

# Cyber Escape: The Firewall Trials

[BATCH-11]

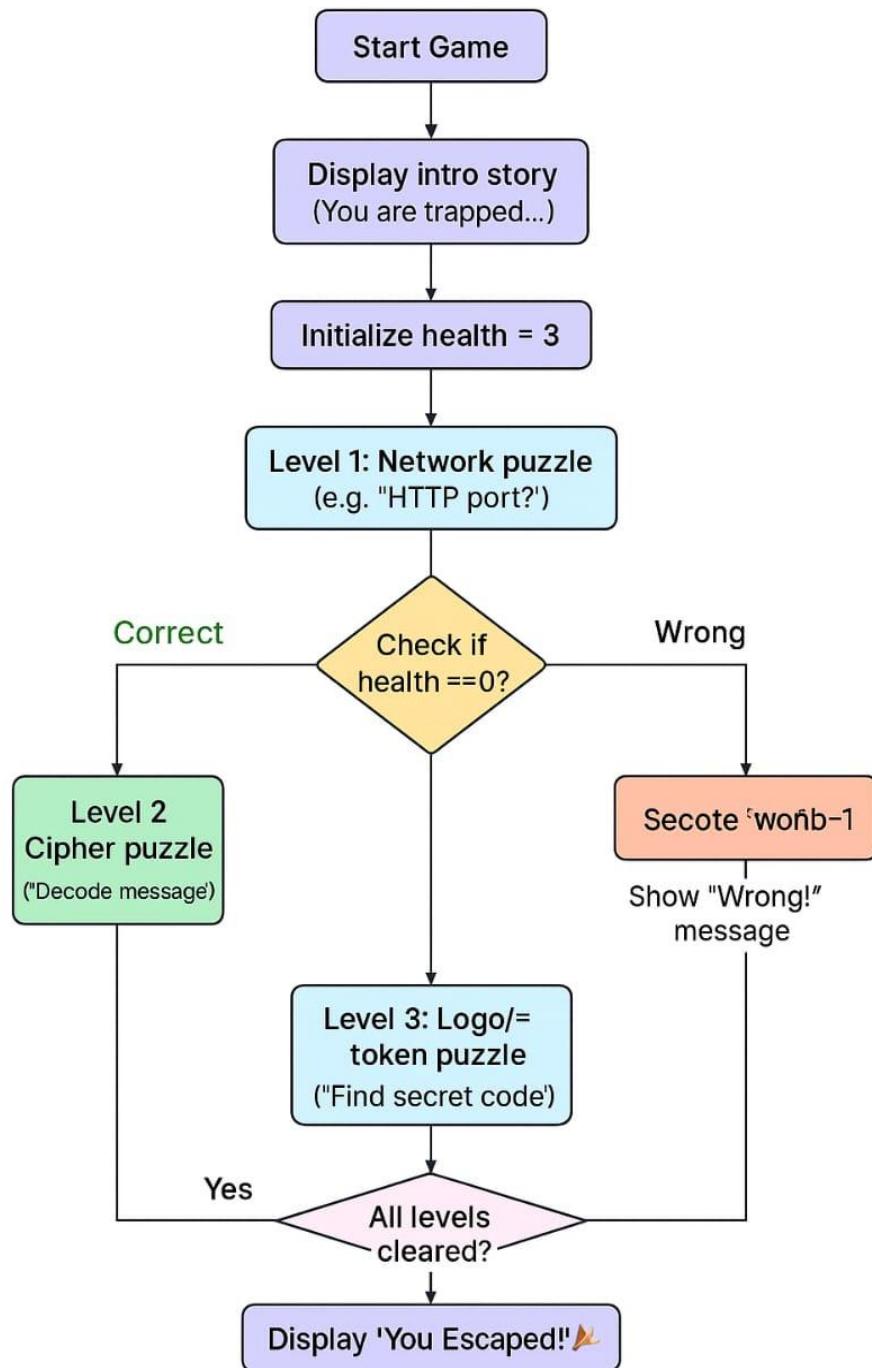
## Final Table

Name of Student	SRN	Module Worked On
MAHANTH SRIRANGA NANDUR	PES2UG25AM149	Level 3 + all level cleared logic
Kushal Joshi	PES2UG25EC067	Start, Intro, Health System, Level 1
Lad Krishna Samir	PES2UG25EC068	Level 2 Cipher System
Monish S	PES2UG25CS315	Integration of all modules, Tester, Debugging, Compiler

