Improving pathfinding cohesion in RTS games

Enrique Wicks Rivas
Dep. de Ciência da Computação
Universidade Federal da Bahia
Salvador, Brasil
enriquewr@hotmail.com

Rodrigo Souza
Dep. de Ciência da Computação
Universidade Federal da Bahia
Salvador, Brasil
rodrigorgs@ufba.br

***Abstract*—*Pathfinding* algorithms are essential to studies on graph navigation. In many video games such navigation is fundamental to control the behavior of game entities, and this is especially evident in games following the classic Real-Time Strategy (RTS) formula. As the genre matured, their *pathfinding* algorithms became more sophisticated, but modern implementations can still lead to unintended entity behaviors that cause frustration, such as a group of entities breaking military formation when given the same order to move to a point. This article details the initial results and methodology of an adjustment to the standard algorithm that seeks to improve entity cohesion during movement without significantly increasing the overall path length taken by the group or slowing the speed of their advance. To measure its effectiveness we have constructed three scenarios that stress the flaw in the standard algorithm and have run the new algorithm with different parameters. As a result, we found cases where the new algorithm provided significantly greater cohesion than the standard method, with minimal increase in path length.**

***Index Terms*—pathfinder, RTS, algorithm, movement**

I. Introduction

Real-Time Strategy games in their classic form are military simulators where players both build the army infrastructure that train soldiers and collect resources as well as control the army's movement and tactics [1]. When the game's *pathfinder* is well implemented, the army follows the player's command with precision and efficiency in a responsive manner that is critical to competitive RTSs, and while that can appear as a solved problem when looking at how polished modern implementations are, there are still edge cases that disrupt player agency. One of such cases is when commanding a group of entities around a large obstacle and the group splits because each entity calculated a different optimal path. While this behavior does result in the shortest time for the whole group to reach the destination, it also means that the army moves in uncoordinated fashion, making it a weaker force in numbers if an engagement with the enemy army occurs.

Our approach to fixing this behavior was to compute paths for each individual entity in a way that each entity's path influence the paths of the other entities. The algorithm was tested on three different scenarios, with different values for its parameters. The results were compared with those of the standard algorithm. In doing so we have noticed a significant increase on the cohesion of the entity group without significantly raising the average path distance.

II. Background

We use the term *pathfinder* to encompass the algorithms by which RTS games calculate the path to be navigated by the entities on the map. While many implementations exist, each with their own drawbacks and benefits, the usual solution is to use the A* search algorithm [2] to find the shortest path from the entity's position to the destination the player wants to reach. While using A* based *pathfinders* the developers can either calculate a path for each entity or treat the whole group as a single entity.

If the developer chooses to calculate each entity's A* path individually they will reach the shortest sum of paths to a given group, but the cohesion during movement isn't guaranteed as entities can split formation to circumvent large obstacles; conversely, if the developers choose to treat the group as a single entity in the *pathfinder* they will achieve great cohesion, but the sum of the paths increases as the whole group has to move around obstacles even if they would have little effect on group cohesion. Our research aims to carve a bridge between the benefits of the two methods, using individual calculations to find shorter paths, but finding a way to keep communication between the members of the group to increase cohesion.

The inspiration for our solution is the Ant Colony algorithm [3]. This algorithm employs the use of markers on the nodes that makes them more favorable than nodes with less markers, simulating nature's use of pheromones to guide ant navigation.

III. Solution

Our idea is to use the marker system to make nodes that multiple entities have chosen as part of their path more favorable to the the other entities, even if they wouldn't be their optimal choice. That way each entity can search for its best possible path but still take into consideration the path chosen by its fellow group members. If the difference in distance isn't too great, the entity should default to keep the group united.

