네트워크 게임 프로그래밍 2 팀

Term Project

WindowPro

김태순, 김지호, 김한준

2025-10-30

개요

"WindowPro"는 2D플랫포머 게임으로 플레이어의 위치에 따라 윈도우 창이 이동하며 윈도우 창과의 상호작용을 통해 스테이지들을 클리어하는 것이 목적인 게임이다.

해당 게임은 2024년 1학기 윈도우 프로그래밍과목에서 김태순, 김준호가 제작한 게임이다.

애플리케이션 기획

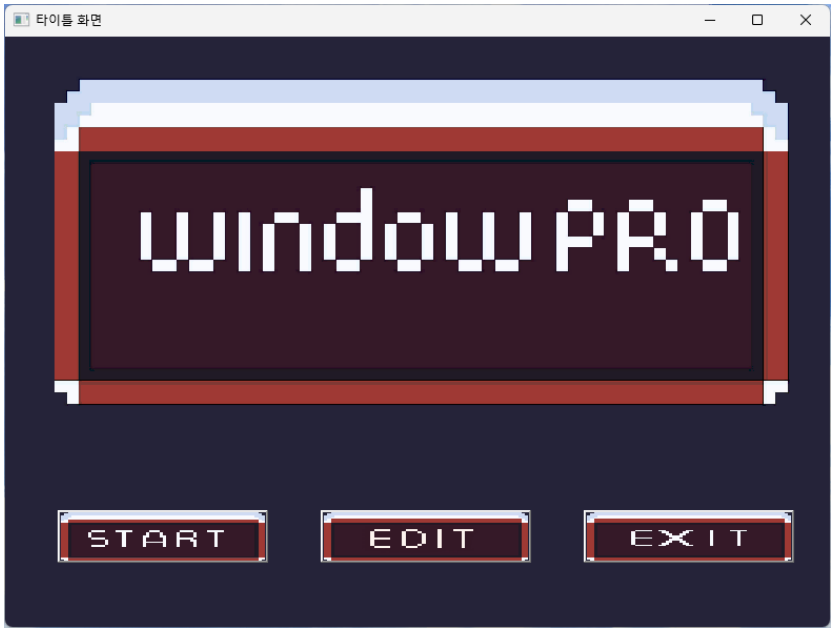
2D플랫포머 형식으로 플레이어는 방향키를 이용한 이동과 Ctrl키의 창 고정 기능, Shift키의 내려찍기 등을 이용해 장애물과 적을 피해 골대를 향해 전진하여 깃발을 얻으면 스테이지가 클리어되는 형태이다.

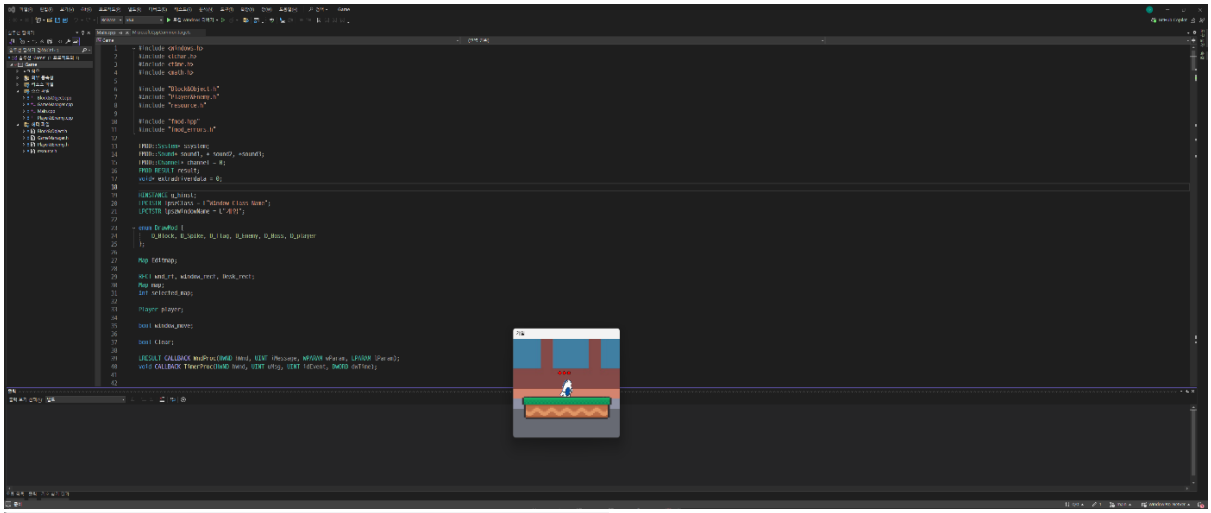
플레이어는 방향키로 이동, 및 점프 동작을 수행할 수 있고 창이 고정된 상태에서는 창의 벽면을 이용한 매달리기/벽 점프가 가능하고 내려찍기 상태에서는 적 객체와 충돌해도 목숨을 잃지 않고 적을 없앨 수 있다.

