

***MARMARA UNIVERSITY***

***FACULTY OF ENGINEERING***

**CSE1242 Computer Programming II (Spring 2024)**

**Term Project**

**TRAFFIC CONTROL SIMULATOR**

**Due: 10.05.2024 / 23:59**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Department** | **Student ID** | **Name & Surname** |
| **1** | Computer Engineering | 150122065 | Zehra Özlen |
| **2** | Computer Engineering | 150123035 | Canan Pazarbaşı |
| **3** | Computer Engineering | 150122015 | Ayşe Yıldız |

**1)PROBLEM DEFINITION:**

We created a traffic game for this term project that we wrote together with our group members. We can briefly explain the purpose of the game as follows. First, we have a starting screen, and when the player clicks on the start button on the screen, a screen with levels to choose appears. Then, the player selects a desired level, and the game screen appears containing the data of the txt files we added for the levels.

In order to win the game, the number of cars specified in the txt files depending on the level must go from their starting points to certain destinations. The player can do this by playing with traffic lights. The opposite situation also exists. Car collisions are another possibility if the traffic light color is changed at the wrong place or at the wrong time. These collisions also have a certain value taken from the file, and if that maximum value is reached, the game is lost.

**2)IMPLEMENTATION DETAILS:**

|  |
| --- |
| Building |
| - buildingColor: Color[] |
| + createBuilding(p: Pane, buildingType: int, buildingRotation: int, colorIndex: int, cellX: int, cellY: int): void |

In the Building class we created, we created an array containing four specific colors for our buildings and created the createBuilding method to adjust their shape. We had three cases in the switch case section of this method because we took the shape and location of the buildings from the txt file. The Building class was a class that we created separately from the Grid class to create the visibility of our buildings, and it has an important place for the game appearance.

|  |
| --- |
| Car |
| + createCar(x:double, y:double):Pane |

In our other class, car, the first part contains the ArrayList we created for cars. This part is very important as it allows us to keep the data of the cars we create. Afterwards, two of our variables stand out. Our purpose in creating the static variables carX and carY was to be able to track the locations of the cars. Next, there is our createCar method, which appeals to a more visual part. In this method, we created the exterior appearance of the cars we will see on the panel according to our own taste.

|  |
| --- |
| PathWay |
| +pathList: ArrayList<Path> |
| + createPath(p:Pane, pathIndex:int, pathType: String, x:double, y:double):void |

As we mentioned in the previous two classes, the PathWay class also contains visual data. First, let me point out our ArrayLists. We created them to hold the pathList, pathX and pathY variables, and then we wrote the createPath method for the data to be read from the file to see our paths in the main pane. This method contains a switch case part. This part was created for two possibilities. For MoveTo and LineTo elements. The PathWay class also serves as the base class for our Test class.

|  |
| --- |
| RoadTile |
| + createRoadTile(p:Pane, roadType: int, roadRotation: int, cellX:int, cellY: int):void |

The topic we cover in the RoadTile class is not much different from the other classes we mentioned. Again, we wrote a method that allows the creation of roads with different characteristics so that the roads on which the cars can move are displayed on the screen. The name of this method is createRoadTile. The method includes the switch case section, which is written to read the data of the file given to us and create paths in appropriate shapes and directions. Roads with four different features are handled under different conditions in each case, and the last part includes the part where they are added to the panel.

|  |
| --- |
| TrafficLight |
| + isRed: boolean  + trafficLights: List<Circle> |
| + createLight(p:Pane, x1: double, y1: double, x2: double, y2:double):void |

The last base class we wrote, which is required for the Game class, is the TrafficLight class, where we control the traffic lights. We used boolean because we had two possibilities (red or green) as the light color. Depending on whether the isRed variable is true or false, we were able to adjust the movements of the cars, which we will talk about in the next class. In this class, we also defined an Array List for trafficLights and then wrote the createLight method, which allows us to create the shape of the traffic lights and their color change according to the boolean state when clicked. In the last part of the code, there is a section to add them to the pane.

