**Network Game Programming**

**최종 보고서**

**Team 10**

2015180040 채영문

2018180010 김연규

2018184036 최재준

* 목차

1. 애플리케이션 기획

1-1. 게임 설명

1. High - Level Design
2. Low - Level Design

3-1. Server, Client 공통 구조체, 함수

3-2. Server 전용 구조체, 함수

3-3. Client 전용 구조체, 함수

3-4. 쓰레드 동기화

1. 팀원 별 역할분담
2. 개발환경
3. 개발일정

6-1. 채영문

6-2. 김연규

6-3. 최재준

**1. 애플리케이션 기획**

* **메인 화면 씬**



* **게임 플레이 씬**



**게임이름 – 교과목 및 제작자**

Power Ing (윈도우 프로그래밍 – 김연규)

* **게임목표**

3명의 플레이어들이 오브를 튕겨 점수를 얻어 경쟁하는 게임

* **게임내용**
* 각 플레이어는 메인 화면에서 모두 준비(레디)를 하면 애니메이션과 함께 게임이 시작
* 게임을 시작하면 오브가 움직이기 시작하고 플레이어들은 자신에게 부여된 색깔의 패널을 레일에 따라 움직인다.
* 패널을 움직여 오브를 튕기면 득점한다.
* 패널은 서로 부딪힐 수 있으며 겹치지 않는다.
* 일정 점수를 얻거나 오브가 경기장 밖으로 나가 오브가 모두 소모되면 게임을 종료한다.
* 종료 이후 우승자와 함께 모든 플레이어 점수를 알려준다.



* **게임조작**

메인 씬

Ready: 레디 버튼 클릭하여 준비한다.

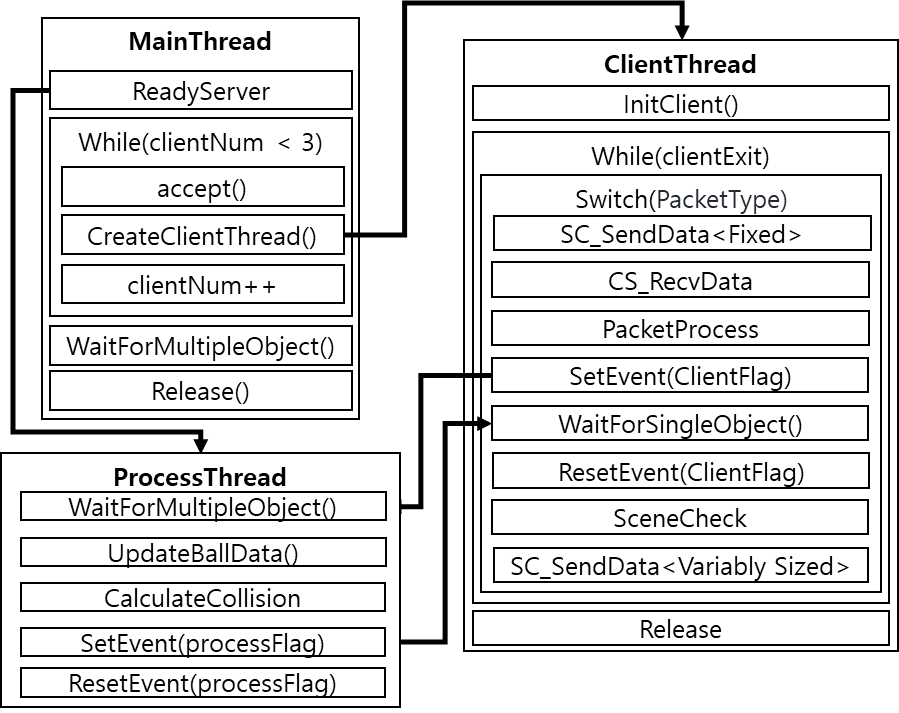
Quit: 게임을 종료한다.

게임 플레이 씬

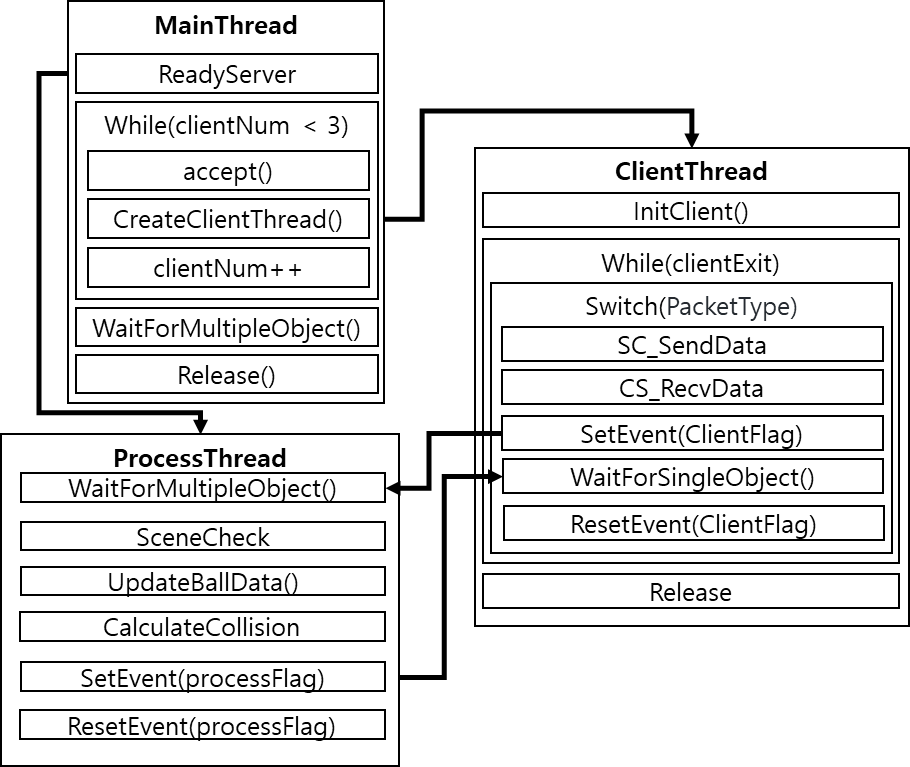
좌/우 방향키: 패널을 움직일 수 있다.

