Hardware Design Final Project

**FPCAT – Battle Cat on FPGA**

Team 01

112062122 Chia Chin Hsieh 112062144 Sheng Wei Fan

Table of Contents:

[1. Introduction 1](#_Toc185795950)

[1.1 Motivation 1](#_Toc185795951)

[1.2 Overview 1](#_Toc185795952)

[2. Game Scene Architecture and Implementation Details 2](#_Toc185795953)

[2.1 Start and Menu Scene 2](#_Toc185795954)

[2.2 Play Scene 3](#_Toc185795955)

[2.3 Win and Lose Scene 4](#_Toc185795956)

[3. Game Material Design 5](#_Toc185795957)

[3.1 Static Object Design 5](#_Toc185795958)

[3.2 Character Design 6](#_Toc185795959)

[3.2 Stage Design 6](#_Toc185795960)

[4. Game Engine 7](#_Toc185795961)

[4.1 Storage Protocols 7](#_Toc185795962)

[4.2 Character FSM 7](#_Toc185795963)

[4.3 Game State FSM 7](#_Toc185795964)

[5. Graphics Rendering 8](#_Toc185795965)

[5.1 Rendering Module Architecture 8](#_Toc185795966)

[5.2 Layer Implementation 8](#_Toc185795967)

[5.2.1 Statis Objects 8](#_Toc185795968)

[5.2.2 Characters 8](#_Toc185795969)

[6. Conclusion 9](#_Toc185795970)

[6.1 Gameplay Demonstration 9](#_Toc185795971)

[6.2 Comparison with Original Game 9](#_Toc185795972)

[6.3 What We Learned 9](#_Toc185795973)

[A. Appendix 10](#_Toc185795974)

[A.1 Contribution 10](#_Toc185795975)

[A.2 Python Tool - PNG to COE 10](#_Toc185795976)

