Implementing Racing AI using Q-Learning and Steering Behaviours

Blair Peter Trusler and Dr Christopher Child
School of Informatics
City University London
Northampton Square, London, UK
Email: btrusler@gmail.com / C.Child@city.ac.uk

KEYWORDS

Q-Learning, Reinforcement Learning, Steering Behaviours, Artificial Intelligence, Computer Games, Racing Game, Unity.

ABSTRACT

Artificial intelligence has become a fundamental component of modern computer games as developers are producing ever more realistic experiences. This is particularly true of the racing game genre in which AI plays a fundamental role. Reinforcement learning (RL) techniques, notably Q-Learning (QL), have been growing as feasible methods for implementing AI in racing games in recent years. The focus of this research is on implementing QL to create a policy which the AI agents to utilise in a racing game using the Unity 3D game engine. QL is used (offline) to teach the agent appropriate throttle values around each part of the circuit whilst the steering is handled using a predefined racing line. Two variations of the QL algorithm were implemented to examine their effectiveness. The agents also make use of Steering Behaviours (including obstacle avoidance) to ensure that they can adapt their movements in real-time against other agents and players. Initial experiments showed that both types performed well and produced competitive lap times when compared to a player.

INTRODUCTION

Reinforcement learning (RL) techniques such as Q-Learning (QL, Watkins 1989) have grown in popularity in games in recent years. The drive for more realistic artificial intelligence (AI) has increased commensurably alongside the high fidelity of experience which is now possible with modern hardware. RL can produce an effective AI controller whilst removing the need for a programmer to hard-code the behaviour of the agent.

The racing game used for performing the QL experiments was built using the Unity game engine. The game was built as a side-project in conjunction with this research. The cars in the game were created so that the throttle and steering values could be easily manipulated to control the car.

The biggest challenge when considering implementing RL is to determine how to represent and simplify the agent's state representation of the game world in an effective way to use as input for the algorithm. The information needs to be abstracted to a high level in order to ensure that only necessary details are provided. Two versions of the QL algorithm were implemented; an iterative approach and a traditional RL approach.

The results from the experiments demonstrate that when combined with steering behaviours both QL implementations produced an effective AI controller that could complete competitive lap times.

BACKGROUND

Reinforcement Learning and Steering Behaviours

RL is the method for teaching an AI agent to take actions in a given scenario. The goal is to maximise the cumulative reward, known as the *utility* (Sutton and Barto, 1988). The result of the RL process is a policy which provides the agent a roadmap of how to perform optimally. The RL process can be performed *online* or *offline*.

Online learning is the process of teaching the AI agent in real-time. Offline learning involves teaching the agent before releasing the game. Both methods have their merits and issues. For several reasons the offline version is most commonly used when RL is applied to games (and is used in this research). Primarily, it ensures that the agent will behave as expected when the game is finished. It also means there is less computational expense in real-time as the AI is behaving based on a saved policy and does not need to perform as many calculations in real-time. The offline RL process works by performing a large number of iterations (*episodes*) of a simulation in order to build up a data store of learned Q values relative to their state-action combination.

The concept of steering behaviours (SBs) was first introduced by Craig Reynolds (1999). SBs provide a mechanism of control for autonomous game agents. Reynolds proposed myriad behaviours which could be used independently of one another or holistically to achieve different behaviours.

There were three relevant SBs for this project; seek, obstacle avoidance and wall avoidance. Whilst SBs are not the focus of this paper, they were used to perform real-time avoidance techniques during the game when multiple agents were in the scene.

Q-Learning

Q-Learning is one of the most commonly used forms of RL and is a type of temporal difference learning (Sutton and Barto, 1988). QL is used to find the best action-selection policy for a finite number of states. It assigns utility values to state-action pairs based on previous actions which have led to a goal state. As the number of episodes increases, the utility estimates and predictions improve and become more reliable.

A *state* can comprise of any piece of information from the agent's environment. An *action* is the operation that the agent can perform at each state. The action selection policy is a key component to the learning process. The two common types of action selection are *greedy* and *ε-greedy* (Sutton and Barto, 1988). Greedy always chooses the optimal available action according to the current utility estimates. In contrast, ε-greedy has a small probability of selecting a random action to explore instead of choosing the greedy option.

