**네트워크 게임 프로그래밍 Term Project 추진 계획서**

5조

2014180021 박진우

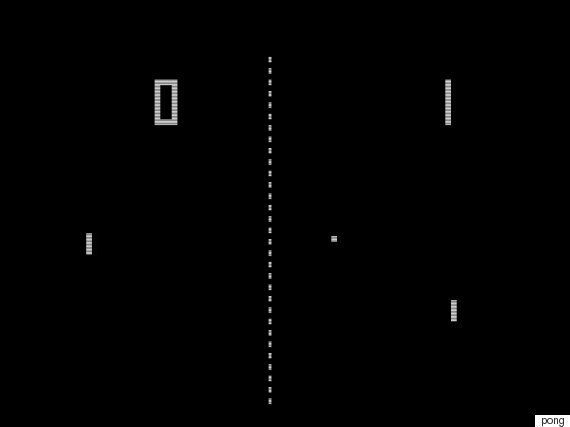
2014182038 장규현

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
8. 개발 간 발생했던 버그와 해결법
9. 지적 사항 개선

**1. 게임 소개**

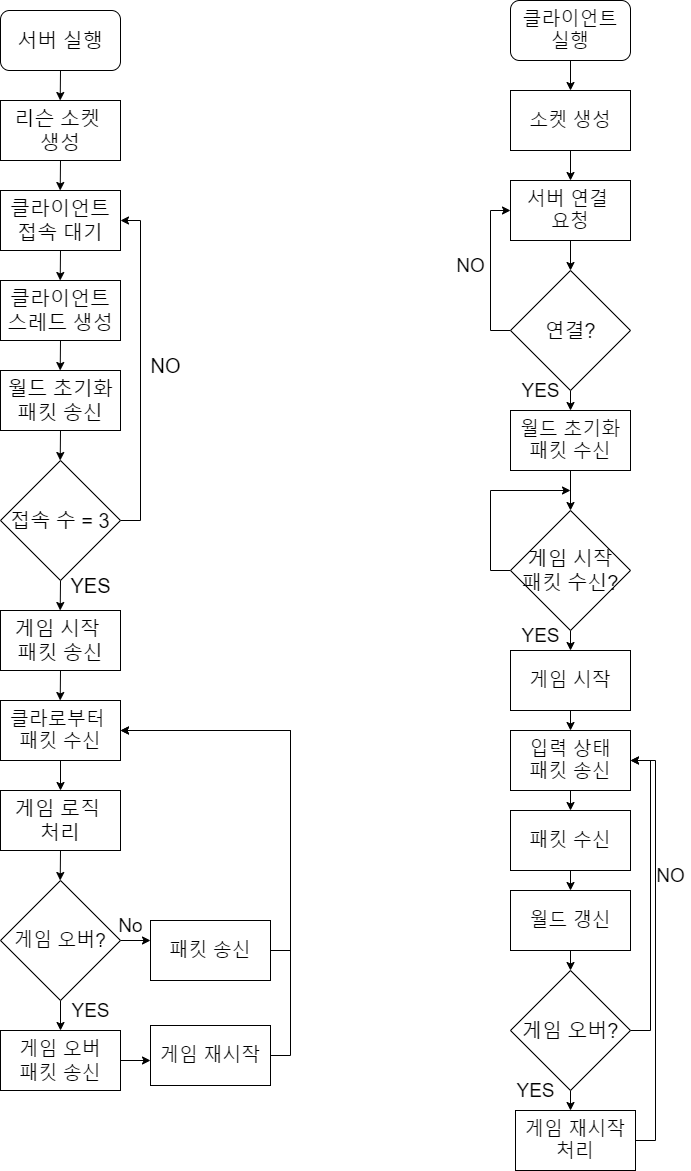
* 게임 제목 : 퐁
* 게임 장르 : 아케이드 게임
* 게임 설명 : 퐁은 탁구를 모방한 게임이다. 플레이어는 각자 막대를 조종하여 공을 쳐낸다. 공이 상대편 막대를 넘어 반대편 벽에 도달하게 되면 승리한다.
* 게임 조작법 : W,S키를 이용해 막대를 움직인다.
* 게임 예시

(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.
* 서버느 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 클라이언트 번호를 패킷에 담는다.
* 서버에선 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.~~
* 서버는 각각의 클라이언트로부터 하나씩 패킷을 수신했음을 확인한 후ㅡ예를 들어 세 개의 클라이언트가 접속해 있다면 각각으로부터 하나씩, 총 세 개의 패킷을 받은 후ㅡ 게임 로직 처리를 거쳐 갱신된 게임 월드 데이터를 클라이언트에 보낸다.
* **(추가)** 모든 클라이언트 스레드가 각자의 클라이언트에게 Hello 패킷을 보낸 이후에 주 스레드가 GameStart 패킷을 보내야 한다. 순서 제어가 필요하기에 이벤트를 사용한다.

