COSC 3P91 – Assignment 2 – 7376726

PARKER TENBROECK, Brock University, Canada

Traffic simulation design and implementation

1 DESIGN OVERVIEWS

1.1 RoadMap

The `RoadMap` contains all information about the `Intersections`, `Roads`, and `Vehicles` that are apart of the map. The map is modeled as a directed graph where nodes are `Intersections` and edges are `Roads`.

1.1.1 Road. Roads act as edges in the graph and consist of one or more Lanes, each `Lane` holds the `Vehicles` that are currently on it. `Roads` exist only as a child to two intersections, one being the incoming and the other being the outgoing. this also implicitly determines the 'direction' of the road. Vehicles can only turn onto a road from the incoming end, and turn off a road when at the outgoing end. `Vehicles` can only travel in a single direction along the road. `Vehicles` must always start at the back most position on the incoming end of the road. The `Road` is responsible for updating all `Vehicles` currently on it.

1.1.2 Intersection. `Intersections` act as the nodes in the graph and can have none to many outgoing and incoming `Roads`. `Intersections` have a position x,y that specify where is exists in the `RoadMap`. Every `Intersection` is responsible of keeping track of what `Turns` are possible. Each `Turn` consists of a `Lane` from an incoming road, a name (used to indicate Left, Right, Forward, etc.) and a `Lane` from an outgoing road to turn onto. A turn also has a flag indicating if the turn is enabled. An arbitrary number of Turns are possible in any direction from any given `Intersection`.

`Intersection` can be extended to provide additional functionality. Three subclasses of `Intersection` are provided as an example. `SourceIntersection`, `DrainIntersection`, and `TimedIntersection`. `SourceIntersection` and `DrainIntersection` provide ways to add/remove `Vehicles` to the road system respectively. and `TimedIntersection` will act like a timed traffic light, preventing `Vehicles` from making turns until the 'light' turns green for them.

1.2 GambleHandler

The `GambleHandler` decides how to handle vehicles who have made a decision that may or may not be considered a gamble. Different `GambleHandlers` allow for different behavior if wanted.

1.3 Vehicle

A `Vehicle` is some entity that can exist on a `Lane` on a `Road` in a `RoadMap`. A `Vehicle` can contain a `Controller` that influences how the Vehicle makes navigation decisions. `Vehicles` are abstract because they can represent many different forms of transportation however, navigation decisions should be left up to the Controller and not overridden. Other

Author's address: Parker TenBroeck, Brock University, 1812 Sir Isaac Brock Way, St. Catharines, ON, L2S 3A1, Canada.

aspects of `Vehicle` should be modified like how it is drawn, and other specialized behavior related to the specific kind of `Vehicle` being implemented.

Two implementations of `Vehicle` are examples of how a `Vehicle` can be made, `Car` is a standard vehicle and `Truck` is a larger and slower vehicle.

1.4 Simulation

`Simulation` ties all these components together allowing the simulation to actually be ran. it allows access to all the different parts of the simulation alongside other information / configuration allowing things like pausing the simulation to allow players to choose a turn they want to take.

1.5 IO

1.5.1 TextDisplay. The `TextDisplay` both displays to the terminal, gets user input and allows for player control of vehicles.

The following are for future use as we only need a terminal UI currently.

1.5.2 Display. The `Display` is responsible for drawing and displaying simulation visuals, it is constructed with `Input` so it can attach `Inputs` listeners to its window.

1.5.3 Input. `Input` listens to user input present on `Display` and provides easy access to event data. `Storage` holds all event data per frame, two are used to record new input for the next frame saving old input from the previous frame for consistency.

1.5.4 View. The `View` holds a `Display` and `Input`, it provides a 'view' into the simulation that can be zoomed / moved around. It also provides functionality to provide mouse events in the same coordinate space as the `Simulation`. A `View` can optionally follow a Vehicle.

2 IMPLEMENTATION OVERVIEWS

2.1 RoadMap

The `RoadMap` holds all information about all `Road`s and `Intersectionss`. Each `Intersection` and `Road` have a unique ID `String` to identify each part of the map. This is used to identify each part of the map when loading and writing maps to files and also to get specific parts of the map to add certain behavior. This is stored as a sort of bidirectional map using a `HashMap` between a `String`(ID) and both a `Road` and `Intersection` and the opposite. This allows is to find the ID from a `Road`/`Intersection` and also a `Road`/`Intersection` from a ID.