# 1. Introduction

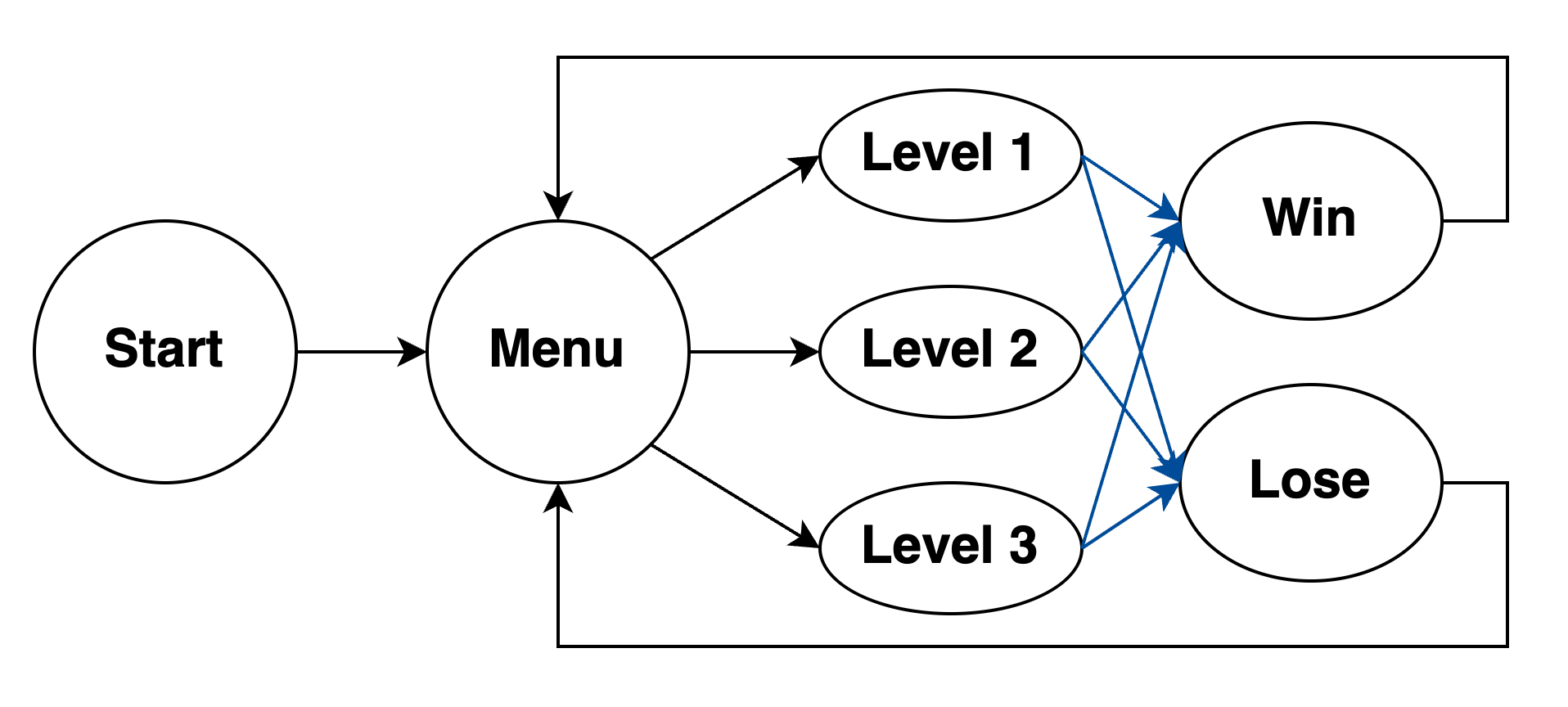
## 1.1 Motivation

## 1.2 Overview

- game name / what’s the game’s mechanism / what devices do we use / Top Module

# 2. Game Scene Architecture and Implementation Details

This chapter introduces our game's scene design and its detailed components. The following sections will explain our implementation methods, challenges we encountered during development, and how we solved these problems.

First, here is our scene's FSM (Finite State Machine):

▲ Figure 2.1: Scene's FSM

The scene transitions marked in blue are the only ones triggered by the game engine itself, while all other transitions are initiated through button interactions. All mouse click detection implementations include **Debounce** and **One Pulse** mechanisms to ensure reliable input processing. For consistency, we used the clk\_25MHz for these detections, which is the same clock signal used for VGA output and scene transitions.

## 2.1 Start and Menu Scene

We designed separate Start Scene and Menu Scene components based on our reference game Battle Cat, which features an initial game start button before entering the main menu. We decided to keep this design element as it provides a clear game entry sequence for players.

▲ Figure 2.2: Screenshot of Start and Menu Scene of ours and Battle Cat

Additionally, both scenes feature interactive buttons with hover effects - when the mouse cursor hovers over a button, its color changes to provide visual feedback to the player.

## 2.2 Play Scene

Below is our game interface. We will explain each implemented feature in detail.

▲ Figure 2.3: Screenshot of Play Scene

* The Stage number is displayed in the top-left corner, indicating the current level.
* The top-right corner shows both our current money and maximum money capacity.
* In the bottom-left corner is the purse upgrade button. The Purse Level affects both the money generation rate and the maximum money capacity. The button appears gray when there's insufficient money for an upgrade and brown when an upgrade is available. The button displays both the current level and the cost for the next upgrade. The current money will also be displayed on Seven Segment Display.
* The bottom-right features the tower firing button. When fully charged, it shows a red "ready-to-fire" state. After firing, it turns dark gray and gradually transitions to light gray in segments as it recharges. This charging mechanism pays homage to the original Battle Cat game.
* Our tower is positioned on the right side with its HP displayed in blue, while the enemy tower is on the left with HP shown in red.
* The character deployment buttons are located at the bottom of the screen. Unit deployment is only triggered if sufficient money is available. When a unit is deployed, money is deducted and a cooldown effect appears on the button to prevent rapid consecutive deployments of the same character.
* Enemy unit deployment follows a pre-designed Enemy Queuing system, which will be detailed in Chapter 3.2.
* The red and blue little blocks on the sides of the screen indicate the current number of enemy and army (our cats) on the battlefield respectively (this information is simultaneously displayed on the FPGA's LED display). Both sides are limited to a maximum of 8 units on the field at any time. When this limit is reached, no additional units can be deployed. These frames serve as visual indicators, flashing when the 8-unit limit is reached to alert players that they cannot deploy more units, or that enemy unit deployment has been temporarily paused. When one or more enemy units are defeated, multiple new enemy units may be spawned simultaneously, potentially creating challenging combat situations.
* The battle between both sides takes place on a one-dimensional line. However, to prevent complete overlap of all unit layers during multiple-unit combat, units are randomly offset upon generation. All attack and detection calculations remain one-dimensional. This mechanism aligns with the original game Battle Cat.

