Marmara University

Faculty of Engineering

CSE 1242.1 Computer Programming II

Spring 2023

FUN TRAVEL

Date Submitted: May 26, 2023

Problem Definition

The purpose of this project is to create a game by using JavaFX framework. The basic target of the game is to transfer passengers from one city to another one. Players get a score for every level played. It is calculated based on the distance travelled and the number of passengers moved. Highest scores are updated accordingly and displayed in the scores menu. Players can rotate the vehicle to any city they want and can see the left passengers in cities by clicking that city.

When all the passengers are moved, the next level is unlocked. Game is always saved automatically, so the player can continue to game anytime by clicking the continue button on the start menu.

Implementation Details

Before the implementation process is started, all team members gathered to decide the fundamental necessities, and tasks that must be done. After the first meeting; sketch of the project, time plan and the distribution of tasks were worked out.

To design the project process, to-do's broken into smaller pieces and deployed among the team members. At regular intervals, pieces were combined and tested. Problems were detected and tasks were reshared until everything works faultless and harmoniously.

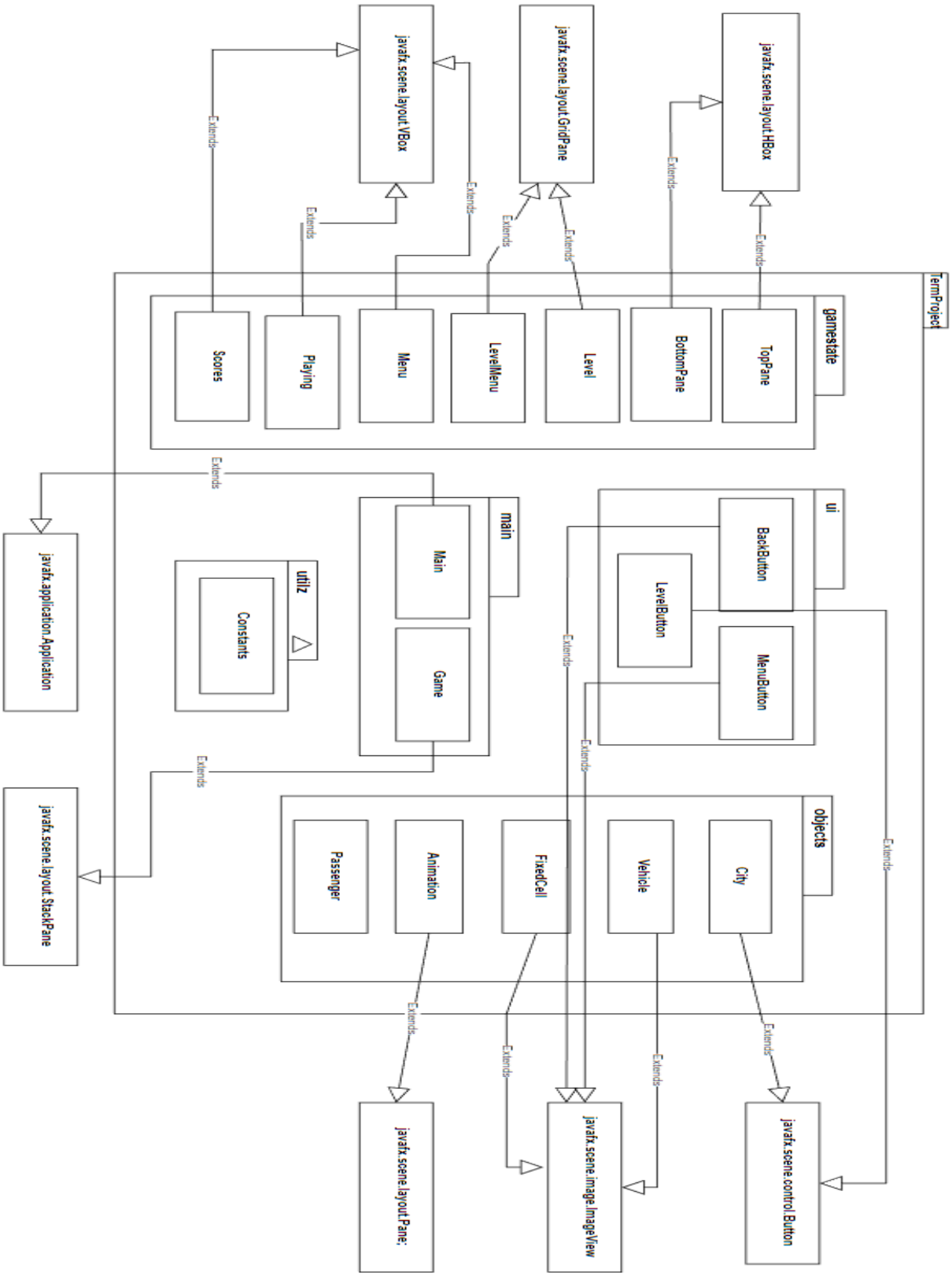
After all minimum requirements were fulfilled, remaining time has enabled us to put some additional features to the game such as saving the game and a start menu which includes 'Play', 'Continue', 'High Score', and 'Quit' buttons.

