Self-Driving Car in a Game Environment

Anirban Mookherjee
Anirban_Mookherjee@student.uml.edu

Murtaza Shams
Murtaza_Shams@student.uml.edu

ABSTRACT

This paper discusses an approach to a self-driving AI in a game environment. Such an AI will steer a car through a vertical scrolling driving simulation game. The paper proposes the division of the road into a grid wherein an evaluation function is used to calculate the utility of each cell based on its distance from a vehicle and its relative speed of approach to it. The AI then uses these values to steer the car in the direction of the cell with highest utility.

AUTHOR KEYWORDS

Artificial intelligence, reflex agent, evaluation function, driving, traffic, game, self-driving, simulation, utilities, collision, optimal path

INTRODUCTION

Many variations of simple traffic racing games exist. These are never ending simulations where a user steers a car in an attempt to avoid obstacles. Such games are usually single agent where a user controlled car changes lanes to avoid collision with other cars. The only actions available to the user are move left or move right. The performance of the user is determined by a score which is a function of time and speed. As the game progresses, the relative speed of the agent car increases, making the game more challenging. With the introduction of an AI agent we have demonstrated their dominance in an environment that needs quick response times.

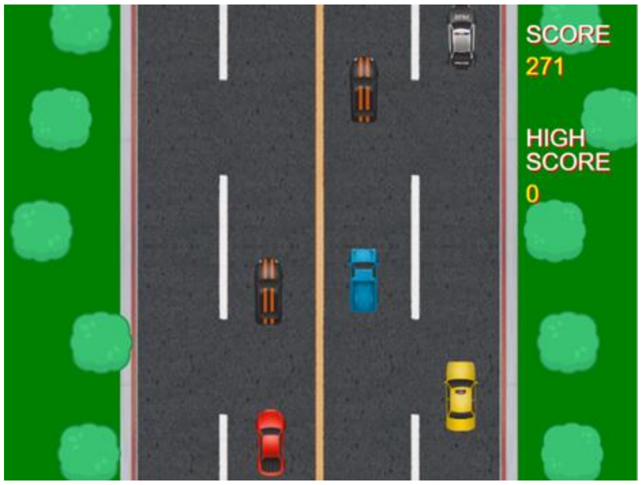

Figure 1. Graphic representation of the game environment

PROJECT DESCRIPTION

This approach to a self-driving car was carried out on an existing game by the name of Simple Traffic Racer. This game, written in JavaScript is generic and simple. Such an implementation is well suited for our needs.

Game Environment

This game is implemented as a continuous infinite stretch of road comprising of 4 lanes. It consists of a car controlled by an agent which remains within the environment, in one of these 4 lanes, at all times. It also consists of other traffic vehicles, not under the control of any player or AI. Henceforth, these vehicles shall be referred to as obstacles. At any given time, the agent's car or obstacles can only exist within the defined boundaries of a single lane. Transition from one lane to another is smooth yet instantaneous. Actions that possibly lead to a transition outside of the road boundaries are ignored, and has no effect whatsoever on the dynamics of the game.

The game starts out slow, that is the relative speed of approach of the agent's car towards any obstacle on the road is low. The agent's car is faster than any other car on the road, which leads to a deterministic scenario of overtaking. Every obstacle must therefore transition smoothly into the game screen from the top, and transition out from the bottom. These obstacles will remain in a specified lane during their transition in and out of the environment. As time progresses, the speed of the agent's car and consequently its relative speed of approach to obstacles increases. The game applies a collision mitigation system between two or more obstacles. This ensures that the only possible collision in the game would involve an agent car and an obstacle, not just two obstacles. This system detects a collision and sets the speed of both obstacles as the minimum of their individual speeds, when approaching a collision among them.

The agent has only two possible actions. It can either move to its left lane or its right lane. If a move right request detects the end of the road boundary, the agent will remain in its current lane. Actions of accelerating or slowing down are not available to the agent. Any collision would end the game and display the score achieved as a performance indicator.

State Space Formulation

The evaluation function discussed later on this paper requires the game state to be static. This ensures that the utilities of cells do not change before an action takes place. The game environment is dynamic, therefore a static representation of the environment must be perceived which would enable the function to perform operations on it. To accomplish this, calculations are performed in fixed time frames at regular intervals such that at each given time frame, the game environment is static.

The environment is partitioned into 16 identical cells. Each of these cells have a fixed range of x and y coordinates. This ensures that the cell position remains constant in our dynamic game environment. The boundaries of these cells are well defined rectangles which do not overlap with each other, therefore at any given time an object can only exist in a single cell. Although not overlapping, the cells are positioned in a manner which leaves no undefined space between them.

