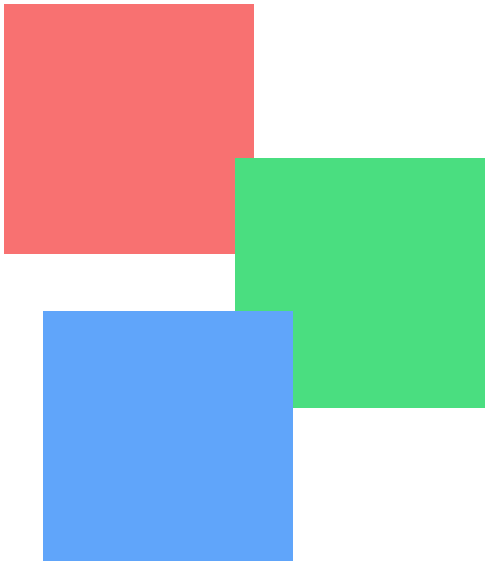
FPGA-Based RGB Dice Game

Team 9: 김준회, 변지인, 윤호승, 장준우 | 2025. 12. 05.



CONTENTS

Project Outline

01 프로젝트 개요

- 개발 환경

HDL

Design &
Synthesis Tool

Development
Environment

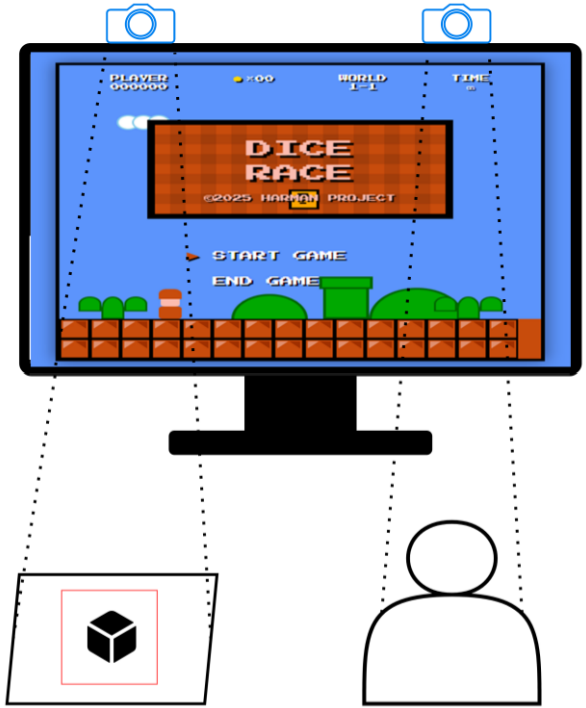
HW Setup

Diagram Editor

Code Manage &
Co-Work

01 프로젝트 개요

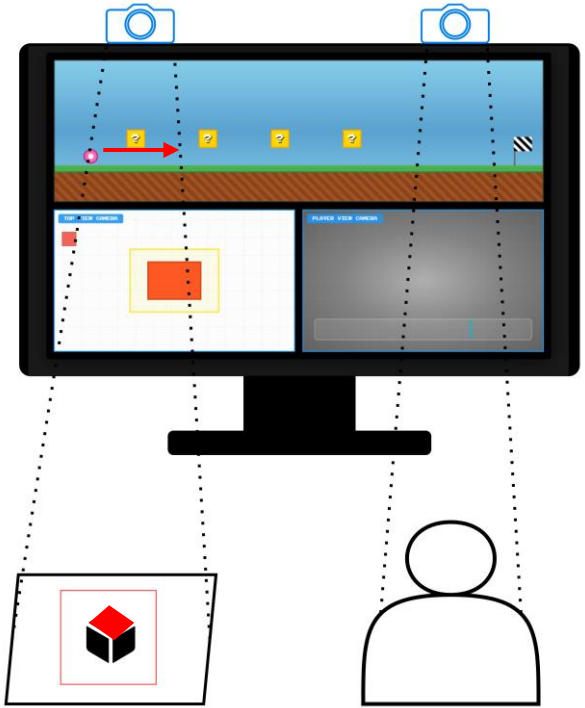
- **Game Rule : 1. Dice Throw**



- Btn_U push 후, 영역에 맞춰 주사위 던지기

01 프로젝트 개요

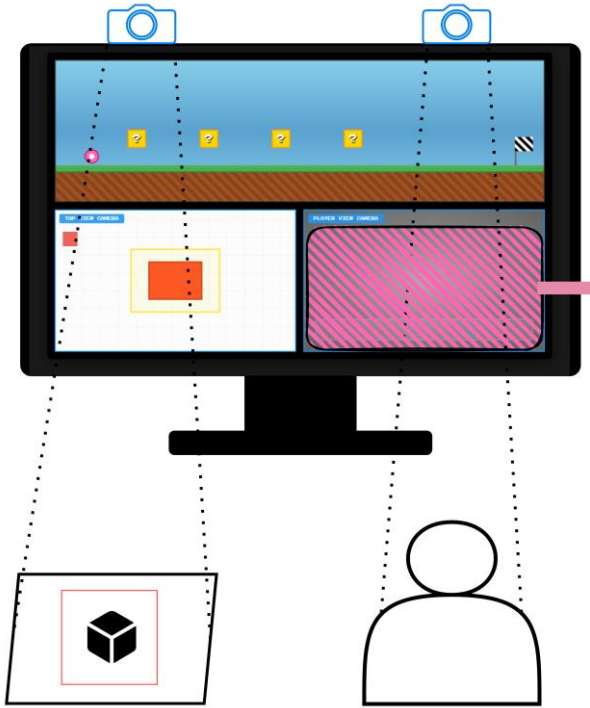
- **Game Rule : 2. R/G/B Detection**



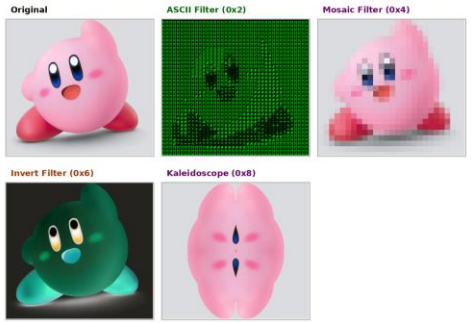
- 주사위의 R/G/B Color 인식 후 player 이동
- R(1) /G(2) /B(3)

