DX11 포트폴리오

김성훈 010 - 3251 - 6989

Blog	https://hooony23.tistory.com/category/GameDevLog/Portfolio
GitHub	https://github.com/hooony1324
Mail	hooony1324@gmail.com

Portfolio

Part1. KatanaZero
- DX11, 개인

Part2. Fall Guys
- DX11, 팀

DevLog

- 2021.06
  Unity졸업작품
- 2022.05
  뱀파이어 서바이벌 모작
- 2022.06
  Worms Armageddon모작
- 2022.08
  KatanaZero모작
- 2022.12
  FallGuys모작

https://www.youtube.com/@cogason1616/videos

Part1 Katana Zero

YouTube	GitHub

https://www.youtube.com/watch?v=bF8dJTWS-kE

https://github.com/hooony1324/KSH_KatanaZero_DX11

개발 인원

1인

개발 기간

2022/07/11 ~ 2022/08/14

작업 내용

- FSM
- 쉐이더 프로그래밍
- 역재생
- 적의 플레이어 추격

GameStateManager 클래스

상태 중복을 피하고 상태 변경을 명확하게 하기 위함

GameStateManager

- CreateState()
- UpdateState()
- ChangeState()

```
std::map<std::string, GameEngineState> AllState;
GameEngineState* CurState;
```

std::map으로 관리, 상태중복 방지

GameEngineState

- std::function **Start**
- std::function **Update**
- std::function **End**

State마다 Start/Update/End함수 바인딩할 수 있도록

Part 1 FSM

PlayerState, Play상태 의 관리

Animation State

- 애니메이션 State
- 충돌 State

PlayLevel

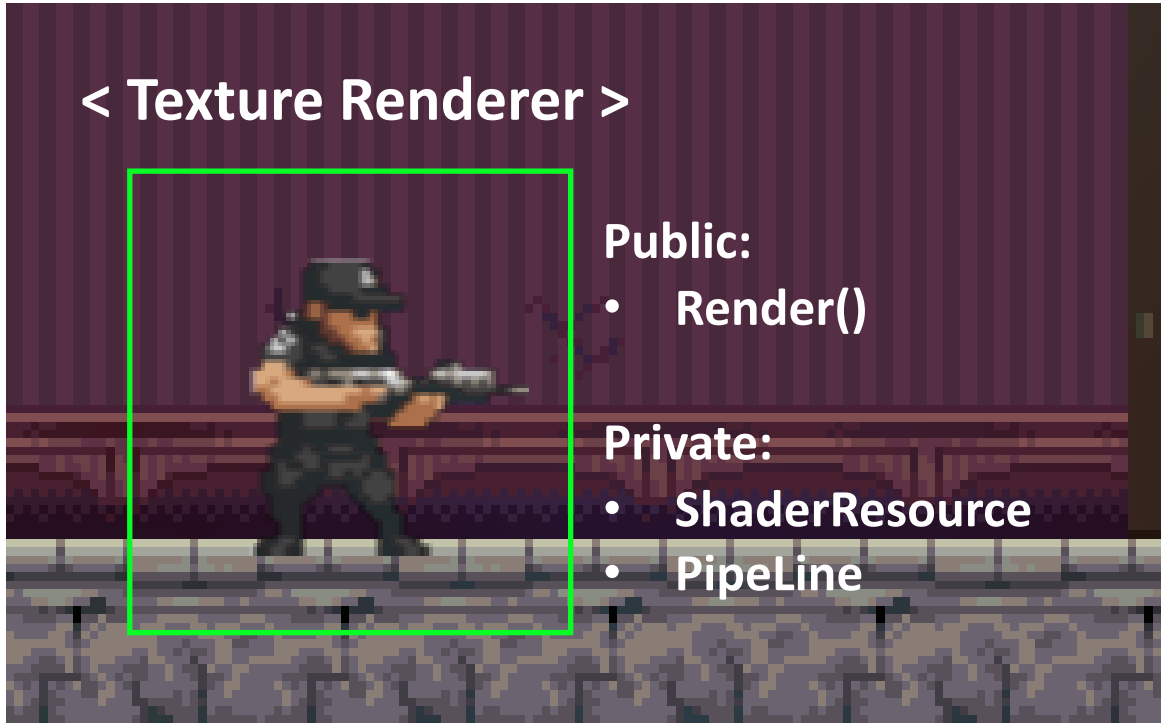
Play Level의 State

현재 플레이 중

- **Room1**
- **Room2**
- **Room3**
- **…**

- **RoomChange**
  다음 방으로 변경
- **RoomInit**
  방 정보 초기화
- **RoomPlay**
  플레이 업데이트
- **RoomExit**
  다음 방
  변경 직전 처리
- **RoomRestart**
  같은 방을 재시작
- **RoomReverse**
  Restart전
  역재생플레이

쉐이더 프로그래밍



Render()

ShaderResource와 PipeLine을 이용해서
RenderTarget에 메쉬 1개를 그림

ShaderResource

PipeLine에 해당하는 리소스 관리
(렌더링할 텍스쳐 리소스 이름, 상수버퍼, 샘플러 종류 등)

PipeLine

PipelineStage(InputAssembler ~ OutputMerger)수행

Part 1

쉐이더 프로그래밍

바닥찍힘 표현 쉐이더

```
Texture2D Tex : register(t0);
SamplerState Smp : register(s0);
#define ConcavePlayerRange 0.1
#define ConcaveBossRange 0.05
float4 TextureConcave_PS(Output _Input) : SV_Target0
{
    float2 TexPos = _Input.Tex.xy;

    // 플레이어
    if(ConcaveByPlayerPos.y >= TexPos.y && ConcaveByPlayerPos.y > -1.0f)
    {
        float DistanceRatio = abs(TexPos.x - ConcaveByPlayerPos.x);
        if (DistanceRatio <= ConcavePlayerRange)
        {
            TexPos.y += -ConcavePlayerRange + DistanceRatio;
        }
    }

    // 보스 주사기
    if (ConcaveByBossPos.y >= TexPos.y && ConcaveByBossPos.y > -1.0f)
    {
        float DistanceRatio = abs(TexPos.x - ConcaveByBossPos.x);
        if (DistanceRatio <= ConcaveBossRange)
        {
            float Diff = (1 / ConcaveBossRange) * pow(TexPos.x - ConcaveByBossPos.x, 2);
            TexPos.y += -(ConcaveBossRange * ConcaveStrength) + Diff * ConcaveStrength;
        }
    }

    float4 Color = Tex.Sample(Smp, TexPos);

    // 색상 보정
    if (Color.r <= 0.05f)
    {
        Color.r = 0.2f;
        Color.g = 0.1f;
        Color.b = 0.1f;
        Color.a = 0.7f;
    }

    return Color;
```

사용한 ShaderResource

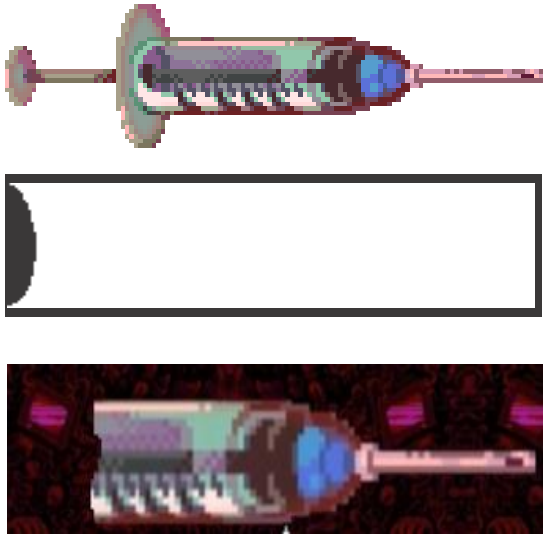
플레이어와 바닥이 충돌한 좌표를 이용
좌표 기준으로 위로 볼록하게 샘플링