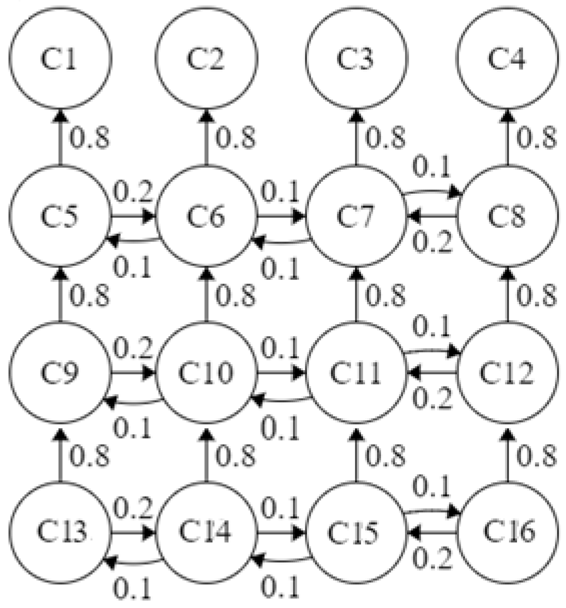

Figure 2. Grid representation with assigned weights

Each cell has a value associated with it known as its utility. This value is calculated at run time and constantly varies, as observed during game execution. The figure above demonstrates the layout of the cells and shows assigned weights used in the calculation of their utility. Arrows from one state to another shows which cells a given cell acquires its utility from. An outward arrow from cell A to cell B shows that state A's utility is partially calculated from state B's utility. An assigned weight shows to what extent B's utility affects A's utility.

Cells C1, C2, C3, C4 have no outwards arrows. This is because the value of their utility depends solely on the presence of an obstacle within their boundaries. If an obstacle is present, the utility of these cells will take on a negative value. This negative value is proportional to the relative speed of the obstacle. Specifically, for an obstacle whose relative speed of approach is low (travelling at a high speed), the utility will take on a lower negative value than that of an obstacle whose relative speed of approach is high (travelling at a low speed). The logic behind this is that an obstacle with a high relative speed of approach (travelling at low speed) poses a bigger threat to the agent's car. Therefore, the larger the negative value of a utility is, the sooner will the AI agent take a series of actions to avoid it. If no obstacle is present, the utility will take on a high positive value, equal to 200 if it is in the same lane as the agent car and 100 if it is on a different lane. For the remaining cells the utility is determined by the cells next to it, namely the cells to its left, right, and top. The following formulas demonstrate this principle.

$U(C_5) = 0.8U(C_1) + 0.2U(C_6)$

$U(C_6) = 0.8U(C_2) + 0.1U(C_5) + 0.1U(C_7)$.

$U(C_7) = 0.8U(C_3) + 0.1U(C_6) + 0.1U(C_8)$.

$U(C_8) = 0.8U(C_4) + 0.2U(C_7)$.

$U(C_9) = 0.8U(C_5) + 0.2U(C_{10})$

$U(C_{10}) = 0.8U(C_6) + 0.1U(C_9) + 0.1U(C_{11})$.

$U(C_{11}) = 0.8U(C_7) + 0.1U(C_{10}) + 0.1U(C_{12})$.

$U(C_{12}) = 0.8U(C_8) + 0.2U(C_{11})$.

$U(C_{13}) = 0.8U(C_9) + 0.2U(C_{14})$

$U(C_{14}) = 0.8U(C_{10}) + 0.1U(C_{13}) + 0.1U(C_{15})$.

$U(C_{15}) = 0.8U(C_{11}) + 0.1U(C_{14}) + 0.1U(C_{16})$.

$U(C_{16}) = 0.8U(C_{12}) + 0.2U(C_{15})$.

However, if an obstacle exists in any one of the cells C9 – C16, the value of the utility will be overridden by a high negative value equal to -100. This is because, we later check for this value to override the car movement decision, which is initially solely based on the cell utility values in the same row, in order to avoid collision with adjacent obstacles while attempting to move to the cell with highest utility.

