애플리케이션 기획

| 800x 800(UI 200 x 800) | **<갤러가>** 리소스를 이용한 1:1 슈팅게임  상대플레이어를 추가하여 비행기를 파괴하거나 화면 너머로 드론을 보내서 체력을 0으로 만들면 승리하는 게임  사용자는 1P 위치에서 플레이하며  상대가 2P역할을 수행한다 |
| --- | --- |

|  | TITLE: 타이틀 씬  1p,2p 중 하나를 선택하여 게임준비  ROOM: 대기방  -1p: 서버 플레이어  -2p: 클라이언트 플레이어  양 플레이어가 준비되면 PLAY로 이동  PLAY: 게임 씬  - 둘 중 한명의 HP가 0이 되면 Print Score로 이동  Print Score: 점수 출력  - 통신을 종료하고 각 플레이어가 얻은 점수 출력 후, 아무 키나 누르면 타이틀로 이동 |
| --- | --- |

좌우 방향키: 비행기가 좌, 우로 움직인다.

Z키: 탄환 발사

X키: 드론 출격

비행선이 파괴되면 HP-1

드론이 화면뒤로 넘어가면 HP-2

캐릭터

| 75x80 | 55x60 | 75x80 | 55x60 |
| --- | --- | --- | --- |
| 사용자 비행선 | 사용자 드론 | 상대 비행선 | 상대 드론 |

하이레벨 디자인

| 서버 | 클라이언트 | |
| --- | --- | --- |
|  |  | |
| 게임상의 오브젝트에 변화가 발생할 경우 4바이트의 데이터 전송  1.게임에서 변화된 정보를 대상 플레이어 정보와 변화의 주체가 된 오브젝트의 ID를 전송  2. 전달된 정보를 토대로 게임상에 반영시킴 | 클라이언트 플레이어가 조작 키를 입력하면 해당 조작과 관련된 1바이트 데이터를 서버에 전송  1. 1바이트 크기의 데이터를 서버에 전송  2. 데이터값에 따라 서버 상의 클라이언트의 행동 수행  3. 수행한 정보와 클라이언트 플레이어의 위치를 합쳐 4바이트의 데이터로 반환  4. 반환된 값을 기준으로 클라이언트에 반영 | |

로우레벨 디자인

| char clientRequest;  클라이언트가 키입력시 서버에 전송하는 정보 | | |
| --- | --- | --- |
| Define Name | Mean | value |
| CLIENT\_PLAYER\_NONE | 클라이언트 플레이어 정지 | 0x01 |
| CLIENT\_PLAYER\_LEFT | 클라이언트 플레이어 왼쪽으로 이동 | 0x02 |
| CLIENT\_PLAYER\_RIGHT | 클라이언트 플레이어 오른쪽으로 이동 | 0x04 |
| CLIENT\_PLAYER\_SHOT | 클라이언트 플레이어 총알 발사 | 0x08 |
| CLIENT\_PLAYER\_DRONE | 클라이언트 플레이어 드론 발사 | 0x10 |