01 프로젝트 개요

- **Game Rule : 3. Apply Filter**

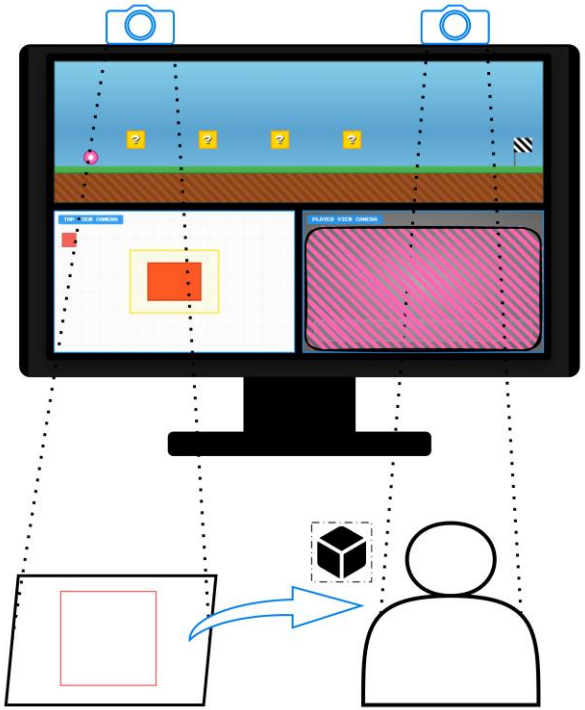


- {2, 4, 6, 8} 번째 칸에서 도착하면, CAM2(사용자)에 filter가 적용



01 프로젝트 개요

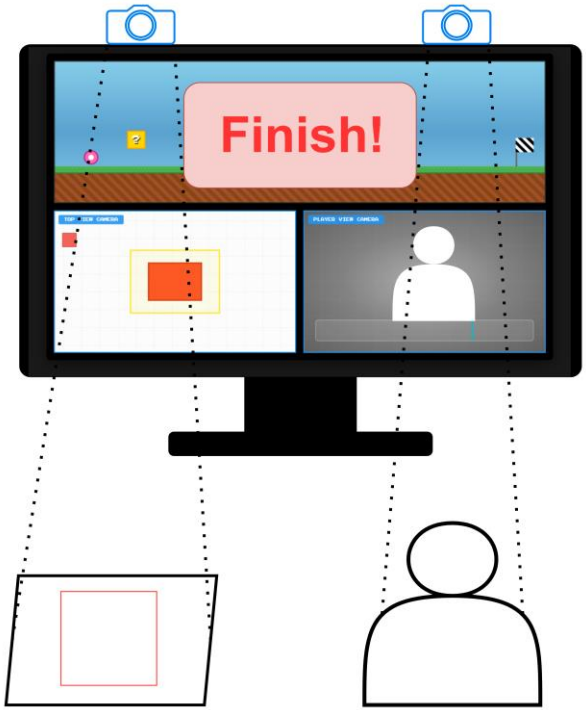
- **Game Rule : 4. Turn End**



- 명확한 Turn Change를 위해,
  흰색 화면 display를 Turn 분기점으로 설정

01 프로젝트 개요

- **Game Rule : 5. Game End**

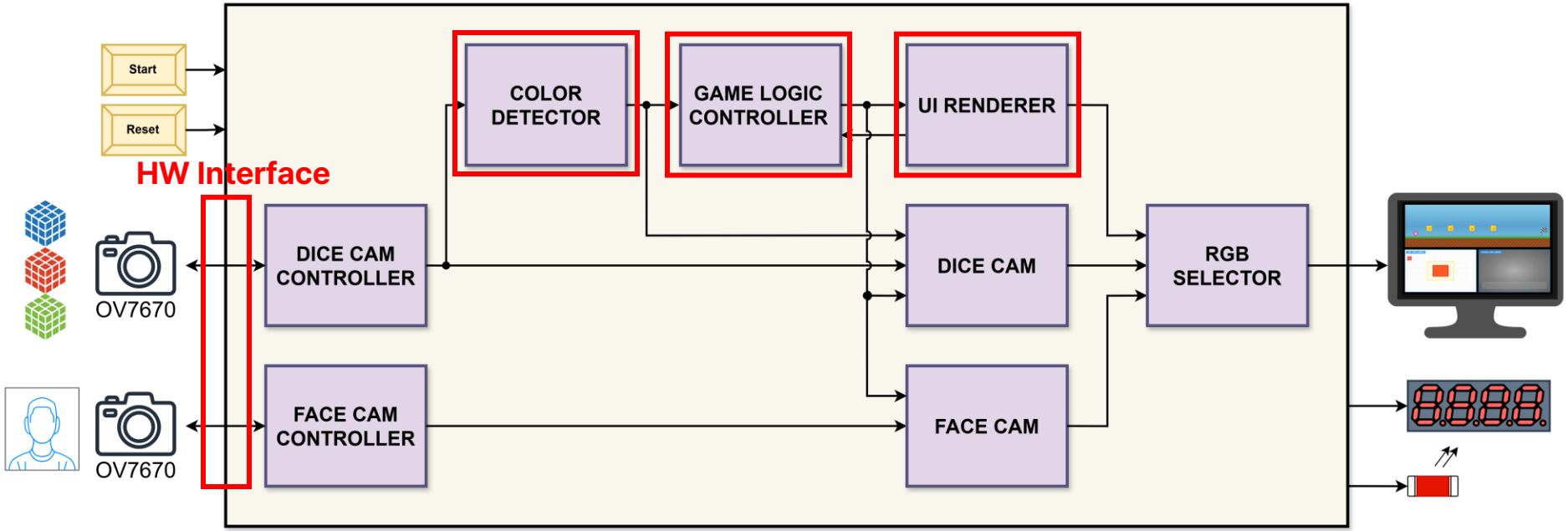


- 먼저 10번째 타일에 도착 시 승리!

System Architecture

02 시스템 구성

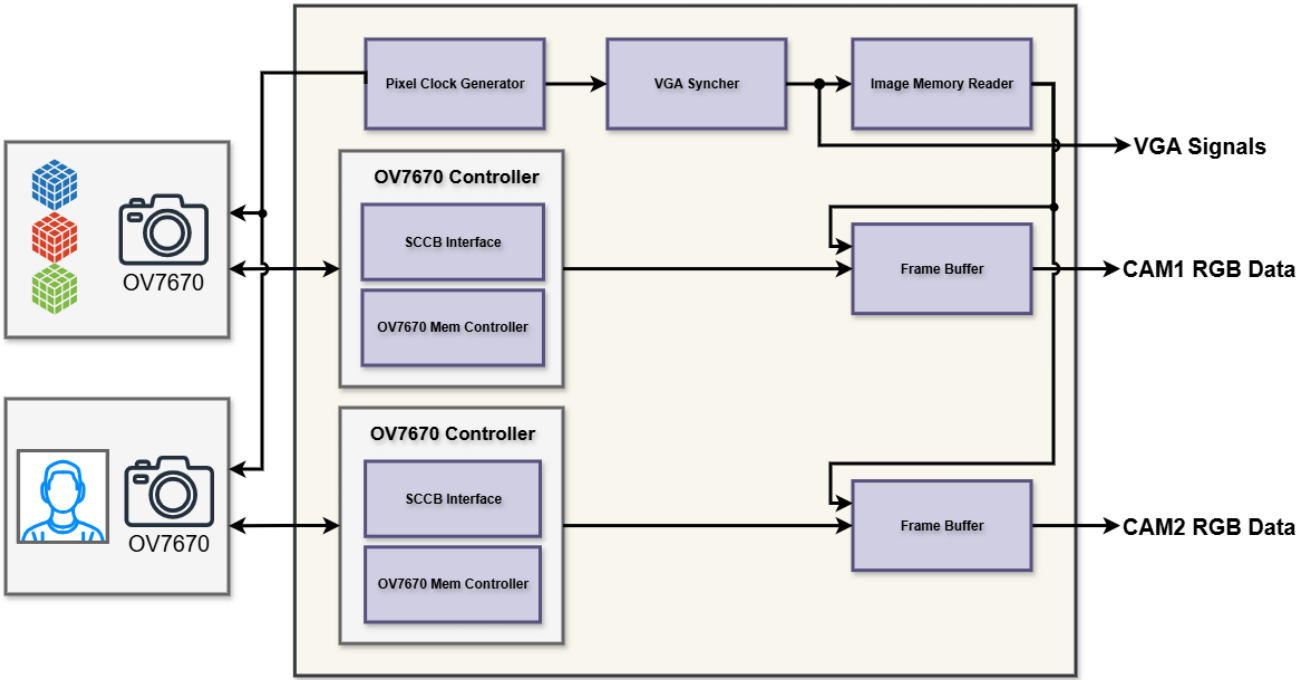
- **Overall Block Diagram – 4 Main Modules**



