**네트워크 게임 프로그래밍 Term Project 추진 계획서**

5조

2014180021 박진우

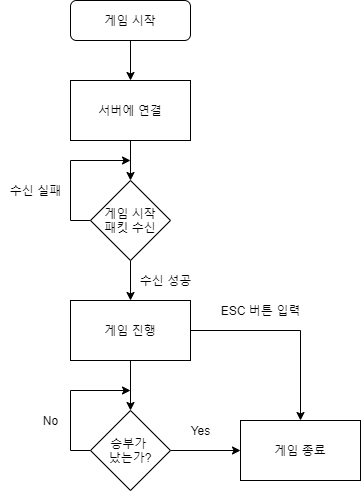
2014182038 장규현

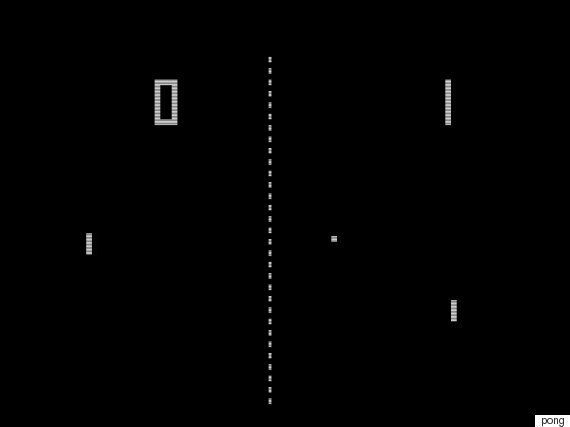
2016180007 김명규

**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. 2명의 플레이어는 각자 막대를 조종하여 공을 쳐낸다. 공은 시간이 지날수록 점점 빨라진다. 공이 상대편 막대를 넘어 반대편에 도달하게 되면 승리한다.
* 게임 조작법 : 화살표 위아래 키를 이용해 막대를 움직인다. ESC를 입력하면 게임이 종료된다.
* 게임 흐름

* 게임 예시

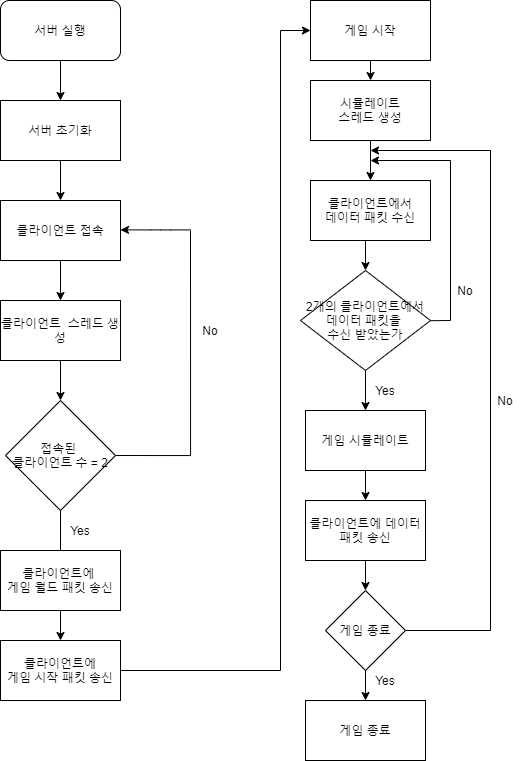
(출처 : https://www.huffingtonpost.kr/2016/06/01/story\_n\_10250946.html)

