# Nanodegree Engenheiro de Machine Learning

## Capstone Project

Cristian Carlos dos Santos 22 de setembro de 2019

## I. Definition

### Project Overview

#### Summary

The project is themed on AWS Deep Racer from Amazon. As Amazon itself describes:

“AWS DeepRacer is a reinforcement learning (RL) -enabled autonomous 1 / 18th-scale vehicle with supporting services in the AWS Machine Learning ecosystem. It offers an interactive learning system for users of all levels to acquire and refine their skill set in machine learning in general and reinforcement learning in particular. You can use the AWS DeepRacer console to train and evaluate deep reinforcement learning models in simulation and then deploy them to an AWS DeepRacer vehicle for autonomous driving. You can also join AWS DeepRacer League to race in the online Virtual Circuit or the in-person Summit Circuit. ” [1]

This model of competition and RL learning makes us learn about the subject which is a fascinating but very complex field of study. Recently Udacity launched the AWS DeepRacer Scholarship Challenge and as I had not selected my proposal for the final work I decided to accept this challenge.

### Problem Statement

The goal of my project is to develop and make the cart reach the end by completing 100% of the track. However, as the cost with AWS DeepRacer can quickly become high, I will look for the best solution with few features. My personal goal is $ 50.

#### Datasets and Entries

AWS DeepRacer trains models using the Proximal Policy Optimization (PPO) algorithm. According to AWS DeepRacer course “Value Functions” (L4: Reinforcement Learning) video [2], this algorithm is used because it is efficient, stable and easy to use compared to other algorithms. The Algorithm uses two neural networks during training: Policy Network (Actor-Network) and Value Network (Critic Network).

- Policy Network: Decides what action to take according to the image received in the input.
- Value Network: Estimates the cumulative result we are likely to get, considering the image as an input.

#### Reward

The reward function will be the guide of the algorithm. It will reward positive actions, ie what I would like the cart to accomplish. Besides, it will also penalize unwanted actions such as getting off track during the race. For the construction of the reward function, we have an input of a variable called `params`. This variable is a library in the following format:

```
{
    "all_wheels_on_track": Boolean,    # flag to indicate if the vehicle is on the track
    "x": float,                        # vehicle's x-coordinate in meters
    "y": float,                        # vehicle's y-coordinate in meters
    "distance_from_center": float,     # distance in meters from the track center
    "is_left_of_center": Boolean,      # Flag to indicate if the vehicle is on the left side to the track cen
    "heading": float,                  # vehicle's yaw in degrees
    "progress": float,                 # percentage of track completed
    "steps": int,                      # number steps completed
    "speed": float,                    # vehicle's speed in meters per second (m/s)
    "steering_angle": float,          # vehicle's steering angle in degrees
    "track_width": float,              # width of the track
    "waypoints": [[float, float], … ], # list of [x,y] as milestones along the track center
    "closest_waypoints": [int, int]    # indices of the two nearest waypoints.
}
```

More information on the reward function is available in the Amazon Deep Racer developer documentation. [3].

### Hyperparameters

Algorithmic hyperparameters are options available in the above mentioned neural networks. With them, we can accelerate role adjustments and define key points in the Reinforcement Learning process, such as having our algorithm prioritize highest reward actions ever discovered or explore more actions to find a better action in a given state. In the AWS Deep Racer console we have the following parameters:

- Gradient descent batch size
- Number of epochs
- Learning rate
- Entropy
- Discount factor
- Loss type
- Number of experience episodes between each policy-updating iteration - (Number of experience episodes between each policy update iteration)

More detailed information about each can be found in the AWS Deep Racer Developer's Guide in the Parameter Settings section. [4].

### Solution

To solve the proposed problem I will use as a starting point the examples of reward function and hyperparameters available in the AWS Deep Racer documentation examples.

### Reward

For the reward function, I will use ways to maximize expected actions such as staying on track, gaining speed and completing laps. For this, the idea is to use as the basis of the reward function the parameters `distance_from_center` and `all_wheels_on_track`. Also, I will use the progress multiplier as the reward multiplier so that the reward will be higher as the track progresses.

### Hyperparameters

Para os hiperparâmetros, conforme vídeo “Intro to Tuning Hyperparameters” (L5: Tuning your model - AWS DeepRacer Course): “Figuring out what works best for your model is usually done through trial and error.”

### Metrics

As an evaluation metric, the idea is to use **track completion percentage**, **lap completion time**, and visualization of rewards by iteration. To facilitate this analysis I will use the Jupyter notebook available in the article “Using Jupyter Notebook for analysing DeepRacer’s logs” [5].

# II. ## Analysis

## ### Data Exploration - Training

From the training and assessment performed on the AWS Deep Racer console, training and assessment log files are generated. Using the analysis model available in the AWS Deepracer Workshop Lab Github [6] you can download these logs and start the evaluation through the `log-analysis / DeepRacer Log Analysis.ipynb` file. In addition to the training/evaluation logs, a model is generated for the neural networks that are part of the algorithm already explained above. You can download this model from the AWS console so that it can also be evaluated on the Jupyter Notebook already mentioned above.

My initial goal was to find, with the reward function based on the basic standards of the documentation examples and standard hyperparameters, the shortest training time required for the cart to complete one lap. The minimum time I found on the track **re:Invent 2018** was 1H. Following is the reward function used and the hyperparameters:

**Reward Function:**

```
def reward_function(params):
    # Read input parameters
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']
    all_wheels_on_track = params['all_wheels_on_track']
    steering = abs(params['steering_angle']) # Only need the absolute steering angle
    progress = params['progress']
    speed = params['speed']
    SPEED_THRESHOLD = 1.0
    SPEED_THRESHOLD_3 = 3.0
    # Steering penality threshold, change the number based on your action space setting
```

```
    ABS_STEERING_THRESHOLD = 20

    # Calculate 3 markers that are at varying distances away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    if not all_wheels_on_track:
        # Penalize if the car goes off track
        reward = 1e-3
    elif speed < SPEED_THRESHOLD:
        # Penalize if the car goes too slow
        reward = reward - 0.1
    else:
        # High reward if the car stays on track and goes fast
        reward = reward * speed

    if steering > ABS_STEERING_THRESHOLD:
        # Penalize reward if the agent is steering too much
        reward *= 0.8

    reward = reward + (reward * (progress / 100))

    return float(reward)
```

This reward function is to keep the cart on track without major zigzag and adding a multiplier for speed.

**Hyperparameters:**

| Hyperparameters | Value |
|---|---|
| Gradient descent batch size | 64 |
| Entropy | 0.01 |
| Discount factor | 0.999 |
| Loss type | Huber |
| Learning rate | 0.0003 |
| N. of experience episodes between each policy-updating iteration | 20 |
| Number of epochs | 10 |

Hyperparameters will be kept initially.

### Log Structure

By using the `DeepRacer Log Analysis.ipynb` notebook we can download the logs straight from AWS as long as the AWS Client is configured. The files needed to run the notebooks evaluated in this report will all be available in the **log-analysis /** (logs) and **log-analysis / intermediate_checkpoint** (templates) folder.

### Analysis Notebook

Initially, we will evaluate the resulting 1-hour training notebook with the functions defined above. It is located in the `log-analysis /` folder named `06 - DeepRacer Log Analysis - MyTrain 60 min speed.ipynb`.

Training logs after being uploaded to a DataFrame are displayed as shown below:

| | iteration | episode | steps | x | y | yaw | steer | throttle | action | reward | done | on_track | progress | closest_waypoint | track_len | timesta |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 305.00 | 68.32 | 0.0003 | 0.00 | 0.0 | 0.0 | 0.9071 | 0 | True | 0.7922 | 0 | 17.67 | 1567385469.347 |
| **1** | 1 | 0 | 1 | 305.00 | 68.32 | 0.0001 | 0.26 | 0.5 | 6.0 | 0.9071 | 0 | True | 0.7923 | 0 | 17.67 | 1567385469.4232 |
| **2** | 1 | 0 | 2 | 305.22 | 68.17 | -0.0113 | 0.52 | 0.5 | 8.0 | 0.7258 | 0 | True | 0.8048 | 0 | 17.67 | 1567385469.4867 |
| **3** | 1 | 0 | 3 | 305.63 | 67.90 | -0.0304 | -0.52 | 1.0 | 1.0 | 0.8066 | 0 | True | 0.8278 | 0 | 17.67 | 1567385469.5519 |
| **4** | 1 | 0 | 4 | 305.74 | 67.81 | -0.0379 | 0.00 | 1.0 | 5.0 | 1.0083 | 0 | True | 0.8341 | 0 | 17.67 | 1567385469.6183 |

Para avaliar as estatísticas selecionei as colunas que achei mais importantes para a avaliação:

| | iteration | episode | steps | x | y | throttle | reward | progress | track_len |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 29947.000000 | 29947.000000 | 29947.000000 | 29947.000000 | 29947.000000 | 29947.000000 | 29947.000000 | 29947.000000 | 2.994700e+04 |
| **mean** | 1.485357 | 18.765753 | 461.961666 | 371.896770 | 217.855306 | 0.776956 | 1.051539 | 45.726766 | 1.767000e+01 |
| **std** | 0.499794 | 12.113513 | 290.214894 | 216.021933 | 139.219151 | 0.249887 | 0.443145 | 28.274458 | 4.895721e-12 |
| **min** | 1.000000 | 0.000000 | 0.000000 | 28.570000 | 26.580000 | 0.000000 | 0.000000 | 0.763300 | 1.767000e+01 |
| **25%** | 1.000000 | 8.000000 | 208.000000 | 159.070000 | 73.540000 | 0.500000 | 0.766700 | 20.972750 | 1.767000e+01 |
| **50%** | 1.000000 | 19.000000 | 445.000000 | 365.060000 | 206.570000 | 1.000000 | 1.073400 | 44.044500 | 1.767000e+01 |
| **75%** | 2.000000 | 30.000000 | 704.000000 | 563.155000 | 331.050000 | 1.000000 | 1.365050 | 69.465650 | 1.767000e+01 |
| **max** | 2.000000 | 39.000000 | 1138.000000 | 767.160000 | 495.350000 | 1.000000 | 2.000000 | 100.000000 | 1.767000e+01 |

In the image it is possible to evaluate the following points:

- **Iteration (2) and episode (39)**: We have a low number of iterations and episodes due to the short training time.
- **Throttle:** With an average of 0.7765 it is possible to estimate that the trolley has been accelerating much of the training time.
- **Reward:** We have a reward average of 1.05. If we are to evaluate the current reward function, we can see that this reward is awarded when the cart is too close to the center of the track. Since the focus of the race is lap time, possibly the basis of the reward cannot be the distance from the center of the track.
- **Progress:** Well, we reached 100%.

