**네트워크 게임 프로그래밍 추진 계획서**

2016184024 신봉섭

2016184001 강진주

**앱 기획 :**

장르: 2인용 대전 슈팅 게임



* 핫 라인 마이애미 리소스 추출

**개발 환경: Direct2D**

게임 방식:

게임 핫 라인 마이애미의 리소스를 추출하여 기존의 1인 게임에서

2인 대결방식의 게임으로 변경

플레이어는 지정된 리스폰 지역에서 태어나게 되고 무기를 사게 되면 지정된 위치로 이동하게 된다 그후에 전투가 이루어지고 적의 체력을 모두 소진 시키면 게임을 승리하게된다.

플레이어 컨트롤 :

1. 이동: WASD(상하좌우 이동키)
2. 공격: Mouse Left(사격), Mouse Pos XY(조준), Key R(재장전)
3. 상호작용: Key E(무기 구입)

게임 진행

시작 준비: 클라이언트가 서버에 접속하게 되면 다른 플레이어를 기다리는 대기상태에 들어가고 두명의 플레이어가 접속하게 되면 캐릭터를 출력하면서 게임을 시작한ㄷ.

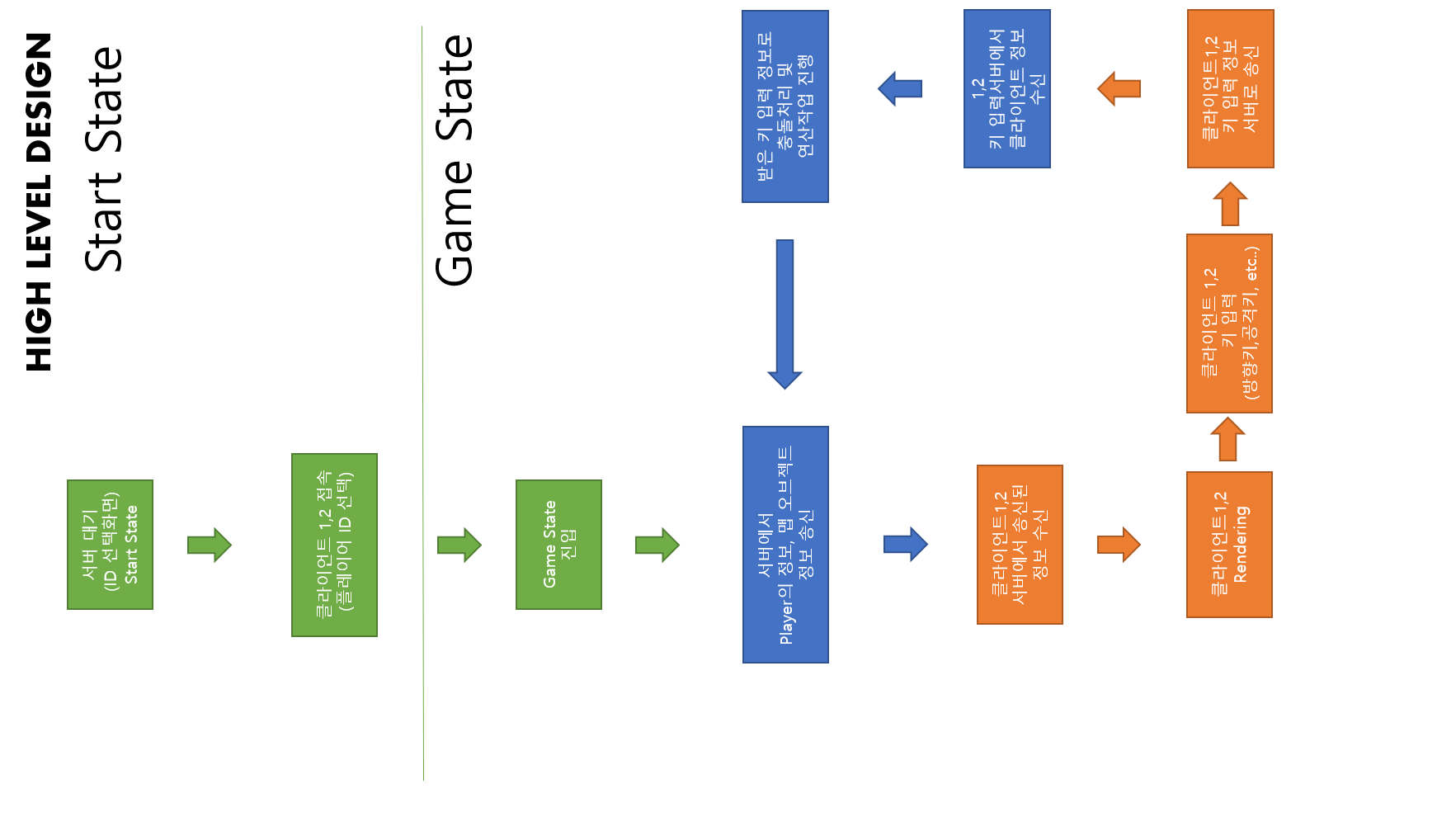
**Game State**

Game State: Game State는 총 3단계로 구성된다.

1. 게임 준비: 게임이 시작되면 각 플레이어는 각자 정해진 구역 내에서 보유한 돈으로 게임 진행에 필요한 무기를 구매할 수 있다
   1. 게임이 처음 시작하면 플레이어는 3000원을 가지고 시작한다
   2. 플레이어는 준비구역에 배치된 각 무기의 앞에 가서 상호작용 키를 누르면 소지한 금액에서 무기의 금액만큼 차감되고 무기를 구입할 수 있다.
