**네트워크 게임 프로그래밍 Term Project 추진 계획서**

5조

2014180021 박진우

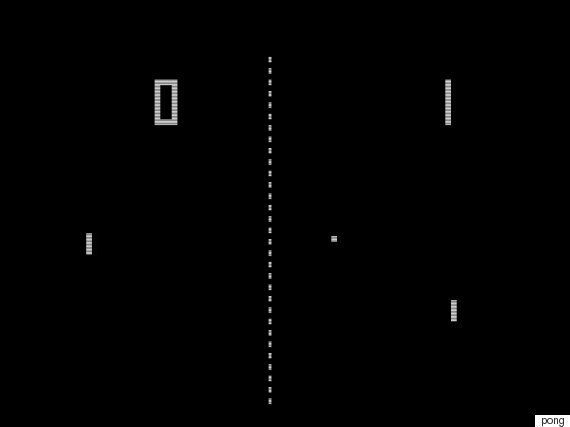
2014182038 장규현

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 반대편 벽에 도달하게 되면 승리한다.
* 게임 조작법 : W,S키를 이용해 막대를 움직인다. ESC를 입력하면 게임이 종료된다.
* 게임 예시

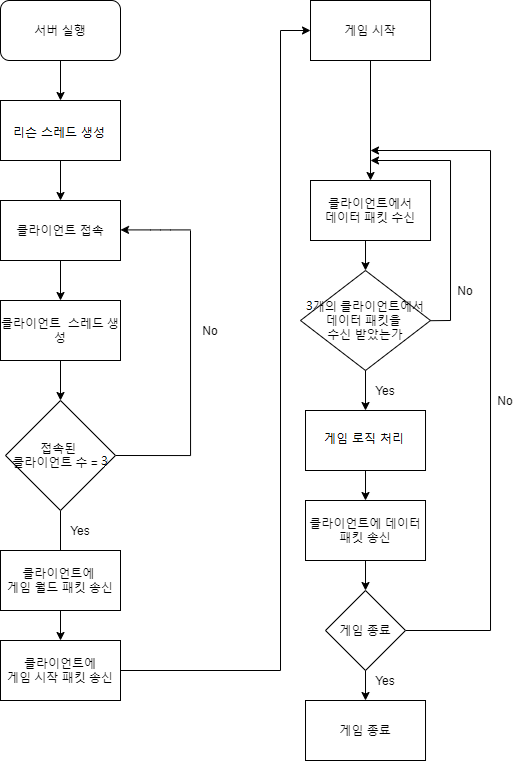
(출처 : https://www.huffingtonpost.kr/2016/06/01/story\_n\_10250946.html)

**2. 어플리케이션 기획**

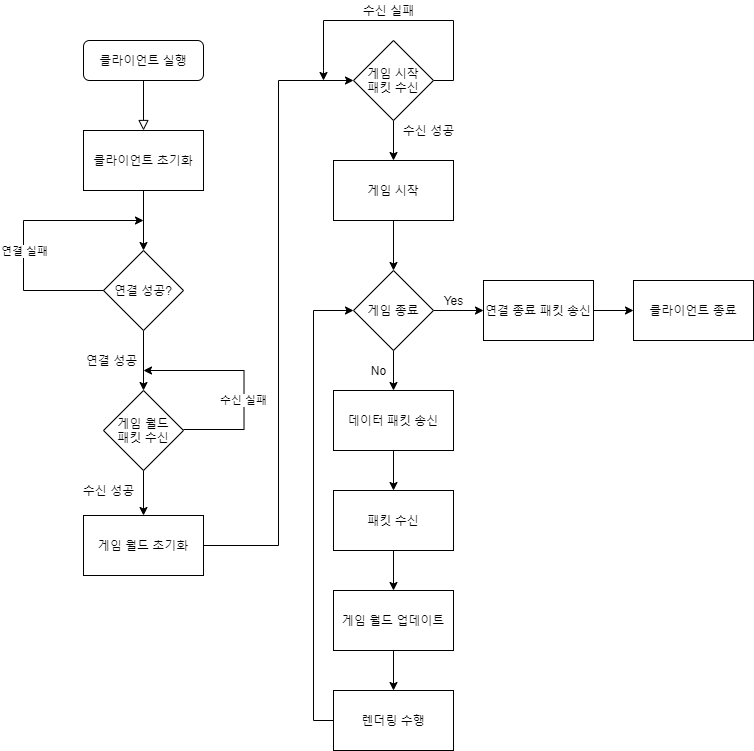
* TCP 사용
* 게임 실행 시 서버의 IP 주소와 포트 번호를 입력 받는다.
* 서버는 3개의 클라이언트를 수용한다.
* 서버에 3개의 클라이언트가 접속이 완료되면 게임 월드 초기화 패킷을 전송한다.
* 클라이언트의 입력 상태를 서버에 전송한다.
* 서버는 클라이언트에서 송신한 패킷 데이터로 플레이어의 패들 위치 갱신, 공 위치 갱신, 충돌 처리를 수행한다.
* 서버에서 갱신된 월드 데이터를 클라이언트에 송신, 클라이언트는 수신한 데이터로 오브젝트의 위치를 갱신하고 렌더링을 수행한다.
* 서버는 mutex를 사용해 클라이언트 데이터를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

