BZ 214 Visual Programming Project Report

Group No: 64 **29/05/25**

Number: *1030510953*	**Name Surname: Türker Kıvılcım**
Created the model classes. Defined the behaviors of vehicles and traffic lights. Developed the algorithms for calculating traffic light durations.	
Number: *1030510877*	**Name Surname: Seren Kaya**
Connected model with view with TrafficController class. Made to show appropiate data on view and sends user inputs to model.	
Number: *1030510932*	**Name Surname: Furkan Mete Çalış**
Designed the user interface using SceneBuilder. Integrated the UI with the controller and model layers.	

Abstract

In this project, a traffic simulation that optimizes traffic lights based on traffic density was developed using Java and JavaFX. The duration of the traffic lights was automatically adjusted according to the vehicle density at intersections. Counters placed above the traffic lights displayed the remaining time for each light. The system was controlled by buttons that allowed the user to start, pause, and reset the simulation in two different modes.

Software Design

The project was developed in accordance with the Model View Controller (MVC) architectural pattern. Accordingly, responsibilities were distributed among team members. All code was managed professionally via GitHub branches. Classes were functionally separated, with each class customized to handle a specific task. This structure ensured that the code remained sustainable and flexible.

- Use Case

Vehicles can be generated in two different ways: Randomly or Based on user input.

Visuals of the vehicles were added to the project to provide a more realistic appearance. Turning animations enabled vehicles to move in a more natural way. Also The **Start** button automatically starts the simulation. The **Reset** button resets the entire system to its original state. The **Pause** button halts animations and the traffic cycle.

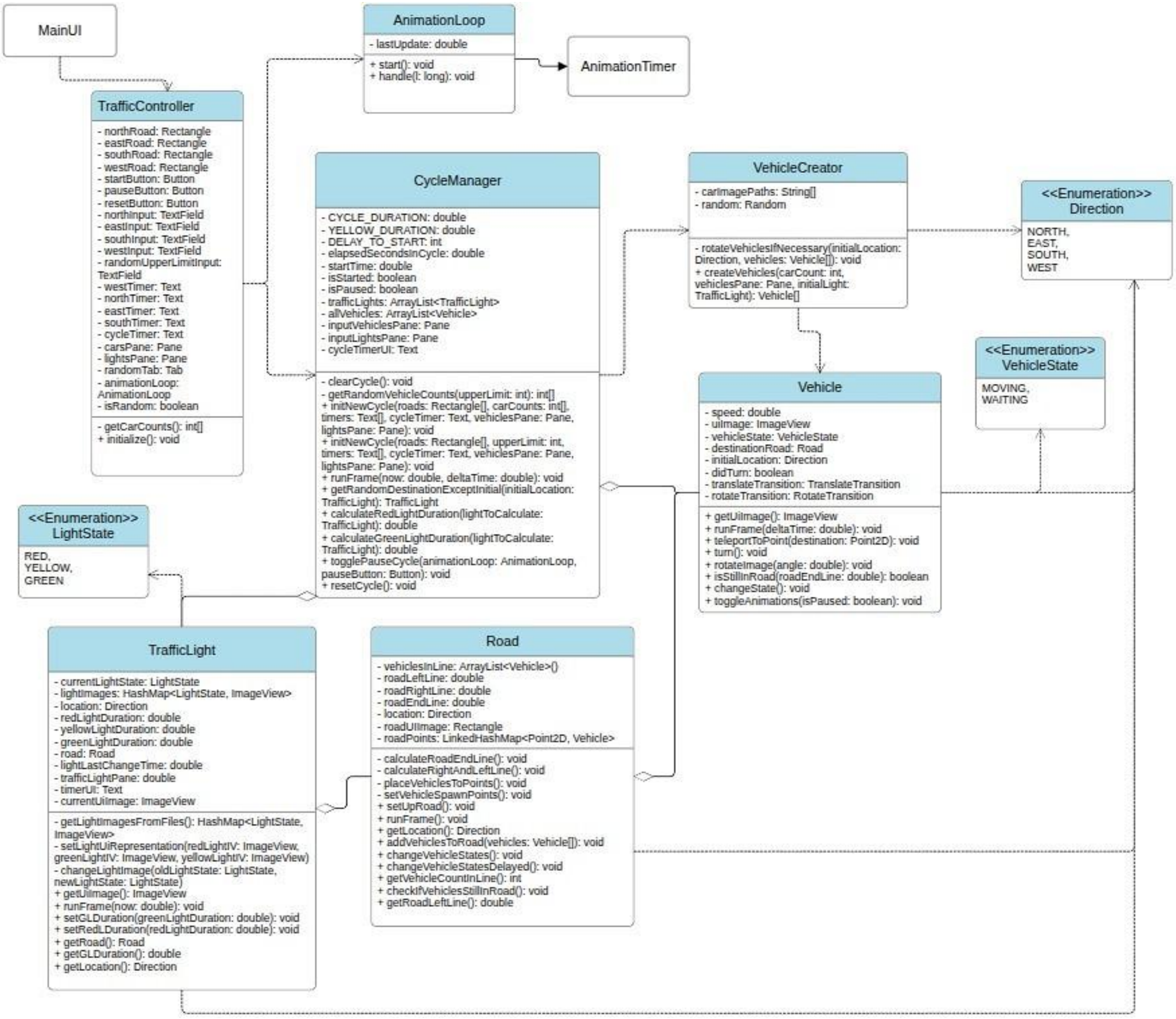
- Model: AnimationLoop Class: Initiates and maintains the animation loop. **CycleManager Class**: Initializes all traffic lights and calculates light durations based on traffic density. It also manages pausing and resetting the simulation. **Road Class**: Defines road dimensions and directions. Controls the proper positioning and movement of vehicles within lanes. **TrafficLight Class**: Manages the light durations, appropriate visuals, and color changes of traffic lights. **Vehicle Class**: Defines vehicle speed, direction, and movement, as well as stop-and-go behavior and turning to any direction. **VehicleCreator Class**: Handles vehicle generation randomly.

