네트워크 게임프로그래밍 추진 계획서

2023.10.30

박유민 2017182013 최현호 2019180041 조혜연 2020180038

2023

**목 차**

1. **애플리케이션 기획**
2. 게임 소개 ( 2P )
3. 게임의 구성요소 ( 2 ~ 4P )
4. 개발 환경 ( 4P )
5. **High Level Design**
6. 전체 흐름도 ( 5P )
7. 로그인 흐름도 ( 6, 7P )
8. 인게임 흐름도 ( 8, 9P )
9. **Low Level Design**
10. 전역 상수 ( 10, 11P )
11. Client ( 12 , 13P )
12. Server ( 14P ~ 16P )
13. **팀원별 역할분담 ( 17p )**
14. **개발 일정 ( 18 ~ 20p )**

**1. 애플리케이션 기획**

**1) 게임 소개**

● 제목 : 프렌드런

● 게임장르 : 2D 횡스크롤 러닝액션게임

● 게임 방식 : 2명의 플레이어가 함께 경쟁하는 방식으로 진행됩니다.

**2) 게임의 구성요소**



[그림 1] 게임의 실제 실행화면

1. **Objects**

 기본 젤리 : 쿠키가 획득하면 점수가 올라간다. 코인을 통해 업그레이드하면 획득하는 점수의 값이 늘어납니다.

 강화 젤리 : 강화된 젤리로 기본 젤리에 비해 높은 점수를 획득가능.

 특수 젤리 : 일부 쿠키들의 능력 발동을 위한 조건으로 사용되는 젤리입니다. Ex) 실행화면의 는 능력 발동 조건 확인을 위한 숫자입니다.

 장애물 : 쿠기가 장애물과 충돌할 시 쿠키의 체력이 감소하고, 일정 시간동안 느려집니다.

 스코어보드 : 획득한 점수와 코인을 보여줍니다.

 미니맵 : 스테이지의 진행도를 보여줍니다

.

 체력바 : 쿠키의 현재 체력을 보여줍니다.

 플레이어 : 게임을 진행하는 플레이어입니다.

 펫 : 플레이어를 따라다닙니다.

 코인 : 맵에 배치되어 있는 코인입니다. 쿠키 구매, 젤리 강화, 체력증가 등 게임 요소 강화에 사용됩니다.

 거대화 : 쿠키가 일정 시간 거대해지며 장애물에 부딪힐 경우 체력이 감소하는 대신 장애물을 파괴합니다.

 자석 : 코인, 젤리와 같은 아이템들을 펫이 대신해서 먹어줍니다

.

 회복 포션 : 쿠키의 체력을 일정량 회복시켜줍니다.

**2. 조작법**

SPACE – 점프(JUMP)

아래 방향키 – 슬라이드(SLIDE)

P – 게임 시작 (준비 화면 & 죽었을 때), 일시 정지(게임 진행 중)

* 개발 과정에서 마우스 버튼 조작(게임 시작)으로 대체할 예정입니다.

마우스 – 마우스로 일시정지, 게임 시작, 준비와 같은 버튼을 눌러 조작.

