21 Human Intelligence

J. Mitchard

From early on, we had decided that human entities are to be more intelligent than their undead counterparts. In order to establish a sense of realism in our simulation, the human entities have to make complicated decisions based on their current situation and what is going on around them. The way in which an moves is covered in section 20.1, as this is generic to both types of entity within the simulation. This section will explain which behaviours have been implemented, which decisions are made in order to achieve them and what is taken into consideration to make them believable actions.

21.1 Human Behaviours

The human behaviours in the simulation are modelled through a heavily extended version of the Boids algorithm, as introduced in section 14.1. In order to survive, the human entities have to avoid the zombies, however it is also necessary for humans to eat food found around the world in order to maintain energy. Over time the humans become hungry which, in turn, will cause the humans to slow down as they lose energy, making them easier for the zombies to catch.

21.1.1 What Behaviours are Present?

Human entities in our simulation are currently able to replicate a number of different behaviours, these are:

Collision Avoidance - The human avoids hitting both other entities and obstacles.

Zombie Repulsion - In order to survive, a human entity must try and avoid contact with zombies.

Flock Attraction - Humans are social creatures, this behaviour will allow the humans to attempt to group together with other humans and move as a group.

Food Attraction - When hungry, and there is nearby food a human will be attracted to the item of food.

Maintaining Memory - Food can be scarce, so it is important for a human to remember the location of food that it doesn't need at present.

Pathfinding to Food - When hungry, a human will try and move towards known locations of food.

Search - When there is nothing else to do, a human can search the area around him, looking for food or other humans.

21.2 Human Intelligence Implementation

21.2.1 The Decision Making Process

Humans decide what to do next based on a set of weighted priorities, taking into consideration their current surroundings and circumstances. Each time a human runs through its state machine cycle, introduced in section 16.1.1, it will calculate new levels of hunger and energy and request new, updated lists of nearby entities from its current tiles' Viewer, this process is explained in section 18.3. These include a list of other humans, a list of zombies, and a list of items; allowing the human to know what is happening around him.

Based on the information received from the Viewer the human can now make an informed decision. The priorities are ordered as such:

1. Move to avoid crashing into other entities.
2. If there are nearby zombies, move away to avoid them.
3. If hungry and currently next to the food, attempt to pick it up.
4. If hungry, there are no zombies and there is a path to an item set, follow the next position from the path.
5. If hungry, there are no zombies and there is nearby food, move towards it.

6. If there are no zombies, group together with other human entities.

7. If nothing else matches, search for food.

The order of priority has been heavily considered and tweaked to ensure the humans act as realistically as possible. For example, we believe it unlikely that a human would run aimlessly towards a source of food knowing full well that there is a number of zombies between itself and the food. In order to represent this decision we have put finding food lower in the chain of priorities than avoiding zombies.

21.2.2 Implementing the Choices

Implementing these behaviours in Erlang presented some unique challenges, but also some very interesting advantages.

Due to the nature of a concurrent system, with lots of processes running simultaneously, timing can be a complicated issue. An example of where this sort of issue could become a problem is that multiple humans could attempt to pick up an item of food at the same time. The process for the item of food would receive both requests in a queue, however would only respond an 'accept' message to the first one, any others it would send a 'reject' message. This allows us to pattern match against what is received from the food process and deal with it accordingly, whilst avoiding causing a deadlock in the system.

This snippet of code from *human_fsm.erl* shows how the human process matches against what is returned:

```
% There are no zombies, I have no path, move towards food and attempt to eat it
make_choice(_,[],{ItemId,{ItemX,ItemY,food,_}}, very_hungry, _Path, #state{x=X,
    y=Y}) ->
   case swarm_libs:pyth(X,Y,ItemX,ItemY) of
      Value when Value =< 2 ->
         Item = supplies:picked_up(ItemId),
         case Item of
            ok ->
               {X,Y,eaten};
            _ ->
               {X,Y}
         end;
      _ ->
         boids_functions:super_attractor(X,Y, ItemX, ItemY, ?SUPER_EFFECT)
   end;
```

On the other hand, functional programming has presented us with a number of interesting ways of solving problems. Deciding what behaviour to next carry out lends itself perfectly to pattern matching, whereby the function will fall through multiple function headers until one meets the current state. Using pattern matching in Erlang, we have created a 'one-size-fits-all' function that decides on what the entity should do next; The behaviours are each implemented in a different branch of the function, each meeting a different specification until one is accepted by the function and the behaviour is carried out.

In the above example, the human entity knows that there are no nearby zombies in sight, and that it is near an item of food. The human will attempt to pick it up if it is close enough, and if another human hasn't picked it up before them it will return to the main state machine that the human has eaten. Otherwise, the human will make an attempt to move towards the food.