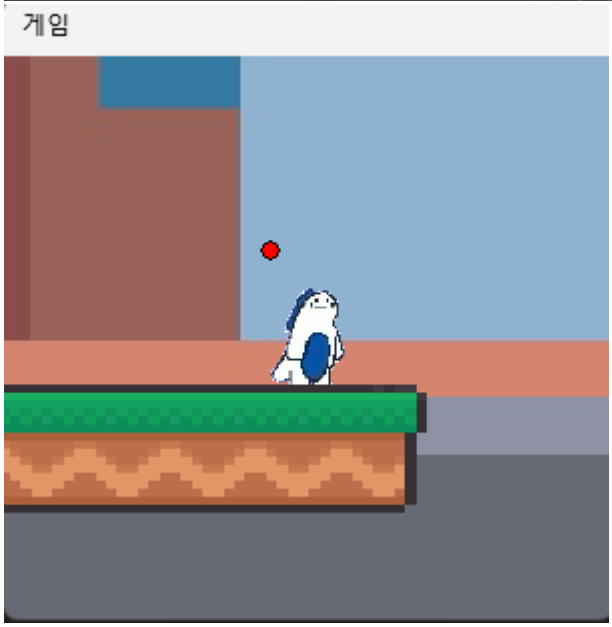
스테이지의 구성은 간단한 조작법을 익히는 1단계, 벽 점프를 이용해야 하는 2단계, 몬스터가 등장하여 몬스터를 공격을 해야 하는 3단계가 존재하고 보스전에서는 발판이 계속해서 이동하여 지속적인 이동과 벽 점프를 적극적으로 활용해야 하는 환경이 조성되어 있다.

보스 스테이지에서는 맵 끝에서 끝으로 이동하여 보스를 공격하여 총 3번의 피해를 입히면 게임이 클리어 되며 메인 화면으로 이동하고 로고가 "Clear"로 변하게 된다.

기존에는 한 명의 플레이어와 하나의 창을 이용한 게임이 진행되었는데 네트워크 기능을 추가하면 두명의 플레이어가 네트워크 통신을 통해 서로의 화면을 실시간으로 확인하며 경쟁이나 협동을 통해 게임을 더욱 다채롭게 즐길 수 있으며, Edit모드에서는 플레이어가 제작한 스테이지를 다른 플레이어들이 플레이할 수 있다.

