EEC 172 Final Project: Creature Catch

Heather Howe and Darius Ehsani
College of Engineering - University of California, Davis
Davis, CA
hhowe@ucdavis.edu, dsehsani@ucdavis.edu
8 June 2025

I. Project Description

Creature Capture is an embedded game system developed to provide users with an engaging experience of character exploration and virtual creature collection. Inspired by games like Pokémon, the system allows players to create a character using an IR remote, and attempt to capture hidden creatures through accelerometer-based motion input.

At the start of each game session, players can either create a new character or select a previously saved one. The OLED display will then display the user's character in which they can navigate the environment using directional input from the IR remote. When a hidden creature is encountered, a capture screen is triggered, allowing the user to perform a throwing motion with the CC3200's onboard accelerometer to simulate capturing the creature with a creature ball.

Captured creatures are stored in a temporary in-game collection. Upon exiting the map, players are prompted to save their data, which is uploaded via AWS IoT and stored remotely, enabling future retrieval. If players opt not to save, the data is discarded.

The system integrates several key components: IR signal decoding for remote input, OLED rendering for UI feedback, accelerometer-based gesture recognition, and cloud communication through AWS for consistent data storage. A future goal of multiplayer trading is outlined, with plans to enable cloud-based creature exchanges between users.

By combining embedded hardware with cloud infrastructure, Creature Capture offers an interactive, persistent gaming experience built entirely on resource-constrained hardware without requiring any new materials.

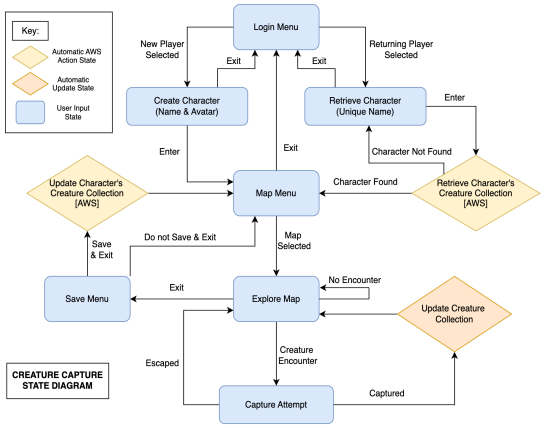

Fig. 1. **Creature Capture State Machine**

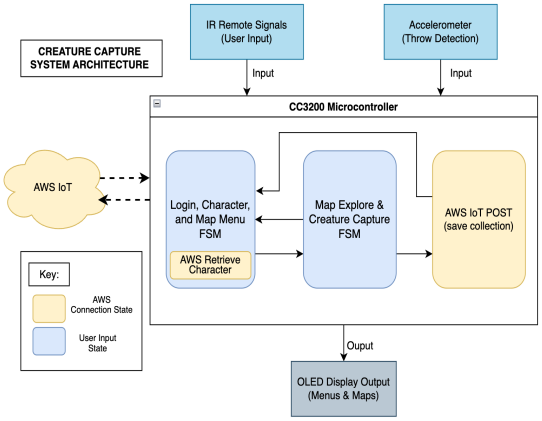

Fig. 2. **Creature Capture System Architecture Diagram**

II. Design

A. Functional Specification

The high-level behavior of Creature Capture is defined by the finite-state machine (FSM) shown in **Figure 1: Creature Capture State Machine Diagram**. Starting at the **Login Menu**, the player has a choice to create or retrieve a character (via AWS IoT). Once in the **Map Menu**, selecting a map transitions to **Explore Map**, where IR-remote inputs move the character on the OLED. If the avatar's position matches a hidden creature, the FSM enters **Capture Attempt**, where the accelerometer will detect a user's "throw". A successful capture moves to the **Update Create Collection**; failure returns to **Explore Map**. Exiting at any point brings up the **Save Menu**, where choosing **Save & Exit** triggers **Update Character's Creature Collection** over AWS (or **Discard** rolls back to the last save), then returns to the **Login Menu**.