## Problem Statement

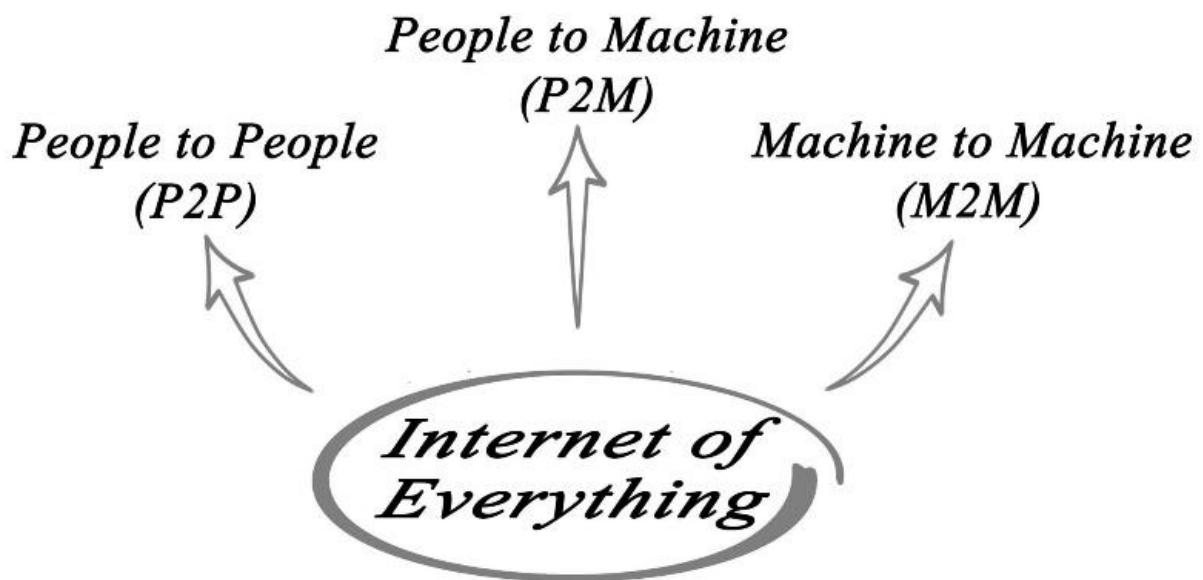
The challenge is to design and implement a gamified educational experience, Cyber Escape: The Firewall Trials, that tests a player's knowledge across three key cybersecurity domains, Network Protocol basics, Cryptography (Ciphers), and Token/Authentication methods, using a structured, multi-level puzzle format. The system must incorporate a health-based failure mechanism (starting at health = 3) to provide immediate feedback on incorrect answers, ultimately determining whether the player successfully escapes the firewall or is trapped by depletion of health.

# Cyber Escape: The Firewall Trials



# Approach Used To Solve

The approach used in the "Cyber Escape" project is a State Machine Architecture combined with Object-Oriented Programming (OOP). The entire program flow is governed by a central game state variable (e.g., "PLAYING", "QUIZ", "LOST") which dictates the specific code block executing and the rules of interaction, ensuring clean separation between the intense space shooter boss fights and the static, decision-based quiz segments. This state management is coupled with OOP, where core game entities like Ship, Player, Enemy, and Projectile are organized into an inheritance hierarchy. This structure allows game objects to manage their own behaviour, health, and collision logic efficiently, making the code both modular and highly maintainable, particularly for handling the sequential levelling and the health-dependent transitions required by the game's design.



## INPUT

### CODE-INTERFACE

A screenshot of a code editor interface, likely Microsoft Visual Studio Code, showing a workspace for a game project. The left sidebar shows a file tree with files like `board.py`, `jackfruit problem.py`, `subject_utils.py`, and `sudoku.py`. The main editor area displays a Python class definition for a ship:

```
class Ship:
    def __init__(self, x, y, width, height, color, health=100):
        self.rect = pygame.Rect(x, y, width, height)
        self.color = color
        self.health = health
        self.cooldown = 0
        self.projectiles = []
    def draw(self, win):
        pygame.draw.rect(win, self.color, self.rect)
    def draw_health_bar(self, win):
        # Red background
        pygame.draw.rect(win, RED, [self.rect.x - 20, self.rect.y - 5, self.rect.width + 40, 5])
        # Green bar
        pygame.draw.rect(win, GREEN, [self.rect.x - 20, self.rect.y - 5, self.rect.width + 40, 5 * self.health / 100])
    def move(self, vel):
        self.rect.x += vel
```

The terminal tab at the bottom shows command-line output related to the game's dependencies and setup.

### GAME-INTERFACE



## OUTPUT

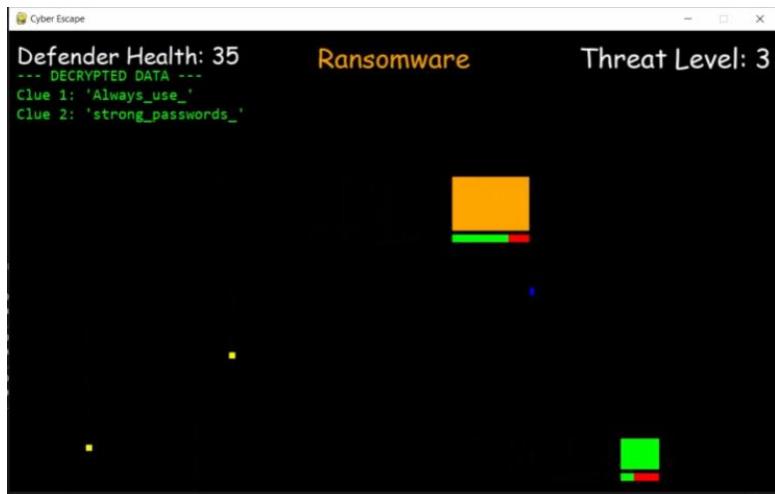
### LEVEL-1 (WORM)



### LEVEL-2(TROJAN)



### LEVEL-3(RANSOMWARE)



# **Challenges Faced**

- 1) Projectile and Collision Cleanup Issues
- 2) Screen Element Overflow and Layout Bugs
- 3) Pygame Font/Text Rendering Artifacts and Overlap
- 4) OOP Health/Projectile Logic Mismanagement