**2021 NGP**

**Term Project Progress Report**

제출일: 2021.12.16

9팀

2018182021 윤성주

2018184033 최경훈

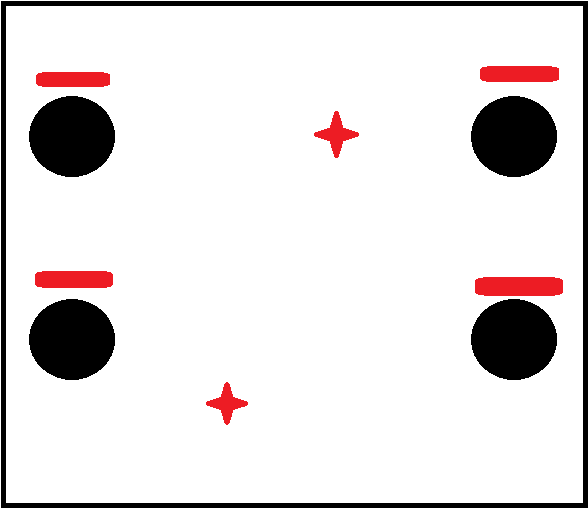
2015182020 안정인

1. **어플리케이션 기획**
   1. **참고 게임**

텍스트이(가) 표시된 사진

자동 생성된 설명  
‘위자드 오브 레전드’ 모작 인게임 화면

* 1. **구상도**



* 1. **게임 설명**

2:2 전투 게임이다. 2명씩 팀으로 하여 상대팀을 공격한다. 플레이어마다 체력바가 있으며 체력바가 소진되면 사망한다. 마지막까지 살아남는 팀이 승리한다.

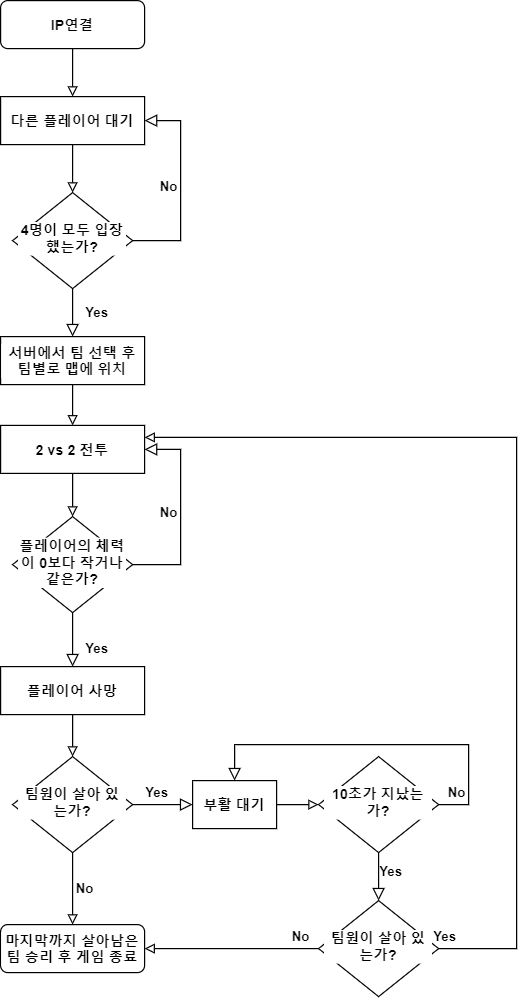
* 1. **게임 진행 순서**

1. **대기실 입장**
   * 1. 게임 시작화면
     2. IP 연결을 하고 대기실에 입장한다.
     3. 클라이언트 4명이 입장할 때까지 기다린다.
     4. ~~4명이 모두 입장하면 게임이 시작된다.~~

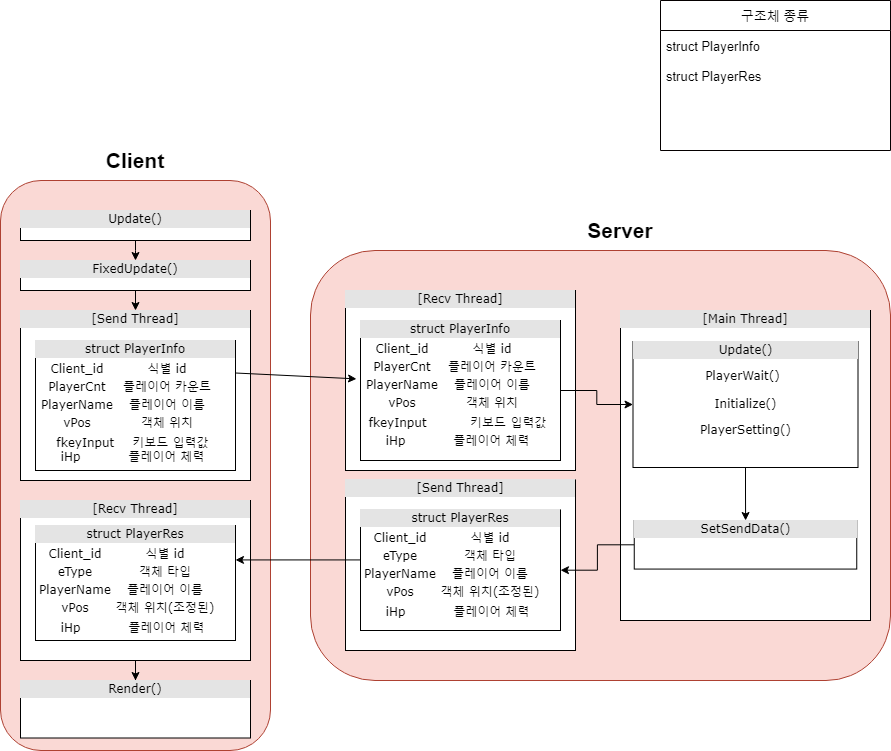
④ 4명이 모두 입장하면 5초 후 게임이 시작된다.

