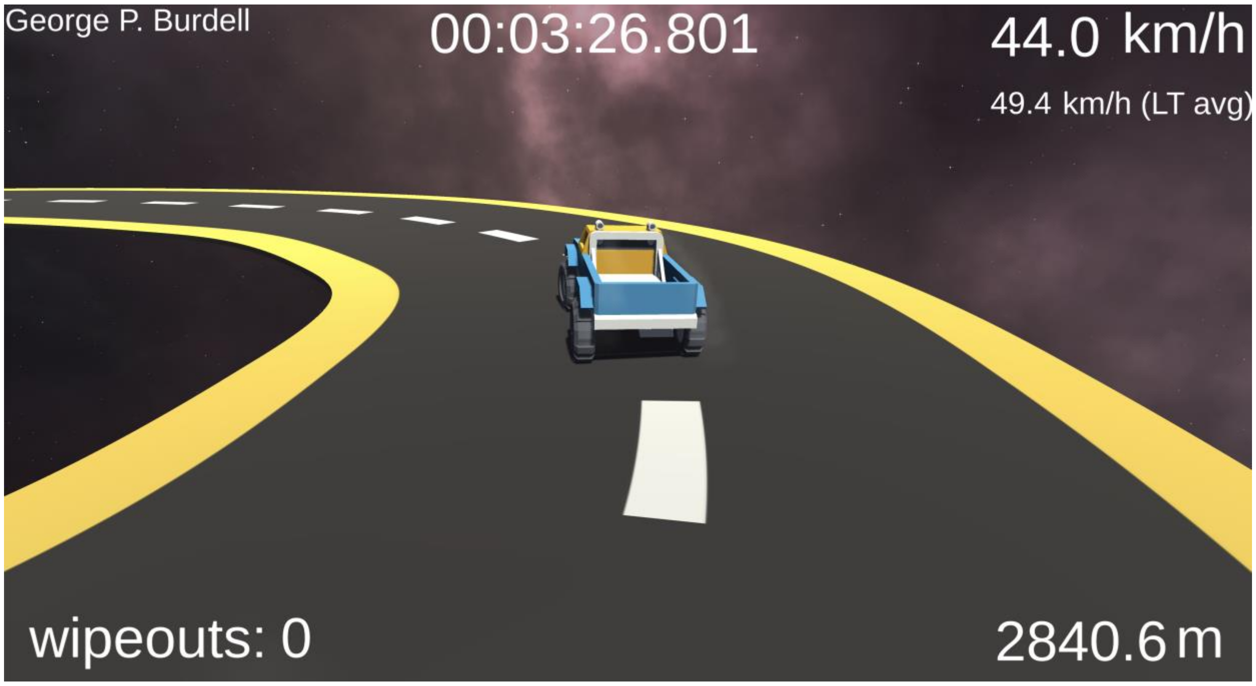
Race Track



In this assignment you will be implementing a Fuzzy Logic Agent that can race on a procedurally generated track. Optionally, you can also train a Neural Net agent to race on the same track.

Your agents will go as fast as they can without crashing, getting, stuck, turned around, etc.

The vehicle is a complex physically simulated truck provided by the Free Unity Asset Store Package: Arcade car physics by saarg. You can try driving it yourself if you open the /scenes/RaceTrackHuman. It is quite challenging with interesting dynamics.

Fuzzy Logic

A fuzzy logic library is provided for you to use. It was developed by t0chas on Github and is based on Buckland's *Programming Game AI by Example*. See https://github.com/t0chas/FuzzyLogic

Note that some minor changes have been made to the library such as fixing spelling. Additionally, there is *Alternate Fuzzy Rules* syntax (see below) that you might find easier for defining human readable rules.

You will use what you have learned from the Fuzzy Logic lecture to build Fuzzy Logic control of your racing truck.

Check out the Weapons example that is straight from the Buckland book. It is a good example to refer to designing your Fuzzy Logic driving agent.
https://github.com/t0chas/FuzzyLogic/blob/master/UnitTest/WeaponsExample.cs
Weapons Example of full evaluation:
(Refer to link above for fuzzy set declaration functions)

```
public void TestInference()
{
  var desirability = this.GetDesirabilitySet(); // output fuzzy set
  var distance = this.GetDistanceToTargetSet(); // input fuzzy set
  var ammo = this.GetAmmoStatusSet(); // input fuzzy set
  // rules for interpreting fuzzy inputs and determining fuzzy output
  var rocketRuleSet = this.GetRocketLauncherRuleSet(desirability);
  FuzzyValueSet inputs = new FuzzyValueSet(); // stores fuzzy inputs
  distance.Evaluate(200f, inputs); // store distance fuzzy input based on crisp input
  ammo.Evaluate(8f, inputs); // store ammo fuzzy input based on crisp input
  var result = rocketRuleSet.Evaluate(inputs); // evaluate fuzzy rules
  // crisp value is returned and then tested here:
  Assert.AreEqual(60.625f, result, 0.25f);
}
```

You need to develop metrics related to the vehicle state and the track that can be fuzzified, and interpreted by fuzzy rules and then defuzzified to crisp outputs (Throttle and Steering).

