Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2015, Article ID 839721, 6 pages
http://dx.doi.org/10.1155/2015/839721

Research Article

Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques

Marvin T. Chan, Christine W. Chan, and Craig Gelowitz

Software Systems Engineering Program, Faculty of Engineering and Applied Science, University of Regina, Regina, SK, Canada S4S 0A2

Correspondence should be addressed to Christine W. Chan; christine.chan@uregina.ca

Received 29 June 2015; Revised 30 September 2015; Accepted 8 October 2015

Academic Editor: Manuel M. Oliveira

This paper presents a car racing simulator game called *Racer*, in which the human player races a car against three game-controlled cars in a three-dimensional environment. The objective of the game is not to defeat the human player, but to provide the player with a challenging and enjoyable experience. To ensure that this objective can be accomplished, the game incorporates artificial intelligence (AI) techniques, which enable the cars to be controlled in a manner that mimics natural driving. The paper provides a brief history of AI techniques in games, presents the use of AI techniques in contemporary video games, and discusses the AI techniques that were implemented in the development of *Racer*. A comparison of the AI techniques implemented in the Unity platform with traditional AI search techniques is also included in the discussion.

1. Introduction

Games have become an integral part of everyday life for many people. A traditional game often presents a situation where "players engage in an artificial conflict, defined by rules and results in a quantifiable outcome" [1]. Such artificial conflicts are often represented as a puzzle or a challenge, and having the puzzle solved or the challenge resolved provides a real-world purpose to the game players [2]. This type of games is sometimes referred to as "serious" games [3]. However, this kind of traditional or "serious games" has been increasingly replaced by electronic games, especially for the so-called "game generation" [4]. This generation typically consists of "digital natives," who, in contrast to the "digital immigrants" [4] of the older generation, grew up playing a lot of games and who are trained in skills such as "dealing with large amounts of information quickly even at the early ages, using alternative ways to get information, and finding solutions to their own problems through new communication paths" [5].

The artificial intelligence (AI) community has witnessed a similar transition from the "classical AI games" such as Samuel's Checker Player [6] and Waterman's Poker Player [7] to the contemporary AI techniques adopted in electronic games. The objective of the game is no longer a quantifiable outcome of beating the opponent in a checker or poker game. Instead, a contemporary game contains changing environments, multiple objectives, and dynamic aspects of the game that are revealed to the game player as the game unfolds. The objective is to offer to the human player an enjoyable experience through his or her interaction with the game, and this does not involve any specific quantifiable outcome.

The game *Racer* is a contemporary video game, and its objective is to offer the player a challenging and enjoyable experience in a car race against a game-controlled car. Although the player's goal within the game is to win the race against the game-controlled car, the AI techniques adopted in the game are primarily designed to give the player an enjoyable time racing his or her car. In other words, the objective of winning by either side is not given the highest priority.

This paper proceeds as follows. Section 2 provides some background literature relevant to the work. Section 3 presents the design of the software system of *Racer*. Section 4 presents implementation of the game software system using some AI techniques. Section 5 gives a sample execution of the game system. Section 6 presents some discussion on the AI techniques adopted in *Racer* and its performance, and

Section 7 includes the conclusion and some ideas on future directions of the work.

2. Background Literature

There has been substantial research work that focuses on developing AI-controlled components of game systems, which can approximate or emulate human game playing styles. These components are often referred to as "bots." The motivation for developing the "bots" is that a human player's enjoyment in the gaming experience will be higher if he or she can be led to believe that the opponents in the game are other human players. An example of this type of game is the popular multiplayer first person shooter game Counter-Strike [8], in which the objective is to eliminate all players on the enemy or opponent team.

Similar AI-controlled components or "bots" also appear in the early video game of PacMan; the computer-controlled ghosts in the maze are able to move towards the player-controlled characters because the former incorporated some path finding algorithms [9]. Other AI algorithms that were adopted by games include finite state machines, fuzzy state machines, and decision trees. Decision trees are used to represent the decision making process involved in games like Checkers or Chess. A decision tree can specify a number of possible game states given a current state and support a search process for a goal state, where the human player is defeated provided the AI-controlled game can identify the path to the goal state more efficiently than the human player.

