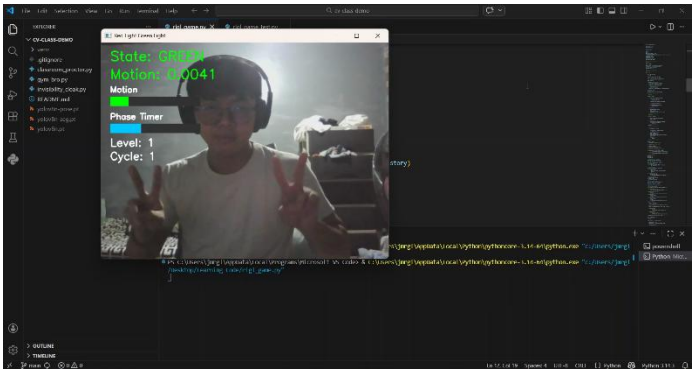
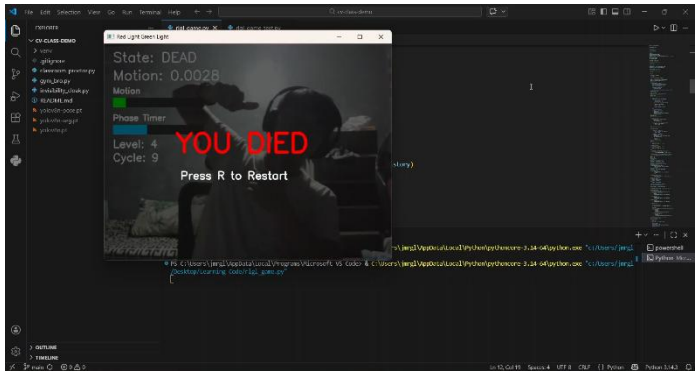




Red Light Green Light Game Using Computer Vision (SHORT REPORT)

1. Introduction

This project implements the classic **Red Light Green Light** game using computer vision techniques. The goal is to detect player motion using a webcam and enforce the rules of the game based on movement behavior.

The system continuously reads frames from the webcam and calculates motion between consecutive frames. Depending on the game state (GREEN or RED), the player is either required to move or remain still. If the player violates the rules, the system transitions to a DEAD state and ends the game.

The project demonstrates the use of real-time image processing, motion detection, and a finite state machine to control game logic.

---

2. System Overview

The system consists of four main components:

1. Webcam input and frame preprocessing
2. Motion score calculation
3. Red Light Green Light state machine
4. User interface and game feedback

The webcam continuously captures frames which are processed to detect motion. Based on the computed motion score and the current game state, the system determines whether the player survives or loses.

---

3. Input and Preprocessing

Each frame from the webcam undergoes several preprocessing steps to improve motion detection reliability.

Steps used in preprocessing:

1. Capture frame from webcam.
2. Resize the frame to width **640 pixels** to improve processing speed.
3. Convert the frame to **grayscale** to simplify computation.
4. Apply **Gaussian blur** to reduce noise and minor pixel fluctuations.

These steps help reduce noise from camera sensors and lighting changes while making motion detection more stable.

---

4. Motion Detection Algorithm

Motion is detected using the **frame difference method**, which compares the current frame with the previous frame.

The motion score is computed using the following process:

1. Compute the absolute difference between two frames.

diff = abs(current_gray - previous_gray)

2. Compute the mean pixel difference.

motion_score = mean(diff) / 255.0

The motion score ranges between **0 and 1**, where higher values indicate more movement.

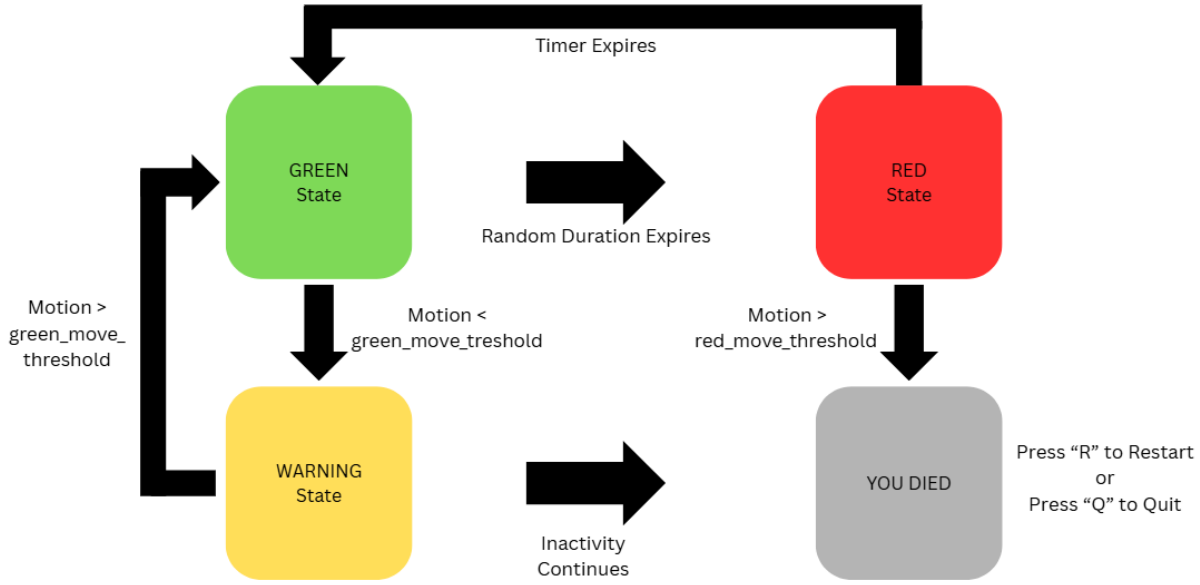
Typical observed values during testing:

