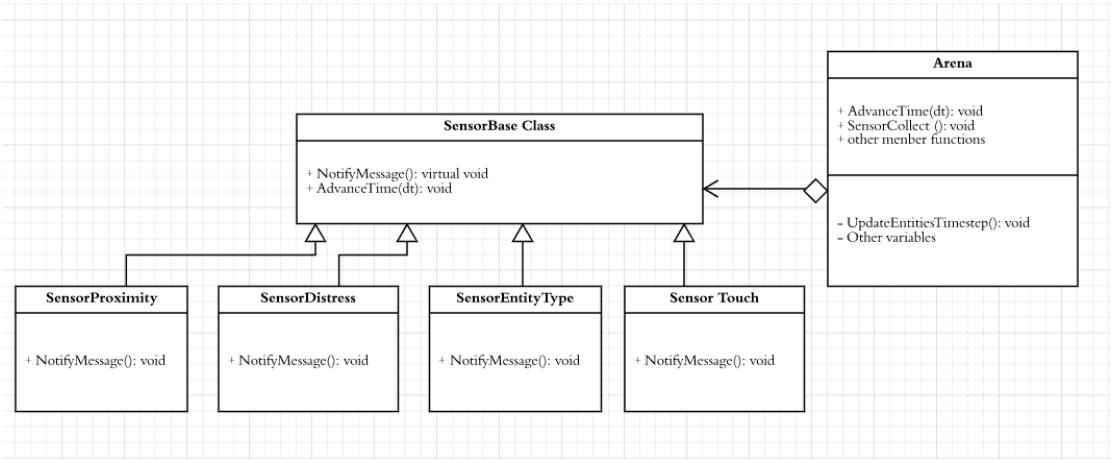
CSCI-3081w
Zijing Mo
Iteration 2

Design Documentation for Different Interfaces

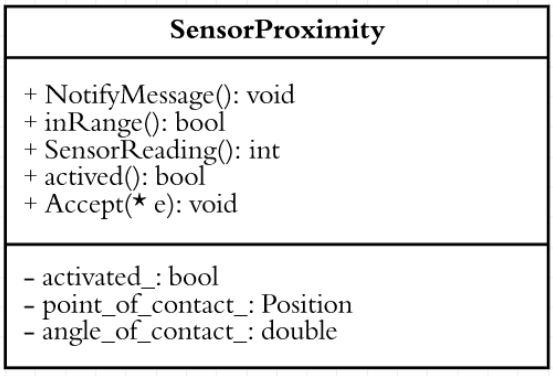
In this paper, we will understand some basic design idea of iteration 2 for our programming project. This writing would introduce the new features of iteration 2 with graphic illustrations. On the other hand, the readers or other developers also can understand basic design concept in this paper. It might be noticed that all designs do not implement indeed so the designs might be modified in future. First of all, we will introduce some new properties compared with the first iteration.

In this new version of project, we will introduce new entities in our arena system, named SensorBase class. For sensor class and its subclasses, SensorProximity, SensorDistress, SensorEntityType, and SensorTouch, those satisfy observer pattern. The SensorBase class would watch any change from arena class and its subclasses. As a usual, this class will transmit the data from the arena to its subclasses by update() function. If the situation in arena has triggered the condition of sensor events, the subclasses of SensorBase class will notify the outcomes or changes to their parent class. The SensorBase class will told the arena class what will happen in next frames. The graph of basic specified relations as follows.



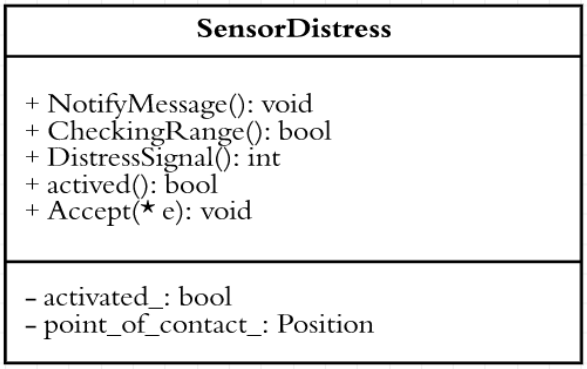
The SensorBase class will would gain the notify message from its subclasses and tell the arena class how to do that. In the next part of paper, we will analysis how to implement functionality of each subclass.

