R. Jacob Schonthaler

schon099

Design Document

Overview

This program is one that simulates an arena that contains entities that interact with each other. The arena contains both immobile and mobile entities. Obstacles and the recharge station are the two types of immobile entities in the arena. The mobile entities consist of a home base, player, robots, and superbots. Robots and superbots contain sensors that assist sensors that they use to maneuver about the arena.

Immobile entities are scattered about the arena and cause the mobile entities to bounce off them upon collision. Immobile entities cause a loss of twenty units of charge to the player upon the player colliding with it. There can be any number of obstacles in the arena, but only one recharge station. The recharge station is a special obstacle that completely recharges the player's battery upon the player colliding with it.

The home base is a mobile entity that travels in a straight line and only interacts with other entities by colliding with them. The home base randomly changes directions as the program runs. If the home base collides with a robot, it causes the robot to become a superbot, and it and the superbot both bounce away from each other. The player loses 20 units of charge upon colliding with the home base, and both entities bounce away from each other. The superbots cause the home base and itself to bounce away from each other upon colliding with the home base. The home base just bounces off any immobile entity or wall without moving the immobile entity.

The player is a mobile entity that is controlled by the user. The user maneuvers the player by pressing the left, right, up, and down arrow keys to cause the player to turn left, turn right, speed up, and slow down respectively. If the player collides with any entity or wall in the arena other than the recharge station, it causes a loss of 20 units of charge to the player, and the player bounces off it. If the player collides with the home base, the two entities simply bounce off each other. The player gains full charge upon colliding with the recharge station. The player freezes a robot upon colliding with a robot. The player is frozen for a brief time if it collides with a superbot.

Robots are mobile entities that move around the arena avoiding the player and trying to collide with the home base. Upon colliding with the home base, a robot becomes a superbot. If a robot collides with the player, it freezes and sends out a distress signal until another robot or superbot collides with it. If a robot collides with another robot or superbot, they simply bounce off each other. A robot simply bounces off all immobile entities and walls upon colliding with them. A robot uses an entity type checking sensor and proximity sensor to aid in maneuvering about the arena. A robot receives distress signals via a distress sensor.

Superbots are very similar to robots with a few exceptions. If one collides with the home base, they bounce off each other. A superbot also freezes the player for a brief time upon colliding with it. Otherwise, robots and superbots are the same.

The arena determines when collision occur and when the game is completed. The arena fills out events and delivers them to the mobile entities to signal collisions as well as distress signals, entity types, and proximity to the sensors on the robots and superbots. The game is completed upon either the player runs out of battery which results in a loss for the player, or when the player manages to freeze all the robots which results in a win for the player.

Entity Details

Arena:

The arena contains only private variables. These pointers to a player, homebase, and recharge station named player_, home_base_, and recharge_station_ respectively. It also contains two doubles named x_dim_ and y_dim_ for x and y dimensions of the arena. Arena contains two unsigned ints for the number of robots and number of obstacles named n_robots and n_obstacles. There are two vectors named entities_ and mobile_entities which contain all the entities from each category respectively. Finally, arena contains a gameover_ bool to signal when the game is finished.

The arena contains getters for all the above variables and as well as a setter for gameover_. It contains a constructor which takes in arena_params struct. There's also an AdvanceTime function which takes in an unsigned int and updates the arena. The Accept function reads in key presses from the keyboard. Finally, the Reset function resets the arena.

Obstacle:

An obstacle contains only private variables. These include a position named pos_, a double named radius_, a Color named color_, and an int named id_.

The functions for obstacles are all public. One is a constructor which takes in a double, position, and color. The other is get_name which returns a string with the id appended to the end of obstacle.

RechargeStation:

The recharge station is exactly like an obstacle.

HomeBase:

The home base contains only private variables. It contains three doubles, a Color, an int, a position, and a touch sensor named heading_angle_, collision_delta_, radius_, color_, speed_, pos_, and sensor_touch_ respectively.

The home base contains only public methods. It contains a constructor which takes in a home_base_params. It also contains a get_name function which return the string HomeBase. TimestepUpdate updates the position of the home base and retruns void. The Accept function takes in an EventCollision and determines if a collision has occurred and what to do. The Reset function resets the home base and returns void. Getters for radius_ and color_ are present the home base. Getters and setters for speed_, heading_angle_, and pos_ are also contained in the home base.

