Arduino Roguelike

DESIGN SPECIFICATION

ED DIBBS

UNR - CPE 481: EMBEDDED GAMES DEVELOPMENT

November 3, 2015

Contents

Game Overview

Concept

The Arduino Roguelike features a nameless alchemist who is forced to delve into various dungeons in order to obtain the reagents required for a special brew. Along the way, the alchemist will encounter numerous monsters that stand in their way.

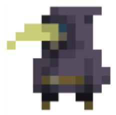


Figure 1: The current sprite for the alchemist protagonist.

Genre

The Arduino Roguelike is a single-player roguelike game. The roguelike genre is characterized by procedurally generated levels, difficult gameplay with permanent death, and tile-based graphics. Specifically, the Arduino Roguelike is modeled after the Binding of Isaac series by Edmund McMillan, as seen in Figure 2 below.

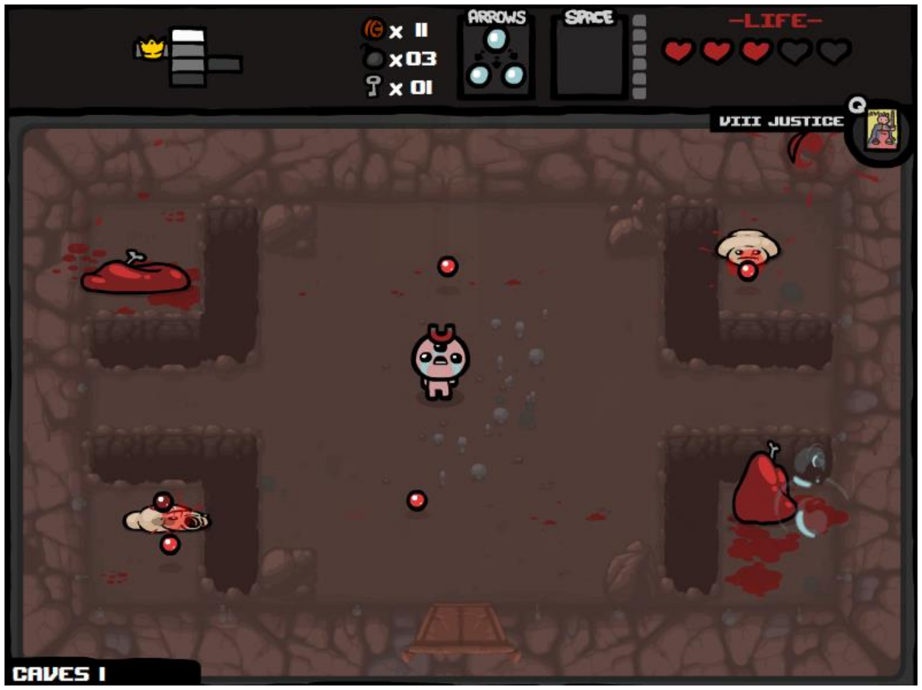


Figure 2: A screenshot from the game The Binding of Isaac. The game features procedural level generation and randomly chosen power-ups for the player.

Target Audience

Due to its platform, the Arduino Roguelike is not expected to be played by many people, though it has been developed to be appropriate for an audience of ages 10 and up. The game may prove to be difficult and frustrating for younger players, however, given its random difficulty curve.

Gameplay and Mechanics

Gameplay

The player controls an alchemist who must delve deeper and deeper into various dungeons in order to obtain the items they need to concoct a special brew. As the player progresses through each level, they have a chance of getting random power-ups that increase their offensive and defensive capabilities. Each level will feature tougher and smarter enemies that will challenge the player.

The objective of the game is to get all of the items needed to concoct the main character's brew. Due to its procedurally generated levels and random item drops, the game will vary in difficulty based on the players luck. This randomness will provide the player with a different experience for each play through. If the player runs out of health, the game will end and there will be no checkpoints or restarting. Instead, the player will have to start a new game if they wish to continue playing. This form of permanent death is iconic of the game's genre.

Mechanics

Movement

The player moves by adjusting the left joystick. Subtle movements will move the character slightly while full movements will have the player run at an increased speed. The player needs to move around in order to pick up items, move to different rooms, or to position themselves for attacking.

Objects

Currently, there are no objects in the game besides the player and enemies. In the near future, the player will be able to interact with doorways, power-ups, and health pickups. Moving to a doorway will move the character to an adjacent room. Power-ups will provide the character with a permanent modification to their offensive or defensive capabilities. Heart containers will restore some of the player's health when picked up.

Combat

By adjusting the right joystick, the player is able to attack. Initially, the player attacks by throwing test tubes at regular intervals. As the game progresses and power-ups are collected, the player will be able to throw larger and more powerful projectiles. Enemies that collide with the player's projectiles are injured and, upon reaching 0 health points, they will die. Once all enemies in a room are killed, the player will be able to advance to a different room.

Levels

Each level will consist of a collection of rooms. Each room can contain a power-up, some monsters, or nothing at all. One room deep in the level will contain a boss monster that has increased health and improved AI. Once the boss is defeated, the player will be able to advance to the next floor, where more difficult monsters and better treasures await.