'Play' button starts the game from the beginning. 'Continue' button shows all the unlocked and locked levels so enables the player to make a decision between any unlocked levels or to continue the game where she/he left. In order to provide a continue option, game is saved automatically. 'High Score' button shows the highest scores of all the levels seperately. And finally, users can leave the game by clicking the 'Quit' button.

To make the implementation process fast and easily modifiable against unexpected problems may occur in the future, project was divided separate packages and they also were divided into several classes.

- **objects** package includes City, Vehicle, FixedCell, Passenger, and Animation classes.
- **gamestates** package includes TopPane, Level, BottomPane, Menu, LevelMenu, Playing, and Scores classes.
- **main** package includes Game, and Main classes.
- **ui** package includes BackButton, LevelButton, and MenuButton classes.
- **utilz** package includes Constants class.

General Class Diagram of the Project



main

Main	
-	xOffset : 0
-	yOffset : 0
+	main(args : String[]) : void
+	start(primaryStage : Stage) : void

Main class extends Application class. It is where new Game object is created and displayed on a scene. Main class overrides the start method of its super class. main method calls the launch method of its super class.

Game	
-	menu : Menu
-	levelMenu : LevelMenu
-	playing : Playing
-	scores : Scores
+	STATE : String
+	Game()
-	initClasses : void
+	update(STATE:String, levelNum:int) : void
+	hideAll : void
+	getMenu : Menu
+	getLevelMenu : LevelMenu
+	getPlaying : Playing
+	getScores : Scores

Game class extends StackPane class. It contains all game states such as Menu. In the constructor, background is set and initClasses and update methods are called.

initClasses method initializes menu, levelMenu, playing and scores objects. The method is passing current Game object as an argument to the constructors. Thanks to this, objects can access to the current Game object.

update method calls any game states' update method based on what STATE is. If a level is to be displayed on scene, it is called with a level number. If not level number is zero.

hideAll method sets visibility of all nodes on Game to false. This method is called in game states' update method. Therefore before displaying that game state, it hides all nodes.

objects

Animation	
-	cells : int [][]
-	level : Level
-	moves : ArrayList<Integer[]>
-	pathTransition : PathTransition
-	polyLine : PolyLine
+	Animation(level : Level)
+	drive() : void
+	isPlaying() : boolean
+	drawPath(destination : City) : void
-	createPath(destination : City) : void
-	findEmptyCell(i : int, j : int) : int[]

Animation class extends Pane class.

cells is a 2-dimensional array that stores whether a cell is empty or filled. If the cell is empty then it is 1, if not then it is 0.

polyLine shows the lines on the path.

drive method starts the PathTransition animation of car moving from a city to another.

isPlaying method returns whether animation is running or not.

drawPath method displays the path on the pane.

createPath method decides which cells should be used serially to reach the destination city and puts them in the moves ArrayList.

City	
-	name : String
-	CityID : int
-	columnIndex : int
-	rowIndex : int
-	level : Level
-	passengers : ArrayList<Passenger>
+	City(level : Level, name : String, cellID : int, cityID : int)
-	updateInfo() : String
-	calculateDistance(curretCity : City, destCity : City) : int
-	setImage() : void
+	addPassenger(passenger : Passenger) : void
+	getPassengers() : ArrayList<Passenger>
+	getColumnIndex() : int
+	getRowIndex() : int
+	getCityID() : int

City class extends Button class.

columnIndex and rowIndex identify city's location on the coordinate system. Every City object has an ArrayList of Passenger objects. It holds the information of passengers located in that city.

updateInfo method updates the information on the bottom pane when a City button is clicked.

calculateDistance method takes two City objects. It calculates the euclidian distance between them by using their rowIndex and columnIndex properties and returns the ceiling of it.

setImage method assigns a random city image to the City button and puts city name on the top of it.

addPassenger method adds the given Passenger object to the passengers ArrayList.

FixedCell	
-	columnIndex : int
-	rowIndex : int
+	FixedCell(cellID : int)
+	getColumnIndex() : int
+	getRowIndex() : int

FixedCell class extends ImageView. It represents an off road cell.

columnIndex and rowIndex represent the location of the fixed cell on the coordinate system.

Passenger	
-	numOfPassengers : int
-	destCity : City
+	Passenger(numOfPassengers: int)
+	getNumOfPassengers() : int
+	setNumOfPassengers(numOfPassengers : int) : int
+	getDestination() : City
+	setDestination(destCity : City) : void

numOfPassengers represents the number of passengers the Passenger object has.

destCity represents the destination city of the Passenger object.

Vehicle	
-	capacity : int
-	MAX_CAPACITY : int
-	destination : City
-	location : City
+	Vehicle(location : City, maxCapacity: int)
-	setImage() : void
+	updateLocation() : void
+	setDestination(destination : City) : void
+	getCapacity() : int
+	getMaxCapacity() : int
+	setCapacity(capacity : int) : void
+	getDestination() : City
+	getLocation() : City

Vehicle class extends ImageView.