**2. High – Level Design**

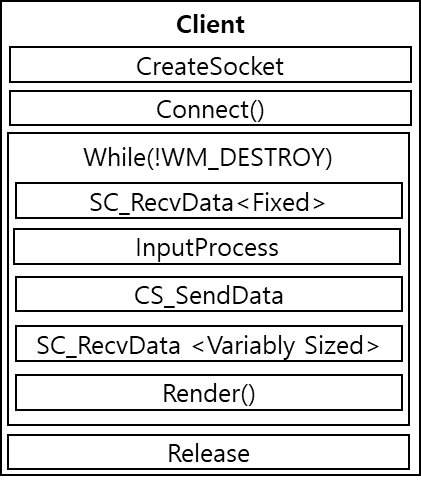
**Server**

****

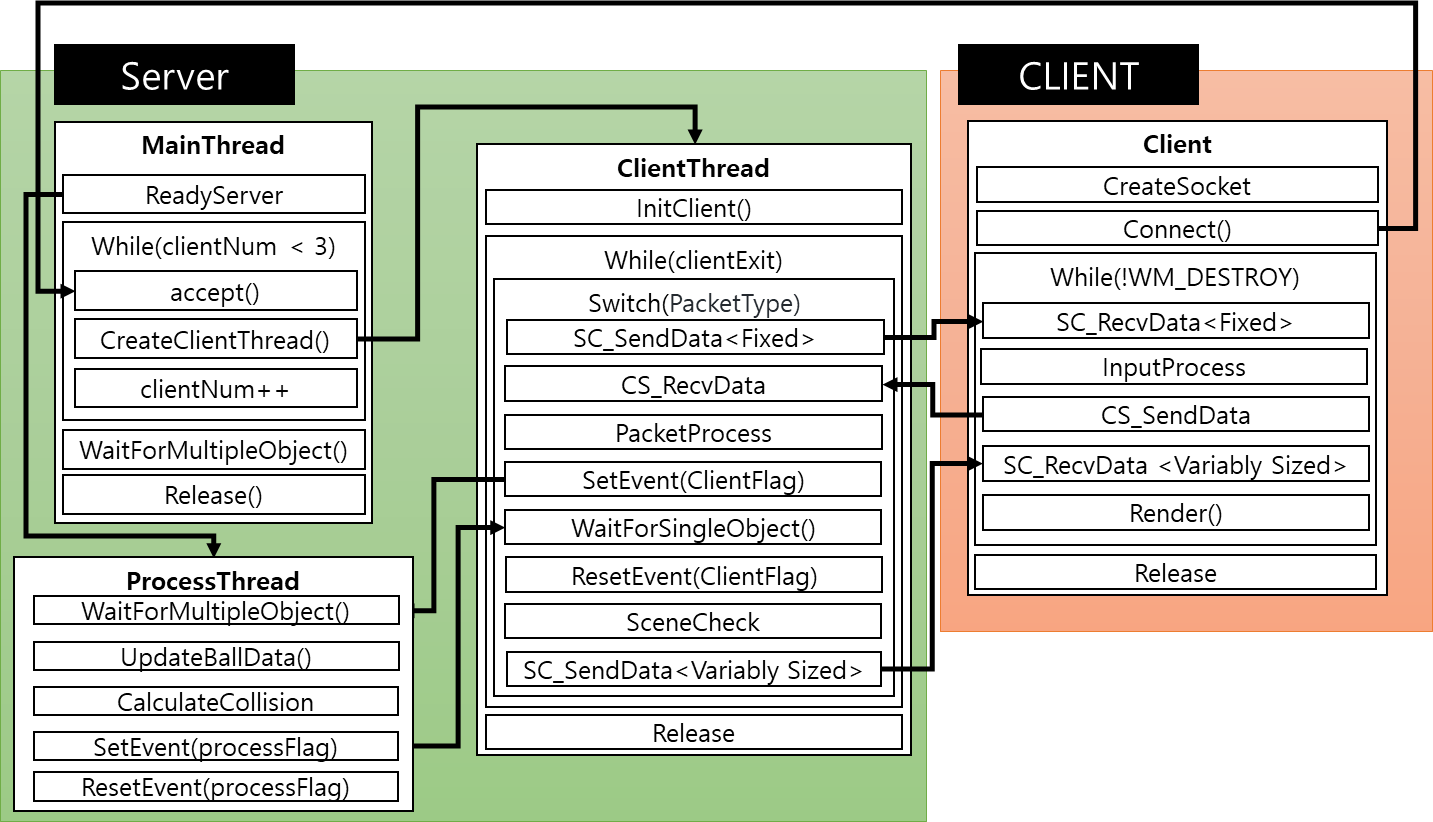
**Server**

****

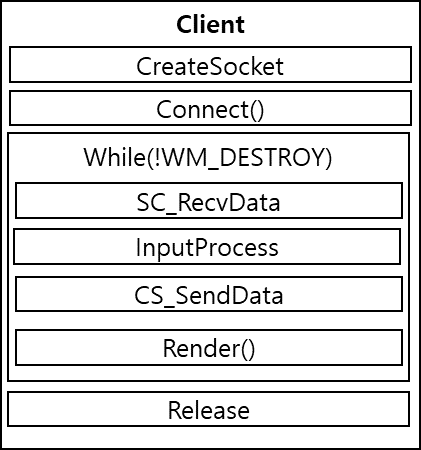
**Client**

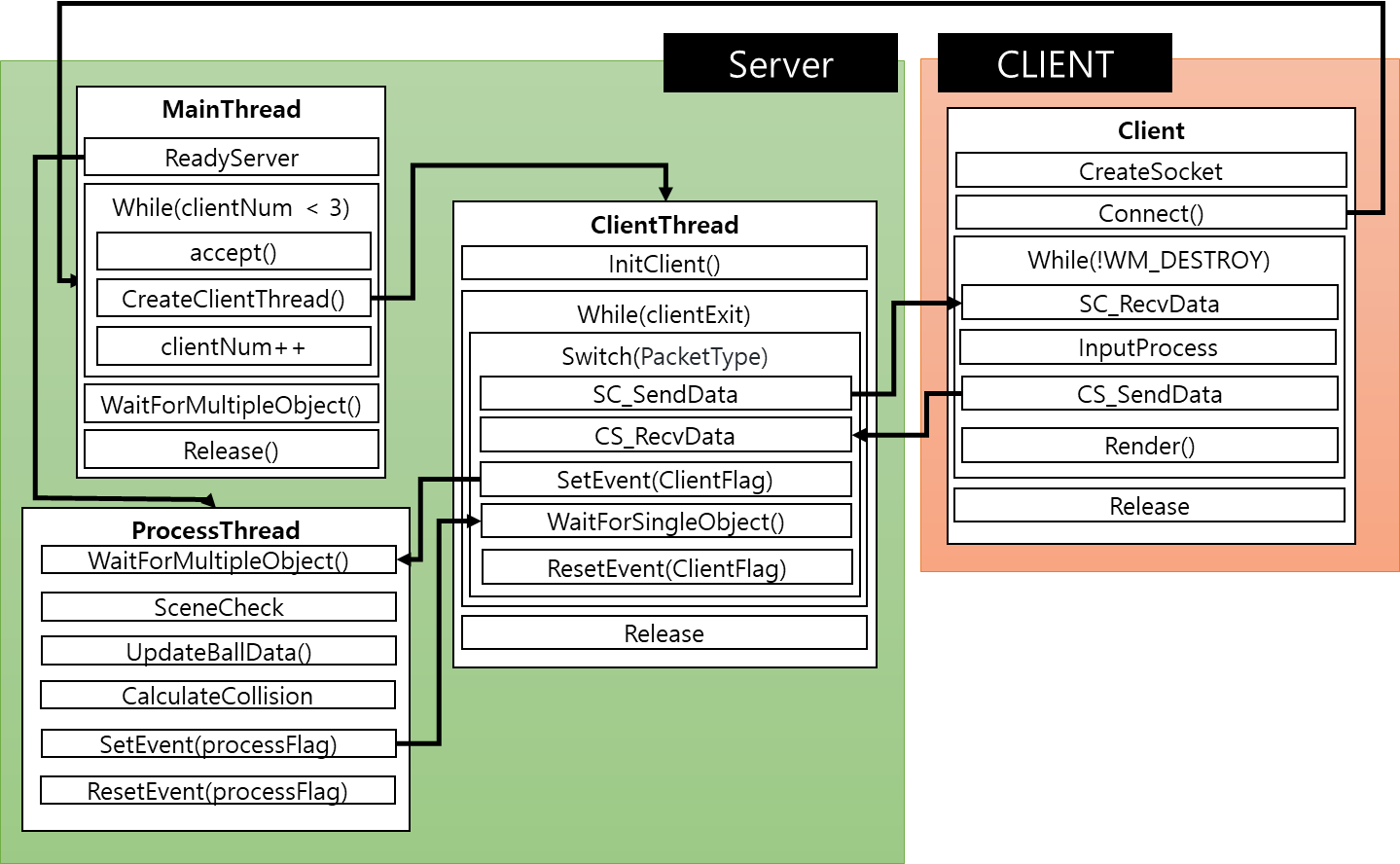


**Server-Client 관계도(이전)**



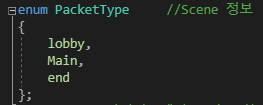
**Server-Client 관계도**

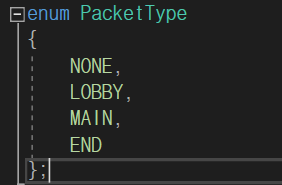
****



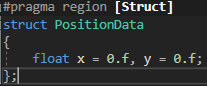
**3. Low – Level Design**

**3-1: Server & Client 공통 함수, 구조체**

 **->**

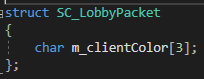
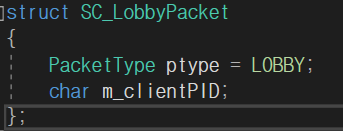


