Robot Simulation

Software Design Document

Name: Gregory Star

Lab Section: Friday at 2:30pm

Date: 11/22/2017

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes Iteration2 of the Robot Simulation project. It is intended for both the author (myself) and also anybody who needs to inspect my code for grading purposes or otherwise.

1.2 Scope

This software is strictly for educational purposes, and is being constructed as part of an assigned project to help us learn about large code bases. It contains a fairly involved inheritance structure with hundreds (if not thousands) of lines of code.

1.3 Overview

This document explains the design of the project, and discusses certain problems and how they are solved. It also provides a description of the way in which components work together, and attempts to flesh out the structure of those components (inheritance).

2. SYSTEM OVERVIEW

In Iteration2, the Robot Simulation is designed as a game. There are circular entities within a rectangular arena. Some entities are static, but others move. Furthermore, some entities behave with some level of intelligence to fulfill certain goals, and one entity is controlled by the player. Mobile entities include SuperBots, Robots, HomeBase and Player. If a Player collides with a Robot, the Robot freezes. Robots get unfrozen when another Robot collides with them. They also send out distress signals to other Robots in the area and Robots react to those distress calls by moving towards them. The goal of the game is to freeze all regular Robots without running out of fuel. However, SuperBots freeze Player for a few seconds when they collide with Player, adding some challenge to the game. Furthermore, when Robots collide with HomeBase, they become SuperBots. If all Robots become SuperBots, you lose. Essentially the game plays much like freeze tag, with the added twist that the tagger (Player) can also be frozen.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

This program can be surmised by the interplay between Events, Entities, Sensors, and Arena. Arena is, as its name implies, is the container of all Entities. It's responsible for creating all Events and passing them down to their proper Entity. Arena passes information to Sensors via an Observer/Subject relationship, and Sensors then process that information and pass it up to their Entities (Entities have Sensors). Entities are the logical representation of objects that appear on screen. They are broken up into those that move (Mobile Entities) and those that don't (Immobile Entities).

GraphicsArenaViewer is responsible for actually drawing the Arena and its entities. It does this by getting periodic updates of the state of Arena. Its implementation is kept

separate from Arena, so that the visual and logic ends of the system interact only in controlled ways.

3.2 Decomposition Description

All Events inherit from EventBaseClass. These events used to describe only relatively simple things like collisions, keypresses, and recharges. However, as of Iteration2, events also describe things like distress calls, entity type, and proximity.

All Sensors inherit from Sensor, but there exist specializations (children) like SensorTouch, SensorProximity, SensorDistress, and SensorEntityType; each different sensor knows how to handle a different type of event.

ArenaEnitity is the base class of all Entities in Arena. Entities are broken up into ArenaImmobileEntity and ArenaMobileEntity. From here, Mobile Entities are further broken down into Robot and Player. SuperBot inherits from Robot. All Entities have a position and color (and a few other propertie), and mobile entities have additional properties like direction and Sensors.

3.3 Design Rationale

The rationale for the design was actually not up to us, but that's not to say that is isn't without merit. The idea that GraphicsArenaViewer should only be able to get state from Arena is a good one because it keeps Graphics separated from logic. Also, polymorphism is used to great effect by having all events inherit from one base class, all entities inherit from one base class, and all sensors inherit from one base class. It is up to interpretation whether or not an Observer pattern is appropriate in this project, especially since Sensors also accept Events. However, it is certainly a valid way to satisfy the purpose of a Sensor.

4. DATA DESIGN

4.1 Data Description

Data is read in through a text file and then Main constructs an Arena with all of the specified Entities. ArenaEntities are stored in Arena as a list. Color is represented as a struct with four integer values between 0-255 (red, green, blue, alpha). Each Entity has a Color attribute.

Information for the constructors of Entities is passed in via structs. There is a struct for every major class because otherwise the number of arguments to the constructor would be unmanageable. That being said, relatively simple classes like Sensor do not have such structs.

4.2 Data Dictionary

For brevity, only items new to this iteration will be mentioned here.

EventDistressCall: An event that gets passed to SensorDistress. When Arena passes this Event to the proper sensor, it first calculates the distance in question and then passes that information to the sensor.

EventTypeEmit: An Event that contains information about the type of an Entity. This is important for certain interactions between Robot, SuperBot, and Wednesday.

Sensor: A base class for all other sensors. It contains basic functionality to comply with the observer pattern (update()), as well as an activated() method for the Entity holding it to get its state.

SensorDistress: A Sensor which can detect EventDistressCall within a certain proximity.

SensorProximity: A Sensor that can detect an object when it's nearby.