**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*

*//////////////////////////////// 추가 ///////////////////////////////////*

**static** void AdjustXPos(**const** Vector2& paddlePosition, **const** Vector2& ballDirection, Vector2& outBallPosition); *// 공과 패들이 충돌했을 때 공의 x 위치값을 조절*

**static** CollisionSide GetCollisionSide(**const** SDL\_Rect& paddleRect, **const** SDL\_Rect& ballRect); *// 패들의 위 또는 아래 변에 공이 충돌했는지 검사*

**static** void IncreaseSpeed(float& outSpeed); *// 속도를 증가*

*///////////////////////////////////////////////////////////////////////////*

}

**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**:

*//////////////////////////////// 추가 ///////////////////////////////////*

void WaitAllPlayers(); *// 모든 플레이어 접속 대기 함수*

*/////////////////////////////////////////////////////////////////////////*

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~~*

WhoLose CheckBallAndWall(); *// 공-벽 충돌 체크 & 승패 반환*

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

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

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

void SendPacketToClient(**const** ServerToClient& packet, **const** TCPSocketPtr& target = **nullptr**);

int RecvPacketFromClient(ClientToServer& outPacket, **const** TCPSocketPtr& target);

**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*

HANDLE mIsAllHelloPacketSent; *// 모든 클라이언트 스레드가 HelloPacket을 보냈다면 신호 상태가 된다. 주스레드는 이 이벤트에 의해 깨어나 GameStart 패킷을 보냄*

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**; *// ProcessInput, Update, Render 순서대로 호출*

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

void SetClientNum(int num);

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

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

**private**:

~~bool NetworkInit();~~ *~~// 윈속 초기화~~*

void ProcessInput(); *// 활성 씬의 ProcessInput 호출*

void Update(); *// 활성 씬의 Update 호출*

void Render(); *// 활성 씬의 Render 호출*

**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;~~

**virtual** void ProcessInput(const uint8\_t\* keystate) = 0;

**virtual** void Update(float deltaTime) = 0;

**virtual** void Render(SDL\_Renderer\* renderer) = 0;

**protected**:

Game\* mOwner;

}

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

{

**public**:

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

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

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

**virtual** void ProcessInput(const uint8\_t\* keystate); *// 서버 IP 주소를 입력받음*

**virtual** void Update(float deltaTime);

**virtual** void Render(SDL\_Renderer\* renderer);

**private**:

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

bool Connects(); *// 서버에 연결 요청. 연결 성공 시 true 반환.*

}

**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~~**~~;~~

void ProcessInput(const uint8\_t\* keystate); *// 유저의 입력 상태를 패킷에 담아 서버에 송신. 송신에 Client::SendPacketToServer() 호출. 입력이 없다면 0을 보냄.*

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

void Render(SDL\_Renderer\* renderer); *// 렌더링 수행*

**private**:

void ProcessPacket(**const** ServerToClient& packet); *// 패킷 처리 함수*

void ProcessHelloPacket(**const** ServerToClient& packet);

void ProcessGameStartPacket(**const** ServerToClient& packet);

void ProcessUpdatePacket(**const** ServerToClient& packet);

void ProcessGameOverPacket(**const** ServerToClient& packet);

**private**:

int mClientNum; *// 클라이언트 번호*

bool mShouldSend; *// 초당 30번의 패킷만 보내기 위한 변수*

}

**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*

};

**enum** **class** **WhoLose**

{

Left,

Right,

LeftWin,

RightWin,

Draw,

None

};

**struct** **ServerToClient**

{

PacketType PType;