In addition to decision trees, other AI techniques are also used for building bots so that they would mimic or emulate human behavior. Khoo and Zubek discussed their use of a behavior-based action selection technique in the software development kit (SDK) of Flexbot for developing an AI-controlled bot, called Groo [10]. Based on Groo, 32 bots were developed and can run simultaneously on a single machine, and human players who played against the bots believed the bots effectively emulated human behavior [10]. Similarly, Tatai and Gudwin [11] discussed the use of semiotics in their development of a bot, which can assist or attack the human player in the game of Counter-Strike. The objective of their work was to design the AI system so that the bot can entertain and challenge the human player and not simply to defeat him or her. Therefore, the performance of the bot is measured not by its efficiency in killing its opponents, but by its "capability to emulate as close as possible all the activities commonly performed by a human player" [11]. This consideration also means that the bot cannot be seen to be waiting because human players expect bots to be continuously working. Hence, the bot's reaction time was one of the criteria in measuring its performance [11].

Similarly, in driving and racing games, human players expect to control their cars by making small adjustments to the car's direction while driving, just like how they would control a car in real life. Therefore, the race car bot must make continuous adjustments to its car's direction as opposed to simply driving in a straight line and turning only when a curve in the road approaches. To ensure the race car bot can mimic human behavior in this manner, some AI techniques were used for implementing the bots in *Racer*, which are explained in detail in the following section.

Figure 1: Waypoints at key positions on the racetrack are shown as (red) spheres.

3. Game System Design

The design of the *Racer* game software involved three contemporary game AI concepts and techniques, which are presented as follows.

3.1. Waypoint System with Vector Calculations. The first technique used was the simple waypoint system used for controlling the car. The waypoint system includes a set of waypoints and each waypoint is a coordinate in 3D space that represents a key position on the racetrack. This system was used by many early car racing games because of its effectiveness and simplicity. In *Racer*, waypoints are stored as a sorted list of positions in 3D space using the List generic class from the System.Collections.Generic namespace [12]. The game software system reads this list of waypoints as input and then iterates through the list until all the waypoints have been passed by the game-controlled car. The game system only iterates to the next waypoint in the list when it detects that the game-controlled car is within a specified distance from its current waypoint. The (red) spheres along the racetrack in Figure 1 indicate the waypoints at key positions on the track.

In order to turn the car towards its current waypoint, the game system performs a series of vector calculations, which determine the amount of steering the game system needs to provide to the car. These calculations are based on an initial vector created by both the position of the current waypoint and the current position of the car itself. This series of calculations produces an output vector, which provides the bases for the steering and braking output levels of the car. However, during development of the game, it was found that using only the method of waypoints with vector calculations was ineffective in controlling the car. Due to the nonlinear relationship between the input and output vectors, the car could not be controlled using this method alone, and, therefore, the conditional monitoring system was adopted to augment the method of waypoints with vector calculations.

3.2. Conditional Monitoring System. Due to the nonlinear relationship between the input and output vectors, the steering and braking output levels of the game-controlled car could not be determined solely from the calculated output vector in the waypoints system. It was necessary for the

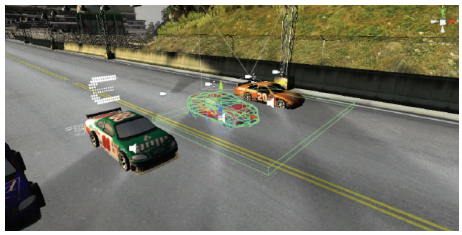

Figure 2: The race car and its associated trigger detection area are highlighted in the (green) wireframe mesh.

Figure 3: The interface of the Unity editor.

game system to also employ a conditional monitoring system that can further refine the steering and braking output levels applied to the car. In other words, depending on the values of the initial calculated output vector produced from the waypoint system's vector calculations, the steering and braking output levels applied to the car are adjusted by the conditional monitoring system. When the foundational waypoint system was enhanced with the conditional monitoring system, the car was able to traverse the track with satisfactory results.

Instead of adding the conditional monitoring system, the alternative design of deriving and incorporating a nonlinear mathematical function into the game system was also considered. Since this may require complex mathematical manipulations or modeling of the mathematical function using techniques like artificial neural networks, this alternative was abandoned.

