Software Design Document

Robot Simulator: Iteration 2

WING YI (PINKI) WONG

NOVEMBER 29, 2017

FINAL REVISION

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to describe the implementation of the Robot Simulator described in the Iteration 2 Requirements. The Robot Simulator is improved with the addition of various types of sensors, a user-control robot and autonomous robots.

1.2 Scope

The software continues from iteration 1, which is a robot simulator that robot behavior is visualized within a graphic window. The software now consists of a sensor base class with four different types of sensors. The interface of each sensor is first designed, and each of them will connect with other entities, which enhance the functionality of the entities. This document will not specify any testing of the software.

1.3 Definitions, Acronyms, Abbreviations

Player – The entity that user can control with the arrow keys.
Robot – The autonomous entity that moves around within the graphic window and needed to be all frozen to win the game.
Frozen Robot – A autonomous robot will be frozen if it collided with the player.
Superbot – A frozen robot will turn into a Superbot if it is collided with another autonomous robot.
Home Base – The entity that allows ordinary Robot becomes Superbot through collision.
Obstacles – An entity that depletes player's energy(battery) through collision.
Recharge Station – Allow player to recharge battery to max level through collision.

Sensor – The "eye" of the entity that sense objects in a specific range.
Fields of view – The visible angular area viewed by a sensor, in degrees.
Range – The range of number of degree, fields of views, of the sensor.
Sensor Proximity – The sensor that has a range and field of view, which can sense objects in a limited cone to prevent collision.
Sensor DistressCall – The sensor that emits a distress call to ignore senor proximity when approaching object is in range.
Sensor EntityType – The sensor that returns the entity type of the sensed approaching objects.
Sensor Touch – The sensor that senses for collision event.

2. System Overview

The Robot Simulator for iteration 2 has added in several features. There is a sensor base class for SensorProximity, SensorDistress, SensorEntityType, and SensorTouch. All of these sensors help sensing the entities within the range of the field of views of the sensors to either prevent collision, distress call, identify the approaching entities or confirm any collision events.

Besides sensor classes, it also includes the Events interface. There are EventCollision, EventProximity, EventDistressCall, and EventTypeEmit. Each of these provide information regarding the entities involved and the location of the event.

For the implementation of the Robot Simulation, an abstract MotionHandler base class is now used, which is the base class for MotionHandlerRobot, MotionHandlerPlayer, and MotionHandlerHomeBase. Robot class is renamed as the Player class, while there are also a new Robot interface and a SuperBot interface.

This document describes the implementation details of the Robot Simulation.

3. Architecture Overview

3.1 Architecture Design

Figure one shows the higher-level system architecture. The system will be constructed from multiple distinct components:

SensorProximity – The sensorProximity is important for avoiding collision. Each automated robot will have 2 proximity sensors, which has a range and field of view that allows sensing object in a limited cone size. When other entities reach the sensing cone, Sensor will sense the entities by checking if EventProximity is accepted. If sensor activated, the output function will return the calculated distance between entity and obstacle at the end.

In my design, there is a constructor, a destructor, a getter for position, a InRange function, an accept function and a reset function. In the constructor, sensor is not activated. Using the InRange function with getter function for position, it will return a Boolean value indicates if the approaching object is in range. If yes, sensor activated. Otherwise, not activated. Distance between entities will be calculated within the accept function. At the end, if sensor is activated, distance will be returned.

SensorDistress – The sensor_distress is important for the frozen robots. Sensor will check if EventDistress is accepted. If yes, it will call on the output function to emit distress call to arena to deactivate the sensorProximity

when the robot is frozen due to collision with player. It will return 1 for a sensed call and 0 for none. It will allow approaching robots to ignore the proximity sensor and bump into the frozen robot to help de-frozen.

In my sensor_distress, I have a constructor, a getter and setter for activated, an accept function, an output function, and a reset function. The constructor will take in a robot_params and a range. My design is to use the given range for sensing the surrounding, which means any robots within the given range range will be detected. Inside using two cone of range, the sensorDistress is more like a round-shape sensor that allows sensing everything from different angles that is within the range.