Key Implementation

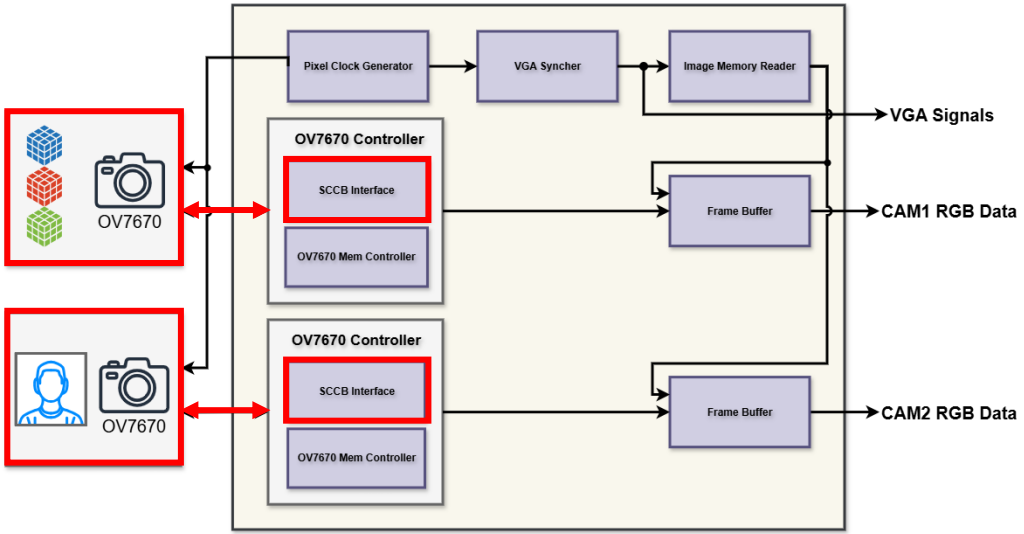
03 상세 설계

1. HW Interface (Camera System)



03 상세 설계

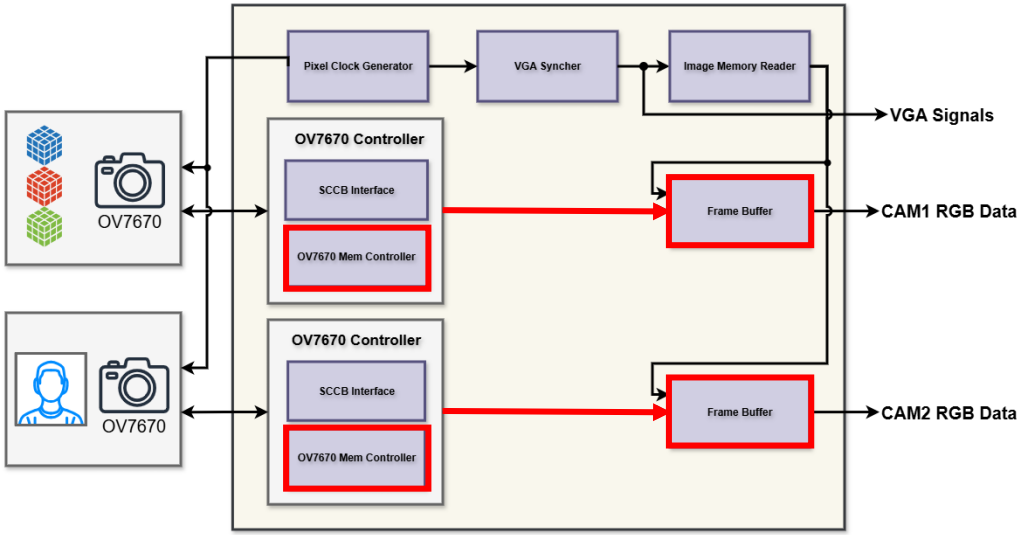
1. HW Interface (Camera System)



1. 카메라 초기화: OV7670 Controller가 SCCB로 카메라 레지스터 설정

03 상세 설계

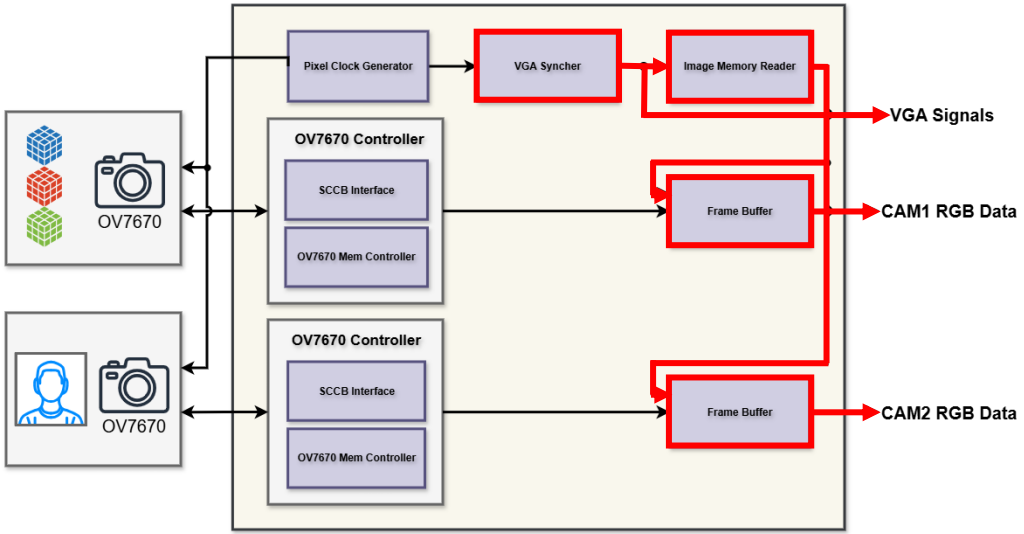
1. HW Interface (Camera System)



2. 데이터 저장: 8비트 데이터를 RGB565로 결합해 Frame Buffer에 저장

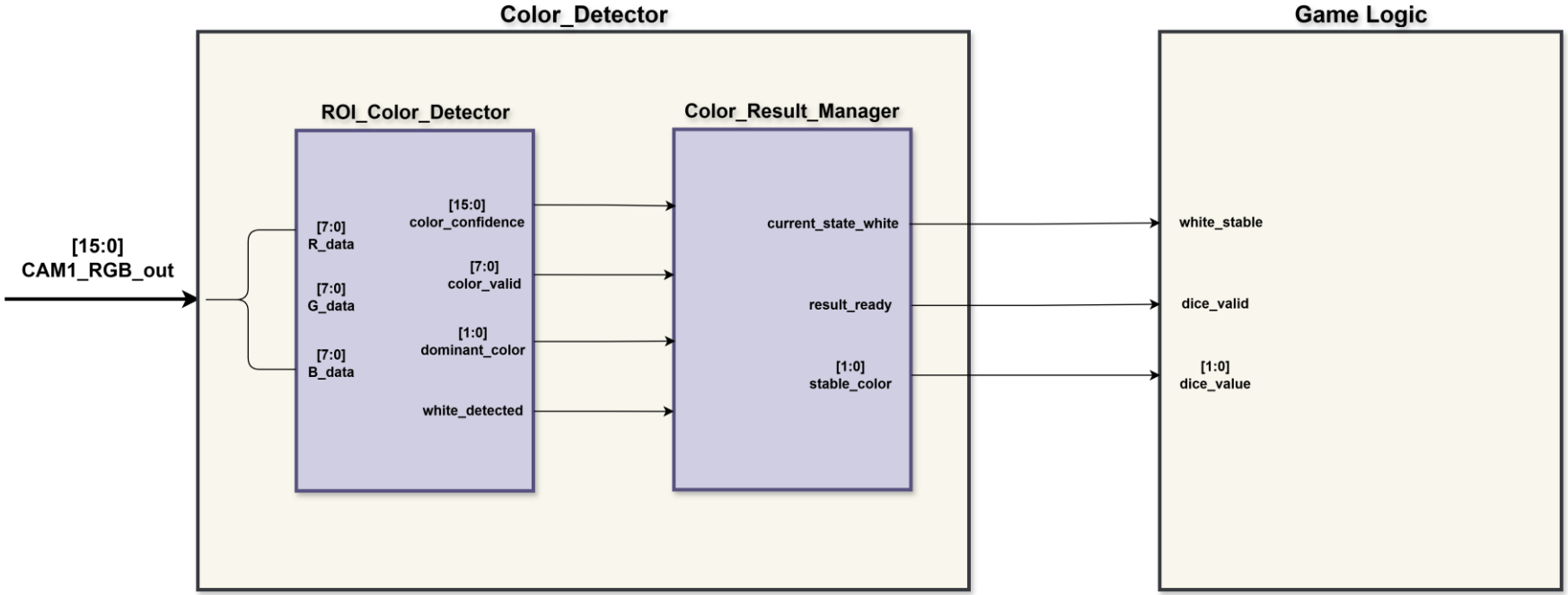
03 상세 설계

1. HW Interface (Camera System)



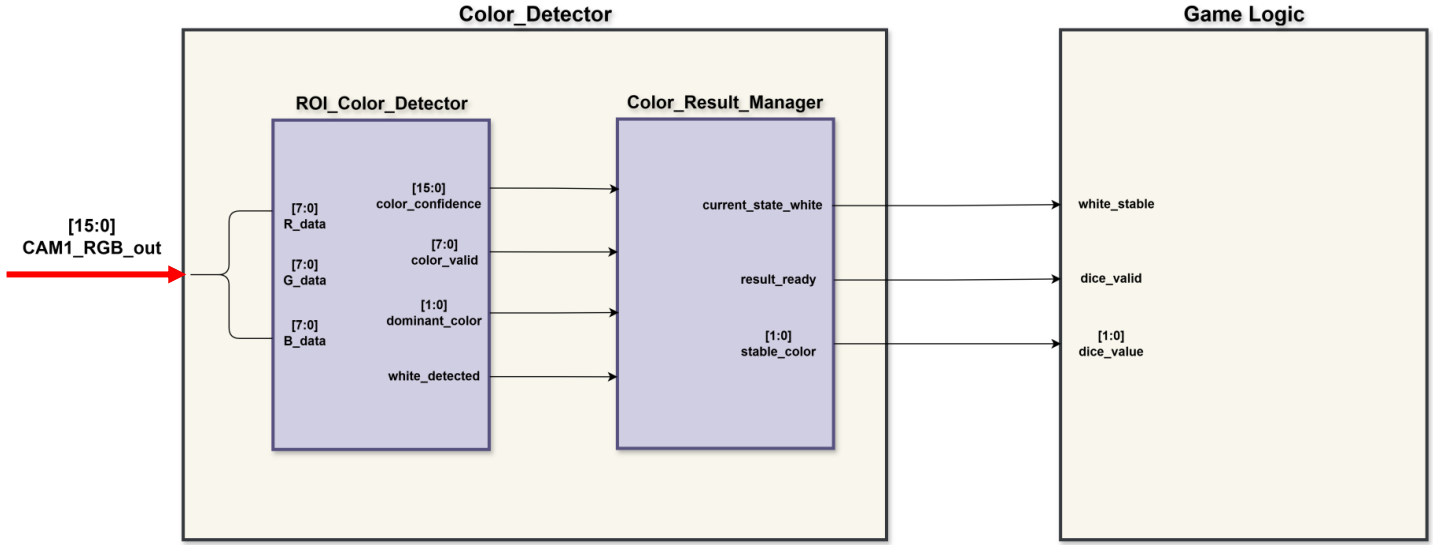
3. VGA 출력: VGA 타이밍에 맞춰 Frame Buffer 데이터를 출력

