**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**DEPARTMENT OF**

**COMPUTER ENGINEERING**



CSE1142

Computer Programming II, Spring 2020

Term Project

Date of submission: 10 May 2020

NAME:                                                   STUDENT NUMBER:

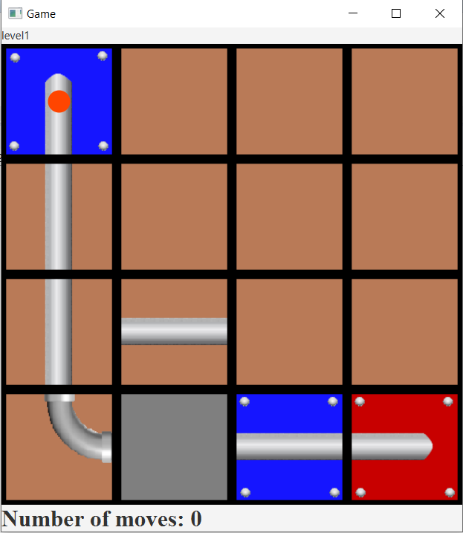
     Zeynep Ferah AKKURT                         150119824

     Merve Rana KIZIL                                  150119825

Problem Definition

This game aims moving a ball from a start point to an end point if there is an appropriate path. The player is must create a path using pipes given as sliding tiles. There are several levels. Default level is 5. But it can be set as different number of levels which is explained in part of implementation details.

Example: Game board of level 1:



To play the game, player must choose a tile, drag it into somewhere and drop it. But there are some rules for sliding tiles and playing the next level.

Here are the rules:

- Player can move the pipes by using only mouse. Keys on the computer are useless.

- Player can move the pipes only to free spaces which is colored grey.

- Player can only move non-static pipes (red and blue pipes is static).

- The blue pipe that starts in the middle of a tile and has a ball on it, is start point. And the red one is end point.

- Player can move pipes only one tile distance vertically or horizontally.

- If the appropriate path appears, then the ball falls from starter to end point.

- Once the ball falls and complete that level, player will be able to play the next level.

- There is no limit for player’s moves.

If all levels are passed one by one, pane will disappear, and then player will understand that the game is finished with a message and a song.

Implementation Details

**Test** class is using the following UML diagram:

|  |  |
| --- | --- |
| ***Test*** | |
| -  - | start (primaryStage: Stage): void  main (args: String []): void |

• Test class is subclass of Application.

• Test class must implement the inherited abstract method **start**(stage). Inside of it, stage will be set.

**Tile** class is using the following UML diagram:

|  |  |
| --- | --- |
| ***Tile*** | |
| -  -  -  - | id: int  type: String  property: String  image: ImageView |
| +  +  +  +  +  + | Tile ()  Tile (tile: Tile)  Tile (id: int, type: String, property: String)  determineImage (): void  toString (): String  getter/setter methods |

• Tile class creates an object that has four data fields: **id** is the tile’s number. Number can be minimum 1 and maximum 16 since there are 16 tiles on the board. **type** shows if tile is a pipe, empty space, start or end point. **property** keeps tile’s shape that can be free, vertical, horizontal or basically none if there is no property. Every tile has an **image** that matches with its type and property.

• There are 3 kinds of constructors just in case.

• The method **determineImage** () checks the tile’s type and property. And from that information sets the image which is an ImageView object, to a .png file.

• The method **toString ()** returns an information about Tile object.

• There are **getter/setter** methods for each data field.

**Road** class is using the following UML diagram:

|  |  |
| --- | --- |
| ***Road*** | |
| -  -  -  - | roadTiles: ArrayList<Tile>  tiles: ArrayList<Tile>  start: Tile  end: Tile |
| +  +  +  +  + | Road ()  Road (tiles: ArrayList<Tile>)  levelRoad (): boolean  toString (): String  getter/setter methods |

• Road class is to find an appropriate path from start to end point. To do that it has four data fields:

Game board has 16 tiles need to be check for path (tiles are kept in Tile ArrayList named **tiles**, there is a **start** and **end** point inside of it**.**). Every time player makes a move, path must be checked with updated tiles from GameBoard class. Appropriate path kept in ArrayList named **roadTiles**.

• The method **levelRoad** () return true if there is path. To determine the path first it sets the start and end tile. It checks tiles that stands right, left, up and down of the source tile with their id’s from start tile to end tile. At first source is start point. when the source is end, it returns true.

roadTiles will be updated every time source changes because this means there is possible right tile came after source.