**2. 어플리케이션 기획**

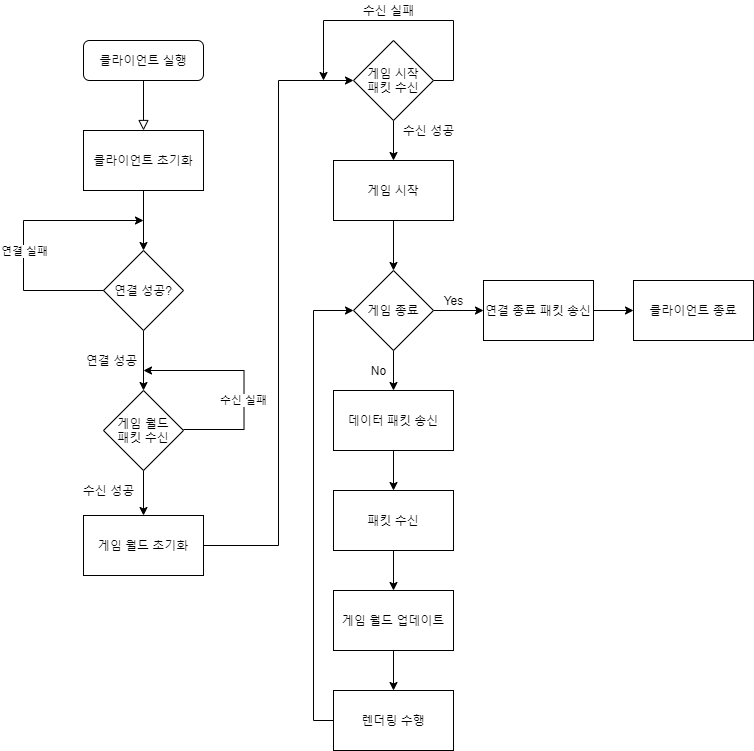
* TCP 네트워크 방식 사용
* 게임 실행시 서버의 IP 주소와 포트 번호를 입력 받는다
* 서버는 2개의 클라이언트를 수용한다
* 서버에 2개의 클라이언트가 접속이 완료되면 초기 게임 스테이트 정보를 전송한다
* 클라이언트의 입력 상태 및 플레어어의 정보 구조체를 서버에 전송한다
* 서버는 클라이언트에서 송신한 구조체를 수신하고 플레이어 정보 갱신 및 오브젝트간의 충돌처리를 연산한다
* 서버에서 갱신된 게임 데이터를 클라이언트에 송신, 클라이언트는 수신된 데이터를 통해 오브젝트를 갱신하고 렌더링을 수행한다
* 서버는 클라이언트 데이터를 처리하는 스레드와 게임 시뮬레이터 스레드간의 동기화를 관리한다

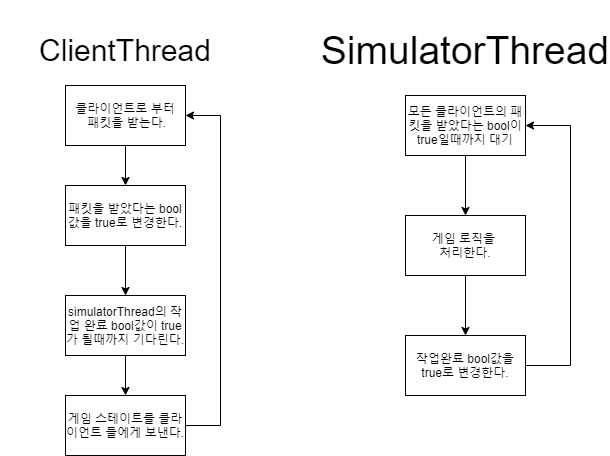
**High Level 디자인**

- 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**

**서버 프레임워크**

**class** **Server**

{

**public**:

void ClientThreadFunc(); *// 클라이언트 송수신 스레드 시작 함수*

void SimulatorThreadFunc(); *// 시뮬레이션 스레드 시작 함수*

void Run();

