Design Document

The key component of the design consisted of creating 4 different types of sensors (which all inherit from Sensor) that will be used by the autonomous robots to avoid other arena entities when appropriate. The Sensor base class can accept an EventRecharge, EventCollision, and EventCommand. Moreover, each type of sensor accepts different types of events (which all inherit from EventBaseClass) in order to update the speed and direction of the robot/player accordingly. The types of sensors and the events they accept are as follows:

- SensorDistress accepts an EventDistressCall
- SensorEntityType accepts an EventTypeEmit
- SensorProximity accepts an EventProximity
- SensorTouch accepts an EventCollision

The interface for these sensors is modeled after the observer pattern in that the entities themselves have a list of their respective sensors and they notify such sensors of any changes by sending them events. The sensors then activate themselves based on the information provided. Afterwards, the distinct motion handlers for the mobile entities (i.e. Robot, Player, and HomeBase) update the velocity and heading angles of such entities based on the activation status of the sensors.

My main design choice for how sensors are notified differs from Professor Larson's suggestion in that instead of having the Arena contain a struct or list of sensors and updating them directly, it instead has a list/struct of the entities and events and sends the events to the entities, which then update the sensors according to the event information provided. The reason for this design choice is that I wanted to be consistent with my implementation of iteration 1 where the Arena only had the populating entities and updated them by sending events. I didn't want to provide sensor information to the arena because it doesn't seem necessary to have the arena contain sensors when its arena entities are the ones utilizing the sensors and, therefore, should be the only ones to directly communicate with the sensors by sending them information. Furthermore, I wanted to model a real-life behavior where the sensors aren't a part of the Arena, but are physical parts of the robots and player. Therefore, like how it was implemented in Iteration 1, the Arena checks for the collision under the arena method CheckForEventCollision() and checks for proximity events under the arena method CheckForEntityProximity() at each time step and such events are sent to the populating entities so that they can pass along the information to the sensors and have their velocity and direction appropriately changed (if at all). Furthermore, EventDistressCall and EventTypeEmit are also sent to the robots to appropriately activate such sensors when the robot is frozen. SensorProximity has a method called inRange() that's called at each time step to determine if the entity is within range of the robot's sensor. If so, the robot uses entity type sensor to determine the type of the sensed entity. If the sensed entity is a robot, the sensing robot checks if the distress sensor of the sensed robot is activated. If the distress sensor isn't activated or the sensed entity is not another robot, the arena calculates the angle of incidence in CheckForEntityProximity() (similar to how it's implemented under CheckForEventCollision()) at which the sensing robot should bounce off of the sensed entity in order to avoid collision.

The final major design choice was how to implement SuperBot. I decided to create a separate SuperBot class that inherits from ArenaMobileEntity since its behavior varies enough from Robot in that it shouldn't use MotionHandlerRobot. So, whenever a Robot collides with HomeBase, the robot is deleted from the arena and a SuperBot is initialized in the arena at the same position that the deleted robot was initialized in. I plan on changing this design to instead follow the strategy pattern where if the robot collides with home base, it switches its MotionHandlerRobot for MotionHandlerSuperBot so that it doesn't ever freeze. This would mean deleting the SuperBot class entirely in favor of a MotionHandlerSuperBot class. This is preferable to the way I have it currently implemented because it seems redundant to have a SuperBot class that contains everything that Robot has, except for the motion handler. Therefore, it'd be more efficient if SuperBot stays as a Robot, albeit with its distinct motion handler. Then, no entities would have to be removed from the arena and the implementation for SuperBot may become more streamlined.