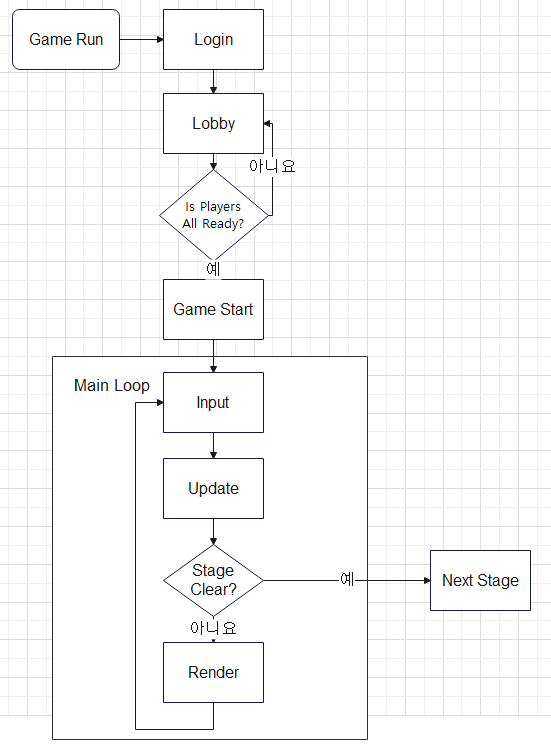
**3) 개발환경**

박유민 학우의 윈도우프로그래밍 프로젝트에 네트워크 기능을 추가하여서

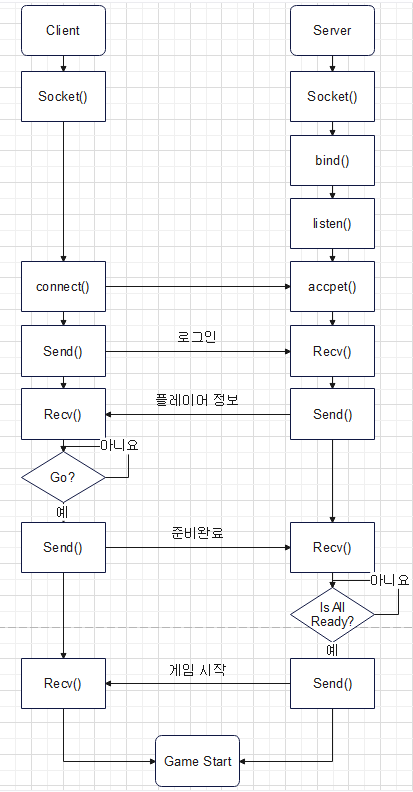
멀티플레이가 가능한 게임으로 만들 예정입니다. Visual Studio 2022를 이용해서 C++ 언어 기반으로 개발을 진행합니다. 개발 과정은 Github Desktop을 이용하여 팀원들이 함께 관리할 예정입니다.

**2. High Level Design**

1. **전체 흐름도**

****

1. **로그인 흐름도**

****

**클라이언트 흐름**

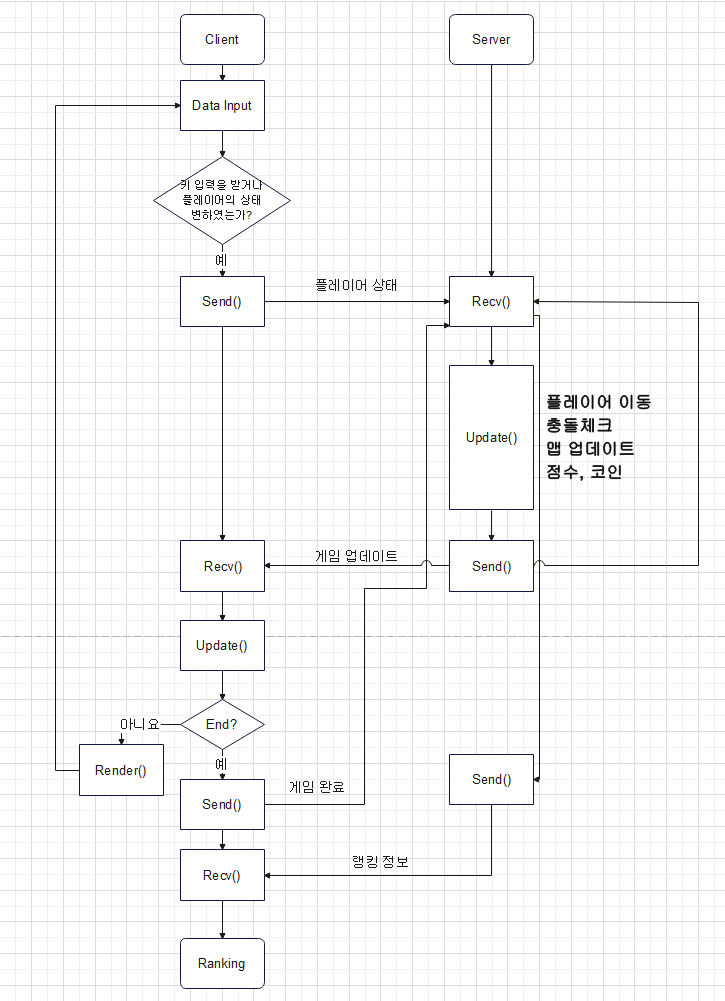
1. 클라이언트 실행 후 소켓 생성, 서버에 connect 요청
2. 서버에 로그인 정보 입력
3. 서버로부터 플레이어 정보를 받는다.
4. 서버에 준비완료 정보를 보낸다.
5. 다른 클라이언트들이 준비가 완료될 떄까지 기다린다.
6. 게임시작

**서버 흐름**

1. 서버 실행 후 소켓 생성, 바인드를 한 후에 대기 상태에서 클라이언트의 connect 요청을 accept 해준다.
2. 클라이언트로부터 로그인 정보를 받는다.
3. 클라이언트에 플레이어 정보를 전달해준다.
4. 클라이언트로부터 준비 여부에 대한 정보를 받는다.
5. 다른 클라이언트들의 준비 여부를 받아서 모두 준비되면

게임을 시작할 정보를 보내준다.