1. **인게임**
   1. 게임이 시작되면 2명씩 양 사이드에 배치된다.
   2. 다양한 스킬을 통해 상대팀을 공격한다. 피격 시 체력이 줄어든다. 이때, 스킬에 따라 다르게 줄어든다.
   3. 게임 중간 랜덤하게 아이템(체력약)이 생성되며 1명이 얻으면 사라진다.
   4. 체력이 모두 줄어들면 사망하고 10초후 다시 살아난다. 10초안에 팀원도 사망하면 패배하고 게임 종료
2. **High-level design**

* **게임 전체 플로우차트**

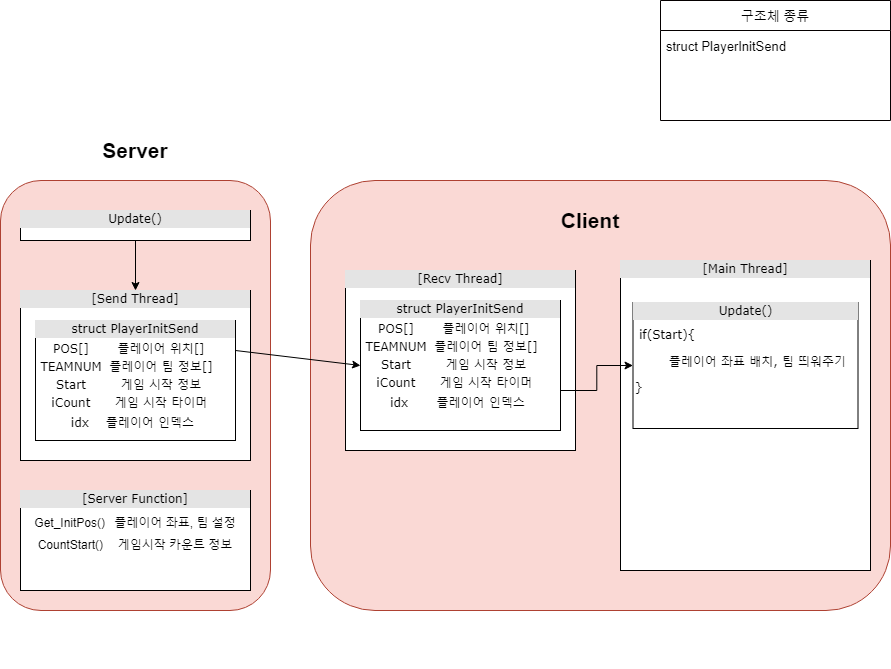


* 1. **대기실**

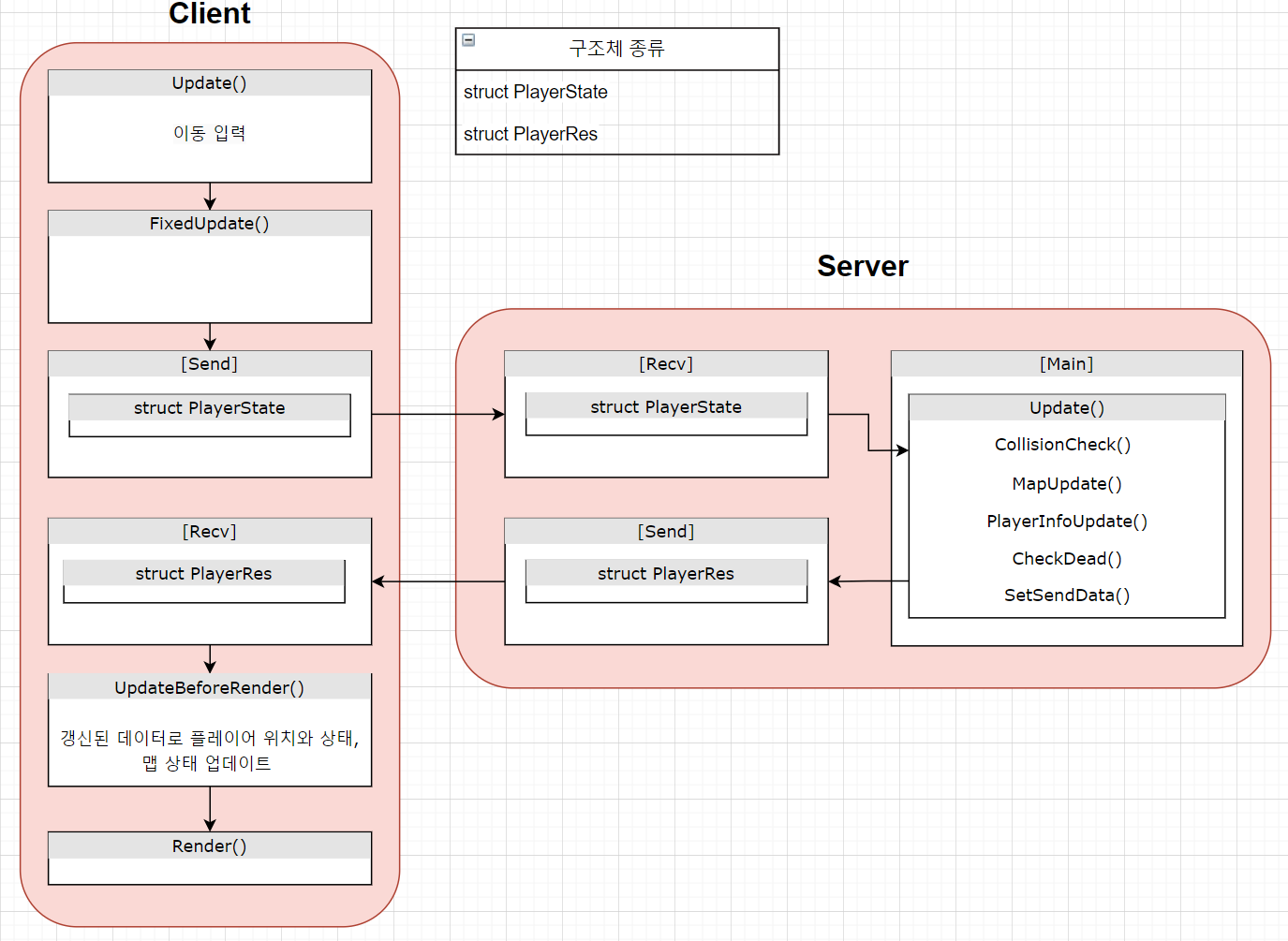


1. FixedUpdate() 호출 후 PlayerInfo 구조체를 서버로 send한다. 넘겨주는 구조체에 있는 정보는 식별 id, 플레이어 이름, 플레이어 카운트, 객체 현재 위치, 키보드 입력 값이다.
2. 서버는 이 정보를 받아서 Update 함수로 넘기고 객체의 타입을 확인하여 어떤 임무를 해야 하는 객체인지 확인하여 키보드 입력 값에 따라 이를 처리한다.
3. 처리된 결과값을 Client에게 send하고 Client는 이를 Recv한다. 받은 정보를 토대로 객체의 위치를 재설정 후 렌더링을 한다.
4. 4명의 플레이어가 로비에 연결이 되면 Initialize()에서 Client\_id와 PlayerName을 제외한 플레이어 정보 초기화.
5. PlayerSetting()에서 팀을 나누고 Client\_id, PlayerName을 클라이언트로 전달

(추가)



1. PlayerInitSend 구조체를 Sever에서 Client로 send한다. 넘겨주는 구조체에 있는 정보는 플레이어들의 위치, 팀, 게임시작 정보, 게임시작 타이머, 플레이어 인덱스이다.
2. CountStart()에서 설정한 시간이 지나면 Start 정보를 true로 바꾼다.
3. Server의 Start가 true인 경우Get\_InitPos()에서 플레이어 인덱스에 따라서 팀과 좌표를 설정한다.
4. 클라이언트는 정보를 받아서 게임시작시 플레이어 좌표 배치, 팀 정보 띄워주기 등으로 사용한다.
   1. **인게임 – 전투**



1. 클라이언트는 Update()에서 받은 이동 입력 값을 저장해 놓는다.

