

**CSE 1242 - COMPUTER PROGRAMMING II TERM PROJECT**

**Project Name:** Pipe Puzzle **Authors:**

**1-)**150121069 Kadir Pekdemir

**2-)**150120025 Murat Albayrak

**3-)**CSE1242 Computer Programming II, Spring 2022.

**4-)**Date Subbitted: May 09, 2022

# PROBLEM DEFINITION

The game consists of 9 different levels. The purpose of this game is to connect the pipes to provide that the ball animation.

In these tiles, different tile types such as "starter", "end", "empty free", "empty", "horizontal and vertical pipestatic", "horizontal and verticalpipe", "00,01,10,11 curved pipe" are placed.

***Starter*:** The starter tile is the pipe where the ball starts to slide at each level and there is one at each level. Starter tiles can not move. Starter tiles can be vertical and horizontal.

***End*:** The end tile is the pipe where the ball finishes to slide at each level and there is one at each level. Starter tiles can not move. End tiles can be vertical and horizontal.

***Empty Free*:** Empty tiles are tiles without pipe. These tiles cannot move but other tiles can move to the position of this tile.

***Empty*:** Empty tiles are tiles without pipe. Empty tiles are tiles without pipe. These tiles cannot be replaced by other tiles, but they can move to the positions of empty tiles.

***Pipe Static*:** Pipe static tiles cannot move; their locations are static and they can be horizontal or

vertical.

***Pipe*:** It is the most common type of pipe in the game. They can be horizontal or vertical and can moved to the position of empty free tiles.

***Curved Pipe*:** Curved pipes are available in four different shapes. Four of these numbers are 00,01,10,11. Different names are given in terms of shape properties. Curved pipes can be placed in the empty tiles position.

The game consists of 9 levels, each with different levels of difficulty. These levels are shown in the game by reading from the input file. There are 9 different buttons of levels on the intro screen.

Gameplay is provided by mouse. Movable tiles can be moved to the positions of the empty tiles when they are moved by holding them on top. The movable tiles can be carried one unit if there are empty free tiles in the vertical or horizontal directions, but they cannot be moved diagonally.

In the opening screen of the game, it writes a welcome message and there are buttons to select the levels of the game. When we start the game, there is a counter that calculates the number of moves the player has made and able to print on the screen how many moves the player has made until the end of the game. Furthermore the game screen also includes the "Main Menu" button, which allows the players to return to the main menu at any time. When the players click on this button, they return to the home screen and they can select level. If the tiles in the game have been moved correctly, the ball animation will be activated automatically and the ball will start to move.

# IMPLEMENTATION DETAILS

1. **Implementing a Tiles class with the following UML diagram.**

|  |  |
| --- | --- |
|  | Tiles |
| + | id: int |
| + | type: String |
| + | property: String |
| + | Tiles ( id : int, type : String, property : String) |

* + The aim of this class is to carry the required values of the tiles.
  + Each tile has an id, a type, a property.
  + Id value which type is an integer, is represents to id number dedicated to tiles. Type value which type is a string, is represents to types of tile. Information about the types of tiles is given above but if we want to talk about briefly, these tiles have 7 different types and this type value indicates these types. Property value which type is a string, is represents to properties of tiles and again if we want to talk about briefly, these tiles can have different properties such as 00, 01, 10, 11, vertical, horizontal, free, none, etc.
  + We have a Tiles constructor whose parameters are id, type, property and whose accessibility is public.

1. **Implementing a Game class with the following UML diagram.**

|  |  |
| --- | --- |
|  | Game |
| + | board: Tiles[][] |
| +  + | numberOfUnlockedLevel: int  moveCounter : Label |
| + | Game() |
| + | createMapData (inputFilePath: String): void |
| + | isLegal(x0: int, y0: int, x1: int, y1: int): boolean |
| +  +  +  + | makeMove(x0: int, y0: int, x1: int, y1: int): void  loadGame():void  saveGame():void  checkOver(): boolean |
|  |  |

* + Game class consists of the game's backend, mechanics and actions.
  + board variable which a multi-array of Tiles type is store tiles in order of indexes: x and y.

numberOfUnlockedLevel variable which type is int stores last unlocked level in case of level change, saving and loading.