**3) 인게임 흐름도**



**클라이언트 흐름**

1. 클라이언트에서 마우스, 키보드와 같은 조작이 입력되면 정보를 보냄.
2. 서버에서 보낸 업데이트 정보를 받는다.
3. 정보를 업데이트한다.
4. 게임이 완료될 때까지 렌더링을 계속하고 완료되면 게임 완료 정보(점수, 코인)를 서버로 보내준다.
5. 서버에서 보내는 랭킹 정보를 받는다.
6. 현재 랭킹을 보여준다.

**서버 흐름**

1. 클라이언트가 보내는 정보를 받는다.
2. 플레이어의 움직임, 맵 정보, 충돌체크 여부, 점수, 코인에 대한 정보를 업데이트 해준다.
3. 클라이언트에 업데이트된 정보를 보내준다. 1 ~ 3 과정을 반복한다.
4. 게임이 완료되면 랭킹 정보를 보내준다.

**3. Low Level Design**

**1) 전역 상수**

**1. GLOBAL DEFINED**

- NAME\_LEN : 닉네임 길이

**-** BUF\_SIZE : 서버-클라간 정보 송수신시 사용할 배열의 크기

- SERVER\_PORT : 서버 포트 번호

- MAX\_FRAME : 최대 프레임

- CLIENT\_NUMBER : 서버에 연결할 클라이언트의 수(게임 플레이어 수)

- SCENE\_NUMBER : 스테이지 진행에 따른 Scene의 변화

- JUMP\_SPEED : 점프 속도

- GRAVITY : 중력값. 점프에 영향

**2. ENUM CLASS**

**1) OBJID (아이템 종류)**

- OBJ\_BACKGROUND : 배경을 그려주는 오브젝트

- OBJ\_ITEM\_HEAL : 체력 회복 아이템

- OBJ\_DEVILJELLY : 악마맛 쿠키 능력을 위한 특수젤리

- OBJ\_ITEM\_GIGA : 거대화 아이템 1

- OBJ\_ITEM\_GIGA2 : 거대화 아이템 2

- OBJ\_DEVILCOIN : 날아다니는 특수 코인

- OBJ\_PET : 펫

- OBJ\_FLYING : 왼쪽에서 날아와서 등장하는 오브젝트

- OBJ\_PLAYER : 플레이어

**2) DIRECTION(방향**)

- DIR\_LEFT, DIR\_UP, DIR\_RIGHT, DIR\_DOWN, DIR\_END : 상하좌우 방향

**3) SCENE\_ID : 현재 SCENE의 번호**

- SC\_LOGO : 로고 SCENE

- SC\_LOBBY : 로비 SCENE

- SC\_STAGE : 스테이지 SCENE

- SC\_SCORE : 점수 결과창

**4) STATE(상태**)

- IDLE : 기본 상태

- WALK : 걷기

- JUMP : 점프

- DOUBLE\_JUMP : 2단 점프

- SLIDE : 슬라이드

- DEAD : 죽음

- BOOST : 부스트(빠르게 달리는 상태)

- GIGA : 거대화(크기가 커지는 상태)

- DEVIL : 악마 쿠키의 능력(특수 상태)

- REVIVE : 좀비 쿠키의 능력(특수 상태)

**5) CHOICE(캐릭터 선택)**

- COFFEECOOKIE : 커피맛 쿠키

- DEVILCOOKIE : 악마맛 쿠키

- ZOMBIECOOKIE : 좀비맛 쿠키

- CHOICE\_END : 캐릭터 선택완료

**6) JELLY(게임 내의 젤리들)**

- BASE, NORMAL, RARE, JELLY\_END : 젤리의 강화 등급

- CHANNELID : 사운드 채널

- SOUND\_EFFECT : 사운드 효과

- SOUND\_BGM : BGM

- SOUND\_PLAYER : PLAYER 효과음

- SOUND\_COLLISION : 충돌 효과음

**2) Client**

1. MainGame

- 모든 오브젝트를 관리해주고 있음.

- FPS 계산을 진행함.