The solution we developed is very similar to the original A* algorithm [4] [5], fused with the path marking found in the Ant Colony algorithm. After using our altered A* algorithm (Algorithm 1) to find an entity's path we register a marker on each node of the found path. These markers act as a multiplier on the node distance used in the ranking of the open list, making each node that is used in an entity's path more desirable to the next entity. The base of the algorithm

is shown on Algorithm 2; it serves to iterate over our entity group calling the altered A* algorithm for each one, delivering the found path to the entity and updating the markers on our nodes.

```
Input : The entity's node and the target position
Output: The path for the entity
Create an openList and a closedList;
Insert the entity's node into openList with its
 totalValue set to 0;
while NotEmpty(openList) do
    currentNode ← node with lowest totalValue from
     openList;
    if currentNode == target position then
        Return currentNode and its parents recursively
         as the output path;
    end
    neighbours ←
     GetNeighbours(currentNode);
    foreach auxNode in neighbours do
        realDistance ←
         GetRealValue(currentNode) +
         Distance(currentNode, auxNode);
        heuristicDistance ← Distance(target
         position, auxNode);
        totalDistance ← (realDistance +
         heuristicDistance) *
         GetMarker(auxNode);
        if Neither openList nor closedList have an
         instance of auxNode with totalValue <
         totalDistance then
            SetParent(auxNode, currentNode);
            SetRealValue(auxNode,
             realDistance);
            SetTotalValue(auxNode,
             totalDistance);
            Insert auxNode into openList;
        end
    end
    Insert currentNode into closedList;
end
```

Algorithm 1: A* using marker values

The parameters that the algorithm take are the *entity group* we are commanding, the *target position* we wish to reach and the *multiplier marker value*. The ordering of the entities in the entity group influences the result, as each time we run the altered A* algorithm we update the marker values, influencing the next entity. To have deterministic results we ordered the group by how close each entity is to the target position, measuring in a straight line. Other ordering methods might change the results.

```
Input : A group of entities, a target position and a
        value to update the marker (between 1 and 0)
Output: Nothing
foreach entity in the group of entities do
    path ← AStarWithMarkers(entity, target
     position);
    SendPath(entity, path);
    foreach node in path do
        MultiplyMarker(node, value to update
         the marker);
    end
end
```

Algorithm 2: Base of the proposed pathfinder

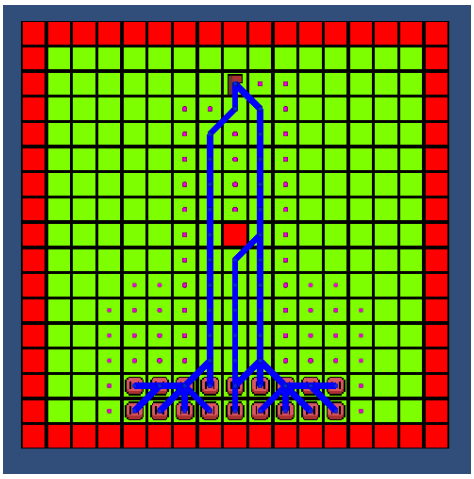

Fig. 1. Lone Tree with 0.95 marker

IV. Evaluation

To evaluate the new algorithm we have developed two objective criteria: group cohesion and average path length.

For each node, we compute the number of nearby paths, i.e., paths that include the node or one of its neighbors. Group cohesion is defined as the average number of nearby paths over all nodes that are themselves part of a path.

Average path length is computed by summing the length of all paths and dividing it by the number of entities. The shorter the average, the shorter it should take for the entity group to reach the target destination.

We tested five values for our marker: 1.0, 0.99, 0.95, 0.90, 0.75. As the marker acts as a multiplier, the lower the marker value, the more it impacts the pathfinder. The value 1.0 has no impact at all, making it our control value, as its results are the same as the A* algorithm.

With our metrics set, we came up with three different scenarios that stress the flaws our algorithm is trying to fix, which we named Lone Tree, Chinese Wall, and Boulder. The maps have red blocks representing obstacles, blue lines representing the path each entity took, and small pink dots representing nodes that the algorithm visited.