< 바닥찍힘 연출 >

연출 결과물

< Post Processing >

< 애니메이션 마스킹 >



< 배경 SinWave >

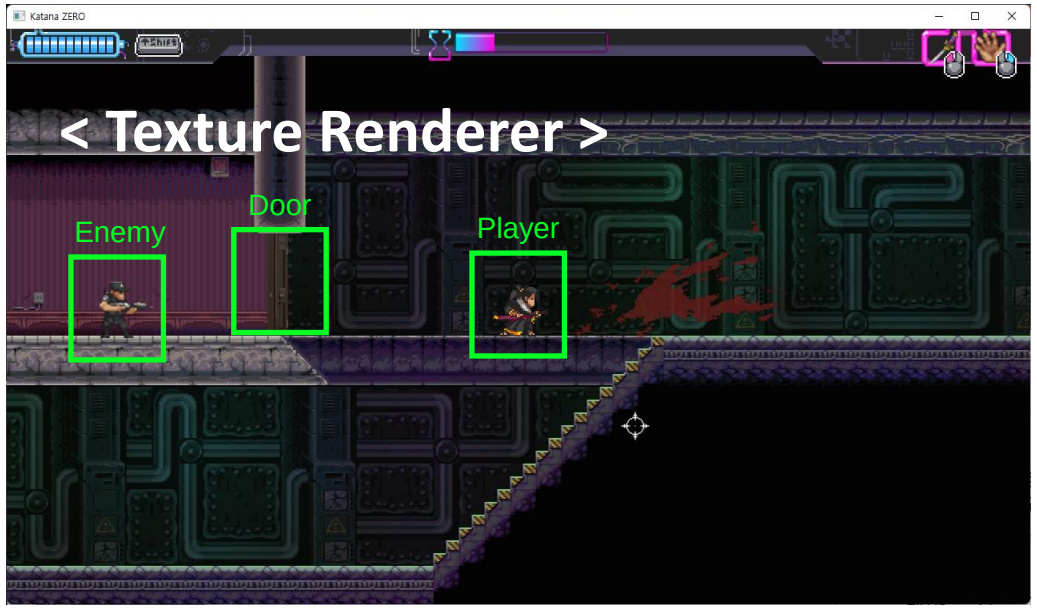
< 바닥찍힘 연출 >

Part 1

역재생

일정 타이밍마다 TextureRenderer Capture

역재생 순간, 캡처 정보 리스트를 뒤에서부터 POP



CapturedData Info

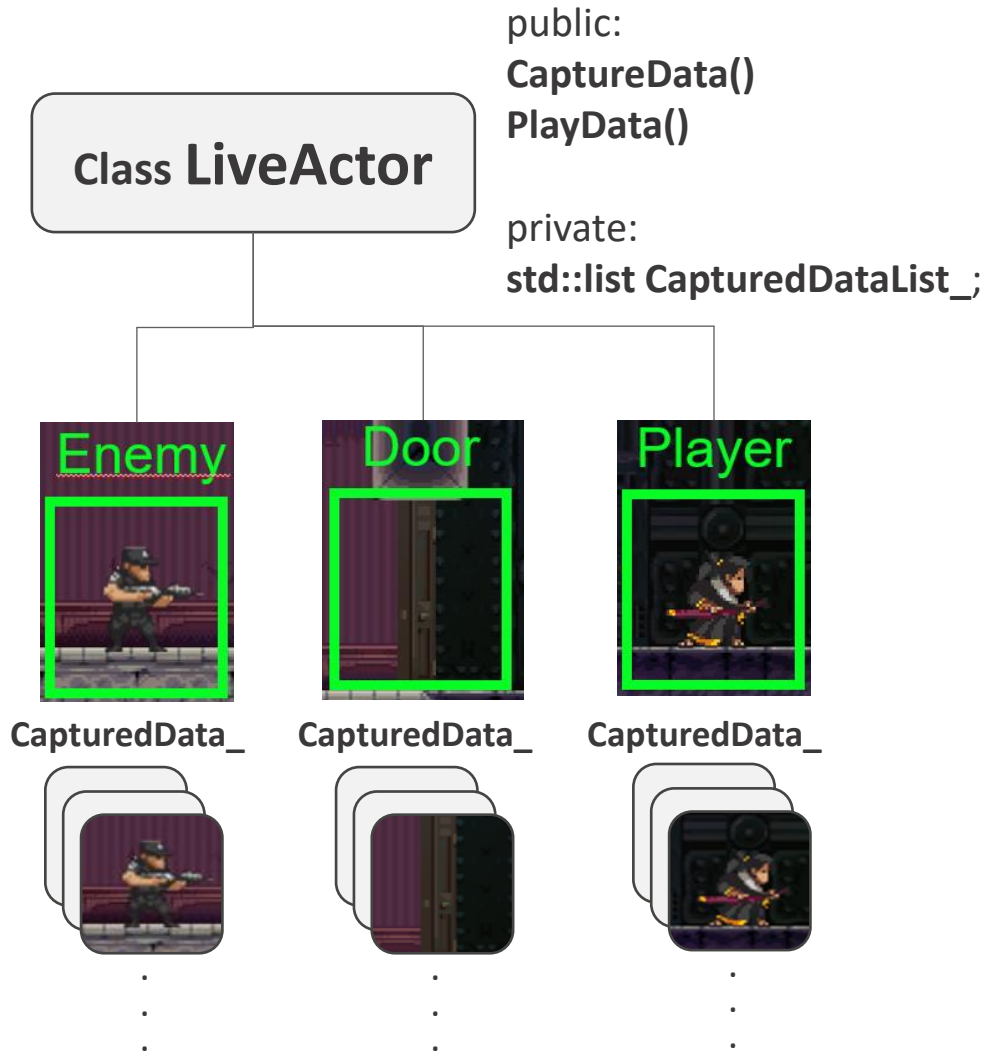
텍스처 리소스
- 애니메이션의 캡처 순간 텍스처

렌더러 크기
- 좌우 반전 판별용

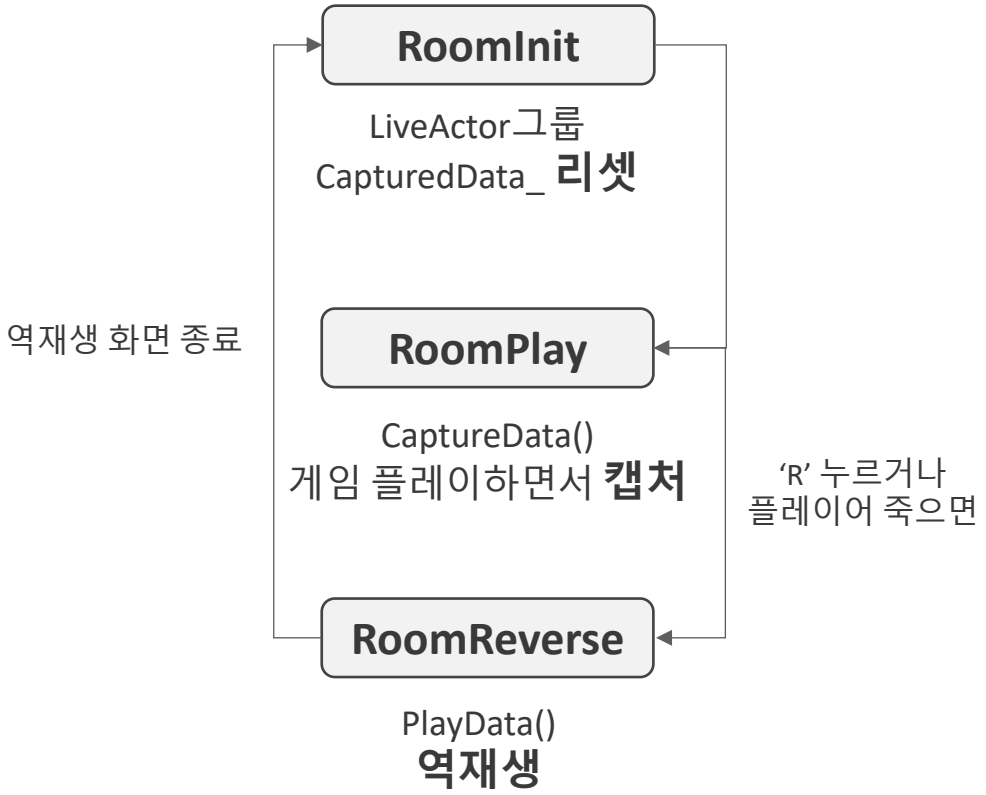
위치