|  |
| --- |
| Game |
| - mainPane: Pane  - winCount:int  - requiredWin:int  - transitionList: List<PathTransition>  - collisionCount:int  - maxCollision:int  - currentLev:int  - collisionLabel:Label  - gameOverLabel:Label  - winLabel:Label  + paneWidth:double  + paneHeight:double  + rows:int  + columns:int  + cellWidth:double  + cellHeight:double  + spawnTime:double |
| + start(primaryStage:Stage):void  - createLevelButtons(mainPane:Pane, coverImageView:ImageView):VBox  - loadLevel(lev:int,mainPane:Pane, coverImageView:ImageView, levelButtons:VBox):void  - createObject(com:String, mainPane:Pane):void  - loadNextLevel(mainPane:Pane, coverImageView:ImageView, levelButtons:VBox):void  - back(mainPane:Pane, coverImageView:ImageView, levelButtons:VBox):void  - createTraffic(mainPane:Pane, coverImageView:ImageView, levelButtons:VBox):void  - spawnCar(mainPane:Pane, coverImageView:ImageView, levelButtons:VBox):void  - checkLights(pt:PathTransition):void  - checkCollisions():void  - updateCollisionCounter():void  - updateWinLabel(coverImageView:ImageView, levelButtons:VBox):void  - stopCarAnimations():void  - removeCar(carPane:Pane):void  - exit():void |

In the Game class, which is an extension of the Application class and contains the start and main method, where all data and classes are collected and combined, the general image is completely created, and the steps of the game seem to be more clear. The variables we specified in the UML diagram are in the first part, before the start method. Here their values ​​are initialized. Then, arrangements are made for the first screen that will appear before the player in the start method. For example, the name of the game, the login screen, and the buttons on it, etc. While doing these, we took a jpg from our own directory for the image and imported the necessary parts for the buttons or shapes. We wrote the createLevelButtons method using VBox to add level buttons that the player can select and play the desired level after the login screen. In this method, depending on the selected level, the relevant txt files are opened, and the game interface is created.

We have created two more extra buttons. These are used optionally. One is the back button, and the other is the exit button. Another method in the UML diagram and our code, loadLevel, reads the data from the file and in the last part, the createTraffic method created to activate the game is called. In the createObject method, the data taken from the file is split and the relevant sections are created by reading all the data sequentially, starting from index 0. That's why this method is of crucial importance. The loadNextLevel method that we see is a method we wrote to run after creating the nextLevel button. Five different levels in the project are loaded here in the if else section, and after they are completed, it contains a message informing that the levels are completed.

We used AnimationTimer while writing the createTraffic method, which controls the actual movement of the cars. In the handle method within this method, we determined the frequency of cars coming out and the probability of them coming out, and in the last part, we added the checkLights, which controls the lights, and the checkCollisions method, which includes the control mechanism for cars hitting each other, to call them. In the spawnCar method, which is of great importance in the game, we created randomIndex and randomPath variables by taking the coordinates of the cars from the List. Here is the section where we add them to the mainPane, where we make sure that these are added to the screen, and we create a counter for the cars going to the destination required for the cars to win.

We mentioned before that we call the checkLights method that follows. Here, the distance of the cars from the traffic lights and, of course, their stopping and moving conditions are adjusted according to the traffic lights’ color. Again, we encounter the for loop in checkCollisions, which is a method we called before. Within them, we assign the transitionList of the ArrayList to the pt variables, and then we enable them to control each other, similar to cars controlling traffic lights. However, in some cases, cars collide, which causes the collision count counter to increase, as we saw in the else section. To make this part concrete, we added gameOverLabel and create the screen where the player can see that he lost when the maximum collision is reached. In the last part of this method, we inform the player by showing a concrete congratulations message on a blank black screen when all levels are completed.

The updateCollisionCounter and updateWinLabel methods, which are two methods that contribute to the layout of the game screen and write the winning and losing values ​​of the cars, contain code pieces that enable the variable values ​​​​due to the movement of the cars to be displayed on the upper left. The removeCar method simply makes the colliding cars that come together become invisible and disappear after a certain period of time. In the stopAnimations method located at the last part of the Game class, the Array List we created before is cleared in the called places, and with the exit method, the game is completely exited by pressing the exit button.