SensorProximity class will be the first part. A proximity sensor should have a range and field of view, such that it has a limited cone in which it can sense objects. The range and field of view, expressed as an angle, are attributes of the sensor. Sensor output is the distance to an obstacle. If there is no obstacle, it should output -1. The specified design in UML graph as follow:

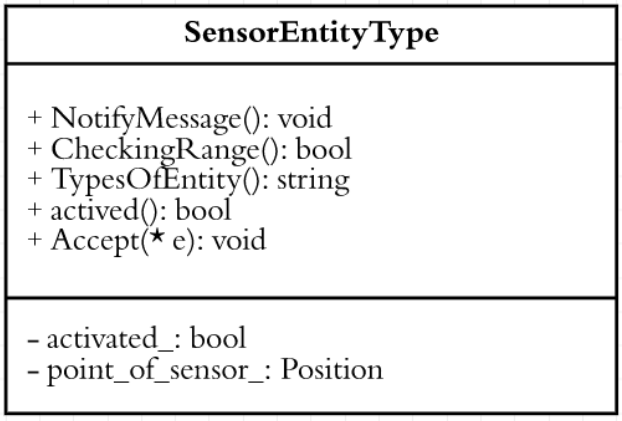


Form the SensorProximity class table, we can find some specified functions to detect the range of contacting entities and check the state of the events.

In the next, we will talk about the SensorDistress class. In this entity, the system will create a return value which is called distress signal. The distress signal can be sensed when it is within a defined range, but the direction of the signal cannot be determined. Sensor output is 1 for a sensed call and 0 for none. Its UML graph as follows.

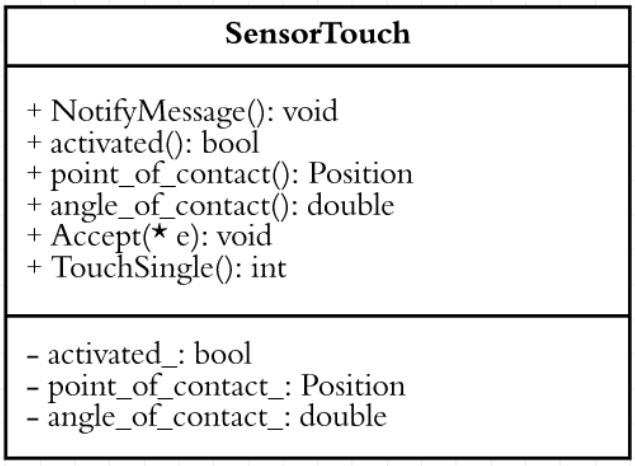


The SensorEntityType class is similar to SensorDistress class. The only difference is the type of the entity emitting the signal can be sensed when it is within a defined range, but the direction of the signal cannot be determined. Sensor output is the enumerated entity type.

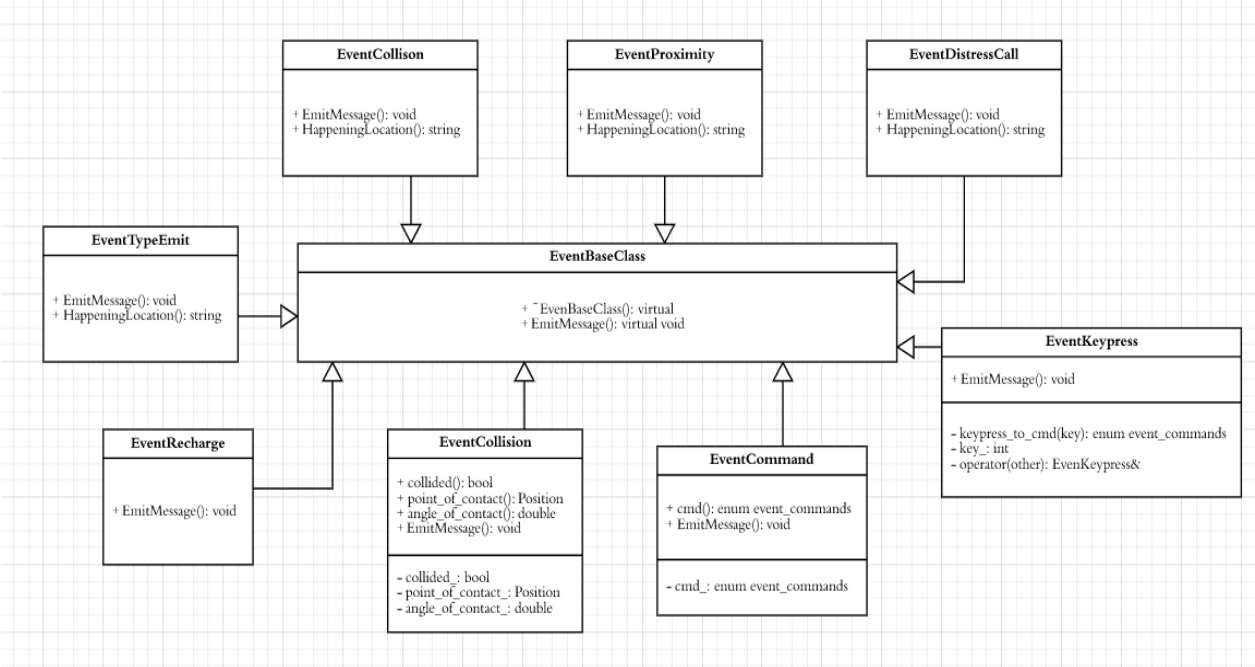


The last subclass of SensorBase class is modified from the SensorTouch in iteration 1. The difference from the first iteration is Sensor output is 1 for a sensed

collision and 0 for none. That means the output will not be the Boolean values so that SensorReading() function could identify it.