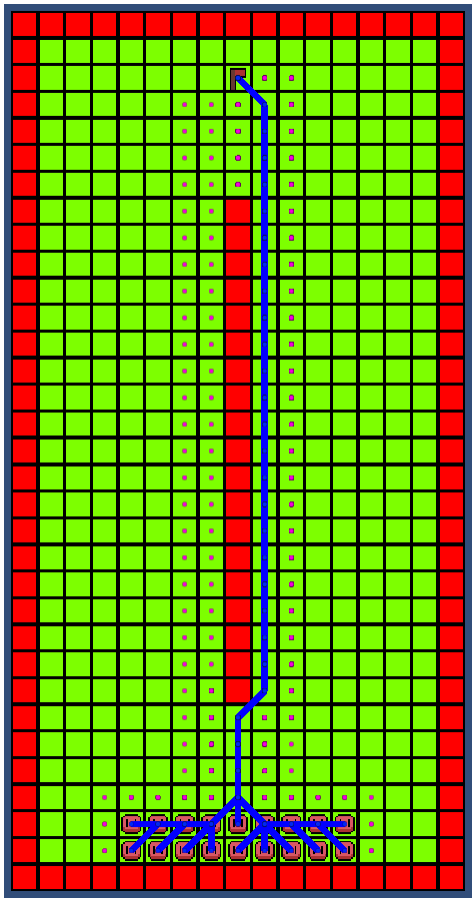

Fig. 2. Chinese Wall with 0.95 marker

Lone Tree. The first scenario contains a single blocking node to simulate a very small obstacle; the entity group should split around the obstacle and avoid clumping up needlessly.

Chinese Wall. The second scenario consists of a long obstacle line that makes clear that splitting the group at the first node of the obstacle completely separates the group, heavily decreasing its cohesion.

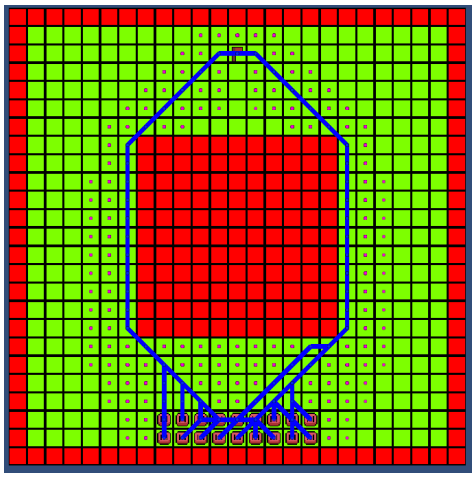

Fig. 3. Boulder with 0.95 marker

Boulder. The third one is the same as the second one but with a much wider obstacle spreading the direction decision on more nodes than one.

Table I: Results

Figure	*Map*	*Marker*	*Avg. Path*	*Cohesion*
*	LT	1.0	13.51	6.90
*	LT	0.99	13.51	8.03
1	LT	0.95	13.86	10.43
5	LT	0.90	14.88	12.73
*	LT	0.75	14.88	12.73
*	CW	1.0	29.51	6.07
*	CW	0.99	29.55	8.04
2	CW	0.95	30.88	14.56
*	CW	0.90	30.88	14.56
*	CW	0.75	30.88	14.56
*	B	1.0	25.16	8.50
*	B	0.99	25.16	8.50
3	B	0.95	25.86	9.10
4	B	0.90	27.89	14.15
*	B	0.75	27.89	14.15

*Fig. not included.

A. *Results*

The results can be viewed on Table I. The first column shows which Figure in the article the result refers to (if it was included), followed by the initials of the map used (Lonely Tree, Chinese Wall, Boulder), then we have the value of the marker used, the average path length and finally the cohesion.

The most promising result we found was the 0.95 marker configuration, as it splits the group on small obstacles (Fig. 1), chooses a side on the Chinese Wall (Fig. 2), but unfortunately separates the entities on the Boulder (Fig. 3). On the Chinese Wall the average path length increased by 4.6% while cohesion went up by almost 140%.

Using a 0.99 marker groups the entities more but doesn't avoid mistakes on the Chinese Wall. Its behavior didn't significantly deviate from the control (marker value 1.0).

With a 0.90 marker you finally get to see grouping in the Boulder (Fig. 4), but you stop having fluidity on the Lone Tree as the whole group is behaving as a block going around small obstacles (Fig. 5) with an average path length increase of 10%.

The 0.75 marker scored the same values of the 0.90 marker (Table I) indicating there might be diminishing results on how low the marker can go.