MAX_CAPACITY represent the number of seats in the vehicle. capacity represents the current empty seats in the vehicle. At the beginning it is equal to the MAX_CAPACITY. location represents the current location and destination represents the destination location of the vehicle.

setImage method set the vehicle image according to its capacity. (Less than 6 car; less than 14 minubus; greater than 13 bus)

updateLocation method updates the location to the destination city.

gamestates

BottomPane	
-	playing : Playing
-	drive : Button
-	info : Label
+	BottomPane(playing:Playing)
+	setInfo(info:String) : void

BottomPane extends HBox class. It contains drive button and displays information about passengers when a city is clicked.

Level	
+	cells : int[][]
-	unlocked : boolean
-	score : int
-	levelNum : int
-	cities : ArrayList<City>
-	fixedCells : ArrayList<FixedCell>
-	animation : Animation
-	playing : Playing
+	Level(levelNum:int, unlocked:boolean, playing:Playing)
-	loadLevelFromFile : void
+	loadLevelToPane : void
+	movePassengers : void
+	calculateScore : void
-	calculateDistance(currentCity:City, destCity:City) : int
+	checkCompleted: boolean
+	updateHighestScore : void
+	setUnlocked(unlocked:boolean) : void
+	isUnlocked : boolean
+	getScore : int
+	getVehicle : Vehicle
+	getAnimation : Animation
+	getPlaying : Playing

Level class extends GridPane class. It represents the level itself. It has vehicle, cities, fixed cells and animation.

cells is a 2 dimensional array that stores whether a cell is empty or filled. If the cell is empty then it is 1, if not then it is 0.

In the constructor, loadLevelFromFile and loadLevelToPane methods are called. loadLevelFromFile method loads the level from a txt file. It creates objects according to the file. loadLevelToPane method adds created objects to the pane.

movePassengers method moves passengers from one city to another. It is called when drive button is pressed.

checkCompleted method checks if a level is completed. The method iterates through the cities array list and checks if any passenger is left in cities. If the level is completed updateHighestScore method is called. If new score is greater than the highest score of that level, the method updates the highest scores file.

LevelMenu	
-	buttons : LevelButton[]
-	backButton : BackButton
-	game : Game
+	LevelMenu(game:Game)
-	loadButtons : void
-	setEventHandlers : void
+	update : void

LevelMenu extends GridPane class. It enables the user to choose a level between unlocked levels to play.

Menu	
-	buttons : MenuButton[]
-	game : Game
+	Menu(game:Game)
-	loadButtons : void
-	setEventHandlers : void
+	update : void

Menu extends VBox class. It has four buttons. PLAY, CONTINUE, SCORES and QUIT. setEventHandlers method sets event handlers for each button according to their purpose.

Playing	
+	currentLevelNum : int
-	levelSave : int
-	numOfLevels : int
-	game : Game
-	topPane : TopPane
-	bottomPane : BottomPane
+	levels : ArrayList<Level>
+	Playing(game:Game)
-	loadAllLevels : void
+	update(levelNum:int) : void
+	deleteSave : void
-	updateSave : void
+	drive : void
+	getBottomPane : BottomPane

Playing class extends VBox class. It contains top pane, bottom pane and level. In the constructor loadAllLevels method is called. This method creates all levels in the levels folder and add them to levels array list.

update method loads the level with given level number to the pane.

deleteSave method is called when user starts a new game. The method locks all levels and updates level-save.txt to 1.

updateSave method updates level-save.txt file. It is called in update method when a new level is unlocked.

drive method starts the car animation and calls all other necessary methods. It is defined here because it is easier to access all objects from Playing class.

Scores	
-	game: Game
-	backButton : BackButton
-	text : String
-	highScores : Label
+	Scores(game: Game)
+	update : void

Scores class extends VBox class. It displays the highest score of all levels. To store the score records permanently, they are written to a file (highest-scores.txt). Initially, they all are 0 and anytime a better score is recorded, highest-score.txt file is updated with the new value. Since the initial content of the file depends on the number of levels when it is created, anytime a new level is added to the game, highest-score.txt must be arranged accordingly. (Content of the highest-scores.txt file can be found in the appendix.)

TopPane	
-	levelNumLbl : Label
-	scoreLbl : Label
-	nextLevel : Label
+	TopPane(playing: Playing)
+	setLevelNum(levelNum:int) : void
+	setScore(score:int) : void
+	getNextLevel : Label

TopPane class extends HBox. It displays current level and score. Also contains next level button. setLevelNum and setScore methods are used for updating score and level number. getNextLevel method is used for accessing nextLevel label and updating its color if next level is unlocked.

ui

BackButton	
-	images : WritableImage[]
+	BackButton()
-	loadImages : void

BackButton class extends ImageView class. When it is clicked it goes back to the previous menu.

loadImages method loads a button atlas and using PixelReader class crops button atlas into 3 images. These images are assigned to images array. If a mouse enters, exits or presses on back button, its image changes. Thanks to this, user interface becomes more interactive.

