Sensors and Events

The Sensor abstract class defines an int output, its setter and getter, and basic initialization and reset functionality that sets output to 0. It also have a pure virtual function “get_sensor_type()” which is implemented in each subclass by returning an enum. Sensors do not have pointers to the Entities which contain them.

There are four types of sensors, each with its own corresponding event class which it accepts once per timestep.

SensorTouch accepts EventCollision, which has fields for the point of contact, angle of contact, type of entity that was collided with, and whether the collision actually took place. If the collision took place, it stores all these variables locally, then sets its output to 1; otherwise, it sets its output to 0.

SensorProximity has a bool indicating whether it is on the left or right side of the robot which contains it, and doubles for the range and field of view. It accepts EventProximity, which has a bool indicating whether an entity is actually in view, and a double for the distance to the entity. If an entity is in view, SensorProximity sets its output to the distance stored in EventProximity; otherwise it sets its output to -1.

SensorEntityType has a double representing its range of view. It accepts EventTypeEmit, which has a bool indicating whether an entity is actually in range, and an enum for the type of the entity. If the entity is in range, SensorProximity sets its output to the entity type stored in EventTypeEmit; otherwise, it sets its output to an enum representing a null object.

SensorDistress has a double representing its range of view. It accepts EventDistressCall, which has a bool indicating whether an entity in range is actually emitting a distress call. If so, SensorDistress sets its output to 1; otherwise, it sets its output to 0.

Arena

The Arena has an enum that marks whether the game is running, paused, won, or lost. It has two component vectors, one for all entities and one for all mobile entities. It also has a vector of 2-tuples which each contain an entity and a sensor.

Upon construction, the Arena initializes all entities in its parameter list and adds them to one or both entity vectors. For each entity, it adds each sensor of the entity to the tuple list after pairing it with the entity.

Every timestep, the Arena scans to see if any Robots should be turned into SuperBots. If they should, the Arena removes the Robots from the entity and mobile entity vectors, removes their sensors from the tuple vector, and adds the Robots to a hidden vector of removed robots. It

then creates new SuperBots at the position of the old Robots. The old Robots and their component Sensors will not be referenced again until the Arena is reset.

Following this, the Arena checks if there are no more regular Robots remaining, or if the player has run out of battery, in which case the player loses. If a regular Robot exists, the Arena checks to see if all such Robots are frozen, in which case the player wins.

The Arena then iterates through every sensor tuple in the vector. The Arena, not the Sensor, decides whether the Sensor should be activated and with what it should be activated. (This is not consistent with the Observer pattern and I may change it later.) For SensorEntityType, SensorProximity, and SensorTouch, the Arena iterates through all Entities and walls and determines whether they should activate the given Sensor of the given Entity. For SensorDistress, the Arena simply iterates through all regular, non-removed Robots.

When the Arena is reset, all SuperBots and their component objects are destroyed, and their sensors are removed from the tuple vector. All removed Robots are re-added into the entity and mobile entity vectors, and their sensors are re-added into the tuple vector. After this is done, all entities and sensors are reset to their initial state.

Motion Handlers

The MotionHandler superclass has variables representing the heading and speed, setters and getters for them, and functionality to update velocity when given a SensorTouch. There are four subclasses, all of which have UpdateVelocity functions that take in all the respective entity's sensors.

MotionHandlerHomeBase checks to see if the direction is changed due to a collision; if not, it has a chance to change to a random direction.

MotionHandlerPlayer accepts EventCommands that change the Player's heading and speed. If it detects a collision with a SuperBot, it sets its speed equal to zero for a certain number of timesteps, then restores the old speed once those timesteps have elapsed.

If MotionHandlerRobot detects a collision with Player, it freezes; if it detects a collision with Robot or SuperBot, it unfreezes; if it detects a collision with HomeBase, it unfreezes and activates a bool that indicates the Robot should be made into a SuperBot. If the robot is frozen, one of the proximity sensors is active, no distress call is sensed, and the HomeBase is not nearby, it slows down as a linear function of its distance to the nearest object in view.

MotionHandlerSuperBot is a subclass of MotionHandlerRobot. It does not freeze and responds to proximity events in the same way, with the difference that it will not change direction if Player is nearby, rather than if HomeBase is nearby.

Entities

All entities are initialized with param structs, which they then save as private or protected const variables. When Reset is called on an entity, it re-initializes itself with the information in its struct.

All mobile entities - HomeBase, Robot, SuperBot, and Player - contain a pointer to their corresponding MotionHandler subclass. They also contain their component Sensors as private variables. ArenaMobileEntity lists SensorTouch and MotionBehavior as protected variables, since each mobile entity has one.

During the TimestepUpdate member function of each mobile entity, the MotionHandler subclass takes in pointers to the sensors and updates the entity's velocity, then the MotionBehavior updates the position from the new velocity.

The TimestepUpdate function of Player additionally checks if the SensorTouch detected a collision with RechargeStation, in which case it recharges the battery. If SensorTouch detected a collision with anything else, it depletes the battery by a given constant. In any case, it then depletes the battery by an amount linearly correlated to speed.