| struct ActValue // 서버가 클라이언트에 전송하는 정보  {  char infoType; // 정보의 종류  char infoOptioin; // 정보 종류의 따른 옵션값(총알/드론 ID, 플레이어 상태값)  short PublicPlayerX; // 해당 충돌과 관계있는 플레이어의 X값  } | | | |
| --- | --- | --- | --- |
| 변수 | Define Name | Mean | value |
| char  infoType | PLAYER\_STATE | 플레이어 상태변화 | 0x01 |
| SPAWN\_PLAYER\_BULLET | 플레이어 총알 생성 | 0x02 |
| ERASE\_PLAYER\_BULLET | 플레이어 총알 소멸 | 0x03 |
| SPAWN\_PLAYER\_DRONE | 플레이어 드론 생성 | 0x04 |
| ERASE\_PLAYER\_DRONE | 플레이어 드론 소멸 | 0x05 |
| OVER\_WINDOW\_PLAYER\_DRONE | 플레이어 드론이 적 플레이어의 뒤편으로 넘어갔다. | 0x08 |
| PLAYER\_BULLET\_COLLISION | 플레이어 총알과 적 플레이어가 충돌 | 0x12 |
| PLAYER\_DRONE\_COLLISION | 플레이어 드론과 적 플레이어가 충돌 | 0x14 |
| PLAYER\_DRONE\_AND\_ENEMY\_BULLET | 플레이어 드론과 적 플레이어 총알의 충돌 | 0x24 |
| ENEMY\_PLAYER\_STATE | 적 플레이어 상태변화 | 0x10 |
| SPAWN\_ENEMY\_BULLET | 적 플레이어 총알 생성 | 0x20 |
| ERASE\_ENEMY\_BULLET | 적 플레이어 총알 소멸 | 0x30 |
| SPAWN\_ENEMY\_DRONE | 적 플레이어 드론 생성 | 0x40 |
| ERASE\_ENEMY\_DRONE | 적 플레이어 드론 소멸 | 0x50 |
| ENEMY\_BULLET\_COLLISION | 적 플레이어 총알과 플레이어가 충돌 | 0x21 |
| ENEMY\_DRONE\_COLLISION | 적 플레이어 드론과 플레이어가 충돌 | 0x41 |
| ENEMY\_DRONE\_AND\_PLAYER\_BULLET | 플레이어 드론과 적 플레이어 총알의 충돌 | 0x42 |
| OVER\_WINDOW\_ENEMY\_DRONE | 적 플레이어 드론이 플레이어의 뒤편으로 넘어갔다. | 0x80 |
| char  infoOptioin | <Define 없음>  총알/드론 생성시 | 현재 화면의 총알/드론의 갯수를 전송 | 1~256 |
| <Define 없음>  총알/드론 제거시 | 총알/드론의 ID값을 전송 | 0~255 |
| enum state  {  IDLE = 1  LEFT  RIGHT  } | 해당 플레이어의 상태값 | 1~3 |
| short  PublicPlayerX | <Define 없음> | 변화 발생시 관련있는 플레이어의 X위치 전달 | 233 ~ 777 |