B. *Discussion*

The 0.95 marker was the one that kept most of the average path length, which is fundamental to competitive RTS games, while keeping group cohesion in the Chinese Wall scenario. Anything further (0.90 and 0.75) started to affect the path lengths too much.

There might be a middle ground that can accept one of the more radical markers. If the player had agency over whether they want their group to stay more cohesive or prioritize speed,

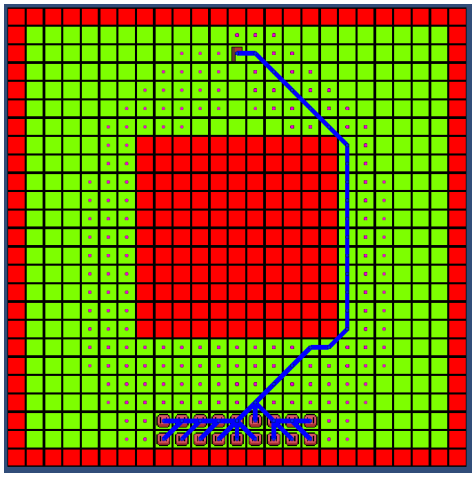

Fig. 4. Boulder with 0.90 marker

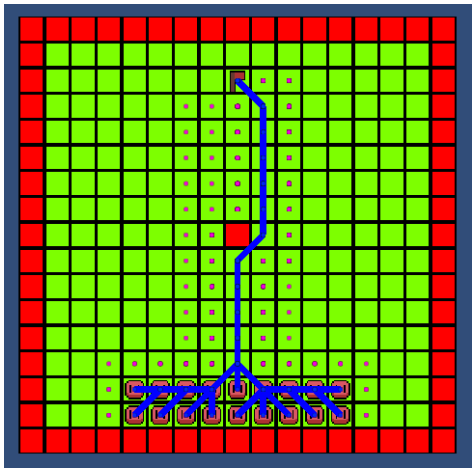

Fig. 5. Lone Tree with 0.90 marker

the game could either use a 0.90 marker or a 1.0 marker to run pure A* and let each entity path be independent.

Having that as an explicit option would suffocate the player with more micromanagement; luckily there is a natural division of movements in standard RTS games: usually a player has the option to give an *attack move* command, which orders the entity group to move towards a position but engage in combat with any enemies it sees in their way and a pure *move* command that makes entities move ignoring enemies on their way. If we decide that *attack move* commands use a 0.90 marker, the group will keep cohesion on engagements with enemy troops. Keeping the 1.0 marker value to *move* commands the player still has the best average path lengths to move in situations where speed is more important than cohesion, such as moving in reinforcements or retreating. That however has the risk of becoming an obscure game mechanic that the player won't understand or be able to replicate during competitive gameplay which itself could lead to frustration.

V. Conclusion

This work presents an algorithm that aims to increase entity cohesiveness so they don't separate in graph navigation without increasing the average path length significantly. While some promising results were found, none could cover every issue. Still, a developer looking to increase their game's responsiveness to player commands can use the algorithm to polish the edge cases that induce frustration, specially common in competitive RTS games.

As a future experiment it would be interesting to change the map structure so it better simulates modern navigation meshes, perhaps even testing results inside Starcraft 2's engine, and see if entity movement and engagement are significantly changed.

References

[1] Real-time strategy (rts). [Online]. Available: https://www.techopedia.com/definition/1923/real-time-strategy-rts

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] M. G. Sahab, V. V. Toropov, and A. H. Gandomi, "2 - a review on traditional and modern structural optimization: Problems and techniques," in *Metaheuristic Applications in Structures and Infrastructures*, A. H. Gandomi, X.-S. Yang, S. Talatahari, and A. H. Alavi, Eds. Oxford: Elsevier, 2013, pp. 25 – 47. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123983640000024

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[5] I. Millington, *AI for Games*. Taylor & Francis, a CRC title, part of the Taylor & Francis imprint, a member of the Taylor & Francis Group, the academic division of T&F Informa, plc, 2019. [Online]. Available: https://books.google.com.br/books?id=HMKjuwEACAAJ