Neural Net

[NOTE: The Neural Net part of this project is not graded and is not required to be completed.]

You will use the ml-agents Unity Package to train a neural net to drive the racing truck.

Follow the installation instructions here to set up the tools that you will need:
https://docs.unity3d.com/Packages/com.unity.ml-agents@2.1/manual/index.html

Note that you need the dependencies that match **ML-Agents Release 18 Package** (see https://github.com/Unity-Technologies/ml-agents/releases). Note that the Unity project should already have **com.unity.ml-agents (C#) v2.1.0-exp.1** imported.

You will use the ML Agents package and tools to train your agent to drive the truck effectively on the track.

Tips for the neural net are provided below, but it is recommended that you read through all the introductory documentation that Unity provides.

Vehicle

Your agent (either FuzzyVehicle.cs or NNVehicle.cs in Scripts/GameAIStudentWork/) extends a controller for the vehicle. For FuzzyVehicle you indirectly set Throttle [-1, 1] (brake through full throttle) and Steering [-1, 1]. This is done by passing FuzzyRuleSets for throttle and steering to **ApplyFuzzyRules**<>(). Also, you can access the Speed (km/h), transform position, and other settings. You should not use the hand brake, boost, or other advanced features, or change the default settings.

```
ApplyFuzzyRules<T, S>(
            //T – enum type for throttle fuzzy states,
            //S – enum type for steering fuzzy states,
           FuzzyRuleSet<T> throttleFRS, //throttle fuzzy rule set
           FuzzyRuleSet<S> steerFRS, //steering fuzzy rule set
           FuzzyValueSet fuzzyValueSet, // input fuzzy value set
            //intermediate values useful for debugging follow
           out FuzzyValue<T>[] throttleRuleOutput,
           out FuzzyValue<S>[] steeringRuleOutput,
           ref FuzzyValueSet mergedThrottle,
           ref FuzzyValueSet mergedSteering
)
```

- This method evaluates your fuzzy rules according to the input fuzzyValueSet and sets the crisp outputs to Throttle and Steering. This is the only way to assign Throttle and Steering in FuzzyVehicle.cs. (Note that NNVehicle.cs has direct access to Throttle and Steering)

HardCodeSteering(f)

- For debugging, you can hardcode the steering value. Use of this method causes autograder to fail, so don't use it in your submission.

HardCodeThrottle(f)

- For debugging, you can hardcode the throttle value. Use of this method causes autograder to fail, so don't use it in your submission.

Speed – Measured in km/h. Divide by 3.6 for m/s.
transform – Access WCS position, forward and up vectors
Velocity – Vector3 velocity in m/s
AngularVelocity

Procedurally Generated Track

The procedurally generated track is based on SebLague's Path-Creator, a Bézier curve tool for Unity. https://github.com/SebLague/Path-Creator

It has been heavily modified to support real time procedurally generated curves.

Your agent (either FuzzyVehicle.cs or NNVehicle.cs in GameAIStudentWork/) can access details about the track with the PathTracker. These are useful for generating crisp input values for the Fuzzy Logic or for Neural Net observations.

Useful PathTracker details:

pathTracker.halfRoadWidth

- Access to constant road half width from center (5m).

pathTracker.distanceTravelled

- The distance (m) the car is from the beginning of the current partial track (since it is always adding new segments and deleting old segments, this value changes relative to the current partial track)

pathTracker.totalDistanceTravelled

- The overall distance (m) the car has traveled in the forward track direction since the beginning of the race regardless of the current track path segments

pathTracker.closestPointOnPath

- The Vector3 world position that is on the path closest to the car (will be on track centerline)

pathTracker.closestPointDirectionOnPath

- The curve tangent of the closest point on path to car (see above)

pathTracker.currentBezierSegmentIndex

- The index into the array of Bézier segments that the car is closest to

pathTracker.currentClosestPathPointIndex

- The index into the array of discretized linear segments that the car is closest to

pathTracker.maxPathDistance

- The full length of the current segments (m)

pathTracker.pathCreator.bezierPath

- The curved path. This data is difficult to work with and is discretized in a linear format as “path” (see below)

pathTracker.pathCreator.path

- This is the discretized linear form of the path representing the Bézier path

pathTracker.pathCreator.path.GetPointAtDistance(v)