->Update()에서 m\_tPlayerInfo구조체에 담긴 정보들(위치, 체력, 이미지 프레임)을 업데이트 한다.

1. 클라이언트는 구조체 PlayerState의 데이터를 서버에 Send 한다.

->Render()가 끝나고 SendRecvPlayerInfo()에서 자신의 플레이어 정보를 send하고, 서버로부터 모든 플레이어의 정보를 recv한다

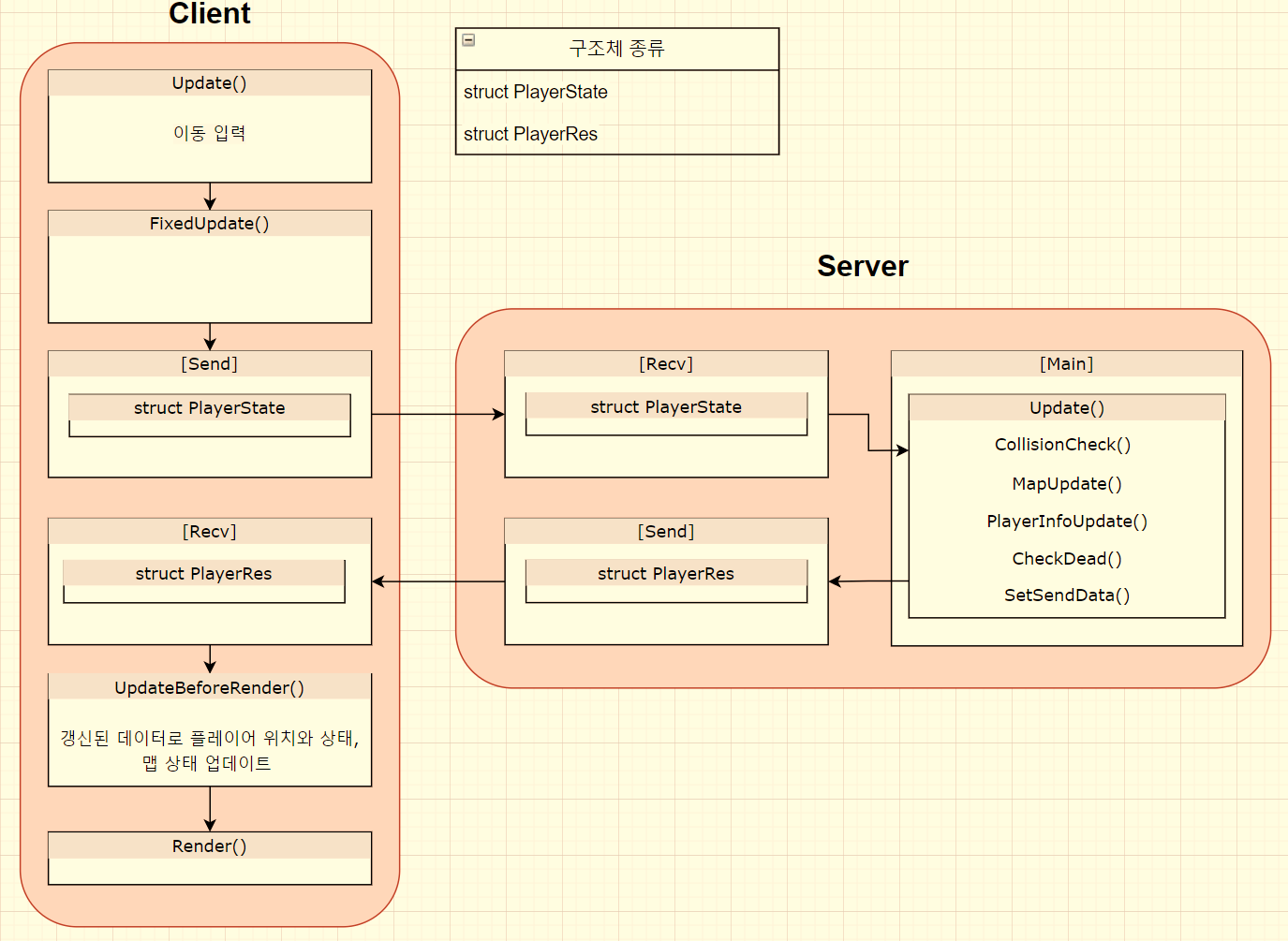
1. 서버는 전달받을 데이터를 Recv 후, 각 플레이어들의 좌표와 상태를 이용해 맵의 아이템과 충돌 체크, 부활 여부, 맵 통과 여부를 서버의 메인 스레드에서 확인 후 처리한다.

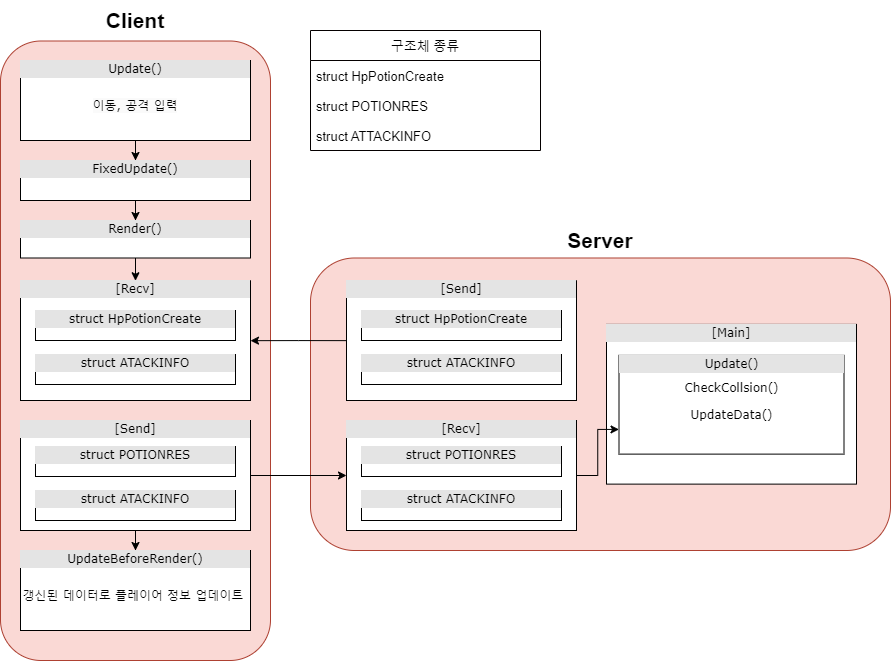
->서버에서 CollisionCheck()에서 스킬들 좌표와 플레이어 좌표로 충돌체크 후 충돌을 했는지에 대한 여부를 판단한다.