패킷 타입: 각 Scene의 이름으로 타입을 구별. 고정 길이로서 서버에서 Send()

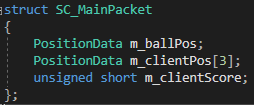
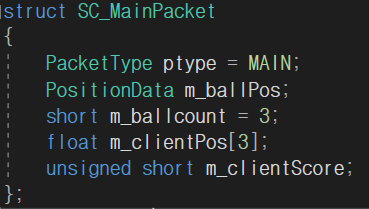


2D 클라이언트 기반 위치정보 구조체

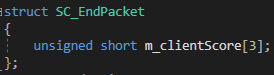
**3-2: Server 전용 구조체, 함수**

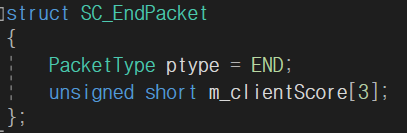
** ->** 

로비 Scene에서의 Server to Client 패킷 구조체: 클라이언트 ID

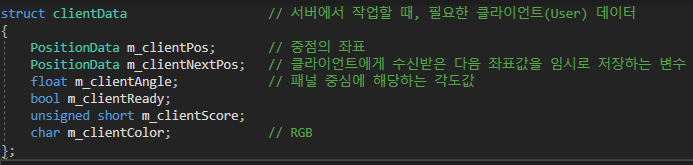
**->**

메인 Scene에서의 Server to Client 패킷 구조체: 패킷 타입, 공 위치, 패널 3개위치, 점수, 공 개수

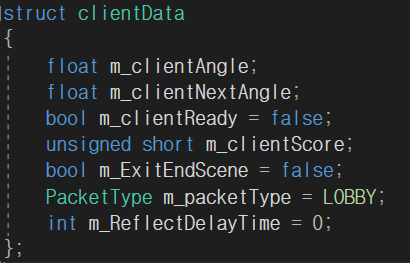
** ->**

****

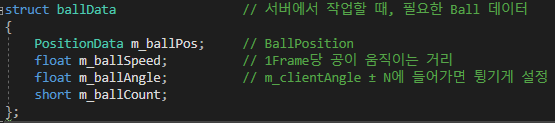
엔드 Scene에서의 Server to Client 패킷 구조체: 모든 플레이어 점수 송신

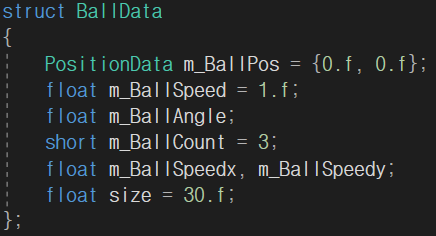


**->**



서버에서 관리하는 클라이언트 데이터 구조체, 컨테이너 타입 array[3], 전역변수로 관리

**->**

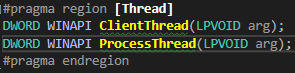


서버에서 관리하는 공 데이터 구조체, 전역변수로 관리

 -> 이름만 변경

HANDLE clientThreadFlag[3]: 통신 쓰레드에서 계산 쓰레드로 신호상태를 전달하기 위한 Flag, 전역으로 선언

HANDLE processThreadFlag: 계산 쓰레드가 종료되었음을 통신 쓰레드에게 알리는 Flag, 전역으로 선언



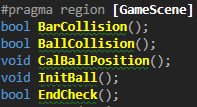
**ClientThread(LPVOID arg):** 클라이언트 통신 쓰레드, 클라이언트 연결 요청마다 호출