B. System Architecture

The block diagram in **Figure 2: Creature Capture System Architecture Diagram** illustrates how the components' inputs and outputs feed into each other. User inputs arrive via

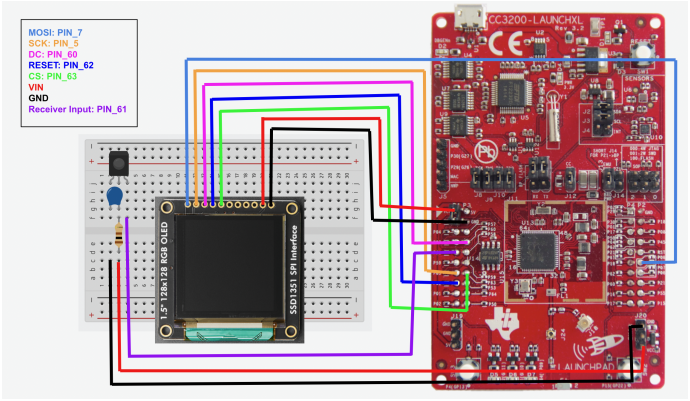

Fig. 3. **Circuit Schematic**

Fig. 4. **Tile Designs**

the **IR Remote** and **Accelerometer** into the **TI CC3200 board**, which runs two FSM modules (Login/Map Menu and Explore/Capture). All menus and maps render on the **OLED Display**. When loading or saving character data, the board communicates over WiFi with **AWS IoT Core**, to send an email updating the user on their Creature Collection.

III. Implementation

The implementation of Creature Capture uses a FSM that coordinates the flow of game play through several interconnected logic blocks. These blocks interface with external I/O devices—including an OLED display, an IR remote receiver, and an I2C accelerometer—to enable responsive user interactions and immersive game mechanics. The sections below describe the detailed implementation of each major game state and its associated hardware interactions.

A. Login/Map Menu

This stage handles user login through character creation or character retrieval. In this state, the user will enter their character name using the IR remote, providing initial access to the game. For character typing on the IR remote, the multi-tap technique was used. To implement this the IR receiver outputs the unique signals from the IR remote corresponding to each number on the keypad to the CC3200 Launchpad, where the pulses are measured and the signal is decoded using a timer and falling-edge GPIO interrupt. To distinguish between single and multi-tap use, a 2-second buffer was tracked to allow that much time for the user to tap the same button twice. The button used for our selection implementation was the MUTE button that the user pressed to indicate the completion of input. This would also be an indicator to change states and trigger the AWS post containing the character name, which updates the device shadow. The LAST button could also be used to delete any undesired character in this process. The other hardware involved in this implementation is the OLED display screen that uses the SPI protocol and displays the login prompt and real-time character input. This was implemented using a print character function created to print text to the center of the screen.

B. Explore Map

The map is composed of a 32×32 grid of tiles, each 16×16 pixels in size. We designed six tile types—grass, tree, bush, water, rock—and the player sprite in Piskel.com to ensure correct dimensions. We then converted each image into plain byte arrays using `image2cpp` by selecting "Plain bytes" output, "Horizontal" draw mode, and 2 bytes per pixel (RGB565). Each array was stored in `tiles.c`, where we initialized and labeled the data with its hexadecimal values, and defined a `tile_set` array that maps types to numeric identifiers (0: Grass, 1: Bush, 2: Water, 3: Tree, 4: Rock).

In `map_bitmap.c`, we created a two-dimensional array matching the map's width and height, populating it with those identifiers to define the layout. In `main.c`, we implemented four functions for map traversal:

- `drawTile(x, y, tile_id)` renders a single 16×16 tile at the given OLED coordinates.
- `drawMap(width, height)` iterates over the array defined in `map_bitmap.h` to draw the entire map.
- `drawPlayer(x, y)` draws the player sprite (tile_id 5) at the specified tile coordinates.
- `erasePlayer(x, y)` restores the background tile at those coordinates.

In the main loop, we track the player's position using `player_x` and `player_y`. When an IR-remote input is detected, the code calls `erasePlayer`, updates the coordinates based on the movement direction, and then calls `drawPlayer` to render the avatar in its new position. This implements the complete map and tile logic for character navigation.