| 서버 |
| --- |
| // 임의의 오브젝트간에 충돌이 발생하는 경우  bool Collision\_send(object objA, object objB)  {  if(CollisionCheck(objA, objB) == false)  return false;  ActValue actValue; // 전송할 데이터  int objANum = obj.getID(); // 오브젝트A ID  int objBNum = obj.getID(); // 오브젝트B ID  int objAType = obj.getType(); // 오브젝트A 종류  int objBType = obj.getType(); // 오브젝트B 종류  actValue.infoOption = objNum;    if(//플레이어와 총알){  actValue.infotype = PLAYER\_BULLET; // 플레이어/상대  actValue.infoOption = ID; // 총알의 ID  actValue.PublicPlayerX = player.position.x; //플레이어의 x위치 대입  }  else if(//플레이어와 드론){  actValue.infotype = PLAYER\_DRONE; // 플레이어/상대  actValue.infoOption = ID; // 드론의 ID  actValue.PublicPlayerX = player.position.x; //플레이어의 x위치 대입  }  else if(//총알과 드론){  actValue.infotype = PLAYER\_DRONE\_AND\_ENEMY\_BULLET; // 플레이어/상대  actValue.infoOption = ID; // **드론의 ID**  actValue.PublicPlayerX = //**예외로 총알의 ID 대입**  }  // 데이터 전송  send(socket, (char\*)actValue, sizeof(ActValue), 0);  return true;  }  // 임의의 플레이어의 상태가 변경될 때  void ChangePlayerState(PlayerInfo player, PlayerState state)  {  ActValue actValue; // 전송할 데이터  player.state = state; // 대상 플레이어의 상태값 변경  // 구조체에 해당되는 플레이어의 정의값 대입  actValue.infoType = PLAYER\_STATE; // or ENEMY\_PLAYER\_STATE  actValue.infoOption = state; // 구조체에 상태값 대입  actValue.PublicPlayerX = player.position.x; //플레이어의 x위치 대입  // 데이터 전송  send(socket, (char\*)actValue, sizeof(ActValue), 0);  }  **void ActByClientRequest**(char clientRequest)  {  ActValue actValue; // 전송할 데이터  //행동값에 따라서 해당 값을 대입  switch(clientRequest){  case CLIENT\_PLAYER\_NONE:  case CLIENT\_PLAYER\_LEFT:  case CLIENT\_PLAYER\_RIGHT:  actValue.infoType = ENEMY\_PLAYER\_STATE;  break;  case CLIENT\_PLAYER\_SHOT:  actValue.infoType = ENEMY\_BULLET;  break;  case CLIENT\_PLAYER\_DRONE:  actValue.infoType = ENEMY\_DRONE;  break;  }  // 행동의 종류의 따라서 해당행동 실행과 동시에 옵션값 대입  switch(actValue.infoType){  case CLIENT\_PLAYER\_NONE://////// 상태변경 /////////  player.state = IDLE;  actValue.infoOption = IDLE;  break;  case CLIENT\_PLAYER\_LEFT:  player.state = LEFT;  actValue.infoOption = LEFT;  break;  case CLIENT\_PLAYER\_RIGHT:  player.state = RIGHT;  actValue.infoOption = RIGHT;/////////////////////////////////////////////  break;  case CLIENT\_PLAYER\_SHOT:  SpawnBullet(player.position); // 플레이어 위치에 총알 생성  actValue.infoOption = bullet.maxcount; // 화면상 총알의 최대갯수  break;  case CLIENT\_PLAYER\_DRONE:  SpawnDrone(player.position); // 플레이어 위치에 드론 생성  actValue.infoOption = drones.maxcount; // 화면상 드론의 최대갯수  break;  }    actValue.PublicPlayerX = player.position.x; //플레이어의 x위치 대입  // 데이터 전송  send(socket, (char\*)actValue, sizeof(ActValue), 0);  }  //충돌에 쓰이는 것이 아닌 단순 소멸 - 총알/드론에 사용  void EraseObject(Player player, Object objects, int objID)  // 지울 오브젝트들 정보, 대상 오브젝트ID  {  objects.erase(objID); // 대상 오브젝트 제거  actValue.infoType = ERASE\_ENEMY\_BULLET; // 플레이어/상대의 총알/드론 소멸  actValue.infoOption = objID; // 대상ID  actValue.PublicPlayerX = player.position.x; //플레이어의 x위치 대입  // 데이터 전송  send(socket, (char\*)actValue, sizeof(ActValue), 0);  }  void OverWindowDrone(Drones drones, int droneID)  {  //드론들의 정보를 가진 데이터에서 화면을 넘은 드론을 제거한다.  Drones.erase(droneID);  player.HP -= 2; // 넘어가서 사라지는경우 상대방에게 데미지  actValue.infoType = OVER\_WINDOW\_PLAYER\_DRONE; // 플레이어/상대  actValue.infoOption = drones.size; // 현재드론의 최대갯수  actValue.PublicPlayerX = player.position.x; //플레이어의 x위치 대입  // 데이터 전송  send(socket, (char\*)actValue, sizeof(ActValue), 0);  } |
|  |