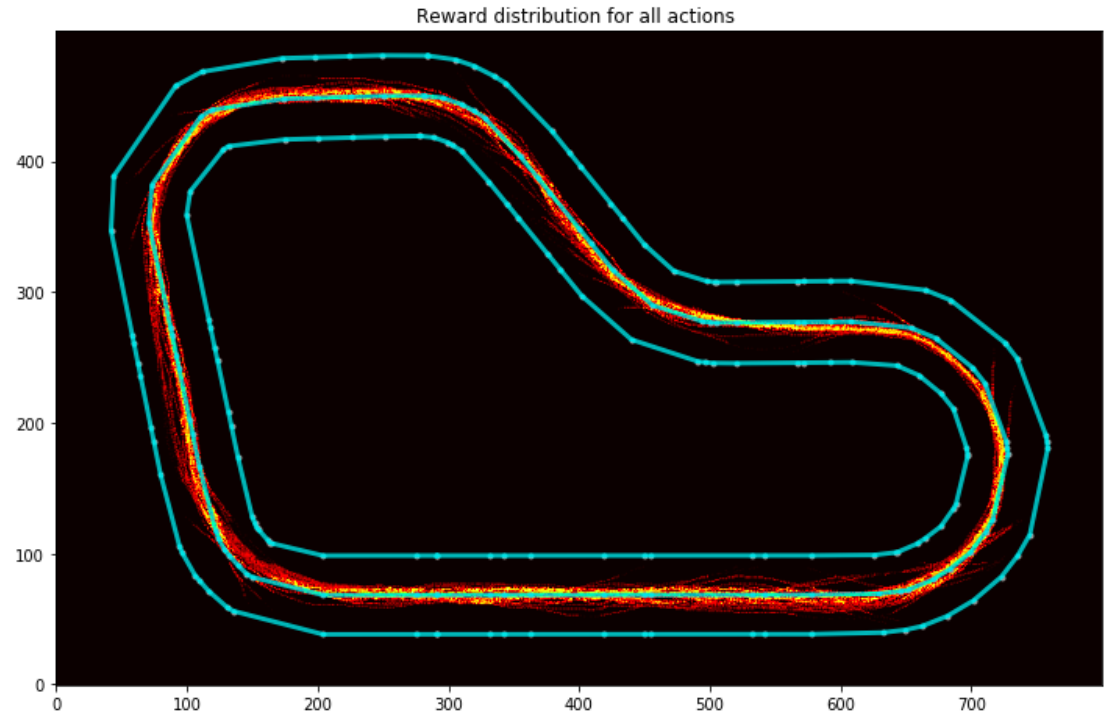
## ### Exploratory Visualization - Training

#### Action reward assessment

For evaluation of the following graphs, it is necessary to take as the base that the X-axis and Y-axis are, respectively, columns X and Y of the Dataframe evaluated in the section "analysis notebooks".

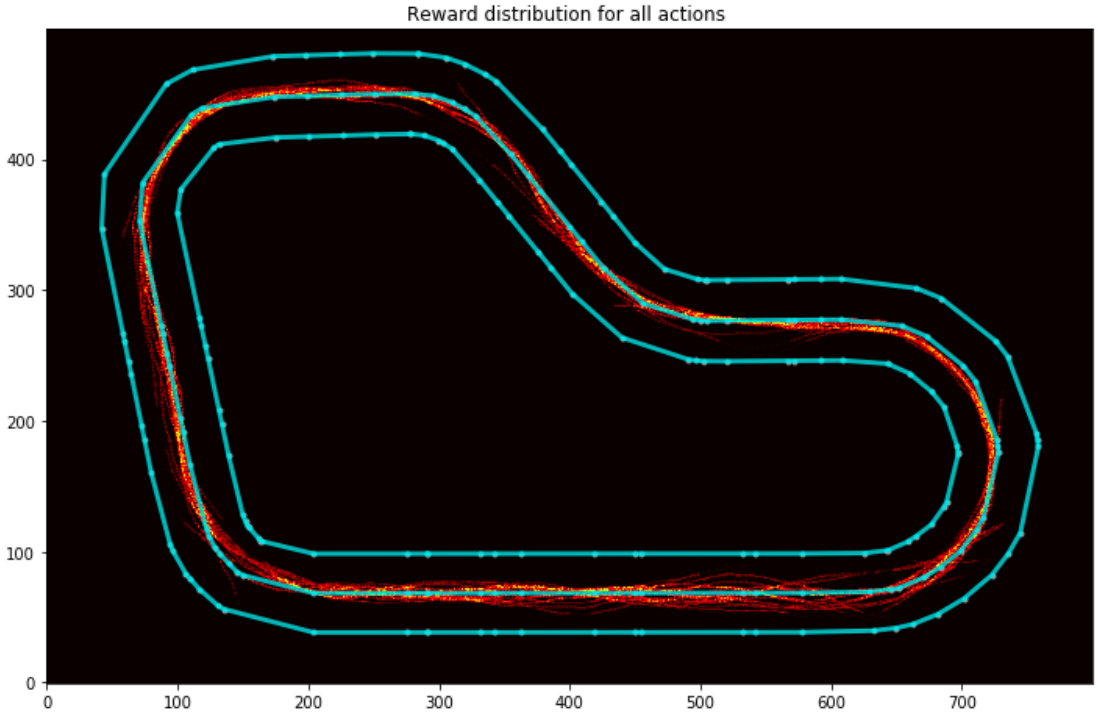
##### General evaluation

In the image below you can see the rewards returned in the training process.

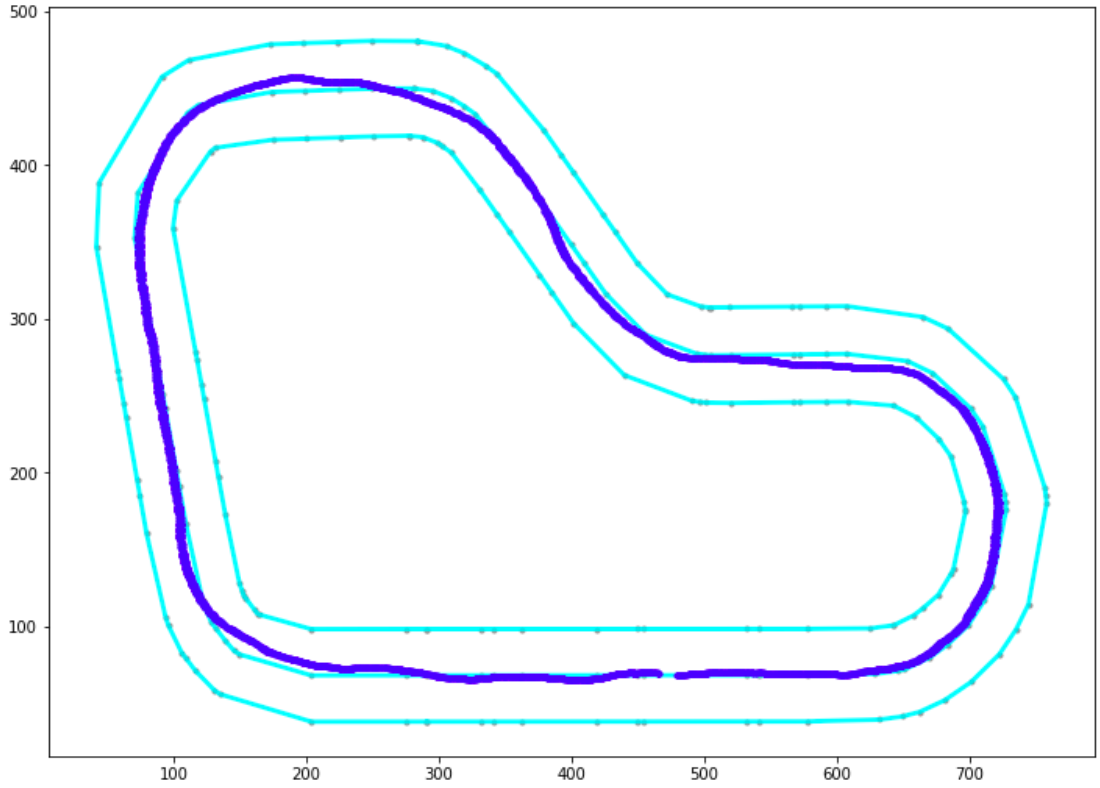



**Evaluation Iteration 2**

In the image below you can see the rewards of a specific training iteration.




**Path taken for top reward iterations**




In the images above it is possible to confirm what was raised in the "Analysis Notebook" section regarding the issue of the reward having an average close to 1 because the basis of the reward composition is the distance from the center of the track. Both in the overall evaluation image and the evaluation image of a specific iteration, in this case, iteration 2, it is seen that the reward distribution is very close to the center of the track.

Besides, the path taken for the most rewarding iterations is extremely aligned with the center of the track.

