**네트워크 게임 프로그래밍 Term Project 추진 계획서**

5조

2014180021 박진우

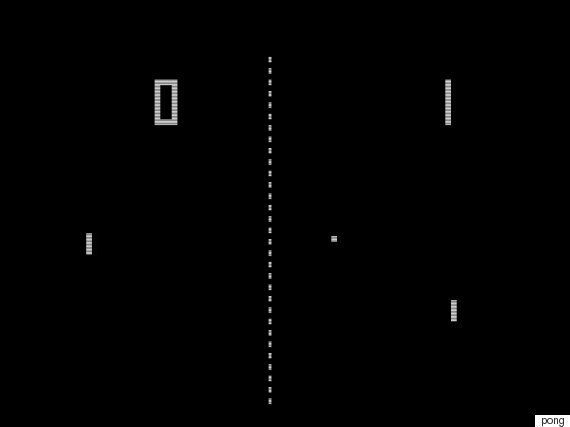
2014182038 장규현

2016180007 김명규

**목차**

1. 게임 소개
2. 어플리케이션 기획
3. High Level 디자인
   1. 서버 디자인
   2. 클라이언트 디자인
   3. 스레드 동기화
4. Low Level 디자인
5. 팀원 역할 분담
6. 개발 일정
7. 개발 환경

**1. 게임 소개**

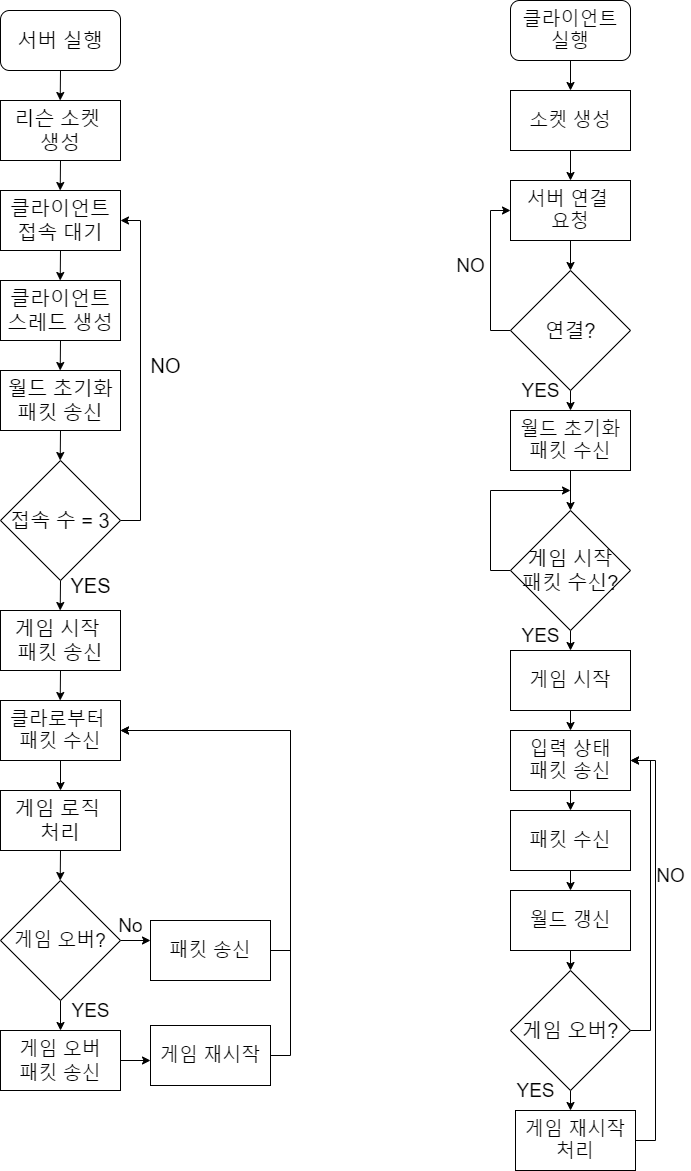
* 게임 제목 : 퐁
* 게임 장르 : 아케이드 게임
* 게임 설명 : 퐁은 탁구를 모방한 게임이다. 플레이어는 각자 막대를 조종하여 공을 쳐낸다. 공이 상대편 막대를 넘어 반대편 벽에 도달하게 되면 승리한다.
* 게임 조작법 : W,S키를 이용해 막대를 움직인다.
* 게임 예시

(출처 : https://www.huffingtonpost.kr/2016/06/01/story\_n\_10250946.html)

**2. 어플리케이션 기획**

* TCP 사용
* 게임 실행 시 서버의 IP 주소를 입력 받는다.
* 서버는 3개의 클라이언트를 수용한다.
* 서버는 클라이언트가 접속할 때마다 접속한 클라이언트에게 게임 월드 초기화 패킷을 송신한다.
* 서버에 모든 클라이언트가 접속을 완료하면 게임 시작 패킷을 전송한다.
* 클라이언트는 사용자의 입력 상태를 서버에 전송한다.
* 서버는 수신 패킷 데이터로 플레이어의 패들 위치 갱신, 공 위치 갱신, 충돌 처리를 수행한다.
* 서버에서 갱신된 월드 데이터를 클라이언트에 송신, 클라이언트는 수신한 데이터로 오브젝트의 위치를 갱신하고 렌더링을 수행한다.
* 서버는 임계 영역을 사용해 클라이언트 스레드와 로직 스레드 간의 동기화를 관리한다.

