18 Entity Interaction

J. Mitchard

As explained in section 16.2, the simulation has been implemented with a tile-viewer architecture. Whilst this allows for the the system to run more efficiently, it meant that a slightly more creative way of entities maintaining awareness had to be created. This section will cover the specifics of how tile and viewer processes interact, and how each entity interacts with them throughout the simulation.

18.1 Tiles and Viewers

When the map of tiles and viewers are created by *environment.fsm*, they are each assigned a unique viewer along with the viewers of all of the neighbouring tile processes. This creates a grid as shown in section 9.2. The purpose of these viewers is to allow entities to ask the viewer for their current tile what is around them, or, more specifically, what other entities are currently located on the same tile and in each of those surrounding it.

Each tile updates each of the viewers making up its 'neighbourhood' of local tiles and viewers with new data regarding which entities are on it and where. Each tile reports to its viewers from four sets of data held in its state. These are:

zombie_map - A map of each zombie currently on the tile with the zombies Process ID as the key.

human_map - A map of each human currently on the tile with the humans Process ID as the key.

item_map - A map of each item currently on the tile with the items Process ID as the key.

obs_list - A list of obstructed coordinates in the tile.

18.2 Keeping the System Current

Each time an entity moves, enters or leaves a tile, the tile in which the action has taken place will update its associated neighbouring viewers with a new set of data relating to the type of entity that has been updated. For example, if a zombie moves from one position within the tile to another, the tile would only update the viewers with its zombie data, rather than that of the items or humans as well.

The viewer holds similar information to the tiles, however instead of mapping each individual process as a key, it is mapped by tile with a list of entities as the values. This allows for a simple method of updating the viewer processes.

Because these data sets change so often and unpredictably, Erlang Maps are perfect for the job as when new data is inserted into the map with a pre existing key, the values are overwritten with the new set. This is considerably quicker than having to iterate through a list and changing its value.

18.3 Entities and Awareness

In order for humans and zombies to have an awareness of their surroundings, it is necessary for them to ask the Viewer process for their current Tile for information; regarding nearby zombies, items, obstacles, and other humans. Because this information is constantly changing, it will have to do this at the start of each cycle of its state machine.

However, because each entity has a defined line of sight that is considerably less than the length and height of three tiles, which is what the viewer would provide information for, these lists will have to be filtered down to have this taken into consideration.

Firstly, each list is filtered to remove entities further than a defined distance away, the line of sight varies between humans and zombies. Secondly, the list uses the *findline* function from the *los.erl* module, which is introduced in section 19, to filter out entities in which line of sight is obstructed by obstacles. This will provide each entity with the information needed to make informed decisions of what to do next.

18.4 Example of Process Communication

This diagam shows an example snapshot of communication between a human, tile and viewer processes in which the human requests information from the viewer, and then requests to move to a new position. The example assumes that the requested position was not obstructed.

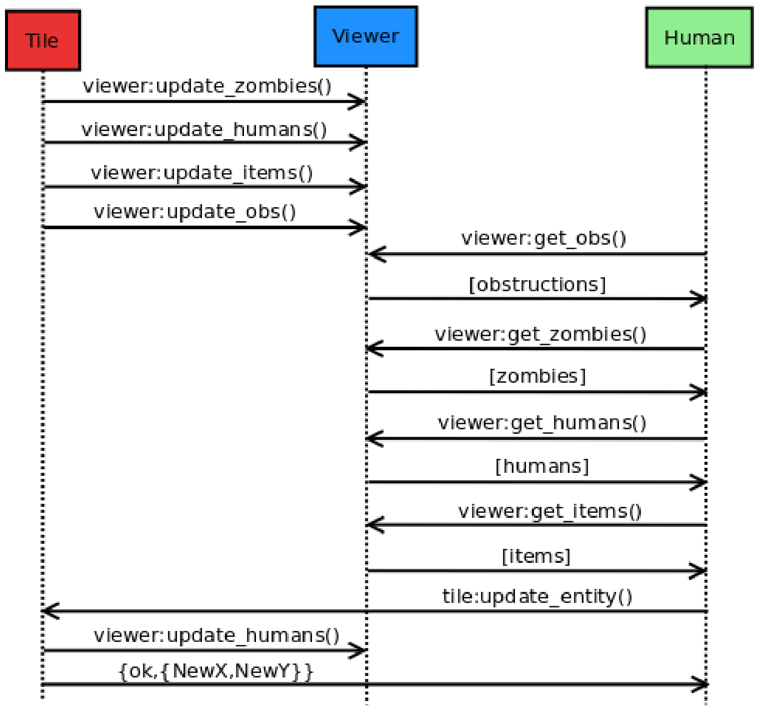

Figure 12: Human, Tile and Viewer Communication

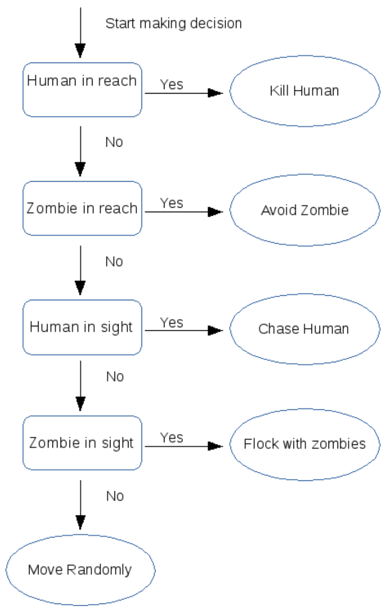

Figure 14: Zombie priorities diagram

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