## 2.3 Win and Lose Scene

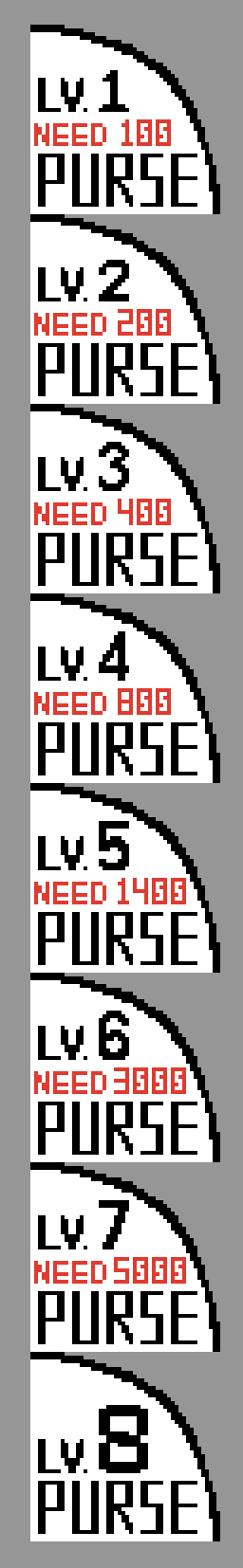
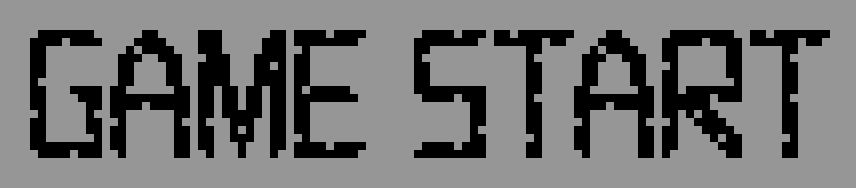
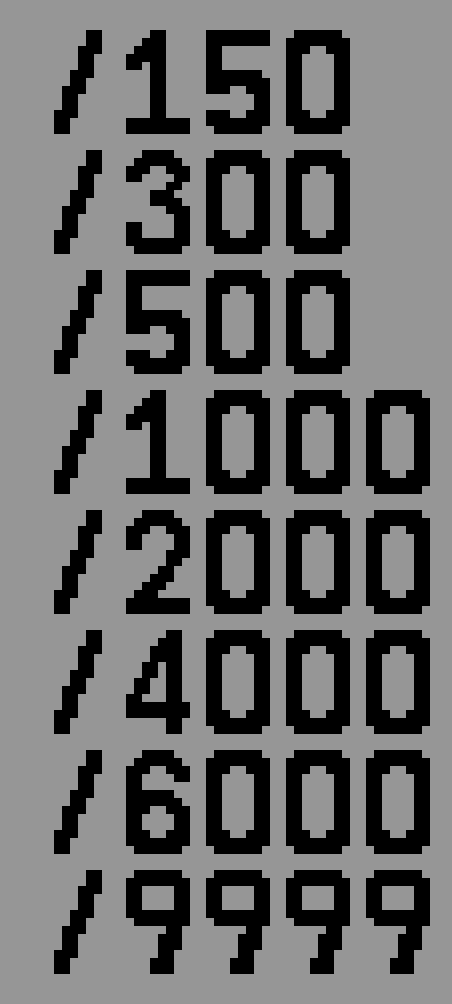
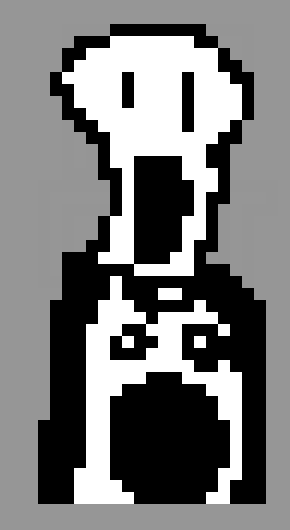
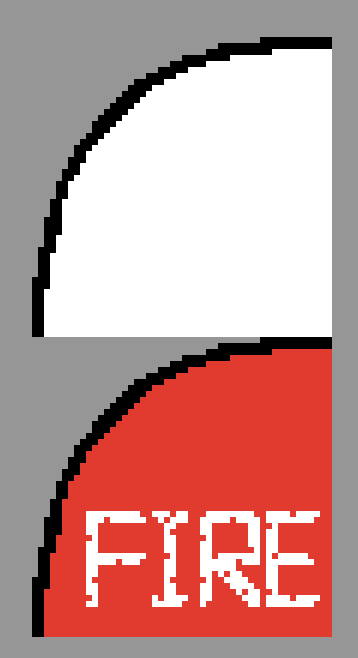
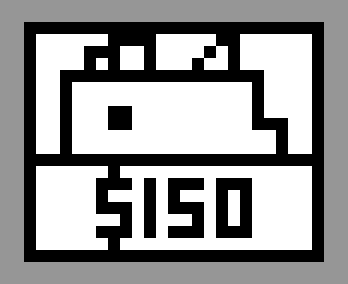
For these two scenes, we overlay a horizontal bar on the game screen displaying the win/loss text. A flashing "Tap to Continue" prompt is added to inform players they can return to the Start Scene. Notably, in addition to the horizontal bar, we invert the colors of the final game screen behind the bar to emphasize it.