- 캡처 순간의 위치

Part 1

역재생



Play Level의 State

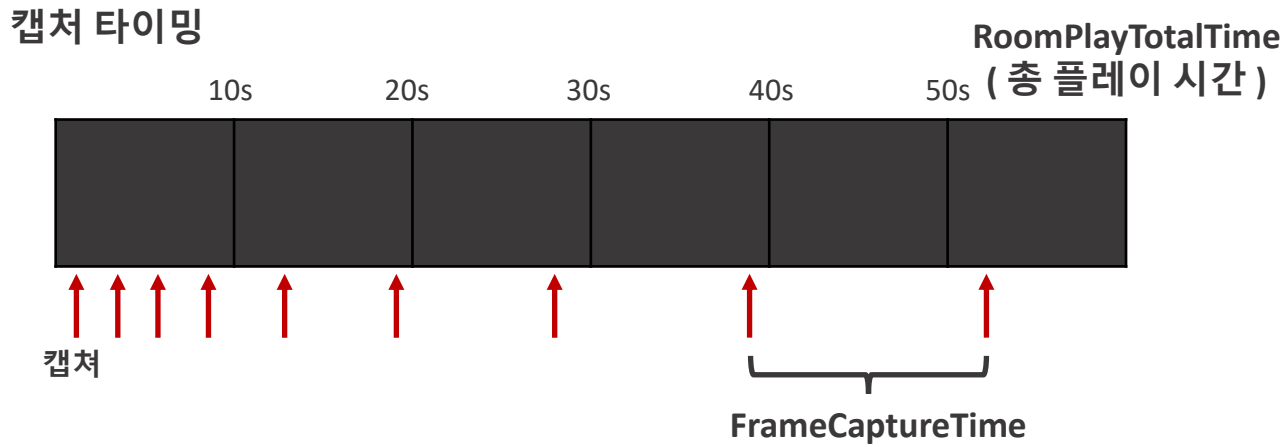


캡처 타이밍 조절

50초 플레이, 역재생 영상 50초 재생은 너무 김

```
// 역재생용 프레임 저장
FrameCaptureTime = 0.0164f * RoomPlayTotalTime;
if (ShotFrameTime >= FrameCaptureTime)
{
    for (LiveActor* Actor : CaptureGroup)
    {
        Actor->PushFrameCpaturedData();
    }
    ShotFrameTime = 0.0f;
}
```