• Ex: After a tile that has a type of vertical, either a vertical pipe or curved pipe (00,01,10,11) can come after that. So, there is only a few options for appropriate path. In this case, since Starter’s type is vertical and because of a vertical starter only goes down, as a curved pipe 00 or 01 can come. If it does not, method returns false. If it does, that tile will be the source and added to the roadTiles list.

• To understand about the path’s direction, every time, it checks the tile that comes before it.

Ex: If the source is a horizontal pipe, ball’s direction may be to right or left. To figure that out, roadTiles’s last element – 1 (because last element is already the source) must be checked that which tile is added. If there is appropriate tile on the left that means ball is going to right. Each step must be do that otherwise program will go infinite. Because every time it checks, it will find the right property (horizontal) but not the end tile. This part extremely important especially for curved pipes.

• Method **toString** () gives an information about the road (roadTiles list).

• There are **getter/setter** methods for each data.

**GameBoard** class is using the following UML diagram:

|  |  |
| --- | --- |
| ***GameBoard*** | |
| -  -  -  -  -  -  -  -  -  -  -  - | game: BorderPane  pane: GridPane  tiles: ArrayList<Tile>  source: Tile  moveNumber: int  move: Label  filename: String  N: final int  xcoordinate: double  ycoordinate: double  gameCircle: Circle  mediaPlayer: MediaPlayer |
| +  +  +  +  +  +  +  -  -  -  -  -  -  -  -  + | GameBoard ()  setOnDragDetected (image: ImageView): void  setOnDragOver (): void  setOnDragDropped (targetTiles: ArrayList<Tile>, targetReal: Tile): void  setOnDragDone (targetTiles: ArrayList<Tile>, targetReal: Tile): void  readLevel (fileName: String): ArrayList<Tile>  setBoard (fileName: String, temp: ArrayList<Tile>): void  checkTargets (targets: ArrayList<Tile>): void  columnNumber (sourceId: int): int  rowNumber (sourceId: int): int  checkCondition (): void  setCircleGame (start: Tile): void  whichLevel (fileName: String): void  getXCoordinates (id: int, x: double): int  getYCoordinates (id: int, y: double): int  toString (): String |

GameBoard class is to create the game board with lots of properties. It controls mouse events, the number of moves that made by player, level controlling and playing it and the animation of ball’s falling. In order to do that, it has a several data fields and methods.

There are **N** number of levels. It can be set as different number, but the default is 5. This is constant value for the game. And each level has a different 16 number of images. Images of each tile from a **tiles** list that has been read from a file named **fileName,** is all in **pane** (GridPane) with an order by their id numbers. (Ex. first row: 1 2 3 4) The tile that player wanted to move is the **source**. And every time source is removed by player, it counts and stored in **moveNumber**. To show the number of moves, label object **move** is used. In the beginning of the game, ball will be on start point which has a **x and y coordinates** depend on the id number of a Starter. To show the ball, circle object **gameCircle** is used. And at the end, all those properties gather in one border pane which named as **game**.

• There is a default constructor sets the gameboard. And given default fileName is “level1”.

• The method **checkCondition** () is the key method in this class. It determines the tile which can be move or not. Once it finds one, it checks that is there any mouse event on that tile’s image.

Following four methods is for event handling. They implement the handle(event) method inside.

• **setOnDragDetected** (ImageView), if given image is choose by player’s mouse then handle method will set the given image as a source. And calls the method setOnDragOver to check if the source is move to any target or not.

• **setOnDragOver** (), first checks the targets and if the source is dragged on one of the targets, the handle method will call the setOnDragDropped method with chosen target.

• **setOnDragDropped** (), if source is drop to the target then handle method will drop the image and calls setOnDragDone method.

• **setOnDragDone** (), if everything works fine with the source, handle method will update the board. First it puts the source on a target tile and a Free tile on a source tile on the board. This means the move is made. This part must update tiles list too. Then after a movement, the road will be checked and if there is an appropriate path, animation of the ball will start and ball will fall, then next level will appear on the screen. If there is no path, then player can make another move until there is a path or the player closes the game.

• The method **checkTargets** (ArrayList<Tile>) determines one tile distance free tiles. And sets the targets list.

• The method **setCircleGame** (Tile), creates the ball on starter point.

• The method c**olumnNumber** (int) returns the tile’s column number on the grid pane depend on its id number.

• The method **rowNumber** (int) returns the tile’s row number on the grid pane depend on its id number.

• Method **toString** () gives an information about the tiles on pane (tiles list).

• The method **setBoard** () sets both grid and border pane by calling readLevel () with given fileName.