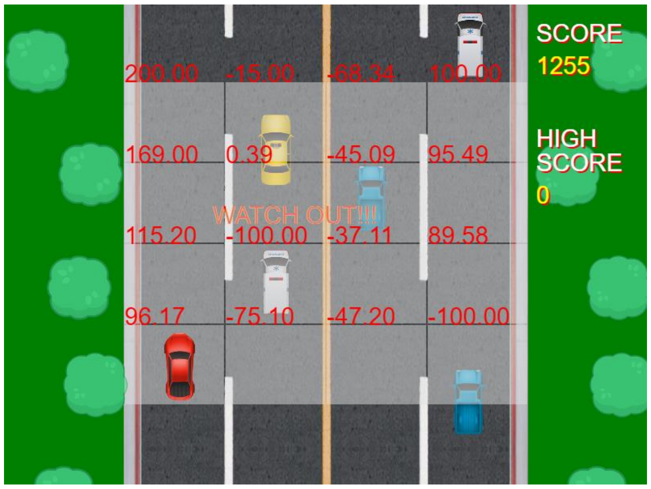

Figure 3. Values calculated by evaluation function

ANALYSIS OF RESULTS

TEST MODELS

Before testing and analyzing for the results, three models of the game were made. Each model had some performance tweaks to the evaluation function and AI agent's behavior. Each model was run for 150 times and the AI agent's performance was measured. We noted the average, maximum and minimum scores for each.

Model 1

While testing our primitive model we noticed that our AI agent took a lot of actions between cells whose utilities were almost identical. This is because the AI agent had been programmed to reach the cell with the highest utility. This was usually unnecessary and sometimes led to easily avoidable crashes. Moreover, this driving behavior looked far from normal. An AI agent continuously traversing between two relatively safe cells brings no positive outcome to the performance, and is merely a computational overhead. An AI agent should only perform an action if it is quantitatively justifiable.

This model rectified this behavior of the agent by implementing a utility difference check function. Thus, we not only determined the cell with the highest utility, but also evaluated their temporal difference. With this added information, the agent could make a rational decision in determining if an action at any given time is justifiable or not. A temporal difference of greater than 2 seemed to be an optimal threshold for a justifiable action. The outcome of this model supported the idea of an informed AI that took reasonable actions.

Model 2

While testing of Model 1, we noted that the AI agent at times drove almost recklessly to reach the cell with highest utility and in the process ended up colliding with a fast approaching obstacle. Initially the AI agent, when traversing to a cell not immediately next to it, only checked if there was an obstacle in the intermediate cell. This check is done by checking the utility of the intermediate cell. If this utility was equal to -100 as explained in our discussion of the evaluation function, the agent was aware of the existence of an obstacle in that cell. In such case, the agent would avoid traversing through that intermediate cell, to reach the numerically evaluated best cell. However, it turned out that at higher speeds, this condition check was not enough as the agent car was colliding with a fast approaching obstacle in succeeding cell. It was appropriate that we also check the cell diagonally in front of the agent car to determine if it is safe to switch lanes.

An obstacle with a relative high speed of approach leads to a much smaller window of opportunity for the agent car to switch lanes without crashing into approaching obstacle. Therefore, a look ahead was needed to see if such an obstacle existed in the cells above the intermediate one. The AI agent would now check the utility of the intermediate cell and of the cell directly on top of it. Each state would be checked by a different bound, and when any of these conditions was true, the agent's transition was deferred. As the game progresses, obstacles with such characteristics spawn more frequently. Therefore, this modification to the agent's behavior had a positive impact on the long term performance of the agent.

Model 3

Earlier, we touched upon the implementation of collision detection between two obstacles. This function avoided a collision between two obstacles by setting both of their speeds equal to the minimum of the two. This behavior sometimes led to a scenario where obstacles would start to pile up in a lane. The relative speed of these obstacles become quite low as they are set to the minimum of them all. This pileup would hinder the transition of the AI agent, which would continuously search for the best state for it to be in and perform a series of checks to see if that state was reachable. At times it would take a risky action to traverse through the lane containing piled up obstacles.

In such a scenario, it was best for the AI agent to remain in the current state until such obstacles were no longer in the way. To achieve this the utility of the top most state of the current lane had to be increased. This encourages the AI agent to remain in the lane and avoid performing a risky action. As discussed earlier, we make the utility of the top most cell in same lane to be 200 as long as there are no obstacles in it. This seemed optimal to make sure that the agent car does change lane when necessary while avoiding frequent unnecessary changing of lanes. After the implementation of this model, tests were run to check if the modification produced positive results. While there were some negative impacts of this approach, the positive impacts outweighed them.

	Average	Minimum	Maximum
Model 1	3946.211	414	21166
Model 2	5549.217	567	30778
Model 3	5073	377	35097
Manual Run	3600	487	6868

Table 1. Average, Minimum and Maximum scores of the three models and a manual play.