1. 모든 처리가 완료된 후 PlayerRes 구조체에 서버에서 변화된 데이터를 담고, 이 구조체를 클라이언트로 Send 한다.

-> 서버에서 각 플레이어 정보를 tPlayersInfo[4]에 넣고, 클라이언트로 Send한다.

* 1. **인게임 – ~~맵, 아이템~~, 체력약, 공격스킬**





1. 클라이언트는 Update()에서 받은 이동 입력 값을 저장해 놓는다.
2. ~~클라이언트는 구조체 PlayerState의 데이터를 서버에 Send 한다.~~서버는 구조체 HpPotionCreate, ATTACKINFO를 클라이언트에 Send한다.
3. ~~서버는 전달받을 데이터를 Recv 후, 각 플레이어들의 좌표와 상태를 이용해 맵의 아이템과 충돌 체크, 부활 여부, 맵 통과 여부를 서버의 메인 스레드에서 확인 후 처리한다.~~  
   클라이언트는 전달받을 데이터를 Recv 후, 체력약 정보 저장 및 공격 정보를 저장한다.
4. ~~모든 처리가 완료된 후 PlayerRes 구조체에 서버에서 변화된 데이터를 담고, 이 구조체를 클라이언트로 Send 한다.~~클라이언트에서 공격의 생성, 체력약과의 충돌체크를 하고 이를 각각 구조체 POTIONRES, ATTACKINFO로 서버에 Send한다.
5. ~~클라이언트는 서버에서 보낸 PlayerRes 구조체의 데이터를 Recv 후, UpdateBeforeRender 함수에서 받은 데이터를 통해 플레이어, 맵의 정보를 업데이트 한다.~~서버는 클라이언트에서 Recv한 구조체들로 저장된 체력약, 공격 정보를 갱신한다.
6. ~~모든 데이터가 갱신되면 Render한다.~~
7. **low-level design**
   1. **대기실**
      1. **Client send() -> Server recv()**

|  |  |  |
| --- | --- | --- |
| **PlayerInfo 구조체** | | |
| 인자 | 변수명 | 설명 |
| int | Client\_id | 식별 id |
| int | iPlayerCnt | 플레이어 수 |
| string | PlayerName | 플레이어 이름 |
| float2 | pos | 플레이어 위치 |
| float | fkeyInput | 키보드 입력값 |
| int | iHp | 플레이어 체력 |

1. 클라이언트에서 서버로 PlayerInfo 구조체를 전달한다. 구조체의 정보로 Update()에 있는 함수를 실행한다.
   * 1. **Server send() -> Client recv()**

|  |  |  |
| --- | --- | --- |
| **PlayerRes 구조체** | | |
| 인자 | 변수명 | 설명 |
| int[] | Client\_id | 식별 id |
| TEAM\_TYPE | TEAM1, TEAM2 | 플레이어 팀 구분 타입 |
| string[] | PlayerName | 플레이어 이름 |
| float2[] | pos | 플레이어 위치(조정된) |
| int[] | iHp | 플레이어 체력 |
| bool | bStartGame | 게임 시작 여부 |

* + - 1. Login
* 서버: PlayerData(): 접속한 클라이언트의 주소를 받아 식별id 생성.
  + - 1. Lobby
* ClientThread(): 서버가 클라이언트의 요청을 recv후 send
* CreateLobby(): 로비 생성함수
* PlayerWait(): 4명의 플레이어가 서버에 준비되었다고 전달 후 자동으로 게임 시작.
* Initalize(): 플레이어의 객체 타입, 팀, 좌표, 체력 등 플레이어에 대한 정보 초기화
* PlayerSetting(): 플레이어의 객체 타입, 팀 등 설정
* 멀티스레드 동기화
  + - 1. 각 클라이언트가 접속하는 순서대로 이벤트를 생성시 4개의 이벤트 생성
      2. 생성된 순서대로 스레드 함수가 실행되고, recv(), send()함수가 호출된다.
      3. 플레이어 팀에 따라 2개의 스레드는 레드 팀 스레드 함수가 실행되고, 다른 2개의 스레드는 블루 팀 스레드 함수가 실행된다.

(추가)

* + 1. **Server send() -> Client recv()**

|  |  |  |
| --- | --- | --- |
| **PlayerInitSend 구조체** | | |
| 인자 | 변수명 | 설명 |
| Float2 | tPos[] | 플레이어 위치 |
| TEAMNUM | TEAM[] | 플레이어 팀 구분 타입 |
| bool | Start | 게임시작 했는지 정보 |
| int | Count | 게임시작 카운트 정보 |
| int | idx | 플레이어 인덱스 |

* + - 1. Server
         * void Get\_InitPos(int idx, PLAYER\_INIT\_SEND& tPlayerInitSend): 플레이어의 좌표, 팀을 플레이어 인덱스에 따라서 설정해주는 함수이다.
         * void CountStart(): 설정한 카운트가 되었을 때 PlayerInitSend의 Start 정보를 true로 바꿔준다.
  1. **인게임 – 전투**
     1. **Client Send**

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 각 클라이언트는 SKILL\_TYPE, PLAYER\_STATE, TEAM\_TYPE 정보를 가진다.
2. PlayerUpdate()는 플레이어 좌표와, 플레이어 상태를 변경한다
3. PlayerInfo는 현재 좌표, 플레이어 상태, 팀 타입을 저장한다.

->싱글톤 클래스인 CDataMgr에 PLAYER\_INFO구조체 데이터를 저장한다

1. SendData()를 통해 서버에 구조체 전달한다.

-> SendRecvPlayerInfo()를 통해 서버에 구조체 정보를 넘기고, 모든 플레이어의 정보가 담긴 구조체인 STORE\_DATA 정보를 받는다

* + 1. **Sever Recv & Send**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 서버는 각 클라이언트들에게 전달받은 데이터를 공유 데이터에 저장한다. 이는 크기가 4인 Vector로 저장되며, 각 클라이언트 인덱스로 벡터 인덱스에 접근한다.

-> g\_tStoreData.tPlayersInfo[4] 배열에 각 클라인트 정보를 담는다.

1. StoreData 구조체는 플레이어들의 좌표, 플레이어들의 스킬 좌표들, 플레이어들의 상태를 저장한다.
2. CheckCollision() 함수를 통해 저장된 공유 데이터로 충돌 연산을 진행하고, 충돌 여부를 저장한다.
3. SendData() 함수를 통해 충돌 여부와 공유 데이터를 담은 ResultInfo 구조체를 Send 한다.

-> ResultInfo구조체가 아님 STORE\_DATA 타입의 g\_tStoreData 구조체를 보낸다.