- No movement: 0.002 – 0.003
- Small movement: 0.004 – 0.005
- Large movement: 0.01 – 0.02

Thresholds are used to determine whether movement is significant enough to trigger game rules.

---

5. RLGL State Machine



The game logic is implemented using a **finite state machine** with four states:

- GREEN
- RED
- WARNING
- DEAD

GREEN State

During GREEN, the player is required to move. The system monitors motion levels and tracks idle time.

Rules:

- If motion is below the movement threshold, an idle timer starts.
- If idle time exceeds the warning threshold, the system enters the WARNING state.
- If idle time exceeds the death threshold, the player dies.

After a randomized duration, the system switches to the RED state.

WARNING State

This state alerts the player that they have not moved for too long.

Rules:

- The player must move to return to GREEN.
- If inactivity continues, the player transitions to DEAD.

RED State

During RED, the player must remain still.

Rules:

- The first **red_grace_ms** milliseconds allow the player to stop moving.
- After the grace period, motion above the threshold results in immediate death.
- If the player remains still until the RED timer expires, the game returns to GREEN.

DEAD State

The DEAD state indicates game over.

When this state is reached:

- The game displays a **YOU DIED** message.
- The player may restart the game using a "**R**" key command.

---

6. Parameter Values

The following parameter values were used during testing and tuning:

green_move_threshold = 0.004

red_move_threshold_base = 0.006

red_move_threshold = red_move_threshold_base + (level - 1) * 0.003 for level increase

red_grace_ms = 650

idle_warning_ms = 1800

idle_death_ms = 3600

Green duration range: 2600 ms – 4200 ms

Red duration range: 1700 ms – 2900 ms

These parameters were adjusted based on observed motion values from the webcam.

---

7. Observed Failure Cases and Mitigations

Failure Case 1: Idle Timer Not Reset During State Transition

Problem

During testing, the idle detection timer (idle_start_time) continued running even after the system switched from GREEN to RED state. Because of this, the idle timer eventually triggered during the RED phase, which should not occur since the idle rule is only intended for the GREEN state.

This resulted in incorrect behavior where the player could receive warnings or die during the RED phase despite following the correct rule of staying still.

Mitigation

The issue was resolved by resetting the idle timer whenever the game transitions between states. Specifically, idle_start_time was set to None during state transitions. This ensures that the idle timer is only active during the GREEN state and does not carry over into the RED state.

Failure Case 2: Player Not Dying During RED State

Problem

While implementing the level and cycle mechanics of the game, a bug was observed where the player did not die during the RED phase even when significant movement occurred. Instead, the player would only die when the idle timer was triggered during the GREEN phase.

The issue was traced to improper threshold tuning for motion detection. The baseline threshold values provided were not suitable for the motion score values produced by the webcam and environment.

Mitigation

Motion thresholds were recalibrated based on actual observed motion score values during testing. The following adjustments were made:

green_move_threshold: 0.04 → 0.0004
red_move_threshold: 0.055 → 0.006

These new thresholds better matched the observed motion score ranges and allowed the system to correctly detect movement violations during the RED phase.

Failure Case 3: Overlapping "YOU DIED" Display

Problem

When implementing the restart functionality in the DEAD state, the "YOU DIED" message appeared duplicated on the screen. The text overlay overlapped with itself, creating a cluttered display.

This occurred because an additional "YOU DIED" rendering block was added while the original rendering logic was still active.

Mitigation

The issue was resolved by removing the redundant "YOU DIED" rendering block and ensuring that the message is only drawn once during the DEAD state. This prevented overlapping overlays and produced a cleaner game over screen.

---

8. Conclusion

This project successfully demonstrates how computer vision techniques can be used to implement an interactive Red Light Green Light game. By combining frame preprocessing, motion detection, and a state machine, the system can enforce movement rules in real time using a webcam.

The implementation highlights practical challenges such as noise, lighting variation, and threshold tuning. Future improvements could include player detection, region-based motion tracking, and integration with object detection models.