0.0164f : 기본 프레임 캡처 타이밍, 60fps재생 기준



플레이 시간이 늘어날 수록 캡처 타이밍 늘려서 역재생영상의 총 재생시간을 줄임

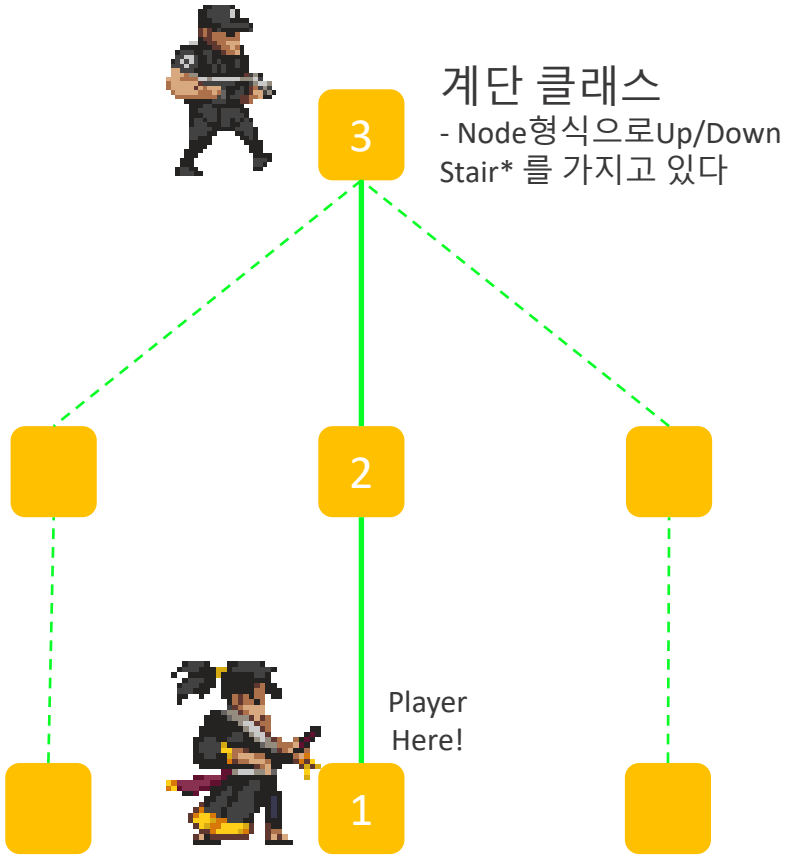
Part 1

적의 플레이어 추격

계단 클래스와 FSM이용하여 추격 구현

Part 1

적의 플레이어 추격



Enemy가 거쳐야 되는 계단 리스트 생성

1. PlayerHere!을 외치고 있는 계단부터 한 단계씩 올라가며 Enemy가 있는지 확인
2. Enemy와 같은 층에 있는 계단까지 리스트에 저장(1->2->3)

추격 & 공격

1. Enemy는 계단 리스트를 통해(3->2->1) 플레이어가 있는 층까지 도달
2. 플레이어와 같은 층 확인하면 공격 범위까지 이동 후 공격

Part2 Fall Guys

YouTube	GitHub

https://www.youtube.com/watch?v=dxX4zYDZWt0

https://github.com/jmin0220/3DTPortfolio

개발 인원

6인

개발 기간

2022/07/11 ~ 2022/08/14

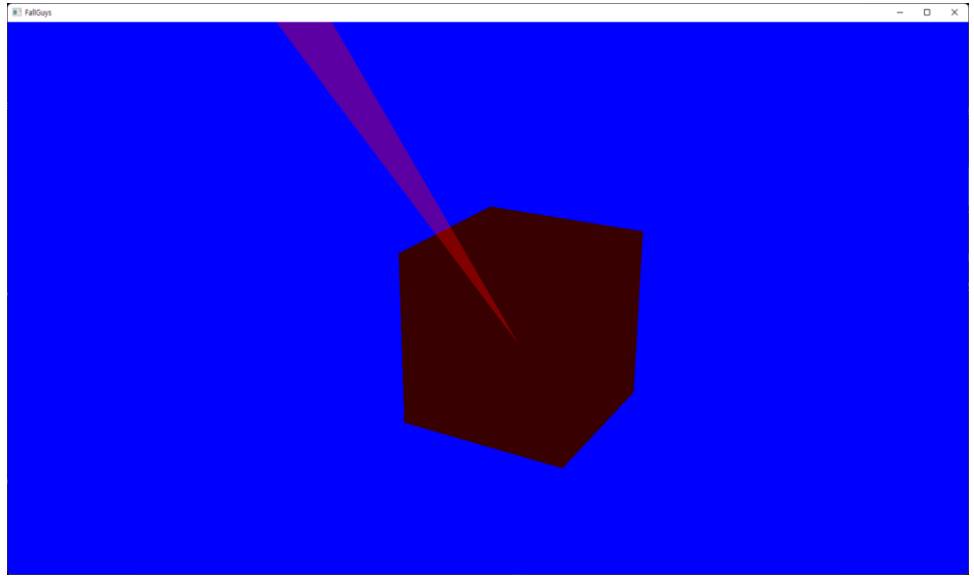
작업 내용

- 엑터 피킹
- 카메라암
- 캐릭터 스킨
- 멀티스레드 로딩