Interface

The interface for the Arduino Roguelike consists of a main playfield with an overlaying heads up display (HUD). Currently the player's health and some debug information is shown on the HUD (as seen in Figure 3). By the end of its development, the game's HUD will display the player's health, a mini-map showing the current level's layout, and the player's active power-ups.

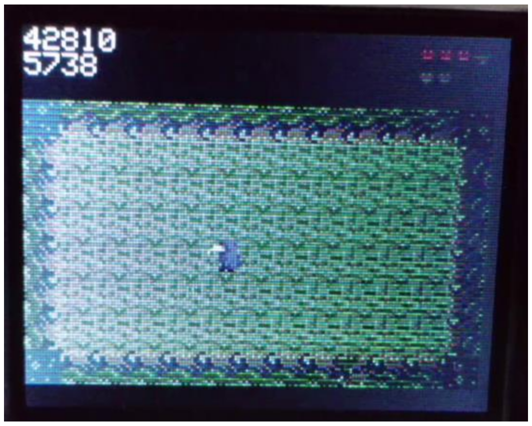


Figure 3: A screenshot from an early development of the game's interface.

Technical

Target Hardware

The target platform for the Arduino Roguelike is the Arduino MEGA 2560. The decision to program for the Arduino drastically limits the accessibility of the game, but the platform was chosen to provide for a greater challenge during the development process.

The Arduino MEGA has a 248KB of usable flash memory and only 8KB of RAM. It also only has a 16MHz clock speed, so the code needs to be optimized to provide for smooth gameplay. All of these constraints can prove to be frustrating to work with, but they make the final product all the more impressive.

Development hardware and software

The hardware for this project includes:

1. One Arduino MEGA 2560
2. One Adafruit 160x128 pixel TFT display (part ST7735)
3. Two Parallax 2-axis joysticks
4. One breadboard
5. One ABS project enclosure
6. One 9v battery and 2.1mm plug

A picture of all the hardware wired up can be seen in Figure 4.

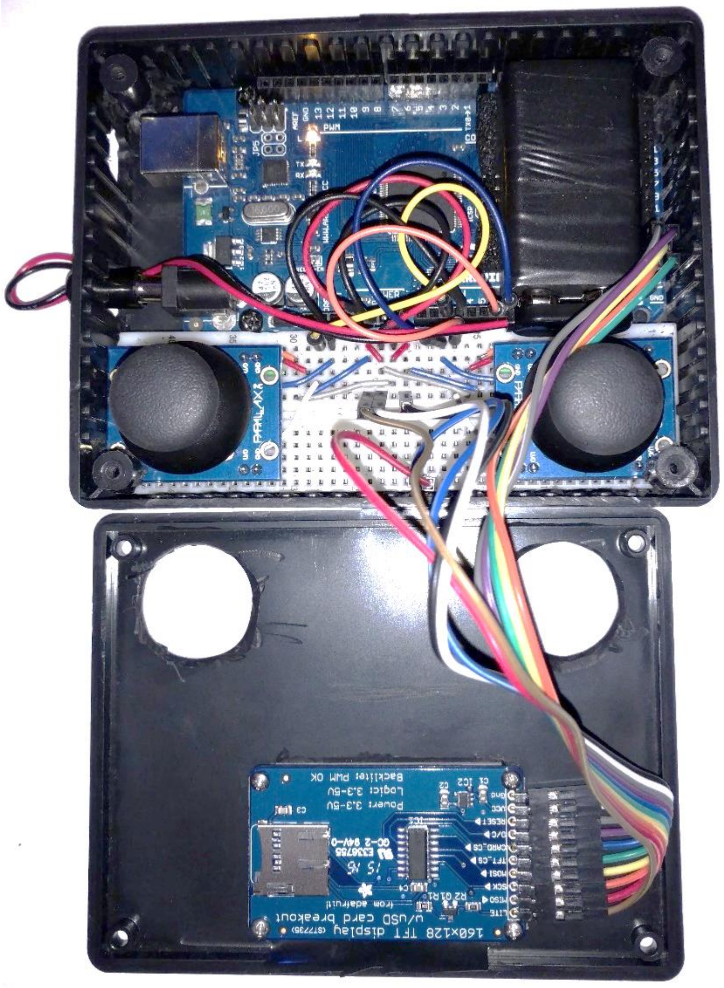


Figure 4: The internal hardware that the game is composed of.

The schematic for the electrical components can be seen in Figure 5.

For software, this project utilizes the Arduino IDE for programming, compiling, and downloading all the code. Additionally, a custom Python script was written to convert art assets into the format required to be displayed on the LCD screen. This is explained in further detail in the Game Art section below.