* + 1. **Client Recv & Render**

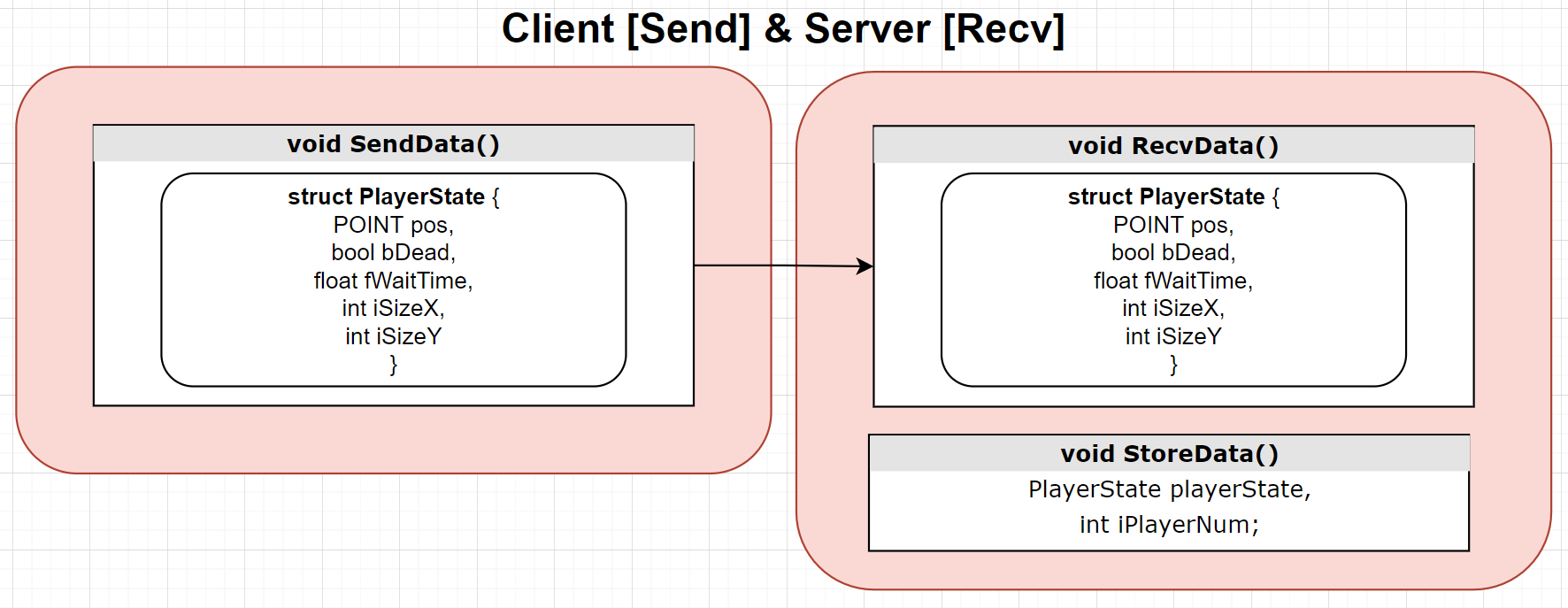
1. 서버로부터 전달받은 데이터를 ResultInfo에 저장한다.

-> CDataMgr의 m\_tStoreData에 저장한다

1. Render() 함수에서는 ResultInfo에 저장된 플레이어, 스킬들의 좌표를 이용해 자신과 다른 클라이언트들을 포함하여 모두 그린다.

->Recv -> Render 순서가 아닌, Render -> Recv로 변경했다

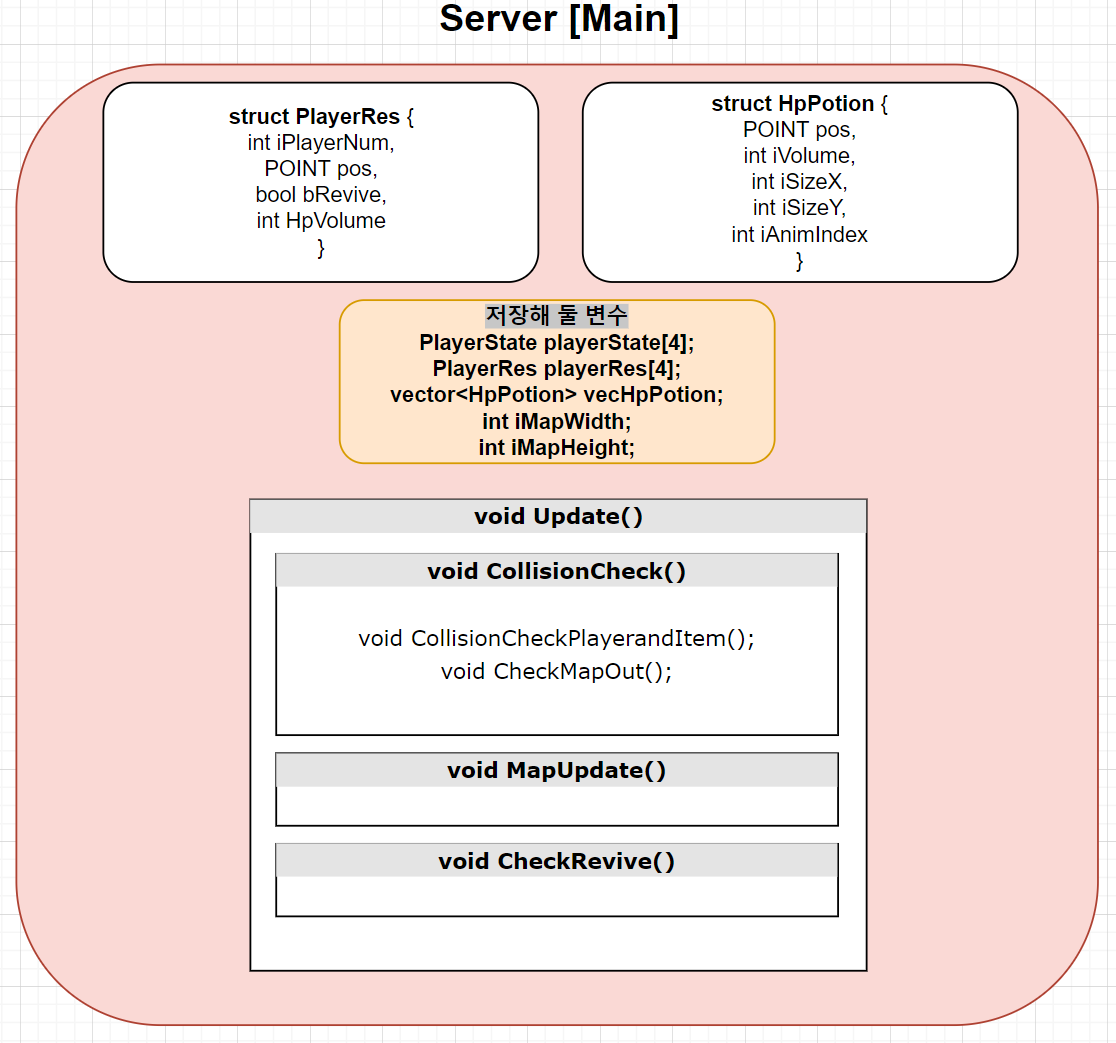
* 1. **인게임 – ~~맵, 아이템,~~ 체력약, 공격스킬**
  2. **~~Client – [Send] & Server – [Recv]~~**

~~~~

1. ~~클라이언트에서 Send할때는 구조체 PlayerState를 보낸다.~~

|  |  |  |
| --- | --- | --- |
| **~~PlayerState 구조체~~** | | |
| ~~인자~~ | ~~변수명~~ | ~~설명~~ |
| ~~POINT~~ | ~~pos~~ | ~~플레이어의 위치~~ |
| ~~bool~~ | ~~bDead~~ | ~~현재 죽어 있는지, 살아있는지~~ |
| ~~float~~ | ~~fWaitTime~~ | ~~부활을 위한 대기 타임 -> 10초가 넘고, 팀원이 살아있으면 부활.~~ |
| ~~int~~ | ~~iSizeX, iSizeY~~ | ~~충돌체를 만들기위한 플레이어 가로, 세로 사이즈~~ |

1. ~~서버는 받은 구조체 PlayerState를 플레이어가 몇 번째 클라이언트인지 확인하여 StoreDate() 함수로 넘긴다.~~
2. ~~StoreData() 함수에서는 서버의 전역 변수인 4명의 클라이언트의 PlayerState를 저장하는 배열에 iPlayerNum 인덱스에 받은 구조체 PlayerState 데이터를 넣는다.~~
   1. ~~Server – [Main]~~

