19 Obstructions and line-of-sight

J. Pearse

19.1 Physical obstructions in the simulation space

One of the primary motivations for including obstructions in the simulation was to allow the implementation of path finding and the potential for more complex or unexpected behaviours to arise. There was concern that the resolution of the simulation could incur to high a computational cost with multiple agents all executing heuristic searches concurrently over a large space. As a result it was decided to implement obstructions at a lower resolution than the one used naturally by entities in the simulation. Although the resolution of the 'obstruction grid' was parametrized, it has only been tested at a single resolution $\frac{1}{10}$ of the the resolution of the natural simulation space. As each tile in the simulation has a resolution of 50^2 the resolution of the obstruction grid was set as a matrix of 5^2 cells, so each tile contains a grid of 10^2 obstruction cells. We wanted a simple method of drawing maps of obstructions so we selected a paint program with the capability to export ascii art pictures. Each ascii art map was stripped of line breaks and included in the main JavaScript application as a string. The strings were then indexed to locate the integer positions of obstructed cells and using modulo and integer division, translated into co-ordinates within the obstruction grid space. The implementation of the obstruction grid within the simulation although simplistic with hindsight was a fairly complicated process involving a lot of experimentation, testing and a great deal of drawing everything out on graph paper to make sure it was functioning correctly. However once the correct co-ordinate translations had been established the obstruction grid worked smoothly. The process of instantiating the obstruction map is;

1. Iterate over the map string (in JavaScript) to create an array, of indices, of obstructed cells
2. Pass the array over the web socket into the swarm application when the simulation is being set-up
3. As part of the `make_grid` method of the `environment` module, the obstructed cells are assigned to the state of their respective tile when the tile is created.

Each tile maintains a list of obstructed cells inside its geometry as co-ordinate pair on the obstructed grid scale. When entities call a function which requires consideration of obstructions the native co-ordinates are simply divided by 5 to determine if they fall inside an obstructed cell.

19.2 Line-of-sight

The main system dynamic which makes use of obstructions is line-of-sight, when we first implemented the obstruction system we observed the behaviour of entities become very unrealistic, there was nothing in place to prevent entities from 'seeing' other entities through obstructed cells. This caused large swarms to become trapped up against the opposite side of obstructions as they continually tried to close the distance between themselves and the target. It was obvious that to maintain any sense of realism we would need to prevent entities from seeing through obstructions. It was clear that data obtained from the viewer (already filtered by distance) would need an additional filter to restrict it to only those entities, to which an uninterrupted line segment could be traced i.e a line segment which did not intersect an obstructed cell. An extra module was written to provide the linear algebra functions required to trace these line segments (`los.erl`), again a seemingly simple task in hindsight, this required a great deal of testing and drawing out of graphs with pencils and paper. Once the list of visible entities has been established it is passed over the the WebSocket as part of the entity state. The lists of zombie-type entities and human-type entities are separated in the state but each entity in either list is stored with its corresponding position. SVG line elements are simply defined by four points $(X1, Y1, X2, Y2)$ these obviously correspond to the entity (X, Y) and target (X, Y). In the final version of the client, the animation has been removed from the sight-lines visualisation. As the entity is still being animated this sometimes causes the origin of the lines to jump ahead of the entity as they move forward, however the animation of a very large number of lines proved too computationally expensive on some of our test systems.

19.3 Conclusion

As a result an extra functionality was added to the client (see section: 17.7) to visualise every uninterrupted line which could be drawn from a given entity. This new functionality when added to the client proved to be an unexpected source of information about the system state and inadvertently revealed one of the more interesting aspects of trying to

visualise a highly concurrent system. Unfortunately, due to time constraints real path finding was never fully integrated into the system although we did make some progress in that direction. We consider the obstruction system as a key component of the simulation which contributes to the emergence of many interesting entity interactions.

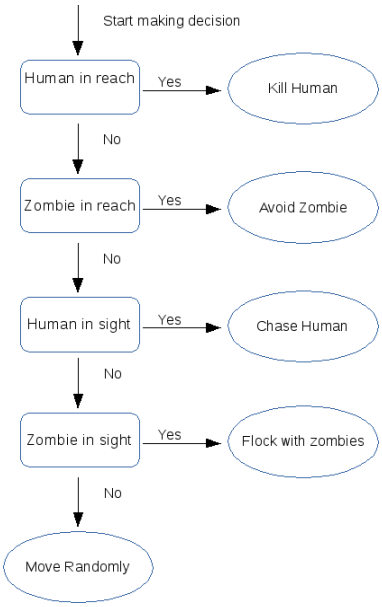

Figure 14: Zombie priorities diagram

way as moving towards the centre point, but uses the X and Y velocity of the other zombies instead. the two resulting changes in velocity are added together to get the total change in X and Y velocity. This action is also based off the boids algorithm, in particular the idea of flock cohesion and alignment.

20.3.7 No entities in sight

The lowest priority action is if there are no humans or zombies in sight.The zombie will start to change it's velocity randomly, this is to ensure that if a zombie has no targets it will still move and appear to search for other entities to interact with.

20.4 Behaviour After Decision Making

After making a decision the change in velocity resulting is added to the zombies current velocity then several limits will be checked to prevent the zombie moving against the rules of the simulation.

20.4.1 Limit speed

Zombies have a maximum distance they can move in one action,if the velocity of the zombie exceeds this value then a new velocity is calculated that keeps the same heading but with a distance at the maximum allowed, this prevents zombies moving unrealistically fast if they repeatedly change velocity to go the same direction.

20.4.2 Energy

Energy represents how recently a zombie has eaten. Energy decreases every time a zombie makes an action. If the energy of a zombie falls too low then the maximum distance a zombie can move per action is decreased to represent the zombie slowing down from hunger.Energy is refilled by killing humans and eating them.

20.4.3 Obstructions

If the spot the zombie intends to move to is obstructed either by another entity or an obstacle then the zombie will bounce off it to avoid collisions.If the obstacle is a wall, the zombie is placed next to the wall and its velocity is changed to -1 of whatever direction it hit the wall in but kept the same in the other axis to enable the zombie to keep