Pathfinding and Movement System for a Real Time Strategy Game

Ivan Antešić
Faculty of Computer and Information Science
University of Ljubljana
Email: ia6382@student.uni-lj.si

***Abstract*—Moving units across the map in real time strategy (RTS) games needs to be done quickly while also looking natural to the player. Depending on the map size and number of units this can pose a problem. Standard pathfinding algorithms can become insufficient and tricks need to be used to avoid or hide their limitations. In this paper several known approaches are presented and tested in a prototype RTS game. Results offer insights to gameplay mechanics of some popular RTS games of late 1990s and early 2000s.**

I. Introduction

Moving a unit around the map in games is achieved with a pathfinding and movement system [1]. The former finds a traversable path to the goal and is usually the focus of research, while the latter moves agents along the path in a desired way, avoiding possible collisions. This has to be achieved quickly especially in real time strategy (RTS) games where the players control many units moving on a large (possibly random and dynamically changing) map. The units also have to move in a way that makes sense to the player. This can pose a problem that standard pathfinding algorithms alone cannot solve.

Little is known for sure how do commercial games approach this problem. Most companies are not sharing their valuable knowledge and games up to this day struggle with moving large amounts of units in a natural way around the map.

II. Methods

In order to experiment and find possible solutions, a prototype RTS game was implemented in Python with tcod library[1] which emulates a terminal console split into a grid for easier printing. Several approaches based on papers and reverse engineering of games from late 1990s and early 2000s were tested. Following is their general description.

A. Naive A*

The simplest solution is to use A* (with Manhattan distance as heuristic) to find a path for every single agent selected by the player. A path is a stack of waypoints - individual tiles or cells of a grid that makes up our map. A movement system simply moves agents along the path by poping a waypoint and placing the agent on the returned tile.

There are additional details that we need to think of such as selecting an unreachable goal in which case the closest reached tile will become a new goal (the one with lowest h score after it searched the whole map).

If an unwalkable (i.e. obstructed) tile is targeted we need to find a new goal by gradually searching out with breadth first search to find the closest new goal before searching for a path.

Offset of selected units also needs to be calculated so they do not all share the same goal. It takes into account the relative position in selected group but bounds it by defined maximal distance between agents.

B. Shared path A*

Because pathing separately for every single agent is too slow for most cases, next improvement was to select a leader that finds a single path all agents share. The leader is defined as a medoid of selected agents. Agents' paths therefore consist of moving to the centre of group, continuing down on shared path, before branching out to their offset goals.

This speeds up the algorithm but it makes path suboptimal and movement unnatural because agents travel in a line down the shared path.

C. HPA*

The next improvement that probably most games use is introduction of segmentation or hierarchy to pathfinding. Hierarchical pathfinding A* algorithm (HPA*) [2] is used to split the map into clusters that are connected to each other by defined entrance tiles (an example can be seen in figure 2). Entrances inside of one cluster are also connected (if possible), their edge costs calculated with a simple flood-fill algorithm. Thus a high level graph can be produced on which A* can search for high level path.

Now every time the agents paths to the goal it firsts finds a rough, high level path between clusters and then a low level path inside the cluster. Both naive and shared path A* can be used with this enhancement.

This speeds up pathfinding significantly and is also beneficial because we do not needlessly search for a whole path at the beginning every time a move order is given. Often times we will interrupt the moving of an agent with a new order or a collision, therefore a big part of the path will be discarded anyway.

The downside is that once again the path is a little more suboptimal and the agents move less naturally.

[1]https://github.com/libtcod/python-tcod

TABLE I
TIME (IN SECONDS) REQUIRED TO SEARCH THE WHOLE MAP FOR A GROUP OF 7 AGENTS.

naive A*	shared path A*	naive HAP*	shared path HPA*
2.242	0.322	0.204	0.036

D. Collision system

Movement system needs to predict and solve possible collisions that occur while moving. First a collision predictor finds any possible collisions for each agent and categorizes them into *towards, static, glancing* or *rear* type, based on future moves of involved agents [3]. Then a collision solver decides how the agents will react. They might wait, push each other around or replace a part of their path. Reaction depends on agent priority which is a heuristic defined as number of unwalkable tiles surrounding the agent. Without priority agents might endlessly cycle their actions to resolve multiple collisions.