2. Vision Processing (Color Detect)



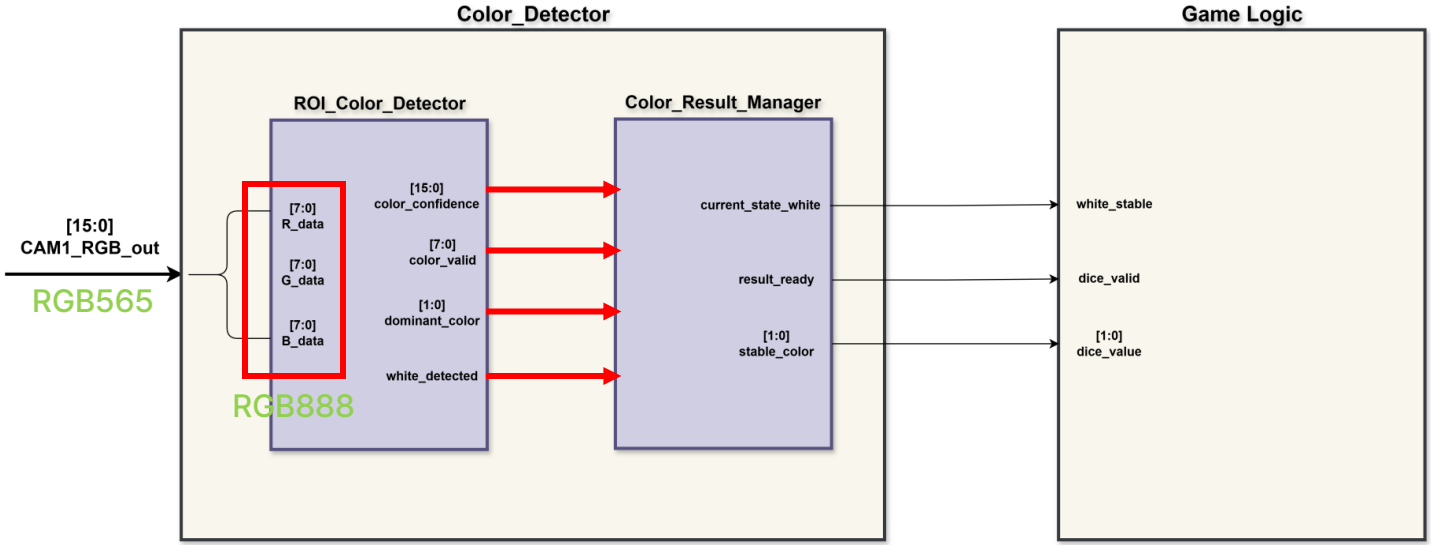
03 상세 설계

2. Vision Processing (Color Detect)



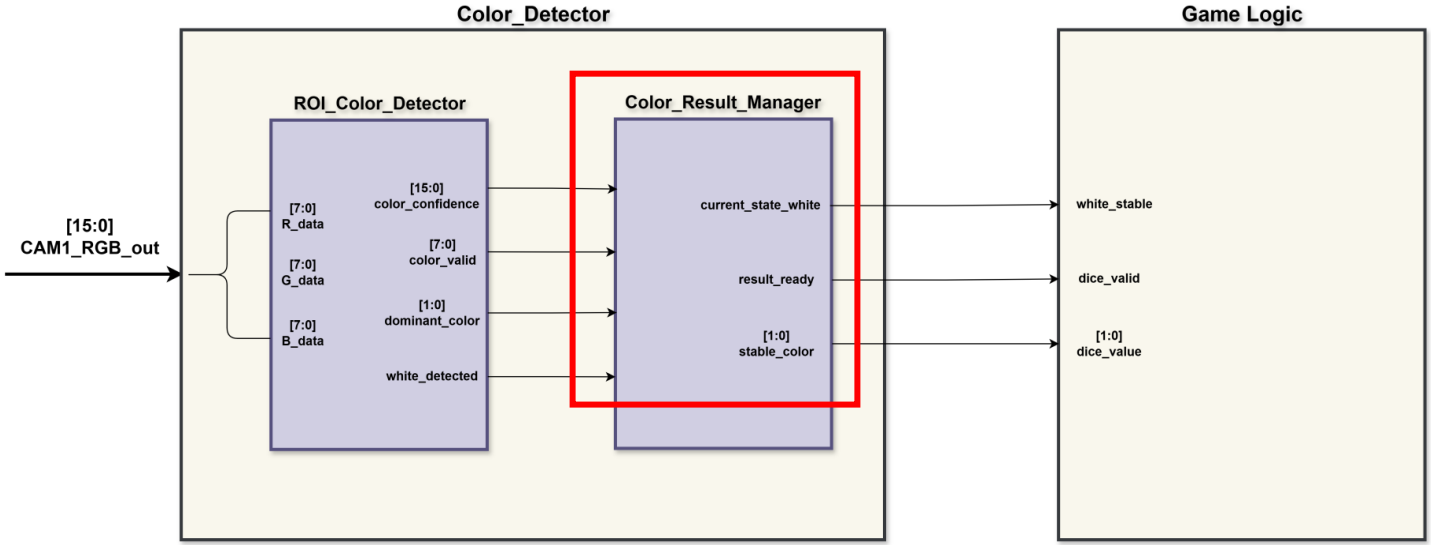
1. 데이터 입력: Frame Buffer를 거친 영상 데이터가 Color Detector로 전달

2. Vision Processing (Color Detect)



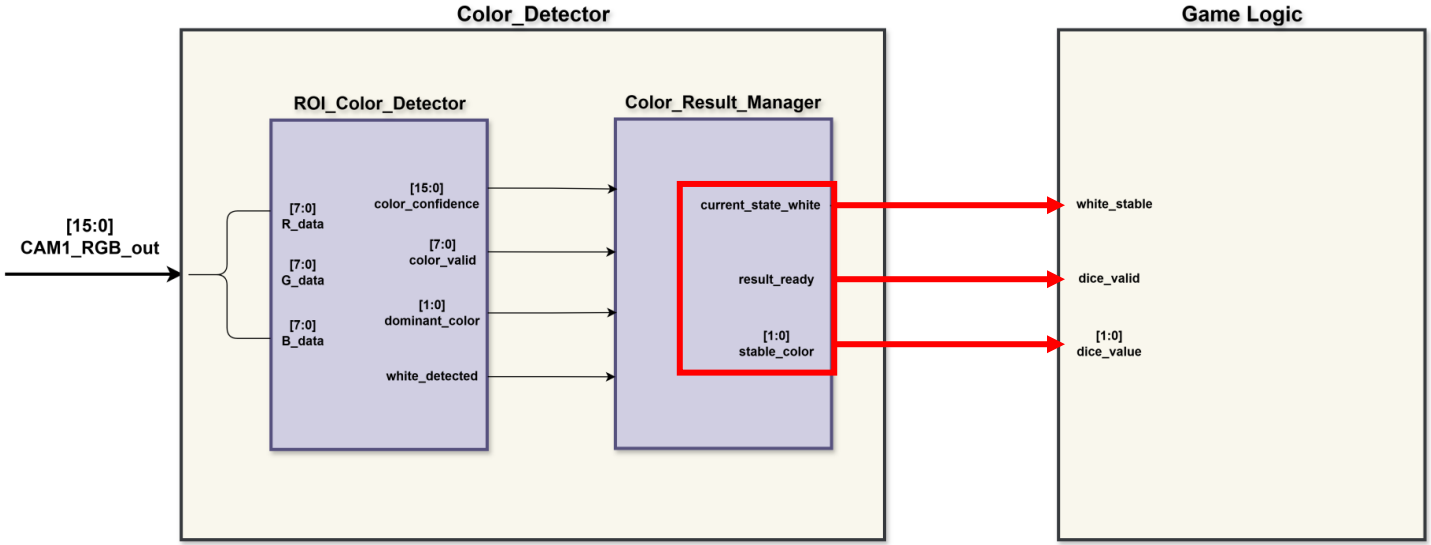
2. 색상 판별: RGB565를 RGB888로 변환하여 Threshold 비교 후 감지한 색상 출력