* + We have a Game constructor whose parameter is empty and whose accessibility is public.
  + We have createMapData method whose parameters is inputFilePath type string and return type is void.This method generates gameboard inside the backend using inputFilePath and we defined in it “br” variable which type is BufferedReader to read level from inputFilePath line by line. This method creates a tiles of the tiles type and holds ID, type, and property, and places these values in the coordinates specified in the row and column. It prints an error messages when the input file encounters an error, close BufferedReader whatever happens.In the same time, if it cannot be closed it prints an error message.
  + We have isLegal method whose parameters are x0,y0,x1,y1 types int and return type is boolean. This method checks if move is legal(applicable) by given coordinates of tile that is being moved (x0,y0) and tile that is new position for tile to stay (x1,y1) finally it returns legality of move.After that if the block is imovable it returns false.Furthermore, it controls the displacement state of a unit on the horizontal and vertical axises.
  + We have makeMove method whose parameters are x0,y0,x1,y1 types int and return type is void.It changes the coordinates of tiles which coordinate values is given.
  + loadgame() method loads the new game according to the numberOfUnlockedLevel.
  + The saveGame() method saves the number of unlocked levels to the save.data file, so that without solving a level in the game, we prevent the transition to another level.
  + We have checkOver method whose parameter is empty and return type is boolean. This method checks if the pipes are connected correctly and if the game is over. It checks the proper connection of the pipes before and after the pipe that we look at the pipe by evaluating it is doing.This method have a currentTiles array which type is Tiles and it stores current tile to get type and property.Then by the way currrentTiles value is taken from board by giving positions as indexes.

1. **Implementing a Main class with the following UML diagram.**

|  |  |
| --- | --- |
| Main | |
| - | game: Game |
| -  -  -  -  -  -  -  - | numberOfLevel: int |
| numberOfMoves: int |
| levels: String[] |
| tiles: Rectangle[][] |
| boardPane: GridPane |
| gameStatusLabel: Label |
| ball: Circle |
| introScene: Scene |
| gameScene: Scene |
| +  +  +  +  +  + | start(window: Stage): void  displayAnimation(): PathTransition  createGridPane(): void  checkTiles(): void  backToStarter():PathTransition  nextLevel(): void |
| + | exchangeRectangles(x0: int, y0: int, x1: int, y1: int): void |

* + game which type is Game, is created to use the game mechanics of backend. level variable which type is int, represents the number of levels we are in. numberOfMoves variable which type is int calculates user’s moving. levels array which is in type of String, stores game level files.

boardGrid array which is in type of Rectangle, stores rectangles respectively. boardPane which type is GridPane, is a layout that provides Rectangle 4 to stop 4. gameStatusLabel which type is Label, is an article that whether the game is over and which level is shown in it. moveCounter variable which type is Label shows our move number. Ball which type is Circle, is a ball that will move when the animation starts. welcomeScene which type is Scene, is scene of the main menu. gameScene which type is Scene, is scene of the game.

* + We have standard start method whose parameters is window type Stage and return type is void and accessibility is public. In this method, we create an instance of our class Game to start backend of our game, create Rectangle array which we later use to show our game elements in gui and define levels string array here which we later use when generating gameboard. We load unlocked level from save file with loadGame method. We construct GridPane layout which will later store our Rectangles in 4x4 row and column structure. We set padding for this our boardPane (GridPane object), set gap between Rectangles to 0, and define our rectangles, set constraints and add them to boardPane.

When the Main menu button in gameScene is clicked, the scene changes to the intro scene.We create a VBox object which will store levels and create a buttons for levels. We designed separate buttons for each level, we set the number of moves and level indicator text of each level.