**3. High Level 디자인**

****

**- 스레드 동기화**

* 서버에선 임계 영역을 이용해 스레드 간 동기화를 관리한다.
* 클라이언트가 서버에 접속할 때 서버는 클라이언트의 번호를 지정한다. 클라이언트가 서버에 패킷을 송신할 때 자신의 번호를 패킷에 담는다.
* 서버에선 최대 4개의 스레드를 생성하고 실행한다. 게임 로직을 처리하는 주 스레드, 연결된 클라이언트로부터 패킷을 받아 주 스레드에 전달하는 클라이언트 스레드 3개.
* 클라이언트 스레드와 주 스레드는 패킷 데이터를 공유하기 때문에, Data race가 일어날 수 있다. 따라서 임계 영역을 이용해 공유 자원 접근을 관리한다.
* 서버는 클라이언트 스레드로부터 패킷을 수신할 때마다 주 스레드에서 게임 로직 처리를 수행하고 갱신된 월드 데이터를 모든 클라이언트에 보낸다.

**4. Low Level 디자인**

**공통 프레임워크**

**class** **Game** *// Client, Server 프레임워크의 베이스 클래스*

{

**friend** **class** **Entity**;

**public**:

**virtual** ~Game();

**virtual** bool Init() = 0;

**virtual** void Shutdown() = 0;

**virtual** void Run() = 0;

Entity\* CreateEntity(); *// Entity 생성*

Entity\* CreatePaddle(); *// Paddle 생성*

Entity\* CreateBall(); *// Ball 생성*

**protected**:

**const** int WINDOW\_WIDTH; *// 클라이언트 윈도우 너비*

**const** int WINDOW\_HEIGHT; *// 클라이언트 윈도우 높이*

**const** float PADDLE\_WIDTH; *// 패들 너비*

**const** float PADDLE\_HEIGHT; *// 패들 너비*

**const** float PADDLE\_SPEED; *// 패들 속도*

**const** float BALL\_WIDTH; *// 공 너비 및 높이*

**const** float BALL\_SPEED; *// 공 이동속도*

std::unordered\_map<uint8\_t, Entity\*> mEntities; *// key는 Entity ID*

}

**class** **Entity**

{

**public**:

Entity(int id, Game\* game);

**template**<**typename** **T**, **typename**... Args>

T& AddComponent(Args&&... args); *// Component 추가*

**template**<**typename** **T**>

void AddTag(); *// Tag 추가*

**template**<**typename** **T**>

T& GetComponent(); *// 요구한 Component 반환*

**private**:

int mID;

Game\* mGame;

}

**struct** **TransformComponent**

{

Vector2 Position;

}

**struct** **RectComponent**

{

float Width;

float Height;

}

**struct** **MovementComponent**

{

float Speed;

Vector2 Direction;

}

**struct** **IdComponent**

{

uint8\_t ID;

}

**class** **Systems**

{

**public**:

**static** void DrawRect(SDL\_Renderer\* renderer, float w, float h, **const** Vector2& position); *// 전달된 매개변수의 값으로 화면에 사각형을 그림*

**static** void UpdateDirection(**const** uint8\_t\* keystates, float& outDirection); *// 키 입력을 조사해서 -1.0f ~ 1.0f의 값을 outDirection에 넣음*

**static** void UpdatePosition(float speed, **const** Vector2& direction, Vector2& outPosition, float deltaTime); *// 속도를 입력 받아 위치를 갱신*

**static** bool Intersects(**const** SDL\_Rect& a, **const** SDL\_Rect& b); *// 두 사각형이 충돌하는지 체크*

}

**서버 프레임워크**

**class** **Server** : **public** Game

{

**public**:

**virtual** bool Init () **override**; *// 초기화 작업 및 클라이언트 접속 대기*

**virtual** void Shutdown() **override**; *// 리소스 정리 수행*

**virtual** void Run() **override**; *// 로직 처리가 실행되는 곳. mPacketsFromClient에서 패킷을 읽어 모든 Entity 위치 갱신, 각 Client에게 패킷 송신*

**private**:

void ClientThreadFunc(SOCKET s, int clientNum); *// 각 클라이언트로부터 패킷을 수신해 mPacketsFromClient에 삽입*

void CreateGameWorld(); *// 게임 월드 생성*

void UpdatePaddlesPosition(); *// 모든 패들 위치 갱신*

void UpdateBallsPosition(); *// 모든 공 위치 갱신*

void CheckBallAndPaddle(); *// 공-패들 충돌 체크*

void CheckBallAndWall(); *// 공-벽 충돌 체크*

void ResetWorld (); *// 공이 벽에 닿았을 때 게임 월드 초기화*

void SendPacketToClient(PacketType pType); *// Client에 ServerToClientPacket 송신*

void RecvPacketFromClient(const ClientToServer& packet); *// Client로부터 ClientToServerPacket 수신*