2. Vision Processing (Color Detect)



3. 노이즈 제거: 연속된 3개 프레임의 다수결 연산으로 최종 색상 결정
(예: R-R-G → Red 판정)

2. Vision Processing (Color Detect)

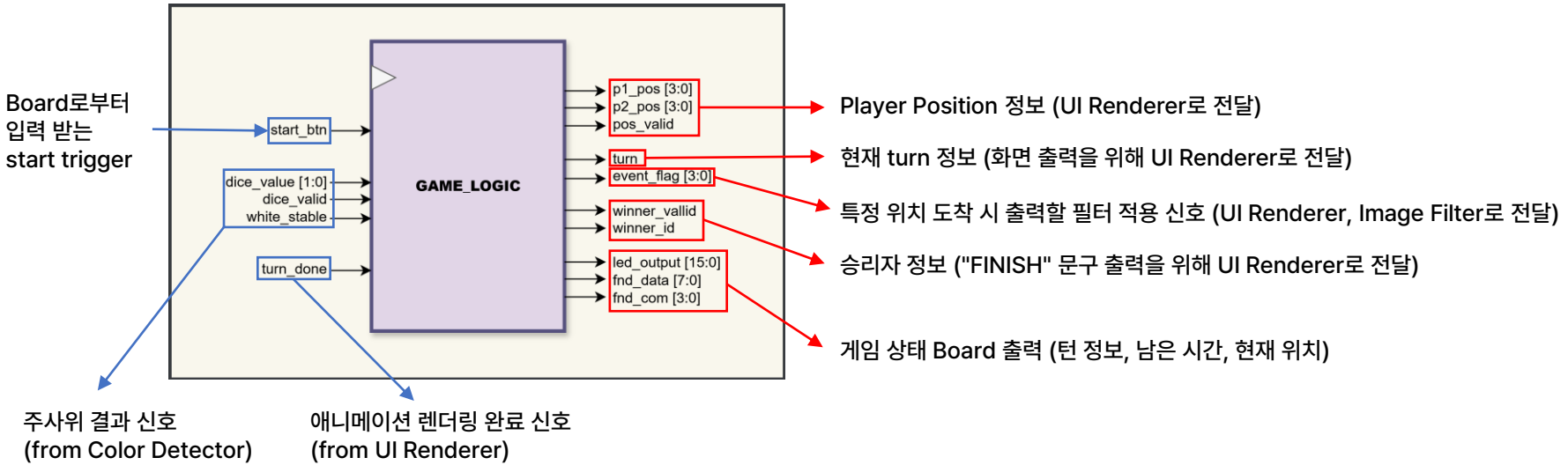


4. 데이터 전송: 노이즈가 제거된 색상 값을 Valid 신호와 함께 Game Logic으로 전달

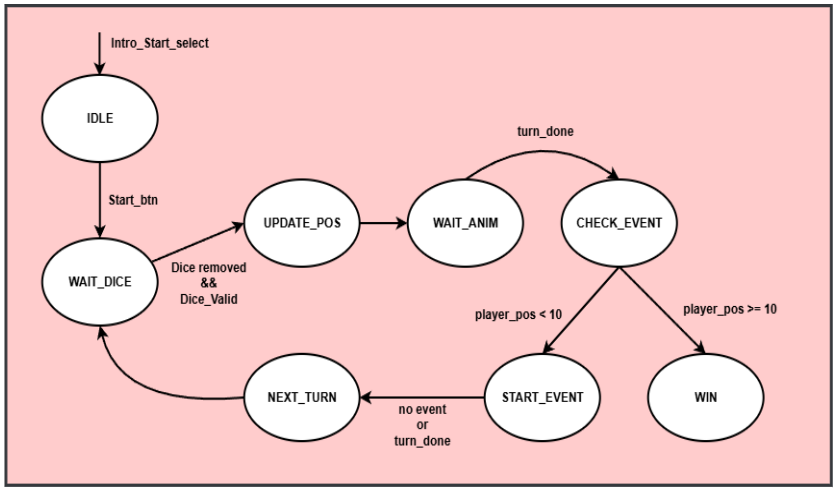
3. Game Logic Design Main Signals (I/O)

Key Roles:

주사위 색 인식 결과를 기반으로 플레이어의 위치, Turn 관리, 승리 판정 등 게임의 전체 진행 흐름을 제어



3. Game Logic FSM



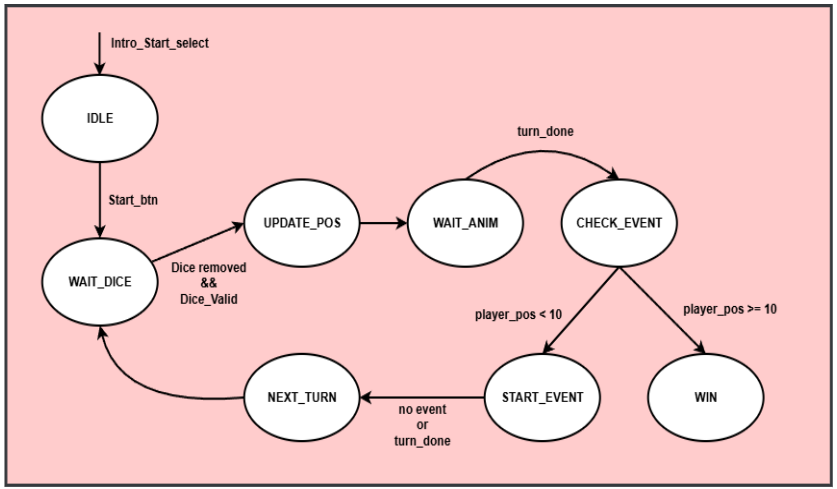
IDLE
Start 버튼 대기

WAIT_DICE
Color Detector 모듈로부터 주사위 신호 수신,
8초 동안 주사위 신호가 오지 않으면 턴 전환

UPDATE_POS
현재 플레이어의 위치 업데이트 후,
UI Renderer 모듈로 위치 정보 전달

WAIT_ANIM
애니메이션 처리를 위한 딜레이 구간

3. Game Logic FSM



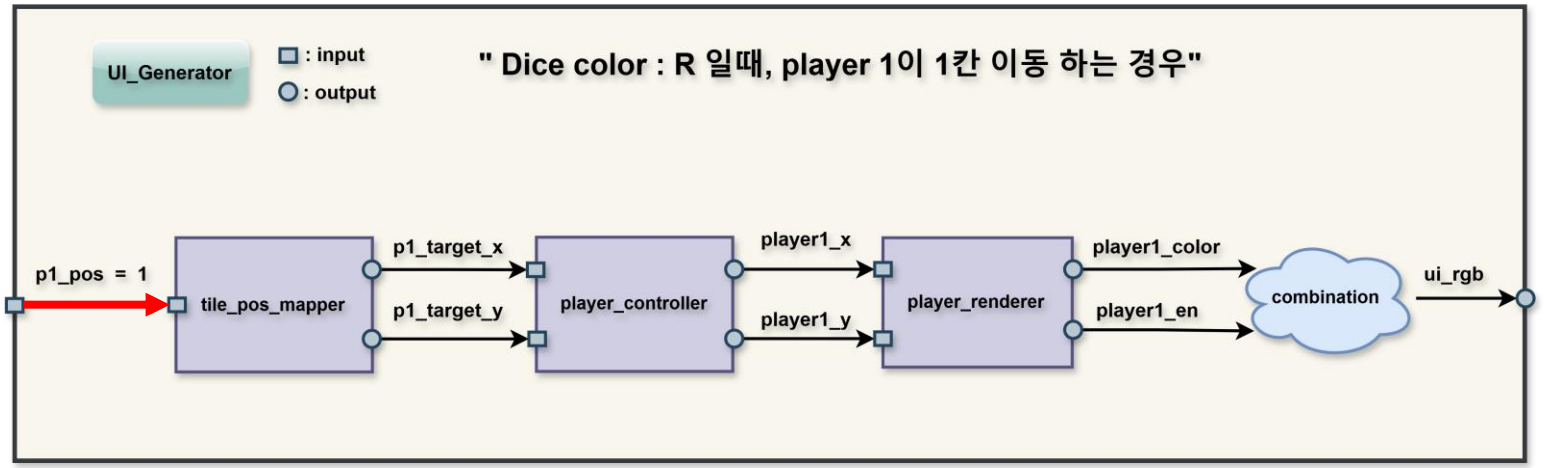
CHECK_EVENT
플레이어의 위치에 따른 필터 적용 요청 신호 전달

START_EVENT
필터 적용 및 게임 알고리즘 처리를 위한 딜레이 구간

NEXT_TURN
턴 전환 완료 후 내부 신호 초기화
(끝날 때까지 반복)

