**네트워크 게임 프로그래밍 Term Project 추진 계획서**

5조

2014180021 박진우

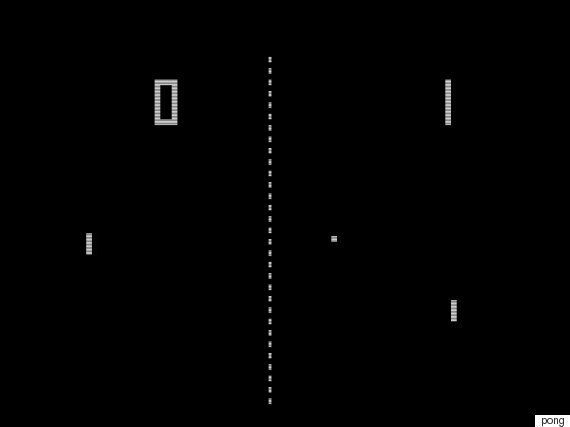
2014182038 장규현

2016180007 김명규

**목차**

1. 게임 소개
2. 어플리케이션 기획
3. High Level 디자인
   1. 서버 디자인
   2. 클라이언트 디자인
   3. 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. 3명의 플레이어는 각자 막대를 조종하여 두 개의 공을 쳐낸다. 공이 상대편 막대를 넘어 반대편에 도달하게 되면 승리한다.
* 게임 조작법 : 화살표 위아래 키를 이용해 막대를 움직인다. ESC를 입력하면 게임이 종료된다.
* 게임 예시

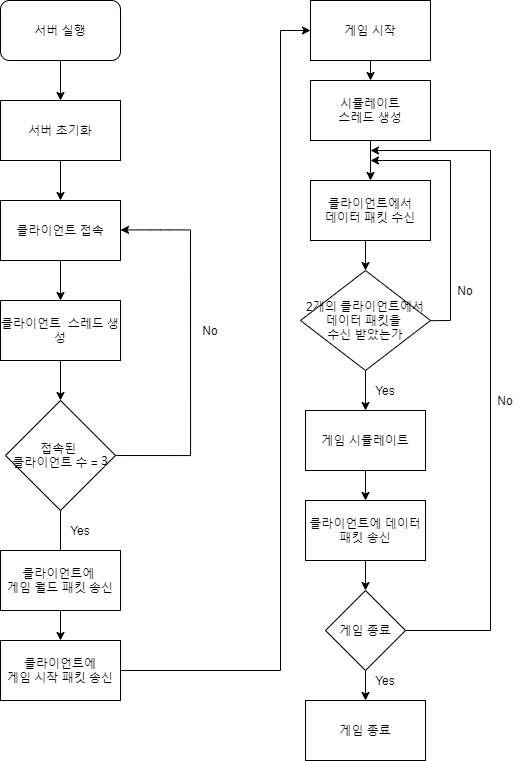
(출처 : https://www.huffingtonpost.kr/2016/06/01/story\_n\_10250946.html)