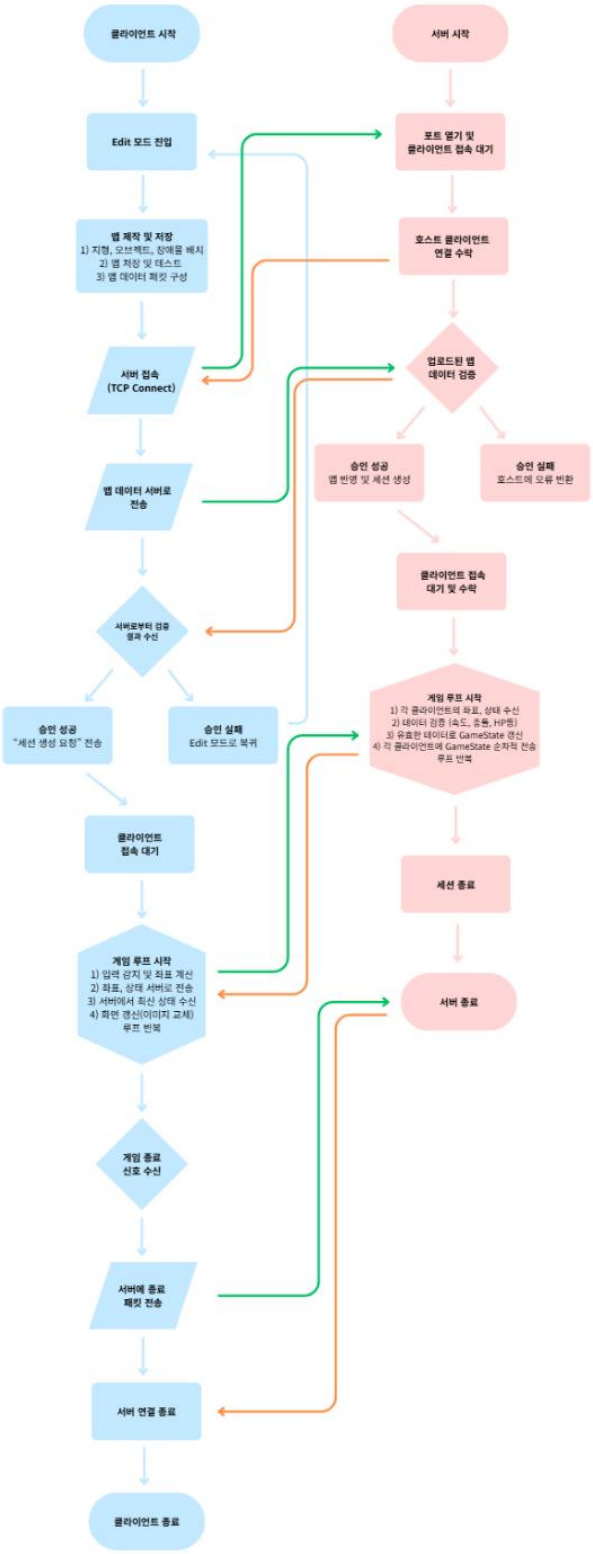






2025.10.30 스크린샷 이미지 추가

High-Level 디자인



2025.10.30 호스트 클라이언트 -> 일반 클라이언트로 변경(병합)

