Designing for Interactive Spaces

Embedded Systems Capstone

Haoyuan Xia
University of Washington
Department of Electrical and Computer Engineering
Seattle, WA
kevin623@uw.edu

ABSTRACT

This project is dedicated to the E E 475 Embedded Systems Capstone course at the University of Washington. The purpose of this project is to create a “Creature” that is capable of sensing motion, making sounds, emitting light, and transmitting and receiving radio packets. Each “Creature,” created by each student group, serves as a node in a larger network. All “Creatures” obey a common set of rules, which guide the interactions between each “Creature.” They can mimic the emergent behaviors of birds or insects. One “Creature” receives an external stimulus and responds with light and sound. The response action stimulates neighboring “Creatures,” which respond accordingly. The action spreads out with decay and finally dissipates.

The “Creatures” can be deployed in a large public space for entertainment purposes. In this course, they were tested in the Microsoft Atrium in the Paul G. Allen Center for Computer Science and Engineering at the University of Washington. The testing result was successful.

Besides entertainment purposes, the wireless sensor network (WSN) in general has more applications. In this course, each node only has a motion sensor. For various purposes, other sensors, such as for temperature, light, sound, and pollutant, can be added. The WSN can be used for environmental monitoring, health monitoring, and home automation. Compared with traditional wired sensors, a WSN is more flexible as the position of each sensor can be easily changed at any time. It is also advantageous when each sensor is remotely separated, where wire connection is very expensive or even impossible.

KEYWORDS

wireless sensor network, WSN, embedded system, complex system, emergence, Creature, Flock, Arduino, Adafruit, Feather, M0, ARM, Cortex M0, radio, communication

ACM Reference Format:
Haoyuan Xia. 2018. Designing for Interactive Spaces: Embedded Systems Capstone. In *Proceedings of E E 475 (Embedded Systems Capstone).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This paper is delivered directly to the project supervisor, Bruce Hemingway. Its target readers include students, educators, as well as any individual or entity who is interested in embedded systems and wireless sensor networks.

This project focused on creating a “Creature” that was capable of sensing motion, making sounds, emitting light, and transmitting and receiving radio packets. Each “Creature,” created by each student group, worked as a node in a larger network. All “Creatures” obeyed a common set of rules, which guided the interactions between each “Creature.” They can mimic the emergent behaviors of birds or insects. One “Creature” received an external stimulus and responded with light and sound. The response action stimulated neighboring “Creatures,” which responded accordingly. The action spread out with decay and finally dissipated.

This project was done in groups of four. My group members included Muzhi He, Zhonghao Wang, and Tianhao Zhang (in no particular order). We were assigned with a group number 9.

2 RELATED WORKS

This “Creature” project was adopted from *The Flock: Mote Sensors Sing in Undergraduate Curriculum* by Bruce Hemingway et al [Citation]. In the article, the authors described the significance of integrating this project in an undergraduate course. The authors also included how to implement each “Creature” by milestones and what to expect from the final product. Different from the “Flock of Birds” project, the “Creature” project used Feather M0 with ATSAMD21G18 [Citation], rather than MICA2DOT with ATmega128 [Citation]. LED rings were added for some visual expression. The piezoelectric transducers were also replaced by higher-power speakers for better sound quality.

The light gestures used in this project were inspired by the public event *BOREALIS, a festival of light* [Citation]. This event was hosted by *Comcast* in Seattle from October 11 to October 14, 2018. During the event, decorative lighting was installed in public spaces. Some light effects included rainbow color, rotating wheels, and lightning. The light could also interact with people. When people were not around, the effects looked slow and calm. When people passed by, however, the effects became more active, usually with very fast and sudden transitions. These patterns were referred to when we designed our own light effects for the “Creatures.”

3 SOLUTIONS

An individual “Creature” includes the following components:

- Adafruit Feather M0 RFM69 Packet Radio

- Adafruit Music Maker FeatherWing
- Adafruit NeoPixel Ring with 16 RGBW LEDs
- HC-SR505 Mini PIR Motion Sensor
- Ultralife UBP005 Lithium Ion Rechargable Battery, 3.7V
- Switch
- Speaker

The components were assembled and soldered according to the instruction provided by Bruce Hemingway [Citation]. Test programs were provided along with the instruction to verify all the physical connections were proper.

The software was developed in Arduino IDE with C++. We were building on top of the provided starter code.

Each "Creature" was a finite state machine (FSM) with predefined states, namely one "startle" state, three "active" states, and three "ambient" states. We used a `State` class to represent a state. The specific states were inherited from the `State` class, each of which had a unique name and ID. The seven states were represented by `Startle`, `Ambient1`, `Ambient2`, `Ambient3`, `Active1`, `Active2`, and `Active3` respectively. The "Creature" entered the "startle" state when there was an external stimulus, either from a person walking by, or from neighboring "Creatures." This happened no matter which state the "Creature" was in. Also, the state transition logic was the same for all states. To achieve this, the functions related to startling and transition were implemented in the `State` class. The actions and properties that were unique to each state were implemented in the seven subclasses. For example, each state had different light and sound effects. Therefore, each subclass called different functions from the Neopixel and Midi controllers. In general, light and sound effects for the "ambient" states were more gentle than the ones for the "active" states. The effects for the "startle" state were extremely exciting.

