Sensor part:

1. Since we need 4 sensors to do all detect work, so we need a sensor base class, which will contain the base function or identity for it's child. So basically, we have activated and reset function in the sensor base (which is just sensor.h).

2. Proximity Sensor & Event Proximity: the proximity sensor is inherited from sensor.h, so combine with event proximity, I “draw” the sensor in the proximity sensor (like we implement the detect range, heading angle etc.), I did not create much functions in proximity sensor class, only emit message and the function check if sensor catch the object or not. Since we have event class, so I think most work should be done in the event class. According to that, and since we need 2 proximity sensor, so I made that left and right part of proximity sensor, each of them will find object in the arena. So once the proximity sensor catch the objects, it will activated the sensor and call the function “get\_distance” which will return the distance between the object with robot. Also if nothing is detected, it will return -1.

3. Contact Sensor & Event Contact: also “draw” the sensor in the contact\_sensor.h/.cc then do the function (within event\_contact.h/.cc) if the sensor knows it contact, it will make reflect action based on the information it gathered. For this part, most things are similar to the sensor touch, but we still contact sensor, since we may not only need to reflect, we may also need some other function while robot contact something (example: robot touch the home\_base).

4. Distress Call Sensor & Event Distress Call: “draw” in the distress call sensor, “function” in the event distress call. Once the robot touch by the player, it will be stunned, the it will sent out the distress signal, which will be catch by other robots distress call sensor. According to that, we need a boolean function to say whether the sensor detected the signal or not, then we need change\_direction function, which will turn the moving robots’ head to the frozen robot . so if it’s in range, then change\_direction.

5. Entity type Sensor & Event Entity Type: “draw” in the sensor, “function” in the event too. Then we need a entity type(a new enum), I call then kWall, kRobot…, event entity type will have a detect function say whether there are any objects in the detect\_range, if yes, then call get\_object, this function will get the object, and set the object type to event’s private value obj\_type, this value will pass into sensor, and sensor has the getter to return this type.

Player part:

Player is “old version” robot, (old version here means in iteration1), so what we need to do is just change the all “robot”->”player”. If robot in iteration1 make some sense, then the player will also make sense.

Robot part:

We don’t need the battery for robot now, first I copy many thing from homebase, but since we need the id for the robot and we may implement more than 1 robot, I keep the id number, which will help us to identity different robots. And for outside of the robot class, anywhere have “homebase” we just follow it and add the same things as homebase did, also include robotmotionhandler and robotmotionbehavior.(SInce home base has these two files.)

SuperBot:

Contact sensor feels “robot”->”homebase” then, call power\_up to make robot into super bot. Which will change the situation when player hit the super bot. Also the color will change too, which will make super bot looks more “dangerous”