- This method gets the world Vector3 point on the path a certain distance down the path, possibly offset from where the car is

pathTracker.pathCreator.path.GetDirectionAtDistance(v)

- This method gets the tangent on the path a certain distance down the path, possibly offset from where the car is

pathTracker.pathCreator.path. GetPointAndDirAtDistance (v)

- Get both point and path tangent at distance at same time

pathTracker.pathCreator.path.GetClosestPointOnPath(p)

- Get the closest point on path to position p

pathTracker.pathCreator.path.GetClosestPointDirectionOnPath(p)

- Get the tangent at point on path closest to position p pathTracker.pathCreator.path.GetClosestPointAndDirectionOnPath(p)

- Get both closest point on path and tangent at that point, closest to position p

NOTE: There are some public methods used in tracking vehicle performance. You cannot make calls to override performance metrics. Static code analysis will be performed on your submitted code to detect calls not allowed. If found, your submission will not be accepted.

Alternate Fuzzy Rules

Previous OMSCS student, Bob Kerner, made an extension to the Fuzzy Rules framework. You can write rules in an easier to read form if you like:

```
var eatRules = new List<FuzzyRule<FzEat>>() {
        If(And(FzTaste.Sweet, FzAmount.Tiny)).Then(FzEat.Chomp),
        If(Or(FzTaste.Sour, FzAmount.Large)).Then(FzEat.Spit),
        ...
};
```

He also implemented:

- Added Nand, Nor
- Implemented Fairly, Very (not done in the current online framework)
- And/Or/Nand/Nor now take up to 4 expressions instead of being limited to 2. (Extensible, too.)

These extensions are in: AIVehicle.Ext.cs

Grading Criteria

Your Fuzzy Logic agent must achieve a 45 Km/h average speed on the track over 5 minutes from the start with no more than one wipeout (fall, stuck, flip, turned around, etc.) for full credit. Extra credit may be earned for exceeding the requirements (> 45 km/h and 0 wipeouts). See the RacingTest.cs test for the partial credit and extra credit calculation. Note that the same track configuration that you are provided with in the test is used for grading.

Fuzzy Logic Implementation Penalties

You must control your agent exclusively with Fuzzy Logic Rules. You cannot circumvent the Fuzzy Logic interface in order to assign steering or throttle by some other means. Both throttle and steering must utilize Fuzzy Logic rules that dynamically and smoothly adjust across the [-1.0, 1.0] crisp output ranges. You aren't implementing Fuzzy Logic unless the result is "fuzzy". This generally means that:

- Fuzzy Input States within a Fuzzy Set are associated with Degree of Membership functions that overlap in the x-axis (crisp value) range.
- More than one Fuzzy Rule for a particular Fuzzy Output Set should be activated at the same time in most cases (largely due to the first point above).

- Both Fuzzy Input Sets and Output Sets should almost always have at least three states each (and definitely at least two).
- Fuzzy Rule Sets should have more than one rule, and generally at least as many as the Fuzzy Output set (these states show up in rule consequent).

Preparing crisp input values for fuzzification should involve minimal conditional logic, if any. For instance, an if statement used to conditionally apply a sign to a *vehicle-distance-from-center-line* is fine (though multiplying by the result of Mathf.Sign(expr) is better). However, a hard-coded decision tree to create an abstract "*steering-intent*" crisp value is not. All the decision-making logic should reside exclusively in the Fuzzy Rules. In general, the crisp input values should come from relative distances and angles or vehicle state such as speed.

Your submission will be subject to penalties if you don't implement a Fuzzy Logic solution. A combination of static code analysis, runtime monitoring, and TA code check will be used to assess submissions.

You must change the HUD student name as well.

Your Neural Net agent will not be submitted but you can participate on Piazza regarding training ideas, performance, etc.

Submission:

You will submit FuzzyVehicle.cs. Don't forget to change your HUD name!

Machine Learning with ML Agents Tips:

What is Proximal Policy Optimization (PPO)?

https://openai.com/blog/openai-baselines-ppo/

Any guidelines to number of layers/nodes?

https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

Guidance on rewards:

https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Design-Agents.md

Using an Environment Executable (faster training) https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Executable.md#using-an-environment-executable

Also helps to reduce graphics demands by setting windowed mode with low resolution

Might need to increase buffer size per: https://forum.unity.com/threads/num-envs-n-help-what-are-unity-instances.975918/

Training Using Concurrent Unity Instances (faster training)

https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-ML-Agents.md#training-using-concurrent-unity-instances