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

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

**WhoLose** Who;

*// 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 | 클라이언트 게임 시작 패킷 수신 및 처리 |
| 클라이언트 프레임워크 | 클라이언트 입력 상태 송신 | 서버 프레임워크 |
| 게임 씬 | 클라이언트 월드 초기화 패킷 수신 및 처리 | 클라이언트로부터 패킷을 받아 공유 자원 관리 |
| 클라이언트 월드 업데이트 패킷 수신 및 처리 | 게임 오버 패킷 수신 및 처리 | 충돌 체크 구현 |
| 게임 재시작 구현 | 서버 패킷 송신 구현 | 서버 게임 월드 갱신 |

**6. 개발 일정**

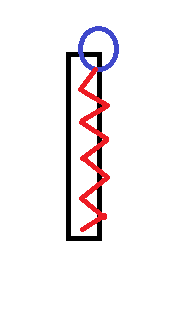
|  |  |  |  |
| --- | --- | --- | --- |
|  | **김명규** | **장규현** | **박진우** |
| **11월 10일** | 추진 계획서 수정, 프로젝트 생성 | | |
| **11월 11일** | 씬 클래스 구현 (Enter, Exit, Run) | 로그인 씬 구현 (Enter, Exit, Run) | 서버 프레임워크 (Init, Shutdown, Run) |
| **11월 12일** | 게임 씬 구현 (Enter, Exit, Run, Render) | 로그인 씬 구현 (GetIP, Render) | 서버 (ClientThreadFunc) |
| **11월 13일** | ~~게임 씬 구현 (Enter, Exit, Run, Render~~ | ~~서버 프레임워크 (HelloPacket 송신)~~ | ~~서버 (ClientThreadFunc)~~ |
| **11월 14일** |  | 서버 프레임워크 (HelloPacket 송신) |  |
| **11월 15일** | 로그인 테스트 | | |
| **11월 16일** |
| **11월 17일** | 클라이언트 패킷 구현 (PacketType::Hello) | 클라이언트 패킷 구현 (PacketType::Hello) | 클라이언트 패킷 구현 (PacketType::GameStart) |
| **11월 18일** | 게임 씬 구현 (ProcessInput, Update) |  | 클라이언트 패킷 구현 (PacketType::GameStart) |
| **11월 19일** | 게임 씬 구현 (ProcessInput, Update)  서버에 이벤트 추가  (mIsAllHelloPacketSent) | 클라이언트 패킷 구현 (ClientToServer) |  |
| **11월 20일** |  | ~~클라이언트 패킷 구현 (ClientToServer)~~ | ~~스레드 동기화~~ |
| **11월 21일** |  | ClientToServer 패킷 수신 | 스레드 동기화 |
| **11월 22일** | 서버 게임 월드 갱신, 업데이트 패킷 송신 및 수신  원격 접속 구현 | | |
| **11월 23일** | 네이글 알고리즘 끄기  초당 30번 패킷을 송신하기로 변경 | | |
| **11월 24일** | ~~클라이언트 패킷 구현 (PacketType::Update)~~ | 서버 충돌 체크  (패들-벽, 공-벽) | 서버 충돌 체크  (공-패들) |
| **11월 25일** | ~~클라이언트 패킷 구현 (PacketType::Update)~~  화면 위에 점수 표시 | 클라이언트 패킷 구현 (PacketType::GameOver) | 서버 패킷 송신 (PacketType::GameOver) |
| **11월 26일** | ~~패들, 공 움직임 테스트~~ | | |
| **11월 26일** | 두 번째 공 추가  L2, R2 패들 추가 |  |  |
| **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)~~  플레이어 패들 색깔 변경 |
| **12월 1일** | ~~게임 씬 구현(재시작)~~  R2 패들 크기 조절 | ~~클라이언트 패킷 구현 (PacketType::GameOver)~~  튕길 때 마다 공 속도 증가  공이 튕기는 방향 다양화 | ~~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))

**8. 개발 간 발생했던 버그와 해결법**

**충돌 처리**

**** (문제 상황 예시)

11월 29일 수업 시간에 진행된 충돌 테스트 중, 공이 패들에서 벗어나지 못하고 패들에 박혀 부르르 떠는 오류가 관찰되었다. 문제 상황은 패들의 윗변과 아랫변에 공이 충돌했을 때, 충돌면과 동일한 방향으로 플레이어가 패들을 움직이면 발생했다. 즉 공이 윗변에 충돌했을 때 플레이어가 W키를 눌러 패들을 위로 움직이거나 공이 아랫변에 충돌했을 때 S키를 눌러 패들을 아래로 움직였을 때 100% 확률로 이런 문제가 나타났다. 플레이어의 조작으로 인해 공이 패들과의 충돌에서 미처 빠져나가지 못하고 연속적으로 충돌하게 된 것이 원인이었다.