**ProcessThread(LPVOID arg):** 서버내의 모든 객체 위치 값 계산, 충돌처리 등 게임 내 Update를 위한 함수



**bool ReadyCheck():** 플레이어들의 준비 상태 값을 수신 받고, if (ReadyCount == 3) 확인 후 Scene State를 Lobby에서 MainGame으로 넘겨주는 함수

**void InitClient():** ~~클라이언트 접속시에 데이터(PID) 초기화~~, 게임 시작 시 각 클라 게임패널 위치, 점수 초기화



**bool BarCollision():** 플레이어 간 충돌 처리 함수. 충돌 시 True를 반환

**~~bool BallCollision():~~** ~~플레이어와 공 간 충돌 처리 함수. 충돌 시 True를 반환~~

**void SetBallPosition():** 공의 다음 위치를 계산 후 이를 저장

**void InitBall():** 공 속도 및 각도(벡터) 초기화, ~~남은 공 개수 감소~~

**bool EndCheck():** 게임이 종료 여부 확인



**int SC\_SendData(SOCKET soc**k): 클라이언트로 패킷을 송신

**int CS\_RecvData(SOCKET sock):** 클라이언트로부터 패킷을 수신, 이후 패킷 데이터 저장

**기존 게임 내 함수. 서버 재활용 함수들**

**~~void GeneralReset():~~** ~~공, 버튼, 타이머, 씬, 점수, 이펙트 타임 등 변수 Looby초기화~~

**Power\_Orb\* OrbPosition(Power\_Or**b\*): 속도에 따른 공의 다음 위치 계산

**Power\_Orb\* OrbSpeed(Power\_Orb\*):** 공 속도 계산 **– 로직 활용**

**~~Void OrbCreate(Power\_Orb\*, int, bool, float x, float y, float radian)~~**

**~~: 공~~** ~~객체 생성 초기화~~ -> InitBall()에 포함

**~~Void OrbRemove(Power\_Orb\* postOrb, Power\_Orb\* orb):~~** ~~공 1개 삭제~~

**~~Void OrbClear(Power\_Orb\* orb):~~** ~~공 삭제 및 리셋~~ InitBall()에 포함

**~~Power\_Orb\* OrbApply(Power\_Orb\*, int Type, bool, float x, float y, float radian)~~**

~~: 공 생성 후 크기 값 초기화 적용~~ InitBall()에 포함

**void CollisonDetect(Power\_Orb\* orb):** 공 충돌 처리 함수

**Power\_Orb\* ReflectReflectorOrb(Power\_Orb\*, Power\_Reflector\*)**

: 충돌 후 공 위치 값 조정 - **로직 활용**

**Power\_Orb\* ReflectOrb(Power\_Orb\* orb, float angle)**

**:** 충돌한 공의 각도(방향) 조정 - **로직 활용**

**void ReflectDetect(Power\_Orb\*, Power\_Reflecto**r\*): 공과 패널 충돌 처리

**void ReflectReflector(Power\_Orb\*, Power\_Relector\*):** 패널 충돌 처리 후 부가작업(점수 계산, ~~이펙트 애니메이션~~) 처리 함수

**~~void ReflectorCreate(struct Power\_Reflector\* Reflector, int Count): 패널 객체 생성 초기화~~ ->initClient 포함**

**struct Power\_Reflector\* ReflectorReset(struct Power\_Reflector\* Reflector):**

**패널 리셋 (엔드씬 적용) ->initClient 포함**

**float OrbScore(float Speed): 스피드에 따른 점수 계산 -> 충돌함수에 포함**

**충돌용 보조함수**

**double AnglePosition(double x, double y): 위치를 통해 각도를 계산**

**bool DistanceOvercmp(double x, double y, double dis): 거리 초과 여부**

**bool DistanceDetect(double x, double y, double Angle, double Distance, double Size): 충돌판단에 필요한 거리 체크**

**bool AngleDetect(double x, double y, double Angle): 충돌판단에 필요한 각도 체크**

**bool ObtuseDetect(double Angle): 둔각 여부 판단**

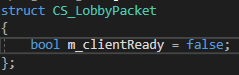
**float AngleOverflow(float angle): 각도가 360을 넘어가면 다시 0으로 변환**

**void SpeedCaculate(double time): 시간에 비례해서 점점 공속도가 빨라짐**

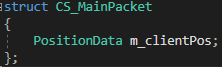
**void ChangetoLobby(): 플레이어 입력에 엔드씬에서 로비씬으로 변경해주는 함수**

**void ChangePacket(PacketType pType): 패킷타입 변경해주는 함수**

**3-3: Client 전용 구조체, 함수**



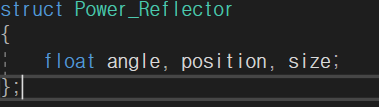
~~로비Scene에서의 Client to Server 패킷 구조체: 클라이언트의 Ready값 송신~~



~~메인 Scene에서의 Client to Server 패킷 구조체: 클라이언트 Position값 송신~~

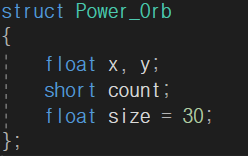
클라이언트 -> 서버 패킷 데이터 구조체를 받은 데이터 전역변수 수정으로 변경

텍스트이(가) 표시된 사진

자동 생성된 설명 -> 

클라이언트 렌더링을 위한 클라 내 패널 정보 구조체

텍스트이(가) 표시된 사진

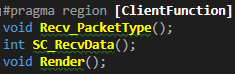
자동 생성된 설명 -> 

클라이언트 렌더링을 위한 클라 내 공 정보 구조체

텍스트이(가) 표시된 사진

자동 생성된 설명

~~클라이언트 렌더링을 위한 클라 내 이펙트 관리 구조체~~ -> 이펙트 전면 삭제

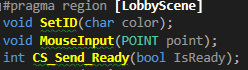


**~~void Recv\_PacketType():~~** ~~서버로부터 고정 길이 데이터인 패킷 타입을 받는 함수~~

**void recv\_data(PacketType\*, bool\*, int\* ): 고정길이+가변길이에서 완전 가변으로 변경, 각 패킷의 가장 첫 부분에 패킷 타입을 달고 패킷 송수신**

**int SC\_RecvData():** 서버로부터 받은 가변 길이 패킷 데이터를 클라이언트 메모리에 저장하는 함수

**~~void Render():~~** ~~메모리에 저장된 데이터로 이를 출력하는 함수~~



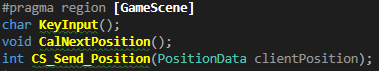
**~~void SetID(char color):~~** ~~서버로부터 부여받은 클라이언트 ID를 저장하고, 이를 토대로 본인 색을 지정하는 함수~~

