RL for PySuperTuxKart

Asaad Mohammedsaleh
Computer Science Program
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
asaad.mohammedsaleh@kaust.edu.sa

Abstract

This is a report on the implementation of reinforcement learning approaches for PySuperTuxKart as part of Spring 2025 semester course CS294X: Introduction to Reinforcement Learning. In this report, we will discuss the implementation of Q-learning and DQN for PySuperTuxKart. The code for the implementation can be found in the GitHub repository `https://github.com/Asaad47/RL-Project`.

1 Introduction

In this project, the task is to use reinforcement learning techniques to train an agent to play PySuperTuxKart. In the homework assignment 6, we implemented a simple controller that uses a pre-defined policy to play the game. I used this controller as an initial point to implement a more sophisticated controller using reinforcement learning. Throughout this report, we will discuss the implementation details of expanding on the simple manual controller using Q-learning, and we will also discuss the implementation details using DQN on this task, inspired by the Atari paper (Mnih et al., 2013, 2015).

1.1 PySuperTuxKart Installation

The package `pystk` (PySuperTuxKart) requires x86_64 architecture. To run the game locally on my M2 Macbook Air, I used `Rosetta` to emulate x86_64 architecture of `Ghostty` terminal app and initialized `Conda` environment to handle x86_64 packages. However, whenever I run python scripts with `pystk` package, I get the following warning:

```
    Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced Vector
Extensions (Intel(R) AVX) instructions.
```

In addition, I get the following error the first time I run a `pystk` script:

```
OMP: Error #15: Initializing libiomp5.dylib, but found libiomp5.dylib
    already initialized.
OMP: Hint This means that multiple copies of the OpenMP runtime have been
    linked into the program. That is dangerous, since it can degrade
    performance or cause incorrect results. The best thing to do is to
    ensure that only a single OpenMP runtime is linked into the process,
    e.g. by avoiding static linking of the OpenMP runtime in any library.
    As an unsafe, unsupported, undocumented workaround you can set the
    environment variable KMP_DUPLICATE_LIB_OK=TRUE to allow the program to
    continue to execute, but that may cause crashes or silently produce
    incorrect results. For more information, please see
```

Preprint. Under review.

```
    http://www.intel.com/software/products/support/.
[1]    49696 abort      python DQN.py --track lighthouse --mode test --verbose
```

I avoid this error by running the following command:

```
export KMP_DUPLICATE_LIB_OK=TRUE
```

However, I think this is not the best solution but I did not spend more time on this issue. I think this also creates differences in computations between different machines, and I noticed that when testing runs locally versus on Ibex. I will discuss the results in the experiments section.

1.2 PySuperTuxKart

The game is a 3D racing game where the goal is to navigate a kart through the track while avoiding obstacles. The game is played from a third-person perspective.

The game documentation provides multiple settings for the image quality through the `pystk.GraphicsConfig` class. The available settings are:

- `hd()`: "High-definitaiton graphics settings"
- `ld()`: "Low-definition graphics settings"
- `sd()`: "Standard-definition graphics settings"
- `none()`: "Disable graphics and rendering"

[TODO: add citation of game documentation]

[TODO: add different quality images of the game]

To speed up the training, I used the `ld()` setting during training, and the `hd()` setting during testing for the Q-learning agent. I avoided using the `none()` setting because as the game is not rendered, training shows incorrect results. However, I used the `hd()` setting for the DQN agent because the training needs to be exposed to the high quality images reflecting the actual game.

A race track has a start line and a finish line, and the `kart` object can track the distance from the start line using the `distance_down_track` attribute. In the initial code given for the homework, `kart.overall_distance / track.length` is used to check if the kart has finished the race. However, the `overall_distance` attribute gives negative values when the kart just starts the race, giving incorrect results for `kart.overall_distance / track.length` in the beginning of the race with a value close to -1. `distance_down_track / track.length` gives a correct range from 0 to 1, and is used in the code for tracking the progress of the kart. Checking for finished race is not changed and is still using `overall_distance / track.length`.

2 Q-learning

Under this section, we will discuss the implementation of two Q-learning agents:

- `simple_RL_controller`: A simple Q-learning agent that only decides on steering angle out of three discrete actions (forward, left, right).
- `discrete_RL_controller`: An extended version of the simple Q-learning agent that also decides on acceleration, brake, drift, and nitro.

The main differences between the agents are how the state space, action space, and reward function are defined. Once these are defined, the Q-learning algorithm is the same. Whenever a new action is taken, the Q-value is updated using the formula from Lecture 10 notes (Orabona, 2025):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

where s is the current state, s' is the next state, a is the action, r is the reward, γ is the discount factor, and α is the learning rate.

2.1 Simple Q-learning agent

The simple Q-learning agent motivation was to minimally enhance the performance of a manual controller that uses a pre-defined policy to play the game and make sure the kart can finish the race with reinforcement learning.

State space The state space is defined as the x-coordinate of the aim point as defined in the homework assignment 6. The state space then is a single value between 0 and 127.

Action space The action space is defined as the steering angle, which can take 3 discrete values: -1, 0, 1. All other action attributes are fixed as follows: (acceleration, brake, drift, nitro) = (0.8, False, False, False).