~~~~

1. ~~서버는 구조체 PlayerRes 형태로 클라이언트에서 받은 데이터를 갱신한 정보를 저장해 두어 클라이언트에 Send한다.~~

|  |  |  |
| --- | --- | --- |
| **~~PlayerRes 구조체~~** | | |
| ~~인자~~ | ~~변수명~~ | ~~설명~~ |
| ~~int~~ | ~~iPlayerNum~~ | ~~4개의 클라이언트 중 몇 번째 클라이언트 인지~~ |
| ~~POINT~~ | ~~pos~~ | ~~갱신된 플레이어 위치~~ |
| ~~bool~~ | ~~bRevive~~ | ~~부활 여부~~ |
| ~~int~~ | ~~iHpVolum~~ | ~~체력약을 먹었다면 0보다 큰 값(체력 상승 값), 먹지 않았다면 0~~ |

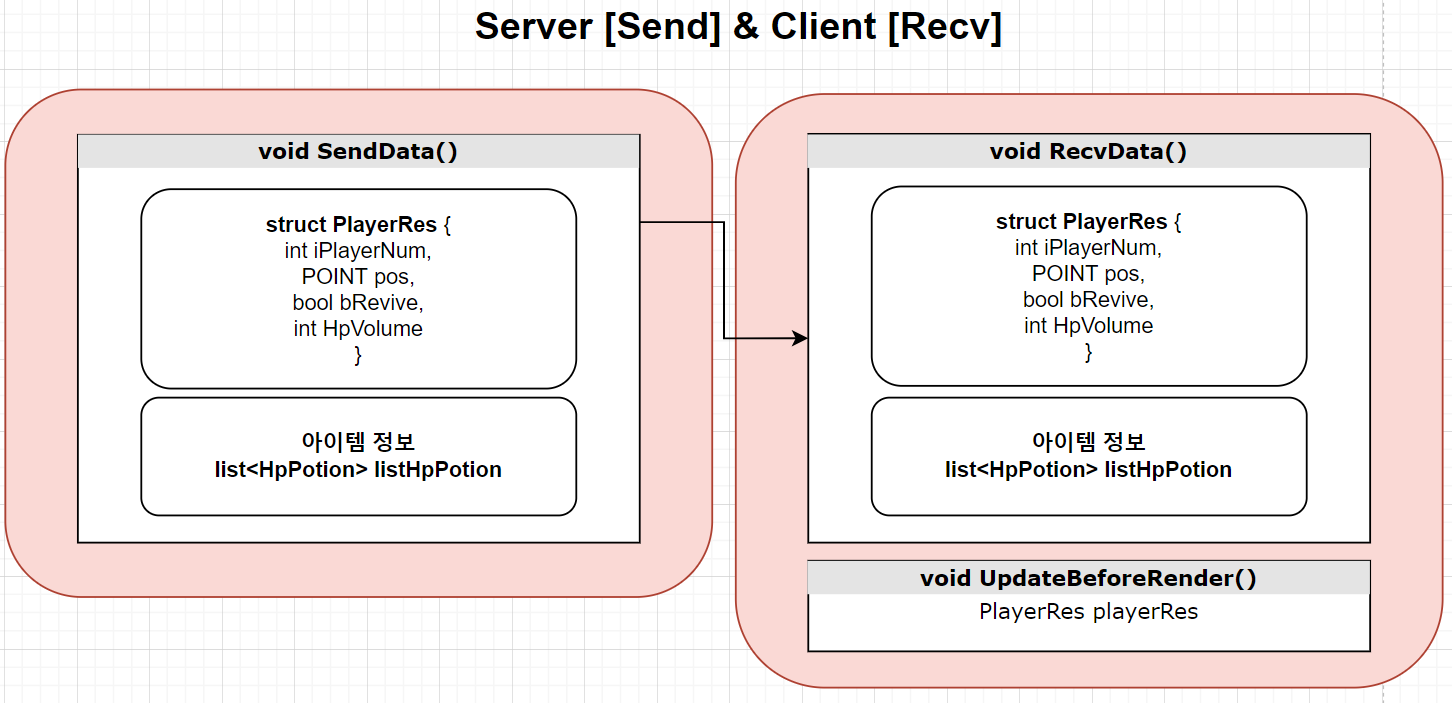
1. ~~서버에서 일정시간마다 HpPotion(체력약)을 맵에 생성한다.~~

|  |  |  |
| --- | --- | --- |
| **~~HpPotion 구조체~~** | | |
| ~~인자~~ | ~~변수명~~ | ~~설명~~ |
| ~~POINT~~ | ~~pos~~ | ~~위치~~ |
| ~~int~~ | ~~iVolume~~ | ~~용량~~ |
| ~~int~~ | ~~iSizeX, iSizeY~~ | ~~충돌체를 만들기위한 플레이어 가로, 세로 사이즈~~ |
| ~~int~~ | ~~iAnimIndex~~ | ~~애니메이션 순서~~ |