- Controller: **TrafficController Class**: Responsible for controlling user interface interactions. Accesses UI components through the FXML file. The interface was visually designed using Scene Builder and linked to the controller layer via the FXML file. Interface elements were accessed using the @FXML annotation. Button activations were implemented and managed here.

- View

To enhance the realism of the simulation, the user interface was carefully designed using the Scene Builder tool. FXML files were used to integrate this design with Java code. Visual modeling of roads, traffic lights, and surrounding elements was implemented. Timers displaying the remaining time of each traffic light were also included. The use of Scene Builder contributed to the code's readability.

- UML Class Diagram


MainUI
AnimationLoop
- lastUpdate: double
+ start(): void
+ handle(l: long): void
AnimationTimer
TrafficController
- northRoad: Rectangle
- eastRoad: Rectangle
- southRoad: Rectangle
- westRoad: Rectangle
- startButton: Button
- pauseButton: Button
- resetButton: Button
- northInput: TextField
- eastInput: TextField
- southInput: TextField
- westInput: TextField
- randomUpperLimitInput: TextField
- westTimer: Text
- northTimer: Text
- eastTimer: Text
- southTimer: Text
- cycleTimer: Text
- carsPane: Pane
- lightsPane: Pane
- randomTab: Tab
- animationLoop: AnimationLoop
- isRandom: boolean
- getCarCounts(): int[]
+ initialize(): void
CycleManager
- CYCLE_DURATION: double
- YELLOW_DURATION: double
- DELAY_TO_START: int
- elapsedSecondsInCycle: double
- startTime: double
- isStarted: boolean
- isPaused: boolean
- trafficLights: ArrayList<TrafficLight>
- allVehicles: ArrayList<Vehicle>
- inputVehiclesPane: Pane
- inputLightsPane: Pane
- cycleTimerUI: Text
- clearCycle(): void
- getRandomVehicleCounts(upperLimit: int): int[]
+ initNewCycle(roads: Rectangle[], carCounts: int[], timers: Text[], cycleTimer: Text, vehiclesPane: Pane, lightsPane: Pane): void
+ initNewCycle(roads: Rectangle[], upperLimit: int, timers: Text[], cycleTimer: Text, vehiclesPane: Pane, lightsPane: Pane): void
+ runFrame(now: double, deltaTime: double): void
+ getRandomDestinationExceptInitial(initialLocation: TrafficLight): TrafficLight
+ calculateRedLightDuration(lightToCalculate: TrafficLight): double
+ calculateGreenLightDuration(lightToCalculate: TrafficLight): double
+ togglePauseCycle(animationLoop: AnimationLoop, pauseButton: Button): void
+ resetCycle(): void
VehicleCreator
- carImagePaths: String[]
- random: Random
- rotateVehiclesIfNecessary(initialLocation: Direction, vehicles: Vehicle[]): void
+ createVehicles(carCount: int, vehiclesPane: Pane, initialLight: TrafficLight): Vehicle[]
<<Enumeration>> Direction
NORTH,
EAST,
SOUTH,
WEST
<<Enumeration>> VehicleState
MOVING,
WAITING
Vehicle
- speed: double
- uiImage: ImageView
- vehicleState: VehicleState
- destinationRoad: Road
- initialLocation: Direction
- didTurn: boolean
- translateTransition: TranslateTransition
- rotateTransition: RotateTransition
+ getUiImage(): ImageView
+ runFrame(deltaTime: double): void
+ teleportToPoint(destination: Point2D): void
+ turn(): void
+ rotateImage(angle: double): void
+ isStillInRoad(roadEndLine: double): boolean
+ changeState(): void
+ toggleAnimations(isPaused: boolean): void
<<Enumeration>> LightState
RED,
YELLOW,
GREEN
TrafficLight
- currentLightState: LightState
- lightImages: HashMap<LightState, ImageView>
- location: Direction
- redLightDuration: double
- yellowLightDuration: double
- greenLightDuration: double
- road: Road
- lightLastChangeTime: double
- trafficLightPane: double
- timerUI: Text
- currentUiImage: ImageView
- getLightImagesFromFiles(): HashMap<LightState, ImageView>
- setLightUiRepresentation(redLightIV: ImageView, greenLightIV: ImageView, yellowLightIV: ImageView)
- changeLightImage(oldLightState: LightState, newLightState: LightState)
+ getUiImage(): ImageView
+ runFrame(now: double): void
+ setGLDuration(greenLightDuration: double): void
+ setRedLDuration(redLightDuration: double): void
+ getRoad(): Road
+ getGLDuration(): double
+ getLocation(): Direction
Road
- vehiclesInLine: ArrayList<Vehicle>()
- roadLeftLine: double
- roadRightLine: double
- roadEndLine: double
- location: Direction
- roadUIImage: Rectangle
- roadPoints: LinkedHashMap<Point2D, Vehicle>
- calculateRoadEndLine(): void
- calculateRightAndLeftLine(): void
- placeVehiclesToPoints(): void
- setVehicleSpawnPoints(): void
+ setUpRoad(): void
+ runFrame(): void
+ getLocation(): Direction
+ addVehiclesToRoad(vehicles: Vehicle[]): void
+ changeVehicleStates(): void
+ changeVehicleStatesDelayed(): void
+ getVehicleCountInLine(): int
+ checkIfVehiclesStillInRoad(): void
+ getRoadLeftLine(): double

Conclusion

The developed system created a smooth traffic simulation by managing traffic lights based on real-time traffic density. Our project is open to enhancements and provides a flexible structure that meets the expected features to a high degree.