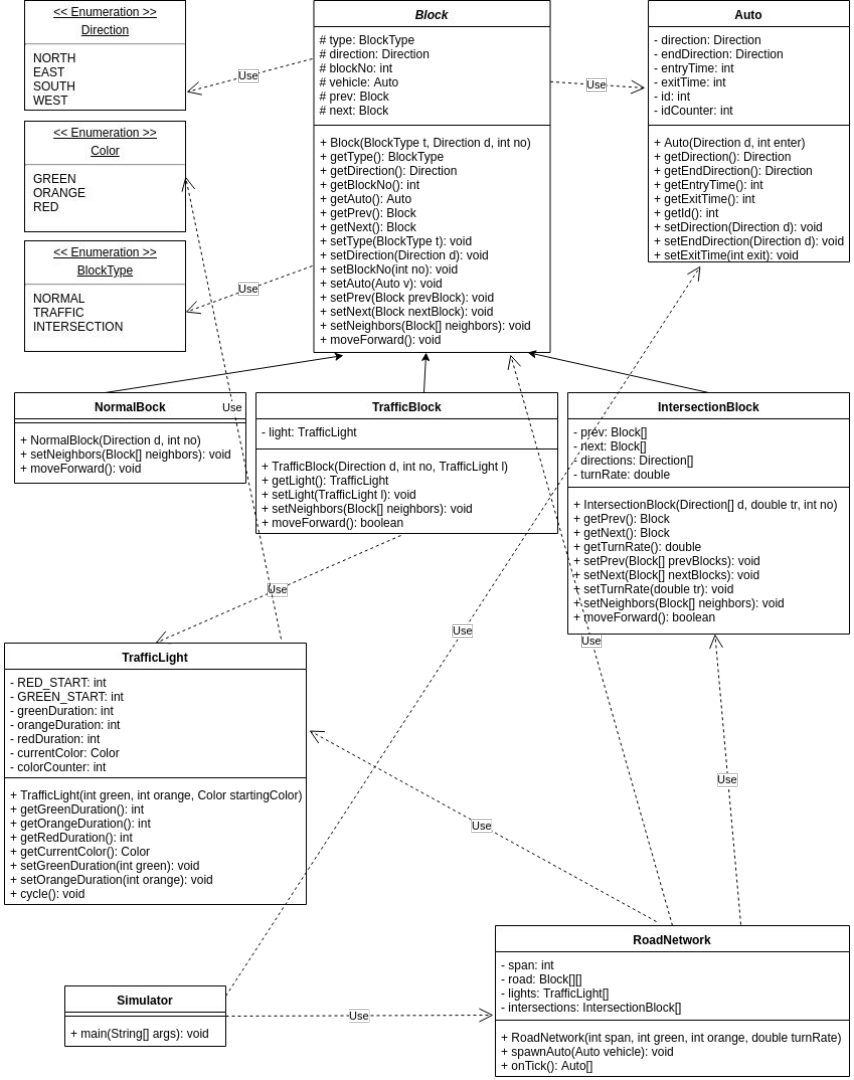
# Traffic Simulator

Hunter Negron

# UML


<< Enumeration >>
Direction
NORTH
EAST
SOUTH
WEST
<< Enumeration >>
Color
GREEN
ORANGE
RED
<< Enumeration >>
BlockType
NORMAL
TRAFFIC
INTERSECTION
Block
# type: BlockType
# direction: Direction
# blockNo: int
# vehicle: Auto
# prev: Block
# next: Block
+ Block(BlockType t, Direction d, int no)
+ getType(): BlockType
+ getDirection(): Direction
+ getBlockNo(): int
+ getAuto(): Auto
+ getPrev(): Block
+ getNext(): Block
+ setType(BlockType t): void
+ setDirection(Direction d): void
+ setBlockNo(int no): void
+ setAuto(Auto v): void
+ setPrev(Block prevBlock): void
+ setNext(Block nextBlock): void
+ setNeighbors(Block[] neighbors): void
+ moveForward(): void
Auto
- direction: Direction
- endDirection: Direction
- entryTime: int
- exitTime: int
- id: int
- idCounter: int
+ Auto(Direction d, int enter)
+ getDirection(): Direction
+ getEndDirection(): Direction
+ getEntryTime(): int
+ getExitTime(): int
+ getId(): int
+ setDirection(Direction d): void
+ setEndDirection(Direction d): void
+ setExitTime(int exit): void
NormalBock
+ NormalBlock(Direction d, int no)
+ setNeighbors(Block[] neighbors): void
+ moveForward(): void
TrafficBlock
- light: TrafficLight
+ TrafficBlock(Direction d, int no, TrafficLight l)
+ getLight(): TrafficLight
+ setLight(TrafficLight l): void
+ setNeighbors(Block[] neighbors): void
+ moveForward(): boolean
IntersectionBlock
- prev: Block[]
- next: Block[]
- directions: Direction[]
- turnRate: double
+ IntersectionBlock(Direction[] d, double tr, int no)
+ getPrev(): Block
+ getNext(): Block
+ getTurnRate(): double
+ setPrev(Block[] prevBlocks): void
+ setNext(Block[] nextBlocks): void
+ setTurnRate(double tr): void
+ setNeighbors(Block[] neighbors): void
+ moveForward(): boolean
TrafficLight
- RED_START: int
- GREEN_START: int
- greenDuration: int
- orangeDuration: int
- redDuration: int
- currentColor: Color
- colorCounter: int
+ TrafficLight(int green, int orange, Color startingColor)
+ getGreenDuration(): int
+ getOrangeDuration(): int
+ getRedDuration(): int
+ getCurrentColor(): Color
+ setGreenDuration(int green): void
+ setOrangeDuration(int orange): void
+ cycle(): void
RoadNetwork
- span: int
- road: Block[][]
- lights: TrafficLight[]
- intersections: IntersectionBlock[]
+ RoadNetwork(int span, int green, int orange, double turnRate)
+ spawnAuto(Auto vehicle): void
+ onTick(): Auto[]
Simulator
+ main(String[] args): void
Use

# Design Breakdown: Blocks

- Abstract Block class