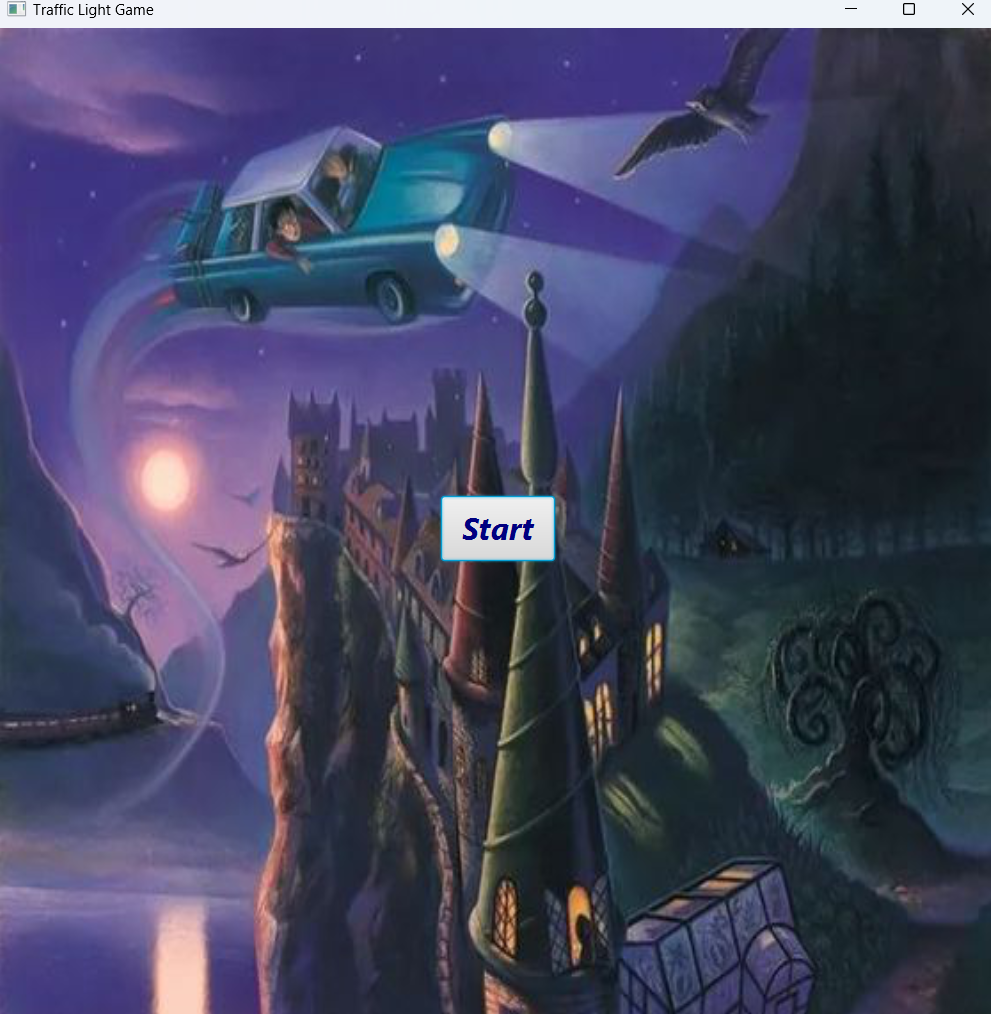
Even though we hope that all the codes we wrote will run the game correctly, we occasionally encountered problems after running it, which we checked with a lot of effort to fix but could not have fully solved.

One of these was due to the buttons we added. Although the Next Level button worked quite normally, as the levels changed according to the file txt, there were disruptions in the game algorithm that we could not understand. This situation also occurred in the back button.