3.3. Artificial Environment Perception. The second method that the game system employs in controlling the car is the use of trigger detection and artificial environment perception. Trigger detection is a popular mechanism used in contemporary video games, and it supports detecting a specific game object when it is within the vicinity of another game object. This method can determine if the game-controlled car may run into some object such as a wall. To prevent the collision from happening, the game system can adjust the steering and braking output levels applied to the car. The mechanism that supports this feature is the trigger area added to the car. If a wall enters into this trigger area, the game system becomes aware of where the wall is with respect to the car and it would adjust the output levels accordingly so as to steer the car away from the wall. With this addition of the trigger area, the game system is able to control the car so that it can navigate the track satisfactorily. The car in the middle of Figure 2 highlighted in the (green) wireframe mesh has a trigger detection area defined as the (green) rectangle around it. When an obstacle or another car enters into the trigger detection area, the output levels of the highlighted car will be adjusted so as to steer the highlighted car away from the obstacle.

An alternative technique considered for implementing this environment perception component of the game was the ray-casting technique. Ray-casting is a mechanism commonly used in contemporary video games, which involves drawing a ray or line from an object towards a specified direction, which, in the *Racer* game, would be the direction the car is heading. This drawn line would be used to determine if the car is about to collide with another object. However, the ray-casting mechanism is inadequate in that it can only detect collisions in one direction while the trigger detection mechanism can detect collisions in all directions.

4. Implementation of Game System

The component technologies that function as the higher-level abstraction programming tools and implemented the design of the game system described in the previous section are presented as follows.

4.1. Unity Game Engine. The game's 3D environment was built using the Unity game engine developed by Unity Technologies [13]. Unity is a game development engine combined with an integrated development environment (IDE) that is capable of creating applications for Windows, Mac OS X, Linux, and other mobile platforms [13]. The Unity game engine can perform graphical rendering and physics calculations, thereby enabling game developers to focus on the creation and development of the game software. Using the Unity engine, developers can position, scale, and add realistic physics to objects in the game system. Figure 3 shows the Unity editor's interface. The left panel shows a list of defined game objects, the middle visualization shows the game objects in the 3D space of the racetrack, the right panel shows the attributes of a selected game object, and the bottom panel consists of icons of the scripts, each of which defines the behavior of a game object.

4.2. MonoDevelop. The Unity game engine works in conjunction with MonoDevelop for controlling the behavior of objects. MonoDevelop is an open source Integrated Development Environment or IDE developed by Xamarin and the Mono community [14], which is primarily used for development in the C# programming language. With MonoDevelop, developers are able to write scripts in C#, JavaScript, or Boo and attach them to objects in the Unity engine; these scripts enable developers to control the logic and behaviors of objects within the Unity environment [13]. For example, if a developer was implementing the braking system on the wheels of the player-controlled car, she/he would write and attach a script that takes keyboard input from the user and applies the input value to the wheel's velocity. In this way, the combined tools of Unity and MonoDevelop enable developers to focus on the development of the AI components

Figure 4: An excerpt of the script developed with the MonoDevelop IDE interface.

of the game rather than on issues related to 3D graphics rendering and physics calculations. Figure 4 shows the script inside one of the icons shown on the bottom panel of the Unity editor; development of the script was supported by the MonoDevelop IDE.

4.3. Programming Languages and Data Structures. The primary programming language used for writing the scripts to control the behaviors of objects was C#. C# was chosen because it is object-oriented and class-based. Also, it has a variety of libraries and frameworks built specifically for development with Unity, which contain many generic classes and interfaces such as the List, Stack, and Dictionary [13]. These built-in classes have been optimized and are accessible with online documentation. For example, the List generic class from the System.Collections.Generic namespace was adopted during development to become an integral part of the game's waypoint system.

4.4. Detailed Implementation of Racer Game System

4.4.1. Player-Controlled Car Module. The player-controlled car module consists of four submodules: (i) the body, (ii) the wheels, (iii) the heads-up-display (HUD), and (iv) the player controller script. The body of the car is the 3D model that the player sees in the game environment; this submodule also contains the colliders, which enable the car to collide with objects in the game environment. The wheels of the car contain wheel colliders, which enable the wheels to make contact with the road and drive the car forward. The wheels submodule also contains a script which animates the rotation of the wheels as the car moves. The HUD component of the car is controlled by the HUD.cs script, which handles displaying the current speed, position, and lap of the car to the player. Lastly, the submodule of the player-controller script, called PlayerCarController.cs, handles the user input from the keyboard and applies the appropriate steering, accelerating, and braking output levels of the player-controlled car.

4.4.2. Game-Controlled Car Module. The game-controlled car (or the AI-controlled car) consists of four submodules: (i) the body, (ii) the wheels, (iii) the trigger area, and (iv) the AI controller script. The body and wheels of the game-controlled car are identical to the body and wheel components of the player-controlled car. The trigger area component

Figure 5: The start screen of *Racer*.