There is both a list for every `Road` and `Intersection` that exist in the `RoadMap`. This allows for easy iteration over all `Road`s and `Intersection`s in the map for drawing/updating.

Each road is linked to exactly two `Intersection`s, One being the outgoing and the other being the incoming. This is stored as a `HashMap` with `Road` as the key and `Intersection` as the value for both the outgoing and incoming.

Finally the `RoadMap` stores outgoing and incoming `Road`s for each `Intersection`. This is stored as a `HashMap` with `Intersection` as the key and `ArrayList<Road>` as the value.

To ensure that RoadMap is constructed in a consistent way MapBuildingException are thrown depending on what errors a user makes when constructing a map. These Include (Road/Intersection)IDClash that occur when a provided ID for a new Road/Intersection clashes with an existing ID already in the RaodMap. Another exception is InvalidRoadLink that occurs when two Intersections (outgoing and incoming) already have a Road linking them in the specified direction. A InvalidTurn exception is thrown when a turn is attempted to be made between two lanes that aren't strictly part of a incoming road for a Intersection to an outgoing road of the same Intersection. Finally a CustomIntersectionLoadException is thrown when some error occurs when trying to load a custom intersection type. This is also where generic wild cards are used as we can't possible know the generic parameter of the Class<?> inner type when loading from file. This feature uses runtime reflection to load any kind of intersection that another user might define.

The RoadMap can be written to / read from using javas Readable and OutputStreamWriter. Both MapBuildingExceptions and IOExceptions are thrown if either happen when reading/writing maps to files. These are standard java utility classes for streaming I/O

2.2 GambleHandler

The GambleHandler is an interface to allow for the implementation of different behavior for gambling events, a simple DefaultGambleHandler is provided to show basic capabilities of how gambling could work. It increases and decreases a Vehicles reputation from good and bad decisions respectively. also decreasing health if very bad actions are taken like blowing a stop or changing lanes into another vehicle / off the road.

2.2.1 Road. Each Road has a name, length, speed limit, direction vector, last updated tick, and most importantly an array of Lanes. Each Road must contain at least one Lane, the Lane in the 0th index is the left most lane, and the lane in the last index is the rightmost lane. The name of the Road doesn't need to be unique but is displayed to the user. The speed limit determines how fast Vehicles *should* go when their speed multiplier is 1. Both the direction vector and length are set/updated when the Road is initially linked/made and updated if either its incoming/outgoing intersections position is changed. The tick is stored so when vehicles turn off/into the road it can ensure that the Vehicle only gets ticked once per actual tick.

On every tick each road first checks for Vehicles at the end of each lane that can make a turn, if so placing them on the road they have decided to turn onto. Then updating each Vehicle in order from furthest along to furthest back. also checking if each Vehicle would like to make a lane change. If a Vehicle makes a lane change / turn that is disallowed or dangerous then

2.2.2 Intersection. Each Intersection contains a name, position and a map of all valid turns from every incoming lane. The position is stored as two doubles (x,y) and the name is a simple String. The turn map is stored as a HashMap with the key being a Lane (A lane from an incoming road to this Intersection) with the value being ArrayList<Turn>. A Turn object is a storage class that contains a name(String) that is a human readable name for the turn, and a Lane (where the lane is a lane from an outgoing road from this

Intersection).

Each Turn can be enabled / disabled to signal things like stop lights and other restrictions for intersections.

Intersections can be extended to add specific behavior like SourceIntersection, DrainIntersection and, TimedIntersection. These all perform a specific task like Adding Vehicles, Removing Vehicles and controlling when/how certain turns are enabled / disabled all respectively. A SourceIntersection can take a variadic list of functional interfaces for constructing vehicles. A example list is provided showing the use of a method reference to the constructor and also a lamda to construct a custom vehicle.

