Gambling-Like Machine Using TinyCircuits and a Python Backend

Author: Tan Yehan

Date: 25/11/2025

1. Introduction

This report documents the design, implementation, and limitations of a gambling-machine-like system constructed using three TinyCircuits boards (one **TinyScreen+** and two **TinyScreen** units) equipped with **WiFi TinyShields**, acting as the slot-machine reels. The system communicates with a Python server that interprets results, enforces winning conditions, and issues commands back to the devices. Devices and Python server need to share the same Wifi AP.

The implementation served as part of a larger group project. This report covers only the components I directly developed and contributed, excluding architectural changes introduced later by teammates (e.g., DHCP support, credit display logic, or integration with an RFID-based credit and turret subsystem).

2. Hardware Overview

The system consists of:

- **3 TinyCircuits microcontroller units**, each attached to
    - one **TinyScreen(+) display or TinyScreen(default) display**,
    - one **WiFi TinyShield** capable of basic TCP communication.

Each device is programmed individually via the **Arduino IDE** using a dedicated .ino sketch.
All graphical assets (six slot icons) are stored as bitmaps:

- Bell
- Cherry
- Clover
- Diamond
- Lemon

- Seven

These are indexed 0–5 respectively on the microcontroller.

---

3. Software Architecture (Device Side)

3.1 Hardcoded Network Configuration

In the version I authored, each device contains:

- a hardcoded SSID and WiFi password,
- a fixed local IP address,
- the hardcoded IP address of the backend server.

This static addressing was required at the time because:

1. The device software must know the server IP before initiating the TCP handshake.
2. The backend required identifying devices by a predetermined “device number” (devNum).
3. Initially, each device’s identity was implicitly mapped to its IP address.

Teammates later reworked this into a DHCP-compatible model. I do **not** document those changes here as they were not part of my contribution.

3.2 Device Identity (devNum)

Each TinyCircuit device sends a two-byte header upon connection:

[0xFD, devNum]

Only device numbers **2, 3, and 4** are valid.
These map to slot-machine **columns 0, 1, and 2** respectively. Changes will need to be made for more device numbers and columns.

3.3 Local Odds and Symbol Selection

Each device maintains its own probability distribution for selecting the three row symbols during a spin. The odds I implemented were:

Symbol	Weight
Bell	23

Symbol	Weight
Cherry	23
Clover	16
Diamond	16
Lemon	13
Seven	9

These weights apply **only to the top and bottom rows** after the design was scaled down (see Section 6).
The **middle row is always assigned by the server**, not by the device.

4. Server–Device Communication Protocol

4.1 Server Handshake and Heartbeat

After accepting a connection, the server expects:

Byte 0: 0xFD

Byte 1: devNum

If this header fails validation, the server drops the connection.

To check on the connection between device and server, the server periodically sends a **handshake / keep-alive byte**:

0xFE

If the client does not respond (e.g., due to crash or network loss), the server removes it from the active client list and waits for a new connection.

4.2 Commanding the Middle Row

To control the displayed result for the **middle row**, the server sends exactly one byte:

0–5 (index of symbol to display in middle row)

This allows the server to:

- randomly determine outcomes,

- enforce specific outcomes for testing or demonstration (e.g., guaranteed win/loss),
- synchronise the “middle row” across devices.

This design choice mimics real-world gambling systems where top/bottom rows often simulate randomness but have no effect on payout.

4.3 Device → Server Result Reporting

After each spin, the device sends **four bytes**:

[devNum, top, mid, bottom]

Each value is a symbol index (0–5).
The server aggregates these from all three devices to construct a **3×3 grid**.

5. Backend (Python) Implementation Summary

The backend code, jackpot.py, contains:

- multithreaded TCP handlers,
- a central state table of connected clients,
- a shared results buffer,
- payout calculation logic,
- credit management (not part of my contribution),
- winning-line evaluation.

5.1 Winning-Line Limitation

Although the backend originally supported multiple winning lines:

- horizontal rows,
- vertical columns,
- diagonals,

my implementation restricted the win condition *only to the middle row*. Final used rule:

[(1,0), (1,1), (1,2)]

In this scenario, a win occurs **only if the three middle-row symbols match**.

6. Flashing Winning Symbols

When a win is detected, the server sends a “flash mask” to each device.
The relevant function is:

mask = 0b10000000 | (row_bits)

Where:

- bit 0 = flash top row
- bit 1 = flash middle row
- bit 2 = flash bottom row
- bit 7 = distinguishing flag (ensures byte ≥ 128 and cannot be confused with values 0–5)

Important limitation:

Only the **middle row flashing** is reliable.
The TinyScreen drawing code was originally written for a **3×3 slot machine**, but mid-project the system was reduced to a **3×1 machine**.
The flashing routine for top/bottom rows references out-of-bounds image positions and therefore does not update the display correctly.

I did not have time to redo the rendering logic to match the new 3×1 design.

7. Architectural Limitations and Failure Points

As a critical self-evaluation (important for academic marking):

7.1 Hardcoded Network Parameters

- Static IP assignment is inflexible, fragile, and unsuitable for real-world deployment.
- A single mistyped IP leads to silent failure.
- No attempt is made to validate server reachability or recover gracefully.

7.2 Absence of Security

The system uses:

- no encryption,
- no authentication other than 0xFD header,
- no replay protection,

- no rate-limiting.

Any attacker on the same network can spoof devices or manipulate payouts.

7.3 Blocking I/O and Thread-per-Client Design

- Threads block indefinitely on socket reads.
- No timeout or cancellation is implemented for the main report loop.
- Thread-per-client model scales poorly beyond three devices.

7.4 Poor Error Handling on Device Side

- No retries on WiFi connection failure.
- No fallback server.
- No persistent storage of credits or machine state.
- Device reboots may desynchronise the game.

7.5 Rendering Logic Partially Broken

As noted earlier, flashing for top/bottom rows does not work due to outdated drawing logic.

7.6 Lack of Synchronisation During Animation

Device-side animations are not guaranteed to finish simultaneously.
This creates visual misalignment—undesirable in multi-reel gambling systems.

---

8. Work Done by Teammates (Not My Contribution)

The following features were implemented by teammates and are excluded from assessment of my contribution:

- DHCP support (dynamic addressing)
- A more sophisticated backend integrating:
  - a credit system,
  - RFID input,
  - a turret firing mechanism
- A device-side credit display routine after each spin
- UI improvements and other refactoring

I make no claim to these components.

9. Conclusion

This project demonstrates a functional—though simplified and limited—networked slot-machine system using TinyCircuits hardware.
My contribution includes:

- probability-weighted symbol selection on devices,
- device–server protocol design (including handshake and flashing mask),
- middle-row command system,
- core rendering logic for the TinyScreen,
- result transmission,
- core server-side functions for parsing results and evaluating wins.

While the system works for demonstration and filming, it contains multiple architectural shortcomings typical of a time-constrained student project. A production-quality redesign would require significantly stronger error handling, secure communication, robust rendering code, and a scalable backend architecture.

As of this report, the github repo is public:
https://github.com/XxHanpowerxX/tinygamba

10. Declaration

Chatgpt usage is heavy among the coding (both server and client) and writing of this report. However, it cannot and could not be trusted blindly, especially for the coding. Due to varying versions of Arduino devices and the lack of projects made using Tinycircuit available to the AI model, it struggled heavily to answer many questions or gave completely false answers.

It is important to double check using the Tinycircuit devices and read up on documentation and examples for proper answers.