2. 게임 진행: 게임 준비시간이 끝난 후 각 플레이어는 각자의 준비 구역에서 나와 맵을 돌아다니며 적에게 사격을 해 쓰러뜨린다
   1. 맵에는 엄폐물이 존재하고 총알은 엄폐물을 통과할 수 없다.
   2. 플레이어는 재장전 키를 누르게 되면 재장전을 즉시 할 수 있고 각 무기의 탄창을 모두 비우게 되면 자동으로 재장전을 하게 된다. 재장전을 하는 동안 총알을 발사할 수 없다.
3. 게임 종료: 두 플레이어 중에 한 명이 죽음상태에 빠지게 되면 게임이 종료된다.

.

**High Level Design**

****

1. 서버 대기(Start State): 서버는 2개의 클라이언트가 접속을 마칠 때까지 대기한다.
2. 클라이언트 1,2 접속: 클라이언트는 서버에 접속하여 다른 클라이언트를 기다린다.
3. Game State 진입: 2개의 클라이언트의 접속이 성공하면 Game State로 진입한다.
4. 서버에서 정보 송신: 서버에서 플레이어 렌더링에 필요한 정보를 송신한다. (멀티 스레드로 동시 송신)
5. 클라이언트1,2 정보 수신: 클라이언트는 서버에서 보낸 정보를 수신한다.
6. 클라이언트 1,2 렌더링: 클라이언트는 받은 정보를 기반으로 렌더링을 진행한다.
7. 클라이언트 1,2 키 입력: 클라이언트는 정해진 키(방향키, 공격키, etc..)등을 입력한다.
8. 클라이언트 1,2 키 입력 정보 송신: 클라이언트는 입력된 키의 정보를 서버로 송신한다.
9. 서버 정보 수신: 서버는 클라이언트들의 키 입력 정보를 수신한다. (멀티 스레드로 동시 수신)
10. 서버 연산 작업: 서버는 받은 키 입력 정보로 발생하는 상호작용과 충돌처리 연산작업을 실행한다. ->4번으로 이동