**void MouseInput(POINT point):** 마우스 입력 값을 토대로 버튼 충돌처리 이후 클라이언트 상태 값을 바꾸는 함수

**~~int CS\_Send\_Ready(bool IsReady):~~** ~~서버로 현재 클라이언트 상태정보를 Send하는 함수~~ - -> Send()로 통합

**int Send(bool bReady): 클라이언트가 서버로 main scene에 진입할 준비가 되었는지 전송하는 함수.**

**Int Send(float Angle): 클라이언트가 서버로 main scene진행 중에 자신의 각도값을 보내는 함수**



**char KeyInput(char chInput):** 메인Scene에서 어떠한 키를 입력했는지 반환하는 함수

**~~void CalNextPosition();~~** ~~입력 값을 토대로 클라이언트 다음 위치 값을 갱신하는 함수~~

**-> 서버로 전송 후 서버에서 갱신**

**int CS\_Send\_Position(PositionData clientPos):** 키가 눌렸다면 클라이언트의 다음 위치 값을 전송하는 함수.

**기존 게임 내 함수. 클라 재활용 함수들**

**~~void CreateEffect(struct Power\_Effect\* Effect, double x, double y, double Score): 이펙트 객체 생성~~**

**void ReflectorPosition(struct Power\_Reflector\* Reflector, short Left, short Right): 눌린 키보드 입력에 따른 패널 각도 계산 (서버 전송용)**