1. ~~서버에서 전역에 저장해 둘 변수 설명~~

|  |  |  |
| --- | --- | --- |
| ~~인자~~ | ~~변수명~~ | ~~설명~~ |
| ~~PlayerState~~ | ~~playerState[4]~~ | ~~클라이언트에서 받아온 데이터를 저장해논 배열~~ |
| ~~PlayerRes~~ | ~~playerRes[4]~~ | ~~클라이언트로 보낼 데이터를 저장해논 배열~~ |
| ~~list<HpPotion>~~ | ~~listHpPotion~~ | ~~맵에 존재하는 체력약의 리스트~~ |
| ~~int~~ | ~~iMapWidth, iMapHeight~~ | ~~맵의 가로, 세로 크기~~ |

1. ~~CollsionCheck() 함수에서는 CollisionCheckPlayerandItem(), CheckMapOut()을 호출하여 각각 플레이어와 아이템 충돌 처리(충돌시 맵에서 삭제), 플레이어가 맵을 벗어날 경우 위치 재조정을 한다. 이때 전역변수를 사용하여 처리하므로 인자는 필요 없다.~~
2. ~~MapUpdate() 함수에서는 체력약 생성등의 맵에 영향을 주는 일을 하여 맵의 상태를 갱신한다.~~
3. ~~CheckRevive() 함수에서는 PlayerState 구조체 데이터를 통해 죽은 플레이어의 경우 대기시간이 10초 이상이면 부활하도록 한다. 부활 정보는 PlayerRes에 갱신된다.~~
   1. ~~Server – [Send] & Client – [Recv]~~

~~~~

1. ~~서버에서 갱신된 정보를 담은 PlayerRes 구조체를 연결된 클라이언트의 정보를 구조체의 첫번째 변수인 iPlayerNum으로 확인해서 그 클라이언트에 맞는 데이터를 Send 한다.~~
2. ~~아이템의 정보를 보내는데 이때 리스트를 보낼 수는 없으니, 리스트의 크기를 먼저 고정 길이로 보내고, 데이터를 가변 길이로 보낸다. 이때 HpPotion 구조체를 보낸다.~~
3. ~~클라이언트에서는 데이터를 받으면 UpdateBeforeRender()함수에 인자로 넘겨서 렌더링 전 갱신 작업을 수행한다. 갱신 작업에는 위치 재조정, 부활했을 경우 대기시간 초기화 및 부활, 체력약을 먹었을 경우 용량만큼 체력 상승 등이 있다.~~

(변경)

* + 1. **체력약 생성 및 Send & Recv**
* 사용할 구조체

|  |  |  |
| --- | --- | --- |
| HpPotionCreate 구**조체** | | |
| 인자 | 변수명 | 설명 |
| int | cnt | 현재 몇 개의 클라이언트에 Send 했는 지 |
| int | index | 저장된 체력약 중 몇 번째 인덱스인지 |
| int | bCreateOn | 현재 체력약이 생성되었는지 |
| POS | pos | 체력약을 생성할 위치 |

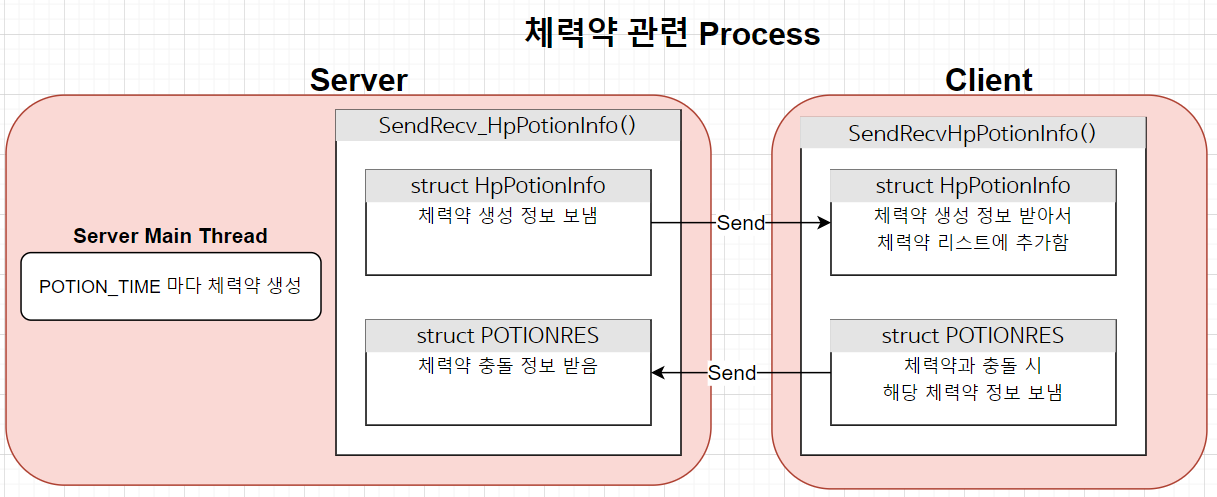
|  |  |  |
| --- | --- | --- |
| HpPotionDelete 구**조체** | | |
| 인자 | 변수명 | 설명 |
| int | cnt | 현재 몇 개의 클라이언트에 Send 했는 지 |
| int | index | 저장된 체력약 중 몇 번째 인덱스인지 |
| int | bDeleteOn | 현재 체력약이 삭제되었는지 |

|  |  |  |
| --- | --- | --- |
| HpPotionInfo 구**조체** | | |
| 인자 | 변수명 | 설명 |
| HpPotionCreate | thpPotionCreate | 생성 정보 |
| HpPotionCreate | thpPotionDelete | 삭제 정보 |

|  |  |  |
| --- | --- | --- |
| POTIONRES 구**조체** | | |
| 인자 | 변수명 | 설명 |
| bool | bCollision | 충돌 여부 |
| int | iIndex | 저장된 체력약 중 몇 번째 인덱스인지 |

* Server에서 전역에 둘 체력약 관련 변수

|  |  |  |
| --- | --- | --- |
| 인자 | 변수명 | 설명 |
| HpPotionInfo | g\_tHpPotionInfo | 체력약 삭제, 생성 정보 저장 |
| float | fPotionCreateTime | 체력약 생성 관련 시간 변수 |
| LONG | iHpPotionIndex | 체력약 생성시 인덱스 값을 무엇으로 할지 |



* + - 1. ServerMain 스레드에서 체력약 생성
* CreateHpPotion() 함수에서 체력약 생성 시간(POTION\_TIME) 마다 체력약 생성하여 g\_tHpPotionInfo에 저장