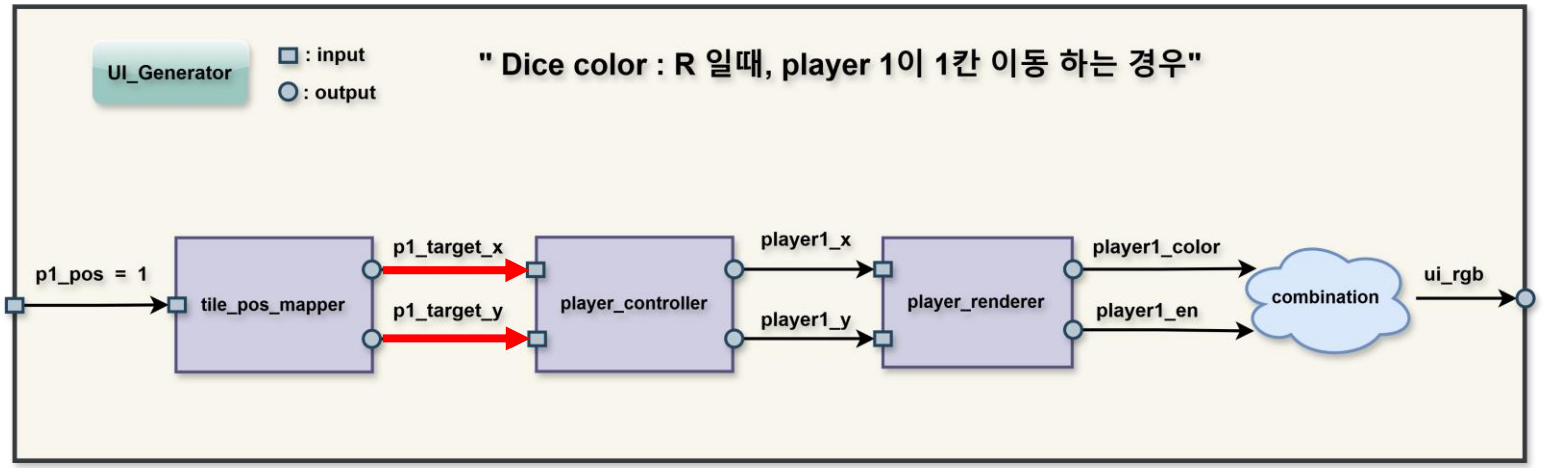
WIN
승자 결정 후 동작 중단

4. UI & Visualization



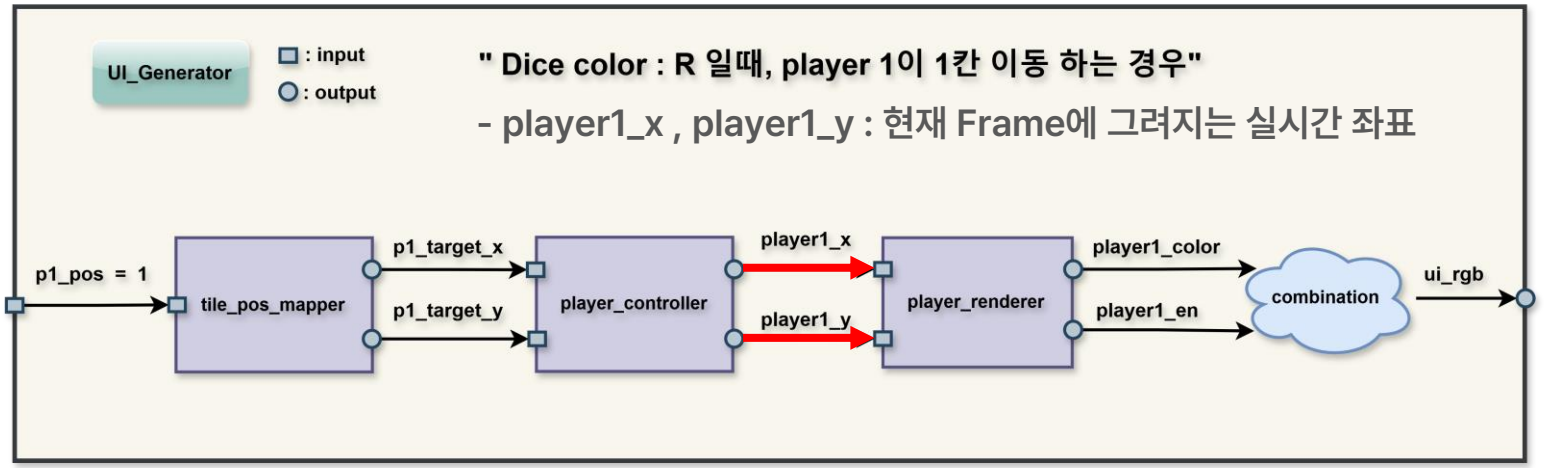
1. Input: Game Logic에서 Player1 위치 값 (p1_pos = 1)

4. UI & Visualization



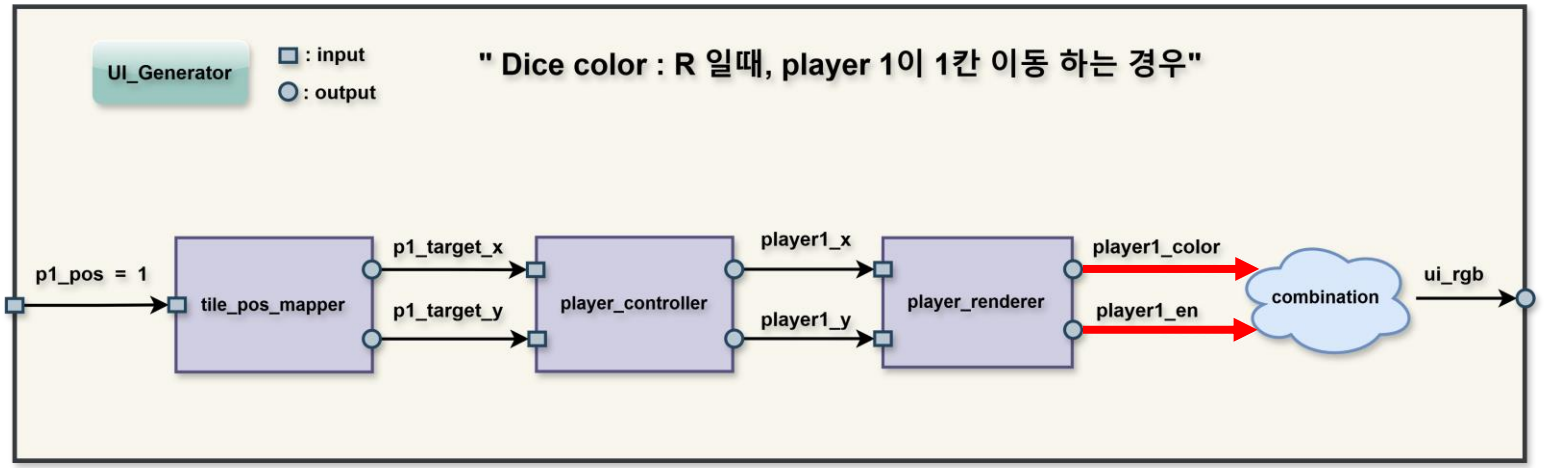
2. Target: Player1 위치 값을 Tile 1번 위치인 target_x = 80으로 출력