Accept function will be used for getting the position from EventDistressCall to calculate the distance and check if it is within the range. If yes, sensor activated and output function will return 1 as an indicator. Otherwise, not activated and output will be 0.

SensorEntityType – The sensor_entity_type is important for getting the types for the approaching entities and decide the next actions. SensorEntityType is similar to the sensorDistress. Instead of returning 1 or 0, it will return the approaching object's entity type if it is within the range.

In my sensor_entity_types, I have a constructor, getter and setter for activate, an accept function, an output function and a reset function. Again, the constructor will take in a robot_params and a range, and the Accept function will be used for getting the position from EventDistressCall to calculate the distance and check if it is within the range. The name function will be used for getting the name(type) for the entity when it is sensed in range.

If sensor activated, output function will return the entity type.

SensorTouch – Sensor will determine if a robot needs to be frozen by checking if EventCollision is accepted. Emit a message "1" indicates that there is collision. If collision is false, it will return "0". The returned message will be used to decide if the robot needs to be frozen.

In my sensor_touch interface, it has a constructor, a destructor, getters and setters for point of contact and angle of contact, an accept(), an output() and a reset(). Sensor touch will detect any collision event by activating the sensor at some point of contact and translate to an angle of contact

EventCollision – Check if any approaching entity collided and activate or deactivate the sensor. If it bumps into the player, it will be frozen. This will be checked using the sensor_entity_type.

There will be a collided function is see if it collides or not. It will keep track of the point of contact and angle of contact by using the getters and setters. It will call on the EmitMessage function at the end to return the checking result.

EventProximity – Use for getting the position for SensorProximity to check if entity is in range.

It contains a constructor and a getter function for position, which allows further calculation for the distance.

EventDistressCall – Check if approaching entities are within range and field of view. Sensor activated which enhance deactivating the sensorProximity.

There is a constructor, getters for position and entity_id. This helps to check if the position falls within circulate range of the sensor. It will call on an update function at the end to activate or deactivate the sensor.

EventEntityType – Check if any approaching entities reach the ranch and If yes, sensor activated and it will be able to detect the type of approaching entities.

There will be getters and setters for name, and position which allow checking the location of event and returning the name.

MotionHandlerRobot – Have a constructor that have default heading angle and velocity for the robots and allow changing of angle and velocity when different events occur. It will keep track on the sensor events and update the heading angles, velocity.

MotionHandlerPlayer – Allow acceptCommand from keypress commands to change the heading angle and velocity. It will keep track on the sensor events and update the heading angles, velocity.

MotionHandlerHomeBase – Have a constructor that have default heading angle and velocity for the robots and allow changing of angle when different events occur. It will keep track on the sensor events and update the heading angles, velocity.

Player – The original robot interface from iteration 1, which allow keypress event if EventCmd is accept. Its enumerate type is kPlayer.

It has a constructor that set up a sensorProximity with two attribute left and right which works like having two sensors. In the constructor, it also has a sensorDistress, a sensorTouch, a sensorEntityType. It will allow updating the change of heading angle and velocity when accepting different events base on the event commands. It will also change the heading angles and velocity based on the MotionHandlerPlayer.

Robot – The robot entity has the enumerate type as kRobot. It has a constructor that set up a sensorProximity with two attribute left and right which works like having two sensors. In the constructor, it also has a sensorDistress, a sensorTouch, a sensorEntityType. It will allow changing of heading angle and velocity when accepting different events. There will be a setter and getter for color in order to show the difference between SuperBot and Robot. Base on MotionHandlerRobot, when sensorTouch accept the EventCollision, robot become SuperBot if the sensorEntityType shows that the colliding entity is player. The enumerated type for SuperBot is kSuperBot.