Low-Level 디자인

```
// --- 패킷 타입 정의 ---
// 클라이언트 -> 서버
constexpr char CS_UPLOAD_MAP = 0;            // 맵 데이터 업로드
constexpr char CS_START_SESSION_REQ = 1;   // 게임 시작 준비 요청
constexpr char CS_PLAYER_UPDATE = 2;        // 나의 위치 및 상태 정보 전송
constexpr char CS_END_SESSION_REQ = 3;      // 게임 세션 종료 요청

// 서버 -> 클라이언트
constexpr char SC_ASSIGN_ID = 10;            // [서버] 플레이어 고유 ID 할당
constexpr char SC_MAP_UPLOAD_RSP = 11;      // [서버->클라이언트] 맵 업로드 결과 응답
constexpr char SC_MAP_INFO = 12;             // [서버] 확정된 게임 맵 정보
constexpr char SC_GAME_STATE = 13;           // [서버] 모든 오브젝트의 최종 상태 (동기화용)
constexpr char SC_EVENT = 14;                // [서버] 게임 이벤트 (승리, 다음 스테이지 등)
constexpr char SC_DISCONNECT = 15;           // [서버] 플레이어 접속 종료

// --- 기본 구조체 및 열거자 ---
enum E_EventType { STAGE_CLEAR, GAME_WIN };
struct Point { int x; int y; };
enum class Direction { LEFT, RIGHT };
```

```
// --- 패킷 클래스 정의 ---
class BasePacket {
public:
    unsigned char size;
    char type;
};

// --- 클라이언트 -> 서버 ---
// 맵 데이터를 담아 서버로 업로드하는 패킷
class CS_UploadMapPacket : public BasePacket {
public:
    int platform_count;
    RECT platforms[160];
    int spike_count;
    RECT spikes[160];
    int enemy_spawn_count;
    Point enemy_spawns[32];
    Point player_start_pos;
};

// 게임 세션 시작 요청 패킷
class CS_StartSessionRequestPacket : public BasePacket {};

// 클라이언트가 자신의 위치와 상태를 계산하여 서버로 보내는 패킷
class CS_PlayerUpdatePacket : public BasePacket {
public:
    Point pos;                  // 클라이언트가 계산한 나의 현재 위치
    int Walk_state;             // 현재 걷기 상태 (애니메이션 동기화용)
    int Jump_state;             // 현재 점프 상태 (애니메이션 동기화용)
};

// 게임 세션 종료 요청 패킷
class CS_EndSessionRequestPacket : public BasePacket {};
```

```
// --- 서버 -> 클라이언트 ---
// 고유 ID를 할당하는 패킷
class SC_AssignIDPacket : public BasePacket {
public:
    int player_id;
};

// 맵 업로드 결과를 응답하는 패킷
class SC_MapUploadResponsePacket : public BasePacket {
public:
    bool is_success;
};

// 확정된 게임 맵 정보를 전달받는 패킷
class SC_MapInfoPacket : public BasePacket {
public:
    int platform_count;
    RECT platforms[160];
    int spike_count;
    RECT spikes[160];
    int enemy_spawn_count;
    Point enemy_spawns[32];
    Point player_start_pos;
};

// 서버가 최종적으로 계산한 '모두의 상태'를 담는 동기화용 패킷
class SC_GameStatePacket : public BasePacket {
public:
    // 플레이어의 최종 상태
    struct PlayerState {
        int life;                   // 현재 체력
        int Walk_state;             // 현재 걷기 상태
        int Jump_state;             // 현재 점프 상태
        int frame_counter;          // 현재 애니메이션의 프레임
    } players[2];

    // 적의 최종 상태
    struct EnemyState {
        bool is_alive;              // 생존 여부
```

```
        Point pos;                  // 서버가 계산한 최종 위치
        Direction dir;              // 바라보는 방향