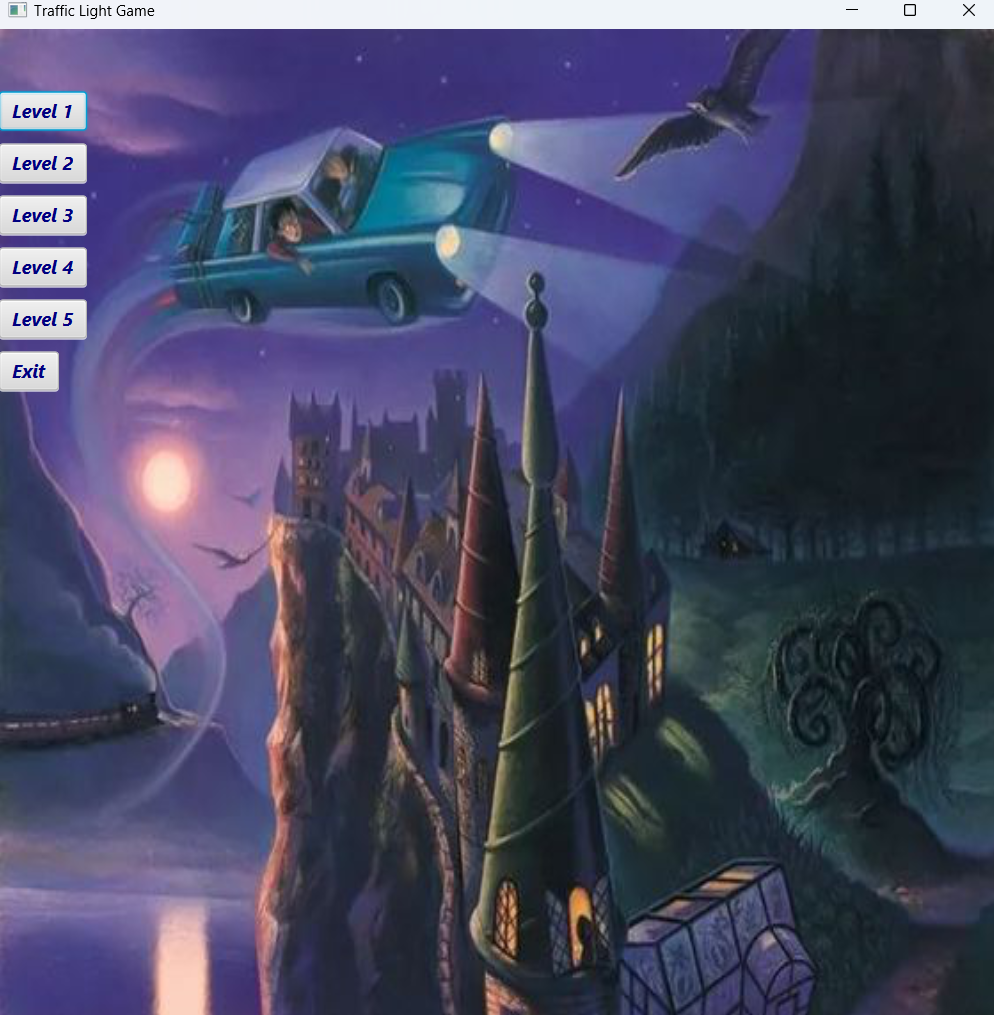
Another problem we encountered was car movements in a specific situation. Our cars, which can perceive the colors of the lights (red or green), continued regardless of the red traffic light in the parts where they could not control their distance between the other cars. Although we think that the reason for this is that we adjust the distance of the cars that spawn one after the other and the cars that spawn after them, do not comply with this condition and leave, we could not make this concrete.

