Iteration 3 Documentation

Technical Introduction

This program creates a robot simulation, in which an autonomous robot moves around a 2D world. The robot moves based upon it's behavior and hunger level. This software is written in C++, making the use of the OpenGL graphics library, as well as the nanogui library. Compilation requires the installation of the MinGfx library. This library creates the 2D graphics window that the arena lives in and animates both the entities and the environment using a continuous control loop.

Model-View-Controller

This software follows the model-view- controller(MVC) design pattern, which corresponds to three primary classes where the bulk of the code is held in: **Arena**, **GraphicsArenaViewer**, and **Controller**.

The Model in this implementation is the **Arena** class. The **Arena** manages all of the entities and their actions. For instance, methods that perform tasks such as removing and adding entities to the simulation are defined here, methods for controlling the movement are defined to some degree here, and there is a substantial method, UpdateEntitiesTimestep, which calculates the proper positioning and action after each time step. A time step in this context is an iteration through the game loop.

The **GraphicsArenaViewer** handles both user input and graphical display of the simulation. This is the part of the project which relies on the MinGfx library mentioned earlier as it inherits from the MinGfx class GraphicsApp. The **GraphicsArenaViewer** controls the timing of the simulation and the graphics display by using a loop that calls UpdateSimulation and DrawUsingNanoVG at each iteration.

The **Controller** class is the last of the three classes that is influenced by the MVC design pattern. This class acts as a connection between the **GraphicsArenaViewer** and the **Arena**. For example, if a user were to add a robot to the simulation they would alter the slider in the **GraphicsArenaViewer** which would in turn call a method of the controller class, which then would decide what method is best to call in the **Arena** class.

Entities

As mentioned previously there are multiple entities that exist within the simulation. The three entities are as follows: **Robot**, **Light**, and **Food**.

Robot is the main entity in the simulation. The robot autonomously moves based on two attributes: hunger level and behavior. If there is food in the arena that robot has a hunger level. The hunger level, which is based on a timer, is defined to be decremented within the robot's TimestepUpdate method. This implementation of the

software uses the dt or delta time to control the hunger timer and not the computer clock. Due to this implementation runtime fluctuates slightly between machines however the total time from no hunger to starvation is ~2 minutes and 30 seconds. The robot's **Food Sensor** which inherits from a parent **Sensor** class guides the robot towards the location of a **Food** entity. Based upon the hunger stage the robot is either only influenced by light, influenced by light and food, or only influenced by food. The other way the robot moves is in response to the light entities. The robot can have one of two behaviors: **Fear** or **Explore**. The **Fear** behavior makes the robot moves slow away from the light and move away quickly when it comes towards the light and the **Explore** behavior moves quickly when not near the light and moves away slowly when near the light in an effort to avoid them. The robot controls the response to lights by using its **Light Sensor** which much like the **Food Sensor** inherits from the same parent **Sensor** class.

The Robot's movement is handled using a combination of differential drive and Sensors. Differential drive is the concept that the robot has two wheels and it turns when one wheel has a higher velocity than another. Both Sensors act similarly just with different stimuli. The robot has two sensors placed 40 degrees away from its main focal point. With each time step, a Notify method notifying the sensor of its respective entities is called. This implementation is based on the Observer Pattern. The Notify method (which is a virtual method of the **Sensor** class is overwritten within in both the **Food** and the **Light Sensor** classes) calculates sensor readings based upon the robot's distance from other entities. The function used to do this is as follows: reading = 1200/(distance^1.08). This reading is then passed into an UpdateVelocity method which is part of the **MotionHandlerRobot** class. In the **MotionHandlerRobot** the Strategy design pattern is used to control the type of robot (fear or explore). This done by having a **Braitenberg** parent class as an attribute and defining the behavior at runtime. The UpdateVelocity method is defined according to the type of the robot. The definitions for these can be seen in their respective definition is **fear.cc** and **explore.cc**. Please note that there is an aggressive behavior for the food as well that is only used when the robot is starving.

The **Light** is the second entity. Lights move in a static fashion. They always move straight until they collide with a wall, at that point they move in an arc back in order to avoid never ending collisions. The Lights collide with walls and other Lights which is shown in the Arena's HandleEntityCollision method. The amount of Lights is handled by the sliders in the **GraphicsArenaViewer**.

The **Food** is the last entity in the simulation. The **Food** has no movement at all and is placed randomly within the arena. The robots consume food to reset their hunger level. There is a slider for food as well. If the slider is set to 0 the robot hunger will be shut off using a Boolean attribute of the robot class called has_hunger_. This attribute will be set to false and the hunger time will not be a factor if the arena has no food.

General Introduction

This robot simulations is used to exhibit the reaction of different types of robots to lights. The robots have a behavior, either fear or explore, and a hunger level. Robots also react to food in different ways depending on the hunger level.

Graphical User Interface (GUI)

The simulation is laid out in a simple fashion. In the far right of the screen there is a toggle menu. At the top of the menu there is a Play/Pause button. This button starts and stops the simulation much like a pause button on a video game. This is set to a default value of paused so Play must be pushed each time a new simulation is made. The button beneath it is a New Game button which resets the simulation by resetting all hunger levels and entity positions. Beneath that is an assortment of sliders to control various attributes of the simulation. In order they are: Number of **Fear** Robots, Number of **Explore** Robots, Number of Lights, Number of **Food**, and Sensitivity of the **Sensor**. All of these are self-explanatory as far as their actions, aside from the sensitivity slider. The lower the sensitivity the less the robots will react to other stimuli. If this is set to zero the robots will move in their default behavior (**Fear** robots will freeze, **Explore** robots will move at maximum velocity) [Please see the next section for descriptions of these behaviors].

The arena itself is held within the white outline seen on the screen. These white outlines act as the wall or boundaries of the simulation and none of the entities can spawn or move outside of them. The three entities - **Robot**, **Food**, **Light** – are colored in blue, red, and white respectively. The two dots seen on the robot are its sensor, which will be explained in further detail in the next section. The robots and lights are randomly sized within a certain range while the food is always the same size.

How the Simulation Works

The robots move autonomously based upon the behavior they exhibit and their hunger level. There are three stages of hunger: full (first 30 seconds), hungry (between 30 seconds and 2 minutes), and starving (between 2 minutes and 2 minutes and 30 seconds). Depending on which stage the robot is in it will react in different ways. If it is not hungry it will only react to the lights, if it is hungry it will react to lights and go towards food and if it is starving it will move only towards food and not to lights. If the robot goes 2 minutes and 30 seconds without consuming food the simulation will end and a “YOU LOST” message will appear in the middle of the GUI. At that point the New Game button can be pressed to restart.

The other way the robot moves is through its behavior. Robots can have two behaviors, **Fear** or **Explore**. The **Fear** behavior moves slowly away from lights and moves quickly away when coming close to lights. The **Explore** robot moves quickly in a straight line when away from lights and slowly turns away when close to lights.

Collisions are handled in multiple ways depending on the entity. Robots collide with other robots and the wall. Lights collide with other lights and the wall. None of the entities collide with food (food obviously does not handle any collisions as it is immobile). Every entity reverses in an arc when it registers a proper collision.