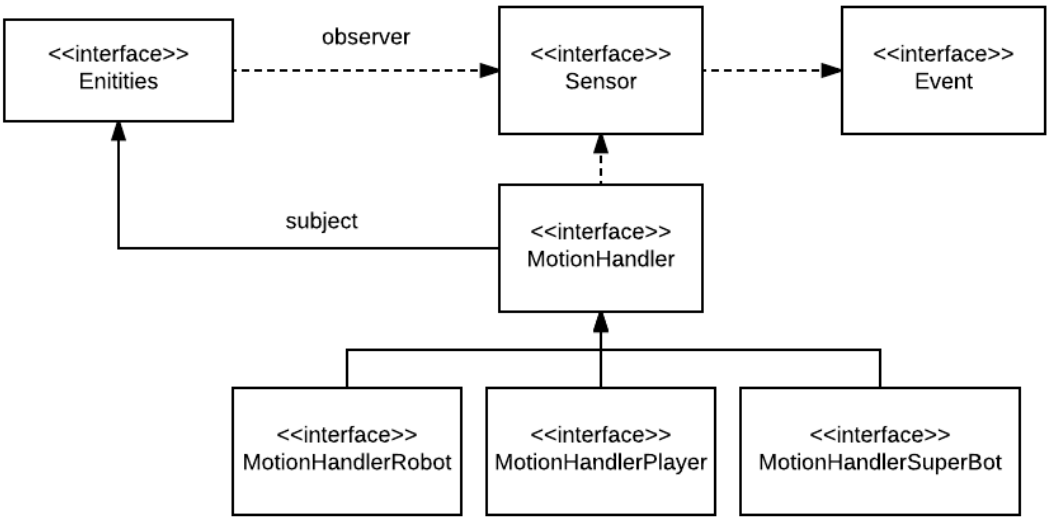

Figure 1: higher-level system architecture

From figure one, it shows an observer pattern that the MotionHandler interface will base on the sensor from each entity to update the motion for each entity. One example could be the sensor from Robot entity detect a collision event with obstacle, it will update the MotionHandler to decrease the velocity of the robot.

3.2 Decomposition Description

The system decomposition description shows the relationship between different entities, or different conditions.

Collisions between robot and other entities:

1. Robot/Wall – Event Collision.
2. Robot/Recharge Station – Event Collision.
3. Robot/Home Base – Event Collision. Robot will also turn into SuperBot.
4. Robot/Obstacle – Event Collision.
5. Robot/Frozen robot – Event collision. Frozen robot defrosts.
6. Robot/Player – Robot freezes.

Collisions between robot and other entities:

1. Player/Wall – Event Collision.
2. Player/Recharge Station – Event Collision. Player's battery level back to max.
3. Player/Home Base – Event Collision.
4. Player/Obstacle – Event Collision. Player's battery level depletes.
5. Player/Robot – Event Collision.
6. Player/SuperBot – Player freezes for few second.

Winning condition:
There will be no Superbots. All robots will be frozen and the player's battery level should be higher than 0.

Losing condition:
There are non-frozen robots. Player's battery level drops to 0.

4. Sensor Interface

From Figure 2, it shows the UML diagram of the sensor base class interface and its derived class interface.