- 서버엔진 적용

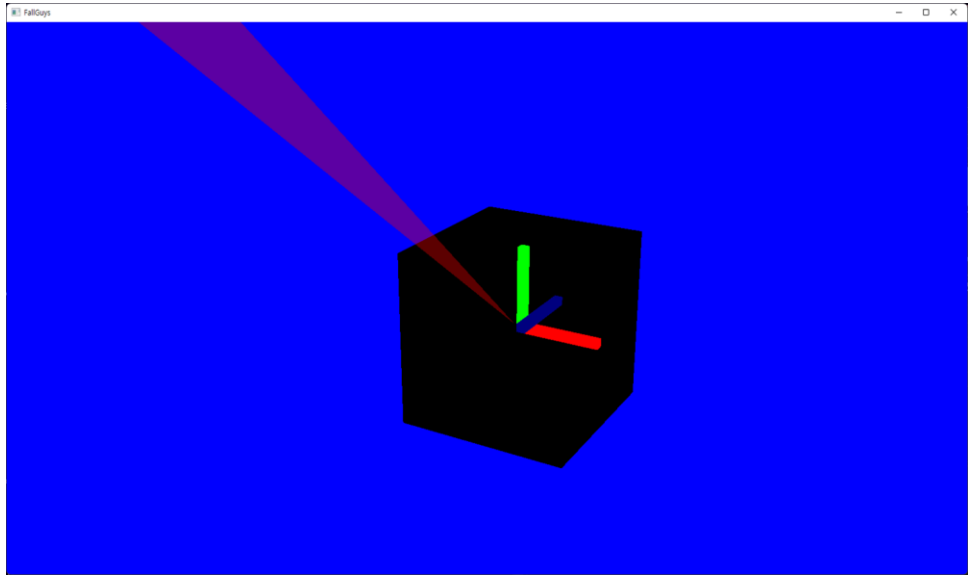
Part 2

엑터 피킹

피킹 레이저 & 피킹 후 위치 편집 기능 구현

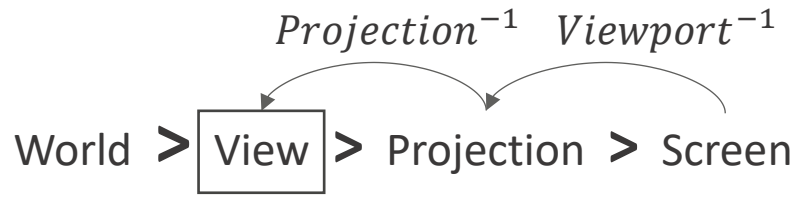


- 카메라 위치 기준 레이저 생성&회전
- 레이저 충돌 시 피킹 대상으로 설정

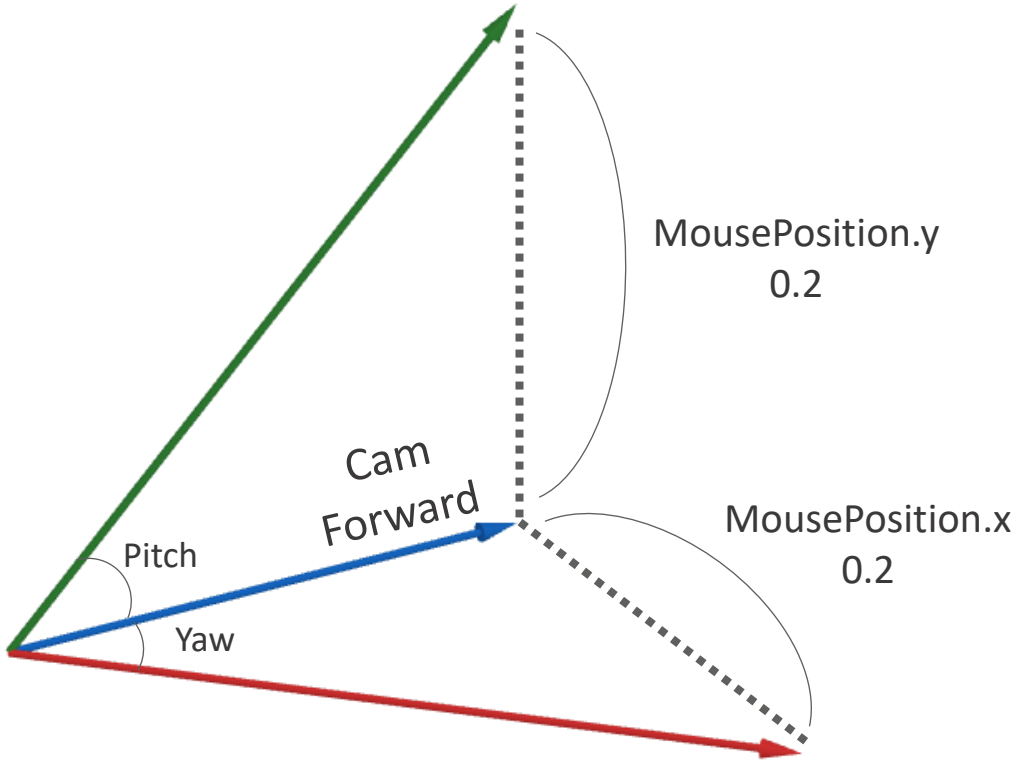


- 클릭하면 위젯 활성화
- 위젯 드래그 시 축을 따라 움직임

레이저 회전각 계산



View Space



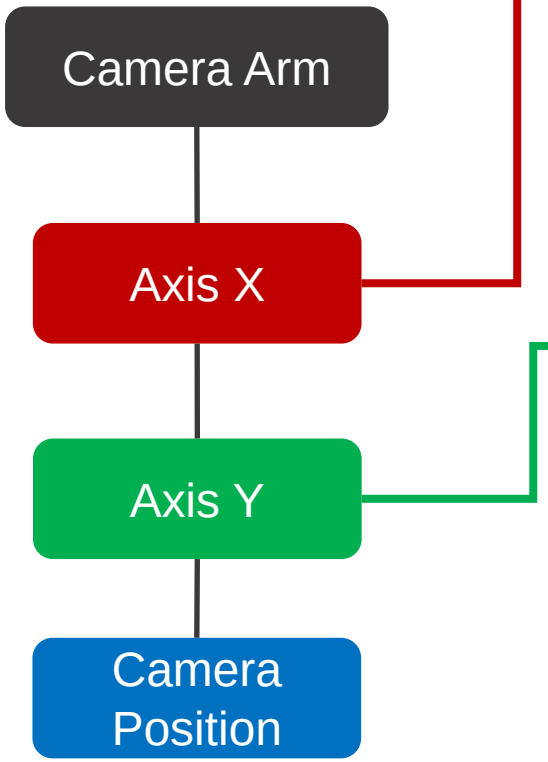
Yaw Angle = $\sin^{-1} MousePosition.x$

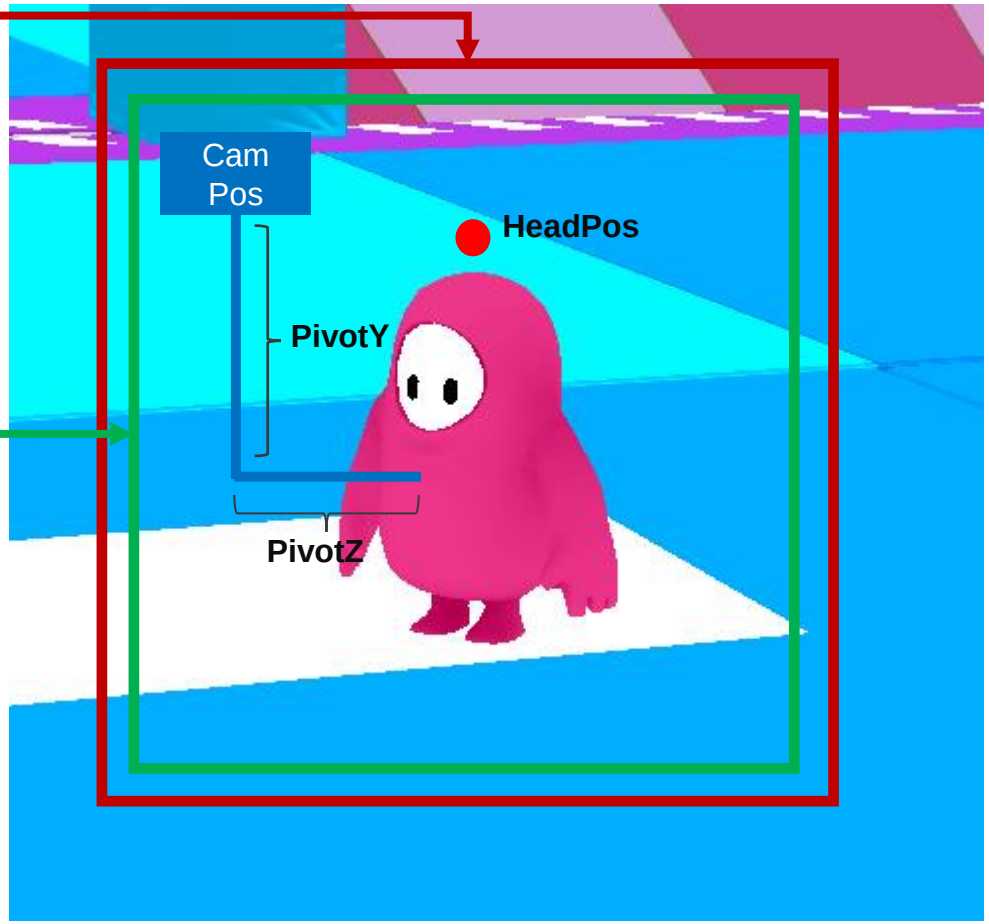
Pitch Angle = $\sin^{-1} MousePosition.y$

Part 2 카메라암

3인칭 플레이어 카메라암 구현

계층구조





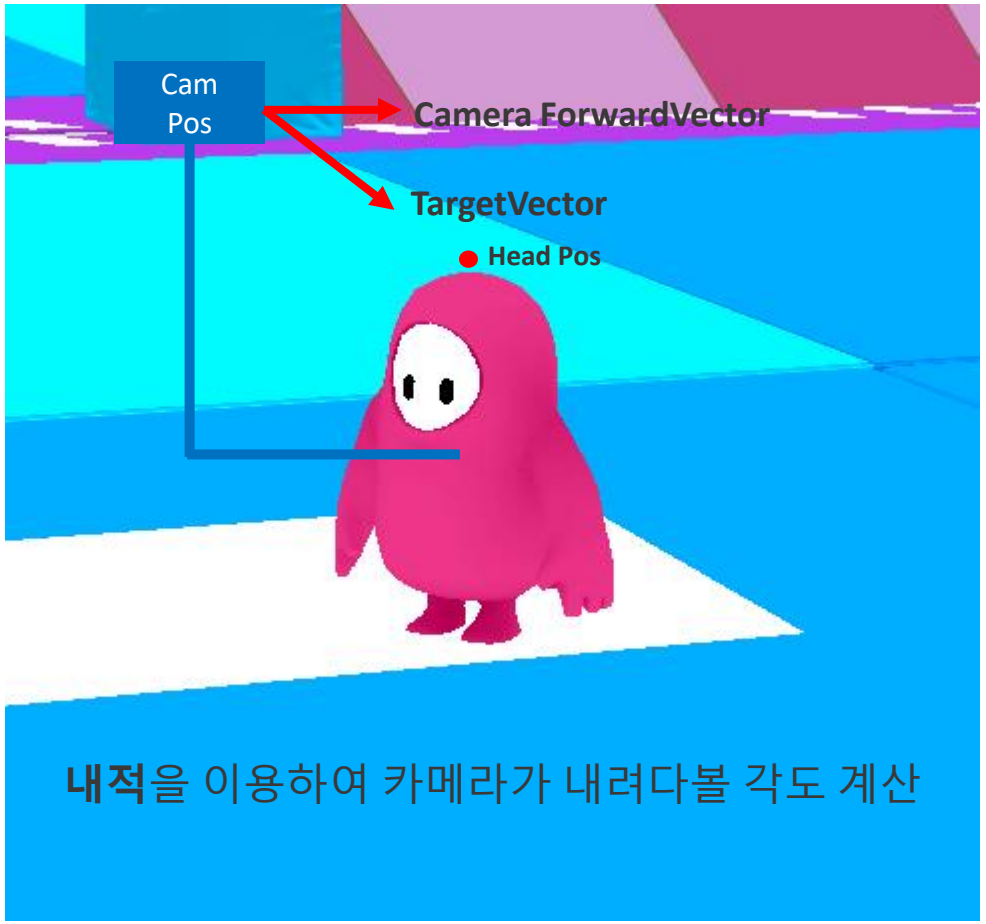
- Update Tick 마다 Camera를 CameraPosition위치로 세팅
- HeadPos를 이용하여 플레이어를 내려다볼 각도 계산
- Pivot 벡터로 Zoom In/Zoom Out 구현

