PCGRL to Generate Racing Tracks in CarRacing-v0

Satyen Singh
New York University
New York, USA
srs833@nyu.edu
Taran Iyengar
New York University
New York, USA
ti2059@nyu.edu
Haike Xie
New York University
New York, USA
hx2084@nyu.edu

***Abstract*—The scope of this paper intends to explore the Procedural Content Generation via Reinforcement Learning (PCGRL) method for CarRacing-v0. We picked the CarRacing platform as we wanted to work with a racing environment as this would bring a novelty to the PCGRL approach. CarRacing-v0 was perfect for this purpose as it was a 2D racing environment readily available in Open AI Gym as well as the use of Stable Baselines. The use of RL for this task was notably different as compared to the original PCGRL paper due to the difference in generating tracks as compared to tiles. The set of control points would define a track that utilizes Proximal Policy Optimization (PPO) to train a Racing Agent to generate a new set of control points for track generation. These control points also act as checkpoints for the generated track. Furthermore, a reward function for this agent is difficult to formulate as the complexity of a track is not easily defined. A combination of the reward (r_{cr}) of the Racing Agent to play the generated track as well as an Entropy based reward function (r_e) from the checkpoints was used to define the PCGRL reward (r_{pcg}). This resulted in a PCGRL agent can maximally optimize the difficulty of a randomly generated set of checkpoints.**

***Index terms*—Reinforcement Learning, Procedural Content Generation, Proximal Policy Optimization, Open AI Gym**[1]

I. Introduction

With an increase in game complexity and demand for content, PCG is used for making new content and is often done through methods such as L-systems, Evolutionary Algorithms (EA) (more so in the context of racing games), General Adversarial Networks (GAN), cellular automata, etc [1]. This paper intends to take a different approach by exploring PCGRL. A common method for PCG in games is Space Based PCG (SBPCG) which is a type of generate-and-test PCG with two properties based on EA. A fitness function whic is to be maximised by testing the content [2] created.

PCGRL offers several advantages over SBPCG. PCGRL has a long training period but can generate new levels faster as search is not required [3]. It also eliminates the need for a large data set to train on [3]. Last, PCG is collaborative-friendly and allows for a smooth integration of content generated through both human input as well as machine input [3].

[1]https://github.com/ssingh1997/PCGRL_CarRacing_v0

As aforementioned, the idea was to test the PCGRL methodology in racing games. Initially, games such as TORCS were considered, but the 3D nature of such games would make this infeasible for the duration of this task. Therefore, we decided to use CarRacing-v0 as it is available easily in the Open AI Gym Library. In addition, its relative simplicity makes it easy to learn from pixels, which will reduce computational power required to implement PCG. The use of Open AI Gym for this task allows us to define a CarRacing-v0 environment for generation of checkpoints. These checkpoints will be fused together to create the new track.

There are a number of possible ways to generate a curve from some control points. B-spline is a popular way to generate a smooth curve line using control points. However, B-splines make it challenging to generate straight lines which are necessary and common on a car racing track. Instead, we use a more intuitive way to gradually generate the track from one control point to the next control point as described in the Track Generation section.

Gym environments allow for implementation of Stable Baselines which allow for RL models to be trained in the environments defined. This results in the use of the same environment for testing the trained agent for racing as well as generating tracks. For both purposes, a PPO [4] based Actor-Critic (A2C) policy was chosen to run with a Convolution and Dense Neural Network, respectively.

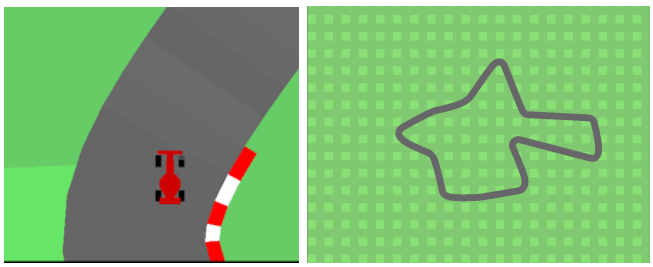

Fig. 1. **(Left)** CarRacing-v0 Example and **(Right)** Track

II. Related Work

There has been plenty of research done on the procedural generation of custom tracks in racing games through the use of evolutionary algorithms and optimization of fitness functions. In [5], both online and offline methods were evaluated [2]. The tracks were represented as b-splines which are fixed-length parameter vectors [2]. Once a fitness function is determined using the driving performance of the car and is optimized for a track, it is tested using a trained racing agent [2].