LevelButton	
-	images : WritableImage[]
-	levels : ArrayList<Level>
-	levelNum : int
+	LevelButton(levelNum:int)
-	loadImages : void
+	update : void
-	setBackground(image:WritableImage) : void
+	setLevels(levels:ArrayList<Level>) : void
+	getLevels : ArrayList<Level>

LevelButton class extends Button class. When it is clicked it loads the desired level.

loadImages method has the same logic as BackButton class's loadImages method. If a level is unlocked, level button's background image is set to second image which looks like an inactive button. Level button's background image is updated every time level menu is displayed.

MenuButton	
-	images : WritableImage[]
-	rowIndex : int
+	MenuButton(rowIndex: int)
-	loadImages : void

MenuButton class extends ImageView class. When it is clicked, it loads desired pane on to the scene or exits the game.

loadImages method has the same logic as BackButton class's loadImages method. But it loads images based on the row index of button. For example if row index is 0, then first row of button-atlas.png is loaded which is PLAY.

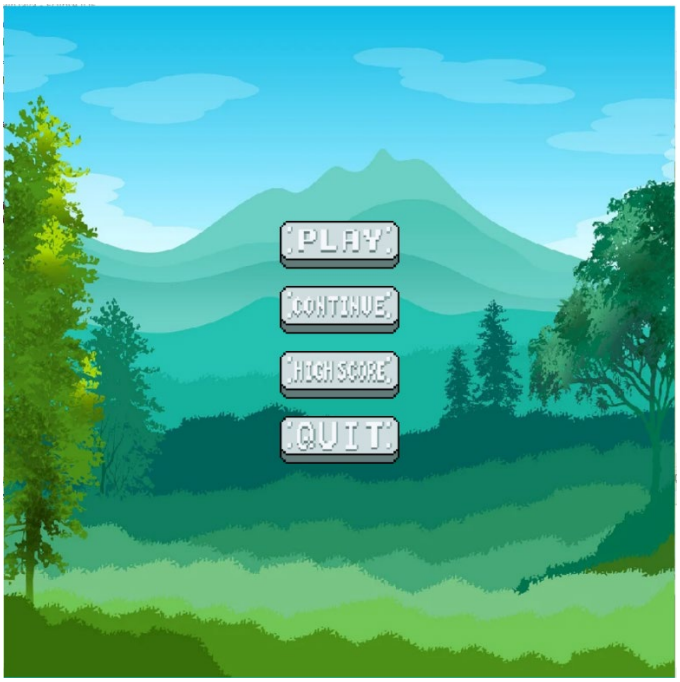
utilz

Constants	
+	SCALE : float
+	CELL_DEFAULT_SIZE : int
+	PADDING_DEFAULT : int
+	PADDING : int
+	CELL_SIZE : int
+	GAME_WIDTH : int
+	GAME_HEIGHT : int
+	LEVEL_FOLDER : String
+	LEVEL_FILE : String
+	LEVEL_FILE_EXTENSION : String
+	FIXED_CELL_IMAGE : String
+	FIXED_CELL_IMAGE_WIDTH : String
+	ROAD_IMAGE : String
+	ROAD_IMAGE_WIDTH : int
+	CITY_FILE : String
+	CITY_FILE_EXTENSION : String
+	CITY_IMAGE_WIDTH : String
+	CAR_FILE : String
+	CAR_FILE_EXTENSION : String
+	CAR_IMAGE_WIDTH : int
+	CAR_IMAGE_HEIGHT : int
+	MENU_BACKGROUND
+	MENU_BACKGROUND_WIDTH : String
+	MENU_BUTTONS : String
+	B_WIDTH_DEFAULT : int
+	B_HEIGHT_DEFAULT : int
+	LEVEL_BUTTONS : String
+	LEVEL_BUTTON_WIDTH : int
+	BACK_BUTTON : String
+	BACK_BUTTON_WIDTH : int
+	BACK_BUTTON_HEIGHT : int
+	LEVEL_SAVE : String
+	HIGHEST_SCORE : String
+	FONT : Font
+	CITY_FONT : Font
+	INFO_FONT : Font
+	HIHSCORES_FONT : Font

