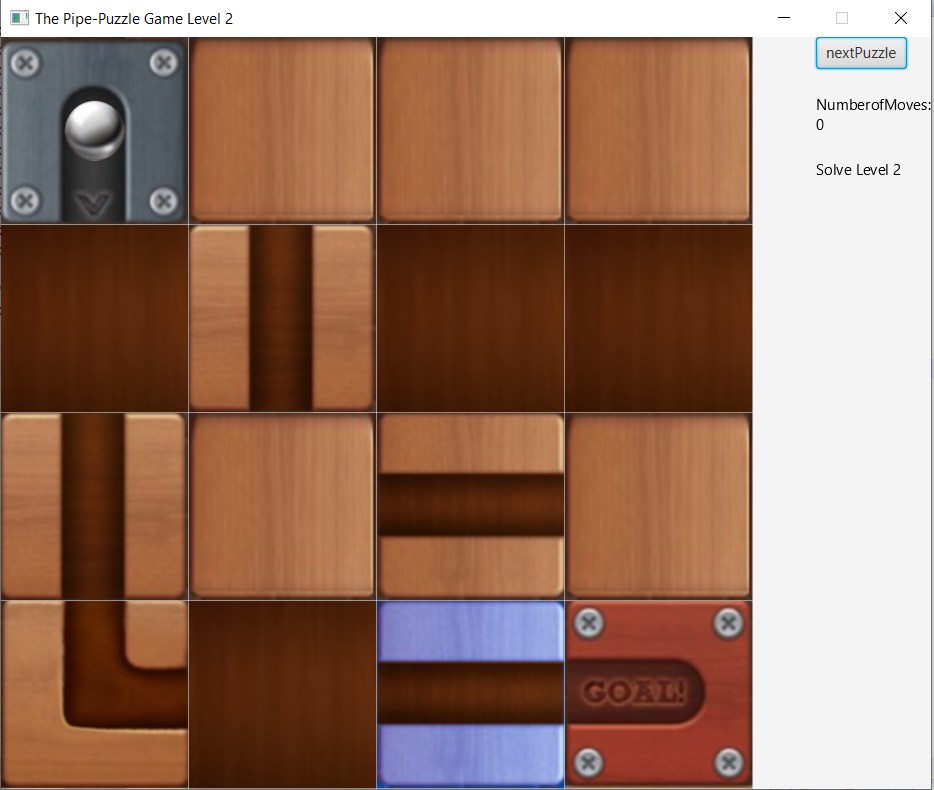
If the player thinks that he has completed the level by combining it correctly, he checks the level by clicking the "nextPuzzle" button. If the solution of the puzzle is correct, the ball moves and reaches the end cell. Then it goes to the next level. If the solution of the puzzle is correct, the ball moves and reaches the end cell. Then it goes to the next level. However, if the solution of the puzzle is not correct, a warning that "There must be a problem with your solution. Try again!" will appear on the right and the movement counter will be reset.





**4-All Level Designs**

****

****