Figure 6: A screenshot of *Racer* during gameplay.

of the car is controlled by the CarSensor.cs script and detects when a wall enters into the trigger area. When this happens, the AI controller script is informed so that it would slightly adjust its control of the car. The AI controller script, called AICarController.cs, contains all the implemented AI techniques and algorithms that handle the driving of the game-controlled car so that it can effectively race against the player-controlled car.

4.4.3. Environment Module. The environment module consists of two submodules: (i) the racetrack and (ii) the waypoints system. The racetrack is the road within the game environment on which the cars race, and the waypoints system represents the key positions around the racetrack. The latter enables the game-controlled cars to navigate around the racetrack.

5. Sample User Run of *Racer* Game System

To control the player car, the user inputs commands to the car through the keyboard: the "W" key is pressed to accelerate, the "S" key is pressed to use the brake, the "D" key is pressed to steer right, and the "A" key is pressed to steer left. The player needs to finish three laps of the racetrack in order to win. Figure 5 shows the start screen of the game. Figure 6 shows the *Racer* game during gameplay. Figure 7 shows the indicators of the game components, which include the two game-controlled cars or "AI Car" in the background, the player-controlled car or "player car" in the foreground, and the "heads-up display" or HUD that indicates the status of the "player car": (i) speed or SPD at 90 km/hr, (ii) position or POS at 4/4, which means the car is at the fourth position among four cars, and (iii) lap or LAP at 0/1, which means the "player car" is in the first lap.

Figure 7: The game interface with indicators of the game components.

6. Discussion

Development of the game system was made easier because of the implementation tools. The Unity game engine supports effective development of the game system of *Racer* with its high-level abstraction programming tools and intuitive user interface. These features support developers in implementation of AI concepts so that they can focus on the game logic and ignore lower level development details such as graphics rendering and physics calculations. The combined tools of MonoDevelop and Unity support the implementation processes because MonoDevelop consists of autocorrection features for many libraries and SDKs used in Unity. For example, the combined tools support the developer in implementation of the waypoint system because the Unity engine supports positioning waypoints in 3D space. Hence, the developer can use the waypoint system instead of more traditional AI search techniques in determining a path for the race car on the track.

Compared to traditional AI search techniques, the waypoint system in the Unity platform enables the car racing system of the game to be more efficiently developed. If the Unity platform was not used, the racetrack would be mapped to a set of coordinates or nodes, which represent the 3D search space that covers the track. Then the path that the race car follows on the racetrack can be determined using either a blind or heuristic search algorithm, which identifies the nodes to be included in the path in the 3D space of the racetrack. In a blind search, the car will proceed to the next node in either a breadth-first or depth-first manner, and the node chosen as the subsequent node at each step will be the one that advances the car furthest along the track. In a heuristic search method such as the A^* search algorithm, the next node will be determined with an evaluation function that combines the length of the path traversed and the length of the path that has yet to be taken. Since the path traversed is not of interest for the race car, the evaluation function can consist of only the length of the path that has yet to be taken, which will move the car farthest along the track in a direction towards the destination. In implementing either search method in the game's 3D space, it is important that the set of nodes included in the path, through which the car traverses, represents a smooth trajectory that mimics human driving. In other words, the path should neither include any sharp turns nor float above or below the track, for a race car is expected to cruise along the racetrack smoothly. This would require substantial calculation of the angle between two succeeding nodes on the path so that, for example, the car would not be seen to be following a zigzagging path along the track. The car also needs to travel on top of the track and any zigzagging along the z-axis would also be unacceptable. Moreover, since this is a racing game, determining the path has to be fast.

The Unity platform provides built-in components that can satisfy these requirements of a smooth path along the track and its fast determination. Instead of representing the racetrack as a set of nodes, through which the race car searches for a smooth path, the Unity platform supports the developer in defining the waypoints at key positions along the 3D racetrack so that the race car can follow that path of waypoints. This path of waypoints is indicated with the (red) spheres shown in Figure 1. This substantially reduces the search space and effort. Unity has a default gravity function to ensure the car would travel on the road and that the car does not zigzag along the z-axis. It also provides functions that support vector calculations between two nodes along a path so that the race car can avoid sharp turns on its path. With the waypoint system and the built-in gravity and vector calculation functions, the development effort involved in implementing the race cars that could mimic human driving and traverse the racetrack smoothly was substantially minimized.

