CMPSC 383: Multi-Agent and Robotic Systems

Final Project Progress Report

1 Language Choice

We chose Haskell as the implementation language for our Boids simulation. Haskell is a purely functional language with lazy evaluation. As our class experience with implementing Multi-Agent simulations has been solely through an Object-Oriented Paradigm, our language choice presented a challenge. Our implementation of the Boid agents involves the definition of an abstract data type, which contains the position, velocity, and neighborhood radius of an individual boid. In Haskell, this is defined as follows:

```
data Boid = Boid { position :: Point
                 , velocity :: Vector
                 , radius   :: Float
                 }
```

Note that Haskell does not allow mutation of existing values. Therefore, once a Boid is created, it cannot be mutated, and instead updating the BoidâĂŹs state requires the creation of an entirely new Boid instance.

Also, in contrast to a corresponding OOP implementation, which might define some Boid behavior to accompany this basic data structure, this Boid data type is kept distinct from its update method. Instead, we define a type called `Update`:

```
type Update = Boid -> Boid
```

Thus, a function of type `Update` is a function that takes a `Boid` and returns a new `Boid`. We use `Update` to define `Behaviour`:

```
type Perception = [Boid]
type Behaviour = Perception -> Update
```

This defines a `Behaviour` as a function that maps a BoidâĂŹs neighborhood to an update function. These examples demonstrate how implementing a multi-agent simulation in a purely-functional language requires a different conception of what it means programmatically for an Agent to behave.