Implementation and Comparison the Dynamic Pathfinding Algorithm and Two Modified A* Pathfinding Algorithms in a Car Racing Game

Jung-Ying Wang and Yong-Bin Lin

***Abstract*—For a car racing game, the most common artificial intelligence is waypoint navigation by carefully placing waypoints in the game environment to move the game-controlled characters between each waypoint. The major drawback of this method is that these waypoints need to be manually setup. Meanwhile, these waypoints will depend upon the speedway track. In addition, the number of waypoints and the location of waypoints are also different due to human factors. In order to overcome these problems, we propose two modified A* algorithms to effectively solve the pathfinding problem in a static obstacles racing game. The first modified A* algorithm uses a line-of-sight algorithm to reduce the waypoints found by the original A* algorithm. For the speedway of F1 in Turkey, the waypoints reduced from 985 points to 28 points, over 97% waypoints could be removed. The second modified A* algorithm improves the performance of original A* algorithm by heuristically considering the truth that the game-controlled car should steer itself towards. That is to say, we only need to check three waypoints in front of the car, instead of checking four waypoints (up, down, left and right) in the original A* algorithm. Finally, a more general dynamic pathfinding algorithm which can solve the random obstacles avoidance problem in a racing game is also proposed.**

***Index Terms*—Artificial intelligence, A* algorithm, pathfinding, racing game.**

I. Introduction

Many different types of pathfinding problems exist. Unfortunately, no one solution is appropriate to every type of pathfinding problem. The solution depends on the specifics of the pathfinding requirements for any given game. For most racing game, the artificial intelligence (AI) for opponent characters is needed to find there path [1-9]. In our paper, we focus on the car racing game, which can be seen as a kind of pathfinding problems.

In a car racing game, pathfinding is one of the most important problems. Poor pathfinding can make game characters seem very headless and artificial. Handling the problem of pathfinding effectively can go a long way toward making a game more enjoyable and immersive for the player. The A* algorithm provides an effective solution to the problem of pathfinding and it also be one of the most popular algorithm used for the game's development [10]. Assuming a path exists between the starting point and the ending point; then the A* algorithm guarantees to find the best path.

Although the A* algorithm is efficient, it still can consume considerable CPU cycles, especially if you want to simulate pathfinding for a large number of game characters. The chief shortcoming of the A* algorithm in a racing game is that it can not solve the problem of random dynamics obstacles avoidance.

In this paper, we will first study the A* algorithm in a car racing game, and then proposes two modified A* algorithm to do pathfinding. After that, we propose a more general dynamic pathfinding algorithm to solve the problem of random dynamics obstacles avoidance. All the three algorithms are able to find the path for a car racing game and can save the most import resource in game, CPU cycles.

II. Materials And Methods

For a car racing game, the most common artificial intelligence is waypoint navigation by carefully placing points (nodes) in the game environment to move the game-controlled characters between each point [3]. The major drawback of this method is that these waypoints need to be manually setup, and it is a time consuming work. Meanwhile, these waypoints will depend upon the speedway track; different speedway track requires different configuration waypoints. In addition, the number of waypoints and the location of waypoints are also different due to human factors. In order to overcome these problems, we propose two modified A* algorithms to solve them. Finally, a more general dynamic pathfinding algorithm which can solve the random obstacles avoidance problem in a racing game is also proposed. In our paper, we will use the A* algorithm [8], [10-13] to find the shortest path while avoiding the obstacles. The A* algorithm uses path scoring to determine the best path from the starting node to the destination node. To actually score each node, A* basically adds together two components. First, it looks at the cost to move from the starting node to any given node. Next, it looks at the cost to move from the given node to the destination node.

Equation (1) shows the equation used for scoring any given node. This equation computes each node's score by adding the cost of getting there from the starting location to the heuristic value, which is an estimate of the cost of getting from the given node to the final destination.

Manuscript received June 6, 2012; revised July 7, 2012. This work was supported in part by the National Science Council of Taiwan via the grant NSC- 100-2221-E-262-026-.

J.-Y. Wang is with the Department of Multimedia and Game Science, Lunghwa University of Science and Technology, Taoyuan, Taiwan (e-mail: wyy@mail.lhu.edu.tw).