III. RESULTS

For testing we defined a base case of 7 agents on a 100 x 80 tile map that includes a few obstacles and a bridge that acts as a choke point (as seen in figure 1).

A. Search time

We evaluated different approaches based on time it takes to find a path for all agents to an unreachable target (worst case - it needs to search the whole map). Results can be seen in table I.

B. Movement quality

For naive A* search even a small number of agents takes to much time but ignoring that it produces the most natural movement because the agents usually do not share paths and can move concurrently as a group.

With shared path A* the agents usually wait until priority decreases and then join in a long line down the shared path. Some unnecessary movement also happens at the start when agents are travelling to a centre of a group.

Similar behaviour occurs with shared path HPA* with addition of a long line to the goal being more rough and suboptimal.

Best approach seems to be naive HPA*, where the search time is reasonable up to 25 units and while some agents still share a common path due to having same high level waypoints, the effect is not so extreme as in shared HPA*.

In all cases the maximum number of agents was 100 because of the small map size map and implementation details. Much depends also on how many agents we select and order on the same time. Moving two large, opposing groups across a bridge results in blockage if their paths intersect, which is more likely with HPA* and shared path approaches.

IV. DISCUSSION

The more we try to speed up pathfinding, the more suboptimal it becomes. We then have to compensate for a bad pathfinding with a better movement system. The rules for collisions expand trying to cover all possible edge cases and it becomes an impossible task to maintain natural behaviour and prevent collisions.

Turns out that same problems were encountered by developers working on well known games at the time described approaches were in use. They ”solved” these problems by simply designing a game around these limitations. Having our results in mind we can start to see reasons for some of game mechanics that at first seem arbitrary and pointless.

In Stracraft 1 (1998) the issue of giant state machine for solving collisions was removed by allowing certain units to pass right through each other [4]. We can observe the same thing in Age of Empires 2 (1999) and most other titles today. Warcraft 3 (2002) has great movement but it only allows for a maximum of 12 units to be selected, possibly to prevent overloading the complex algorithms. When possible units form in a square formation to minimize the walking in a line effect. In Rome Total War (2004) players control hundreds of units but they are clustered into formations which act as one agent for movement purposes. They also do not avoid static obstacles so for the most part pathfinding is reduced to a straight line across the mostly barren map.

Of course today games also use more advanced approaches. It is known that Planetary Annihilation (2014) uses flow fields [5, Chapter 23.3] to achieve smooth movement of large amounts of individual units. It is very likely that Starcraft 2 (2010) uses some form of flocking algorithms [6]. But all approaches come with limitations - Starcraft 2 can field a lot of units in large flocks but is unable to organize them into sensible military formations.

V. CONCLUSION

Making a pathfinding and movement system look natural while performing good is harder that it seems. A* and simple local collision avoidance proves insufficient despite being generally considered a solution for this problem. Tricks are used to hide the limitations as game mechanics. Pathfinding in RTS games is far away from a solved problem and there is still room for a new algorithms and improvements.

REFERENCES

[1] D. Pottinger. (1999) Coordinated unit movement. Accessed: 02.06.2020. [Online]. Available: https://www.gamasutra.com/view/feature/3313/coordinated_unit_movement.php

[2] A. Botea, M. Müller, and J. Schaeffer, “Near optimal hierarchical pathfinding (hpa*),” *Journal of Game Development*, vol. 1, 01 2004.

[3] C. Foudil, N. DJEDI, C. Sanza, and Y. Duthen, “Path finding and collision avoidance in crowd simulation,” *CIT*, vol. 17, pp. 217–228, 01 2009.

[4] P. Wyatt. (2013) The starcraft path-finding hack. Accessed: 02.06.2020. [Online]. Available: https://www.codeofhonor.com/blog/the-starcraft-path-finding-hack

[5] S. Rabin, *Game AI Pro: Collected Wisdom of Game AI Professionals*. USA: A. K. Peters, Ltd., 2013.