The part that bothered us the most in the implementation was trying to set the correct working mechanism of the counters, but this mechanism only works very well when the game is turned on and off, that is, when the level is selected one by one, but a problem occurs in the algorithm of the counters with the next level button.

**3)TEST CASES:**

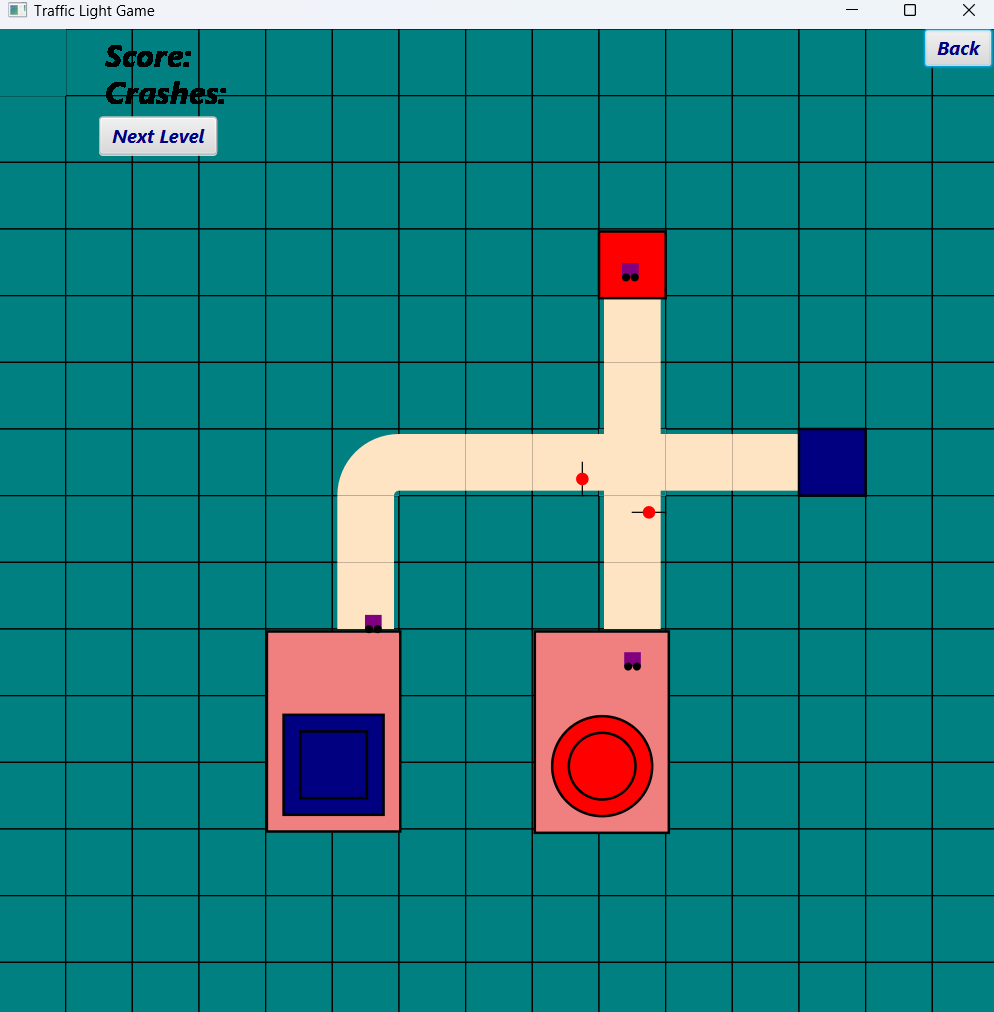


Our game starts with this screen. When the player presses the “Start” button, the screen where the gamer can select the levels appears.



On this screen, player chooses the level. Then our levels will create and appear according to the data in the txt file and displayed on the screen.

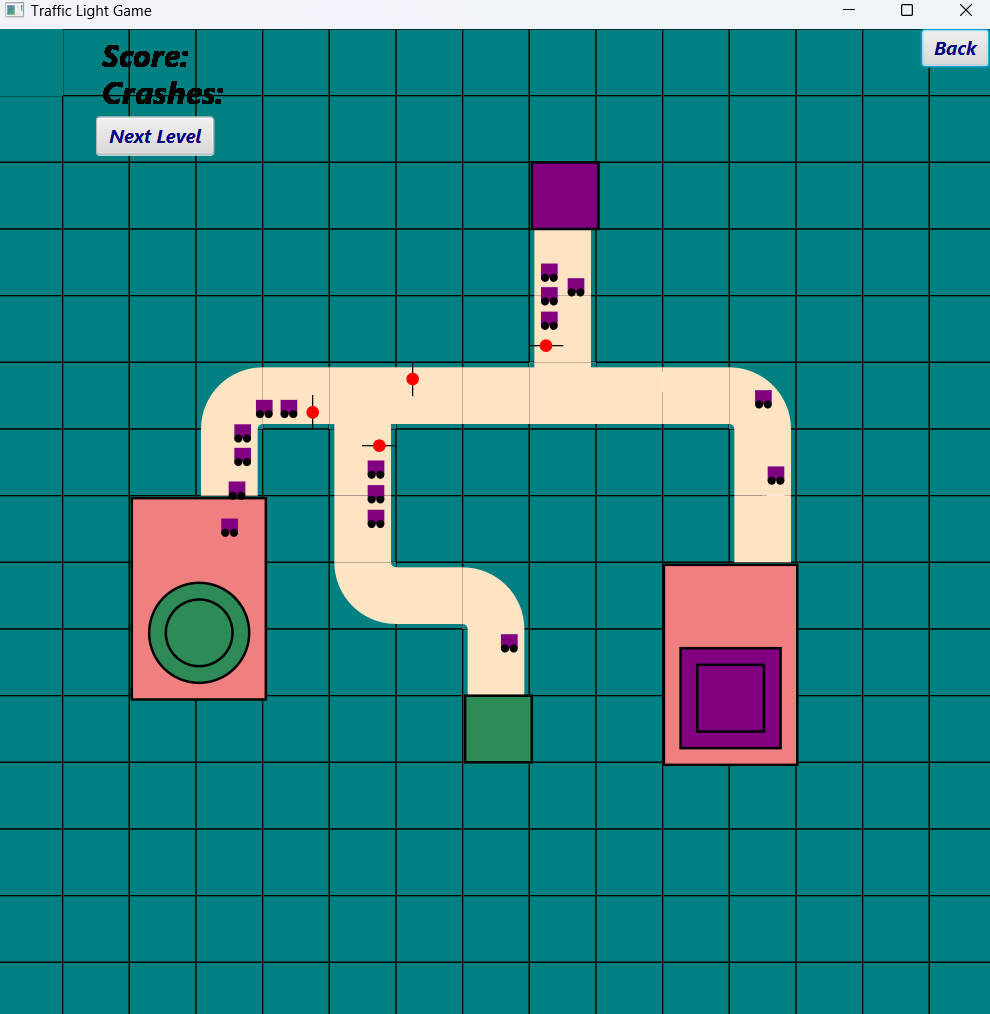
When the player clicks on the “Exit” button, the game is exited.



In this picture we see level 1.