2.3 Vehicle

Each Vehicle stores health, reputation, distanceAlongRoad, speedMultiplier and, size. these are all stored as a float type. health is starts as 1 and decreases till it reaches 0, at that point the vehicle is considered destroyed. reputation can be any positive number and will increase/decrease from the choices made. distanceAlongRoad is a scalar value between 0 and the length of the road the vehicle is currently on. and is updated every tick. size is how long the vehicle is. A Vehicle also stores a Controller which is responsible for making the decisions of how that Vehicle should behave

Vehicles can be extended to add additional functionality but it should be noted that behavior of how the vehicle is controlled should be changed with Controller and not through extending. Examples of Vehicles that can exist are Car and Truck. Cars are much smaller and tend to drive faster while trucks appear different and drive slower.

As of now nothing happens when a Vehicles health reaches zero. Other gamble handlers or additional systems could detect when a vehicle's health reaches zero and perform some special event.

2.3.1 Controller. The Controller interface gives us a way to determine how *any* vehicle behaves. This allows us to control any Vehicle with any Controller allowing us to have a player be a Car or Truck without needing to extend both with player. We can also make a Controller that is made to be used with a specific Vehicle to give more specialized behavior.

2.4 Simulation

The Simulation holds all things necessary for the simulation to occur. most importantly it holes a single RoadMap that contains the map currently being simulated. Information like paused, debug, simTick, simNanos that store information about the state of the simulation. The final part (and the part that actually runs) is a (sorted) ArrayList<SimSystem>. A SimSystem is an object that represents a singular part of the simulation that does a specific task. each SimSystem has a priority associated with it, priorites with a lower value will get ran before priorities with higher values. Each SimSystem has an init and run method. the init function is ran once when Simulation.run is called before anything else. and the run method is ran every simulation loop. Combining multiple SimSystems together in different ways allows us to split behavior and modify how our simulation behaves much more easily. SimSystem is a local static class that is commonly instantiated as an

anonymous class but in the case a simple system is needed with no `init` one can choose to use `SimSystem.simple` to construct a system with a single lamda function for the `run` method. This can be seen in different areas of the code.

2.5 IO

2.5.1 TextDisplay. The `TextDisplay` not only accepts user input and displays information but it also implements `Controller` and can be attached to a `Vehicle` to control it through terminal based input. When a decision is needed to be made the `TextDisplay` will print the needed information and wait for user input to make the decision it needs.

2.5.2 Input.

2.5.3 Display.

2.5.4 View.

. These components work together to eventually provide a GUI with keyboard input for our traffic simulator.

3 LOOKING

The player will be able to see the surrounding cars by looking at the `TextDisplay` output. I text based graphic alongside text shows players what is around them and what options they have.

4 MOVING

The `TextDisplay` will prompt the user when movement input is needed such as turning or lane changing. A list of possible options will be printed to ensure the player knows what to do.

5 ASSERTING

When turning a list of all possible turns are provided, There are some turns that are 'disabled' because of intersection traffic control or the turn is full because of traffic but invalid turns in the sense of ones that don't exist on the graph aren't possible to return simply because they are never provided in the first place. If a turn is chosen that is disabled gambling event occurs. A similar event happens for an invalid lane change.

6 GAMBLING

Different Lane change decisions can be made that differ in how risky they are. The options are `Nothing`, `Wait`, `Nudge`, `Force` (with the latter three having left and right variants). Nothing means to make no lane change, `Wait` will wait until it is clear to make a lane change, `Nudge` will prevent oncoming traffic in the desired lane from passing and wait till it is clear to change lanes, `Force` will simply change lanes no matter what potentially causing damage. `Nudge` can potentially decrease a vehicles reputation due to the fact its annoying. Vehicles making incorrect lane changes and turns also gamble deciding how bad the outcome of the incorrect event is.

7 MOVEMENT DECISIONS

7.1 Intersection Decision

`Vehicles` controlled by a `TextDisplay` will prompt the user when the currently controlled `Vehicle` has arrived at an intersection and can select a turn, or wait till a desired turn is available.

7.2 Lane Changing

At any point a `Vehicle` may decide to make a lane change, Currently the way for a player to do that is by a `Vehicle` controlled by a `TextDisplay`. Every simulation tick the user will be prompted to make a lane change or continue forward.

7.3 Challenge

7.4 Reputation Value

7.5 Damage

Challenges will happen automatically when any `Vehicle` makes a decision that is deemed to be 'unsafe' like a `Forced` lane change. Damage and reputation will be updated accordingly depending on the outcome of the challenge that occurred.