C. Capture Attempt

The Capture Attempt phase is initiated when a character moves to a location on the map that contains a hidden creature. When the user encounters a creature, they are prompted to attempt a "throw", which is a quick toss motion of the CC3200 launchpad, to potentially capture the creature. This tossing motion uses the CC3200's onboard accelerometer (via I2C interface) to track the acceleration produced by the user. Once the user is prompted to throw, the accelerometer continuously samples the x-axis data to detect a negative acceleration that corresponds to a throwing motion. Data is polled in a loop and stored in an array over 200 iterations. During this data collection phase, if the x-axis value dips below 0, it is interpreted as the beginning of a throw. The strength of the throw is then categorized based on thresholds: a small throw (x $>$-20) scores a 1, a medium throw (x $<$-20 and x $>$-50) scores a 2, and a strong throw (x $<$-50) scores a 3. After the data array is filled, the program computes the max throw score to determine throw quality. Depending on the score, the OLED screen provides immediate feedback: "Bad Throw" with "ESCAPED" if the score is 1, or "Good Throw" or "Great Throw" followed by "Creature Caught" if the score is higher. Captured creatures are added to a running JSON array managed in memory by string manipulation functions. Finally, an HTTP POST request is issued to update the AWS IoT device shadow with the new game state, including the captured creature. This mechanism ensures persistent game progress synced with the cloud.

D. Exit and Save

The Save and Exit functionality allows the user to gracefully end gameplay while preserving all progress, including character information and captured creatures. When the user presses the '0' button on the IR remote during map navigation (i.e., when not in an encounter), the program interprets this as a request to save and exit. Upon receiving this input, the CC3200 compiles the current game state into a JSON string, which includes the character name and the complete array of captured creatures. This string is then used to construct an HTTP POST request, which is sent to the AWS IoT cloud via the device's configured endpoint. The device shadow on AWS is updated with this final state, ensuring that the user's progress is retained and can be accessed at a later time. Once the data has been successfully posted to AWS, the OLED screen will display the previous game state and wait for user input.

IV. Challenges

During development we encountered several challenges and devised strategies to address them:

- **IR Remote Decoding.** We initially aimed to support both arrow-key navigation and numeric input from a reprogrammed AT&T remote. However, we discovered the bit lengths for arrow codes and numeric codes differed significantly, requiring two separate decoding schemes. To simplify, we restricted input to the numeric keys only, then repurposed those keys contextually at each stage (e.g., text entry, menu navigation, character movement).
- **Map Design and Memory Constraints.** Our first design used a full 512×512-pixel map, but storing that as a single bitmap exceeded the CC3200's RAM. We reduced the map to 256×256 pixels, but embedding even that as a raw hex array generated over 10,000 lines of data—still too large. Switching to a tile-based approach (16×16-pixel tiles in a 32×32 grid) cut memory usage dramatically while preserving a large playable area.
- **Player Movement Rendering.** When the player moved, drawing the new sprite would overwrite the previous background tile. To prevent visual artifacts, we implemented an `erasePlayer(x,y)` function that redraws the original tile at the old coordinates before rendering the player at the new position.
- **Time Constraints.** Due to project timeline limits, our AWS IoT integration was scoped to basic collection updates only; more advanced features (e.g., full two-way parameter sync or a viewport-scrolling map larger than the OLED display) remain for future work.

V. Future Work

While Creature Capture successfully demonstrates a basic embedded game on the CC3200 platform, several enhancements would greatly improve both user experience and system performance:

- **Viewport Scrolling.** Implement dynamic tile loading and viewport offsets so the player can explore maps larger than the 128×128 OLED window without exhausting RAM.
- **Peer-to-Peer Trading.** Add a secure trading protocol over AWS IoT Core that allows two devices to negotiate and exchange captured creatures in real time.
- **Avatar Customization.** Introduce a character-creation menu where players can choose sprites, colors, and names, persisting those selections in the device shadow for long-term storage.
- **Additional Map Themes.** Provide multiple downloadable map layouts—forest, cave, desert, etc.—either preloaded in flash or fetched via Wi-Fi to extend replayability.

Beyond feature additions, we will explore memory optimizations (e.g., tile-data compression, partial framebuffer updates) to increase frame rates and reduce power consumption. Finally, a full two-way AWS shadow interface could enable remote monitoring of game progress and even leaderboard integration, paving the way for a more connected, multiplayer experience.

VI. Bill of Materials

TABLE I
BILL OF MATERIALS FOR THE CREATURE CAPTURE PROTOTYPE

Component	**Quantity**	**Total Price ($)**	**Distributor**
TI CC3200 Launch-XL	1	55.00	Texas Instruments
Breadboard	1	N/A	Local electronics retailer
Generic AT&T U-verse Remote	1	32.00	Amazon
IR Receiver	1	N/A	Class supplies
RC Network	1	N/A	Class supplies
Adafruit 128×64 RGB OLED	1	39.95	Adafruit