Part 2

카메라암

플레이어를 바라보기

PivotY, PivotZ, HeadPos를 이용하여
플레이어가 화면의 중앙 보다 조금 아래에 위치

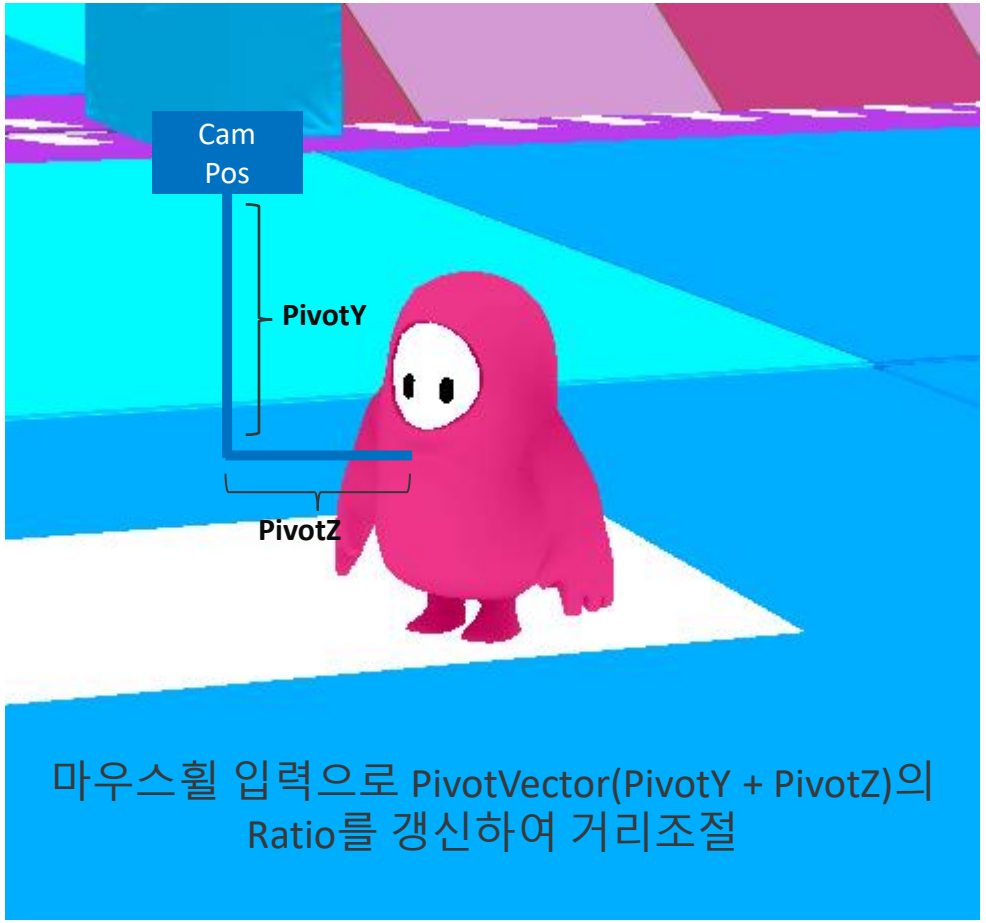


내적을 이용하여 카메라가 내려다볼 각도 계산

Part 2

카메라암

Zoom In/Zoom Out

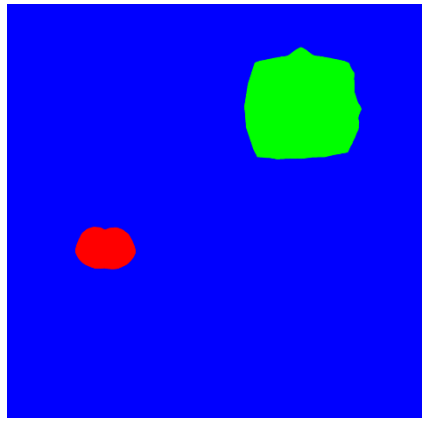


Part 2 캐릭터 스킨

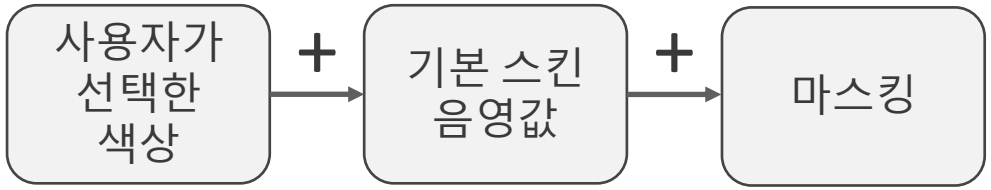
간단한 스킨 색상 지정 구현

기본 스킨 리소스

마스킹용 리소스



Color Result

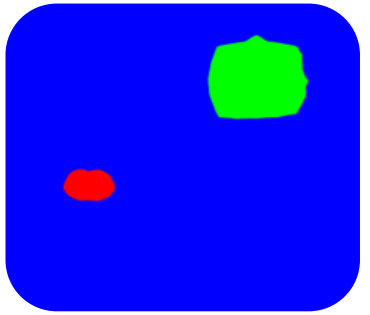


Part 2 캐릭터 스킨

스킨 쉐이더 색상 계산 과정

```
float4 TextureAnimation_PS(Output _Input) : SV_Target0
{

    float2 TexPos = _Input.TEXCOORD.xy;
    float3 TexColor = DiffuseTexture.Sample(LINEARWRAP, TexPos);

    // 마스킹
    // 눈
    float3 MaskColor = FaceEyeMskTexture.Sample(LINEARWRAP, TexPos);
    if (MaskColor.r > 0 )
    {
        TexColor = float3(0, 0, 0);

    }
    // 얼굴
    if (MaskColor.g > 0)
    {
        TexColor = float3(1, 1, 1);
    }
    // 몸
    if (MaskColor.b > 0)
    {
        // 결과    음영(DiffuseTex)  +   사용자입력
        TexColor.r = TexColor.r - 1 + TextureColor.r;
        TexColor.g = TexColor.g - 1 + TextureColor.g;
        TexColor.b = TexColor.b - 1 + TextureColor.b;

    }

    return float4(TexColor.rgb, 1.0);
}
```

DiffuseTexture



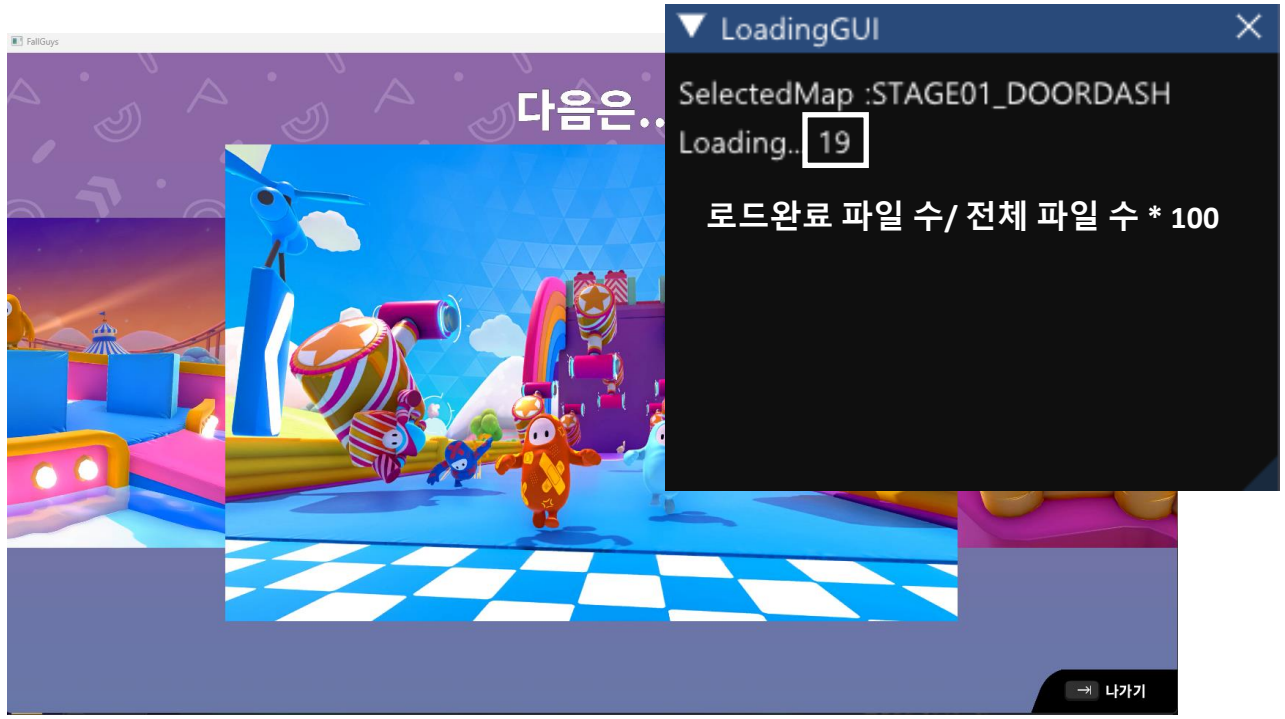
FaceEyeMskTexture

음영적용 후
사용자 색상을 더함

Part 2

멀티스레드 로딩

로딩화면과 리소스 로딩 병렬 처리



#include <filesystem>

LevelResourceLoad()