[6] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 25–34, Aug. 1987. [Online]. Available: http://doi.acm.org/10.1145/37402.37406

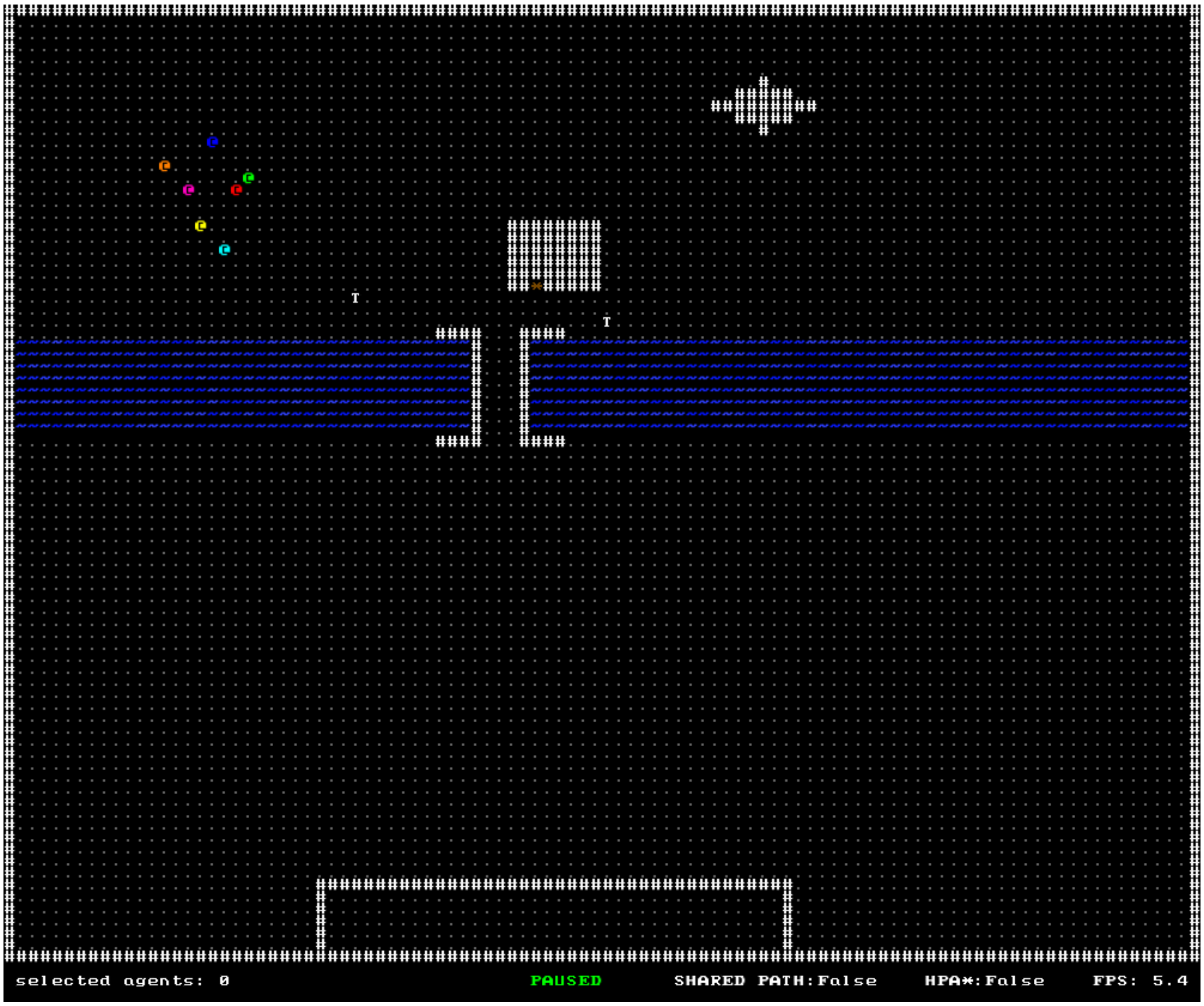
```
selected agents: 0          PAUSED      SHARED PATH:False     HPA*:False     FPS: 5.4
```

Fig. 1. Base case map for testing purposes.