    } enemies[32];

    // 보스의 최종 상태
    struct BossState {
        bool is_active;             // 활성화 여부
        Point pos;                  // 서버가 계산한 최종 위치
        int life;                   // 현재 체력
    } boss;
};

// 특정 이벤트를 알리는 패킷
class SC_EventPacket : public BasePacket {
public:
    E_EventType event_type;
};

// 다른 플레이어의 접속 종료를 알리는 패킷
class SC_DisconnectPacket : public BasePacket {
public:
    int disconnected_player_id;
};
```

```
// --- 통신 및 기본 처리 함수 ---
// 서버로부터 패킷을 수신하는 함수 (스레드에서 계속 호출됨)
bool DoRecv();

// 수신한 패킷의 종류에 따라 아래의 Handle... 함수들을 호출해주는 분류기
void ProcessPacket(char* packet);

// --- 서버로 패킷을 송신하는 함수 ---
// [모든 클라] 클라이언트가 계산한 자신의 위치와 상태를 주기적으로 서버에 송신하는 함수
bool SendPlayerUpdatePacket();

// 맵 데이터를 서버로 송신하는 함수
bool SendUploadMapPacket();

// 맵 승인 후, 게임 세션 시작을 서버에 요청하는 함수
bool SendStartSessionRequestPacket();

// 게임 종료 시, 서버에 세션 종료를 알리는 함수
bool SendEndSessionRequestPacket();

// --- 서버로부터 받은 패킷을 '실제로 처리'하는 핸들러 함수 ---
// (SC_ASSIGN_ID) 서버가 정해준 나의 ID를 변수에 저장하는 함수
void HandleAssignID(SC_AssignIDPacket* packet);

// (SC_MAP_UPLOAD_RSP) 맵 업로드 성공/실패 여부를 처리하는 함수
void HandleMapUploadResponse(SC_MapUploadResponsePacket* packet);

// (SC_MAP_INFO) 서버가 보내준 맵 정보로 내 클라이언트의 월드를 생성하는 함수
void HandleMapInfo(SC_MapInfoPacket* packet);

// (SC_GAME_STATE) 실시간 게임 상태 정보로 모든 캐릭터의 위치, 모습 등을 갱신하는 함수
void HandleGameState(SC_GameStatePacket* packet);

// (SC_EVENT) 레벨 클리어, 승리 등 순간적인 이벤트를 처리하는 함수
void HandleEvent(SC_EventPacket* packet);

// (SC_DISCONNECT) 상대방의 접속 종료를 처리하는 함수
void HandleDisconnect(SC_DisconnectPacket* packet);
```

```
// --- 네트워크 통신 처리 함수 ---
// 새로운 클라이언트의 접속을 받아들이는 함수
bool AcceptClient();

// 특정 클라이언트로부터 패킷을 수신하는 함수
bool DoRecv(int client_id);

// 수신한 패킷의 종류에 따라 아래의 Handle... 함수들을 호출해주는 분류기
void ProcessPacket(char* packet, int client_id);

// 모든 클라이언트에게 실시간 게임 상태를 전송하는 함수
bool SendStateUpdatePacket();

// 모든 클라이언트에게 확정된 맵 정보를 전송하는 함수
bool SendMapInfoPacket();

// 클라이언트에게 맵 업로드 결과(성공/실패)를 전송하는 함수
bool SendMapUploadResponsePacket(int client_id, bool is_success);

// 특정 플레이어의 접속 종료 사실을 다른 클라이언트에게 알리는 함수
bool SendDisconnectPacket(int disconnected_id);

// 특정 이벤트를 모든 클라이언트에게 전송하는 함수
bool SendEventPacket(E_EventType event_type);


// --- 클라이언트로부터 받은 패킷을 '실제로 처리'하는 핸들러 함수 ---
// (CS_UPLOAD_MAP) 클라이언트가 보낸 맵 데이터를 받아 유효성을 검사하는 함수
void HandleMapUpload(CS_UploadMapPacket* packet, int client_id);

// (CS_START_SESSION_REQ) 클라이언트의 게임 시작 요청을 처리하는 함수
void HandleStartSessionRequest(CS_StartSessionRequestPacket* packet, int client_id);

// (CS_END_SESSION_REQ) 클라이언트의 세션 종료 요청을 처리하는 함수
void HandleEndSessionRequest(CS_EndSessionRequestPacket* packet, int client_id);

// --- 서버 내부 게임 로직 함수 ---
// 모든 오브젝트(플레이어, 적, 지형 등) 간의 충돌을 검사하고 처리하는 함수
void CheckAllCollisions();
```