① 리소스 로드할 폴더경로 설정 & 폴더 내 파일 수 확인

.FBX파일만
전체 파일 수 카운트

↓

② 파일 로드 시작

1개 파일로드 완료 시
로드완료 파일 수++

클라이언트에 적용을 위한 작업

ServerEngine

클라이언트 IP설정하면
패킷 전송까지 구현되어 있는 상태

① 송수신 패킷들의 타입 정의 및 관리

```
enum class ContentsPacketType
{
    ClientInit,
    ObjectUpdate,
    GameState,
    PlayerState,
}
```

② 게임 오브젝트와 게임상태 동기화

- 오브젝트 동기화
- 게임 흐름 동기화

서버엔진 적용 ① 송수신 패킷들의 타입 정의 및 관리

어떤 패킷을 보낼지는 콘텐츠에서 정의 해야됨

```
enum class ContentsPacketType
{
    ClientInit,
    ObjectUpdate,
    GameState,
    PlayerState,
}
```

ClientInit

- 게임유저의 서버ID 관리

ObjectUpdate

- ObjectID관리
- 플레이어, 맵의 장애물, 대포 등의 Transform, Animation동기화

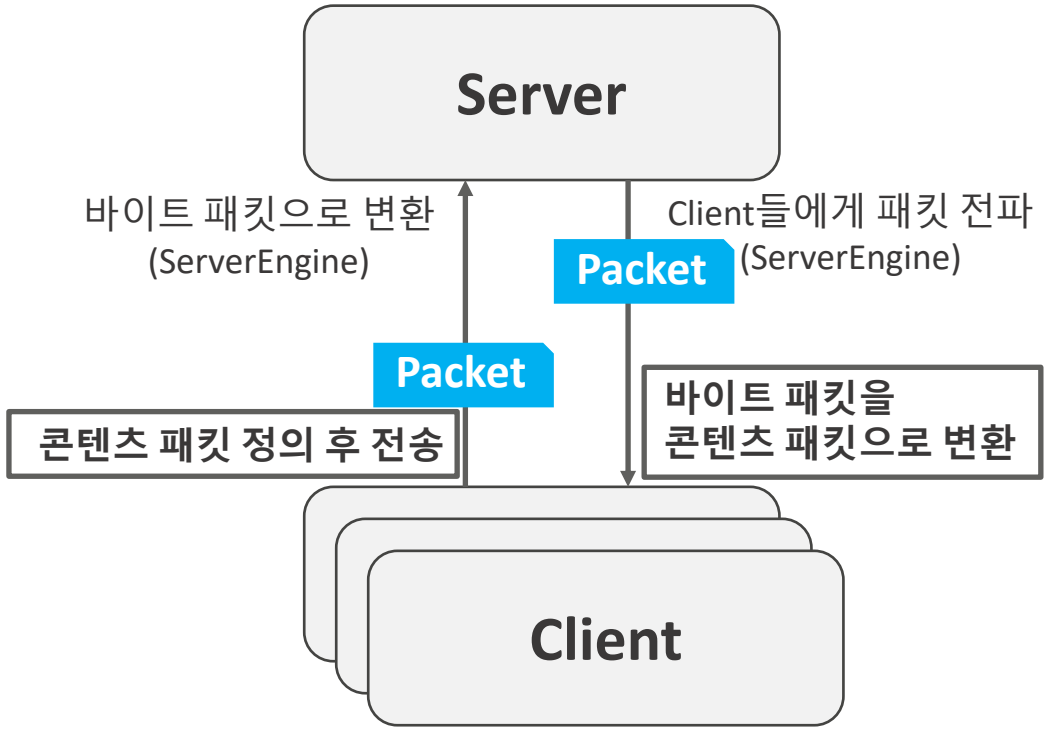
GameState

- 리소스로딩 > 게임 플레이 > 점수집계 등의 게임 상태 동기화

PlayerState

- 유저가 정의한 플레이어 스킨, 게임에서 얻은 점수 등의 정보를 공유

정의한 콘텐츠 패킷을 송수신하는 과정



서버 패킷 정보

- **PacketID**
  콘텐츠 패킷 타입 Int형으로 Cast
  클라이언트가 서버에서 받은 패킷의 타입을 구분함
- **PacketSize**
  패킷 크기
- **MasterSocket**
  송신 후 자신이 보낸 패킷이 역전파 되지 않도록 방지

서버엔진 적용 ① 송수신 패킷들의 타입 정의 및 관리

타입별로 패킷을 처리할 함수 관리

서버에서 받은 패킷을 재조립할 함수 관리

```
std::map<int, std::function<void(std::shared_ptr<GameServerPacket>)>> PacketProcessMap;
```

Part 2

서버엔진 적용 ② 게임 오브젝트와 게임상태 동기화

게임 오브젝트 동기화

① ObjectUpdate 패킷들을 저장

Packet

< ObjectUpdate > 처리용 Func

std::list ObjectPacketList_

Packet

② 패킷을 일치하는 ID의 객체에 전달

Packet

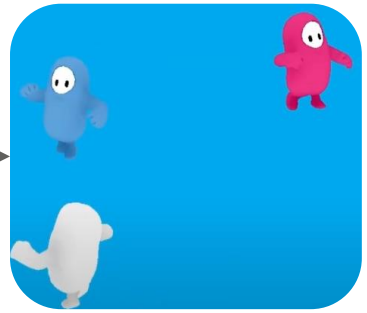
같은 ID를 가진 오브젝트가 생성되어 있는가?

YES

NO

오브젝트 소환

③ 각 객체마다 자신의 패킷리스트에서 POP하여 처리



Character

Character::Update()

- Transform세팅
- Animation세팅
- …

SpinBar

SpinBar::Update()

- Transform세팅

Cannon

Cannon::Update()

- 발사 타이밍
- 발사할 대포알 정보 (탄 종류, 발사방향)

*Atomic & Static으로 생성한 오브젝트의 ID의 중복 방지

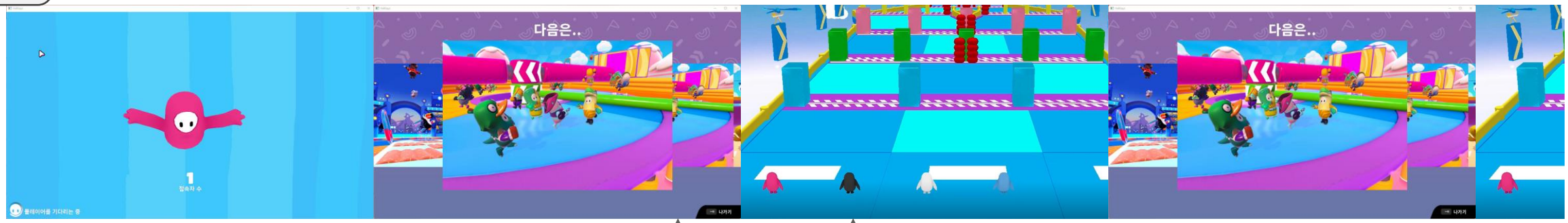
```
static std::atomic<int> IdSeed;
static std::map<int, GameServerObject*> AllServerObject;
```