**3. High Level 디자인**

**- 서버 디자인**



**- 클라이언트 디자인**

****

**- 스레드 동기화**

* 서버에선std::mutex와 std::lock\_guard를 이용해 스레드 간 동기화를 관리한다.
* 서버에선 총 5개의 스레드를 생성하고 실행한다. 게임 로직을 처리하는 주 스레드, 연결된 클라이언트로부터 패킷을 받아 주 스레드에 전달하는 클라이언트 스레드 3개, 그리고 소켓 함수 accept()를 호출한 채 클라이언트 연결 요청을 받고 클라이언트 스레드를 생성 및 실행해주는 리슨 스레드 1개.
* 5개의 스레드 중 동기화가 필요한 스레드는 주 스레드와 클라이언트 스레드들이다.
* 클라이언트 스레드와 주 스레드는 패킷 데이터를 공유하기 때문에, 클라이언트 스레드가 수신한 패킷을 전달할 때 Data race가 일어날 수 있다. 즉, mPacketsFromClientThread에 패킷 데이터를 쓸 때 동기화가 필요하다. 클라이언트 스레드는 패킷 데이터를 쓸 때 락을 건다.
* 주 스레드는 모든 클라이언트로부터 패킷을 수신한 걸 확인한 후에 게임 로직을 처리한다. 따라서 모든 패킷을 수신했는지 확인하기 위해 매번mIsRecvPacket을 읽게 된다. 클라이언트 스레드는 패킷을 수신했다면 mIsRecvPacket의 값을 true로 만든다. 이 과정 중 역시나 Data race가 일어날 수 있기 때문에 주 스레드에서 mIsRecvPacket 값을 읽을 때와 클라이언트 스레드에서 값을 쓸 때 락을 건다.

**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*

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;

}

**서버 프레임워크**

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

{

**public**:

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

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

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

**private**:

void ListenThreadFunc(); *// 클라이언트 연결을 받아주기 위해 accept()를 호출, ClientThread 생성 후 실행*

void ClientThreadFunc(SOCKET s, int clientNum); *// 각 클라이언트로부터 패킷을 수신해 mPacketsFromClient, mIsRecvPackcet에 적절한 값을 씀*

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 (); *// Client에 ServerToClientPacket 송신*

**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; *// 스레드 동기화*

}

**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 게임 월드 생성을 위해 RecvPacketFromServer() 호출.*

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

void Update(); *// 서버가 보낸 패킷을 수신해 게임 월드 갱신*

void Render(); *// 렌더링 수행*

void RecvPacketFromServer(); *// 서버의 패킷을 받아 엔티티 생성, 갱신, 삭제 수행*

void SendPacketToServer(); *// 클라이언트의 입력 상태를 담은 패킷을 서버에 송신*

**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; *// Entity ID*

BehaviorType LeftPaddleBType; *// 생성, 갱신, 삭제 행동 중 하나*

Vector2 LeftPaddlePosition; *// 갱신에 필요한 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 :

**class** **Game**;

**class** **Entity**;

**class** **Client**;

장규현 :

**class** **Server**::CreateGameWorld();

**class** **Server**::UpdatePaddlesPosition();

**class** **Server**::UpdateBallsPosition();

**class** **Server**::CheckBallAndPaddle();

**class** **Server**::CheckBallAndWall();

**class** **Server**::ResetWorld();

**struct** **ServerToClientPacket**;

**struct** **ClientToServerPacket**;

박진우 :

**class** **Server**::Init();

**class** **Server**::Shutdown();

**class** **Server**::Run();

**class** **Server**::ListenThreadFunc();

**class** **Server**::ClientThreadFunc();

**class** **Server**::SendPacketToClient();

**class** **SocketAddress**;

**class** **TCPSocket**;

**class** **SocketUtil**;

**6. 개발 일정**

|  |  |  |  |
| --- | --- | --- | --- |
| 날짜 | 박진우 | 장규현 | 김명규 |
| 11월 10일 |  |  | SDL2, spdlog, entt  오픈 소스 학습 |
| 11월 11일 |  |  | class Game  class Entity |
| 11월 12일 | Client 코드 분석 | Client 코드 분석 |  |
| 11월 13일 | Client 코드 분석 | Client 코드 분석 |  |
| 11월 14일 |  |  | Client::Init  Client::Shutdown  Client::Run |
| 11월 15일 | class SocketAddress  class TCPSocket  class SocketUtil |  | Client::ProcessInput  Client::Update  Client::Render |
| 11월 16일 | Server::Init  Server::Shutdown  Server::Run | Server::CreateGameWorld  struct ServerToClient  struct ClientToServer | class System |
| 11월 17일 | Server::ListenThreadFunc |  |  |
| 11월 18일 |  |  |  |
| 11월 19일 |  |  |  |
| 11월 20일 | 테스트 및 버그 수정 | 테스트 및 버그 수정 | 테스트 및 버그 수정 |
| 11월 21일 | Server::ClientThreadFunc  Server::SendPacketToClient | Server::UpdatePaddlePosition  Server::UpdateBallPosition | Client::RecvPacketFromServer  Client::SendPacketToServer |
| 11월 22일 | 테스트 및 버그 수정 | 테스트 및 버그 수정 | 테스트 및 버그 수정 |
| 11월 23일 | 테스트 및 버그 수정 | 테스트 및 버그 수정 | 테스트 및 버그 수정 |
| 11월 24일 | ClientThread-MainThread 동기화 | Server::CheckBallAndPaddle  Server::CheckBallAndWall  Server::ResetWorld |  |
| 11월 25일 |  | 벽을 튕길 때마다 공의 속도가 변하도록 수정 |  |
| 11월 26일 | 추가 구현 및 버그 픽스 | 추가 구현 및 버그 픽스 | 추가 구현 및 버그 픽스 |
| 11월 27일 | 추가 구현 및 버그 픽스 | 추가 구현 및 버그 픽스 | 추가 구현 및 버그 픽스 |
| 11월 28일 | 추가 구현 및 버그 픽스 | 추가 구현 및 버그 픽스 | 추가 구현 및 버그 픽스 |
| 11월 29일 |  |  |  |

**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))