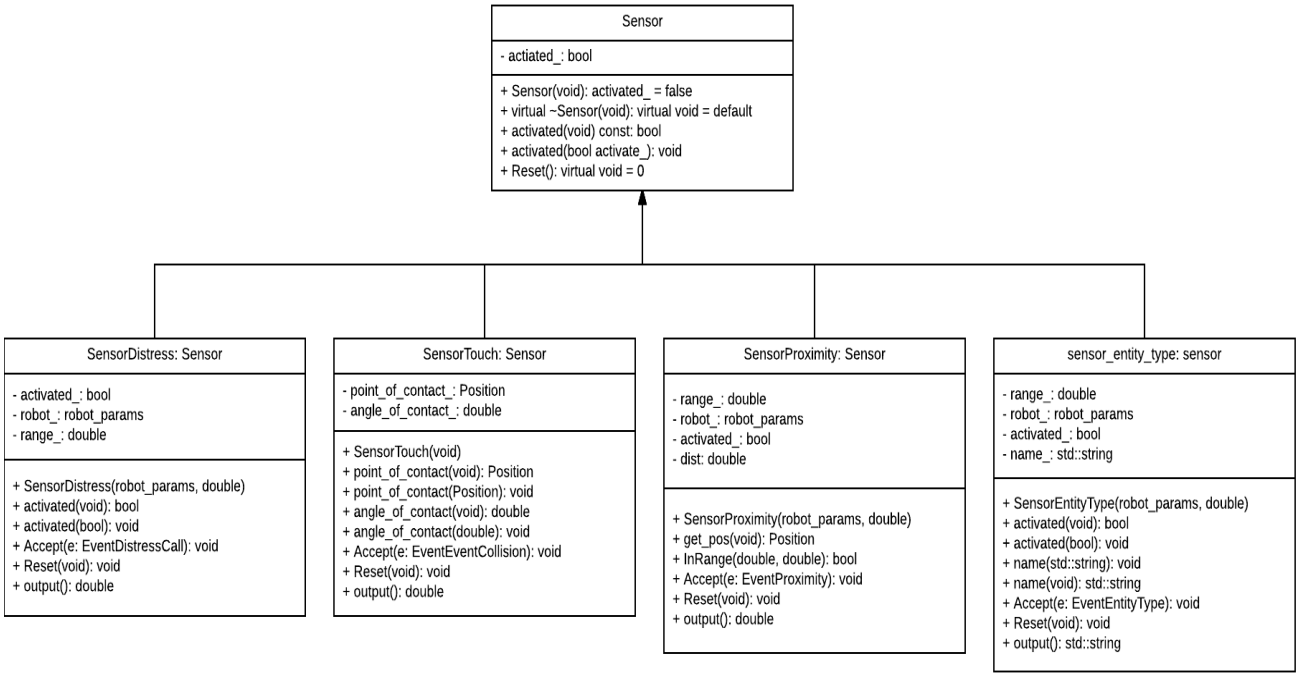

Figure 2: UML of sensors

5. Event Interface

From Figure 3, it shows the UML diagram of the event base class interface and its derived class interface.