• The method **readLevel** (String), reads from the given filename and sets the tiles.

• The method **whichLevel** (String) sets the filename according to final N value. filename must be start with “level” and then the number of the level.

• Ex: level5

• The method **getYCoordinates** (int, double) return y coordinate of the tile by using its id.

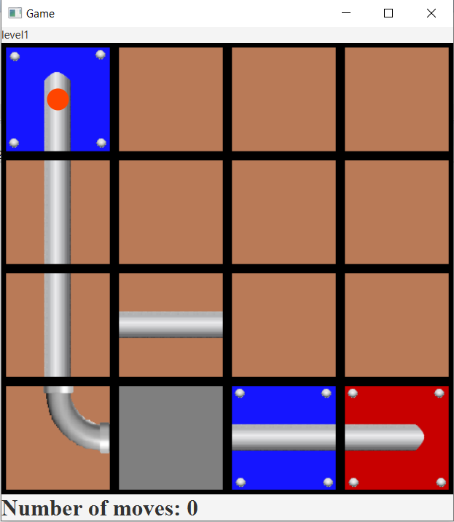
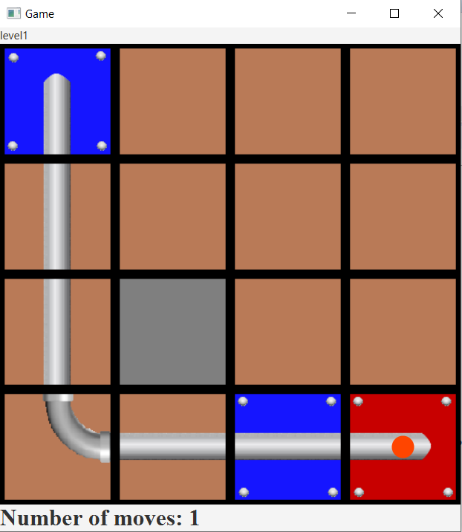
• The method **getXCoordinates** (int, double) return x coordinate of the tile by using its id.

• There are **getter/setter** methods for each data.

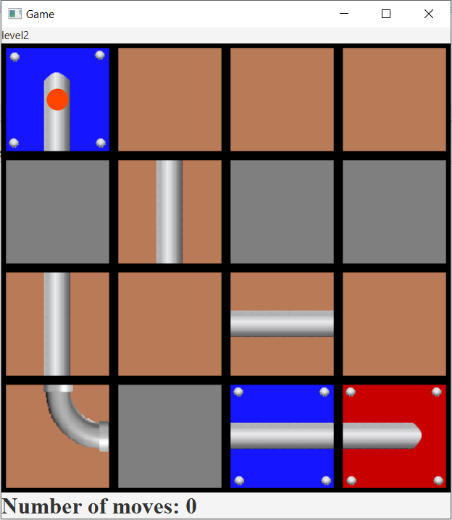
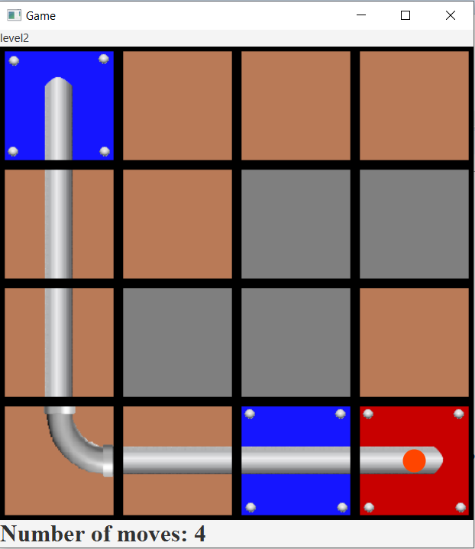
Test Cases

There are several test cases given as an input to the program. Here is an example game which has 5 levels. The screenshots of each level case are given below. As it can be seen each level has different locations for tiles so, the number of moves could be much more for some cases. The screenshots on the left hand-side are taken when the game board is just set and the screenshots on the right hand-side are taken when the current level is completed.