Y.-B. Lin is with the Department of Electronic Engineering, Lunghwa University of Science and Technology, Taoyuan, Taiwan (e-mail: G972321024@ms.lhu.edu.tw).

$$f(n) = g(n) + h(n) \qquad (1)$$

where $g(n)$ is the total distance it has taken to get from the starting position to the current location. $h(n)$ is the estimated distance from the current position to the goal destination. A heuristic function is used to create this estimate on how far away it will take to reach the goal state. $f(n)$ is the sum of $g(n)$ and $h(n)$. This is the current estimated shortest path. The pseudo code of A* algorithm is shown as Fig. 1.

```
add START to OPEN list
  while OPEN not empty
    get node n from OPEN that has the lowest $f(n)$
    if $n$ is GOAL then return path
    move $n$ to CLOSED
for each $n'$ = CanMove($n$, direction)
$g(n') = g(n) + 1$
calculate $h(n')$
if $n'$ in OPEN list and new $n'$ is not better, continue
if $n'$ in CLOSED list and new $n'$ is not better, continue
remove any $n'$ from OPEN and CLOSED
add $n$ as $n'$s parent
add $n'$ to OPEN
end for
  end while
if we get to here, then there is No Solution
```

Fig. 1. Pseudo code of A* algorithm.

In this paper, we select the real speedway of Formula one (F1) as our game motordrome, to simulate and analyze our study. Fig. 2 is the speedway image of F1 in grand prix of Turkey which downloads from the official F1 website. The image isometric scales up to the size of 1280*782 pixels. Our car racing game is implemented by Microsoft XNA Game Studio. The XNA platform is a programming environment that allows users to create games for Windows Phone, the Xbox 360 console, and Windows-based computers.

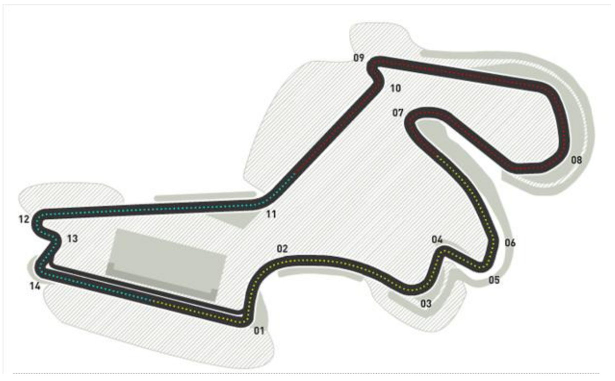

Fig. 2. The image of speedway of F1 in grand prix of Turkey.

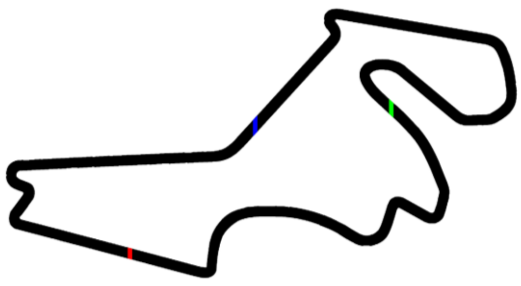

Fig. 3. The collision detection map of F1 in grand prix of Turkey.

A. Implementing the A* Algorithm

Because the first step in pathfinding is to define the search area, we need some way to represent the game world in a manner that allows the search algorithm to search for and find the best path. In our game, we use the trick of color collision to do the collision detection [14]. Using this trick you first need to make a collision detection map, as shown in Fig. 3. This can be done easily by using any image processing software (e.g., Photoshop), and then changes the track to the color which users want to set it as the collision detection color (e.g., block color). Everything else in the collision detection map, where the cars are not allowed to drive, you just need to paint them to white (or any color just do not use the track color black). Ultimately, the game world is simplified by placing 1280*782 nodes, throughout the game environment. White color nodes represent obstacles and other colors nodes represent the nodes which can be passed. After that, we have divided our search area into a 1280*782 square grid. This particular method reduces our search area to a simple two dimensional array. Each item in the array represents one of the squares on the grid, and its status is recorded as passable or un-passable. The path is found by figuring out which squares we should take to get from node A to node B. Once the path is found, the game-controlled car moves from the center of one square to the center of the next until the target is reached.