## Exploratory Visualization - Training v1

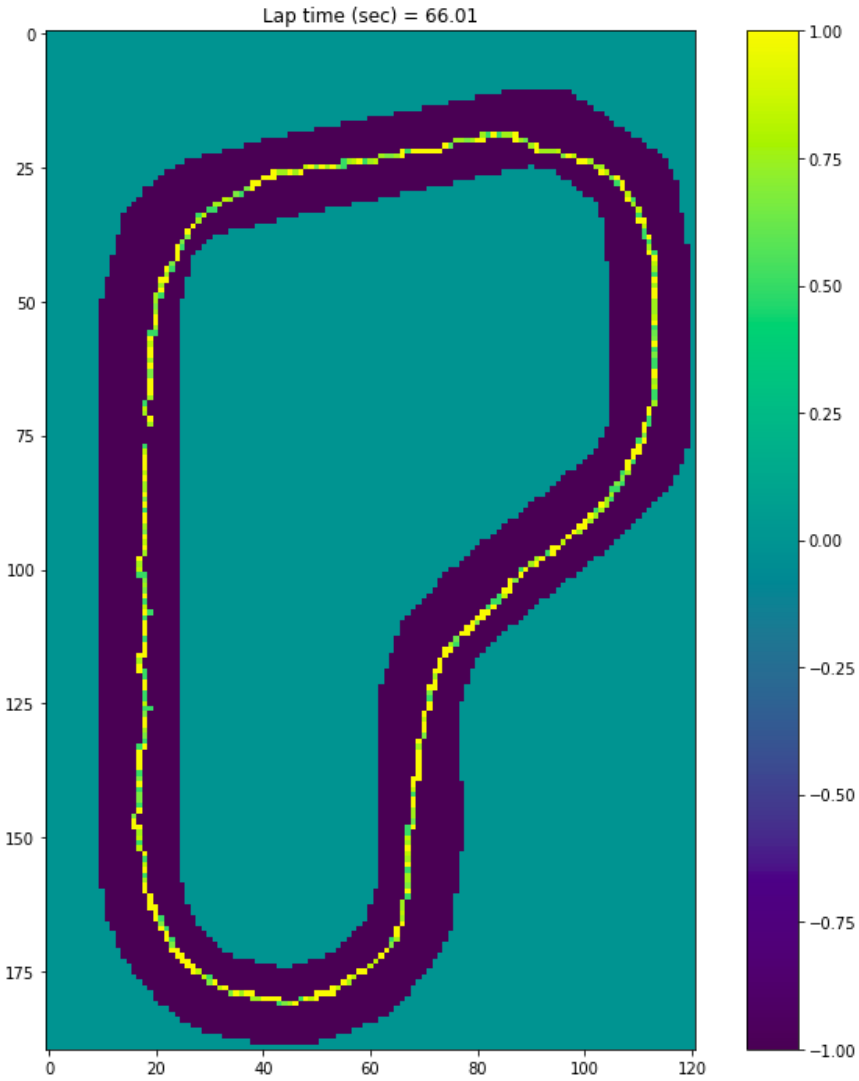
**AWS assessment result:**

Evaluation results

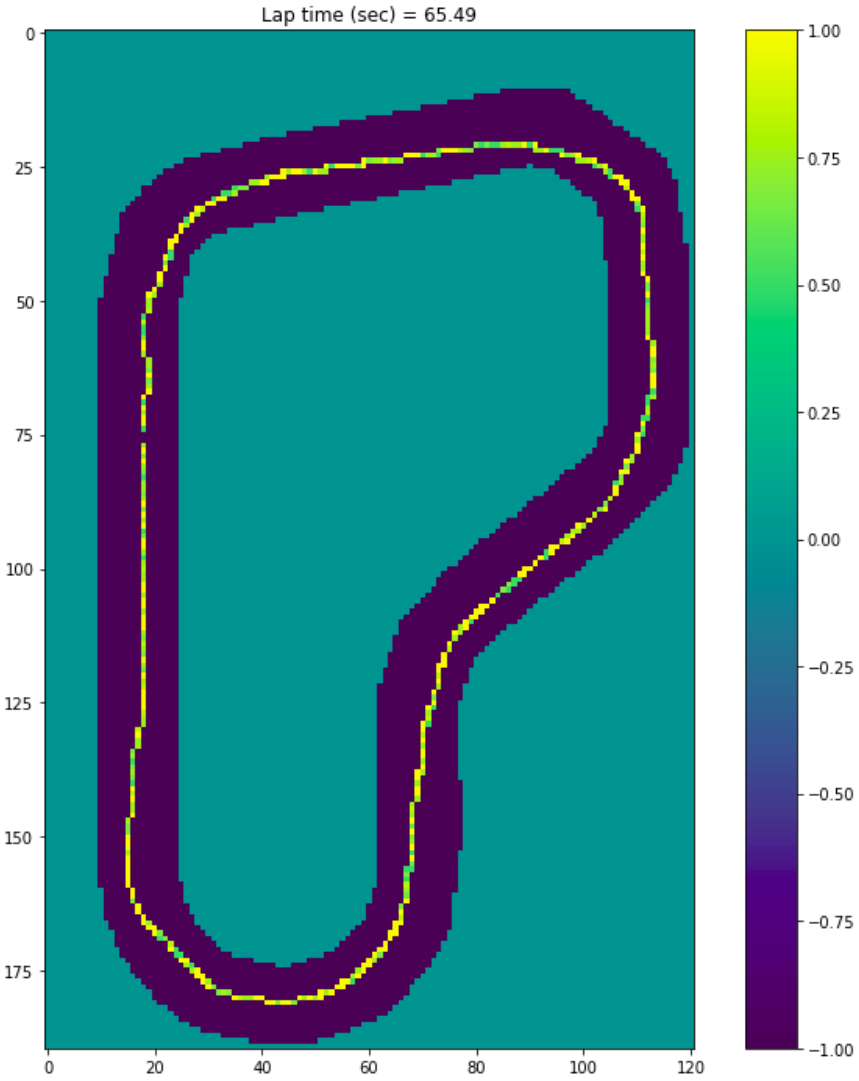
| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:01:10.679 | 100% |
| 2 | 00:01:08.608 | 100% |
| 3 | 00:01:16.482 | 100% |

In the evaluation stage we have to understand how the cart behaved on the track during the three attempts to complete the race.

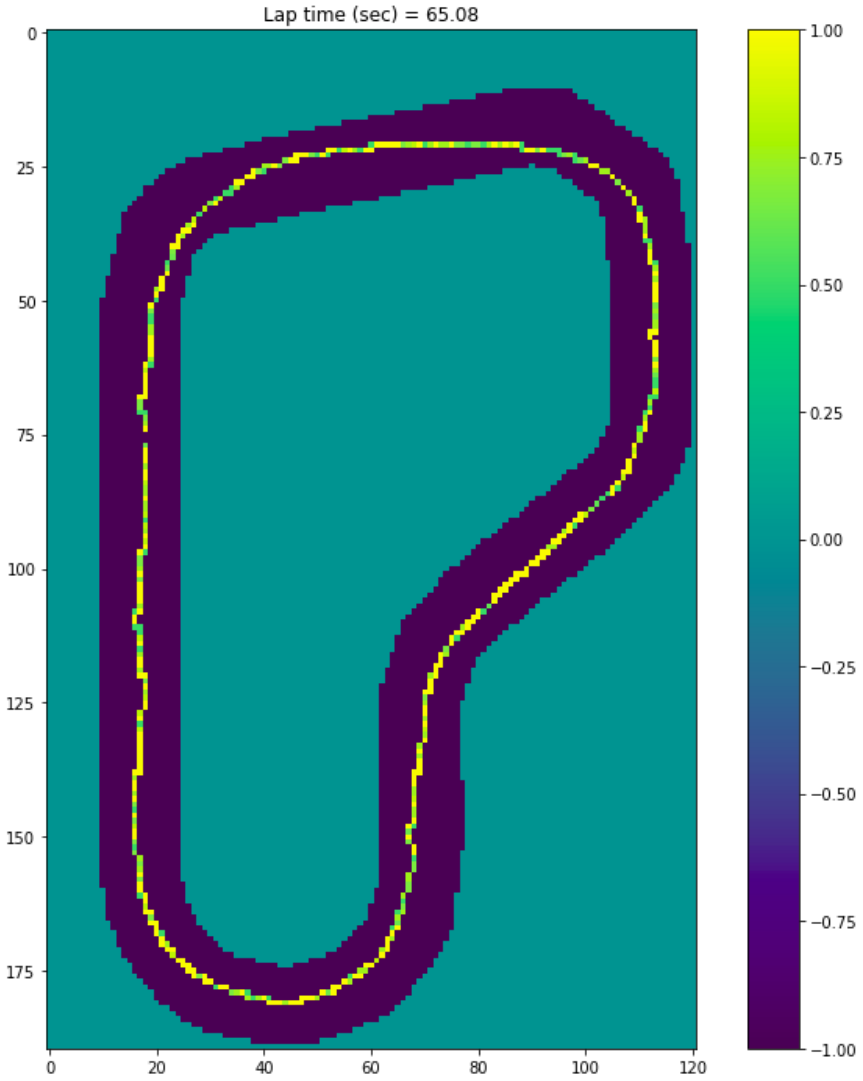
** Attempt 1: **


Lap time (sec) = 66.01
0
25
50
75
100
125
150
175
0
20
40
60
80
100
120
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

**Attempt 2:**


Lap time (sec) = 65.49
0
25
50
75
100
125
150
175
0
20
40
60
80
100
120
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

**Attempt 3:**




You can see that the same training pattern is repeated during the evaluation phase. The Cart tried to stay in the center of the track, sometimes sacrificing speed to make it happen.

## Algorithms and Techniques

As described in step **I.Definition** AWS DeepRacer trains models using the Proximal Policy Optimization (PPO) algorithm. This project is an algorithm optimization project already defined and used by AWS Deep Racer. In the **Data Sets and Inputs** and **Data Exploration** steps the initial default reward function and the hyperparameters used as the starting point for analysis have already been described.

## Benchmark

As a reference, I am based on the “re: Invent 2018” winner who completed the lap in 12.68 secs [7]. For this model the following reward function was used:

```
def reward_function(params):
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']
    all_wheels_on_track = params['all_wheels_on_track']
    speed = params['speed']
    SPEED_THRESHOLD = 1.0


    # Calculate 3 markers that are at varying distances away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
```

```
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    if not all_wheels_on_track:
                # Penalize if the car goes off track
        reward = 1e-3
    elif speed < SPEED_THRESHOLD:
                # Penalize if the car goes too slow
        reward = reward + 0.5
    else:
                # High reward if the car stays on track and goes fast
        reward = reward + 1.0

    return float(reward)
```

In hyperparameters, the following were used:

| Hyperparameter | Value |
|---|---|
| Gradient descent batch size | 64 |
| Entropy | 0.01 |
| Discount factor | 0.666 |
| Loss type | Huber |
| Learning rate | 0.0003 |
| N. of experience episodes between each policy-updating iteration | 20 |
| Number of epochs | 10 |

From these benchmarks, I want to walk the path between this great benchmark and the default parameters. If possible, further refine the model and test the results. **However, it is important to note that the training time used by the competitor was not reported**.

# III. Methodology

## Data Preprocessing

As assessed in the **Data Exploration - Training** section the standard reward function used showed a strong tendency to keep the cart in the center of the track. This is not all bad, but when it comes to a race, the idea is that the lap time will be better.

Because of this, I changed the reward function based on the speed of the cart. Another big change is that the distance from the center of the track will have greater flexibility, with discounts, even if few, at the ends of the track. I kept the reward bonus as the track progressed and the penalty to avoid zigzagging.