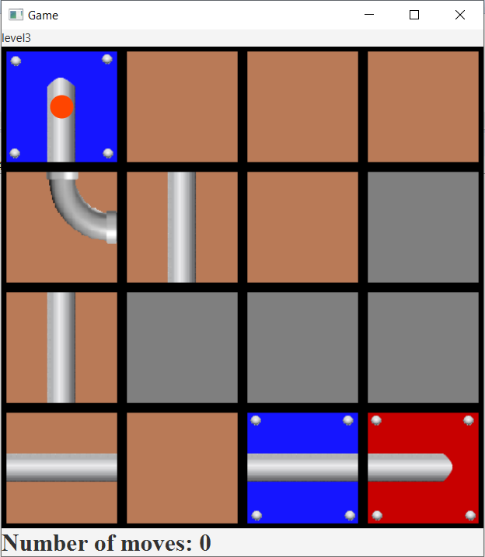
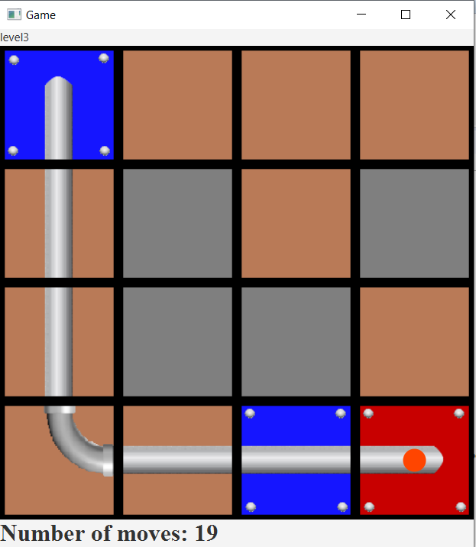
Level 1

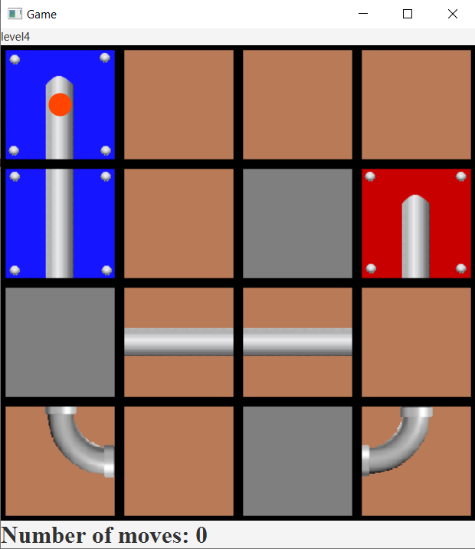
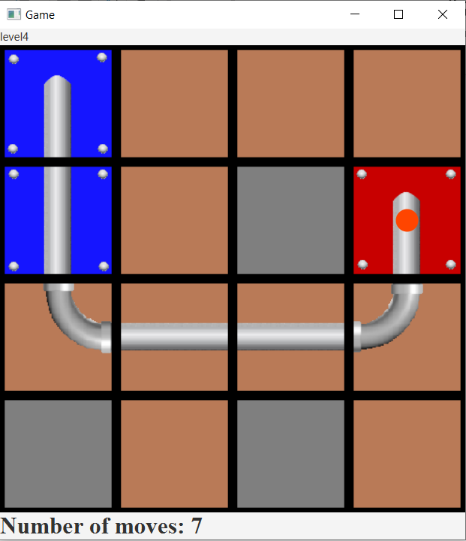
Level 2

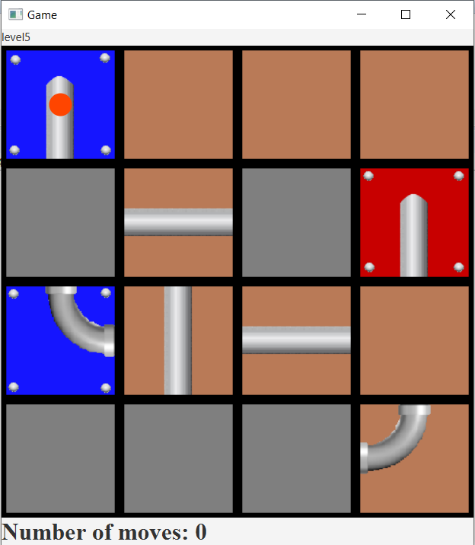
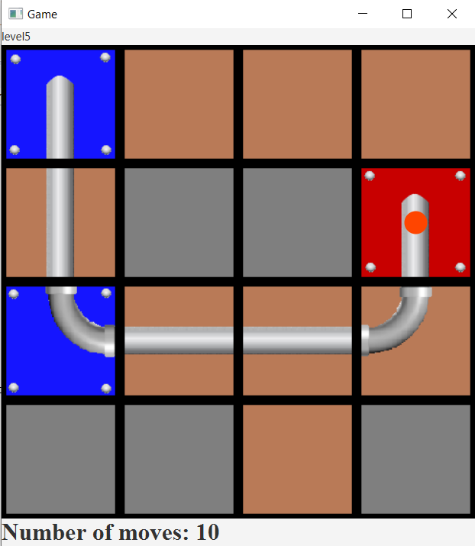
Level 3