After that we add all buttons to levelsBox which is VBox’s object and create the main layout.

We set title to Top section of welcomePane and set levelsBox to Center section of welcomePane.Finally, we set the scene to welcomeScene when the application starts and set the title of window as “”Unblock Ball”. And Show window.

* We have nextLevel method whose parameters is empty and return type is void and accessibility is private. This method unlocks next level and generates gameboard 4x4 Tiles array and 4x4 Rectangle array. With this method, we unlock the next level, by increasing the "unlock level" value one by one. Then, we generate backend which Tiles array with size as 4x4 with parameter levels which stores input level file names.And at the end, we generate frontend which is Rectangle array with size as 4x4.
* We have generateGridPane method whose parameters is empty and return type is void and accessibility is private. This method generates Rectangle array(boardGrid) which size is 4x4 in order to fill rectangles with images to show in frontend. Method allows to fill rectangles with matching images of boardGrid elements. We created an image object which name is temp to provide the necessary conditions then according to type maching image and constructing Image object by URI path using File object.
  1. If the type is Pipe and property is Vertical then matching image "Pipe\_Vertical.png" and construct Image object by URI path using File object.Then, if it is moveable set it draggable with defined setDraggable method which makes Rectangle moveable(draggable).
  2. If the type is Pipe and property is Horizontal then matching image "Pipe\_Horizontal.png" and construct Image object by URI path using File object. Then, if it is moveable set it draggable with defined setDraggable method.
  3. If the type is Pipe and property is 00 then matching image "Pipe\_00.png" and construct Image object by URI path using File object. Then, if it is moveable set it draggable with defined setDraggable method.
  4. If the type is Pipe and property is 01 then matching image "Pipe\_01.png" and construct Image object by URI path using File object. Then, if it is moveable set it draggable with defined setDraggable method.
  5. If the type is Pipe and property is 10 then matching image "Pipe\_10.png" and construct Image object by URI path using File object. Then, if it is moveable set it draggable with defined setDraggable method.
  6. If the type is Pipe and property is 11 then matching image "Pipe\_11.png" and construct Image object by URI path using File object. Then, if it is moveable set it draggable with defined setDraggable method.
  7. If the type is Empty and property is Free then matching image "Empty\_Free.png" and construct Image object by URI path using File object. Empty Free is a element that we can drag other elements into so we set it as a slot with our setSlot method which makes Rectangle a slot which is a place other draggable elements can be moved into.
  8. If the type is Empty and property is none then matching image "Empty.png" and construct Image object by URI path using File object. Then, if it is moveable set it draggable with defined setDraggable method.
  9. If the type is PipeStatic and property is Horizontal then matching image

"PipeStatic\_Horizontal.png" and construct Image object by URI path using File object.

* 1. If the type is PipeStatic and property is 00 then matching image "PipeStatic\_00.png" and construct Image object by URI path using File object.
  2. If the type is PipeStatic and property is 01 then matching image "PipeStatic\_01.png" and construct Image object by URI path using File object.
  3. If the type is PipeStatic and property is 10 then matching image "PipeStatic\_10.png" and construct Image object by URI path using File object.
  4. If the type is PipeStatic and property is 11 then matching image "PipeStatic\_11.png" and construct Image object by URI path using File object.
  5. If the type is PipeStatic and property is Vertical then matching image

"PipeStatic\_Vertical.png" and construct Image object by URI path using File object.

* 1. If the type is Starter and property is Horizontal then matching image

"Starter\_Horizontal.png" and construct Image object by URI path using File object.