**~~void RemoveEffect(struct Power\_Effect\* NextEffect, struct Power\_Effect\* Effect): 이펙트 삭제~~**

**~~void EffectPrint(struct Power\_Effect\* Effect): 생성된 이펙트 출력~~**

**void DisplayLoad(): 이미지 불러오기 및 색상 적용**

**void DisplayWindow(): 윈도우 출력용 사이즈 계산**

**void DisplayColorApply(): 색상이 있는 이미지 색상값 부여**

**Render()에 포함할 함수들**

**void DisplayOrb(struct Power\_Orb\* Orb): 공 출력**

**void DisplayReflector(struct Power\_Reflector\* Reflector): 패널 출력**

**void UIMenu(bool Start, bool Module, bool Option, bool Quit, bool Esc): 로비 UI출력**

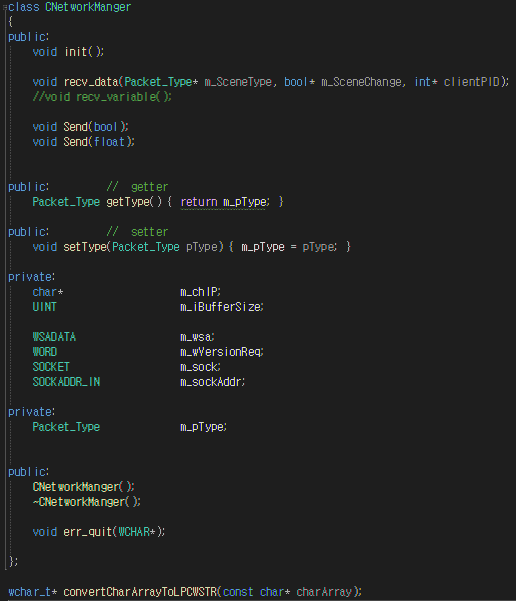
**void UIEndMessage(): 게임 종료 메시지 출력**

**void UIScore(): 현재 클라 점수 출력**

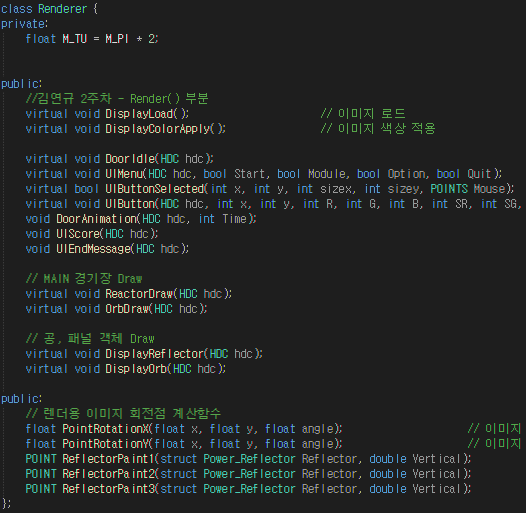
**~~void UIDebugInfo(): 디버그 출력~~**

**void DoorAnimation(): 문 애니메이션 출력**

**신규:**



**Class CNetworkManger: 클라이언트 내에서 서버와의 네트워크 통신을 담당하는 해당 클래스는 소켓 생성, 초기화, 연결, 종료를 모두 책임지고 수행한다.**

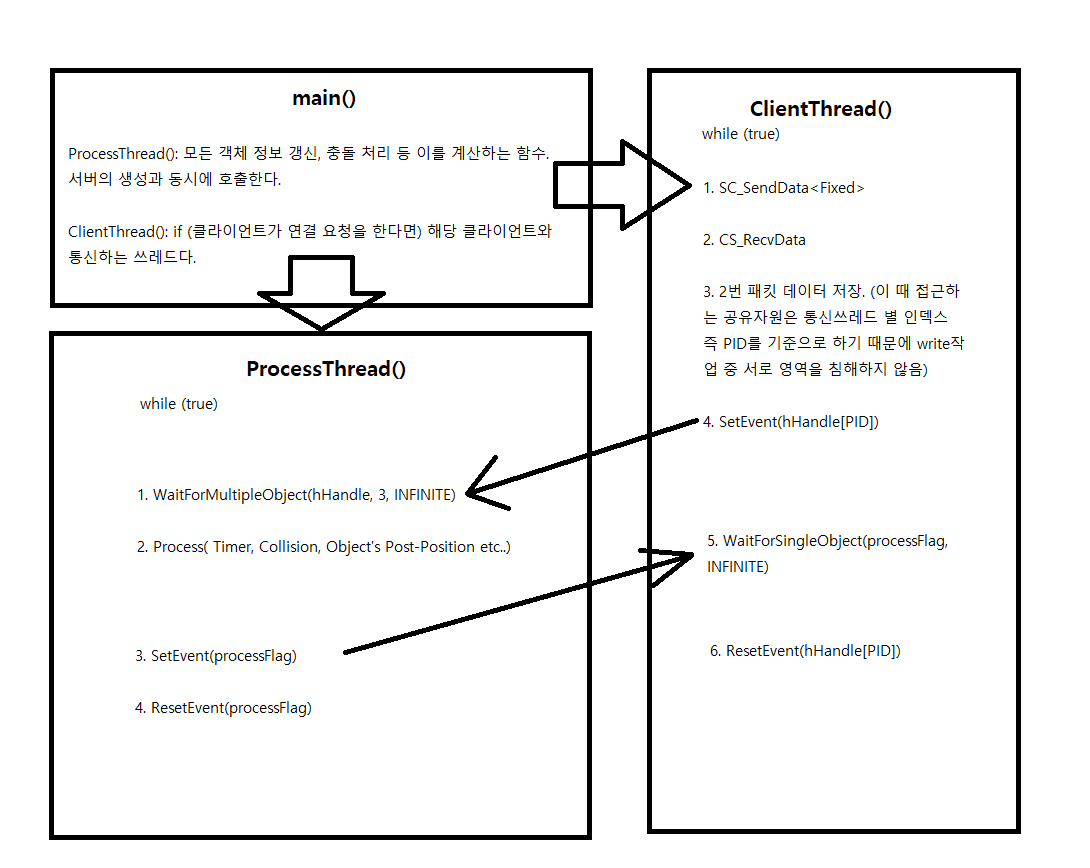


**Class Renderer: 기존 게임의 모든 출력을 담당하는 클래스로, 로드 된 이미지와 오브젝트 객체 및 점수를 출력하며, 출력용 이미지의 좌표 계산을 포함하여 모든 메소드를 관리한다.**

**3-4: 쓰레드 동기화**

이벤트기법을 사용하여 서버 프로그램 내의 쓰레드 간 동기화 구현

쓰레드 동기화 플로우 차트



1. 통신 쓰레드는 클라이언트로부터 수신 받은 데이터를 저장한 후 자신에게 할당 되어있는 Flag를 신호상태로 바꾸고 대기한다.
2. 계산 쓰레드는 3개의 통신 쓰레드로부터 신호를 받은 후 서버내에서 일어나는 모든 계산(타이머, 충돌처리, 객체 위치 계산 등)을 처리한 후 자신에게 할당되어 있는 Flag를 신호상태로 바꾸고 그 즉시 비신호 상태로 대기한다. **무한 대기 였으나, 약 0.5초 대기로 변경, 타임 아웃시에 이전 계산 값 반환**
3. 대기하고 있던 3개의 통신 쓰레드는 계산 쓰레드의 Flag가 신호 상태를 확인하는 즉시 자신의 Flag를 비신호 상태로 바꾸고 위의 1번으로 돌아간다.

해당 쓰레드 동기화의 특징은 모두 수동 리셋 이벤트 방식을 채택하였다.

**4. 팀원 별 역할(원본)**

|  |  |
| --- | --- |
| **채영문** | **김연규** |
| **Server** | **Server** |
| void InitClient() | bool EndCheck() |
| int SC\_SendData(SOCKET sock) | void InitBall() |
| ClientThread(LPVOID arg) | int CS\_RecvData(SOCKET sock) |
| Main() | bool BarCollision() |
| **Client** | **Client** |