The table above shows the results for the three models discussed earlier and a manual run. The manual run refers to scores we achieved by manually playing the game without the presence of an AI agent. The goal of this implementation was to show the extent to which the AI agent performance surpassed that of a human player. The lowest average score of the three models was for model 1. However, when compared with the results of manual run, we can see that even this model achieves a higher average score. Although the highest scores achieved by our later models are leaps ahead than that of the manual run scores, one may feel that an AI agent designed to take the numerically optimal choice in all states should ideally score infinitely high, however that is not the case due to the presence of unavoidable collisions, for example, if obstacles approach in an order such that in many consecutive game states, the path to the cell with highest utility is blocked by an obstacle. Yet we can see that the maximum score amongst all, achieved by model 3, is roughly 5 times better than the maximum score of a human player. The best performing model by average score was model 2. The look ahead modification proved to be very successful. Model 3 although possessing a lower average than model 2, outperformed in the maximum score, which depending on the goal of the user, may be a preferred model. Model 3 had a modification which encouraged the AI agent to avoid risks. This proves successful in the long run.

The results proved to be a major improvement over what an actual human player can achieve. This highlights the potential of AI in the field of games and simulation. During our proposal we had decided that achieving better overall scores in the game than a human player would allow us to declare this project a success. Evidently we have surpassed our initial goal. We are highly pleased with the results as per our planned metric of success.

DISCUSSION

This project provided us with an awesome learning experience since it was our first hands on application of the theoretical AI concepts learnt in class. There was a lot of reflection involved in ultimately choosing the AI construct to use for the task at hand. As would have been evident from our project description, we use a reflex agent that acts based on an evaluation function, manually curated by us to achieve optimal results.

The division of our environment into a grid with several cells where the agent needs to move to the highest utility cell had initially given us the impression that the problem at hand would be best solved by using Markov Decision Process. However, with deeper analysis we realized that this was really an eye-deceiving scenario. The cells in our grid do not represent “states” in the same sense as used in a Markov Decision Process. These cells were merely position representation of several parts of the game environment and the utility of all the cells in a given time frame in fact represented a single “state” in the game.

In our proposal stage we had planned the use of MDP but the reason this did not work out is multifold. Our environment was deterministic and as a result there was not transition function. Markov Decision Process, by definition, is used to find an optimal policy for decision making in situations where outcomes are partly random. This invalidates the use of most MDP based AI constructs.

Our second consideration was to use a search algorithm such as the expectimax. However, this was also not ideal in our case. Although it may look like it in first look, the series of actions performed in the game did not consist of a user move followed by a random move, as is the case in expectimax search. We have a dynamic environment wherein our calculations need to be performed in each time frame. In a given time frame we have a single agent static environment with no randomness and the only move performed is that by the AI agent, to turn left or right. All moves are made based on the current visible state of the environment and not based upon the action of another agent.

Dealing with such a continuous dynamic environment was a new scenario compared to most scenarios encountered during our course. Since calculations had to be performed repeatedly, with continuously decreasing time frame intervals, to keep up with the increasing game speed, we felt that using a reflex agent would also be the most computationally efficient strategy. Moreover, the action that had to be performed in each time frame was very simple, namely move left or move right. This did not really warrant the use of an advanced AI algorithm.

That said, what we could however have done differently, given more time, was to incorporate some learning in our AI agent. It is possible to learn the best evaluation function by rewarding the action taken in each time frame in response to the immediate outcome. This is something we can keep in mind as me move forward from here, expecting to have the opportunity to work in many more projects in the domain of Artificial Intelligence.

Ethical implications of this work

The use of AI in this project is clearly harmless from an ethical perspective. However, if we had to point out one possible scenario with a negative ethical implication, we would say that the use of such AI may be unethical if used in competitive games to score higher than humanly possible. When games have leaderboards with serious rewards for high scorers, the use of AI may be totally unintended and the unadmitted use of AI in such case will obviously be unethical.