**private**:

TCPSocket listenSocket; *// 접속을 받을 리슨 소켓*

TCPSocket clientSocket[2]; *// 두 플레이어와 통신할 소켓*

}

**Client To Server 패킷 구조**

**struct** **ClientPacket**

{

int PacketType; *// 패킷 종류(연결 요청, 게임 종료, 리플리케이션)*

int EntityType; *// 엔티티 타입 ID*

int EntityID; *// 엔티티 UUID*

Vector2 Position; *// 엔티티 위치*

Vector2 Velocity; *// 엔티티 속도*

}

**Server To Client 패킷 구조**

**struct** **ServerPacket**

{

int PacketType; *// 패킷 종류(연결 요청, 게임 종료, 리플리케이션)*

int EntityType; *// 엔티티 타입 ID*

int EntityID; *// 엔티티 UUID*

*// 리플리케이션 데이터*

}

**클라이언트 프레임워크**

**class** **Game**

{

**public**:

void ProcessInput();*// 사용자 입력 갱신하고 패킷 송신*

void Update(); *// 패킷 수신 받고 게임 월드 갱신*

void Render(); *// 렌더링 수행*

**private**:

vector<Entity> entities; *// 엔티티 모음*

}

**네트워크 헬퍼 클래스**

**class** **NetworkManager**

{

**public**:

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

void Write(T inData); *// 패킷을 만들기 위한 직렬화 수행.*

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

void Read(T& outData); *// 패킷을 읽기 위한 직렬화 수행.*

int Send(**const** void\* inData); *// 패킷 송신*

int Recv(void\* outData); *// 패킷 수신*

}

**class** **TCPSocket**

{

**public**:

~TCPSocket(); *// closesocket() 호출*

int Bind(**const** SocketAddress& addr); *// 지역 ip, 포트 번호 결정*

int Listen(int backLog); *// TCP 소켓 상태를 Listen 상태로 변경*

TCPSocket\* Accept(SocketAddress& outAddr); *// 접속을 받아 새로운 소켓 리턴*

int Send(**const** void\* inData, int len); *// len 바이트 수만큼 데이터 송신*

int Recv(void\* outData, int len); *// len 바이트 수만큼 데이터 수신*

int Connect(**const** SocketAddress& addr); *// 연결 요청*

**private**:

SOCKET sock; *// 소켓 멤버 변수*

}

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

{

**public**:

SocketAddress(); *// 기본 IP 주소, 기본 포트 번호로 초기화*

SocketAddress(**const** sockaddr& addr);

SocketAddress(**const** string& ip, ushort port);

size\_t GetSize(); *// sizeof(sockaddr)*

string ToString(); *// "IP:포트" 형태의 문자열 리턴*

**private**:

sockaddr saddr; *// 소켓 주소 구조체*

}

5. 팀원 역할 분담

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

장규현 : 패킷 설계, 서버-클라이언트 송수신 구현

박진우 : 서버 프레임워크 구현, 멀티스레드 설계 및 구현

6. 개발 일정

|  |  |  |  |
| --- | --- | --- | --- |
| 날짜 | 박진우 | 장규현 | 김명규 |
| 11월 3일 | SDL2 학습 | SDL2 학습 | 리소스 수집 |
| 11월 4일 | 개발 환경 준비 | 개발 환경 준비 | 개발 환경 준비 |
| 11월 5일 |  |  | SDL 윈도우 생성 |
| 11월 6일 |  |  |  |
| 11월 7일 |  |  |  |
| 11월 8일 | 소켓 Address 클래스 |  | Game 클래스 구현 |
| 11월 9일 | TCP socket 클래스 |  | Entity, Component  구현 |
| 11월 10일 |  | 클라이언트 패킷 설계 | 클라이언트 프레임워크 완성 |
| 11월 11일 |  |  | 클라이언트 버그 수정 |
| 11월 12일 | 클라이언트 코드 분석 | 클라이언트 코드 분석 | 클라이언트 코드 분석 |
| 11월 13일 |  | 서버 패킷 설계 |  |
| 11월 14일 | 서버 클래스 구현 |  |  |
| 11월 15일 |  | 서버-클라이언트  송수신 구현 | 직렬화 구현(쓰기) |
| 11월 16일 | 멀티 스레드 설계 | - | 직렬화 구현(쓰기) |
| 11월 17일 | - | 네트워크 매니저 구현 | 서버와 송수신 테스트 |
| 11월 18일 |  | - | 직렬화 구현(읽기) |
| 11월 19일 | 클라이언트 스레드  함수 |  | 직렬화 구현(읽기) |
| 11월 20일 |  |  |  |
| 11월 21일 |  |  |  |
| 11월 22일 | 시뮬레이터 스레드  구현 |  |  |
| 11월 23일 | - |  |  |
| 11월 24일 |  |  |  |
| 11월 25일 | 버그 수정 | 버그 수정 | 버그 수정 |
| 11월 26일 | 버그 수정 | 버그 수정 | 버그 수정 |
| 11월 27일 | 버그 수정 | 버그 수정 | 버그 수정 |
| 11월 28일 | 버그 수정 | 버그 수정 | 버그 수정 |
| 11월 29일 |  |  |  |

7.개발 환경

* Visual Studio 2019
* GitHub (프로젝트 관리)