Sensor

- length: double
- projectedNormalizedLength: double

+ Sensor(int, Car)
+ setLength(double): void
+ getLength(): double
+ getProjectedLength(): double
+ setProjectedLength(double): void

Car (partial)

- learning
- CAR_R
- CAR_C
- MIN_SP
- angularV
- velocity:
- fitnessS
- sensorA
- brain: N

+ Car(Po
+ update(
+ multiply
+ getSens
+ getBrai
+ setBrair
+ getFitne
+ setFitne
+ setAng
+ setVelo
+ setLear

---

style.css

Timer

+ handle(long): void

Simulation

- singleSimulation: Simulation
- angVelocity: TextField [0..1]
- blueSlider: Label [0..1]
- borders: Shape [0..1]
- bordersList: ArrayList<Shape> [0..*]
- carList: ArrayList<Car [0..+]
- carListFull: ArrayList<Car [0..*]
- carPane: Pane [0..1]
- carSpeed: TextField [0..1]

SelfDrivingCar   (MAIN)

+ start(PrimaryStage): void
+ main(String[]): void

- WIDTH: int
- HEIGHT: int
- PANE_PADDIN
- displayedCar: C
- neuralNetwork:

```
javafx.scene.shape.Circle
```

△ Extends

Car

```
Rate: float
ADIUS: int
OLOR: Color
PEED: double
Velocity: double
: double
core: int
Array: Sensor[]
euralNetwork
```

```
int2D)
(double, double): void
Vectors(double[], double[]): double
sorArray(): Sensor[]
n(): NeuralNetwork
n(NeuralNetwork): void
essScore(): int
essScore(int): void
ularVelocity(double): void
city(double): void
ningRate(float): void
```

ActivationFunctions

```
+ sigmoid(double x): double
+ relu(double x): double
+ tanh(double x): double
```

◇

HiddenLayer

```
- weights: double[][]
- currentLayerSize: int
- previousLayerSize: int
- random: Random
```

```
+ hiddenLayer(int, int)
+ clone(): HiddenLayer
+ mutate(float): void
+ initRandom(): void
+ activate(double[]): double[]
+ toString(): String
+ getWeights(): double[][]
```

<<Interface>> Drivable

```
- SENSOR_COUNT: int
```

```
+ setAngularVelocity(double): void
+ setVelocity(double): void
+ setLearningRate(float): void
```

NeuralDisplay

```
NG: int
Car
: NeuralNetwork
```

carSpeed: TextField [0..1]

colorSalector: Rectangle [0..1]

deadCars: ArrayList<Car> [0..*]

exit: Button [0..1]

fitnessList: ArrayList<Integer> [0..*]

gen: TextField [0..1]

greenSlider: Label [0..1]

intersections: ArrayList<Shape> [0..*]

lastCar: Car [0..1]

layers: int[] [0..*]

menuBar: MenuBar [0..1]

mutRate: TextField [0..1]

neuralDisplay: NeuralDisplay [0..1]

neuronsPerLayerLabel: Label [0..1]

neuronsPerLayerTextField: TextField [0..1]

noCars: TextField [0..1]

pause: boolean [1]

play: Button [0..1]

player: MediaPlayer [0..1]

recommended: Button [0..1]

redSlider: Label [0..1]

reset: Button [0..1]

root: BorderPane [0..1]

save: Button [0..1]

sliderBlue: Slider [0..1]

sliderGreen: Slider [0..1]

sliderRed: Slider [0..1]

start: Button [0..1]

started: BooleanProperty [0..1]

timer: AnimationTimer [0..1]

title: Label [0..1]

userInterface: GridPane [0..1]

volumeSlider: Slider [0..1]

volumeSliderImage: ImageView [0..1]

+ addFitness()

applySettings()

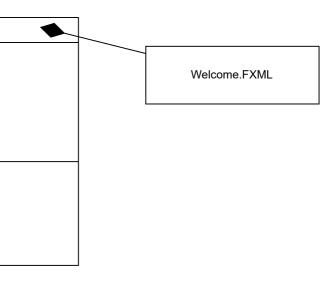- calculateDistanceToPoint(sensor: Line, x: double, y: double): double

---

NeuralDisplay (partial, right)

- neuralNetwork:
- activations: dou
- neuralNetwork:
- layers: HiddenL
- nbLayers: int[]
- lineWeights: Ar
- neuronList: Arr
+ timer: Animatic

+ NeuralDisplay

- generateNeuro

- generateWeigh

+ getDisplayedC

- update(): void

WelcomeController

- welcome: Label
- start: Button
- exit: Button
- volumeSlider: Slider
- player: MediaPlayer
- simulation: Simulation

+ initialize(URL, ResourceBundle): void

+ getPlayer(): MediaPlayer

+ changeVolume(Slider, Media): void

+ highlightButton(Button): void

NeuralNetwork (partial, left, cut off)

NeuralNetwork

uble[][]

: NeuralNetwork

Layer[]

rayList<Line>

ayList<ArrayList<Circles>>

onTimer

v(Car)

ons(): void

nt(): void

Car(): Car

NeuralNetwork

- hiddenLayers: HiddenLayer[]
- activation: double[][]
- learningRate: float
- layers: int[]

+ NeuralNetwork(float, int[])

+ clone(): NeuralNetwork

+ predict(double[]): double[]

+ mutate(): void

+ getActivations(): double[][]

+ getHiddenLayers(): HiddenLayer[]

+ getLayers(): int[]

+ toString(): String

Welcome.FXML

-calculateSensors Length()

- checkCarCollision

+ checkCollision With Wall(order: int, car: Car): double

-checkIfAllCarsDead()

- checkMutate Value

-checkTextInputs(e: KeyEvent, select: TextField, noCars: TextField, mutRate: TextField, carSpeed: TextField, ang Velocity: TextField, save: Button, start: Button)

-checkTextInputsComma(e: KeyEvent, select: TextField, no Cars: TextField, mutRate: TextField, carSpeed: TextField, ang Velocity: TextField, save: Button, start: Button)

-checkTextInputsDecimal(e: KeyEvent, select: TextField, no Cars: TextField, mutRate: TextField, carSpeed: TextField, ang Velocity: TextField, save: Button, start: Button)

#draw()

+highlightButton(button: Button)

-moveCar()

- removeDeadCars()

- showAboutWindow()

-showAlert(title: String, content: String)

-showEndWindow()

-showNeuralDisplay(car: Car)