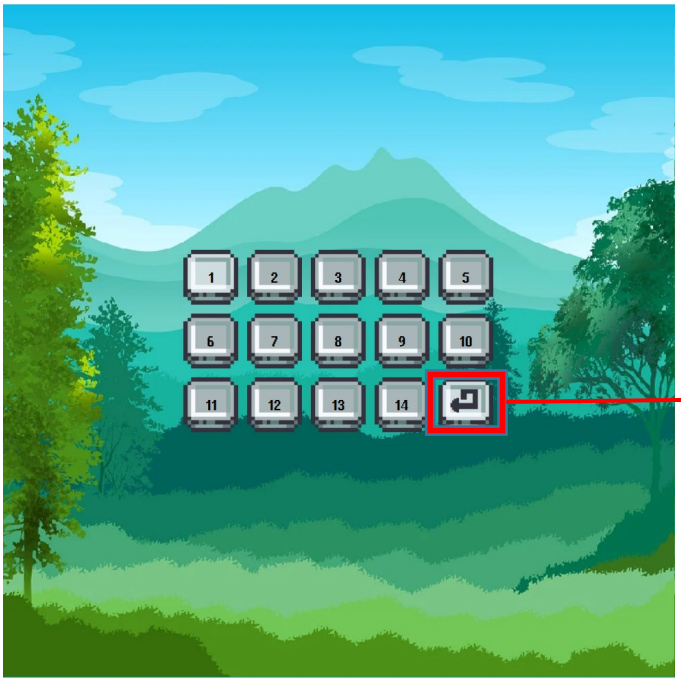
Constants class contains all constants used in the game such as image dimensions and file paths. We implemented this class to make changing things easier. All data fields have initial values but they are omitted in the UML diagram for brevity.

Test Cases

Start Menu

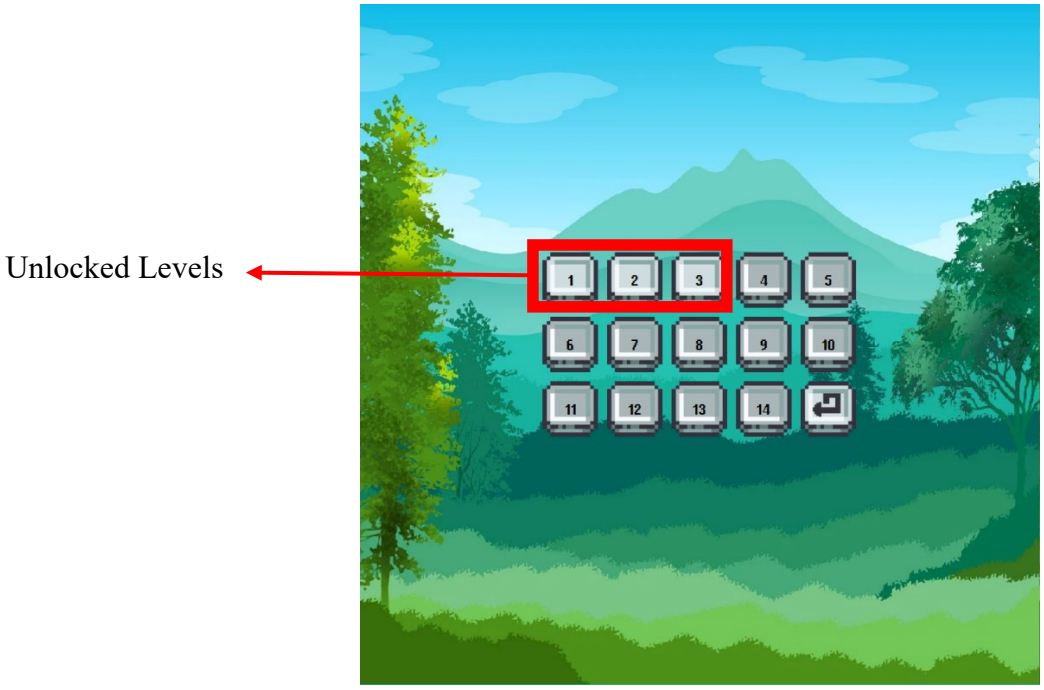


Levels Menu After Clicking Play Button



After play button is clicked, a new game is created. So all levels except level 1 is locked. There are 14 levels in levels folder so 14 level button is created.

