Design Document

This project creates a fundamental robot simulator. Like a video game, the robot can be visualized controlled by users in a graphics window. By using libraries, base code. unit tests with Google Test, class structure, Google style guide compliance, design documentation using UML, and doxygen documentation, I barely finished three priorities. The robot can be controlled by pressing arrow keys. Users need to control the robot to hit a moving home base before running out of battery to achieve the game's objective. As normal, robot need battery to move. Thus, the energy would be decreased as the robot moving and even more when it hitting the obstacles. Luckily, reaching recharge station is a way to renew battery level. If the robot run out of the battery and still cannot reach the home base, user lose the game.

As far as the organization of this project, we can simply divide it to three parts,: Robot Arena, Graphics Environment and GUI and User Input. Firstly I look at the Robot Arena. The Robot Arena is consisted with robot, obstacle, home base and charging station. To following the design pattern, we build ArenaEntity first, which includes the most basic attributes of entity such as radius, position and color. And positions are represented by a structure which name is Position in common.h file. {x, y} would be the position of the robot. Then we simply classify robot, obstacle, home base and charging station by their movement. It it obviously that there are two types of Arena Entity. One is Arena Mobile Entity which can move, the other is Arena Immobile Entity which is static. Both of them inherit from Arena Entity class. The requirement tells me that robot and home base are mobile, others are immobile. So robot and home base inherit from Arena Mobile Entity, while obstacle inherit from Arena Immobile Entity. Plus, recharge station inherit from obstacle because excludes the same function with obstacle, recharge station has other functions such as recharge the robot. For convenience, the project has some structs to represent the complex parameters of those entities, includes arena_entity_params (parameters: radius, pos and color) , arena_mobile_entity_params (parameters: radius, pos, color and collision_delta), home_base_params (parameters: radius, pos, color and angle_delta), and robot_params(parameters: radius, pos, color, battery_max_charge and angle_delta). Additionally, robot_motion_behavior can translates velocity and position to a new position. Robot_motion_handler manages the modification to the velocity based on user input and collisions. Robot_battery files can implement battery deletion for movement and collision.

Graphics Environment has a single window with robots, obstacles, home base and charging station. For convenience, we assume all objects in the window will be drawn as circles.

GUI and User Input consists UI buttons (Restart and Pause), left and right arrow keys (change the direction of the robot) and up and down arrow keys (change the speed of the robot (no reverse).

To meet the demand of three priorities, there is some event classes handled the whole project. Firstly I think there are 6 specific event classes we need to create: collision, command, decreasing battery, decreasing speed, keypress and recharge. Then we creating a event_base_class from which all events should derive from. Then those above 6 event classes inherit from event_base_class. The collision event class should be called when the collision happened, which can record the point of contact and angle of contact. The command event should be called when users give a command to the single view. The decreasing battery event should be called when the mobile entities hit a obstacle, which can decrease robot's battery. The decreasing speed event is similar to the decreasing battery event except decreasing robot's speed. And the keypress event should be called when the pressed a key, which would constraint user can only give some available command to robot. The recharge event should be called when robot hit the recharge_station, which can refill the battery's charge. Additionally, I need a sensor to detect whether something happened such as bump. Thus, I create sensor from which all sensors should derive from. For now, we only need a touch sensor to detect whether bump happened. There is a a representation of a touch sensor. Touch or "bump" sensors are "active" when pressed. For our purposes, imagine a series of bump sensors around the perimeter of the robot. A collision will activate the sensor at a particular point of contact, which translates to an angle of contact.