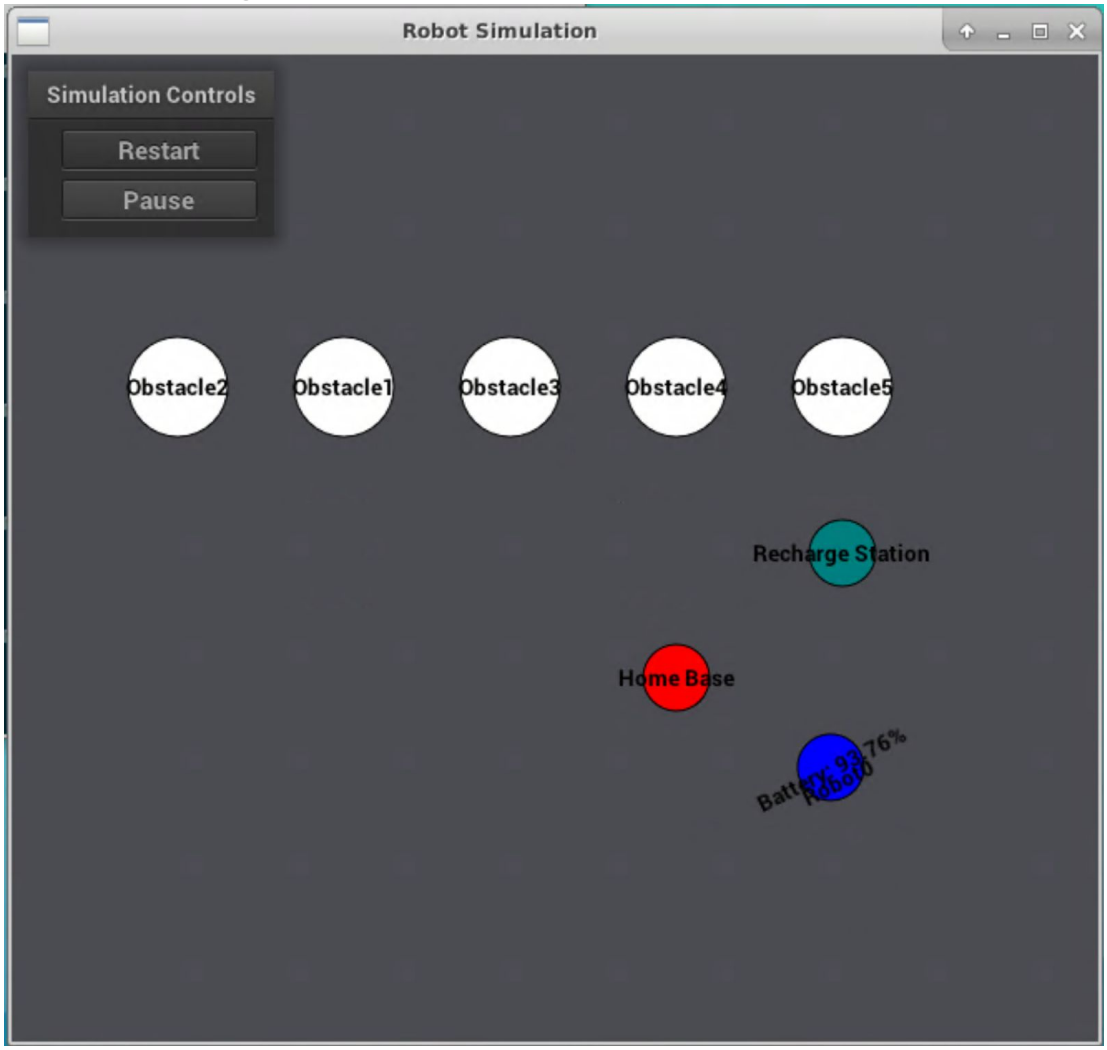
SensorTouch: A Sensor that detects when a collision has occurred and processes the information accordingly.

5. HUMAN INTERFACE DESIGN

5.1 Overview of User Interface

The user interface consists of a window with a grey background. There is a translucent panel on the top left of the window that contains two buttons. The buttons are labeled "Pause" and "Restart", which pause and restart the program respectively (when clicked). There are circles over various sizes and colors, which all have text labels, which represent the various entities. Some of them move, and others don't. The user can move the Player circle with the arrow keys.

5.2 Screen Images



A screenshot of the program as it was at the end of Iteration1. Entities are labeled with text and have different colors to help distinguish them. A control panel is in the top-left corner of the screen. In the final version, there will be many different kinds of entities on screen interacting in more complicated ways.