**2. 어플리케이션 기획**

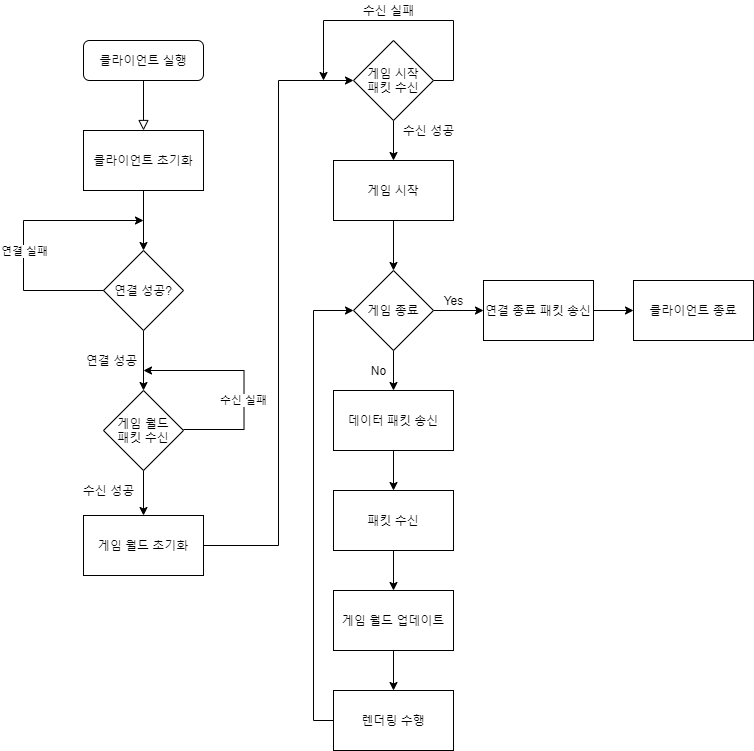
* TCP 네트워크 방식 사용
* 게임 실행 시 서버의 IP 주소와 포트 번호를 입력 받는다
* 서버는 3개의 클라이언트를 수용한다.
* 서버에 3개의 클라이언트가 접속이 완료되면 게임 월드 초기화 패킷을 전송한다
* 클라이언트의 입력 상태를 서버에 전송한다
* 서버는 클라이언트에서 송신한 구조체를 수신하고 플레이어의 패들 위치 갱신, 공 위치 갱신, 충돌 처리를 수행한다.
* 서버에서 갱신된 게임 데이터를 클라이언트에 송신, 클라이언트는 수신된 데이터를 통해 오브젝트의 위치 갱신하고 렌더링을 수행한다.
* 서버는 mutex를 사용해 클라이언트 데이터를 처리하는 스레드와 메인 스레드 간의 동기화를 관리한다.

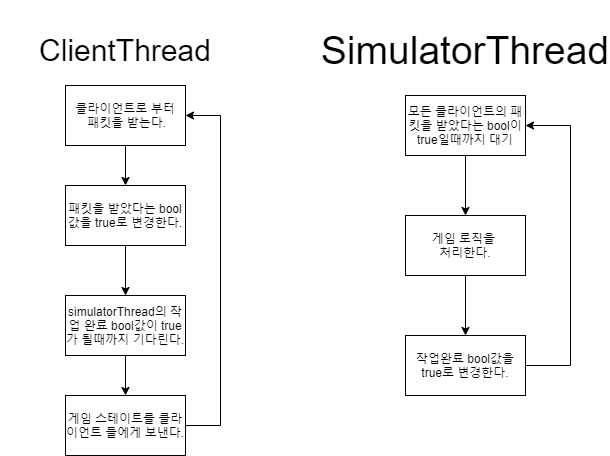
**3. High Level 디자인**

- 서버 디자인



- 클라이언트 디자인

****

- 스레드 동기화

- 동기화 방식

* 클라이언트 60프레임을 기준으로 클라이언트와 서버는 1초에 30번 데이터를 송수신한다.
* SimulatorThread에서 공과플레이어의 위치 속도 등 게임의 흐름을 처리한다. ClientThread는 패킷을 수신하고 SimulatorThread에의해 갱신된 GameState를 송신한다.
* 동기화 부분에서는 ClientThread에서 패킷을 수신하는데, 이때 bool값으로 2개의 클라이언트 수신을 체크한다. 이 bool값이 true가 되면 SimulatorThread에서 클라이언트에게 받은 데이터를 참고해 로직을 진행한다. 진행이 완료되면 로직처리가 완료됐다는 bool값을 true로 바꾼다. ClientThread는 SimulatorThread의 로직처리 완료 여부를 감시하고 있다가 true가 된순간 각 클라이언트에 갱신된 GameState를 전송하고 다시 bool값을 초기화한다.