▲ Figure 2.4: Screenshot of Win/Lose Scene

# 3. Game Material Design

## 3.1 Static Object Design

We utilize IPs (Single-port RAM) to store 19 Static Objects, including:

* Text Elements: 6 instances (mem\_FPCAT, mem\_LEVEL, mem\_GAME\_START, mem\_YOU\_WIN, mem\_YOU\_LOSE, mem\_TAP\_TO\_CONTINUE)
* Numerical Elements: 2 instances (mem\_Numbers, mem\_Money\_Max)
* Control Buttons: 2 instances (mem\_Btn\_Purse, mem\_Btn\_Fire)
* Generation Buttons: 8 instances (mem\_Frame\_...)
* Towers: 2 instances (mem\_Tower\_Cat, mem\_Tower\_Enemy)

5 more …

These elements were all designed by us and converted to **.coe** files using Python scripts (See Appendix A.2 for details) to serve as initial values for the RAM modules (these Static Objects' RAM values remain constant and are used solely for rendering purposes).

To optimize storage requirements, all .coe files storing images, including the characters mentioned in Chapter 3.2, were converted from PNG format using two methods. The first method, applied to single-color images, uses one bit per pixel to represent either colored or transparent states. The second method, used for three-color images such as characters, towers, and buttons, employs two bits per pixel to represent four states: black, white, red, and transparent. These approaches reduce the storage requirement from 12 bits per pixel (RGB) to either 1 or 2 bits per pixel.

## 3.2 Character Design

- each 6 pics, statistics

## 3.2 Stage Design

- IPs for enemy queue

# 4. Game Engine

- Display Time & Blanking Time

## 4.1 Storage Protocols

## 4.2 Character FSM

## 4.3 Game State FSM

# 5. Graphics Rendering

## 5.1 Rendering Module Architecture

- scene change, tint technique

## 5.2 Layer Implementation

### 5.2.1 Statis Objects

### 5.2.2 Characters

# 6. Conclusion

## 6.1 Gameplay Demonstration

## 6.2 Comparison with Original Game

## 6.3 What We Learned

# A. Appendix

## A.1 Contribution

## A.2 Python Tool - PNG to COE