Alex Tuckner: tuckn016
Student ID: 4730040

Iteration 2 Design Document

In the iteration 2 portion of the robot project, we as students were asked to implement and design a game in which uses both a user and robots to achieve a final goal. In doing so, we had to implement sensors for Proximity, EntityType, Distress, and Collision/Touch. In order to achieve this goal, I created a base class for all sensors, and decided to use the observer method to achieve the goals of the sensors within the arena. The arena is set up to look through all of the different sensors attached to all mobile entities, and check for events. When an event occurs, such as a distress call, the arena will then send out that information to all the mobile entities and they will then react accordingly. This type of observer pattern makes sense for this type of project because there could be multiple objects that could use this information, and instead of constantly checking, can just send it to all the objects. The downside of the observer pattern in this situation is that not all objects may need to obtain that information or find it useful, so much of the data sent may be ignored throughout the simulation.

Another important design decision I had to make was how to implement the motion handlers for each mobile object within the arena. I had realized that during the first iteration, I had created the robot motion handler, but had failed to implement it properly as the arena could use the robot to change its angle/speed directly, rather than using the motion handler to do so instead. I had to decide which parts of the motion would have to be decided by the handler and the object itself. Because the mobile entity object already has a Position within, I decided to leave the position inside the entity, but leave the speed and the heading angle up to the robot. In this case, when an event happens that may cause an entity to shift direction, it will pass along this object directly to the motion handler to correct for the new angle. This approach to solving this problem is good in that it separates the complexity of the classes and separates different jobs into each class. Another positive is that it looks cleaner in the code to just send it from the entity to the motion handler, rather than dealing with any calculations in the entity itself. A downside to this approach is that it may slower such that information must be passed along one more time for every event that occurs. This method is also not as direct as before when the arena could change the robot's parameters directly, as it must be passed to the motion handler to resolve.

There are positives and negatives to the observer pattern design approach. This design allows a more realistic approach to the robot as if it were a real entity. For example, if we were building a real-life robot, the sensors on the robot would be telling the robot the information, not the arena that it was put in. The arena in this case just is there to inform the sensors of the objects surroundings. The flaws of this design from a coding perspective is that the information get passed around so much from the arena to the sensor to the entity checking the sensor, to the motion handler, that it can get very confusing about where the information is going and can easily be mixed up or lost along the way, leading to confusing errors.