Part 2

서버엔진 적용 ② 게임 오브젝트와 게임상태 동기화

게임 상태 동기화

게임 상태



Server(Host) State

Client State

< GameState > 처리용 Func

서버가 유저들에게 전파할 서버의 게임 상태

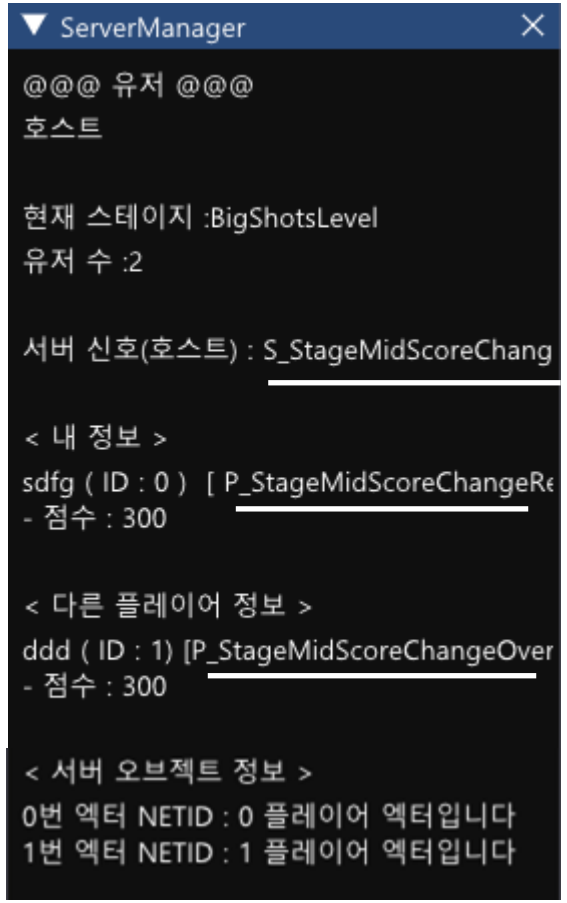
< PlayerState > 처리용 Func

유저들의 진행 중인 게임 상태

Part 2

서버엔진 적용 ② 게임 오브젝트와 게임상태 동기화

상태 변경 신호 종류

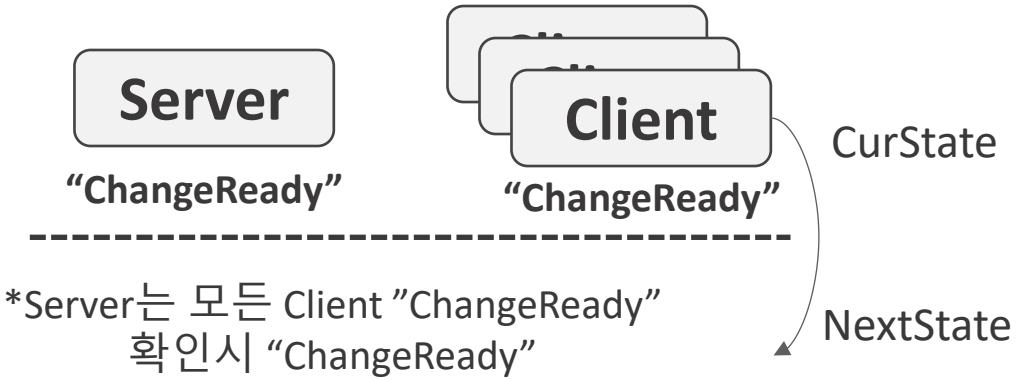


"GameState" + **ChangeReady**
상태변경 시작 신호

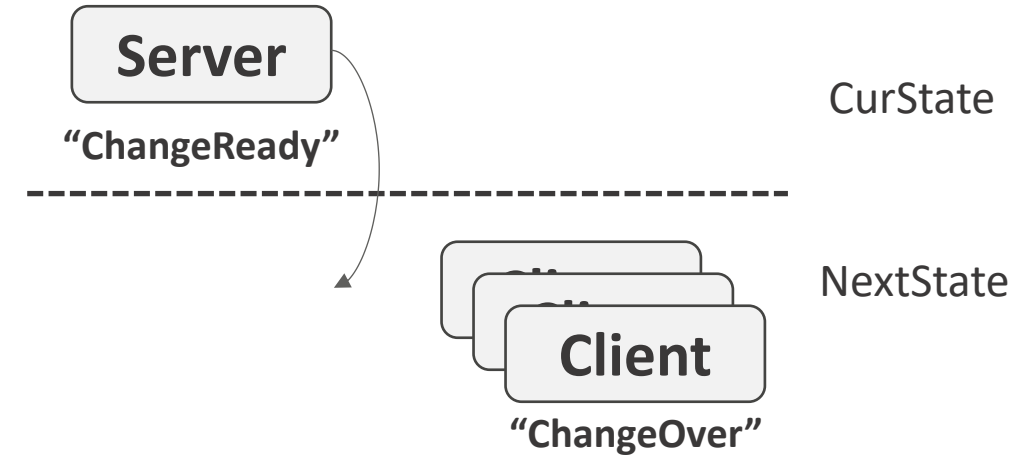
"GameState" + **ChangeOver**
상태변경 완료 신호

서로의 신호를 확인하며 상태 변경

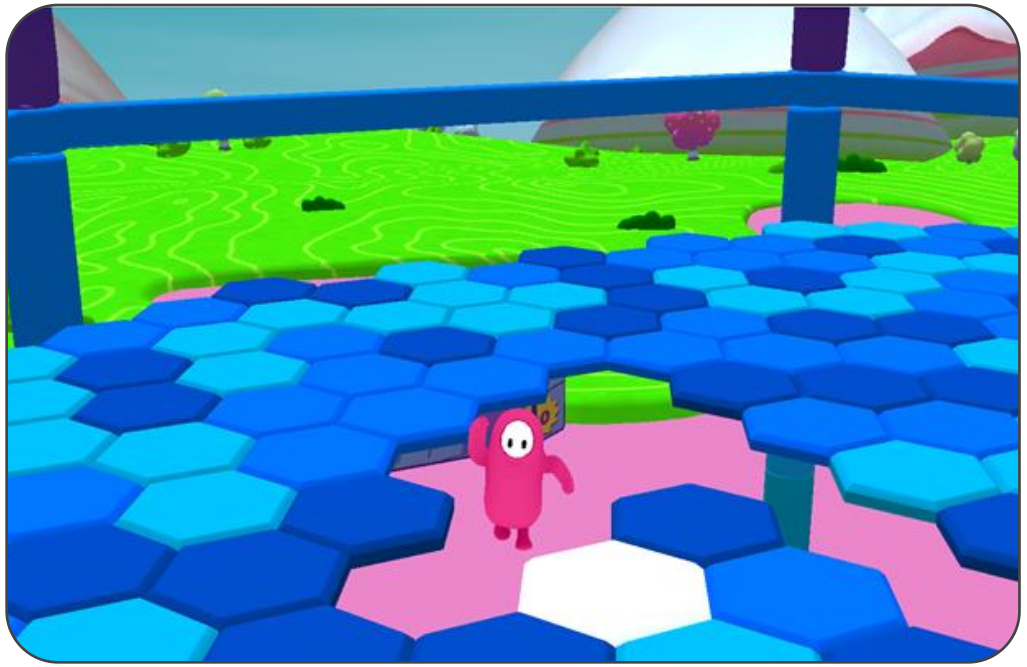
① 서버 "ChangeReady"되면 클라이언트가 먼저 넘어감



② 모든 클라이언트 "ChangeOver"확인, 서버 넘어감



기타 기능



관전

사망 후 좌 클릭시
다른 유저 관전 가능

점수집계

플레이어들의 점수를 관리
하여 각 레벨에 맞게 정보 전송