In [6], an online tool that utilized the SBPCG framework for generation of custom tracks in 3D racing games such as TORCS and Speed Dreams is used. This tool has two components. The first is an interactive interface that maintains the database and can interact with users to collect feedback as well as provide a way for them to access the tracks. The second is an evolutionary algorithm back end that rendered the game tracks available for users in the interface.

An interesting implementation of RL agents is presented in [7], in which the Prioritized Level Replay (PLR) can generate levels in games. Its abilities were demonstrated on CarRacing v0, in which PLR, ends up selecting the most complex randomly designed tracks. These tracks most closely resemble human made tracks. The tracks were generated randomly using Bezier curves, similar to [2].

The PCGRL framework is tested on Binary, Zelda, and Sokoban [3]. The PCG task was cast as a Markov Decision Process (MDP) which was easily transferable to the Open AI Gym Platform. The results were promising as the framework allowed for the generation of many playable levels for Zelda and Sokoban [3].

III. Methodology

Unlike other PCG frameworks, PCGRL splits up the process of performing PCG into iterative tasks. Since PCG is cast as a MDP, it means that the RL agent will get rewarded based on feedback and will base the next action on the reward generated from the previous action [3]. In the original PCGRL paper, a set of random tiles will be generated first. At each iteration of the process, a reward signal is generated based on small changes on a random tile. Based on these rewards, the agent will determine how close it is to the goal state [3]. For simplicity, PCGRL is broken up into three components. The first is the problem module which contains metadata about the level created by PCG [3]. The second is the representation module which converts the custom level into an observable state for the agent to act on [3]. Last, the change percentage module sets a limit on how many possible changes can affect the generated content [3].

The aim is to utilize this framework and its methods in the context of CarRacing-v0. While the essence of the original PCGRL framework and methodology remains the same, there were a few changes that had to be made to suit the purposes of this problem. In this case, the reward signal is generated from the combination of the racing agent's ability to race on the track as well as the entropy heuristic [8] which analyzed the track for its features such as curvature. Change percentage is also to implemented as part of the future work on this given adequate time.

The workflow for PCGRL in CarRacing-v0 entails the steps below. First, an initial track is generated from a random set of checkpoints. This is repeated if the track cannot be created by the track generation algorithm defined below from the set of checkpoints. This track is used to obtain the Entropy based reward (r_e) as well as the reward (r_{cr}) from the Racing Agent. These are the initial checkpoints and rewards for the iteration. The set of checkpoints are used as input to the PCGRL agent. The PCGRL output is a list of values which update the checkpoints and thus, obtaining a new generated track. Using the reward function defined below, the reward for PCGRL, (r_{pcgrl}) is obtained. This is repeated for the duration of training.

The iterative process of calibrating the Car Racing Environment to selecting the player agent was one that required a lot of testing before PCGRL could even be attempted. Once the tracks were finally able to generated, the reward function could be decided and calibrated. The idea of using both the Entropy and PPO allowed the PCGRL agent to take the best of both worlds and generate tracks using the original set of checkpoints.