**Low Level Design**

클라이언트 함수

서버 함수

서버 송수신 관련 함수

**RecvThread(LPVOID arg) //리시브를 동시에 받기 위한 스레드 함수**

**CreateThread(NULL, 0, RecvThread, (LPVOID)client\_sock, 0, NULL);**

**WaitForSingleObjects (Handle, INFINITE)//클라이언트 1이 클라이언트 2를 기다리기 위해서 대기시키는 함수**

충돌처리 함수

**void CollisionRect(COLOBJ objArr[]/\*장애물 배열\*/, COLOBJ bulArr[]/\*총알구조체 배열\*/, BulletInfo bullets[])**

**장애물 과 총알 충돌**

**void CollisionRectPlayerBul(COLOBJ playerArr/\*플레이어\*/, COLOBJ bulArr[]/\*총알구조체 배열\*/, BulletInfo bullets[], PlayerInfo\* p\_pInfo)**

**플레이어와 총알 충돌**

**void CollisionRectEx(COLOBJ objArr[]/\*장애물 배열\*/, COLOBJ playerArr[]/\*플레이어 위치 배열\*/)**

**장애물과 플레이어 충돌**

**void CollisionRectWeapon(COLOBJ weaponArr[]/\*weapon 배열\*/, COLOBJ playerArr[]**

**플레이어와 무기오브젝트 충돌**

**bool CheckRect(COLOBJ pDst, COLOBJ pSrc, float \* pMoveX, float \* pMoveY)**

**충돌 여부 검사**

플레이어 초기화 함수

**void initPlayerInfo(PlayerInfo\* pInfo, int playerNum)**

플레이어 및 오브젝트, 충돌박스 이동함수

**void UpdatePlayerRect(COLOBJ& rect, PlayerInfo p\_info)**

**//캐릭터 박스 위치 업데이트**

**void UpdateBulletRect(COLOBJ bulletrects[], BulletInfo bullets[])**

**//총알 박스 위치 업데이트**

**void UpdateBullet(BulletInfo bullets[]) 총알 위치 업데이트**

**void MovePlayer(Key keycode) 캐릭터 위치 업데이트**

**void RotatePlayer(float mouseX, float mouseY, int playerID)**

**캐릭터 방향 업데이트**

**void PlayerShoot(Key keycode) 총알 발사 함수**

**void CalcDirVec(int playerID, float p\_posX, float p\_posY, float m\_posX, float m\_posY)**

**방향적 요소를 사용하기 위해 방향벡터를 구하는 함수**

구조체

**플레이어 구조체**

**typedef struct PlayerInfo**

**{**

**bool IsDead; 플레이어의 사망여부**

**float PosX; 플레이어의 위치**

**float PosY; 플레이어의 위치**

**float DirX; 플레이어의 방향**

**float DirY; 플레이어의 방향**

**int playerID; 플레이어 ID**

**int weaponID; 무기 ID**

**int CurBulletNum; 현재 보유한 총알 개수**

**int MaxBulletNum; 최대 총알 개수**

**int HP; 체력**

**int money; 돈**

**float angle; 플레이어의 각도**

**BulletInfo bullets[10]; 플레이어가 생성한 총알 구조체 배열**

**}**

**총알 구조체**

**typedef struct BulletInfo**

**{**

**bool IsDead; 총알이 화면상에 존재하는지 여부**

**float StartPosX; 총알 발사의 시작위치**

**float StartPosY; 총알 발사의 시작위치**

**float PosX; 총알의 현재위치**

**float PosY; 총알의 현재위치**

**float DirX; 총알이 날라가는 방향**

**float DirY; 총알이 날라가는 방향**

**float speed; 총알의 속도**

**float range; 총알의 사정거리**

**int damage; 총알의 데미지**

**};**

**키입력 구조체**

**typedef struct KeyInput**

**{**

**bool key\_W\_Press;**

**bool key\_A\_Press;**

**bool key\_S\_Press;**

**bool key\_D\_Press;**

**bool key\_R\_Press;**

**bool key\_E\_Press;**

**bool Mouse\_R\_Press;**

**bool Mouse\_L\_Press;**

**int playerID;**

**float mouseX;**

**float mouseY;**

**};**

**플레이어 접속여부 확인 구조체**

**typedef struct PlayerNumCheck**

**{**

**int enterPlayerNum;**

**int playerID;**

**};**

“**팀원 별 역할분담**

강진주: 플레이어 컨트롤 코드 제작, Server To Client 구조체 FrameWork 제작, 클라이언트 Class 제작, 멀티 스레드 환경 구축,