Levels Menu After Clicking Continue Button

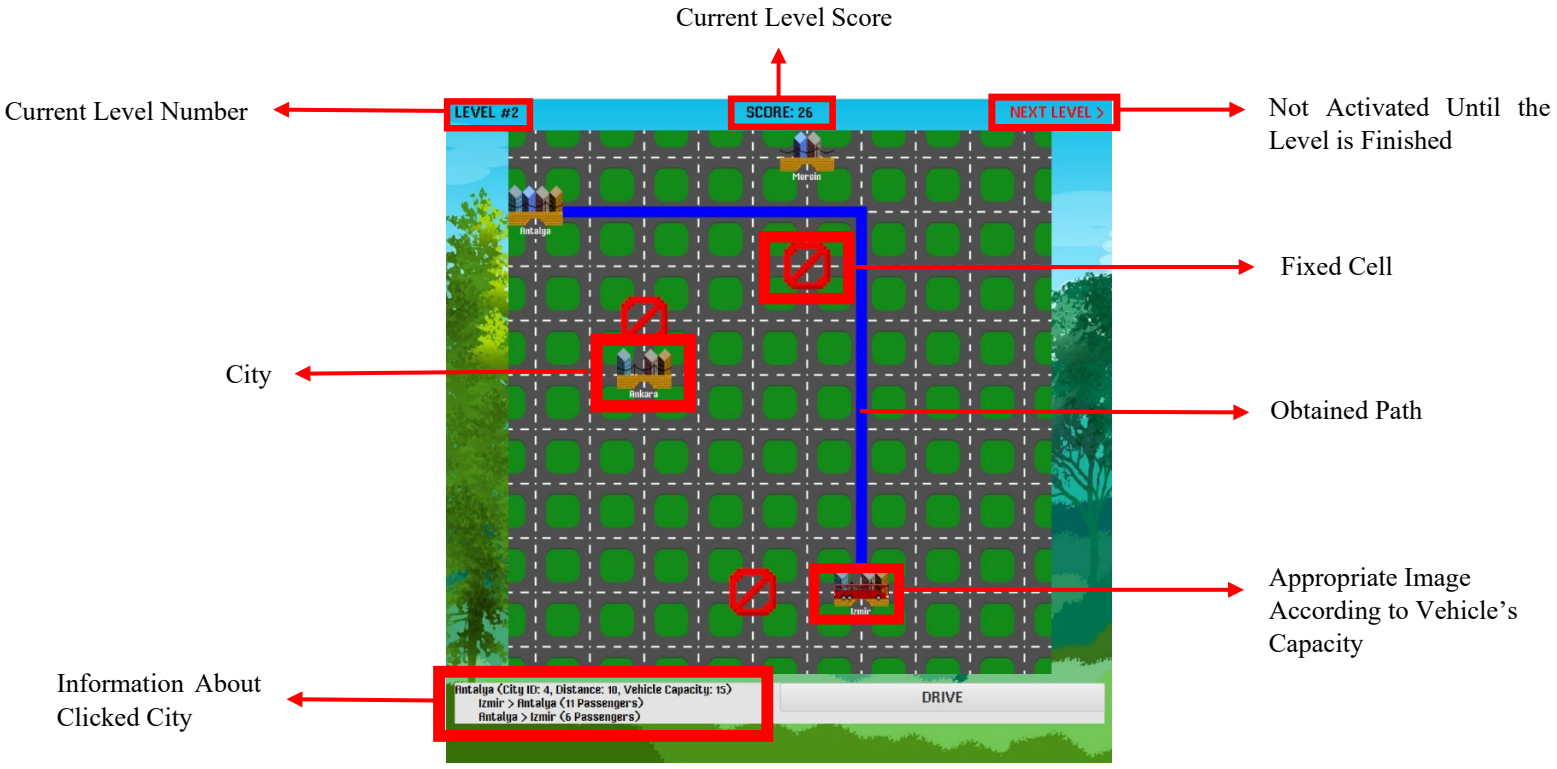


After continue button is clicked, program reads level-save.txt file and checks last unlocked level. Here, last unlocked level is level 3.

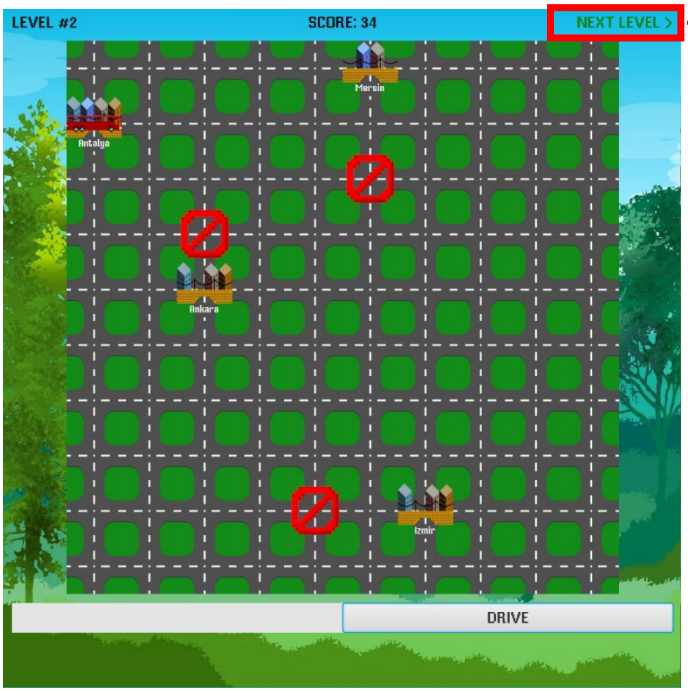
Scores Menu After Clicking High Score Button

Top scores for each level is displayed. There are 14 levels in levels folder so highest-score.txt contains 14 level scores.