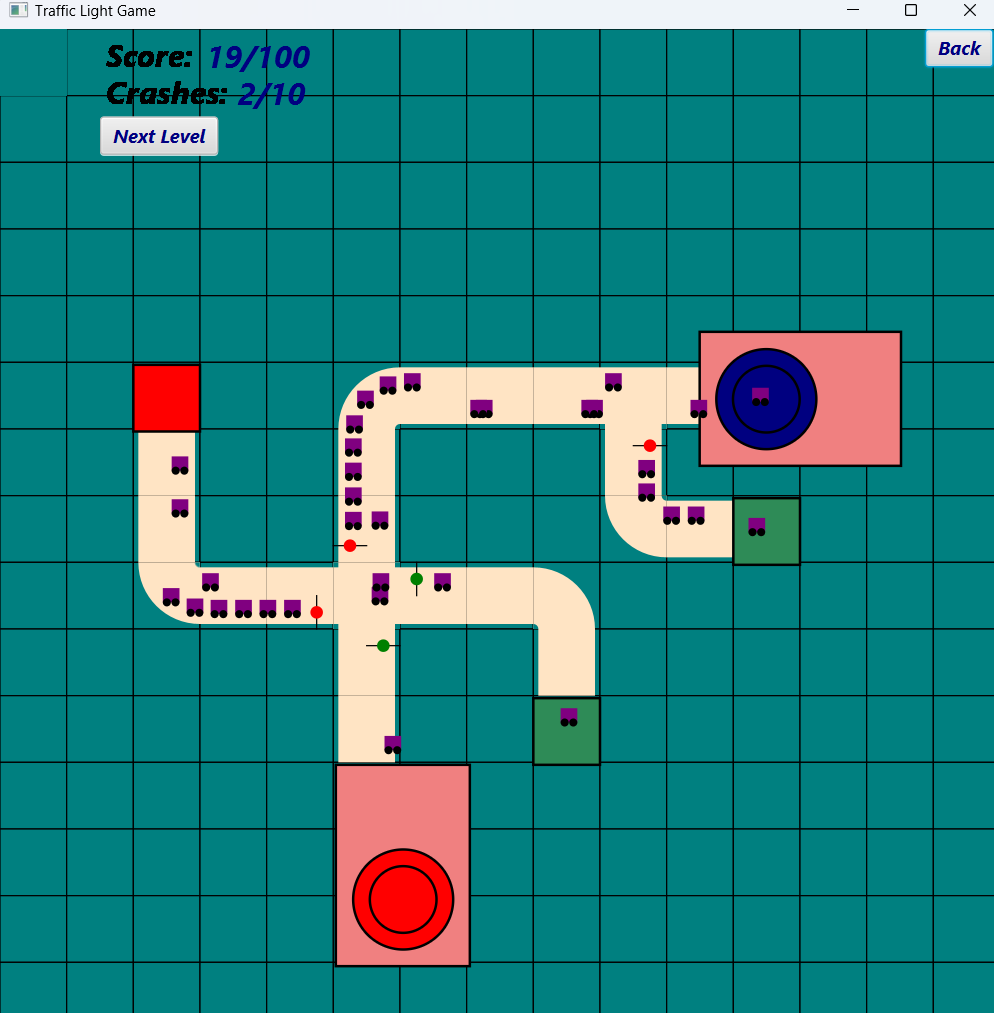
We can move on to the next level with the “Next Level” button on the screen.

We can return to the screen where we selected the levels with the “Back” button.



In this picture we see level 2.