In our implementation, the image size of car is 18*12 pixels and moving speed is two pixels per time frame. The racing game uses the default frame rate setting in XNA (60 frame / sec). Therefore, the maximum linear velocity of the car is equal to 120 pixels per second.

B. Modified A* Algorithm: Reducing Waypoints by a Line-of-Sight Algorithm

As you known, the more nodes placed in the game world, the slower the pathfinding process. If we simplify the search area by using fewer nodes, the pathfinding work will save many CPU cycles.

In our study, we first used the original A* pathfinding algorithm to find the best path between the start waypoint and the end waypoint, as illustrated in above section. There are 985 waypoints found in the speedway of F1 in Turkey. After that, we refine above found path by considering a line-of-sight algorithm to further reduce the number of nodes.

To implement this, the mathematical formula of the Pythagoras Theorem for triangles is used. It first calculates the way length (distance) between the two different nodes. If there is no obstacle across the way, then no collision occurs, as shown in Fig. 4 the waypoints (1, 2) and waypoints (1, 3).

Because the waypoint 2 is between the waypoint 1 and 3 and there is no collision occurs, we can remove the waypoint 2 using the shortcut 1-> 3 instead of the path 1-> 2-> 3. This line-of-sight process will continue until the end waypoint reached. For the speedway of F1 in Turkey, the waypoints reduce from the number of 985 to 28. That is to say, over 97% waypoints could be removed.

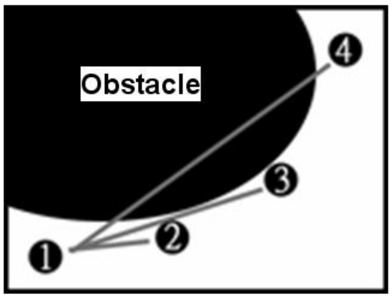

Fig. 4. Modified A* algorithm by a line-of-sight algorithm.

C. *The Second Modified A* Algorithm: Only Searching the Forward Direction*

We further improve the performance of A* algorithm by heuristically considering the truth that the game-controlled car should steer itself towards. Therefore, in this modified A* algorithm, we only need to search three waypoints in front of the car, instead of searching four nodes (up, down, left and right) in the previous study. The three waypoints are front right, directly ahead and front left of the car, as shown in Fig. 5.

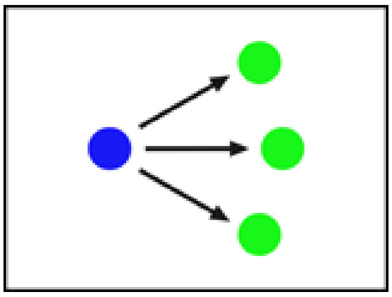

Fig. 5. Modified A* algorithm: only searching the forward direction.

D. *Dynamic Pathfinding Algorithm for Random Obstacles Avoidance*

In order to generalize the pathfinding algorithm in a racing game solves the dynamic obstacles avoidance problem. We have recently proposed a dynamic pathfinding method in [9]. Two collision detection points are put in front of the car's right side and left side, as shown in Fig. 6. Where the variable x is the half width of car, that is 6 pixels, and the collision detection distance y is an adjusted variable indicated the distance from the car center to the center of the two collision detection points.

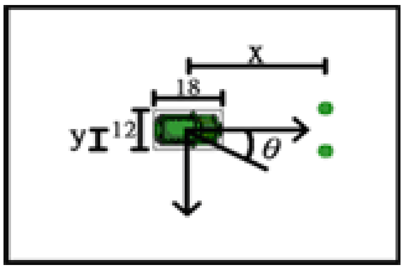

Fig. 6. Collision detection points of car.

How do we actually perform the collision detection in this algorithm? It is easy. We just need to put several color detection points in front of (or around) the moving car. Anytime, if the position of the car's color detection point its color is the same as the track color (black), then no collision occurs. Otherwise, if the position of the color detection point its color is white (or not the track color), indicating the car is leaving the track, which means the car needs to turn a direction to keep the car inside the track. In our game's implementation, we will calculate the car's position in advance. If in the next time frame the collision is detected. We just simply turn a default setting radian for the car to avoid collision. In other words, when the detection point of left front touches the edge of track, the car will turn in a clockwise direction. Otherwise, the car will turn in a counterclockwise direction, if the detection point of right front touches the edge of track. In order to make the game-controlled car look more natural and smooth, the car's rotation speed is set to 0.25 radians (14.3 degrees).