**private**:

**static** **const** int MAXMINUM\_PLAYER\_NUM = 3; *// 최대 접속 가능한 플레이어의 수*

std::**thread** mClientThread[MAXMIMUM\_PLAYER\_NUM];

std::vector<Packet> mPacketsFromClient; *// 각 ClientThread에서 보내오는 패킷을 담아둠.*

vector<SOCKET> mClientSockets;

bool mIsAllPlayerConnected; *// 모든 클라이언트가 접속했는지 확인해주는 변수*

CRITICAL\_SECTION mCS; *// 임계 영역*

};

**클라이언트 프레임워크**

**class** **Client** : **public** Game

{

**public**:

**virtual** bool Init() **override**; *// SDL 윈도우 생성, SDL 렌더러 생성, 윈속 초기화 작업 수행*

**virtual** void Shutdown() **override**; *// 리소스 해제 수행*

**virtual** void Run() **override**; *// 현재 씬의 Run() 호출*

void ChangeScene(Scene\* scene); *// 씬 전환*

void SetClientNum(int num);

**private**:

bool NetworkInit(); *// 윈속 초기화*

void RecvPacketFromServer(const ServerToClient& packet); *// 서버의 패킷을 수신함*

void SendPacketToServer(int clientNum, float yDirection); *// 클라이언트의 입력 상태를 담은 패킷을 서버에 송신*

**private**:

SOCKET mClientSocket; *// 서버와 통신을 위한 소켓*

Scene\* mActiveScene; *// 로그인 씬 / 인게임 씬*

int mClientNum; *// 클라이언트 번호*

}

**class** **Scene**

{

**public**:

**virtual** ~Scene();

**virtual** void Enter() = 0;

**virtual** void Exit() = 0;

**virtual** void Run() = 0;

**protected**:

Game\* mOwner;

}

**class** **LoginScene** : **public** Scene *// 로그인 씬*

{

**public**:

**virtual** void Enter() **override**;

**virtual** void Exit() **override**;

**virtual** void Run() **override**;

**private**:

void GetIP(); *// 대화 박스를 이용해 서버의 IP 주소를 입력 받는다. 서버에 connect 요청.*

void Render();

}

**class** **GameScene**: **public** Scene *// 인게임 씬*

{

**public**:

**virtual** void Enter() **override**; // *RecvPacketFromServer() 호출해서 게임 월드 초기화. 이 때 클라이언트 번호가 지정된다.*

**virtual** void Exit() **override**;

**virtual** void Run() **override**;

**private**:

void ProcessInput(); *// 유저의 입력 상태를 패킷에 담아 서버에 송신. 송신에 Client::SendPacketToServer() 호출. 입력이 없다면 0을 보냄.*

void Update(); *// 서버가 보낸 패킷을 수신해 게임 월드 갱신. 수신에 Client::RecvPacketFromServer() 호출.*

void Render(); *// 렌더링 수행*

}

**패킷 설계**

**enum** **class** **PacketType**

{

Hello, *// 게임 월드 초기화 패킷*

GameStart, *// 게임 시작 신호 패킷*

Update, *// 게임 월드 업데이트 패킷*

GameOver *// 게임 오버 신호 패킷*

};

**struct** **ServerToClient**

{

PacketType PType;

int ClientNum; *// Hello 패킷일 때만 유효한 값을 대입. 클라이언트 번호.*

int WhoLose; *// -1 : 왼쪽 승리! 1 : 오른쪽 승리!*

*// Left paddle*

uint8\_t LeftPaddleID;

Vector2 LeftPaddlePosition;

*// Right paddle*

uint8\_t RightPaddleID;

Vector2 RightPaddlePosition;

*// Left2 paddle*

uint8\_t L2PaddleID;

Vector2 L2PaddlePosition;

*// Right2 paddle*

uint8\_t R2PaddleID;

Vector2 R2PaddlePosition;

*// Ball One*

uint8\_t BallOneID;

Vector2 BallOnePosition;

*// Ball Two*

uint8\_t BallTwoID;

Vector2 BallTwoPosition;

};