4. UI & Visualization



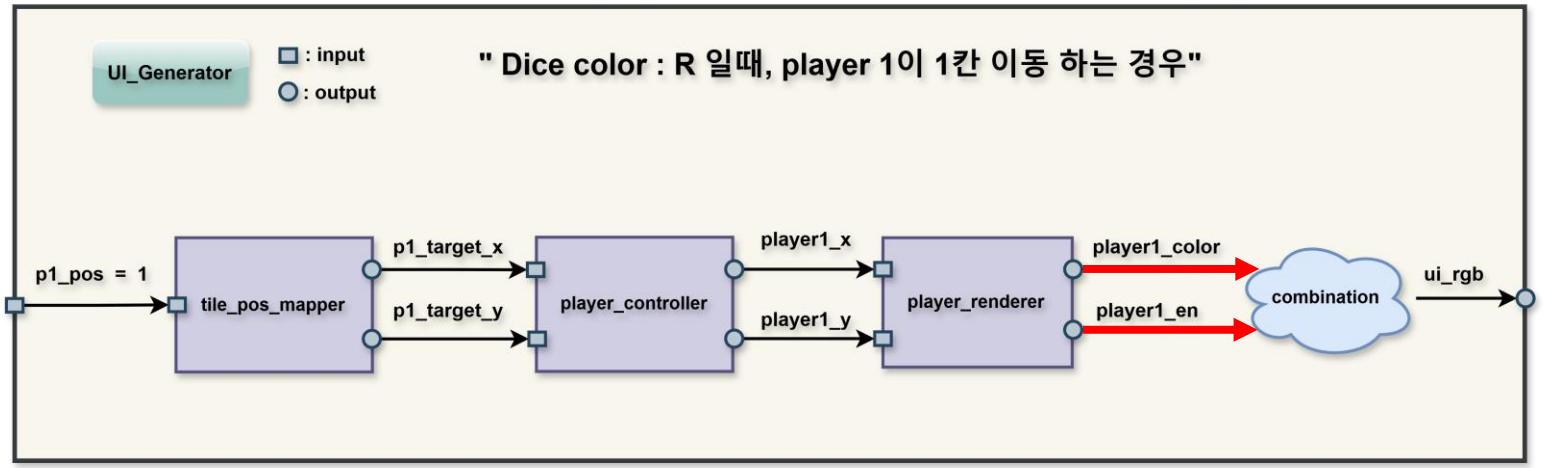
3. Move: Player1이 목표위치까지 24 Frame 동안 이동

4. UI & Visualization



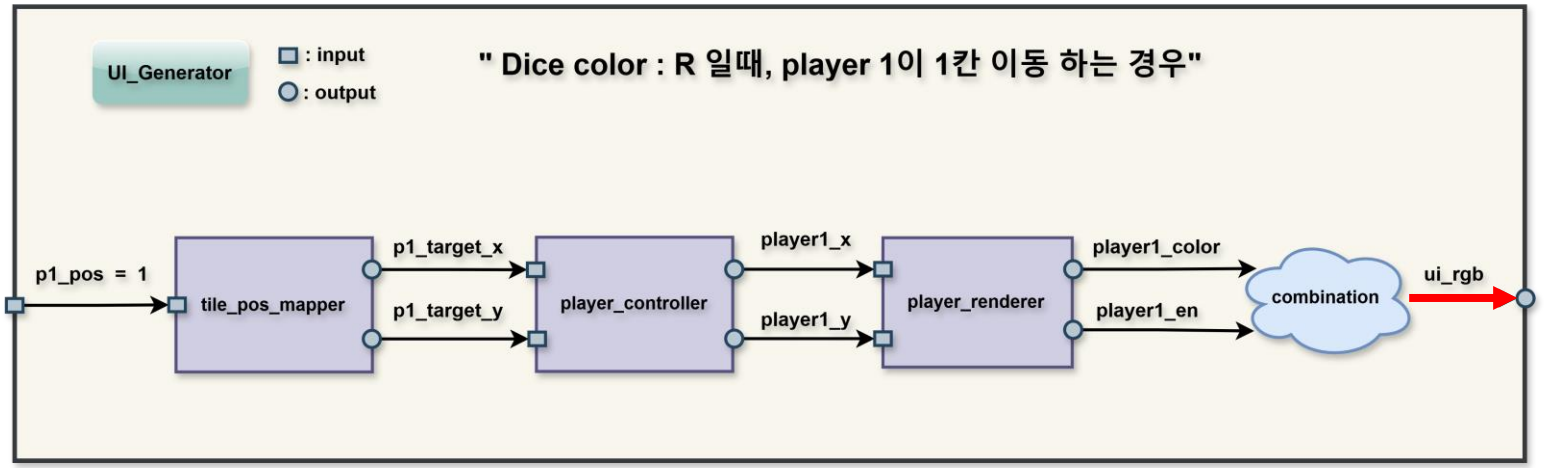
4. Read mem: Player 영역이면 메모리에서 pixel 값을 가져옴

4. UI & Visualization



5. Data : player1_color (RGB 값), player1_en (출력 활성화 신호)

4. UI & Visualization



6. Ouput : enable 신호로 Player1이 Layer의 우선순위가 되어 RGB Data 출력

04 시연 영상

Demo

04 시연 영상


Intro
DICE RACE
2025 VGA PROJECT
START GAME
END GAME
RELEASE - 251205

Trouble Shooting & Insights

05 Trouble Shooting & Insights

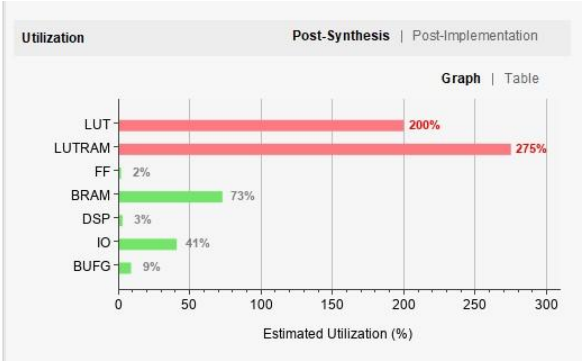
Trouble Shooting 1. Resource Overflow

[문제 개요]

Dual QVGA 카메라 사용 시 FPGA Resource Overflow 발생

- Frame Buffer 1 (QVGA): BRAM 73% 사용
- Frame Buffer 2 (QVGA): BRAM 부족 → LUTRAM 사용

결과: **LUT 200%, LUTRAM 275% Overflow**



[원인 분석]

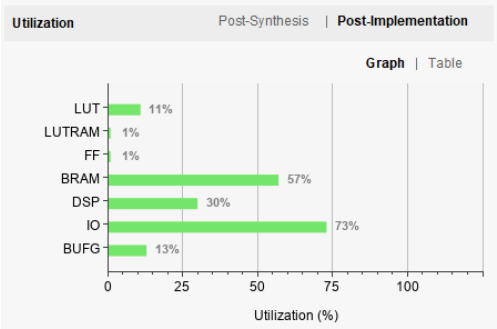
Basys3 FPGA의 제한된 BRAM 용량 → 153,600 pixels (QVGA×2) 동시 저장 불가능

05 Trouble Shooting & Insights

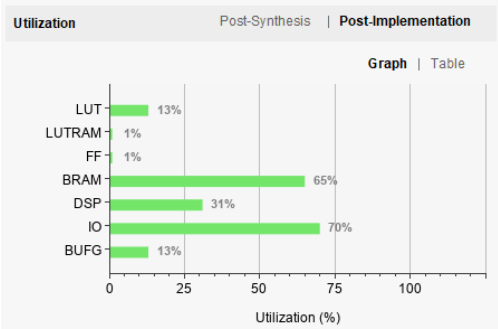
Trouble Shooting 1. Resource Overflow

[문제 해결]

1. OV7670 레지스터 재설정: Input 해상도 축소 [QVGA (320×240) → QQVGA (160×120)]
2. Hardware Upscaler 구현: Nearest-Neighbor (2×2 복제) 적용, QQVGA를 실시간 Upscale하여 VGA에 QVGA 크기로 출력



Optimization
(Dual QQVGA Buffer 적용)



Final Implementation

05 Trouble Shooting & Insights

Trouble Shooting 2. Player_Controller

[문제 개요]

UI상에서 player icon이 너무 빠르게 이동하는 문제

[원인 분석]

UI module의 source clock인 pixel_clk은 25 MHz로, 애니메이션이 너무 짧은 시간에 진행

[문제 해결]

- frame_tick(60Hz)를 생성해 24 Frame 동안 애니메이션을 구현
- Frame 마다 player가 위치해야 할 중간 좌표를 계산해 슬라이딩 움직임을 구현

Trouble Shooting 3. Game Logic : Turn Transition Problem

[문제 개요]

올바르지 않은 주사위 값이 Read 되어서, 플레이어의 이동이 불규칙하거나 업데이트가 누락되는 문제 발생

[원인 분석]

- Color Detector 모듈이 감지한 색상 값을 확정하는 타이밍과 턴 전환 이후 게임 로직의 샘플링 타이밍의 불일치
- 불규칙한 배경 때문에 Color Detector 모듈이 “새로운 주사위 입력” 시점을 명확히 판단하기 어려움
- 모듈 간 상태 전환 기준과 유효 신호(Valid)가 모호해 처리 흐름이 무너짐

[문제 해결]

- 로직 개선: 주사위 인식 과정에 ‘흰색 배경 감지’ 단계를 추가해 기준 프레임 확보
- FSM 수정: 턴 전환 및 배경 감지가 완료된 이후에만 색상 결과를 유효로 처리
- 최종 결과: 안정적인 타이밍에 색상 정보를 샘플링하여 타이밍 정합성 확보 및 문제 해결

05 Trouble Shooting & Insights

Trouble Shooting 4. Anti-Noise Color Detecting

[문제 개요]

