



IKT407 WEB MINING
Pathfinding for ORTS

EXPERIMENT RESULTS

Authors:

Asbjørn BYDAL

Frode NILSEN

November 8, 2007

1 Introduction

This paper describes our work and results from the experiment phase of the project period.

2 Experiment environment (ORTS)

As the project assignment was to create a pathfinder for the Open Real-Time Strategy game engine (ORTS), the obvious choice of experiment environment is ORTS itself. We have used the standard game (game1) as our test bed. This game consists of a 64x64 tile map containing approximately 15 buildings, 3 aircraft and 5 workers, all on the same team. There are also a few scattered mountains and various smaller static obstacles. The command we've used to run the server is `./bin/orts -nfog -game1`. The first argument, `-nfog`, removes the fog-of-war (see section 3) and the second selects game 1, described above.

We have based our client on the default ORTS client. This enables us to use either console-only, 2D or 3D graphics. It does not, however, provide any interaction capabilities, apart from zooming and panning. This means there is (as of now) no way to select units and assign new destinations with your mouse etc. To maintain focus on the pathfinding part, our client simply moves the 5 workers diagonally across the map after finding a path for each one.

3 Experiment results

Even though we had great plans for different kinds of experiments we wanted to do, we found that the implementation of our algorithm into ORTS was going to take longer than we first thought. Our implementation of AStar in ORTS consists of a single game where all the workers are commanded to go to the exact opposite of their starting point. Since all workers are placed randomly around the level at game start, most of the time only half of the workers can actually move to their destination. This is because it often happens that the destination is not walkable and the unit will then not engage into the move routine. The units that do, however, will move according to their AStar path and arrive at destination unless they collide

with other moving units during their move routine. Also a diagonal move where the adjacent squares are not walkable will result in a collision.

We found that to avoid or handle these types of collisions we would have to have some kind of routine for finding new paths during the move routine. We figured that the best way to do this is to set a random wait time for the colliding unit. During this wait time the unit should try to move the path it was originally given. After the wait time is over, if the unit has not been able to continue, it should generate a new path to the destination. This will handle collisions with moving units. When two moving units collide they will both be given a wait time. One unit will wait a shorter period than the other and will generate a new path around the obstructing unit. The other unit will find that the path is cleared during its wait time and continue its original path.

With fog of war turned on we found that the problem is much the same as a collision. Only, the path will never be cleared. The scenario is that a unit has created a path through an area it does not know the layout of (because it is hidden by fog of war). The unit will then move into the unexplored area and find that the path it originally created leads to a collision with a stationary unit. When the collision occurs the unit will generate a new path to the destination and try a new move. This will eventually get the unit where it is supposed to go if there is a way.

The problem with the approaches explained above is that our AStar pathfinder hogs the system for up to four seconds if the path is really long, and in a real time game that is not acceptable. To prevent this major lag we need to separate the pathfinding routine out into a separate process so that the game can continue without stopping. We were also given a tip to change our list operations so that the list sorts itself when new elements are inserted or removed. The priority queue is a data structure in the stl library that does this for us. When the code was written we were not aware that such a list structure existed and it is therefore not implemented yet.