Thus, the current reward function was as follows:

```
def reward_function(params):
    # Read input parameters
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']
    all_wheels_on_track = params['all_wheels_on_track']
    steering = abs(params['steering_angle'])  # Only need the absolute steering angle
    progress = params['progress']
    speed = params['speed']
    # Steering penality threshold, change the number based on your action space setting
    ABS_STEERING_THRESHOLD = 20

    if not all_wheels_on_track:
        # Penalize if the car goes off track
        reward = 1e-3
    else:
        reward = speed
```

```
    # Calculate 3 markers that are at varying distances away from the center line
    marker_1 = 0.4 * track_width
    marker_2 = 0.45 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward *= 1
    elif distance_from_center <= marker_2:
        reward *= 0.9
    elif distance_from_center <= marker_3:
        reward *= 0.85
    else:
        reward = 1e-3  # likely crashed/ close to off track

    if steering > ABS_STEERING_THRESHOLD:
        # Penalize reward if the agent is steering too much
        reward *= 0.8

    reward = reward + (reward * (progress / 100))


    return float(reward)
```

## Implementation

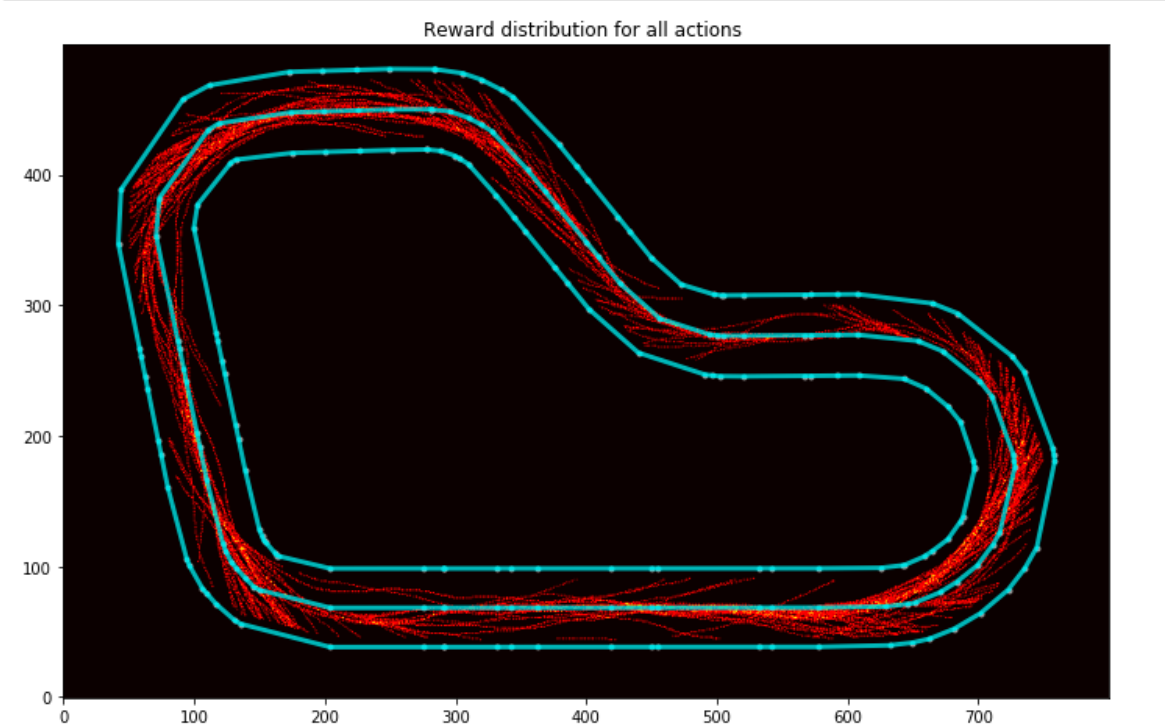
After adjusting the reward function as specified in the **Data Preprocessing** section, 1-hour training was conducted to evaluate the effectiveness of this reward function during the same training time. An important note is that no changes were made to the hyperparameters initially. The analysis notebook is in the `log-analysis /` folder named `09 - DeepRacer Log Analysis - Final 60 min.ipynb`.

### Exploratory Visualization - Training v2

With this training the following results were obtained:

#### Overall Share Reward Rating

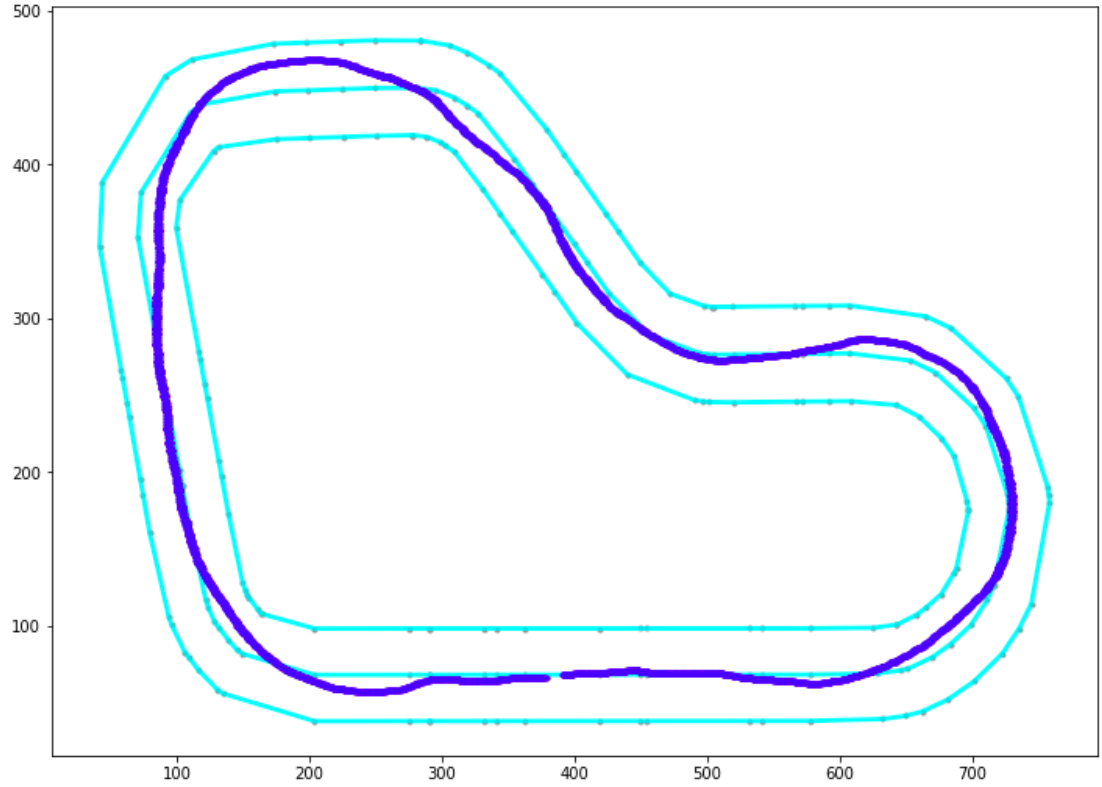
In the image below you can see the rewards returned in the training process.



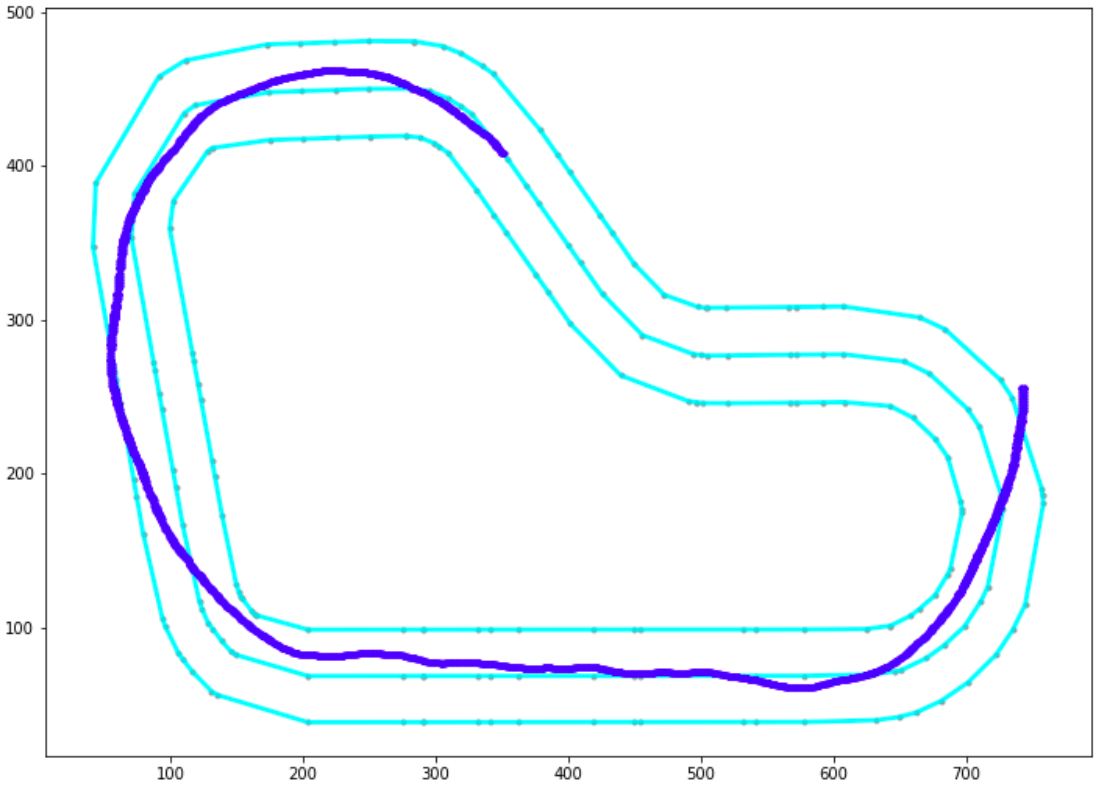

Unlike the first reward function, you can see that the cart is exploring the track more. This allows you to check for more "travel errors" than in the first version.

#### Path taken for top reward iterations

In this first image, you can see that the car maintained a certain proximity to the center of the track, but was not as rigorous as the first model.




I would like to demonstrate that the second highest score iteration was less rigorous than the first in keeping the cart in the center of the track.