Other features of Unity also support the development effort. For controlling the car, the system of conditional monitoring was adopted instead of developing the nonlinear mathematical functions. In terms of collision prevention, the trigger detection technique was used instead of ray-casting for artificial environment perception, because the former is more effective than the latter in that the technique of ray-casting only supports artificial environment perception in one single direction per casted ray. By comparison, the trigger detection technique allows for simultaneous perception in all directions surrounding the car. The Unity engine has built-in high-level abstractions for trigger detection, which also reduced implementation efforts.

7. Conclusion and Future Work

Racer was tested by over twenty users, who all reported that they thoroughly enjoyed the game. It was observed that there was approximately equal number of players who were able to win the race against the game-controlled cars. This suggests that not only did the game provide an entertaining gaming experience, it also provided a reasonably engaging and challenging gameplay.

In general, it can be concluded that the Unity platform supported efficient development of the race car game. The Unity platform supports implementing the race car's search for a path on the racetrack with its components of the waypoint system, the physics engine, and vector calculation functions, all of which are not available if the implementation was done using traditional AI search techniques. With these Unity components, the developer was able to implement the race car's search for a path on the track with less effort and more efficiently, and the developed race car can successfully mimic human driving behavior.

Future work would involve incorporating other path finding techniques into the existing game system so that it can be more exciting. The waypoint system can be used to support a different implementation of the search for the race car's path such that it is more dynamically determined based on its current position. This would make it more difficult for the human players to predict the behavior of the game-controlled cars. This unpredictable behavior would mean a more challenging gameplay for the user. Furthermore, while this game was designed for entertainment, the software can be extended to become a driver simulator for educational purposes. By modifying some of the parameter values of the car, such as the braking coefficients and torque values, and by modifying the game environment to resemble that of a municipal road system, *Racer* can be adapted to become a simulation software for learner-level drivers or for familiarizing drivers with the roadway layouts and traffic laws of foreign or unknown cities.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The first author wishes to acknowledge the generous support of the Graduate Studies Scholarship of the Faculty of Graduate Studies and Research, University of Regina, Regina, SK, Canada.

References

[1] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*, vol. 1, MIT Press, 2004.

[2] Y. Liu, T. Alexandrova, and T. Nakajima, "Gamifying Intelligent Environments," http://www.dcl.cs.waseda.ac.jp/~yefeng/yefeng/pubs/2011/ubimui11_yefeng.pdf.

[3] L. C. Wood and T. Reiners, "Gamification in logistics and supply chain education: extending active learning," in *Proceedings of the IADIS International Conference on Internet Technologies & Society (ITS '12)*, P. Kommers, T. Issa, and P. Isaias, Eds., pp. 101–108, IADIS Press, Perth, Australia, November 2012.

[4] M. Prensky, "Digital natives, digital immigrants part 1," *On the Horizon*, vol. 9, no. 5, pp. 1–6, 2001.

[5] G. K. Akilli, "Games and simulations: a new approach in education?" in *Games and Simulations in Online Learning: Research and Development Frameworks*, D. Gibson, C. Aldrich, and M. Prensky, Eds., pp. 1–3, Information Science Publishing, London, UK, 2007.

[6] A. L. Samuel, "Some studies in machine learning using the game of checkers. II—recent progress," *IBM Journal of Research and Development*, vol. 11, no. 6, pp. 601–617, 1967.

[7] D. A. Waterman, "Generalization learning techniques for automating the learning of heuristics," *Artificial Intelligence*, vol. 1, no. 1-2, pp. 121–170, 1970.

[8] Counter-Strike: Source, http://www.valvesoftware.com/games/css.html.

[9] J. DeNero and D. Klein, "Teaching Introductory Artificial Intelligence with Pac-Man," http://www.aaai.org/ocs/index.php/EAAI/EAAI10/paper/viewFile/1954/2331.

[10] A. Khoo and R. Zubek, "Applying inexpensive AI techniques to computer games," *IEEE Intelligent Systems*, vol. 17, no. 4, pp. 48–53, 2002.

[11] V. K. Tatai and R. R. Gudwin, "Using a semiotics-inspired tool for the control of intelligent opponents in computer games," in *Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pp. 647–652, IEEE, Boston, Mass, USA, September-October 2003.

[12] Microsoft Developer Network, http://msdn.microsoft.com/en-US/.

[13] Unity—Game Engine, http://unity3d.com.

[14] MonoDevelop, http://monodevelop.com.