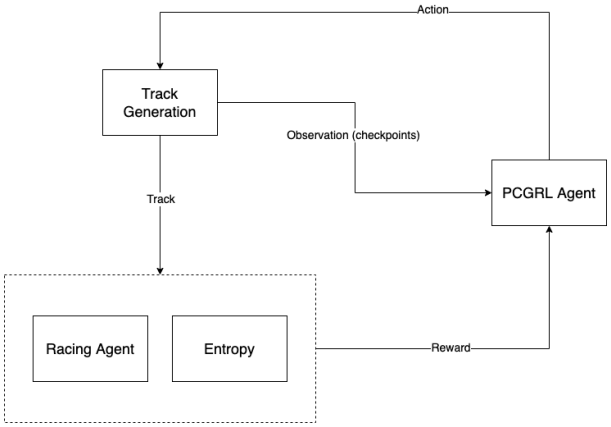

Fig. 2. Modified PCGRL Environment for CarRacing-v0

A. Observation Space

While PCGRL implements action as a change to a random tile, we observed that the large tile size of around 300 as well as the track generation mechanism using checkpoints meant tile based actions were not possible. The resultant PCGRL framework was not usable in this continuous environment. The CarRacing-v0 environment was redefined to work with both pixel observations for the Racing Agent and observations consisting of the checkpoints of the track. The checkpoints generation was defined to act as control points to generate tracks. The reward for the Racing Agent was +N/1000 where "N" is the number of tiles covered over the track and for the PCGRL as the reward defined inside a range of [0, 1]. Unlike the PCGRL for Sokoban for example, instead of selecting a random tile, the action would be a δ by which to update

both the radian and radius of each coordinate. Initially the model would choose only a random checkpoint to update, but early training resulted in large negative rewards due to tracks not being generated with the updated checkpoint. The newer action space allowed for smaller movement of other checkpoints to accommodate track generation between checkpoints. Furthermore, this allowed us to not explicitly define a penalty to constrain the model from clustering checkpoints since the resultant track would not be generated in most cases.

B. Track Generation

As mentioned previously, B-splines was not chosen for track generation. The reasoning for it was that it made it difficult to create straight lines which we would like to see on the car racing track. This is due to B-splines approximating a polynomial function between checkpoints and since initial checkpoints are randomly generated, straight lines would not commonly occur. Therefore a different approach that handled the look of the track as well as its ability to work with the PCGRL agent was utilized.

In the CarRacing-v0 environment, the track is represented as a set of continuous points $\vec{t} = \{t_0, t_1, t_2, ..\}$, while each point t_i has three parameters$(\theta_i, r_i, \alpha_i)$. The parameters θ_i and r_i identify the position of the track points t_i as polar coordinates while α_i defines the tangential direction of this point t_i towards the next point t_{i+1}. Because track representations contain around 300 points, RL models trying to learn policies with too many points produces a large search space which led to tracks not being generated frequently due to the large dimension of the observation as well as the action space. A higher-level representation for the track is required when it is applied to our PCGRL model.

A set of control points called checkpoints is defined to guide the process of track generation. These checkpoints are self-contained as the latent representation of the generated tracks. A fixed set of checkpoints would correspond to only a single track.

During this process, the track was encoded as a sequence of checkpoints $\vec{c} = \{c_0, c_1, c_2, ..\}$, where c_i contains two parameters(θ_i, r_i) which identify the position of the checkpoints c_i in the polar coordinate system (r_i is the distance from the origin or radial, θ_i is the angular coordinate or radian).

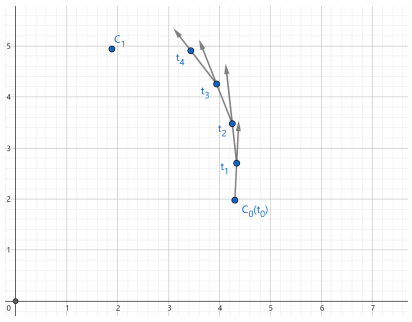

Fig. 3. Track generation between two checkpoints

Using these checkpoints, the tracks are generated. This process will take as input the checkpoints $\vec{c}$ and return a list of new checkpoints $\vec{t}$. To begin with, we start from the first checkpoint c_0 where its position is the position of the first track point t_0 and the initial direction is fixed. We generate the second track point t_1 along the direction of the first track point t_0. After that, we will set the direction of t_1 closer to the direction towards the destination c_1. Fig. 3 shows the procedure.

While the existing PCGRL framework deals with tasks pertaining to tile based environments such as Sokoban or Zelda, problems without discrete representation for both the observation space and as well as the action space led to having to redefine the representation of the CarRacing-v0 environment with respect to the continuous representation of both state spaces.

C. Proximal Policy Optimization

Both the PCGRL model as well as the Racing model are defined by A2C Policies using PPO. The PCGRL model utilizes a Dense Neural Network while the Racing model utilizes a Convolution Neural Network.

PPO is a policy gradient method which trains a stochastic policy in an on-policy way while utilizing actor critic methods. The actor maps the observation to an action and the critic gives an expectation of the reward for the observation while it collects a set of trajectories for each epoch by sampling from the latest version of the stochastic policy. The reward and the advantage estimates are computed in order to update the policy and fit the value function. The policy is updated by gradient ascent while the value function is updated by gradient descent.