The state transitions had two different logic, one for the "startle" state and the other for the other six states. The startling, which had a higher priority than the other transitions, was triggered either by the PIR sensor or by receiving a startling packet (with some decay) from neighboring "Creatures." A "Creature" was actually startled only when it crossed the defined constant texttttSTARTLE_THRESHOLD. After a "Creature" startled, it broadcast a startling packet to other neighboring "Creatures."

The transition logic between the other six states were more complicated. To determine which state to enter into, the following factors were considered:

- The number of "Creatures" in the network that was currently in that state
- The "Creature's" relative likelihood of entering other states based on the its current state
- The relative likelihood, which the other "Creatures" in the network thought, of entering other states based on the "Creture's" current state
- The "Creture's" distances with other "Creatures"

The "Creature's" relative likelihood of entering other states based on the its current state was known as the *local weights*; the relative likelihood, which the other "Creatures" in the network thought, of entering other states based on the "Creture's" current state was known as the *global weights*. The global weights were set in the `State` class and the local weights were set in each subclass. Each group in the class would have different weights. For our group, the likelihood of next state being the same state as the current one (e.g. from Active 1 to Active 1) was the lowest, with a number of 1. The likelihood of entering the same type of state (e.g. from Active 1 to Active 3) was the highest, with a number of 4. The likelihood of entering the other type of state (e.g. from Active 1 to Ambient 1) was in the middle, with a number of 2. For the global weights, we used 4 for the ambient states and -2 for the active states, meaning our "Creature" was more likely to enter a ambient state than a active state. The likelihood of different states of the same type (e.g. from Active 1, Active 2, and Active 3) was the same.

The communications between each "Creature" was achieved by sending and receiving packets using the 900 MHz RFM69 radio. Each packet contained (in order) a packet ID (`uint8_t pid`), a source address (`uint8_t srcAddr`), a destination address (`uint8_t destAddr`), and a payload with a variable length (`uint8_t *payload`). The packet ID could take the following meanings: set globals, stop, start, play sound, play effect, broadcast states, startle, and send state. The "Creature" specific packets, namely startle and send state, were the key for creating emergent behaviors. These two packets had influence on the behaviors of the other "Creatures" and caused some patterns in the network that were not present on any single "Creature."

The other parts of the software implementation were trivial and were not discussed in this paper.

At the end of this course, all the "Creatures" were deployed and demonstrated in the Microsoft Atrium in the Paul G. Allen Center for Computer Science and Engineering at the University of Washington.

4 CONCLUSIONS

This project was finished in 10 weeks, among which 4 weeks were for hardware setup and 6 weeks were for software implementation. My contribution to the project included assembly and soldering of one "Creature." I implemented the seven subclasses of the `State` class. I also implemented the following functions:

- `Creature::_rx(uint8_t pid, uint8_t srcAddr, uint8_t len, uint8_t *payload, int8_t rssi)`
- `Creature::_updateDistance(uint8_t addr, int8_t rssi)`
- `Creature::_rxStart(uint8_t len, uint8_t *payload)`
- `Creature::_rxBroadcastStates(uint8_t len, uint8_t *payload)`
- `State::txStartle(uint8_t strength, uint8_t id)`
- `State::rxStartle(uint8_t len, uint8_t* payload)`
- `State::PIR()`
- `State::startled(uint8_t strength, uint8_t id)`

Besides, I was maintaining our group's repository on GitHub.

The factor that we cared most about was the creation of emergent behaviors. A emergent behavior is a behavior that is not present in any individual but appears in the system due to interactions. This project was very successful in creating such behaviors. Rather than the entire network, only individual "Creatures" were programmed. However, they created some interesting patterns as a system. For example, when one "Creature" was startled, its neighboring "Creatures" were also startled. The startling kept spreading out. This effect was successfully tested during the demonstration in the Microsoft Atrium.

This project can be improved by including a casing for each "Creature" for aesthetics and durability, rather than have the circuit boards and wired exposed outside.

Besides entertainment purposes, the wireless sensor network (WSN) in general has more applications. In this course, each node only has a motion sensor. For various purposes, other sensors, such as for temperature, light, sound, and pollutant, can be added. The WSN can be used for environmental monitoring, health monitoring, and home automation. Compared with traditional wired sensors, a WSN is more flexible as the position of each sensor can be easily changed at any time. It is also advantageous when each sensor is remotely separated, where wire connection is very expensive or even impossible.

ACKNOWLEDGMENTS

I would like to thank Bruce Hemingway for designing and supervising this project. I would like to thank Austin Beaulieu and Ryan Rowe for providing the starter code for the entire class. I would also like to thank Ying-Chao (Tony) Tung and Parker DeWilde for technical support. Finally, I would like to acknowledge my group members for their contributions to make this project possible.

REFERENCES