2. CSceneMgr(게임의 전체 Scene을 관리하는 곳)

Static CSceneMgr\* Get\_Instance()

- void Scene\_Change(SCENEID eID) : Scene 변환

3. CPlayer(플레이어와 관련된 정보 업데이트)

Virtual void Initialize() override : Scene의 Player 확인

* float Set\_Hp() {return m\_fHp } : 플레이어 HP
* int Get\_Playertype(){return m\_iPlayertype } : 플레이어 타입(ID)
* void Get\_PlayerHP(float\_HP) {m\_fHp = \_Hp; } : 현재 HP
* void Collision() : 플레이어 충돌 확인
* void CollisionDevil() : 특수 젤리 충돌체크용
* void Get\_Itemtype(OBJID \_Item) : 오브젝트 아이디(아이템 종류)
* int Set\_State() : 플레이어 상태 설정
* bool Set\_Endgame() { return m\_bEndgame } : 게임 완료

4. CServer

- void InitializeSocket() : 소켓 생성, 연결

- void SendData(char\* arr, int size) : 서버로 데이터 전송

- void RecvData(CScene\* scene, CPlayer\* player) : 서버로부터 받은 정보를 통한 클라이언트 업데이트

5. Collision (충돌체크)

static void Collision\_Player\_Needle(CObj& \_Obj, list<CNeedle>& m\_Needle\_List);

static void Collision\_Player\_Frog(CObj& \_Obj, list<CFrog>& m\_Needle\_List);

static void Collision\_Player\_Fork(CObj& \_Obj, list<CFork>& m\_Fork\_List);

static void Collision\_Player\_Air(CObj& \_Obj, list<CAir>& m\_Fork\_List);

static void Collision\_Player\_Jelly(CObj& \_Obj, list<CJelly>& m\_Jelly\_List);

static void Collision\_Player\_Coin(CObj& \_Obj, list<CCoin>& m\_Jelly\_List);

static void Collision\_Player\_Gold(CObj& \_Obj, list<CGold>& m\_Coin\_List);

static void Collision\_Player\_DevilCoin(CObj& \_Obj, list<CDevilCoin>& m\_Jelly\_List);

static void Collision\_Player\_Devil\_Jellies(CObj& \_Obj, list<DevilJellies>& m\_Jelly\_List);

static void Collision\_Player\_Item(CObj& \_Obj, CObj& \_Heal , OBJID eDir = OBJ\_END);

static void Collision\_Pet\_Jelly(CObj& \_Obj, list<CJelly>& m\_Jelly\_List);

static void Collision\_Pet\_Coin(CObj& \_Obj, list<CCoin>& m\_Jelly\_List);

static void Collision\_Pet\_Gold(CObj& \_Obj, list<CGold>& m\_Jelly\_List);

-> 오브젝트들과 플레이어의 충돌 확인 후 정보를 서버로 전달.

6. 공통 함수(CScene, Collision)

- void Update() : Scene 업데이트(틱마다)