**AWS assessment result:**

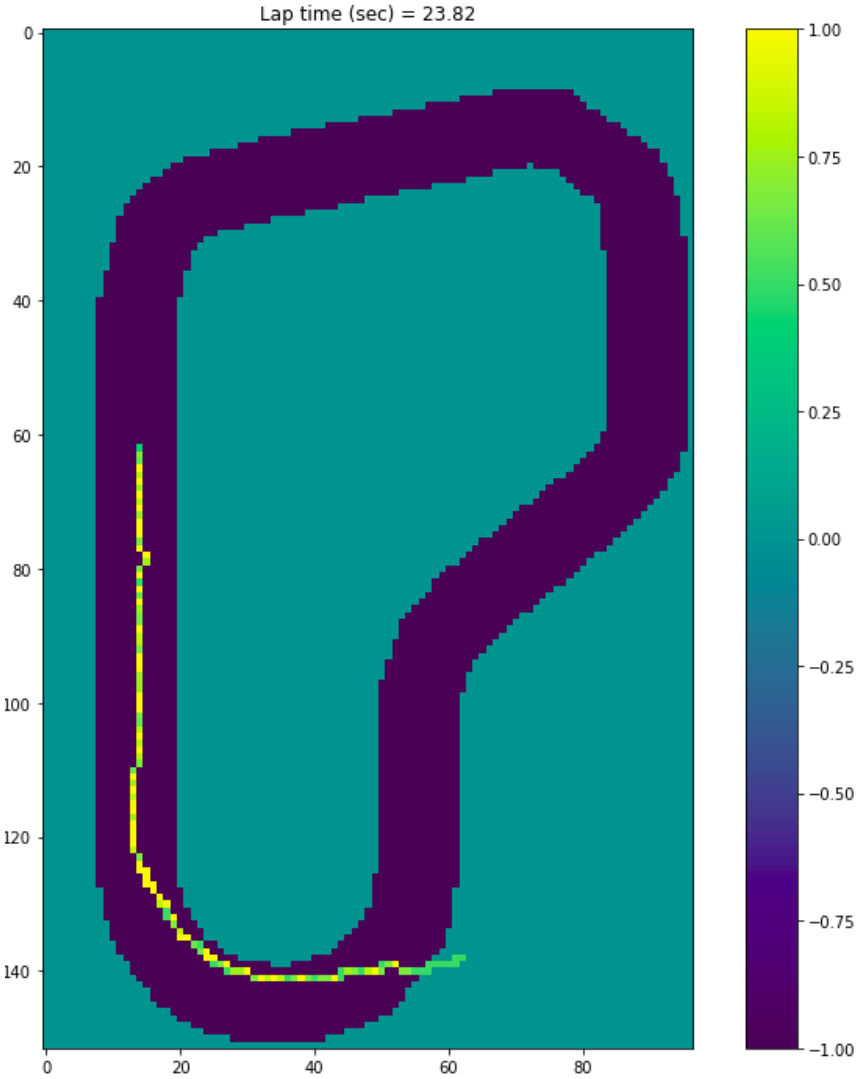
In assessing the speed-based model, 1 hour of training is not enough to complete 100% of the track.

## Evaluation results

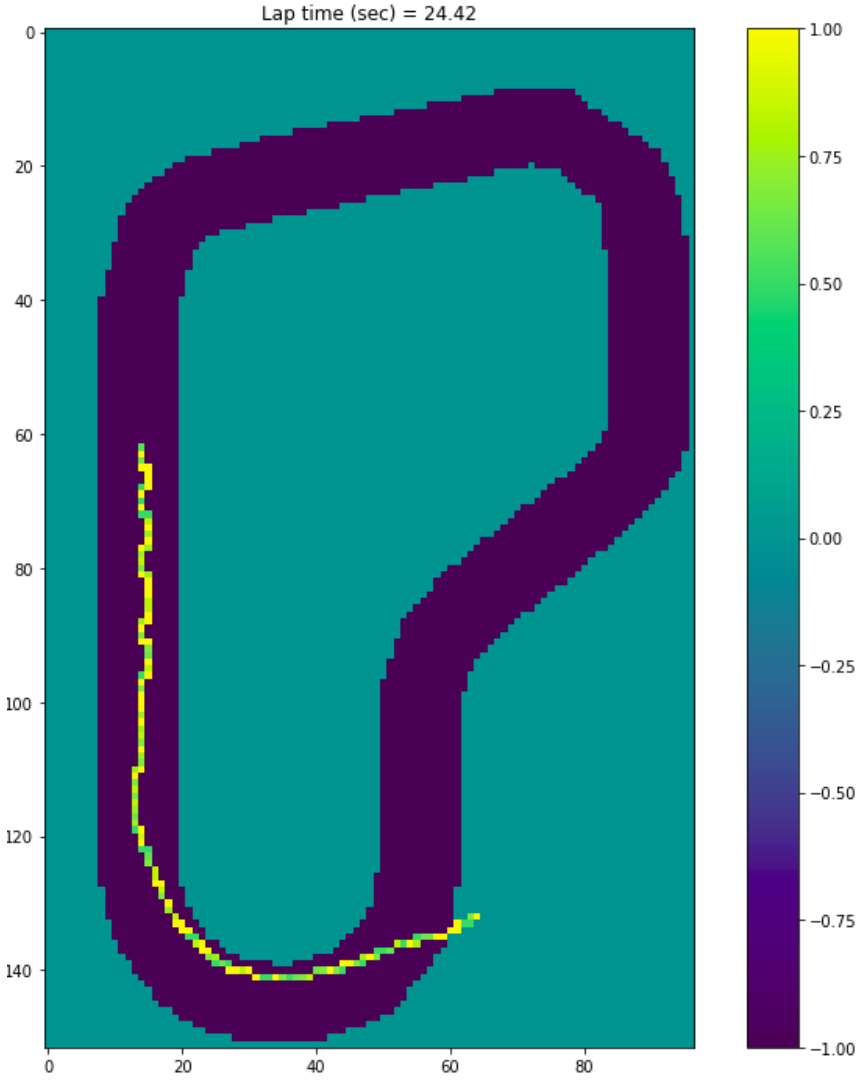
| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:00:24.023 | 34% |
| 2 | 00:00:24.628 | 35% |
| 3 | 00:00:23.358 | 33% |

It is possible to see in the attempts the constant use of the acceleration, causing the exit of the track.

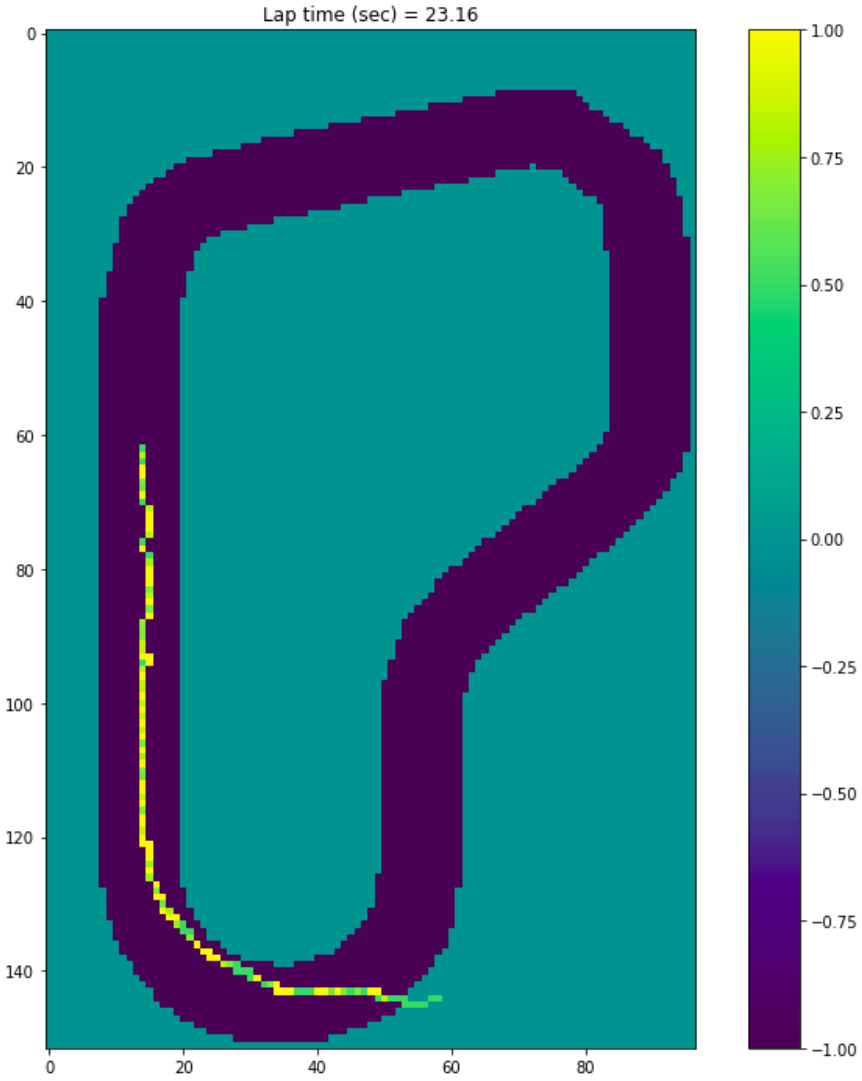
**Attempt 1:**




**Attempt 2:**


Lap time (sec) = 24.42
0
20
40
60
80
100
120
140
0
20
40
60
80
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

**Attempt 3:**




## Refinement

Since in the first model (standard v1) I already reached the goal of completing 100% of the track, I will try to optimize the completion time back. For this I will make some hyperparameter adjustments to speed up the training, remembering that the personal goal is not to exceed the cost of $ 50.00.