The QL formula (1) is performed upon reaching a state. The QL formula is defined as follows:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a}{}'(Q(s', a'))) \qquad (1)$$

Where:

- $Q(s, a)$ – Q value of the current state-action pair
- $Q(s', a')$ – Q value of the next state-action pair
- r – reward value associated with next state
- α – learning rate parameter
- γ – discount value parameter

The learning rate and discount value parameters are crucial in defining the learning process. The learning rate determines to what extent newly acquired information will override the previously stored information. A learning rate value of 0 will mean that the agent will not learn anything whilst a rate of 1 means that the agent will only consider the most recently acquired data. The discount parameter defines the importance of future rewards to the agent. A factor of 0 creates a short-sighted agent which only considers current rewards, whilst a factor of 1 ensures the agent will aim for the highest possible long-term reward.

Q-Learning in Games

Patel et al (2011) used QL to create an AI agent for the popular first-person shooter game *Counter-Strike*. They used QL to train a simple AI agent in order to teach it how to fight and plant a bomb. A higher reward value was assigned to the AI if it accomplished the goal of the game. For example planting the bomb produced a higher reward than killing an enemy. Their results showed that the QL bots performed competitively against the traditionally programmed bots. However, they did note that this was not tested against players. This could identify further issues that would need to be resolved in the learning process

A popular commercial racing game that makes heavy use of RL is the Forza series (*Drivatars*). The development team created a database of pre-generated racing lines for every corner on a race track (several slightly different lines per corner). For example, some racing lines will be optimal whilst others may go wide and miss the apex of the corner. The agent uses QL (offline) to learn the appropriate throttle values to follow each racing line as fast as possible. The cars also learn various overtaking manoeuvres at each part of the track. During a race, the racing lines at each corner are switched to vary the behaviour. This approach meant that the programmers were not required to hard-code the values for each track and corner and produced a reusable and effective tool for creating AI agents for each type of vehicle. This technique has resulted in the Forza series having one of the most realistic AI systems in the racing game market today.

IMPLEMENTING Q-LEARNING

Game World Representation

The first challenge was converting the three dimensional game world into a series of states for the algorithm to interpret. Firstly, a racing line was generated by positioning waypoints along the race track and creating a Catmull-Rom spline by interpolating between these points.

The states were then defined as track segments (points along the racing line). The region was implemented by placing a box collider at each of these points. The collider width was equal to that of the race track width and rotated based on the direction of the spline. The quality of the state is evaluated based on the agent's proximity to the centre of the racing line and time taken to reach the state.

Discrete Action Space

It was decided to focus the QL on learning the cars throttle values whilst using the racing line to generate the appropriate steering values. This helped to reduce the action space to an appropriate size in order to minimise the number of iterations required to perform the learning process. The action space was set to nine evenly spaced throttle values ranging from +1.0 to -1.0 (where +1.0 represents full throttle and -1.0 represents full braking or reversing).

Q-Store Data Structure

A data structure (the *Q-Store*) was implemented to store all of the data required by the learning algorithm. The Q-Store maintained a two-dimensional array of doubles. The first dimension in the array represented the state values whilst the second dimension represented the action values. This allowed for the Q value for each state-action pair to be easily stored and accessed.

Q-Learning Algorithm

As previously mentioned two versions of the QL algorithm were implemented. Both versions are very similar in nature but with some key differences as highlighted in the following sections. The algorithm works by applying each action (throttle values) at each state on the track. A reward was calculated if the car reached or did not reach the next state and the QL formula was calculated and stored. Both versions used the greedy action selection policy.

The action policy generated from each version of the algorithm was stored in a text file. This allowed the policy to be retrieved and utilised without having to re-perform the learning process each time.

First (Iterative) Version

The first version of the algorithm was based on an iterative approach. The learning agent was designed to evaluate each possible action for a state before moving on to the next state. The agent would continually reset to the starting state after each evaluation. This meant that the agent would gradually make its way along the racing line and during the process the agent would ultimately evaluate the actions between the penultimate state and the goal state. This iterative approach meant that the number of episodes could be predetermined (number of states * number of actions).

Second (Traditional) Version

The second version was based on a more traditional RL approach. Unlike the first version the learning process did not continually reset in an iterative manner. It gradually developed a policy over a number of episodes (ranging from 10 to 5000 in testing). Theoretically, an increased number of episodes will make the policy more likely to allow the agent to reach the goal in an effective way.