**struct** **ClientToServer**

{

float YDirection; *// 패들의 y방향 이동값(-1.0f ~ 1.0f)*

int ClientNum; *// 클라이언트 번호*

};

**그 외**

**class** **SocketAddress**; *// 소켓 주소 구조체 래퍼*

**class** **TCPSocket**; *// TCP 소켓 래퍼*

**class** **SocketUtil**; *// 소켓 생성, 오류 출력 등 유틸 함수들*

**class Log**; *// 로그 출력용 클래스*

**5. 팀원 역할 분담**

|  |  |  |
| --- | --- | --- |
| 김명규 | 장규현 | 박진우 |
| 역할분담 | | |
| 공통 프레임워크 제작 | 로그인 씬 | 게임 씬 |
| 클라이언트 프레임워크 | 유틸 클래스 | 서버 프레임워크 |
| 클라이언트 월드 초기화 패킷 수신 및 처리 | 클라이언트 입력 상태 송신 | 클라이언트로부터 패킷을 받아 공유 자원 관리 |
| 클라이언트 월드 업데이트 패킷 수신 및 처리 | 클라이언트 게임 시작, 게임 오버 패킷 수신 및 처리 | 충돌 체크 구현 |
| 게임 재시작 구현 | 서버 패킷 송신 구현 | 서버 게임 월드 갱신 |
| 충돌 체크 구현 |  |  |

**6. 개발 일정**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **김명규** | **장규현** | **박진우** |
| **11월 10일** | 추진 계획서 수정, 프로젝트 생성 | | |
| **11월 11일** | 씬 클래스 구현 (Enter, Exit, Run) | 로그인 씬 구현 (Enter, Exit, Run) | 서버 프레임워크 (Init, Shutdown, Run) |
| **11월 12일** | 클라이언트 패킷 구현 (PacketType::Hello) | 로그인 씬 구현 (GetIP, Render) | 서버 (ClientThreadFunc) |
| **11월 13일** | 클라이언트 패킷 구현 (PacketType::Hello) | 로그인 씬 구현 (GetIP, Render) | 서버 (ClientThreadFunc) |
| **11월 14일** |  |  | 서버 프레임워크 (HelloPacket 송신) |
| **11월 15일** | 로그인 테스트 | | |
| **11월 16일** |
| **11월 17일** |  | 클라이언트 패킷 구현 (PacketType::GameStart) | 게임 씬 구현 (Enter, Exit, Run, Render) |
| **11월 18일** | 게임 씬 구현 (ProcessInput, Update) |  | 게임 씬 구현 (Enter, Exit, Run, Render) |
| **11월 19일** | 게임 씬 구현 (ProcessInput, Update) | 클라이언트 패킷 구현 (ClientToServer) |  |
| **11월 20일** |  | 클라이언트 패킷 구현 (ClientToServer) | 스레드 동기화 |
| **11월 21일** |  |  | 스레드 동기화 |
| **11월 22일** | 서버 게임 월드 갱신 (공) | 서버 패킷 송신 (PacketType::Update) | 서버 게임 월드 갱신 (패들) |
| **11월 23일** |  | 서버 패킷 송신 (PacketType::Update) | 서버 게임 월드 갱신 (패들) |
| **11월 24일** | 클라이언트 패킷 구현 (PacketType::Update) |  |  |
| **11월 25일** | 클라이언트 패킷 구현 (PacketType::Update) |  |  |
| **11월 26일** | 패들, 공 움직임 테스트 | | |
| **11월 27일** |  |  | 서버 충돌 체크 (공-패들) |
| **11월 28일** | 서버 게임 월드 초기화 |  | 서버 충돌 체크 (공-벽) |
| **11월 29일** | 충돌 테스트 | | |
| **11월 30일** |  | 클라이언트 패킷 구현 (PacketType::GameOver) | 서버 패킷 송신 (PacketType::GameOver) |
| **12월 1일** | 게임 씬 구현(재시작) | 클라이언트 패킷 구현 (PacketType::GameOver) | 서버 패킷 송신 (PacketType::GameOver) |
| **12월 2일** | 게임 씬 구현(재시작) |  |  |
| **12월 3일** | 추가 구현 및 버그 수정 | | |
| **12월 4일** |
| **12월 5일** |
| **12월 6일** |

**7. 개발 환경 및 사용 라이브러리**

* Visual Studio 2019
* GitHub([derisan/NGP\_Pong (github.com)](https://github.com/derisan/NGP_Pong))
* SDL2([Simple DirectMedia Layer - Homepage (libsdl.org)](https://www.libsdl.org/index.php))
* spdlog([gabime/spdlog: Fast C++ logging library. (github.com)](https://github.com/gabime/spdlog))
* entt([skypjack/entt: Gaming meets modern C++ - a fast and reliable entity component system (ECS) and much more (github.com)](https://github.com/skypjack/entt))