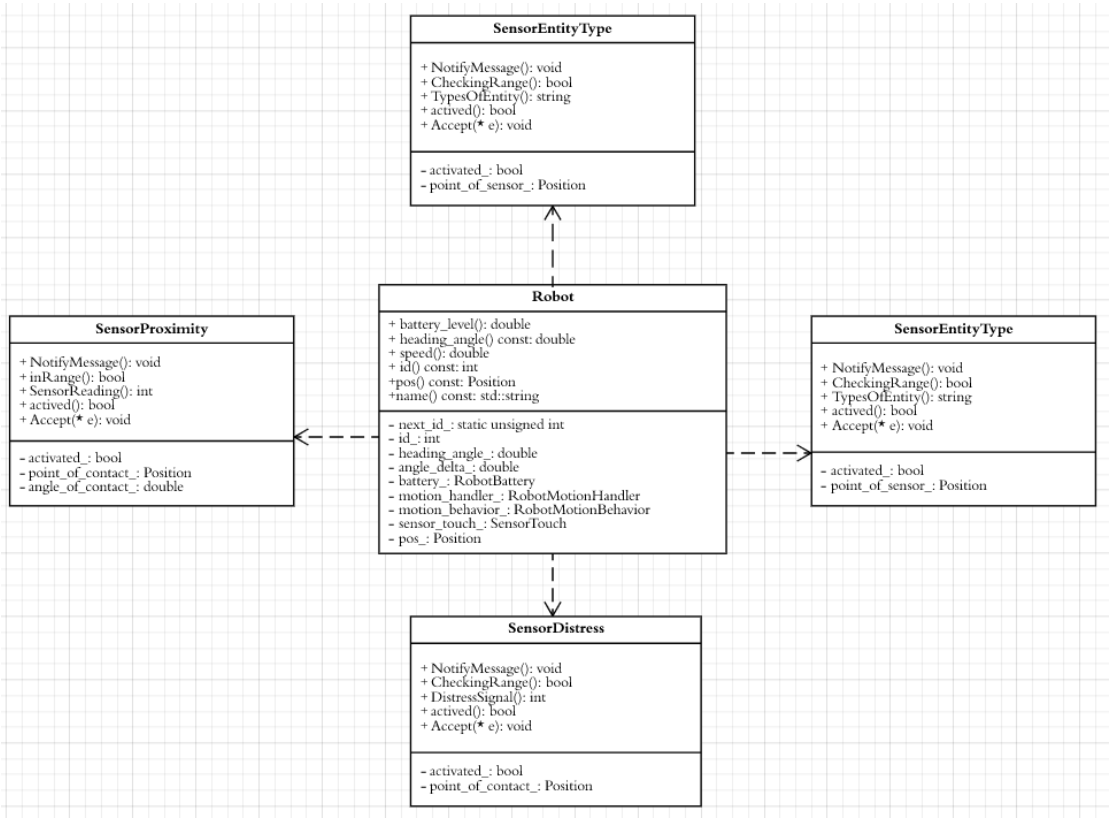
Once the sensors class family has been introduced in our robot simulative system, we need to created corresponding events in the events classes. In this time, we will not apply the observer pattern and will keep the strategy pattern for EventsBaseClass and its subclasses yet. The graph of relation as follows.



After that, we also need to define a new MotionHandler Base class to control all actions for all entities. The structure of MotionHandler Base classes family is alike the Sensor classes family. As a result, it is not necessary to put the UML graph of this family as well. The specified motion handler classes we designed would be RobotMotionHandler, HomeBaseMotionHandler, PlayerMotionHandler.

Finally, we will introduce the combination of robot classes and sensor classes. The promotions from the first iteration and second iteration is that we have distinguished the robot classes and players classes. Although the creating and operating methods of players class and robot class is same. The robot class is

autonomous in our system. That means the user cannot handle these robots. Moreover, we also create a class which is named SuperBots. The entities from this class can unfreeze all robots and freeze the entity the user control. It means that the speed of players will be always zero when it collides with the entity from SuperBots class. The internal structure of robot class is same to the iteration one. As a result, I will not illustrate those class in the paper. The UML graph as follows are regarding to mutual relations between robot classes and sensor classes.



This is the documentation of iteration two. I am glad to hear any advice from you!