Reward Function

The reward function used for the agents produced a reward value based on the quality of the action performed at the current state. The value returned by the function was based on whether the action performed was good or bad. A good move would return a positive scaling reward value based on two key factors (proximity to the racing line and time taken between the two states). A final large multiplier would be added to the reward value if the car reached the goal state (the final point on the racing line). A bad move (eg crashing) would result in the function returning a negative reward value.

Execute Policy

The policy was stored in a text file that consisted of a single value (representing the action number) per line (the state). The agent would identify its current state and apply the corresponding action as specified in the file until reaching the next state.

TESTING AND RESULTS

This initial aim of this research was to investigate whether QL could be used to create a high quality controller for a racing game. Subsequent to this goal, the two versions of the QL algorithm suggested a further area of research in order to determine how they differed and which performed to a higher level. Each version of the agent was taught using the same racing line, race track and car properties. The two agents were taught using the same number of episodes (1,000) for the first two experiments. The third experiment involved varying the number of episodes for the second version of the algorithm.

State-Action Tables (Q Tables)

The first area of comparison was between the Q Tables produced by each version of the algorithm. These tables were produced after the learning process was completed by retrieving the data from the QStore. Tables 1 and 2 show that there was a difference in action selection at state 93 whilst the same action was picked at state 94.

Table 1: State-Action Table (Version 1)

State	Action	Q Value
93	6	2805597255.12183
94	0	2920734984.09786

Table 2: State-Action Table (Version 2)

State	Action	Q Value
93	0	730021813
94	0	531860033

Lap Times

The overall goal of this research was to produce a high quality AI controller for a racing game using the two variations of the QL algorithm. As a result the most tangible measurement of performance provided by the project was in terms of lap-times.

The same race track and racing line was used for each version and they both started from the same position at the beginning of each lap. Ten laps times were recorded for each version The average lap times are shown in Table 3. The lap times were performed with the obstacle avoidance and wall avoidance behaviours disabled as there were no obstacles present in the scene to check for in real-time.

Table 3: Average Lap Time Comparison

Lap Number	Version 1	Version 2
Average	42.73594	42.65832
Standard Deviation	0.52378007	1.597068

Whilst the lap times were very similar, the first version appeared to produce more consistent results.

Episode Variation

Unlike the first version of the implementation, the second version could be taught using an indefinite number of episodes. This raised the question of what effect would varying numbers of episodes have on the lap-time produced by the agent. Up to this point, the results produced for the second version was taught using the same number of episodes as the first version of the algorithm (approximately 1,000).

Table 4: Episode Variation Table

Episodes	Lap Time / Result
10	44.33456 (crashed into wall)
100	44.96534 (crashed into wall)
1000	42.65832
1500	41.74825
2500	***40.95938***
5000	41.46755

The policies which caused the car to crash still managed to complete their laps as the car was built with a reset function to reset the car after 2.5 seconds to a point slightly further long the racing line. Table 4 shows that the fastest lap time was produced by the 2500 iteration version whilst similar lap times were produced by the 1000, 1500 and 5000 versions.

EVALUATION

State-Action Tables (Q Tables)

The state-action tables showed that the learning agents took a different approach entering the corner. The states chosen (93 and 94) were located before the tightest corner on the track. It is interesting to note the different actions selected for state 93. The first version selected a braking action whilst the second version selected the full throttle action. This was because the first version was focused on one individual state at a time. This meant it often braked at the latest possible state as it didn’t keep track of the reward based on the final end goal state. The second version had a more long-term view and as a result performed the braking action earlier (during states 89, 90 and 92) in order to achieve a better speed through the corner. This is because the QL function is aimed at achieving the highest possible long-term reward which is provided upon reaching the goal state. It would have been interesting to see the effect of different action-selection policies on the Q values produced.

Lap Times

The lap time comparison produced an interesting set of results. Table 3 shows the average and standard deviation between lap times for each version. The average lap time between the two algorithms was extremely close. The standard deviation, however, was very different. The first version appeared to produce very consistent lap times and results, whilst the second produced a wider range of very fast and relatively slow lap times. The slow lap times were often a result of going off track or hitting a wall. This would indicate that the number of episodes used to teach the second version was too low.

Episode Variation

This experiment was inspired by the standard deviation result in the lap-time test. The question raised was at what point was it that the number of episodes used cease to have an effect. Lap-times produced by the car were recorded for 10 laps. Table 5 highlights the average lap times produced and the standard deviation between them.