// 모든 동적 객체(플레이어, 적, 보스)의 위치를 AI 및 물리 법칙에 따라 업데이트하는 함수
bool UpdateAllPositions();

// 게임이 끝나는 조건(예: 최종 보스 사망)을 확인하는 함수
bool IsGameEnd();

멀티 스레드 구현 계획

클라이언트

스레드 구성

로직 처리 스레드: 사용자 입력 처리와 화면 출력을 담당한다.

네트워크 수신 스레드: 서버의 응답을 수신하고 게임 상태 데이터를 갱신한다.

로직 처리 스레드는 화면을 그리기 위해 게임 상태 데이터(플레이어, 적, 맵 등) 를 읽는다.

동시에 네트워크 수신 스레드는 서버로부터 받은 데이터를 이용해 같은 게임 상태 데이터를 수정한다.

이때 두 스레드가 동시에 공유 데이터에 접근하면 문제가 발생할 수 있다.

이를 방지하기 위해 임계 영역을 사용하여 한 번에 하나의 스레드만 접근하도록 보장한다.

서버

스레드 구성

로직 처리 스레드: 적 이동, 충돌 처리 등 게임 로직을 계산하고, 모든 클라이언트에게 최신 게임 상태를 송신한다.

네트워크 수신 스레드: 각 클라이언트의 요청을 수신하여 데이터를 갱신한다.

로직 처리 스레드는 게임 시뮬레이션을 수행하기 위해 게임 상태 데이터를 읽는다.

네트워크 수신 스레드는 클라이언트로부터 전달받은 정보를 바탕으로 데이터를 수정한다.

이때 두 스레드가 동시에 공유 데이터에 접근하면 문제가 발생할 수 있다.

이를 방지하기 위해 임계 영역을 사용하여 한 번에 하나의 스레드만 접근하도록 보장한다.

2025.10.29 멀티 스레드 관련 Low-Level 함수 추가

```
// ------------------------------------------------------------------
// [클라이언트] 로직 처리 스레드 (메인 스레드)
// - 입력 처리, 물리 업데이트, 렌더링 담당
// ------------------------------------------------------------------
DWORD WINAPI ClientLogicThread(LPVOID lpParam);

// ------------------------------------------------------------------
// [클라이언트] 네트워크 수신 스레드
// - 서버로부터 패킷을 수신하고 처리 담당
// ------------------------------------------------------------------
DWORD WINAPI ClientRecvThread(LPVOID lpParam);

// ------------------------------------------------------------------
// [서버]  로직 처리 스레드 (메인 스레드)
// - 전체 게임 로직 처리 및 상태 동기화 담당
// ------------------------------------------------------------------
DWORD WINAPI ServerLogicThread(LPVOID lpParam);

// ------------------------------------------------------------------
// [서버] 클라이언트 핸들러 스레드
// - 각 클라이언트의 패킷 수신 및 요청 처리 담당
// ------------------------------------------------------------------
DWORD WINAPI ClientHandlerThread(LPVOID lpParam);
```

역할 분담

김태순: 클라이언트 구조 재작성 및 동기화 작업

김지호: 서버 프레임워크 제작

김한준: 패킷 구조 설계 및 구현

개발 환경

개발 도구 및 언어 : Visual Studio 2022 C/C++

운영체제 : Windows 기반

버전 관리 툴 : GitHub

개발 일정

김태순	일	월	화	수	목	금	토
11/1	-	-	-	-	-	-	클라이언트 구조 변경
11/2~11/8	DoRecv()	ProcessPacket()		HandleAssignID()		SendUploadMapPacket()	
11/9~11/15		SendStartSessionRequestPacket()			SC_GameStatePacket() PlayerState		SC_GameStatePacket() BossState
11/16~11/22	HandleMapInfo()		SendPlayerUpdatePacket()			여러 윈도우 창 생성 및 갱신 구현	
11/23~11/29			HandleGameState()				
11/30~12/6	최종 테스트						

김지호	일	월	화	수	목	금	토
11/1	-	-	-	-	-	-	
11/2~11/8	AcceptClient()		DoRecv(int client_id)		ProcessPacket()		HandleMapUpload()
11/9~11/15	SendMapUploadResponsePacket()		HandleStartSessionRequest()		SendMapInfoPacket()		
11/16~11/22	CheckAllCollisions()				UpdateAllPositions()		
11/23~11/29		SendStateUpdatePacket()			SendDisconnectPacket()		IsGameEnd()
11/30~12/6	최종 테스트						

김한준	일	월	화	수	목	금	토
11/1	-	-	-	-	-	-	SC_AssignIDPacket()
11/2~11/8		CS_UploadMapPacket()			SC_MapUploadResponsePacket()		CS_StartSessionRequestPacket()
11/9~11/15	SC_MapInfoPacket()		SC_GameStatePacket() EnemyState			SC_EventPacket()	SendEventPacket()
11/16~11/22	HandleEvent()		CS_EndSessionRequestPacket()		SendEndSessionRequestPacket()		HandleEndSessionRequest()
11/23~11/29		SC_DisconnectPacket()			HandleDisconnect()		
11/30~12/6	최종 테스트						