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, ...$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

  by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

end for

Fig. 4. Proximal Policy Optimization

PPO uses an improved objective function called Clipped Surrogate Objective as seen in Fig. 4., to limit policy updates within a factor ϵ. This allows our policy to not learn large changes in the case that there are large differences from the previous policy that exceed the clipped value of the expected advantage. The objective function uses an expectation of the ratio of the previous policy to the new policy with the expected advantage function utilizing a discounted sum of rewards. This has led to to having several agents run episodes on the same policy which is then updated using the previous policy. To avoid sampling over very old policies, we update our second policy to the new policy copy every few iterations.

Policy Gradient methods have convergence problems as natural policy gradients involve a second-order derivative matrix which are computationally expensive. Instead of imposing a hard constraint, PPO formalizes the constraint as a penalty in the objective function and solves the problem using first-order derivative methods like Gradient Descent.

While the initial Racing Agent chosen was a PID controller, the agent faced a lot more issues when utilized within the context of PCGRL. PID [9] is used to obtain a difference between the position of the car and the expected position of the car in a few time steps from the current which defined as "error". The current frame error combined with the previous error is used as the input to a heuristic function to determine the action to be taken [Left/Right Turn, Accelerate, Brake]. It was observed that this agent frequently left the track at high speeds in sharp turns such as hairpins or chicane (as seen in Fig. 5). Having a reward function computed based on this would lead to uncertainty while updating the PCGRL reward (r_{pcgrl}) due to not knowing if the low reward (r_{cr}) was caused by the PID agent or the PCGRL agent. To solve this problem, a PPO based Racing Agent was trained which utilized a smaller Convolution Neural Network with 2 layers which used 4 stacked gray scaled frames as observations and thus, allowed for a faster running time for the Racing Agent during PCGRL.

D. Reward Approximation

Difficulties exist in defining a PCGRL reward function in track based environments. The use of an heuristic using parameters of the track also had its challenges. This is not only due to the tracks being limited in representation by a few metrics such as hairpins or chicanes along with the number of turns and straights, but large changes in these values do correspond to similar difficultly of tracks. The total complexity of tracks with large straights into sharp hair pins as in as well as tracks with few straights but frequent chicanes are very comparable. Therefore, we used a combination of the entropy of the curve of the track and racing agent reward (r_{cr}) to generate a reward (r_{pcgrl}) for our track.

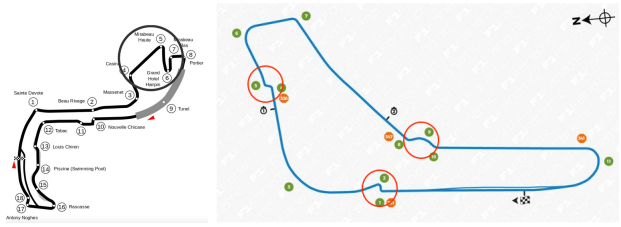

Fig. 5. **(Left)** The hairpin (circled in black) is a bend in the road that is at a very acute angle while the chicane **(Right)** is two tight alternative corners in alternate requiring intense braking (circled in red)

1) Racing Agent Reward: The reward function (r_{pcgrl}) is partly defined to utilize the reward (r_{cr}) based on the Racing Agent for playing the track. By using this measure over a large sample of the existing CarRacing-v0 environment tracks, the ideal value of the expected Racing agent reward (r_{cr}) for the generated tracks was learnt. Subsequently, this was defined as a Random Variable (RV) for a Skewed Gaussian Distribution function (as shown in Fig. 6 Left) with a mean corresponding to the expected Racing agent reward (r_{cr}) and a variance that is proportional largely to the range within which the reward (r_{cr}) is returned for most of the existing CarRacing-v0 tracks. The need for a Skewed Gaussian function exists due to the mean episode reward (r_{cr}) being around 800. This meant that with a standard Mean Square Error (MSE) based reward function or a normal Gaussian function, the symmetry would reduce the PCGRL reward (r_{pcgrl}) to be returned equally in the expected reward range i.e. [600, 1000]. But since this is inaccurate to rewards (r_{cr}) below 600, a Skewed Gaussian function allows for a larger variance in the lower half of the distribution, matching our expected reward (r_{cr}) range [400, 1000] better. This also can be intuitively understood as allowing larger faults in lower rewards (r_{cr}) which means the generated tracks are slightly more difficult for a high performing agent while allowing smaller faults in larger rewards (r_{cr}) which would also be easily achievable for low performing agents.