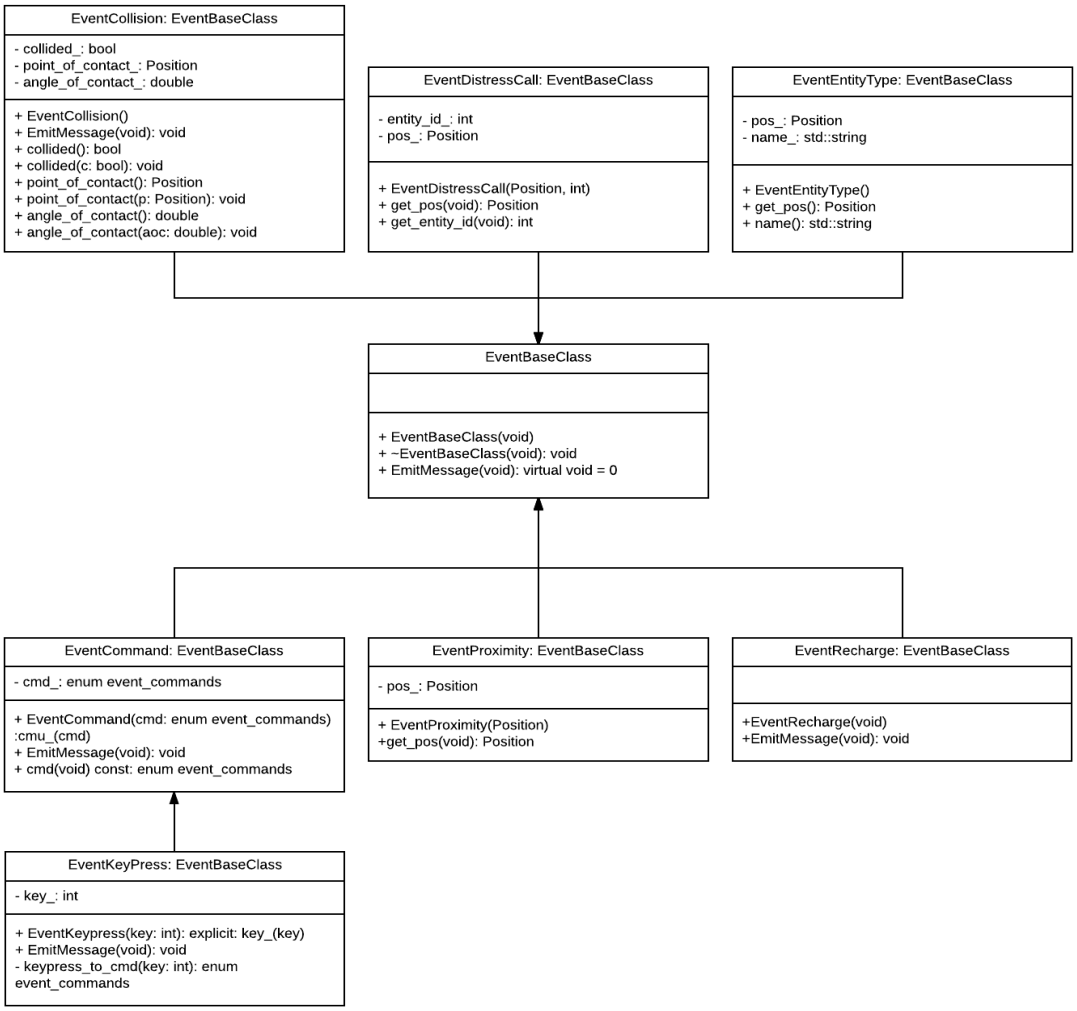

Figure 3: UML of Events

6. MotionHandler Interface

From Figure 4, it shows the UML diagram of the MotionHandler base class interface and its derived class interface.

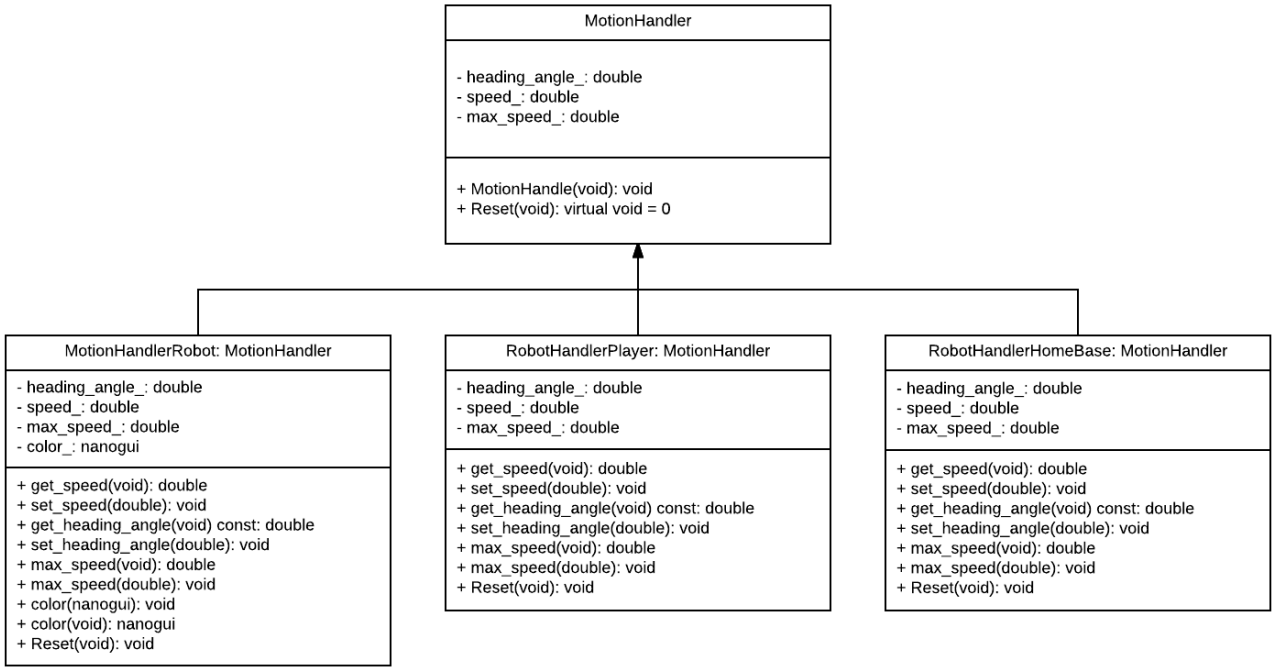

Figure 4: UML of MotionHandlers