  - Inherited by NormalBlock, TrafficBlock, and IntersectionBlock
- Utilizes the BlockType and Direction enums
- Attributes
  - blockNo
  - Vehicle
  - Prev & next
- Abstract methods
  - setNeighbors(Block[] neighbors)
  - moveFoward()

# Design Breakdown: Traffic Management

- TrafficBlock class
  - Like a normal block, but has a TrafficLight
- TrafficLight class
  - Utilizes the Color enum
  - Initialized with green and orange durations
  - Computes red duration
  - cycle() method systematically changes the light color on each tick

# Design Breakdown: Intersections

- IntersectionBlock class
  - Initialized with turn rate
  - Has multiple directions, prev Blocks, and next Blocks
- The moveForward() method
  - Generates a random number 0 - 1
  - If less than the turn rate, changes the vehicle’s direction and advances it
  - If not, then advances it in the current direction

# Design Breakdown: Vehicles

- Auto class
  - Start Direction
  - End Direction
  - Entry time
  - Exit Time
- References are contained in an ArrayList in the Simulator

# Design Breakdown: RoadNetwork & Simulator

- RoadNetwork Class
  - Utilizes all other classes to construct an intersection
  - onTick() method updates TrafficLights, moves cars forward, and return exited cars
  - spawnCar() is used by Simulator to add new cars
- Simulator
  - Collects information from user
  - Runs through each tick, randomly adding new cars
  - Calculates data and outputs it where necessary

# Test Case Format

- The Simulator accepts 6 inputs
  - Entry Rate: real number between 0 and 1
  - Turn Rate: real number between 0 and 1
  - Green: nonnegative integer
  - Orange: nonnegative integer
  - Simulation Time: nonnegative integer
  - Lane Span: positive integer

# Test Case #1

- Test poorly formatted numbers
- Input “n”
- Output “Not a valid number. Exiting.”

# Test Case #2

- Test out-of-range numbers
- Input “-1”, “-1”, “-1”, “-1”, “-1”, “0”
- Output “All values must be nonnegative. Span must be positive. Exiting.”
- Span must be a positive number in order to have traffic lights to control.

# Test Case #3

- Test the minimum allowed numbers
- Input “0”, “0”, “0”, “0”, “0”, “1”
- Output

Average Wait Time: 0.0

NORTH-bound flow rate: 0

EAST-bound flow rate: 0

SOUTH-bound flow rate: 0

WEST-bound flow rate: 0

Total Flow Rate: 0

# Test Case #4

- Test generic numbers
- Input “.4”, “.6”, “8”, “2”, “500”, “6”
- Output

Average Wait Time: 17.64786324786325

NORTH-bound flow rate: 115

EAST-bound flow rate: 185

SOUTH-bound flow rate: 121

WEST-bound flow rate: 164

Total Flow Rate: 585