Table 5: Average and Standard Deviation for Episode Variation of Lap Times (Version 2 only)

Episodes	Lap Time / Result
Average	42.6889
Standard Deviation	1.62844

The results show that for 100 episodes or less, the car crashed or had an incident causing the lap-time to be increased. This was to be expected given the number of possible actions for the number of states in the game world. Interestingly, it also shows that the fastest lap time was produced from a policy created by 2500 episodes. In contrast the policies produced by 1500 and 5000 episodes produced relatively similar lap times.

One would have imagined that the lap time for 5000 episodes would have been at least as quick if not faster than the controller produced from 2500 episodes. This result is possibly due to the algorithm performing further learning and discovering that a policy for this type of lap-time would result in a crash in the tighter parts of the racetrack. Therefore it made safer choices whilst still maintaining a good overall speed.

Results Discussion

The lap-times produced by both versions are relatively competitive compared to player lap-times (with times ranging between 39 and 42 seconds on average depending on the type of player). The overall performance of the algorithm in terms of lap-time is restricted by the optimality of the racing line. The line was generated from waypoints that were implemented by hand and based on what appeared to be the best line around each corner. Better lap times would possibly have been achieved if this line was produced algorithmically to create a minimum-curvature line around the race track. It was also surprising to note that both versions produced relatively similar lap times despite the differing approach to the QL process.

CONCLUSIONS AND FUTURE WORK

This paper has presented the use of QL to produce an AI controller in a racing game. The results have shown that the controller produces reasonable lap-times and performance compared to a player. The QL formula used in this project was the standard QL approach. Other versions could have been used (eg SARSA) which may have produced differing or even improved policies for the AI controller.

There are several other areas that are open to investigation in the future. The most pertinent of these would be to utilise alternative reward functions. This could be used to create different types of AI controllers (ie varying difficulties or driving styles). A further development could have been to use multiple racing lines with differing lines into and out of corners. These lines could have been learnt and switched in real-time to produce more realistic and seemingly human behaviour. Another modification would be to increase the state-space of the game world. This would increase the size of the QStore but in turn increase the number of possible actions that can be taken around the race track. This could result in enhanced behaviour, in particular through tight or twisting corners. The state space could be expanded further by taking other factors into account such as the car velocity.

This project has shown that QL produces a reasonable controller without hard-coding a complex AI system. The racing line is the principle requirement to be implemented into the game world. In the future QL could be used to teach the agent how to steer based on its current position on the track and what lies ahead. This would then allow AI developers to focus their efforts on improving the agent's steering behaviours to create more realistic real-time interactions.

REFERENCES

Lucas, S, Togelius, J. 2007. Point-to-Point Car Racing: an Initial Study of Evolution Versus Temporal Difference Learning. *Symposium on Computational Intelligence and Games*. 1 (1), p260-267.

Moreton, H. 1983. Minimum Curvature Variation Curves, *Networks, and Surfaces for Fair Free-Form Shape Design*. United States: Berkeley. p1-213.

Patel, P, Carver, N, Rahimi, S. 2011. Tuning Computer Gaming Agents using Q-Learning. *Proceedings of the Federated Conference on Computer Science and Information Systems*. 1 (1), p581-588.

Reynolds, C. 1999. Steering Behaviors For Autonomous Characters. *Game Developers Conference*. 1 (1), p763-782.

Sutton, R and Barto, A. 1998. *Reinforcement Learning:An Introduction*. United States: MIT Press. p324-332.

Watkins, C. 1989. Learning from Delayed Rewards. London: King's College.

WEB REFERENCES

FIAS. 2010. Reinforcement Learning. Available: http://www.cs.utexas.edu/~dana/RL08.pdf. Last accessed 20th September 2013.

Microsoft. 2004. Drivatar. Available: http://research.microsoft.com/en-us/projects/drivatar/. Last accessed 16th September 2013.

Candela, J, Herbrich, R, Graepel, T. 2011. Machine Learning in Games. Available: http://research.microsoft.com/en-us/events/2011summerschool/jqcandela2011.pdf. Last accessed 16th September 2013.

Thirwell, E. 2013. Forza 5's AI is "much more engaging than anything you'll see in another racing game". Available: http://www.oxm.co.uk/62293/forza-5s-ai-is-much-more-engaging-than-anything-youll-see-in-another-racing-game/. Last accessed 20th September 2013.