To do this, I will clone version 1 of the training model (which is evaluated on the `09 - DeepRacer Log Analysis - Final 60 min.ipynb` notebook), which would be the speed-focused reward function model. in the first hour of training and I will add another 30 minutes of training with the following adjustments:

- **Learning rate:** 0.0009
- **Gradient descent batch size:** 32

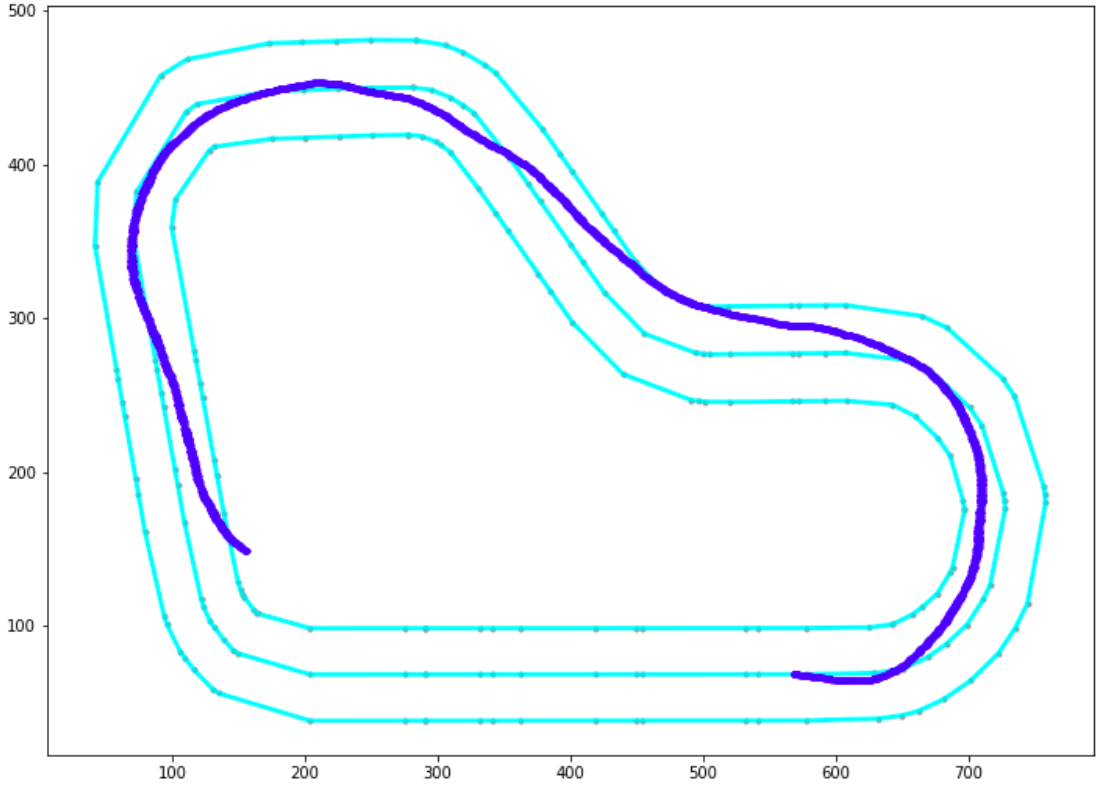
The idea is that with more frequent update streams and larger jumps, algorithm adjustments will be accelerated.

With these adjustments I achieved the following AWS evaluation results:

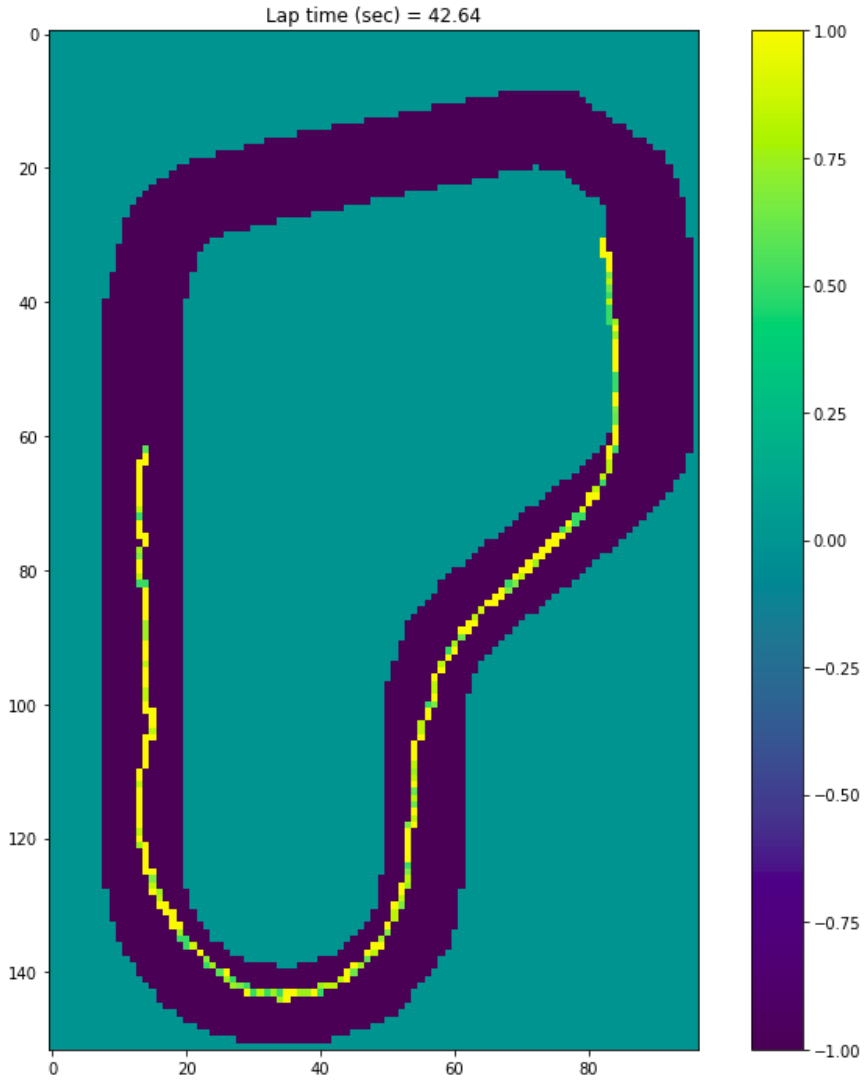
Evaluation results

| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:00:24.023 | 34% |
| 2 | 00:00:24.628 | 35% |
| 3 | 00:00:23.358 | 33% |

We have a breakthrough in terms of track progress percentage, but not lap time. It's worth noting that 30 minutes is a low time for a more concrete assessment, but I want to highlight the following cart iteration:




You can see that in the signaled curve it did not prioritize the center of the track but the speed. You can also see below that the same, during the evaluation left the track exactly at this point:




As for the iteration reward charts, no major changes were observed due to the reward function remaining the same.

To recap, we have so far a model with a speed-based reward function that **has been trained for 1.5 hours**. With that, I will add **another 60-minute training**, but adjusting some reward parameters.

- **Learning rate:** 0.0003
- **Discount factor:** 0.888
- **Gradient descent batch size:** 64

The idea is to move back to more conservative parameters, but with a slight adjustment in the discount factor, as I believe that decreasing the number of future steps to be considered for reward prioritizes momentary actions.

Also, by slightly assessing the reward function, the progress variable may have offset the speed-based reward.

This may have been due to the following code snippet:

```
reward = reward + (reward * (progress / 100))
```

As the reward is based on speed / progress the following cases may occur:

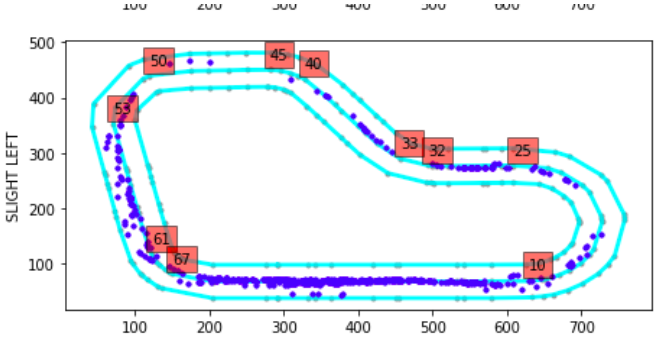
With speed = 1 and progress = 25%, the reward would be: **1.25** = 1 + (1 * (25/100))

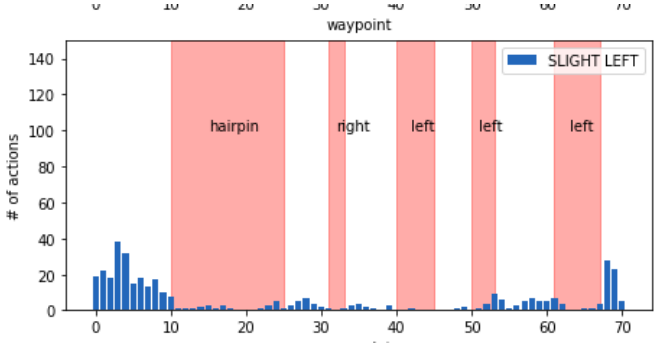
With speed = 0.8 and progress = 75%, the reward would be: **1.4** = 0.8 + (0.8 * (75/100))

With this, it is possible that as the progress increases, the model prefers to slow down to ensure a greater reward.

To resolve this situation, I decided to remove this progress-related factor. Another change was the removal of the penalty based on the `steering_angle` variable for two reasons:

- At times the trolley would slow down in straight stretches of track just so that it could align with the set angle (image below).
- Since initial model training was initially performed with this parameter, the initial zigzag problems were decreased.




Finally, they get the following reward function:

```
def reward_function(params):
    # Read input parameters
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']
    all_wheels_on_track = params['all_wheels_on_track']
    steering = abs(params['steering_angle'])  # Only need the absolute steering angle
    progress = params['progress']
    speed = params['speed']
    # Steering penality threshold, change the number based on your action space setting
    ABS_STEERING_THRESHOLD = 20

    if not all_wheels_on_track:
        # Penalize if the car goes off track
        reward = 1e-3
    else:
        reward = speed

    # Calculate 3 markers that are at varying distances away from the center line
    marker_1 = 0.4 * track_width
    marker_2 = 0.45 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward *= 1
    elif distance_from_center <= marker_2:
        reward *= 0.9
    elif distance_from_center <= marker_3:
```

```
        reward *= 0.85
    else:
        reward = 1e-3  # likely crashed/ close to off track


    return float(reward)
```

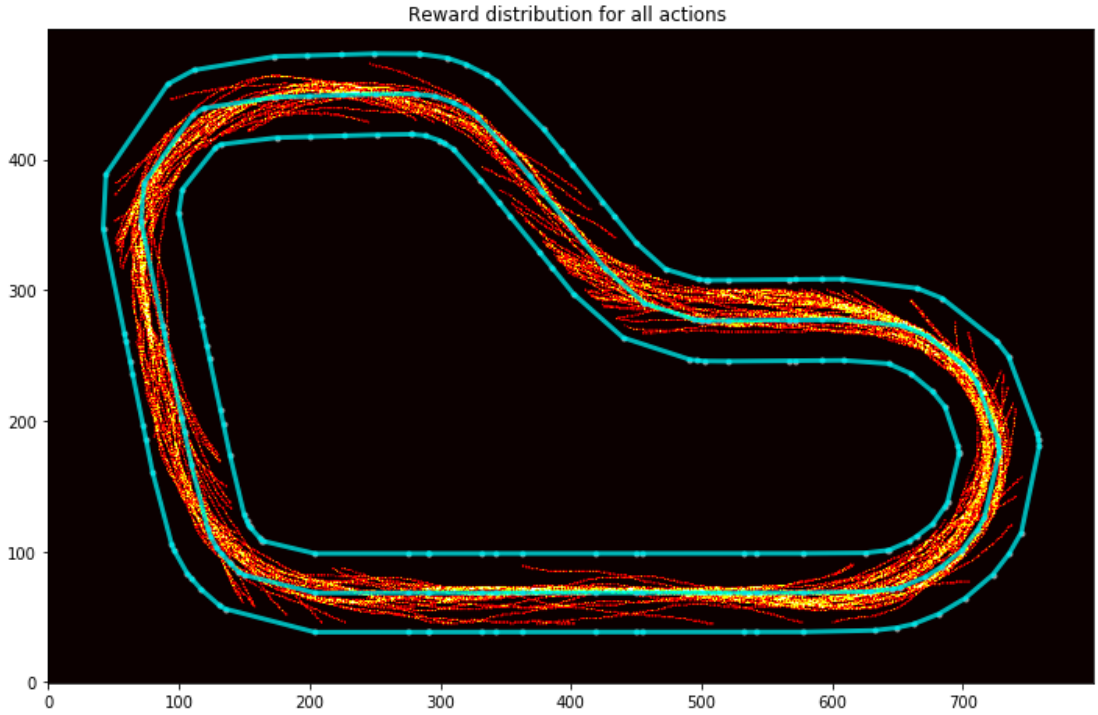
Also, the notebook for this review is located in log-analysis / folder 13 - DeepRacer Log Analysis - Final 60 min - v2-v6.ipynb`.