- void Late\_Update() : 정보 수신 후 업데이트

- void Render(HDC hdc) : 렌더링

- void Release() : 릴리즈

- void Key\_Input() : 키 입력

**3) Server**

1. 플레이어

- int Get\_Hp() {return m\_iHp} : 현재 체력 가져오기

- void Set\_Hp(int \_iHp) { m\_iHp = \_iHp; } : 현재 체력으로 설정

- int Get\_MaxHp() {return m\_iMaxHp} : 최대 체력 가져오기

- void Set\_MaxHp(int \_iMaxHp) { m\_iMaxHp = \_iMaxHp; } : 최대 체력 설정

- int Get\_Coin() {return m\_i Coin } : 획득한 코인 개수

- void Set\_ Coin (int \_iCoin) { m\_iCoin = \_iCoin; } : 개수 설정

- int Get\_Score() {return m\_iHp} : 점수값 가져오기

- void Set\_Score(int \_iScore) { m\_iScore = \_iScore; } : 점수 값을 저장

- int Get\_CurState() {return m\_iCurstate} : 플레이어 상태

- void Set\_CurState(int \_iCurState) { m\_iCurState = \_iCurstate; } : 체력값

- int Get\_Posx() {return m\_fposX; } : X 좌표 가져오기

- void Set\_Posx(float \_fposX) {m\_fposX = \_fposX; } : X 좌표 저장

- int Get\_Posy() {return m\_fposY; } : Y 좌표 가져오기

- void Set\_Posy(float \_fposY) {m\_fposY= \_fposY; } : Y 좌표 저장

2. PACKET

**Client To Server**

* CS\_LOGIN\_PACKET : 클라이언트가 로그인할 때 설정한 게임 내의 이름을 서버에게 보내주는 패킷
* CS\_PLAYER\_READY\_PACKET : 클라이언트가 로비에서 준비를 누르면 서버에게 준비상태를 보내주는 패킷
* CS\_INPUT\_PACKET : 클라이언트가 플레이어의 키버퍼나 상태가 바뀔 때 서버에게 바뀐 키버퍼의 키를 보내주는 패킷
* CS\_PLAYER\_RESET\_PACKER : 클라이언트가 죽을 때나 리셋 버튼을 누를 때 서버에게 리셋을 하라고 보내주는 패킷
* CS\_SENDPLAYERSTATE\_PACKET: 클라이언트가 서버에 자신의 상태 정보를 보내주는 패킷

**Server To Client**

* SC\_READY\_PACKET : 특정 클라이언트가 준비완료 하였다는 것을 다른 클라이언트에게 알려주는 패킷
* SC\_GAMESTART\_PACKET : 모든 클라이언트가 준비완료 했을 때 게임시작을 알리는 패킷
* SC\_UPDATE\_PACKET : 클라이언트들의 정보를 업데이트 하기 위한 패킷
* SC\_SENDPLAYERSTATE\_PACKET: 서버에서 2명의 플레이어 정보 및 상태를 보내주는 패킷

3. 함수

송신

- void sendPlayerInfoLobby() – Lobby 상태일 때 모든 클라이언트의 ID, color 정보를 클라이언트들에게 보내준다.

**-** void Set\_PlayerReady() – 플레이어 준비 상태 확인

- void Set\_Playerdata()– 플레이어 데이터 전달

~~- void sendAddBlock(float x, float z) – 모든 클라이언트에게 설치할 블록 위치를 보내준다.~~

- ~~void sendGameClear() – 게임을 클리어 했다는 메시지를 보내준다.~~

경쟁형 게임으로 변경되면서 클리어 메시지 제거됨.

~~- void sendPlayerLeave(char\* id) – 해당 id를 가진 플레이어가 게임을 종료했다는 메시지를 보내준다.~~

- void sendPauseState() – 게임 중단/재개 메시지를 보내준다.

수신

* void Recv\_ReadyData() – 준비완료 확인
* void Get\_PlayerReady() – 플레이어가 준비 상태인지 확인
* void Get\_Playerdata() - 플레이어 데이터를 받아온다.
* ~~void recvCollide() – 충돌된 정보를 받는다.~~
* ~~void recvAddBlock() – 설치할 블록 위치를 받는다.~~
* void recvPauseState() – 게임 중단/재개 메시지를 받는다.

동기화

* DWORD WINAPI ClientThread(LPVOID arg) – 클라이언트와의 통신을 위한 스레드 arg에 클라이언트 SOCKET 포인터가 들어간다.
* EnterCriticalSection(&cs)
* LeaveCriticalSection(&cs)
* 임계 영역을 사용하여 동기화. 각 임계영역마다 추가로 bool 값을 사용해서 순서를 결정해주는 방식을 사용.

