Project 1

CS 387
Stephen Calabro

The following document describes each class added to the project as well as how the specific controllers work to achieve the desired outcome.

Engine.Vector

For my vector class I created a modified version of the following java vector class
http://introcs.cs.princeton.edu/java/34nbody/Vector.java.html

The modifications induced only having an x and a y value, and a normalize function.

I choose to do this so that I could use a common class with common operations. Additionally, using only the x and y variables made it more relatable to the car game we were going to be working with

Controllers.Controller

For this class I just added a few pieces to make calculating the steering controllers easier. The biggest change here is the addition of the motorControl method. This method takes in a subject, target, and control variables and calculates the steering needed to get the subject to the target.

This method is the base for all of the other controllers because it is how I physically move the car once the target vector has been calculated.

Controllers.SeekController

This class is created to implement the seek steering method for the game. The class takes in a GameObject target and saves it to a private variable for use throughout the class.

The method `seek()` is where the algorithm for calculating the seek vector is. This simply implements the following from the lecture slides.

```
Seek(character, E)
D = E - character.position
ND = D / |D|
A = ND * maxAcceleration
Return A
```

After the vector is calculated it sends it to the `motorControl()` method inside of the Controller class.

Controllers.ArriveController

This class is created to implement the arrive steering method for the game. The class takes in a GameObject target and saves it to a private variable for use throughout the class.

The method `arrive()` is where the algorithm for calculating the arrive vector is. This simply implements the following from the lecture slides.

```
Arrive(character, E, targetRadius, slowRadius, time)
D = E - character.position
Length = |D|
If Length<targetRadius Return (0,0,0)
If Length>slowRadius then targetSpeed = maxSpeed
 else targetSpeed = maxSpeed * Length/slowRadius
targetVelocity = (D/|D|)*targetSpeed
A = (targetVelociy – character.velocity)/time
If |A|>maxAcceleration then A = (A/|A|)*maxAcceleration
Return A
```

After the vector is calculated it sends it to the `motorControl()` method inside of the Controller class.

Controllers.AvoidController

This class is created to implement the avoid steering method for the game. The class takes in a GameObject target and saves it to a private variable for use throughout the class.

The method `seek()` is where the algorithm for calculating the seek vector is. This simply implements the following from the lecture slides.

```
Seek(character, E)
D = E - character.position
ND = D / |D|
A = ND * maxAcceleration
Return A
```

The method `seekTurn()` is where the algorithm for calculating the seek vector is. However, this version of the method is used when there is an obstacle and the car needs to turn away. This method takes in an x and y for the target rather than using the game object.

The method `collideBox()` is used to create a `RotatedRectangle()` object for each of the directions for the car. This method then checks if there is a collision between this casted box and the obstacles in the `Game()` object. If there is the method returns true.

The `update()` method of this controller first sets a default castLength to test for collisions. This is then used with the `collideBox` method at 3 angles to test for collisions. This is front, left, and right.

If there is a collision the car will then turn right, back or left, depending on where the obstacle was. It does this by calculating a new targetX and targetY and passing them to the `seekTurn()` method in `motorControl()`. If there is no collisions the car simply executes a standard seek using the `seek()` method from before.