In very rare cases, the car may get stuck in consecutive curves. Therefore when the two collision detection points are activated at the same time, the detection points were changed to the 45 degree points in front of the right side and left side of the car, as shown in Fig. 7. That is, the collision detection points, in Fig. 7, will change from the inside two points to the outside two points, in order to have more robust control of the car. As seen in Fig. 7, using this trick it can avoid the car get stuck by reducing the two collision points to one. Then, we can make the corrected judgment to control the car to turn right or left.

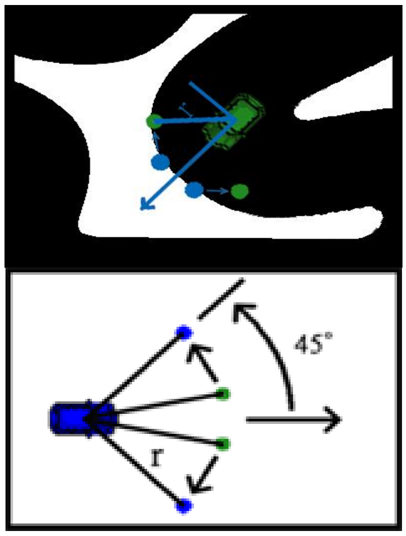

Fig. 7. New collision detection points when Figure 6's two collision detection points are activated at the same time.

E. *Experimental Results and Discussion*

The major drawback of waypoint navigation method is that these waypoints need to be manually setup. Meanwhile, these waypoints will depend upon the speedway track. In addition, the number of waypoints and the location of waypoints are also different due to human factors. Therefore, we could overcome the above drawbacks by using our modified A* algorithm; and both of them only need to find the path once before running the game. The disadvantage is that the finding path will be exactly the same. That is, the game-controlled car is always around the racing track using the same path. The comparison of simulated results for F1 in grand prix of Turkey is summarized in Table I.

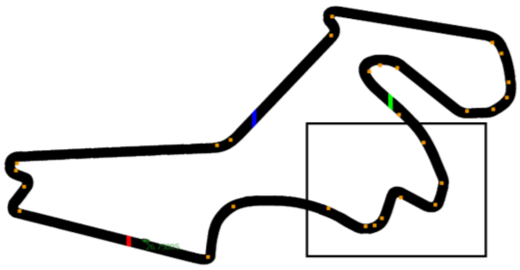

Fig. 8. The pathfinding result by using the first modified A* algorithm.

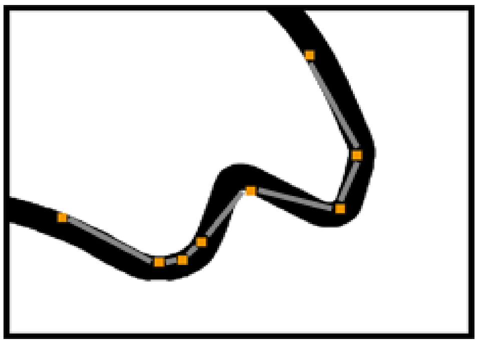

Fig. 9. More waypoints needed in sharp curve.

F. Modified A* Algorithm: Reducing Waypoints by a Line-of-Sight Algorithm

The pathfinding result by using the first modified A* algorithm, combining a line-of-sight algorithm, to reduce the waypoints found by the original A* algorithm is shown in Fig. 8 and Fig. 9. Figure 8 shows that the 28 waypoints created by this algorithm to control the car. For the speedway of F1 in Turkey, the waypoints reduced from 985 points to 28 points, over 97% waypoints could be removed.

In other words, we can remove the nodes inside the straight line part of racing track to saving the CPU cycles. Comparatively speaking, if the racing track is winding, more waypoints are needed, as shown in Fig. 9. In addition, because we use the line-of-sight concept to further reduce the waypoints found from original A* algorithm, the car's moving looks a little bit stiff in the successive curve.