Level 4

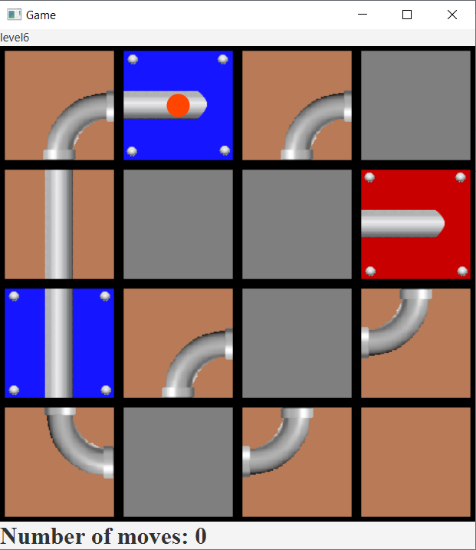
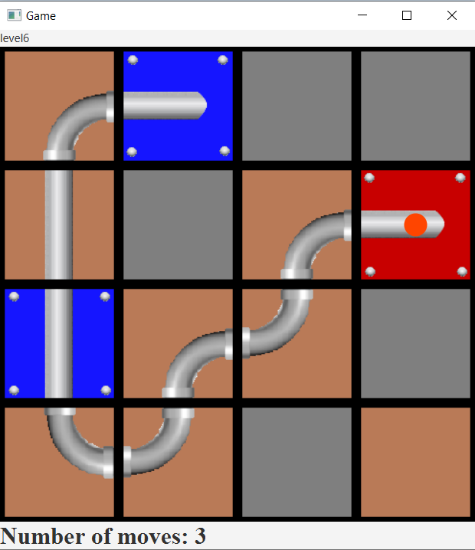
Level 5

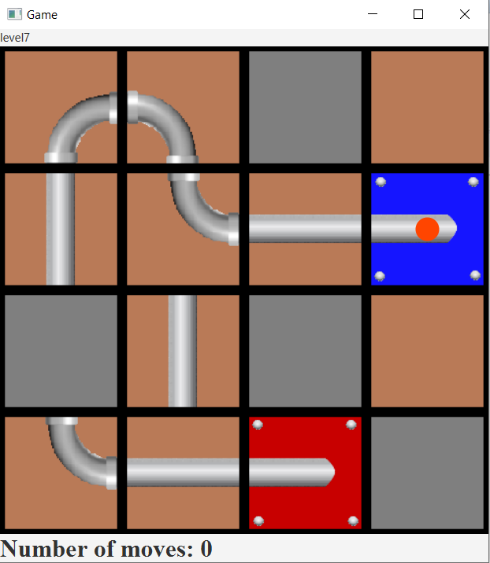
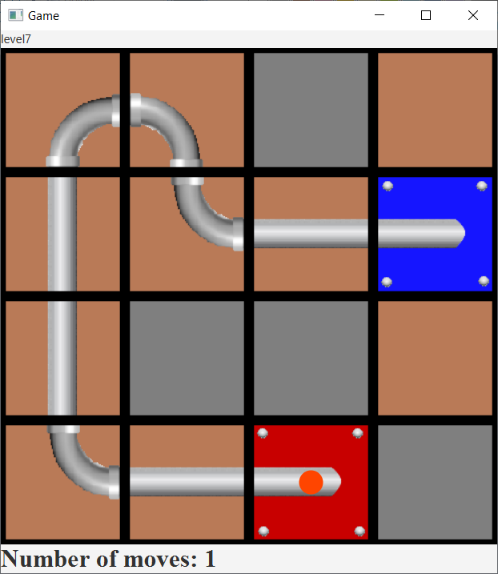
 

When the game starts, the game board is set and the ball appears in the starter tile, which is colored blue. Move number is written at the bottom of the border pane and it is incremented by 1 after each tile movement. And on the top of the board, current level is written. The locations of blue and red tiles are static. Therefore, they cannot be moved by the player as it can be seen in the screenshots. If the appropriate path is built, the ball moves through the pipes until it reaches the end tile, which is colored red. When the ball reaches the end tile, the game board for the next level will be set if the current level is not the last level, and the number of moves will be reset.

Here are some additional test cases. And the .txt files of them and more is also added to the project file.



When the game is finished, given below screen will appear with a little song.