CONCLUSION

To summarize what has been discussed in depth in this paper:

- This project was greatly successful as per our initially planned metric for success. Specifically, we intended our AI agent to perform at least as good a human in an infinite vertical scrolling car driving game. We ended up achieving a performance that on average is twice as good as that of a human and the high scores obtained are way beyond what is humanly ever possible.
- Using a reflex agent that performs based on a manually curated evaluation function, works very well for our given problem.
- If we had more time we would have liked to incorporate a learning feature in our AI agent. Our current evaluation function would then be a good place to start with while gradually tweaking the weights automatically as our agent would reward itself in response to the outcome of each action. This may or may not have led to a better performance because of the randomness of the possible game states in a given time frame. Since our environment is dynamic and continuous, the actual number of possible states to learn from would be almost infinity if we consider what portion of each traffic vehicle is in each cell. We could possibly consider scenarios with different portion of vehicles in different cells as different states, resulting in huge number of total possible states. As a result, some approximation will have to be incorporated, to make learning feasible. Whether this would have worked out, we could only have found out with more time.

RELATED WORK

The idea of our state representation was taken from Biasillo [1] who proposed a representation where race tracks were divided into sectors which included information such as racing path. Although his implementation used representing these values at the boundary of such divisions, we altered the idea to create a cell that held these values.

Paden’s [2] idea of classifying steps for a self-driving car was a useful insight. His idea of motion planning included the information needed by an AI agent before it could take actions such as changing lanes. He talk about how the path to be chosen must be feasible for the agent that controls the car. This proved to be useful, especially in the model 2 of testing where modifications were made to the agent to make it more aware of the consequences of an action, and make sure that the actions that it carried out can be quantitatively justified.

The presence of an obstacle in a cell had impact on the utility of that cell, however using the word presence is a vague term. It was necessary to detect an obstacle and define it with boundaries to know where it exactly existed, as in would continuously move in a dynamic environment. Lee’s [3] idea of obstacle detection proved useful. His research was not similar to ours as he proposed this idea for a self-driving car capable of dealing with changes in weather. However his idea of obstacle detection was quite generic. He was interested in measuring the highest and lowest point of the obstacle in an attempt to make an educated guess of its dimensions. We implied a similar, easier version of this where an obstacle was defined x and y values which represented a region covered by this obstacle.

Our implementation consists of dealing with obstacles with varying speed of approach. Thus the danger posed by different obstacles cannot be the same. A model had to be crafted which took into account the relative speeds on these obstacles and calculate the risk associated with existing in the same lane as they spawned. Workings of this model was taken from Chaocheng [4] principle of *“Threat Probability Area and Vehicle State Prediction”*[4]. This concept talks about the evaluation of an area which is considered to be a state of threat for a self-driving agent. The author’s idea also included the concept of vehicle state prediction associated with it. This idea was dropped as our game was deterministic and random behavior was absent from obstacles. The threat probability area was a degree of variance reflected in our cell utility calculation. The larger the speed of approach, the greater negative impact it would have on the cells utility.

Many of our workings of the state space resemble those of Agapitos “Stateful Program Representation” [5]. His implementation of x and y values for valid location points and division of the environment into a 2D array provide an idea similar to ours. However his cells were smaller which greatly increased the total number of cells, leading to a larger array size. Our cells were kept larger to reduce their total numbers as a method of avoiding computational complexities that would ruin the Ai agent’s performance.

ACKNOWLEDGMENTS
We would like to thank the following parties for making this project possible.

- The work described in this paper was conducted as part of a Fall 2016 Artificial Intelligence course, taught in the Computer Science department of the University of Massachusetts Lowell by Prof. Fred Martin. Without his interactive and captivating teaching methods and everything that we could learn from his class, working on this project would never have been such a blast.
- We would like to thank the University of Massachusetts Lowell for providing every technical facilities required almost 24x7.
- We would like to thank many of our peers for their valuable feedback, comments and suggestions whenever we looked to get an outside perspective on our project.

REFERENCES
1. Gari Biasillo, "Representing a Racetrack for the AI", AI Game Programming Wisdom, Charles River Media, Massachusetts, 2002.
2. B. Paden, M. Čáp, S. Z. Yong, D. Yershov and E. Frazzoli, "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles," in IEEE Transactions on Intelligent Vehicles, vol. 1, no. 1, pp. 33-55, March 2016.
3. U. Lee et al., "EureCar turbo: A self-driving car that can handle adverse weather conditions," 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, South Korea, 2016, pp. 2301-2306.
4. Chaocheng Li, Jun Wang, Xiaonian Wang and Yihuan Zhang, "A model based path planning algorithm for self-driving cars in dynamic environment," 2015 Chinese Automation Congress (CAC), Wuhan, 2015, pp. 1123-1128
5. A. Agapitos, M. O'Neill, A. Brabazon and T. Theodoridis, "Learning environment models in car racing using stateful Genetic Programming," 2011 IEEE Conference on Computational Intelligence and Games (CIG'11), Seoul, 2011, pp. 219-226.