  + - 1. 서버에서 체력약 생성 정보 Send & 클라에서 Recv
* 서버에서 체력약 생성 정보를 연결된 모든 클라에게 보내고, 체력약이 생성되었다면 클라에서 체력약 리스트에 추가하고 생성되지 않았다면 추가되지 않는다.
* 서버에서 생성된 체력약의 개수 변수가 증가된다.
  + - 1. 클라에서 체력약 삭제 정보 Send & 서버에서 Recv
* 클라에서 체력약과 충돌 시 해당 체력약의 정보를 서버로 보내고, 서버에서는 이를 받아 다른 클라에게 삭제 정보를 보내 모든 클라에서 해당 체력약이 삭제되도록 한다.
  + 1. **공격스킬 정보 관련 Recv & Send**
       - * 사용할 구조체

|  |  |  |
| --- | --- | --- |
| ATTACKINFO구**조체** | | |
| 인자 | 변수명 | 설명 |
| size\_t | iType | 공격 타입이 무엇인지(근접, 불, 얼음, 블라스트) |
| INFO | tInfo | 좌표, 크기 정보 |
| int | iFrameScene | Reder를 위한 프레임 관련 변수 |
| int | iFrameStart | Reder를 위한 프레임 관련 변수 |
| bool | bCollision | 충돌 여부 |

|  |  |  |
| --- | --- | --- |
| AttackData구**조체** | | |
| 인자 | 변수명 | 설명 |
| int | iSize | 공격 배열의 크기 |
| ATTACKINFO\* | pAttackInfo | 공격 배열 (동적 배열) |

* + - * + Server에서 전역에 둘 공격스킬 관련 변수

|  |  |  |
| --- | --- | --- |
| 인자 | 변수명 | 설명 |
| AttackData | g\_pAttackData[4] | 4명의 클라의 공격스킬을 담을 변수 |

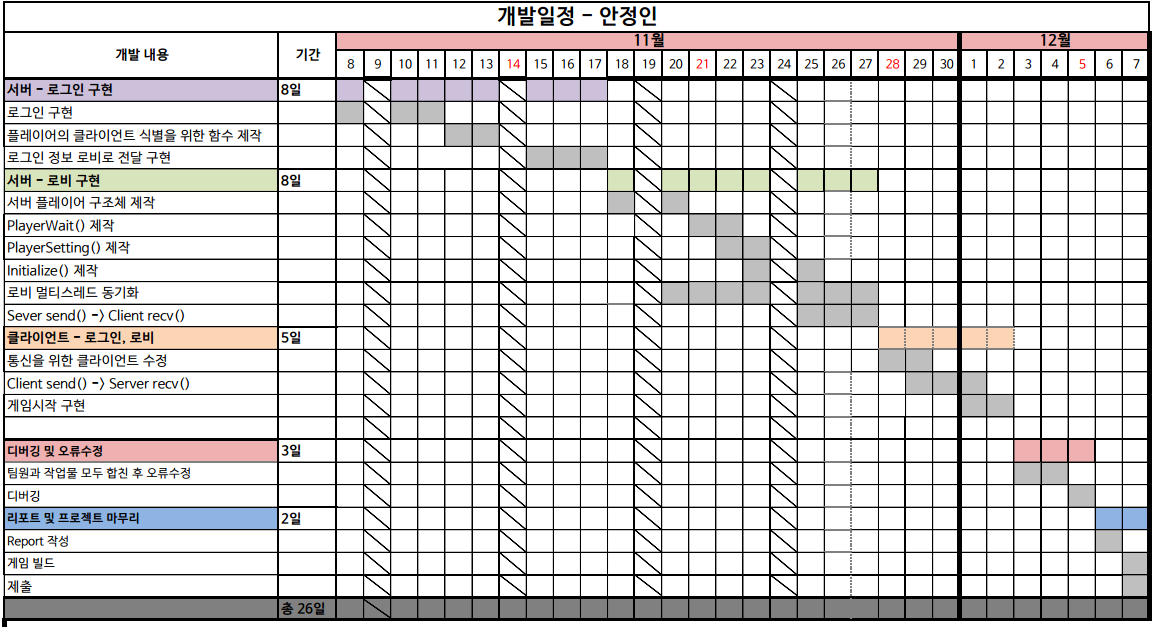
1. 클라에서 공격 정보 Send & 서버에서 Recv

* 클라에서 먼저 공격 스킬 리스트의 크기를 보내고, 크기가 0보다 크면 동적배열에 공격 스킬을 담고 그 배열을 보낸다. 서버에서는 이를 받아 g\_pAttackData변수에 넣는다. 해당 클라이언트의 번호에 맞는 배열의 인덱스를 지정한다.

1. 충돌 체크 후 서버에서 연결된 모든 클라의 공격 정보 Send & 클라에서 Recv

* 서버에서 클라로 모든 클라(4명)의 공격 스킬을 보내는데, 위와 같이 먼저 배열의 크기를 보내고 배열의 크기가 0보다 크면 공격 스킬 정보를 담은 배열을 보낸다. 클라는 이를 받아 저장하고 렌더링할 때 사용한다.
  + 1. **플레이어 부활 처리 & 게임 종료 구현**
       1. 플레이어 부활 처리
* 플레이어가 죽으면 isDead 변수가 false로 바뀌고, 10초 동안 대기한다. 이후 10초 후에 팀원이 살아있으면 isDead 변수를 false로 바꾸고 다시 살아난다.
  + - 1. 승리 판정 및 게임 종료
* 팀원이 죽은 상태에서 같은 팀원이 죽으면 해당 팀은 패배하고 상대 팀은 승리한다. 이는 플레이어의 정보를 주고받는 함수인 SendRecv\_PlayerInfo()에서 판정한다.
  + 1. 멀티스레드 동기화
* 체력약 생성하는 스레드는 동기화가 되어 있지 않아 공유자원 접근 시 오류 발생 위험이 있으므로 임계영역 기법을 통해 체력약 정보를 저장하는 전역 변수인 g\_tHpPotionInfo 접근 시 임계영역을 사용해 한 번에 한 스레드만 접근하도록 한다. 이때 접근 순서는 상관없으므로 이벤트는 사용하지 않아도 된다.

1. **팀원 별 역할분담** 
   1. **안정인**
      1. 대기실 입장 구현
      2. 플레이어 배치좌표, 팀 설정, 게임시작 카운트
   2. **윤성주**
2. 클라이언트 초기 개발 – 아이템 배치
3. ~~인게임의 맵, 아이템 구현~~인게임의 체력약, 공격스킬, 부활 구현
4. 문서정리
   1. **최경훈**
      1. 클라이언트 초기 개발 – 플레이어
      2. 서버, 클라이언트 동기화 구현
      3. 플레이어 정보 Send & Recv
      4. 충돌 체크
5. **개발환경**
   1. **Visual Studio 2019**
   2. **GitHub**
   3. **WINAPI**
6. **개발일정** 
   1. **11/8 ~ 12/7 동안 진행**
   2. **안정인 – 총 26일**
   3. **윤성주 – 총 27일**
   4. **최경훈 – 총 26일**
7. 테이블이(가) 표시된 사진

   자동 생성된 설명**일정표**

