Programming Your Own Car

In order to develop your own artificial intelligence to control your vehicle, we developed an event-based system in the game engine. Throughout a race, the engine will throw out certain events at vehicles, and the AIs will call their corresponding method to respond to that event. The AI classes use an API that controls your vehicle, with access to how much you are accelerating and where you want your car to turn. In future releases, we hope to add more events, and more control over your car. For example, it would be cool to have access to special speed boosts and weapons your car may contain. Below is a simple explanation of the current Race Events and API Calls you will look at when programming your vehicle.

Vehicle API

You will use the following API calls to manipulate your vehicle. The list is separated into calls that will give you information, and methods you will use to control your car. More will be added to this list over time.

Set Information

setTarget(Point p): You can specify a 2D point (x,y) on the game track, that your vehicle will head towards.
setAccelerationPercent(int percent): You can think of this as how much pressure you are applying to your acceleration pedal. >100 is full throttle, and 0 is not applying any pressure at all.
setBrakePercent(int percent): Similar to setting your acceleration, this is how much pressure you are applying to your break pedal.

Get Information

getTarget(); Returns the point you are currently headed towards.
getAccelerationPercent(): Returns the last value you set the Acceleration Percent to.
getBrakePercent(): Returns the last value you set the Brake Percent to.
getAcceleration(): Returns how much you are currently accelerating.
getVelocity(); Returns how fast you are currently moving.
getDamage(); Returns how much damage your vehicle has taken. (Not yet Implemented)
getAttributes(): Returns the set of attributes about your car including Max Speed, Turning Ability, Traction, Armor, and Weight.
getCheckpoint(); Returns the next Checkpoint on the Track in relation to your vehicle. Checkpoint is a line, so you can call further methods to get more points along the line.
getLap(); Returns what lap you are currently on.
calculateMaximumTurning(int acceleration): This is a helper function that, when inputted an acceleration value, will return the maximum angle you will be able to turn based on your current speed and attributes

Utility Class (AIUtils.java)

We currently provide some helper methods that we used on the automated vehicles. They help determine how fast you should turn, and where certain checkpoints, lanes, and vehicles are.

`AIUtils.isCarAhead(Track track, Car us, Car them)`
Returns a boolean stating if the specified cars are ahead of one another or not

`AIUtils.randomUpdateCheckpoint(Car mCar)`

Returns a random point along the checkpoint line for a car to head towards.

`AIUtils.getClosestLane(CheckPoint point, Point pos)`

Returns a point (lane in the checkpoint) that is closest to the current position of your car.

`AIUtils.getAlternativeLane(CheckPoint point, Point pos)`

If you want to pass a car, you can call getAlternativeLane to get a new point to head towards in the next checkpoint.

Race Events

Setting up a AI File:

In order to start creating an AI, you must first extend the Abstract class AbstractCarListener.

```
public class MyCarAI extends AbstractCarListener {}
```

You only have to override any methods you want in the AbstractCarListener. Below are the methods that are called and some basic examples on how to program your car.

The following methods can be called at any time to get a reference to your car and the current track it is on:

getCar() and getTrack();

The following methods are in the AI interface that you will have to implement to get your car running.
onRaceStart(): This event is called once a race is ready to begin. Here you will probably want to start accelerating and choose your initial target destination.

```
@Override
public void onRaceStart() {
	getCar().setTarget(getCar().getCheckpoint().getCenter());
	getCar().setAccelerationPercent(100);

}
```

In this example, your car will simply accelerate as fast as it can towards the center of the first checkpoint line. It retrieves that location by calling your car's nearest checkpoint (getCar().getCheckpoint()), then getting the Center of it. Setting the acceleration percent to 100 accelerates your vehicle as fast as it can go.

onCheckpointUpdated(Checkpoint previousCheckpoint): The game's checkpoint system is described in more detail in a blog post on the community site, but once your vehicle reaches a checkpoint, the game will tell the car the checkpoint it has passed and you can then call getCar().getCheckpoint() to get the next checkpoint it needs to pass through. You will have to calculate where you are going and how fast you want your vehicle to be going to reach the next checkpoint. The method is passed in the previous checkpoint in case you want a reference to it for whatever reason.

```
	@Override
	public void onCheckpointUpdated(CheckPoint oldCheckpoint) {
		getCar().setTarget(getCar().getCheckpoint().getCenter());
		getCar().setAccelerationPercent(100);
	}
```

This is a simplistic example of what you can do when a checkpoint is updated (the same as when the race starts). However, will possibly cause your car to be too fast to reach the point

onOffTrack(): This event is called whenever your vehicle leaves the track. It might be a good idea to recalculate where you are heading to make sure you are going in the correct direction, or didn't get bumped off the road.

```
@Override
	public void onOffTrack() {
		getCar().setBrakePercent(100);
		getCar().setAccelerationPercent(0);
	getCar().setTarget(getCar().getCheckpoint().getIntersectionPoint(getCar
().getRotation(), getCar().getPosition()));
}
```

When this car gets off the track, it will slam on the breaks and set a new target, at the intersection point between its current location and checkpoint.

onOpponentInProximity(Car otherCar): When you are close to another vehicle, this event is called so you can try to avoid it, pass it, crash into it, etc...

```
	@Override
	public void onOpponentInProximity(Car car) {

		if (AIUtils.isCarAhead(getTrack(), getCar(), car)) {
			getCar().setTarget(car.getTarget());
			getCar().setAccelerationPercent(100);
			getCar().setBrakePercent(0);
		}
	}
```

This car tries to ram into the car in front of it.

```
        @Override
        public void onOpponentInProximity(Car car) {

    getCar().setAcceleration(20);
Car().setTarget(AIUtils.getAlternativeLane(getCar().getCheckpoint(), getCar().getPosition()));


        }
```

```
While this car tries to avoid hitting it all together.
```

onCarCollision(Car otherCar): When you do crash into another car, intentionally or not, this event is thrown so you are able to re-orient yourself.

```
        @Override
        public void onCarCollision(Car other) {

                if (getCar().getTarget().equals(other.getTarget())) {
                        getCar().setTarget(AIUtils.getAlternativeLane(
                        getCar().getCheckpoint(),getCar().getPosition()));
                }
                getCar().setAccelerationPercent(0);
                getCar().setBrakePercent(50);
        }
```

```
This car tries to change lanes once it hits another vehicle.
```

onObstacleInProximity(Obstacle obstacle): Similar to the opponent events, these events tell you when you are coming up on an in-game obstacle, such as an oil spill, or blimp.
onObstacleCollision(Obstacle obstacle): Similar to onCarCollision, this will help you re-orient yourself if an obstacle does affect your position.
onStalled(): If your vehicle stops moving for any reason, this event is called so you can adjust whatever you need to.
onTimeStep(): Every world time step, this event is called. That is, about 60 times a second. If you want to throw in any other logic that doesn't fit into those events, you can use this method as a catch-all to figure out what you want to do.