Unfinished Level

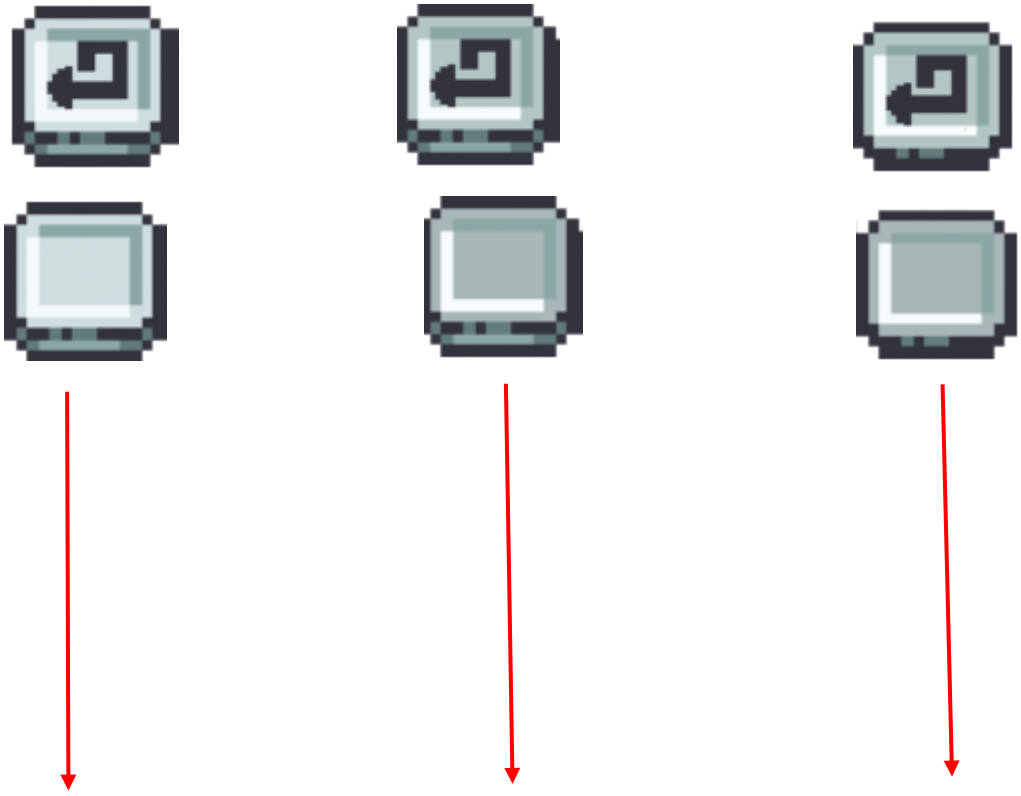


Finished Level



Level is complete so next level button is activated.

