COSC 3P91 – Assignment 2 – 6402176, 6186076

MICHAEL WISNEWSKI, AIDAN LAROCK, Brock University, Canada

The purpose of this paper is to highlight and cover the design choices made in creating TrafficSim, a game on about avoiding traffic upon a city grid. The following paper will discuss and explain the implementation decisions made in it's JAVA code.

Michael Wisnewski, Aidan Larock. 2022. COSC 3P91 – Assignment 2 – 6402176, 6186076.

1 TRAFFICSIM

Using the layout from the original design of the TrafficSIM game, classes, interfaces, and more were broken down. The following design had a Main, Game, Map, Player, Vehicle, and GUI package.

1.1 Class Overviews

Table 1. Classes and Descriptions

Package	Class Name	Description
Game	Game	Creates the game object and is the main logic engine for the game creates map, vehicles, players
	Gamble	The gambling functionality of the game
GUI	Display	The user display (Not yet implemented)
Map	Graph	Stores the main map of road segments and intersections in an adjacency list
	Intersection	Intersection object which links road segments together
	Map	Map interface for the Intersection and RoadSegment objects
	RoadSegment	RoadSegment object which contains lists of vehicles on the road, road lanes and holds players
	Turns	An enumeration of turns that are available (Left, Right, Straight)
Players	Player	The player object which contains a vehicle and allows the player to interact with the vehicle to move around the map
Vehicle	Bus	Bus vehicle object
	Car	Car vehicle object
	SportsCar	Sports car vehicle object
	Truck	Truck vehicle object
	Vehicle	Abstract class for all vehicles which allow for easy implementation of vehicle objects

Author's address: Michael Wisnewski, Aidan Larock, Brock University, 1812 Sir Isaac Brock Way, St. Catharines, ON, L2S 3A1, Canada.

2 DESIGN CHOICES BREAKDOWN

While there were many different design and implementation choices in the code base, the major decisions are outlined bellow for easy of understanding and explanation.

2.1 Abstract Classes

Abstract classes allow for code to be reused in other classes. Classes which extend from abstract classes are able to use the methods and variables from the abstract class.

2.1.1 Vehicle. The `Vehicle` class is an abstract class from which the many types of vehicles extend from. These classes (Bus, Car, etc.) all use the methods and variables from the abstract class which reduces code and improves readability.

2.2 Anonymous Classes

Anonymous classes were used in the `Player` class to initialize AI functionality. Anonymous classes were used here because we needed specific methods in `AI` to override methods in the `Player` class while keeping the rest of the `Player` class intact for use in `AI`.

2.2.1 Player -> AI. The `AI` class is an extension of the `Player` class. Several methods in the `AI` class override methods in the `Player` class. The anonymous class allows for less code to be written for the `AI` class while still providing the same functionality that players have.

2.3 Exceptions

Exceptions are useful inside of the code in order to handle and deal with potential run time errors that may occur. `Catch` blocks cover as many potentials for errors as possible and used as infrequently as possible as they are costly to the performance of the game. Custom exceptions are also used and thrown when possible.

2.3.1 Graph. Upon loading the graphical adjacency matrix, a file reader must be used to access the data inside a `.txt` file. This type of reader may cause an exceptional event at end of file, or when reading the next number from the file. Therefore, errors must be handled with common exceptions such as the `NoFileFound` exception.

2.3.2 Player. The `Player` class allows for users to enter input with the confirm method. Because players are able to enter command, while only some commands are accepted (left, straight, right). Because of this, if players enter a command which would cause a run time exception, a custom exception was created so that the errors are easier to understand and distinctly note when players have entered wrong moves.

2.4 Generics

Generics are used so that objects and types may be treated as parameters. In the code, generics are used in the road segment class.

2.4.1 RoadSegment. In this class generics are used when creating `ArrayLists<?>` and `LinkedLists<?>` of type `Player`. It can be seen that in many cases when creating an adjacency matrix of player lanes, that it looks like `ArrayList<LinkedList<Player>>` where ArrayList takes to the type of LinkedList, which in turn LinkedList takes the type of Player. Thus, creating an array list of linked lists containing a list of players.

2.5 I/O

The program reads input from the user and text files in 2 distinct locations in the program. The `Player` class accepts input from the user, and the `Graph` class reads a map into a graph from a file. In both scenarios, readers are closed so that recourse leaks do not occur.

2.5.1 Player. In the player class a scanner is used to read in input from the `Player` through the console. A scanner is used over a buffered reader as we must parse the string for specific integer inputs such as (-1,0,1) for left, straight, and right respectively.

2.5.2 Graph. The `Graph` class reads in from a file in the assets folder. This allows for the map to be changed quickly or for the program to contain multiple maps without having to hard code the map into the code. The reader is opened only when needed and is closed immediately after.

2.6 Lambda Expressions

Lambda expressions are used within the code in order to improve readability and, grant similar functionality to that of methods without a constant static method within the code. Lambdas are used in the `Gamble` class.

2.6.1 Gamble. The method for calculating the the player who won a dice roll in gamble is a lambda expression. by separating the logic of the calculations away from the the rest of the method, the code becomes more readable and is easier to debug or update in future implementations.

2.7 Utility Classes

2.7.1 RoadSegment. This class takes a clear form of the Collections framework. More specifically an `AbstractCollection`. We chose abstract collections because it has the possibility to write your own override method from the abstract capabilities and create custom retrieval commands. A `LinkedList` and `ArrayList` is also sub-classed by the collections framework thus, when using these methods we gain the possibility to write our own implementations if the default ones do not fit the usage for this project.

3 WRAP UP

The TrafficSim engine has now been created and implemented. The game now consists of a fully functioning game engine allowing for the loading of maps, creation of players, addition of players to vehicles, turns, intersection changes and gambling features. By utilizing the appropriate concepts of object oriented programming, classes and methods are easier to read, handle bugs and run far better than a code base without such systems implemented. the TrafficSim game which implements all of the above features can be found in the ZIP file.