**4. Low Level 디자인**

**공통 프레임워크**

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

{

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

**public**:

**virtual** ~Game();

**virtual** bool Init();

**virtual** void Shutdown();

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

Entity\* CreateEntity(); *// Entity 생성 후 포인터를 반환*

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*

std::unordered\_map<uint8\_t, Entity\*> mEntities; *// pair<Entity\_Id, Entity\*>*

}

**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;

}

**서버 프레임워크**

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

{

**public**:

**virtual** bool Init() **override**; *// 초기화 작업 및 ListenThread 구동*

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

**virtual** void Run() **override**; *// Client로부터 전달된 패킷으로 Entity 위치 갱신, Client에게 패킷 송신*

**private**:

void ListenThreadFunc(); *// 클라이언트 연결을 받아주기 위해 accept()를 호출*

void ClientThreadFunc(SOCKET s, int clientNum); *// 각각의 클라이언트로부터 패킷을 받아 메인 스레드에게 전달*

void CreateGameWorld(); *// 게임 월드 생성*

**private**:

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

std::**thread** mListenThread;

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

std::array<Packet, MAXIMUM\_PLAYER\_NUM> mPacketsFromClient; *// 각 ClientThread에서 보내오는 패킷을 담아둠*

std::array<bool, MAMIMUM\_PLAYER\_NUM> mIsRecvPacket; *// 모든 클라이언트가 패킷을 보냈는지 확인하기 위해 필요*

vector<SOCKET> mClientSockets;

std::mutex m; *// 동기화를 위해 필요*

}

**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**;

**private**:

bool NetworkInit(); *// 윈속 초기화 및 서버에 connect 요청. 서버와 연결되었다면 RecvHelloPacket() 호출.*

void ProcessInput(); *// 유저의 입력 상태를 패킷에 담아 서버에 송신*

void Update(); *// 서버가 보낸 패킷에 담긴 데이터로 게임 월드 갱신*

void Render(); *// 렌더링 수행*

void RecvHelloPacket(); *// 서버의 패킷을 받아 초기 게임 월드 구성*

**private**:

SOCKET mClientSocket; *// 서버와 통신을 위한 소켓*

}

**패킷 설계**

**enum** **class** **BehaviorType**

{

Create, *// Entity 생성 명령*

Update, *// Entity position 갱신 명령*

Delete *// Entity 삭제 명령*

}

**struct** **ServerToClient**

{

*// Player1 paddle*

uint8\_t LeftPaddleID; *// Entitiy ID*

BehaviorType LeftPaddleBType; *// 생성, 갱신, 삭제 중 택 1*

Vector2 LeftPaddlePosition; *// 갱신 시 사용될 Entity의 위치*

*// Player2 paddle*

uint8\_t RightPaddleID;

BehaviorType RightPaddleBType;

Vector2 RightPaddlePosition;

*// Player3 paddle, Ball1, Ball2*

};

**struct** **ClientToServer**

{

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

};

**그 외**

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

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

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

**5. 팀원 역할 분담**

김명규 : 리소스 수집, 클라이언트 프레임워크 구현, 소켓 래퍼 클래스 구현

장규현 : 패킷 설계, 서버-클라이언트 송수신 구현

박진우 : 서버 프레임워크 구현, 멀티스레드 설계 및 구현

**6. 개발 일정**

|  |  |  |  |
| --- | --- | --- | --- |
| 날짜 | 박진우 | 장규현 | 김명규 |
| 11월 10일 |  |  |  |
| 11월 11일 |  |  |  |
| 11월 12일 |  |  |  |
| 11월 13일 |  |  |  |
| 11월 14일 |  |  |  |
| 11월 15일 |  |  |  |
| 11월 16일 |  |  |  |
| 11월 17일 |  |  |  |
| 11월 18일 |  |  |  |
| 11월 19일 |  |  |  |
| 11월 20일 |  |  |  |
| 11월 21일 |  |  |  |
| 11월 22일 |  |  |  |
| 11월 23일 |  |  |  |
| 11월 24일 |  |  |  |
| 11월 25일 |  |  |  |
| 11월 26일 |  |  |  |
| 11월 27일 |  |  |  |
| 11월 28일 |  |  |  |
| 11월 29일 |  |  |  |

**7. 개발 환경**

* Visual Studio 2019
* GitHub (프로젝트 관리)