2) Entropy Reward: While we look at the relative reward (r_{cr}) of a Racing agent with respect to existing CarRacing-v0 tracks, we also impose penalties as a soft constraint in the reward (r_{pcgrl}) of the PCGRL agent. We do so to avoid a few scenarios which lead to large negative updates. An external penalty of -1 is given if the policy output leads to tracks which cannot be generated. This can be attributed to the track generation being unable to connect checkpoints, or checkpoints being very close, leading to overlapping tracks. This approach led to a local minima with latter policy updates for each iteration being negative rewards. Furthermore, penalizing the magnitude of change of checkpoints at each time step proved difficult. To combat these problems, a partial additional reward term (r_e) was introduced. For all points on the track, the curve of 20 track points with a sliding window of 15 points was calculated into 16 bins (as shown in Fig. 6 Right). The entropy of these curves was approximated to a Gaussian with a mean of 3 after sampling for original CarRacing-v0 tracks similar to the player agent reward (r_{cr}) above [8].

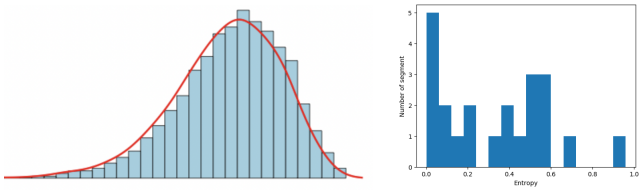

Fig. 6. **(Left)** Skew Gaussian Distribution for Reward and **(Right)** Track Entropy (16 Bins)

A combination of both these reward functions, as weighed over 0.33 for the Racing Agent (r_{cr}) and 0.67 for Entropy (r_e), was used as input for the PCGRL agent (r_{pcgrl}). The value of the weights were tested over different ranges and these are the ones which worked best. This configuration was decided after several test runs to determine which level of both

rewards that were desired. Having just the PPO led to the track giving negative rewards after a certain point. Having only the Entropy reward led to large fluctuations in the track which was not ideal. Giving a bit more preference to the PPO allowed for the model to emulate a real time player as much as possible but the Entropy also allowed to extract key information about the track and make significant changes to the track that would otherwise not be possible by just using PPO.

Further more, since rewards for the Racing Agent (r_{cr}) can be modified largely by changes to the total tile count of the track as smaller tracks give lower rewards and larger tracks give bigger rewards, a penalty on generated tracks outside a $\pm 10\%$ range of the initially generated track to prevent improvements to the reward function by shrinking or expanding the track.

IV. Result

The final trained PCGRL model allows to incrementally improve a randomly generated track without losing on the key features of the track such as the turns, hairpins, chicane at every time step and thus, retaining the uniqueness of each track. This does not come at the cost of the complexity of the track but with the opposite, the Racing Agent reward (r_{cr}) over time steps is adjusted to move towards the mean episodic reward of the existing CarRacing-v0 tracks, which is around 800.

As observed from the graphs, the average KL-Divergence is decreasing over the training period and similarly, the average Policy Loss is converging towards 0 as well. We can also observe that the PCGRL model reward (r_{pcgrl}) is gradually increasing towards the maximum value of 1.0 over the course of training.