* 1. If the type is Starter and property is Vertical then matching image "Starter\_Vertical.png" and construct Image object by URI path using File object.
  2. If the type is End and property is Horizontal then matching image "End\_Horizontal.png" and construct Image object by URI path using File object.
  3. If the type is End and property is Vertical then matching image "End\_Vertical.png" and construct Image object by URI path using File object.
  4. Finally, controlling no image and no info property that If the type is anything other than specified above board cannot be generated as there is no resource for that parts and printing “Board can not be generated” message with exiting system.
* We have setDraggable method whose parameters is r type Rectangle and return type is void and accessibility is private.This method makes Rectangle moveable(draggable).Firstly in this method, detecting drag property is controlled then, source of event is gotton by "getSource" method which use to determine which tile is selected of event and stored the Rectangle inside temp.After this temp variable’s x and y coordinates(indexes) are found and stored by using Point2D which is an object property, we initiate and end dragging event by follow this path.
* We have setSlot method whose parameters is r type Rectangle and return type is void and accessibility is private. This method determines the target tile and enables the movable tiles to be replaced by finding the coordinate of the target tile. We check if the move is legal (applicable) by passing coordinates to method of Game class object game. If move is legal then we make move (swap elements) by passing coordinates to method of Game class object game.Furthermore, increasing numberOfMoves variable which is created to calculate user’s moving, then, updating the label to show our move number with moveCounter value.
* We have exchangeRectangles method whose parameters are x0,y0,x1,y1 types int and return type is void and accessibility is private.Briefly, this method provides us to change rectangles.Firstly, it removes Rectangles that will be exchanged from boardPane (object of GridPane), then, it swaps their coordinates and add to boardPane.Finally, it swap Rectangles inside the boardGrid too to make change successful.
* We have saveGame method whose parameters is empty and return type is void and accessibility is private.We created this method to save cases of levels(unlocked level).Therefore, we save unlocked level to “save.dat” file.While it is doing, an object of file "save.dat" is created and to write in save.dat file, Formatter class is used.Then, the file is closed and in case of any error happens, an error message is printed.
* We have loadGame method whose parameters is empty and return type is void and accessibility is private. We created this method to load cases of levels(unlocked level). Therefore, we save unlocked level to “save.dat” file.While it is doing, an object of file "save.dat" is created and to read save.dat file, BufferedReader class is used.For this this method firstly, reads one character which is level number but string type and it can be cast it to integer it’s string value, we used 0 value which is ineffective element and we defined this operation as integer.

# We have the displayAnimation() method, whose parameters are empty and whose return type is PathTransition, and whose accessibility is public. In this method, we put it in an unbreakable loop until we reach the end rectangle, and inside this loop we move the ball according to the type of each rectangle.

# QUESTIONS

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

All functions requested from us in our project work completely correctly.

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

During the implementation we had the biggest difficulty in animating the ball and checking if the game was over. We worked hard to get the ball to move correctly on the pipes by starting the ball from its starting position.

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

First, the login screen was added to the game and we decorated this screen with various designs, and this user can see all the levels and can enter that level when he clicks on it. In addition, we preferred to use the tiles we designed, not the tiles given to us, in order for the ball to move correctly.

# TEST CASES

This section consists of some screenshots about our game and their explanations.

**A screenshot of a computer

Description automatically generated with low confidence**

**FIRST MENU**

This image is main menu of the our game. The main menu contains 9 level buttons and when clicking on these levels, if the previous level is solved, that level is opened.

**LEVEL 1**

**Graphical user interface, application

Description automatically generated**

This image is first level of our game.

As seen in the figure, the box showing the number of levels is at the top. The movement counter and the button to return to the main menu are located at the bottom of the game.

This level is the simplest of 9 levels and can be solved with one move.

**Graphical user interface

Description automatically generated**

**LEVEL 2**

This image is second level of our game.

This level can be solved with four move.

**Graphical user interface, application

Description automatically generatedLEVEL 3**

This image is third level of our game.

This level can be solved with eight move.

**Graphical user interface

Description automatically generated**

**LEVEL 4**

This image is fourth level of our game.

This level can be solved with six move.

**OTHER LEVELS**

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated