Robot Arena Simulation Design: Iteration 2

Jacob Elafandi

November 29, 2017

1 Overview

This project is a simple game played on a 2D arena which contains circular entities. The player has control over one such entity; all others move autonomously or remain immobile. Every mobile entity, and all its sensors, are updated once per timestep by a central Arena class.

2 Entities

2.1 Overview

ArenaEntity is a virtual class with two subclasses, ArenaMobileEntity and ArenaImmobileEntity, both of which are also virtual. Every entity in the Arena is an instantiation of one of these two subclasses.

Entities are circles of fixed radius at a specific position in the Arena, measured in Y-inverted Cartesian coordinates from the upper left. For example, if an entity was 200 pixels right of the left wall and 350 pixels below the top wall, its position would be (200, 350).

Every entity is initialized with a struct which contains all its parameters. This struct is then saved as a local variable, so that when the game is restarted, the entity can revert to its initial position, angle, etc. Every entity subclass with local variables not found in its parent class has its own corresponding parameter struct.

2.2 Entity Types

- **ArenaMobileEntity:** Abstract class. Moves in a straight line according to its heading angle, stored as a double in the appropriate MotionHandler. Has a SensorTouch for use in detecting collisions. Bounces off walls and other entities at angle of incidence.
  - **Player:** Has a battery that constantly depletes with movement, but is recharged by collision with the RechargeStation. Responds to arrow keypresses: up to accelerate, down to decelerate, left to turn counterclockwise, and right to turn clockwise. Cannot accelerate beyond a maximum speed. If it collides with anything, it will slow down and deplete its battery by fixed amounts. If it touches a SuperBot, it will freeze for a fixed number of timesteps before resuming its motion, and will not change direction.
  - **HomeBase:** In each timestep, if it did not bounce off of another entity or the wall, HomeBase has a chance to change direction to a random angle. If a Robot collides with HomeBase, the Robot is removed from the Arena and is replaced with a SuperBot.
  - **Robot:** Has a unique ID. Has two instances of SensorProximity, a SensorDistress, and a SensorEntityType in addition to the SensorTouch inherited from ArenaMobileEntity. Freezes upon touching Player, unfreezes upon touching another Robot or SuperBot, and is replaced with a SuperBot upon touching HomeBase. If frozen, emits a distress call that can be received by the SensorDistresses of other Robots. If a SensorProximity detects an entity or wall, the SensorDistress is not active, and the HomeBase is not nearby, the Robot will attempt to avoid the entity or wall: it will slow down as a linear function of its distance to the object and will turn to avoid it.
    - **SuperBot:** Subclass of Robot, created when Robot collides with HomeBase. Cannot be frozen; rather, Player freezes upon contact with it. Unlike all other Entities,

    is destroyed rather than reset when the game is restarted. Whereas a normal Robot does not avoid collision if HomeBase is nearby, a SuperBot does not avoid collision if Player is nearby.

- **ArenaImmobileEntity:** Abstract class. Cannot move. No special parameters.
  - **Obstacle:** Sole concrete instantiation of ArenaImmobileEntity. Has a unique ID.
    - **RechargeStation:** Unique instance of Obstacle. If Player collides with it, Player's battery is recharged.

2.3 Encapsulation Classes

- **MotionHandler:** Every subclass of ArenaMobileEntity contains a pointer to the corresponding subclass of MotionHandler: MotionHandlerRobot, MotionHandlerSuperBot, MotionHandlerPlayer, or MotionHandlerHomeBase. Each entity's MotionHandler stores information relating to its speed and heading angle. In every timestep, rather than updating its speed by itself, each entity passes its sensors to its MotionHandler, which does the updating.
- **MotionBehavior:** Stored by ArenaMobileEntity. In every timestep, each entity passes a pointer to itself to its MotionBehavior, which updates its position.
- **Sensor:** Covered in the next section.

3 Sensors

3.1 Overview

Sensors are objects that belong to an entity that measure events in the arena and cause the entity to adjust its heading accordingly. Each Sensor subclass corresponds to an event class, which is derived from EventBaseClass. Every Sensor has a function to accept its respective event from Arena and adjust its (integer) output to match.

The decision of whether an Event should affect the Sensor is made in Arena (using information from the Sensor), not in the Sensor subclass. Each Event has a field that marks its activity; inactive Events do not affect the Sensor. While this is in violation of the Observer pattern, it was judged that this cut down on runtime bloat: this way, no Sensor is given information that does not affect its output. (May change in Iteration 3.)

Entities have pointers to their Sensors, but not the other way around; this was done to reduce coupling.

3.2 Sensor Types

- **SensorTouch:** Has a floating point collision delta; if the distance between the entity that the SensorTouch belonds to and any other entity is less than the collision delta, the sensor should activate. Event class is EventCollision, which stores point of contact, angle of contact, and type of entity which was collided with. After accepting an active EventCollision, all three of its fields are stored in SensorTouch and can be retrieved via getters, and output is set to 1. After accepting an inactive EventCollision, output is set to 0.
- **SensorProximity:** Represented as one of two cones in front of a Robot (with a boolean indicating whether it's on the left or right). Floating point variables indicate the field of view (degrees from center that the Sensor can detect) and the range of view (distance from the entity that the Sensor can detect). After accepting an active EventProximity, output is set to the distance to the entity. After accepting an inactive EventProximity, output is set to -1.
- **SensorEntityType:** Detects the closest entity to a Robot within its range of view. Accepts EventTypeEmit. If the event is active, the sensor sets its output to an enum representing the type of the entity detected; if not, it sets its output to an enum representing no entity found.
- **SensorDistress:** Detects whether a Robot is frozen in a given range. Accepts EventDistressCall. Sets output to 1 if the event is active, and 0 if not.

4 Arena

4.1 General Parameters

Similar to an entity, Arena is initialized with a struct (arena_params). This struct contains the structs which initialize all entities, as well as the number of Obstacles, the number of Robots, the x dimension, and the y dimension. Upon construction, Arena stores the latter four as local variables, constructs all entities, and saves pointers to the Player, the RechargeStation, and the HomeBase. Arena also has an enum representing the game state (running, paused, won, or lost), which is initially set as running.

4.2 Entity Storage Vectors

Arena stores all its entities and sensors in four vectors, which are iterated over in the timestep update loop.

- **entities_:** A list of all instances of ArenaEntity currently in the Arena. Any Robots which have been replaced by SuperBots do NOT appear in this list, as no other Entities can interact with the original Robot. Instead, the new SuperBot appears in the list.
- **mobile_entities_:** A list of all instances of ArenaMobileEntity currently in the Arena. Similar to entities_, Robots which have been superseded by SuperBots do not appear in this list.
- **sensor_tuples_:** A list of 2-tuples. Each tuple contains a pointer to a sensor and a pointer to the entity which the sensor belongs to; each sensor of each entity corresponds to a tuple. This is done to prevent unnecessary coupling between entities and sensors, yet still give the Arena all information needed to update the sensor. When a Robot is replaced by a SuperBot, its sensors are removed from this list, and the SuperBot's sensors are added.
- **removed_robots_:** Initially empty, this list contains pointers to all Robots which have been replaced by SuperBots and thus removed from entities_ and mobile_entities_. These Robots are stored so that they can be re-added to those lists when the game is restarted.

4.3 Timestep Update

In its main loop, Arena does the following in order:

- Call the timestep update function on each entity.
- If any Robot has just touched the HomeBase, remove it from entities_ and mobile_entities_, remove all its sensors from sensor_tuples_, and add it to removed_robots_. Create a new SuperBot using a copy of the Robot's parameter struct, add it to entities_ and mobile_entities_, and add its sensors to sensor_tuples_.
- Check if the game has been won or lost (see section 5).
- For every sensor in sensor_tuples_: Create an event of the respective type, initially inactive. Iterate through all entities (and walls, where applicable) and check if any should activate the sensor; if so, activate the event and update its parameters. Have the sensor accept the event.

4.4 Keypress Accept

Independent of the timestep update loop, a function in Arena accepts an EventKeypress to adjust the speed or heading of the Player. The event is passed to Player, which in turn passes it to MotionHandlerPlayer, which adjusts speed or heading accordingly.

4.5 Reset

When the game is restarted, the Arena sets its state to running, then calls the Reset function on all entities (including removed Robots). Every SuperBot is removed from entities_ and mobile_entities_, and its sensors are removed from sensor_tuples_; the SuperBot itself is then deleted to free up memory. *(This may cause a memory leak with the sensors, motion handler, and motion behavior of SuperBot; something to look at in Iteration 3.)* Every Robot in removed_robots_ is removed from that list and added back into entities_ and mobile_entities_, and its sensors are added back into sensor_tuples_.

5 Win/Loss Conditions

- If the player's battery level is completely depleted, the game is lost.
- If there are no regular Robots remaining (i.e. all of them have been turned into SuperBots), the game is lost.
- If all remaining regular Robots are frozen, the game is won.

6 User Interface

The Arena is visualized through a class that contains it, the GraphicsArenaViewer. Using the cs3081 Simple Graphics Library, it draws a rectangular window which represents the Arena. Two clickable buttons in the upper left corner allow the game to be paused/unpaused or restarted; the latter button calls the Reset function in Arena.

Every timestep, GraphicsArenaViewer calls on the update function of Arena, which updates all entities. GraphicsArenaViewer then draws all entities in the arena, the proximity sensors of all Robots (represented as cones in front of the Robot; yellow for left, blue for right), and a translucent rectangle in the upper right representing the robot battery (which changes color and size as the battery is depleted). If a key is pressed, an EventKeypress is created and sent to Arena.