Player:

The player contains only private variables. These include an unsigned int, an int, a position, a color, three doubles, a RobotBattery, a PlayerMotionHandler, a RobotMotionBehavior, and a SensorTouch named next_id_, id_, pos_, color_, radius_, heading_angle_, angle_delta_, battery_, motion_handler_, motion_behavior_, and sensor_touch_ respectively.

The player contains only public functions. There's a constructor which takes in a player_params. The ResetBattery resets the battery and returns void. The Reset function resets the player and return void. HeadingAngleInc and Dec functions which do just that and return void. TimestepUpdate which updates the position of the player and returns void. An Accept function that takes in a RechargeEvent that recharges the battery and returns void, and an Accept function that takes in an EventCollision which determines if a collision has occurred and what the course of action should be and returns void. The EventCmd function takes in a enum event_commands and adjusts the players heading or speed and returns void. Battery_level returns a double which is the battery level in units, and battery_loss which takes 20 units of energy from the battery and returns void. Getters and setters for speed_, heading_angle_, and pos_. Id returns the id of the player. Getters for color_ and radius_ are included. Get_name returns the string player appended by the id of the player. String_battery_level which returns the battery level as a string.

Robot:

All the variables in robot are private. These include an unsigned int, an int, a position, a color_, three doubles, a RobotMotionHandler, a RobotMotionBehavior, a SensorTouch, a SensorDistress, a SensorEntitiyType, and a SensorProximity named next_id_, id_, pos_, color_, radius_, heading_angle_, angle_delta_, motion_handler_, motion_behavior_, sensor_touch_, sensor_distress_, sensor_type_, and sensor_proximity_.

The functions in robot are all public. The constructor takes in a robot_params. The Reset function resets the robot and returns void. The HaeadingAngleInc and Dec functions do just that and return void. TimestepUpdate updates the position of the robot and returns void. One Accept function takes in an EventDistressCall, an EventTypeEmit, and an EventProximity and determines the robot's speed and heading before returning void while the other takes in an EventCollision, determines if a collision has occurred, and returns void. Getters and setters for heading_angle_, speed_, and pos_ are also included. Getters for color_ and radius_ are included. Id returns the id number of the robot. The get_name function returns the string robot followed by the id number.

Superbot:

The superbot is has all the same variables and functions as a robot. The only difference is that the Accept function that takes in three parameters produces different outcomes than robot for certain inputs.

Sensors

Touch:

The variables contained by SensorTouch are all private. They are a double, a position, and a int named angle_of_contact_, point_of_contact_, and activated respectively.

The functions for SensorTouch ar all public. The constructor doesn't take anything in. The Accept function takes in an EventCollision, determines if a collision occurred, and returns void. The Reset function resets the sensor and returns void. Getters and Setters exist for each variable.

Distress:

The variables for SensorDistress are all private. The two variables are activated_ and range_ which are int and double respectively.

All functions in SensorDistress are public. The constructor doesn't have any parameters. The Accept function takes in an EventDistress, position, and double, and determines if a distress call is sensed and returns an int 0 for nothing and int 1 for distress call sensed. The function Reset resets the sensor and returns void. There are setters and getters for all the variables.

EntityType:

The variables in SensorEntityType are all private. They are an int and a double named activated and range_ respectively.

The functions for SensorEntityType are all public. The constructor has no parameters. The Activated function takes in an EventTypeEmit, a position, and a double, determines if there is an entity within the range and, if so, what kind of entity, and returns an enum entity_types. The Reset function resets the sensor. There are getters and setters for the two variables.

Proximity:

The variables in SensorProximity are all private. These include an int and two doubles named activated, range_, and field_of_view_ respectively.

All the functions in SensorProximity are public. The constructor doesn't have any parameters. The Accept function takes in an EventProximity, a position, and a double, determines if there's an entity in the field of view, and returns the distance to the entity as a double, if so, else it returns -1 for no entity sensed. There are getters and setters for all of the variables.