주사위를 한번만 던졌음에도, 게임 말이 순식간에 맵을 통과해서 게임이 끝나버리는 현상

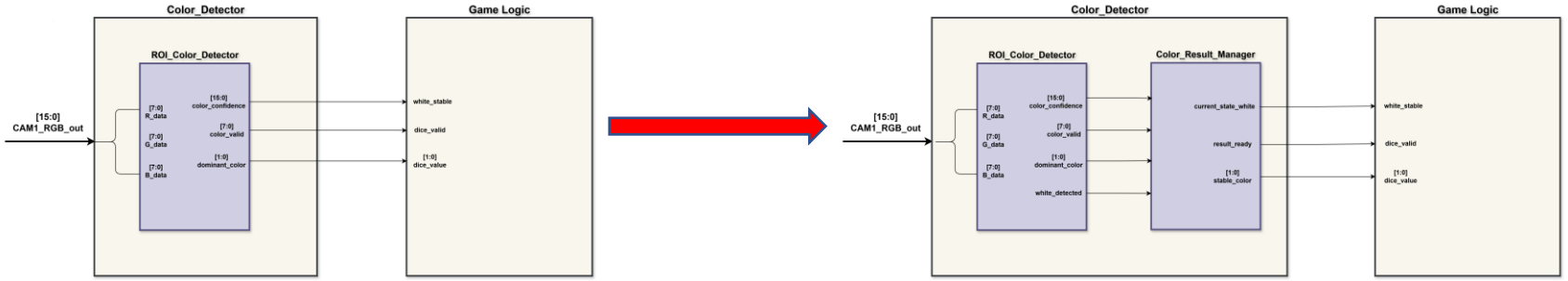
Trouble Shooting 4. Anti-Noise Color Detecting

[원인 분석]

- 기존의 Color_Detector 설계는 Color Detecting 후, 그 값을 곧바로 Game Logic 모듈로 전달
- 카메라의 미세한 노이즈나 손떨림으로 인해 잘못된 색상을 간헐적으로 감지하여 전달

[문제 해결]

- *Color_Result_Manager* 모듈을 추가하고, 'Voting 알고리즘'을 도입
- 연속된 3개의 Frame 동안 감지된 색상을 저장하고, 가장 많이 감지된 색상을 최종적인 Stable Color로 확정



05 Trouble Shooting & Insights (Co-Working, Build-Automation)

Insight : Building Co-Working Development Environment

[협업 환경 구축의 필요성]

설계 초기, 각자 개별 환경에서 수동으로 Vivado 프로젝트를 생성하거나 빌드 절차를 관리했음.

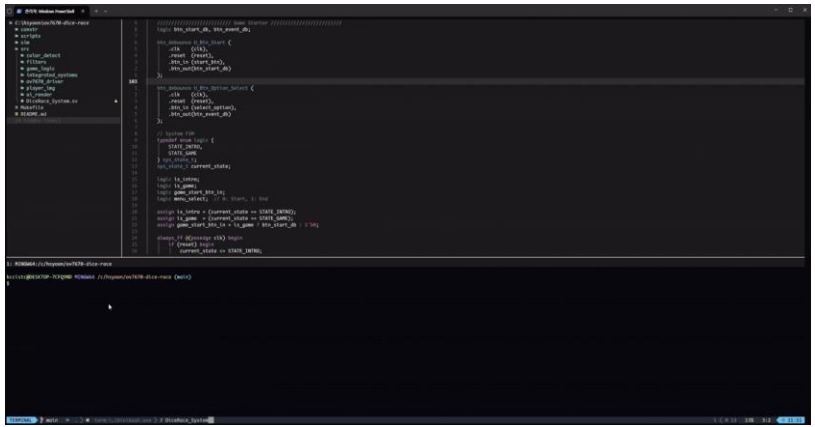
- 일관성이 유지되지 않고 디버깅 및 통합 과정에서 불필요한 반복 작업이 발생
- 프로젝트 구조가 복잡하고 다양한 파일이 존재하기 때문에, 디버깅 중 특정 시점의 빌드 상태를 재현하기 어려움
- 변동되는 파일을 개별 관리하는 방식은 충돌 가능성이 높고 협업에 적합하지 않음

05 Trouble Shooting & Insights (Co-Working, Build-Automation)

Insight : Building Co-Working Development Environment

[사전 대응 전략 및 결과]

- 팀 전체가 동일한 개발 환경을 유지할 수 있도록 디렉토리 구조를 표준화하고 모든 소스를 git 저장소에서 관리
- Makefile과 TCL Script 를 작성해 make 단일 명령으로 프로젝트 생성부터 bitstream 생성까지 Script 기반으로 전체 빌드 과정을 자동화
- 이를 통해 구성원 간 개발 환경 편차를 줄이고, 코드 개선 및 통합이 구조적으로 용이해졌으며, 반복 작업을 제거하며 전체 협업 효율이 크게 향상됨



Conclusion

06 Conclusion

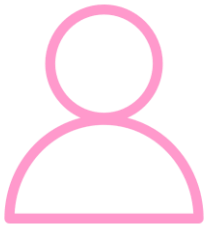
김준회

역할 : Color Detecting Module 설계, ASCII 필터 설계, 프로젝트 발표

소감 : 필터 설계, color detecting 모듈, voting 알고리즘을 활용한 noise 제거 등의 구조를 직접 설계해보며 **하드웨어 설계 역량을 강화**할 수 있었습니다. 특히, Github 기반 환경에서 협업 환경을 구축하고 코드 버전 관리를 효율적으로 수행함으로써 **효율적인 협업 경험**을 쌓을 수 있었습니다.

역할 : SCCB Interface 구현, Dual Camera VGA Controller 설계, Invert/Mosaic/Kaleidoscope 필터 설계

소감 : SCCB Interface를 직접 구현하며 카메라 레지스터를 제어하는 과정에서 **하드웨어 제어와 통신 프로토콜에 대한 깊이 있는 이해**를 쌓을 수 있었습니다. 특히 프로젝트 초기 전체 시스템 아키텍처를 함께 설계하고 Top-level 인터페이스를 먼저 정의한 뒤 각자의 모듈을 구현하는 협업 과정이 인상적이었고, **체계적인 설계 프로세스의 중요성**을 체감할 수 있었습니다.



변지인

06 Conclusion

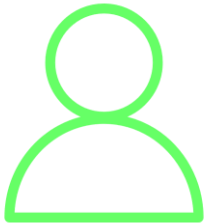
장준우

역할 : 조장, UI & Visualizing, Intro , Player_controller 설계

소감 : 소프트웨어와 달리 타이밍 동기화가 필수적인 환경을 다루며, player_controller에서 매 프레임 좌표를 갱신하며 **하드웨어적으로 실시간 애니메이션을 구현**했던 과정이 가장 기억에 남습니다. 또한 **Git과 빌드 스크립트**를 활용한 협업 으로 완성도 높은 결과물을 만들 수 있었습니다.

역할 : Game Logic 모듈 설계, Top-Level Integration, Script 기반 빌드 자동화 및 git 협업 환경 구축

소감 : 프로젝트의 본래 목표는 필터 설계였기 때문에, **게임로직 설계자로서 어떤 방식으로 필터 효과가 자연스럽게 드러나도록 할지를 고민**해야 했습니다. 팀원 간의 아이디어를 빠르게 공유하고 반영하는 과정이 잘 이루어져, 설계 목표를 충실히 달성한 결과물을 만들어낼 수 있었습니다.
또한 **여러 모듈을 통합하는 과정에서 상위 레벨에서의 프로토콜 정립이 시스템 설계의 핵심**임을 깨닫게 되었습니다.

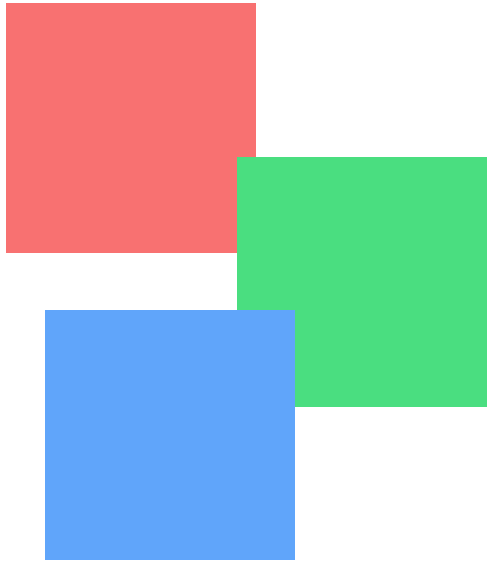


윤호승

Team 9: 김준회, 변지인, 윤호승, 장준우 | 2025. 12. 05.

Q & A

Thank you for listening



Git Repository: https://github.com/yhs1202/ov7670-dice-race