# IV. Results

## Evaluation and Validation Model

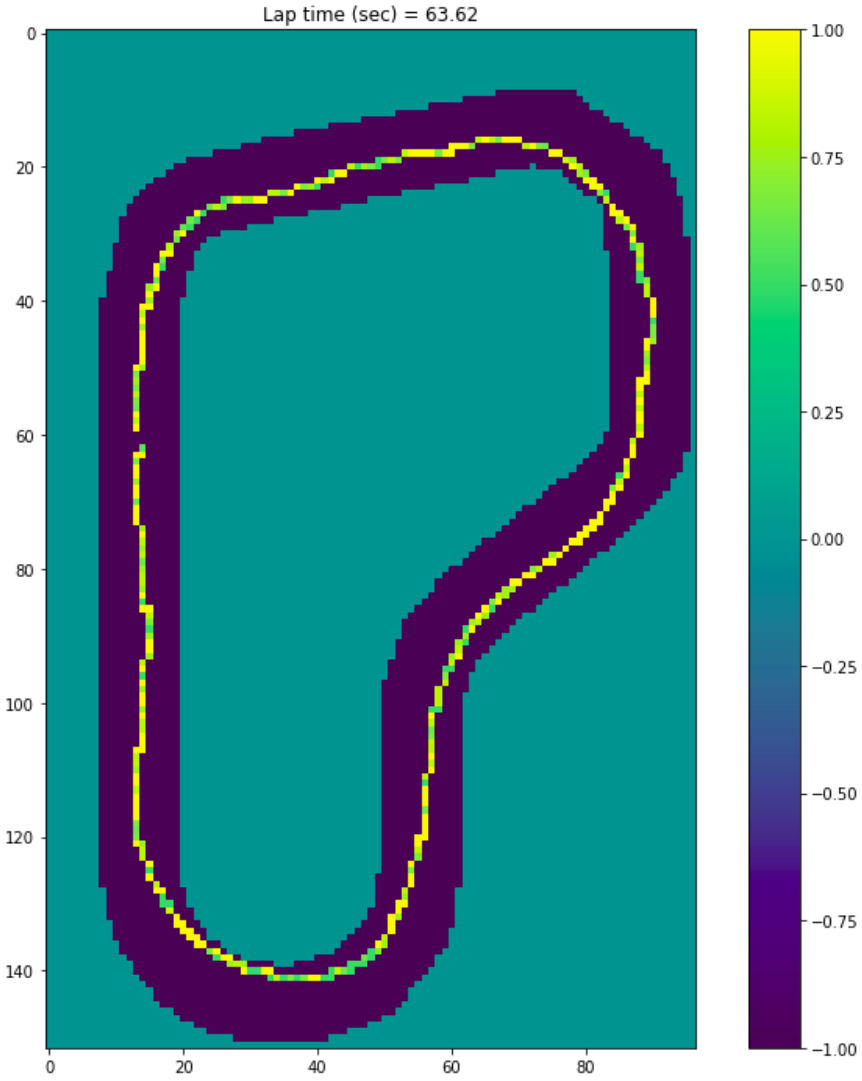
**AWS assessment result:**

Before we evaluate the result of progress x time back, let's just look at how the training distributed the rewards across iterations.

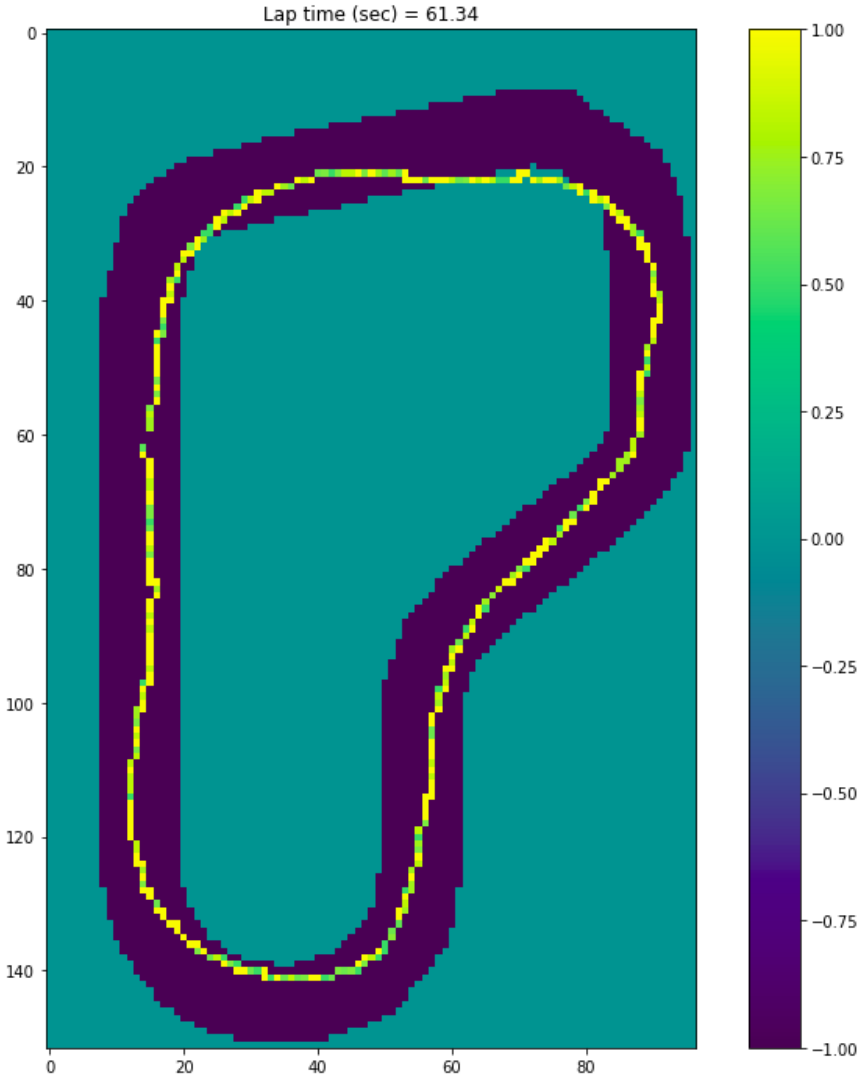



Now let's look at how the model performed in the 3 AWS trial attempts:

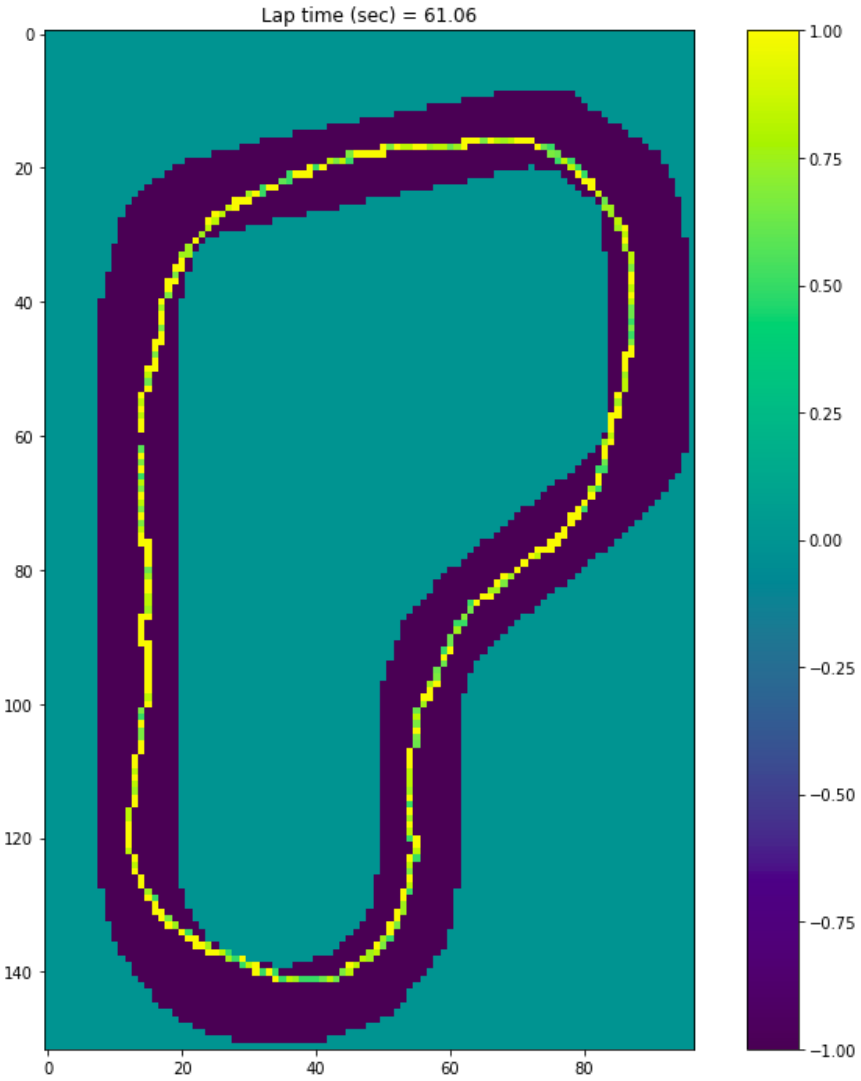
**Attempt 1:**


Lap time (sec) = 63.62
0
20
40
60
80
100
120
140
0
20
40
60
80
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

**Attempt 2:**


Lap time (sec) = 61.34
0
20
40
60
80
100
120
140
0
20
40
60
80
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

**Attempt 3:**




Now let's buy this model with the initial model focused on staying in the center of the track:

**Standard Model (Focused on Track Centering):**

## Evaluation results

| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:01:10.679 | 100% |
| 2 | 00:01:08.608 | 100% |
| 3 | 00:01:16.482 | 100% |

**Current Model (Speed Focused):**

Evaluation results

| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:01:03.815 | 100% |
| 2 | 00:01:01.537 | 100% |
| 3 | 00:01:01.265 | 100% |

From the standard model to the developed model we found a 7.071 seconds difference, with a total of 2.5 hours of training.