이 문제를 해결하기 위해 두 개의 함수를 정의했다. Systems::GetCollisionSide는 공이 패들의 윗변이나 아랫변에 충돌했는지 확인해준다. 패들의 윗변과 아랫변에 작은 사각형을 만들고 이 사각형들과 공이 충돌했는지 확인한다. Systems::AdjustXPos는 만약 공이 패들의 위, 아래변에 충돌했다면 공이 x 위치값을 조정해줌으로써 패들과 추가적으로 충돌이 발생하지 않게끔 만들어준다.

**응답 지연**

기존의 설계에선 서버가 클라이언트로부터 패킷을 받을 때마다 게임 로직을 수행하고 모든 클라이언트에게 패킷을 송신했다. 그러다보니 클라이언트에서 입력 지연이 발생했다. 게임을 제대로 플레이하지 못할 정도였다. 신속히 고쳐야 할 필요가 있었다.

11월 23일 수업 때 교수님의 조언을 받아 초당 보내는 패킷 수를 서버와 클라이언트 모두 30개로 제한했고 모든 TCP 소켓의 네이글 알고리즘을 껐다. 그리고 서버는 패킷을 받을 때마다 게임 로직을 수행하는 게 아니라, 각 클라이언트로부터 패킷을 모두 받으면 게임 로직을 수행하도록 변경했다. 결과는 좋았다. 지연은 없어졌고 정상적인 플레이가 가능하게 되었다.

**9. 지적 사항 개선**

**SDL\_PollEvent**

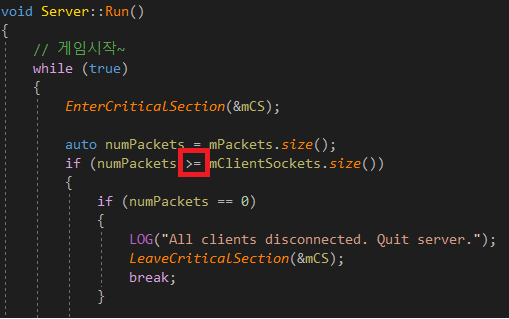
****

(문제의 코드)

윈도우의 메시지 루프와 소켓 코드를 분리하지 않으면 생기는 문제들을 간과하고 클라이언트에선 하나의 스레드로 모든 걸 처리하고 있었다. 지금은 메시지 루프에선 수행 시간이 오래 걸리는 작업을 전혀 하지 않기에 우리의 프로그램은 문제 없이 동작했었다. 하지만 서버에 이상이 생겨 패킷이 도착하지 않는다면 얄짤없이 프로세스가 블록되기에 이런 문제를 해결하기 위해선, 메시지 루프 코드나 소켓 코드 중 하나를 별도의 스레드로 분리할 필요가 있다.

SDL\_PollEvent()는 자신의 스레드에 관련된 메시지만 폴링한다. SDL 윈도우가 주 스레드에 생성되어 있기에 스레드를 두 개로 늘린다면, 윈도우에 관련된 메시지를 폴링하기 위해서 메시지 루프 코드는 주 스레드에서 처리하고 소켓 코드는 별도의 스레드로 분리해야 한다.

**모호한 비교 연산자의 사용**

****

(문제의 연산자)

블로킹 소켓의 사용과 클라이언트의 설계로 지역 변수 numPackets의 값은 Server::mClientSockets의 size값과 같을지언정 절대 초과할 수 없었다. 개발 중 우리도 이 사실을 인지하고 있었으나 문제되지 않는다고 판단해 고치지 않았다. 하지만 검수 과정 중 이 비교 연산자로 인해 코드 파악에 어려움이 생겼고, 우리는 대수롭지 않게 생각했던 이런 작은 부분이 사실은 매우 치명적이라는 걸 깨달았다.

코드는 나 혼자만 보는 게 아니라는 걸, 언제나 내 코드를 읽을 다른 사람을 고려해서 코드를 작성해야 한다는 교훈을 얻게 되었다. 앞으로 코드를 작성할 땐 위와 같이 모호함을 줄 수 있는 코드는 명확하게 바꿀 것이다. (최종제출본에는 ‘==’ 연산자로 교체)