G. Modified A* Algorithm by Searching the Forward Direction Only

The advantages of the second modified A * algorithm is that we can get the shortest lap time. This is because in each time step checking nodes from four nodes of the original A* algorithm reduce to three nodes. The disadvantage is that the car may swing when the car goes around the consecutive sharp bend.

H. Dynamic Pathfinding Algorithm for Random Obstacles Avoidance

The advantages of dynamic pathfinding algorithm for random obstacles avoidance is that it can do the pathfinding in real time and it looks more natural then the static path of A* finding. Because the finding path is not the shortest path, therefore the lap time will a little bit higher then above modified A* algorithms.

TABLE I: COMPARISON OF SIMULATED RESULTS FOR F1 TURKEY.

	The first modified A* algorithm	The second modified A* algorithm	Dynamic pathfinding algorithm
Path	static path	static path	Not static path, dynamic changing
Lap Time	26.75 (second)	26.48 (second)	27.55 (second)
Characteristic	looking a little bit stiff in the successive curve	may swing, the shortest lap time	random obstacles avoidance, more natural

III. CONCLUSION

The most common artificial intelligence in a racing game is waypoint navigation by carefully placing waypoints (nodes) in the game environment to move the game-controlled characters between each point. This is a very time consuming and CPU intensive problem. Using the A* algorithm can effectively solve the pathfinding problem in a static racing game environment; therefore, we present two modified A* algorithm instead of putting waypoints by hand and minimum the lap time. Finally, we propose a more general dynamic algorithm which can solve the random obstacles avoidance problem in a racing game.

REFERENCES

[1] L. delaOssa, J. A. Gamez, and V. Lopez "Improvement of a car racing controller by means of Ant Colony Optimization algorithms," *IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 365-371.

[2] S. Fujii, T. Nakashima, and H. Ishibuchi "A study on constructing fuzzy systems for high-level decision making in a car racing game," *IEEE Congress on Evolutionary Computation*, 2008, pp. 3626-3633.

[3] T. Nakashima and S. Fujii, "Designing high-level decision making systems based on fuzzy if-then rules for a point-to-point car racing game," *Soft Computing*, May. 2009, pp. 1-12. DOI: 10. 1007/s00500 -009-0448-7.

[4] J. Togelius, P. Burrow, and S. M. Lucas, "Multi-population competitive co-evolution of car racing controllers," *IEEE Congress on Evolutionary Computation* 2007, pp. 4043-4050.

[5] J. Togelius and S. M. Lucas, "Evolving controllers for simulated car racing," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2005.

[6] J. Togelius and S. M. Lucas, "Evolving robust and specialized car racing skills," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2006.

[7] A. Agapitos, J. Togelius, and S. M. Lucas, "Evolving controllers for simulated car racing using object oriented genetic programming," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 1543–1550.

[8] Y. C. Hui, E. C. Prakash, and N. S. Chaudhari, "Game AI: artificial intelligence for 3D path finding," *TENCON 2004. 2004 IEEE Region 10 Conference*, pp. 21-24, Nov, 2004.

[9] J. –Y. Wang and Y.-B. Lin, "An Effective Method of Pathfinding in a Car Racing Game," *The 2nd International Conference on Computer and Automation Engineering ICCAE 2010*, pp. 26-28, February, 2010.

[10] D. M. Bourg and G. Seemann, *AI for Game Developers*, O'Reilly, July 2004.

[11] S. Rabin, *AI Game Programming Wisdom 4, Charles River Media,* February 2008.

[12] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company, Wellsboro, PA, 1980, pp. 366-381.

[13] D. M. Bourg and G. Seemann, *AI for Game Developers*, Glenn Seemann Publisher, 1st ed., O'Reilly Media, 2004.

[14] B. Hamboeck, “XNA or game development for everyone – restructuring the game part2,” *Net Developer’s Journal*, vol. 6, pp. 14-29, 2008.

Dr. Jung-Ying Wang received his PhD from National Taiwan University of Science and Technology in computer science and information engineering, MSc from National Taiwan University in computer science and information engineering and BSc from National Taiwan University of Science and Technology, Taiwan. His main areas of research are game AI, bioinformatics, machine learning, intelligent systems on the web, game programming and protein structure prediction. He is currently an associate professor in the multimedia and game science department, Lunghwa University of Science and Technology, Taiwan.