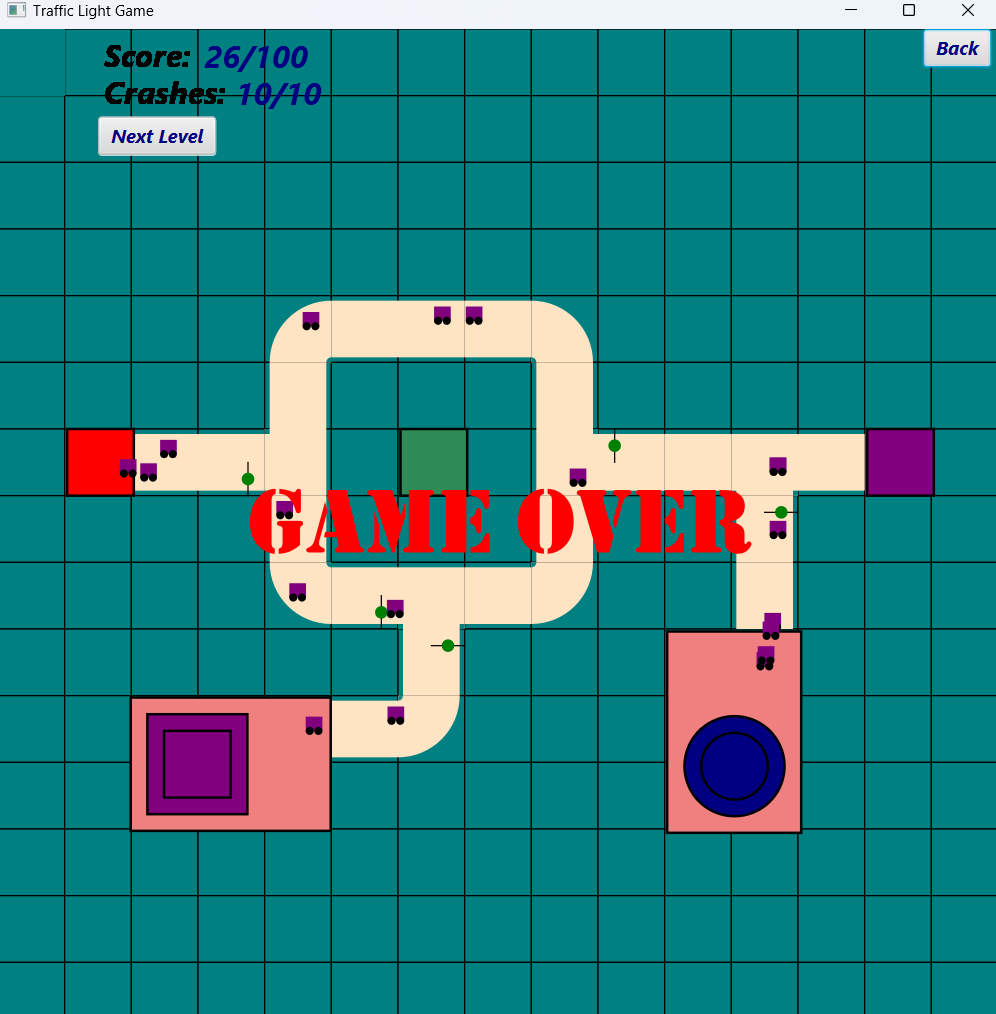
Our cars stop when the red light is on. When the player clicks on the lights and turns them green, the cars continue on their way.



In this picture we see level 3.

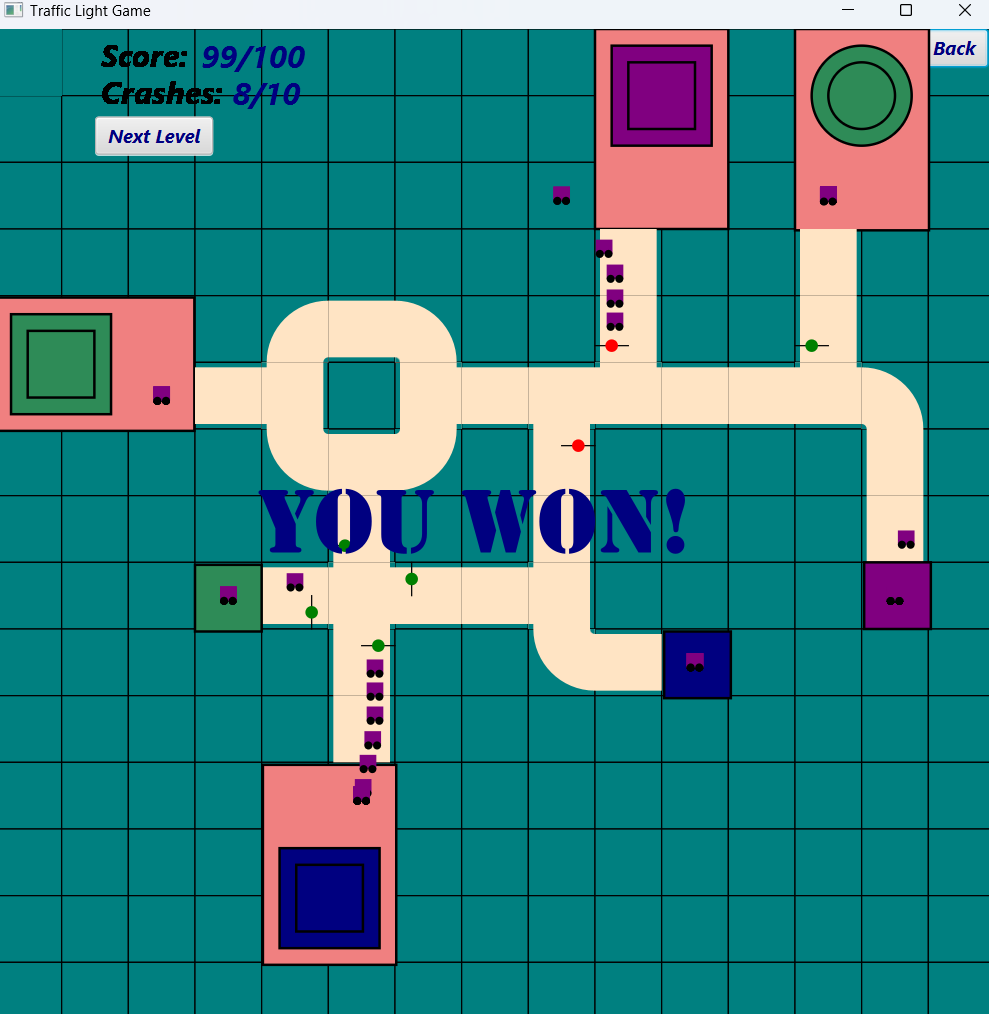
The crashes count inc-reases as our cars collide.