| void SetID(char color) | void CalNextPosition() |
| void MouseInput(POINT point) | void Recv\_PacketType() |
| int SC\_RecvData() | void Render() |

|  |
| --- |
| **최재준** |
| **Server** |
| bool ReadyCheck() |
| void CalBallPosition() |
| bool BallCollision() |
| ProcessThread(LPVOID arg) |
| Client |
| int CS\_Send\_Ready(bool IsReady) |
| char KeyInput(char chInput) |
| int CS\_Send\_Position(PositionData clientPos) |

**5. 개발 환경**

**컴파일러: Visual Studio 2022**

**개발 언어: C, C++**

**라이브러리: Win32 API / WinSock 2.2**

**버전 관리 툴: Git (GitHub)**

**6. 개발 일정(원본)**

6-1: 채영문

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **일** | **월** | **화** | **수** | **목** | **금** | **토** |
| **1주차** |  |  |  | **11월 2일** | **3** | **4** | **5** |
| **채영문** |  |  |  | 검토  및  회의 | ClientThread  (LPVOID org) | ClientThread  (LPVOID org) |  |
| **달성율** |  |  |  |  |  |  |  |
| **2주차** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| **채영문** | InitClient | SC\_RecvData() | SC\_SendData()  Recv\_pType() | 검토  및  회의 | 2주차 피드백 내용 추가 및 수정 | 2주차 피드백 내용 추가 및 수정 |  |
| **달성율** |  |  |  |  |  |  |  |
| **3주차** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
| **채영문** |  | MouseInput() | SetID() | 검토  및  회의 | 3주차 피드백 내용 추가 및 수정 | 3주차 피드백 내용 추가 및 수정 |  |
| **달성율** |  |  |  |  |  |  |  |
| **4주차** | **20** | **21** | **22** | **23** | **24** | **25** | **26** |
| **채영문** |  | ClientThread()쓰레드 동기화 | ClientThread()  쓰레드 동기화 | 검토  및  회의 | 4주차 피드백 내용 추가 및 수정 | 4주차 피드백 내용 추가 및 수정 | 백업  (추가 함수 존재시에 구현) |
| **달성율** |  |  |  |  |  |  |  |
| **5주차** | **27** | **28** | **29** | **30** | **12월 1일** | **2일** | **3일** |
| **채영문** | 백업 | 백업 | 백업 | 프로젝트 최종 디버깅 |  |  |  |
| **달성율** |  |  |  |  |  |  |  |
| **6주차** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |

6-2: 김연규

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **일** | **월** | **화** | **수** | **목** | **금** | **토** |
| **1주차** |  |  |  | **11월 2일** | **3** | **4** | **5** |
| **김연규** |  |  |  | 검토 및 회의 |  | InitBall() |  |
| **달성율** |  |  |  |  |  |  |  |
| **2주차** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| **김연규** | ~~CalNextPosition()~~  EndCheck() |  | CS\_RecvData(SOCKET sock) | 검토 및 회의 | SC\_Recv\_PacketType() | | |
| **달성율** |  |  |  |  |  |  |  |
| **3주차** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
| **김연규** | CS\_Recv\_PacketType() |  | CS\_RecvData(SOCKET sock) | 검토 및 회의 | Render() | | |
| **달성율** |  |  |  |  |  |  |  |
| **4주차** | **20** | **21** | **22** | **23** | **24** | **25** | **26** |
| **김연규** | ~~EndCheck()~~  CalNextPosition() |  | BarCollision() | 검토 및 회의 | CS\_RecvData(SOCKET sock) | | |
| **달성율** |  |  |  |  |  |  |  |
| **5주차** | **27** | **28** | **29** | **30** | **12월 1일** | **2일** | **3일** |
| **김연규** | 부족한 부분 완성 | 부족한 부분 완성 | 부족한 부분 완성 | 구현 테스트 |  |  |  |
| **달성율** |  |  |  |  |  |  |  |
| **6주차** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **김연규** |  |  |  |  |  |  |  |
| **달성율** |  |  |  |  |  |  |  |

6-3: 최재준

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **일** | **월** | **화** | **수** | **목** | **금** | **토** |
| **1주차** |  |  |  | **11월 2일** | **3** | **4** | **5** |
| **최재준** |  |  |  | 검토  및  회의 | ProcessThread |  | CS\_Send\_Ready |
| **달성율** |  |  |  |  |  |  |  |
| **2주차** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| **최재준** | CS\_Send\_Ready |  | CS\_Send\_Position | 검토  및  회의 | CS\_Send\_Position |  | ReadyCheck |
| **달성율** |  |  |  |  |  |  |  |
| **3주차** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
| **최재준** | KeyInput |  | CalBallPosition | 검토  및  회의 |  |  | ProcessThread |
| **달성율** |  |  |  |  |  |  |  |
| **4주차** | **20** | **21** | **22** | **23** | **24** | **25** | **26** |
| **최재준** |  | Thread동기화 ProcessThread | | 검토  및  회의 | Test | | |
| **달성율** |  |  |  |  |  |  |  |
| **5주차** | **27** | **28** | **29** | **30** | **12월 1일** | **2일** | **3일** |
| **최재준** | Test | | | 검토  및  회의 |  |  |  |
| **달성율** |  |  |  |  |  |  |  |
| **6주차** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **최재준** |  |  |  |  |  |  |  |
| **달성율** |  |  |  |  |  |  |  |

**실제 개발 과정**

[ 1주차 (11.2 ~ 11.8) ]

|  |  |  |
| --- | --- | --- |
| 김연규 | 재영문 | 최재준 |
| 보고서 수정(재검사) | 보고서 수정(재검사)  Client Project FrameWork  - NetworkManager 클래스 | 보고서 수정(재검사)  ProcessThread(LPVOID arg)  ClientThread(LPVOID arg)  Server Project FrameWork |

[ 2주차 (11.9 ~ 11.15) ]

|  |  |  |
| --- | --- | --- |
| 김연규 | 재영문 | 최재준 |
| - Renderer 클래스  : Lobby Scene  : Main Scene | - MouseInput  - CS\_SendData  :LobbyPacket | - CS\_RecvData  - ReadyCheck  - BallCollision |

[ 3주차 (11.16 ~ 11.22) ]

|  |  |  |
| --- | --- | --- |
| 김연규 | 재영문 | 최재준 |
| - InputKey  - BarPosition  - Bar-Ball Collision  - Client,Server PacketType | - Bar-Ball Collision  - MulitThreadClient  BackupBranch | - CalBallPosition  - ClientPID  - SC\_SendData |

[ 4주차 (11.23 ~ 11.29) ]

|  |  |  |
| --- | --- | --- |
| 김연규 | 재영문 | 최재준 |
| 충돌오류 해결  -SC\_RecvData | - 쓰레드 동기화  - BallCounting  - ScoreCounting | - 쓰레드 동기화  - BarColorApply  - Bar-BarCollision |

[ 5,6주차 (11.30 ~ 12.13) ]

|  |  |  |
| --- | --- | --- |
| 김연규 | 재영문 | 최재준 |
| - Renderer 클래스  : EndScene | - 쓰레드 동기화 전반 수정 | - 쓰레드 동기화 전반 수정  -EndScene To LobbyScene |
| 프로젝트 검수 및 마무리  테스트 및 수정 | | |