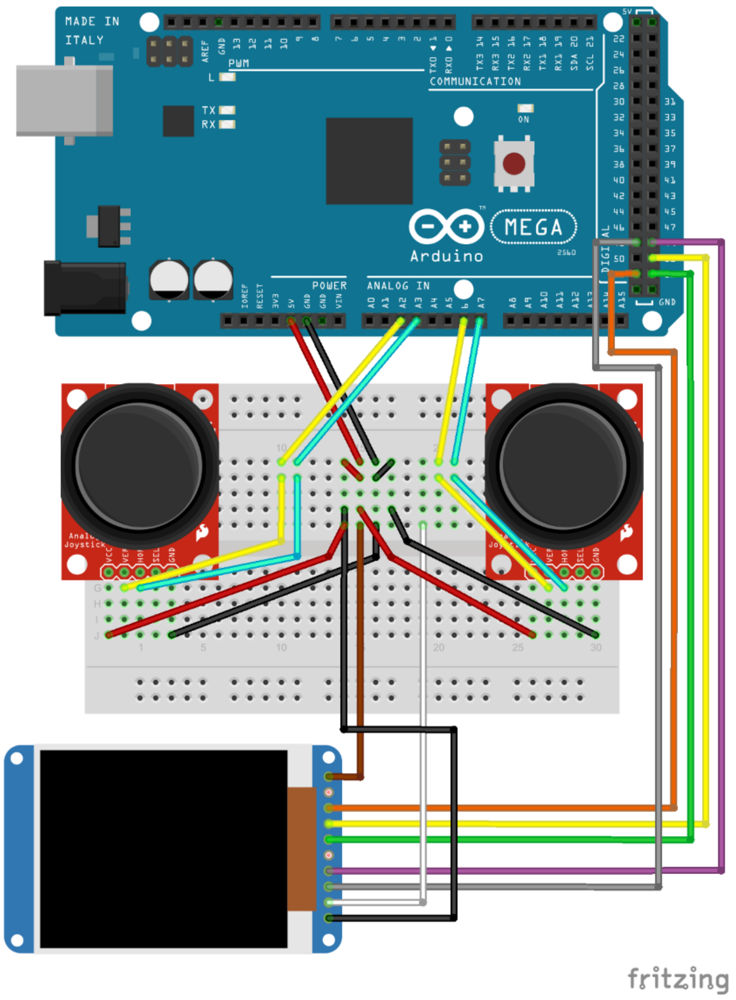


Figure 5: The schematic for the all the hardware involved, produced with fritzing.

Game Art

All art assets for the game have been custom created by my friend Melanie Lee. She produces the art assets and single frame png images with 8 bits per color channel and transparency.

The LCD Screen that displays all the graphics is an Adafruit 1.8" TFT Display (part no. ST7735). Adafruit uses a single 16-bit unsigned integer to interpret color, where the red and blue channels are represented by the first and last 5 bits, while the green channel gets the remaining 6 bits. The green channel gets an extra bit because human eyes are more sensitive to shades of green. This encoding can be seen in Figure 6. Because sprites need to support transparency, the color magenta is used to denote transparent pixels. Magenta is represented as full red and blue pixels, which corresponds to 0xF81F in hexadecimal.

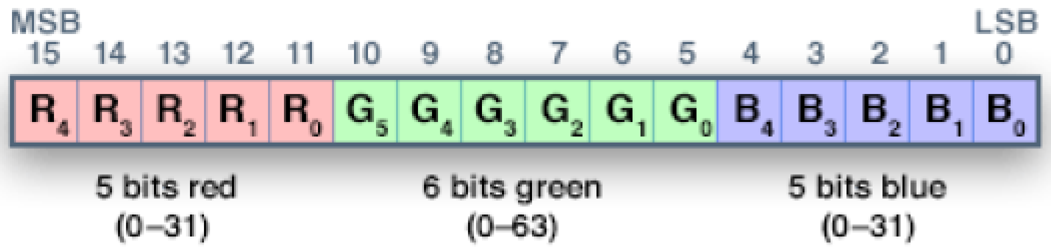
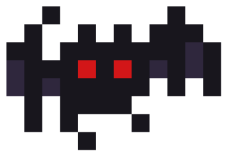


Figure 6: The bit encoding for a single 16bit color pixel for an Adafruit LCD screen.

To store the art assets in code, I developed a custom Python script, utilizing the PyPNG module, which loops through a directory recursively, loads each art asset, and then loops through each pixel in the image and converts the 3 8-bit channels into one 16-bit integer. These pixel values are stored in an array and saved as a header file that I can include in my project. A sample sprite and its code representation can be seen in Figure 7 below.



```
const PROGMEM uint16_t BAT[] = {12, 8,
63519,63519,8453,63519,63519,63519,63519,63519,63519,8453,63519,63519,6
3519,8453,63519,63519,8453,63519,8453,63519,8453,8453,8453,63519,8453,8
453,63519,8453,8453,8453,8453,8453,8453,8453,12679,8453,12679,8453,8453
,8453,35076,8453,35076,8453,12679,8453,12679,8453,12679,8453,12679,8453
,8453,8453,8453,8453,63519,8453,63519,8453,63519,8453,63519,8453,8453,8
453,8453,63519,63519,63519,63519,63519,63519,8453,63519,8453,63519,6351
9,63519,8453,63519,63519,63519,63519,63519,63519,63519,63519,8453,63519
,63519,63519,63519,63519,63519,63519};
```

Figure 7: A sprite of a bat and its text representation after being processed by my custom python script.

The intended art style for the game is a retro style that resembles games from the early to mid-90s.