|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| simge, sembol, logo, yazı tipi, amblem içeren bir resim  Açıklama otomatik olarak oluşturuldu  CSE1142 Term Project  Project Name: Traffic Light Simulator  CSE1142: Computer Programming II, Spring 2024  Instructor: Sanem Arslan Yilmaz  Date Submitted: May 10, 2024   |  |  |  |  | | --- | --- | --- | --- | |  | Names | Surnames | ID Numbers | | 1 | Hüseyin | Aksoy | 150122010 | | 2 | Eren Emre | Aycibin | 150122054 | | 3 | Kerim | Ak | 150123515 |     Description of the game:  We wanted to make a game called traffic light simulator. Our game aims to control a real-life car traffic simulation. Cars spawn randomly from random structures and move on random roads. The user controls traffic lights to get the vehicles to their destination without crashing. In case of any accident, the accident score increases by one. If a car completes its route, winning the game increases its score by one.  While playing this game, the user must reach a score determined by the game producers. Once the user reaches the score, they are greeted with a winning screen and can challenge themselves further by moving on to the next level or enjoy the fun by replaying the same level. When the user reaches the accident limit, he/she encounters the losing screen. And we recommend him to play again until he wins that level so that he can enjoy the game.  Implementation Details:   |  |  | | --- | --- | |  | Building | | ----++ | type: int  rotation: double  gridX: int  gridY : int  gridSize: int  arr: Color | | + ++ | Building (type: int, rotation: double, gridX: int, gridY : int)  GetGridX(): int  getGridY() int |  * Building Constructor: Initializes a Building instance with a specified type, rotation, color, and grid coordinates. It uses a color array to select building colors based on the input parameter. The constructor creates building shapes due to the given type and rotation. * Type 0: Rectangular Building: This type involves creating a larger rectangle with an arc for rounded corners. It also adds inner squares to the outer rectangle.The rotation also affects the positioning and translation of these elements. * Type 1: Circular Building: This type has a circles inside the rectangle. The rotation also affects the positioning and translation of these elements. * Type 2: Basic Square Building: This type represents a simple square building with rounded corners. * getGridX(): Returns the gridX coordinate. (shows the building's horizontal position within a grid layout.) * getGridY(): Return the gridY coordinate (shows the building's vertical position in the grid system.)  |  |  | | --- | --- | |  | RoadTile | | ---- | type: int  rotation: double  gridX: int  gridY: int | | +++ | RoadTile (type: int, rotation: double, gridX: int, gridY: int)  getGridX(): int  getGridY(): int |  * RoadTile Constructor: Initializes a RoadTile instance based on a specified type, rotation, and grid coordinates. It creates the road tiles according to the type of road and its rotation. * Type 0: Straight Road: Creates a straight road tile. The rectangle representing the road. * Type 1: Curved Road: Constructs a curved road using two quarter circles. The rotation of these arcs is set to align them. The arcs are added to the StackPane at specific positions due to rotation, creating a curved effect. * Type 2: Intersection: Represents an intersection by combining two rectangles to form a cross. This intersection creates a junction in the road network. * Type 3: Main Road with Minor Road: A combination of a larger rectangle for the main road and a smaller rectangle for a minor road. It aligns and rotates the smaller road relative to the main road dut to the rotation. * getGridX(): Returns the gridX coordinate of the road tile, shows its horizontal position in the gridPane. * getGridY(): Retrieves the gridY coordinate of the road tile, shows its vertical position within the gridPane.  |  |  | | --- | --- | |  | TrafficLight | | --- | lightLine: Line  light : Circle  isRed: boolean | | +++++ | TrafficLight(double x1, double y1, double x2, double y2)  getLine(): Line  getCirlce(): Circle  isRed(): boolean  isGreen(): boolean |  * TrafficLight Constructor: Initializes a TrafficLight instance with a line and a circle representing the traffic light. The circle's position and radius are calculated based on the length of the line and its coordinates. The circle's initial color is set to green, indicating the light is green at the start. * getLine(): Returns the Line representing the traffic light's structure, allowing further customization or retrieval of its properties. * getCircle(): Retrieves the Circle representing the light, providing direct access for manipulation or observation of its attributes. * isRed(): Checks if the traffic light is currently red, indicating whether cars should stop or proceed with caution. * isGreen(): Checks if the traffic light is green, signaling that cars can proceed safely. * Mouse Click Event: Attaches an event handler to the circle that toggles the light's color when clicked. If the current color is green, it changes to red and sets isRed to true. Otherwise, it changes back to green and sets isRed to false. This mechanism allows interactive control of the traffic light's state.  |  |  | | --- | --- | |  | Car | | +---- | key: double  pathTransition: PathTransition  isStopped: boolean  shouldStop: boolean  isOnLight: boolean | | ++++++++++++ | Car(pathTransition: PathTransition)  move(): void  stop(): void  get\_stopped(): boolean  set\_stopped(): void  set\_shouldStop(): void  get\_shouldStop(): boolean  get\_transtion(): PathTransition  set\_key( var: double) : void  get\_key(): double  set\_isOnLight(): void  get\_isOnLight():boolean |  * Car Constructor: Initializes a new Car instance with specified path transition, and initial color. It also works with the path transition for the car's movement and sets the path to follow. * move(): Begins the PathTransition to start the car's movement along the given path. * stop(): Pauses the PathTransition to stop the car's movement. * get\_stopped(): Returns whether the car is stopped, gives us information about its current state. * set\_stopped(boolean var): Sets the stopped status of the car, allowing control car should stop or not. * set\_shouldStop(boolean var): Updates whether the car should stop. * get\_shouldStop(): Checks if the car has been wanted to stop, provides information about control movement. * get\_transition(): Returns the current PathTransition about the car, allowing for additional control or updating to its movement. * set\_key(double var): Sets a special key for the car, which could be used for changing cars. * get\_key(): Returns the current key of the car. * set\_isOnLight(boolean var): Sets whether the car is on a traffic light for simulating traffic conditions. * get\_isOnLight(): Checks if the car is currently on a traffic light. * set\_path(Path var): Sets a new path for the car to follow, allowing for updating to the car's path. * get\_path(): Returns the current path the car is following  |  |  | | --- | --- | |  | App | | ++++++ -  -++ | paths: ArrayList<Path>  trafficLights : ArrayList<TrafficLight>  metaData: MetaData  cars: ArrayList<Car>  crashed: int  finished: int  scoreLabel: Label  crashLabel: Label  accidentLimit: int  winCondition: int | | ++++++++-+++-++-++ | start(Stage primaryStage) : void  main (args : String[]) : void  getPaths(): ArrayList<Path>  checkTrafficLight(car: Car) : void  collidedCars: ArrayList<Car>  time: double  createTraffic(): void  spawnCar(): void  getRandomPath(): Path  calculatePathLength(path : Path) : double  createMap(file :File) : Scene  checkCollisons(collidedCars: ArrayList<Car>) : void  removeCollidedCars(collidedCars: ArrayList<Car>) : void  isGonnaStep(collidedCars: ArrayList<Car>) : void  inList( arr: ArrayList<Car> , car: Car): boolean  calculateDistance(car1 : Car, car2: Car) : double  calculateDistanceX(car1 : Car, car2: Car) : double  calculateDistanceY(car1 : Car, car2: Car) : double |  * getPaths(): Returns a list of paths used for car transitions. * checkTrafficLight(Car car): Checks if a given car intersects with a traffic light and adjusts its movement accordingly. * createTraffic(): Initializes a recurring event to create traffic at certain intervals. * update(): Updates the state of traffic, spawning new cars at intervals and checking for necessary actions. * spawnCar(): Spawns a new car and sets its path transition. * getRandomPath(): Returns a random path from the list of defined paths. * calculatePathLength(Path path): Calculates the total length of a given path. * createMap(File file): Reads metadata and elements from a file to create a map for the simulation. * checkCollisions(ArrayList<Car> collidedCars): Checks for collisions between cars . * removeCollidedCars(ArrayList<Car> collidedCars): Removes cars that have collided from the scene and from the list of the active cars. * isGonnaStop(ArrayList<Car> collidedCars): Checks for cars should stop due to other stopped cars or traffic lights. * inList(ArrayList<Car> arr, Car car): Checks if a given car is in the Car list. * calculateDistance(Car car1, Car car2): Calculates the distance between two cars. * calculateDistanceX(Car car1, Car car2): Calculates the X-coordinate distance between two cars. * calculateDistanceY(Car car1, Car car2): Calculates the Y-coordinate distance between two cars. * findMoveTo(Path path): Finds the MoveTo element in a specified path.  |  |  | | --- | --- | |  | MetaData | | ++++++++ | gridPane: GridPane  pane: Pane  path: Path  winCondition: int  accident: int  x: int  y: int  scene: Scene | | +  +++++++ | MetaData(double width, double height, int rows, int columns, int path, int winCondition, int accident)  get\_scene(): Scene  get\_pane(): GridPane  get\_overlayPane(): Pane getPath(): Path  setPath(): void  getWinCondition(): int  getAccidentLimit(): int |  * MetaData Constructor: Takes width, height, rows, columns, path, winCondition and accident parameters and consturct them. * get\_scene(): Returns the Scene containing the grid and other elements. * get\_pane(): Give access to the GridPane for customization and other elements in it. * get\_overlayPane(): Returns the Pane hat holds the GridPane and other elements like Path. * getPath(): Returns thePath object, which can be used for drawing lines or shapes. * setPath(Path path): Allows setting a new Path object, useful for updating the drawing elements. * getWinCondition(): Returns the win condition value, specifying the game's success condition. * getAccidentLimit(): Returns the accident limit, which might be used to set boundaries or rules in the simulation.   Test Cases:  WhatsApp Görsel 2024-05-10 saat 22.17.44_9d214651Our stater screen you can see levels and play buttons. you can select the level and then play the game.    Level1:  In the level1 when you click the traffic light the light turns into green to red. Then cars stop. when a collision happens cars that collided will remove from the game screen. When a car finishes it’s path you will gain score points with this way you can win the game or when you make 50 crashes you will lose the game.    Level 2:  WhatsApp Görsel 2024-05-10 saat 22.20.02_a65095be  Level 3:  WhatsApp Görsel 2024-05-10 saat 22.22.07_decf67e2    Level 4:  WhatsApp Görsel 2024-05-10 saat 22.22.45_32c9842b  Level 5:  ekran görüntüsü, diyagram, metin içeren bir resim  Açıklama otomatik olarak oluşturuldu |