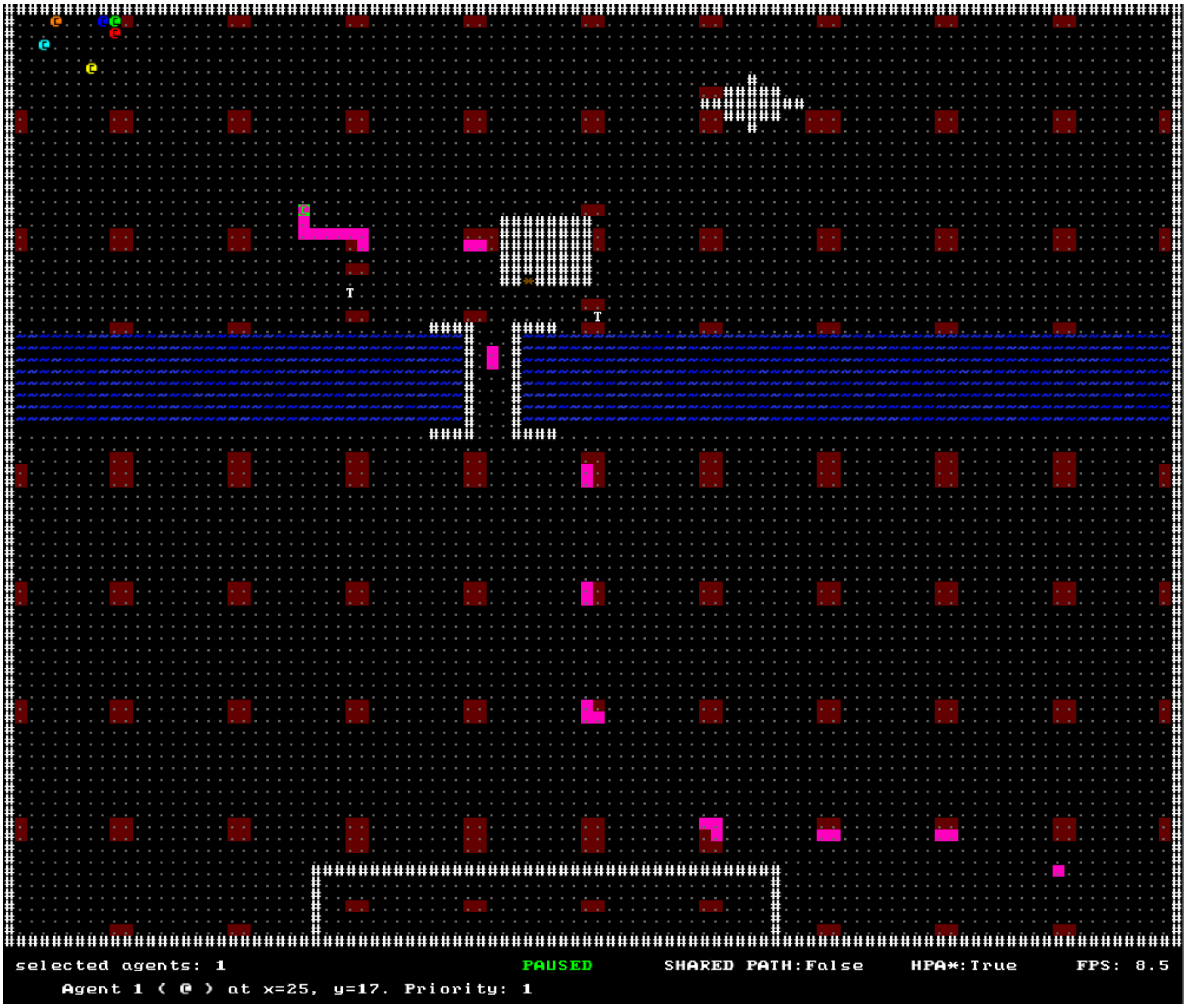
selected agents: 1 PAUSED SHARED PATH:False HPA*:True FPS: 8.5
Agent 1 (@) at x=25, y=17. Priority: 1

Fig. 2. Map is segmented into 10 x 8 clusters, connected by entrances seen as red tiles. An agents path is seen in magenta - a low level path inside the first cluster and then rough waypoints through the rest.