Text UI 삽입

신봉섭: 서버 내 충돌처리 및 연산 코드 제작, 맵 제작. 서버 송수신 코드 제작, State 분할 제작

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| 11/1 | 2 | 3  클라이언트 제작 | 4  클라이언트 제작 | 5  클라이언트 제작 | 6  클라이언트 제작 | 7  클라이언트 제작 |
| 8  클라이언트 제작 | 9  클라이언트 제작 | 10  클라이언트 제작 | 11  클라이언트 제작 | 12  Player Struct  & Class 호환코드 작성 | 13  Player Struct  & Class 호환코드 작성 | 14  코드 병합 및 테스트 |
| 15  Bullet Struct  & Class 호환코드 작성 | 16  Bullet Struct  & Class 호환코드 작성 | 17  Terrain Struct  & Class 호환코드 작성 | 18  Terrain Struct  & Class 호환코드 작성 | 19  화면에 플레이어 두 명 출력 구현 | 20  게임에 맞는 터레인 제작 | 21  코드 병합 및 테스트 |
| 22  게임에 맞는 터레인 제작 | 23  멀티 스레드 구체적 방식 설계 | 24  멀티 스레드 구체적 방식 설계 | 25  멀티 스레드  RecvThread함수 작성 | 26  멀티 스레드  RecvThread함수 작성 | 27  충돌 처리 코드 세분화 및 수정 | 28  코드 병합 및 테스트 |
| 29  충돌 처리 코드 세분화 및 수정 | 30  임계 영역  설정 | 12/1  임계 영역  설정 | 2  임계 영역  설정 | 3  Text UI 제작 코드 작성 | 4  Text UI 제작 코드 작성 | 5  코드 병합 및 테스트 |
| 6  버그 테스트 | 7  버그 테스트 | 8  버그 테스트 |  |  |  |  |

**2016184001 강진주 계획표**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| 11/1 | 2  클라이언트 제작 | 3  클라이언트 제작 | 4  클라이언트 제작 | 5  클라이언트 제작 | 6  클라이언트 제작 | 7  클라이언트 제작 |
| 8  클라이언트 제작 | 9  클라이언트 제작 | 10  클라이언트 제작 | 11  클라이언트 제작 | 12  Player간 이동 코드 작성 방식 연구 | 13  Player간 이동 코드 작성 방식 연구 | 14  코드 병합 및 테스트 |
| 15  클라이언트 키 입력  송신 코드  작성(send) | 16  클라이언트 키 입력  송신 코드  작성(send) | 17  서버  키 입력  수신 코드  작성(recv) | 18  서버  키 입력  수신 코드  작성(recv) | 19  연산 코드 서버로 이동 | 20  연산 코드 서버로 이동 | 21  코드 병합 및 테스트 |
| 22  연산 코드 서버로 이동 | 23  연산 코드 서버로 이동 | 24  연산 코드 서버로 이동 | 25  연산 결과 클라이언트에 송신 코드 작성(send) | 26  연산 결과 클라이언트에 송신 코드 작성(send) | 27  클라이언트와의 호환성 테스트 | 28  코드 병합 및 테스트 |
| 29  클라이언트와의 호환성 테스트 | 30  클라이언트와의 호환성 테스트 | 12/1  클라이언트와의 호환성 테스트 | 2  플레이어 입력 대기 구현 | 3  플레이어 입력 대기 구현 | 4  플레이어 입력 대기 구현 | 5  코드 병합 및 테스트 |
| 6  버그 테스트 | 7  버그 테스트 | 8  버그 테스트 |  |  |  |  |

**2016184024 신봉섭 계획표**