양식의 맨 위

**4. 역할 분담**

|  |  |  |
| --- | --- | --- |
| **구분** | **구현 내용** | **담당** |
| **Server** | 플레이어 준비 확인 | 조혜연 |
| 스레드 함수 | 조혜연 |
| 클라이언트로 패킷 전송 | 조혜연 |
| 클라이언트에서 패킷 받기 | 조혜연 |
| 플레이어 정보 | 최현호 |
| Player Class | 박유민 |
| 상대측 정보 받기 | 박유민 |
| Player State | 최현호 |
| **Client** | 로그인 화면 | 조혜연 |
| 맵 디자인 | 조혜연 |
| 서버로부터 패킷 받기 | 최현호 |
| 서버로부터 받은 패킷 처리하기 | 최현호 |
| 서버로 패킷 전송 | 조혜연 |
| Object 클래스 | 박유민 |
| 충돌처리 버그 수정 | 박유민 |
| 게임 결과/~~랭킹 출력~~ | 조혜연 |
| 플레이어별 상태 정보 | 최현호 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11월 1주차** | | | | | | | |
|  | **월 30** | **화 31** | **수 1** | **목 2** | **금 3** | **토 4** | **일 5** |
| **유민** |  |  | **계획서 확인** | **~~충돌처리 수정~~**  ~~static void Collision\_Player~~ **11.3 계획서 재검사** | |  | **계획서 재검토** |
| **혜연** |  |  | **계획서 확인** | **~~쓰레드 구분해서~~**  **~~사용 준비~~**  **11.3 계획서 재검사** | |  |
| **현호** |  |  | **계획서 확인** | **~~클라이언트~~**  **~~구조 수정~~**  **11.3 계획서 재검사** | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11월 2주차** | | | | | | | |
|  | **월 6** | **화 7** | **수 8** | **목 9** | **금 10** | **토 11** | **일 12** |
| **유민** |  | **계획서 승인** | **계획서 피드백. 경쟁형 게임으로 수정.** | **~~Player Collision 구현~~ 경쟁 게임으로 변경하기위한 구조 수정 OppData 추가** | |  |  |
| **혜연** |  |  |  | **맵 디자인**  void sendAddBlock  void recvAddBlock | |
| **현호** |  |  | **경쟁 게임 변경을 위해 Player State 수정** | |  |
| **11월 3주차** | | | | | | | |
|  | **월 13** | **화 14** | **수 15** | **목 16** | **금 17** | **토 18** | **일 19** |
| **유민** |  | **클라이언트 상태 점검 및 서버 구현 시작** | |  | **NetworkManager.h NetworkManager.cpp 네트워크 구현** | |  |
| **혜연** |  |  | **에코 서버 Servermain.cpp 추가 Default.cpp 추가** | |  |
| **현호** |  |  | **ReceiveData()**  **SendData() 추가 서버와 데이터 송/수신** | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11월 4주차** | | | | | | | |
|  | **월 20** | **화 21** | **수 22** | **목 23** | **금 24** | **토 25** | **일 26** |
| **유민** |  | **Object Class 구현** | |  | **상대 측 정보 받기**  **Get\_oppPos() Get\_oppCoin()**  **Get\_oppscore()** | |  |
| **혜연** | **준비 상태 확인**  **Get\_oppReady()**  **Set\_oppReady()** | |  | **ProcessClient 함수() Thread를 통한 클라이언트 관리** | |  |  |
| **현호** |  | **Set\_PlayerData() 플레이어 정보** | |  | **Get\_curState() Set\_curState()** | |  |
| **11월 5주차 / 12월 1주차** | | | | | | | |
|  | **월 27** | **화 28** | **수 29** | **목 30** | **금 1** | **토 2** | **일 3** |
| **유민** |  | **Recv\_ReadyData() Send\_ReadyData()**  **플레이어 준비** | | **데이터 송수신을 하는 과정에서 클라이언트 연결에 문제가 발생함.  문제 해결을 위해 원인 파악** | **~~구현된 코드 테스트 및 수정할 내용 확인~~ 쓰레드 함수 부분에서 오류 발생 확인.**  **로컬 클라이언트로 오류 수정 후 서버 다시 테스트** | |  |
| **혜연** |  | **~~Server 로그인 구현~~**  ~~LOGIN\_PACKET~~  **Player 정보 함수 Get@, Set@** | |  |
| **현호** |  | **~~Login Scene,~~**  **~~게임 결과/랭킹 출력~~ Player 정보 함수 Get@, Set@** | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **12월 2주차** | | | | | | | |
|  | **월 4** | **화 5** | **수 6** | **목 7** | **금 8** | **토 9** | **일 10** |
| **유민** | **~~랭킹 Scene 추가, 랭킹 데이터 관리~~**  **Opp.cpp 상대방 관리** | | **텀프로젝트 제출전**  **최종 검수 및 수정**  **자잘한 오류들 발견으로 추가 기간 필요. 최종 프로젝트 제출 연기 결정** | | **~~최종~~**  **~~프로젝트~~**  **~~제출일~~ [12일**  **제출]** | **텀프로젝트 제출전**  **최종 검수 및 수정**  **프레임드랍 발견됨[12.10]** | |
| **혜연** | **Thread를 이용한 다중 클라이언트 Server 점검** | |
| **현호** | 임계 영역을 통한 동기화 설정 | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **12월 3주차** | | | | | |
|  | **월 11** | **화 12** |  |  |  |
| **유민** | **텀프로젝트 제출전**  **최종 검수 및 수정** | **최종 프로젝트 제출** |  |  |  |
| **혜연** |  |  |  |
| **현호** |  |  |  |