## Justification

In some ways, the developed models (Track Center Focus and Speed Focus) showed better results than the benchmark model in up to 3 hours of training.

** Benchmark model: **

You can see from the image below that the reward function and benchmark hyperparameters did not complete the lap during the three attempts to evaluate the trained model during 3H.

Evaluation results

| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:00:50.324 | 78% |
| 2 | 00:00:29.963 | 49% |
| 3 | 00:00:10.612 | 17% |

As the benchmark model does not make clear the time in which it was trained, it is not possible to more accurately assess the outcome.

**Standard model (Focused on track centrality):**

Evaluation results

| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:01:10.679 | 100% |
| 2 | 00:01:08.608 | 100% |
| 3 | 00:01:16.482 | 100% |

**Current Model (Speed Focused):**

Evaluation results

| Trial | Time | Trial results (% track completed) |
|---|---|---|
| 1 | 00:01:03.815 | 100% |
| 2 | 00:01:01.537 | 100% |
| 3 | 00:01:01.265 | 100% |

Regarding the question "* Is the final solution significant enough to have solved the problem? *": In assessing the proposed challenge, completing 100% of the track by spending less than $ 50 I believe yes, it is significant enough to complete the challenge by providing continuity of work, inclusive.

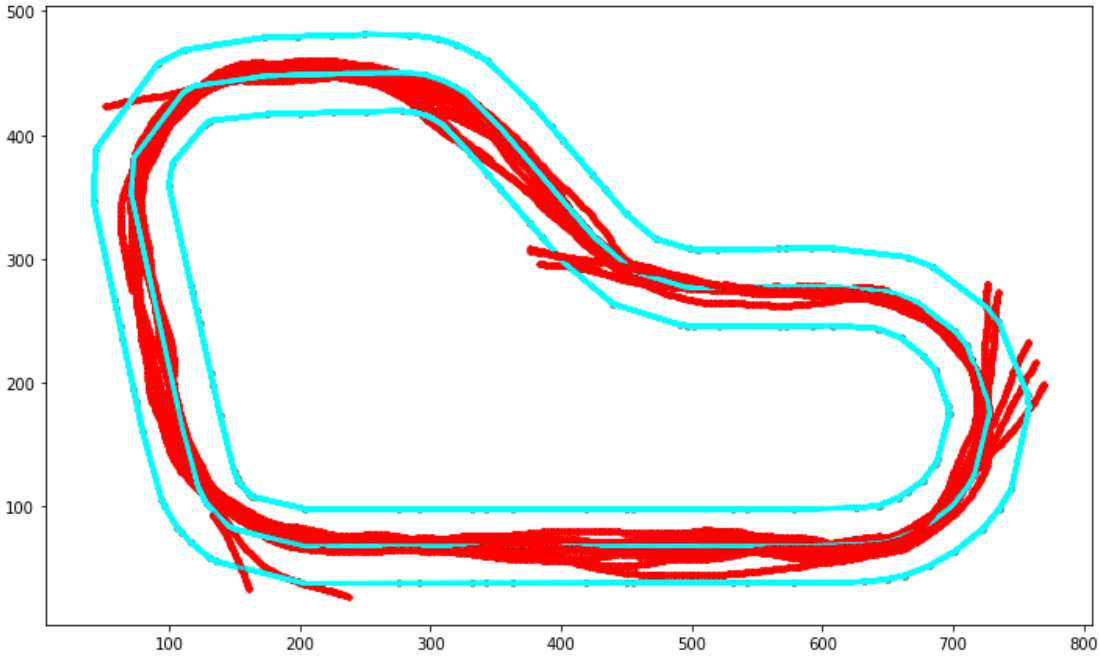
# V. Conclusion

## Preview Free Foma

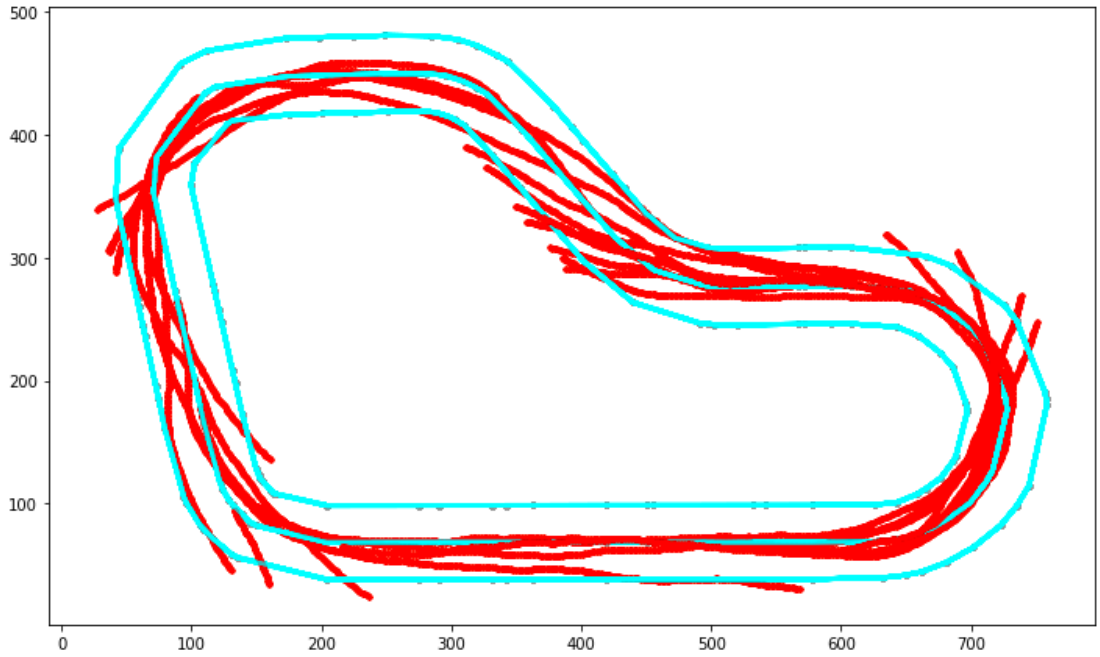
The main quality that could be listed in my project is the knowledge built on the functioning of the defined reward functions and hyperparameters used.

The most representative visualizations of the differences between the initial reward function, focused on distance from the center of the track, and the final, focused on speed, is as follows:

**Initial Standard Model**




**Final Model**




It is visible how the initial standard model did not exploit the full length of the track, sometimes reducing the speed of the cart for this. Already in the second image, it is possible to verify how the cart sometimes "spreads the curve" to compensate the speed. This is all due to the basic definition of the reward function.

Rewards functions use a limited number of parameters. This makes the possibility of parameterization also limited, but leaves most of the learning to the RL algorithm.

The reward function developer will always have to have a balance between defining an extremely complex and comprehensive reward function so that the RL algorithm has the flexibility to find other ways to "solve your problem".

## Reflection

RL tasks are very exploratory and comprehensive. To solve this chosen problem, I segmented my work as follows:

- Experimentation with AWS Deep Racer platform operation.
- Definition of starting milestones as a standard reward function and benchmark model.
- Evaluation of data generated from the first round of training/evaluation.
- Study on the impacts of reward function parameters.
- Implementation of the new reward function and adjustment of hyperparameters.
- Evaluation of the data generated from the new training/evaluation round.
- Study on possible improvements and negative impact points on model performance and reward function.
- New round of training/evaluation.
- Composition and evaluation of the results obtained.

As Reinforcement Learning is a new area for me, the complexity of scenarios (actions, states, rewards, etc.) and the high-cost risk on the AWS platform are the biggest challenges of my project.

As for the final solution, it was below the initial result I expected (the 11 seconds benchmark model). However, the evaluation process was so productive and very explanatory that it made up for a lot of the difference between the return time I expected. Not to mention that even the benchmark model with training less than 3H did not perform as well, sometimes not completing the track in the evaluations.

## Improvements

As for improvements, there are many possibilities. The main ones I would list would be as follows:

- Study of the `heading` parameter for use in the reward function as an aid in the smoother centering of the cart on the track.
- Longer training time (hours) of the final model generated to evaluate this process with more iterations, to verify if the model would be over adjusted.
- Training the model in other tracks, so that it adapts to the most varied possibilities of tracks.

---

## Referências:

[1] AWS DeepRacer - the fastest way to get rolling with machine learning. Retrieved from https://aws.amazon.com/deepracer/?nc1=h_ls

[2] AWS DeepRacer Scholarship with Udacity. Retrieved from https://www.udacity.com/aws-deepracer-scholarship

[3 ] Parâmetros de entrada da função de recompensa do AWS DeepRacer - AWS DeepRacer. Retrieved from https://docs.aws.amazon.com/pt_br/deepracer/latest/developerguide/deepracer-reward-function-input.html

[4] Treinar e avaliar modelos do AWS DeepRacer usando o console do AWS DeepRacer - AWS DeepRacer. Retrieved from https://docs.aws.amazon.com/pt_br/deepracer/latest/developerguide/deepracer-console-train-evaluate-models.html#deepracer-iteratively-adjust-hyperparameters

[5] Using Jupyter Notebook for analysing DeepRacer's logs - Code Like A Mother. Retrieved from https://codelikeamother.uk/using-jupyter-notebook-for-analysing-deepracer-s-logs

[6] aws-samples/aws-deepracer-workshops. Retrieved from https://github.com/aws-samples/aws-deepracer-workshops

[7] How to win at DeepRacer League? (code and model included) | AWS DeepRacer Championship Cup | re:Invent 2019. Retrieved from https://medium.com/vaibhav-malpanis-blog/how-to-win-at-deepracer-league-code-and-model-included-27742b868794