Button Atlas



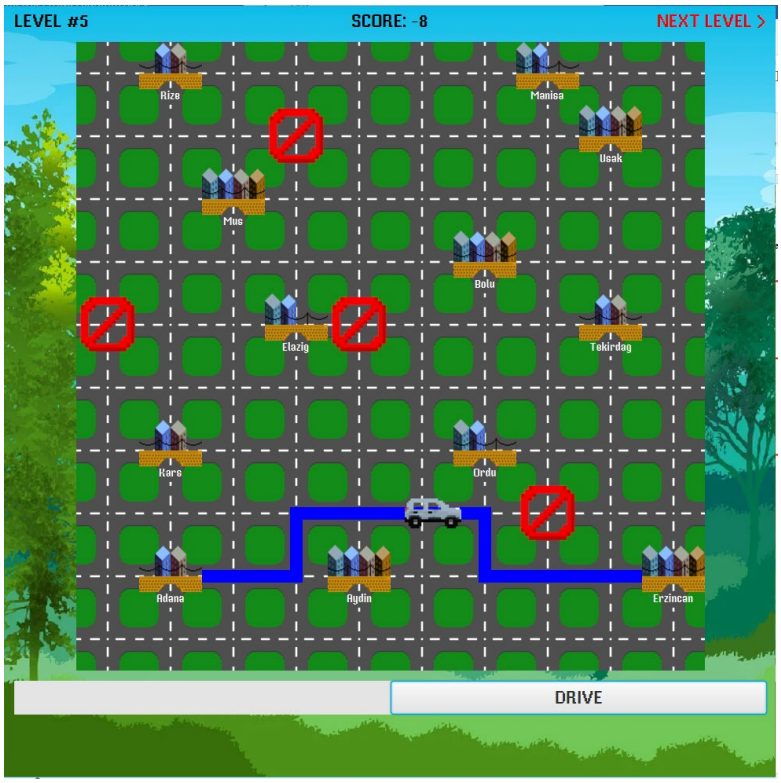
Normally

When Mouse Cursor is Over

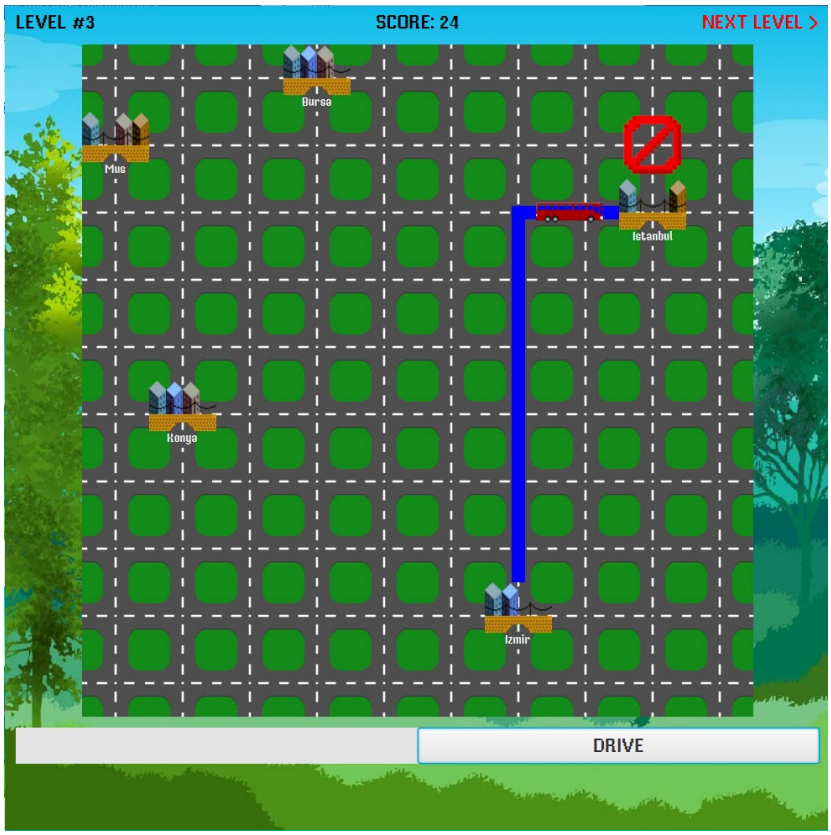
When it is Clicked

Different Vehicles for Different Capacities

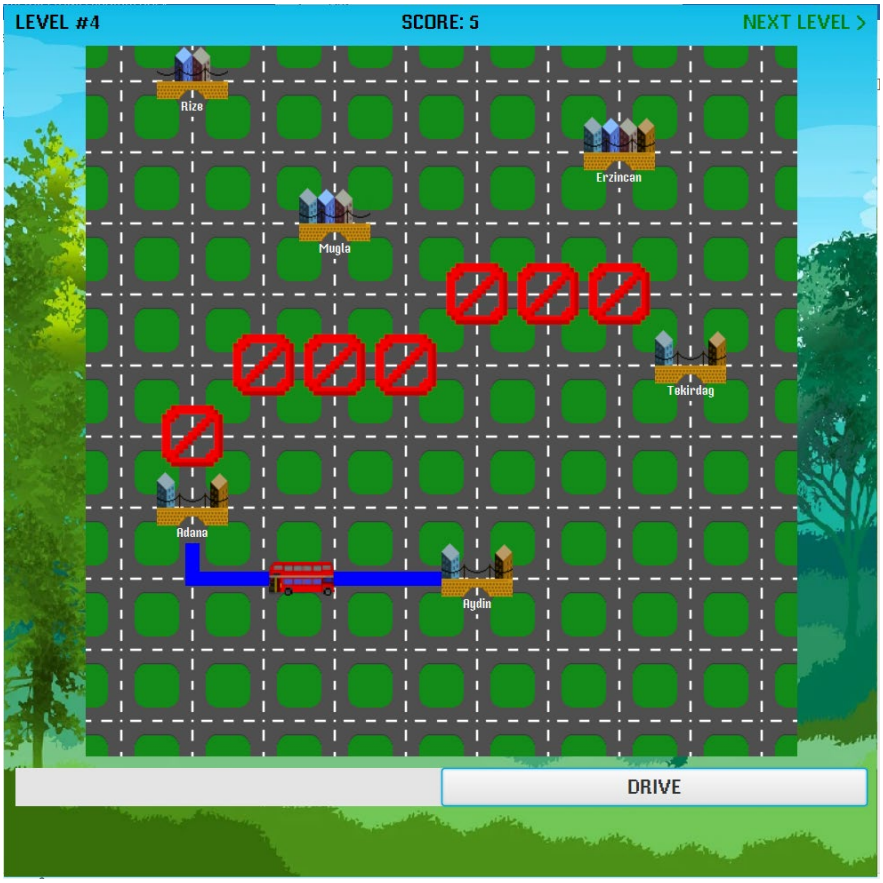
1. Car



2. Minibus



3. Bus



Appendixes

Initial content of highest-score.txt for a 14 levels game

LEVEL 1 : 0
LEVEL 2 : 0
LEVEL 3 : 0
LEVEL 4 : 0
LEVEL 5 : 0
LEVEL 6 : 0
LEVEL 7 : 0
LEVEL 8 : 0
LEVEL 9 : 0
LEVEL 10 : 0
LEVEL 11 : 0
LEVEL 12 : 0
LEVEL 13 : 0
LEVEL 14 : 0

Initial content of level-save.txt after finishing 3 levels

4