When the cars finish the road, the score count increases.



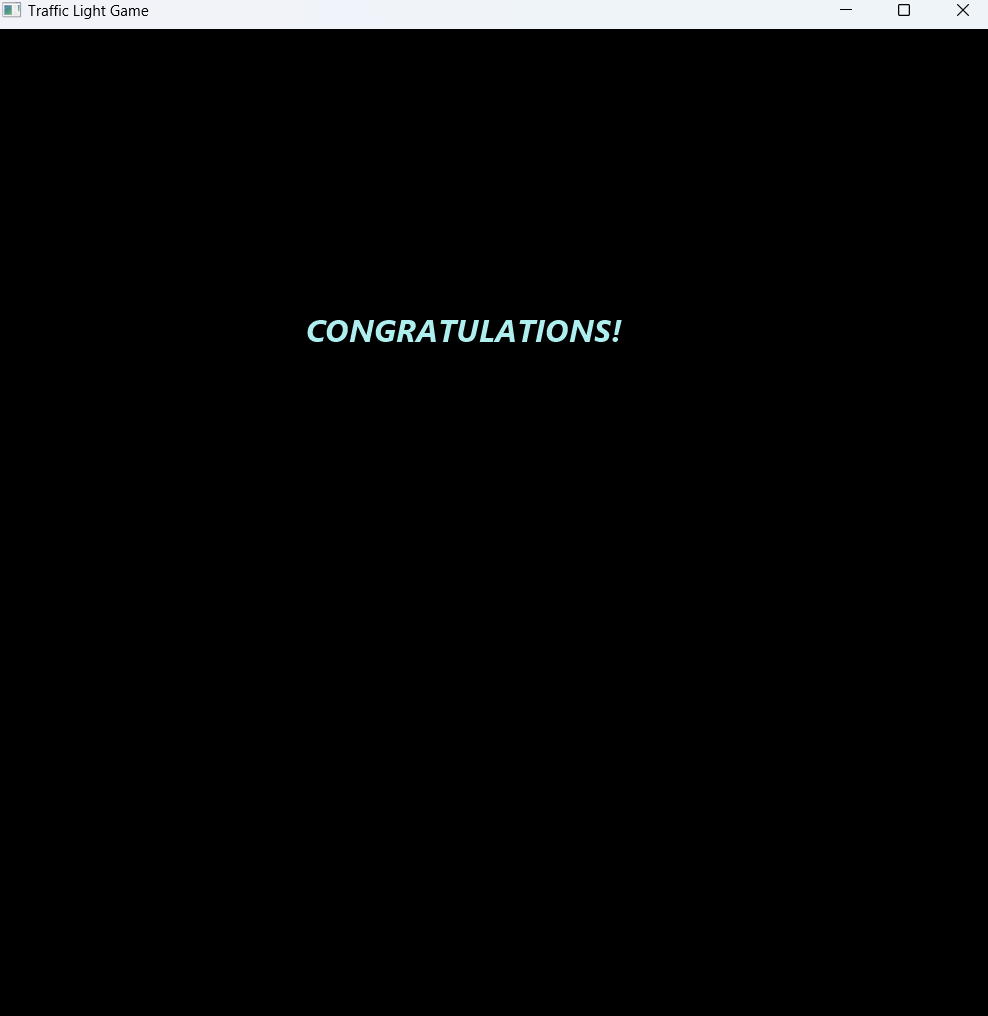
In this picture we see level 4.

When the number of crashes reaches its maximum number, the player loses the game and the word "GAME OVER" appears on the screen.



In this picture we see level 5.

When the number of scores reaches its maximum number, the player wins the game and the text "YOU WON" appears on the screen.



This last screen appears when the player comp-letes all the levels and our game ends here.