Reward function The reward function has three components: `reward_steer`, `reward_rescue`, and `reward_done`. The reward function is the sum of the three components.

$$\texttt{reward\_steer} = \begin{cases} -|1 - \frac{2x}{\texttt{IMG\_WIDTH}}| & \text{if action steer is in the opposite direction of the aim point} \\ 0.01 & \text{otherwise} \end{cases} \tag{2}$$

The motivation for the reward steer component is to encourage the kart to steer towards the aim point. The reward is 0.01 if the steer is aligned with the aim point.

`reward_rescue` is -1 if the kart is rescued, and 0 otherwise.

`reward_done` is 10 if the kart has finished the race, and 0 otherwise.

2.2 Discrete Q-learning agent

The discrete Q-learning agent is an extension of the simple Q-learning agent that also decides on acceleration, brake, drift, and nitro.

State space The state space is composed of the x-coordinate of the aim point as previously defined in the simple Q-learning agent, and the percentage of the distance down the track. The state space then is a pair of two values: (x, d), where x can take values from 0 to 127, and d can take values from 0 to 99.

Action space The action space is composed of the steering angle, acceleration, brake, drift, and nitro. The action space then is a tuple of five values: $(\texttt{steer}, \texttt{accel}, \texttt{brake}, \texttt{drift}, \texttt{nitro})$.

- $\texttt{steer} \in \{-1, 0, 1\}$
- $\texttt{accel} \in \{0.8, 0.5, 0.05\}$
- $\texttt{brake} \in \{False, True\}$
- $\texttt{drift} \in \{False, True\}$
- $\texttt{nitro} \in \{False, True\}$

The total number of actions is $3 \times 3 \times 2 \times 2 \times 2 = 72$, and the size of (state, action) space is $128 \times 100 \times 72 = 92160$.

Reward function The reward function also has three components: `reward_steer`, `reward_rescue`, and `reward_done`. The reward function is the sum of the three components.

`reward_steer` is defined as previously in the simple Q-learning agent, but `reward_rescue` and `reward_done` are defined as follows:

$$\texttt{reward\_rescue} = \begin{cases} -1 & \text{if the kart is rescued} \\ \frac{d}{1000} & \text{otherwise} \end{cases} \tag{3}$$

where d is the percentage of the distance down the track. The motivation for this reward is to encourage the kart to move forward to get higher rewards.

$$\texttt{reward\_done} = \max(0.1, 10 - \frac{t}{100}) \tag{4}$$

where t is the number of steps taken to finish the race. The motivation for this reward is to encourage the kart to finish the race in less steps.

2.3 Implementation details

Note that in both implementations, the Q-values are initialized to 0. During training, the agent follows an ϵ-greedy policy to balance exploration and exploitation, where ϵ is the probability of taking a random action and linearly decays with the number of training steps. In particular, the ϵ is initialized to 1.0 and decays linearly to 0.1 after 1000 training steps and remains constant at 0.1 after that.

It has been observed that the kart can get stuck in a position where it cannot move forward even if the agent is trying to move it. To avoid the issue of updating the Q-values of the stuck state for long periods of time, I added a maximum threshold for the kart being in the same distance down the track. If the kart is in the same distance down the track for more than 60 steps, the kart is considered stuck and the episode is terminated. The choice of 60 steps is kind of arbitrary and can be tuned. This parameter needs to be not so large to avoid the issue of the kart getting stuck for a long time, but at the same time, it should be large enough to avoid cutting the episode too early when the kart can still move.

3 DQN

4 Experiments

5 Conclusion

References

References

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013) Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. https://doi.org/10.1038/nature14236

Orabona, F. (2025) CS294X: Introduction to Reinforcement Learning Lecture Notes. *King Abdullah University of Science and Technology*.

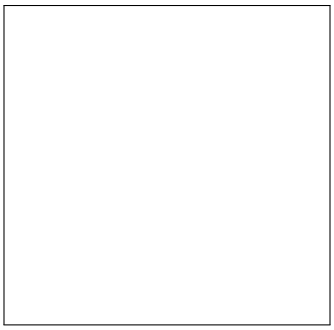

Figure 1: Sample figure caption.

Table 1: Sample table title

	Part	
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

5.1 Figures

5.2 Tables

All tables must be centered, neat, clean and legible. The table number and title always appear before the table. See Table 1.

Place one line space before the table title, one line space after the table title, and one line space after the table. The table title must be lower case (except for first word and proper nouns); tables are numbered consecutively.

Note that publication-quality tables *do not contain vertical rules.* We strongly suggest the use of the `booktabs` package, which allows for typesetting high-quality, professional tables:

`https://www.ctan.org/pkg/booktabs`

This package was used to typeset Table 1.

5.3 Margins in LaTeX

Most of the margin problems come from figures positioned by hand using `\special` or other commands. We suggest using the command `\includegraphics` from the `graphicx` package. Always specify the figure width as a multiple of the line width as in the example below:

```
\usepackage[pdftex]{graphicx} ...
\includegraphics[width=0.8\linewidth]{myfile.pdf}
```

See Section 4.4 in the graphics bundle documentation (`http://mirrors.ctan.org/macros/latex/required/graphics/grfguide.pdf`)