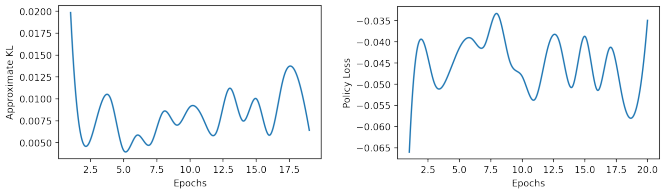

Fig. 7. **(Left)** KL-Divergence and **(Right)** Policy Loss

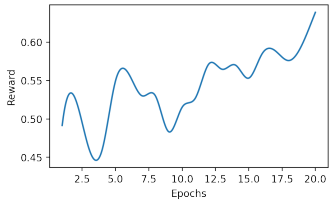

Fig. 8. Rewards over 20 Epochs

As is observed from Figures 9 and 10, the tracks went through a metamorphosis which led them to becoming more complex than their original state. In Figure 9, there are two sharp corners as well as an exaggerated u-turn at the top of

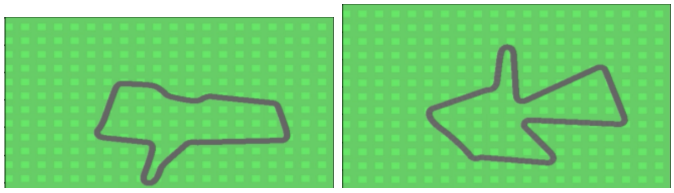

Fig. 9. **(Left)** Original Track **(Right)** New Track

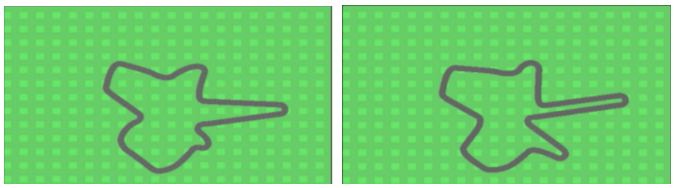

Fig. 10. **(Left)** Original Track **(Right)** New Track

the track. In Figure 10, though the shape of track remains the same, the turns have become more pronounced than the original state. These are the kinds of changes that we are looking for. They allow the player to experience new content while also adding the desired amount of difficulty to them.

V. Conclusion

The aim of this research effort was to explore the effectiveness of PCGRL as a content generation tool as opposed to using evolutionary algorithms. This was a new addition to the existing PCGRL framework as well as we attempted to use RL to procedural generate content in an environment with a continuous observation space. We chose CarRacing-v0 as our platform for its accessibility as well as its compatibility with RL libraries. It is also a 2D representation of a racing environment which allowed to render new tracks in a reasonable time frame. This attempt at PCGRL for CarRacing-v0 proved to be promising and allows us to expand the realm of PCG. It also brought in a new method for racing tracks as it showed that evolutionary algorithms are not the only way to create custom tracks. Given a longer time frame, there is definitely a great scope to further refine on the methods described in this paper. Some of the key features from the original PCGRL framework such as Change Percentage are things that can definitely be incorporated for CarRacing-v0. The goal is to refine these methods to the point that PCGRL can be utilized for sophisticated 3D Car Racing games such as Forza.

References

[1] J. T. M. Kerssemakers, J. Tuxen and G. Yannakakis, "A procedural procedural level generator generator," presented at the IEEE Conference on Computational Intelligence and Games, 2012.

[2] K. S. J. Togelius, G.N. Yannakakis and C. Browne, "Pcgrl: Procedural content generation via reinforcement learning," published in IEEE Transactions on Computational Intelligence and AI in Games, 2011.

[3] S. E. A. Khalifa, P. Bontrager and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," *Arxiv*, 2020.

[4] P. D. A. R. J. Schulman, F. Wolski and O. Klimov, "Proximal policy optimization algorithms," *Arxiv*, 2017.

[5] R. N. J. Togelius and S. Lucas, “Towards automatic personalised content creation for racing games,” presented at the IEEE Symposium on Computational Intelligence and Games, 2007.
[6] L. Cardamone, P.L. Lanzi, and D. Loiacono, “Trackgen: An interactive track generator for torcs and speed-dreams,” *Applied Soft Computing*, 2015.
[7] J. P.-H. J. F. E. G. M. Jiang, M. Dennis and T. Rocktaschel, “Replay-guided adversarial environment design,” presented at the Neural Information Processing Systems, 2021.
[8] P. L. L. Cardamone and D. Loiacono, “Automatic track generation for high-end racing games using evolutionary computation,” published in IEEE Transactions on Computational Intelligence and AI in Games, 2011.
[9] K. Kartha. (2021) Solving openai carracing-v0 using image processing. [Online]. Available: https://medium.com/@kartha.kishan/solving-openai-carracing-v0-using-image-processing-5e1005ee0cb