| 클라이언트 |
| --- |
| //Thread로 수행할 함수  void OrderQueueRecvActivate()  {  ActValue actValue;  while(/\*게임오버가 호출되지 않았다면\*/){  recv(sock, actValue, sizeof(actValue), 0);  mutex.lock();//뮤텍스로 공용자원 보호  queueActValue.push(actValue); // 수신되는 모든정보는 하나의 큐에서 관리  mutex.unlock();  }  }  // update 중에 queue<ActValue>의 크기 만큼 호출되는 함수  void DecordActServerResponse(ActValue actValue)  {  switch(actValue.infoType){  case PLAYER\_STATE:  player.position.x = actValue.PublicPlayerX;  /\*플레이어 상태 변수\*/ = actValue.infoOption;  break;  case SPAWN\_PLAYER\_BULLET:  player.position.x = actValue.PublicPlayerX;  /\*플레이어 총알 생성\*/  if/\*actValue.infoOption(현재 최대갯수)와 현재 총알의 최대갯수를 비교\*/  //다르다면 같아질때까지 가장 먼저 생성되었던 총알부터 소멸  break;  case ERASE\_PLAYER\_BULLET:  player.position.x = actValue.PublicPlayerX;  /\*actValue.infoOption가 ID인 플레이어 총알 소멸\*/  break;  case SPAWN\_PLAYER\_DRONE:  player.position.x = actValue.PublicPlayerX;  /\*플리이어 드론 생성\*/  if/\*actValue.infoOption(현재 최대갯수)와 현재 드론의 최대갯수를 비교\*/  //다르다면 같아질때까지 가장 먼저 생성되었던 드론부터 소멸  case ERASE\_PLAYER\_DRONE:  player.position.x = actValue.PublicPlayerX;  /\*actValue.infoOption가 ID인 플레이어 드론 소멸\*/  break;  case OVER\_WINDOW\_PLAYER\_DRONE:  player.position.x = actValue.PublicPlayerX;  /\*현 드론 중 가장 먼저 생성된 플레이어 드론 소멸\*/  if/\*actValue.infoOption(현재 최대갯수)와 현재 드론의 최대갯수를 비교\*/  //다르다면 같아질때까지 가장 먼저 생성되었던 드론부터 소멸  break;  case PLAYER\_BULLET\_COLLISION:  Bullets.erase(actValue.infoOption) // 총알 객체 중 해당 ID를 가진 대상 소멸  player.position.x = actValue.PublicPlayerX;  /\*플레이어 체력 - 1\*/  break;  case PLAYER\_DRONE\_COLLISION:  Drones.erase(actValue.infoOption) // 드론 객체 중 해당 ID를 가진 대상 소멸  player.position.x = actValue.PublicPlayerX;  /\*플레이어 체력 - 2\*/  break;  case PLAYER\_DRONE\_AND\_ENEMY\_BULLET:  Drones.erase(actValue.infoOption) // 드론 객체 중 해당 ID를 가진 대상 소멸  Bullets.erase(actValue.PublicPlayerX) // 총알 객체 중 해당 ID를 가진 대상 소멸  break;  case ENEMY\_PLAYER\_STATE:///////////////////플레이어 로직과 같음/////////////////////////  break;  case SPAWN\_ENEMY\_BULLET:  break;  case ERASE\_ENEMY\_BULLET:  break;  case SPAWN\_ENEMY\_DRONE:  break;  case ERASE\_ENEMY\_DRONE:  break;  case ENEMY\_BULLET\_COLLISION:  break;  case ENEMY\_DRONE\_COLLISION:  break;  case ENEMY\_DRONE\_AND\_PLAYER\_BULLET:  break;  case OVER\_WINDOW\_ENEMY\_DRONE:////////////////////////////////////////////  break;  }  }  // 키입력시에 대한 내용을 포함한 함수  void keydown(int key){  char buf;  switch(key){  case VK\_LEFT:  state = PS\_LEFT;  buf= CLIENT\_PLAYER\_LEFT;  break;  case VK\_RIGHT:  state = PS\_RIGHT;  buf= CLIENT\_PLAYER\_RIGHT;  break;  case 'z' | 'Z':  buf= CLIENT\_PLAYER\_SHOT;  break;  case 'x' | 'X':  buf= CLIENT\_PLAYER\_DRONE;  break;  }  send(socket, (char\*)buf, sizeof(char), 0);  }  void keyUp(int key){  char buf;  switch(key){  case VK\_LEFT:  state = PS\_IDLE;  buf= CLIENT\_PLAYER\_IDLE;  break;  case VK\_RIGHT:  state = PS\_IDLE;  buf= CLIENT\_PLAYER\_IDLE;  break;  }  send(socket, (char\*)buf, sizeof(char), 0);  }  //총알 추가함수  void AddBullet()  {  Bullet\* bullet = new Bullet();  vector.push\_back(bullet);  }  //총알 제거함수  void EraseBullet(int ID)  {  Bullet\* bullet = &vector[ID];  vector.erease(ID);  delete bullet;  }  //드론 추가함수  void AddDrone()  {  Drone\* drone= new Drone();  vector.push\_back(drone);  }  void EraseDrone(int ID)  {  Drone\* drone= &vector[ID];  vector.erease(ID);  delete drone;  }  void AddEnemyBullet()  {  EnemyBullet\* bullet = new EnemyBullet();  vector.push\_back(bullet);  }  void EraseEnemyBullet(int ID)  {  EnemyBullet\* bullet = &vector[ID];  vector.erease(ID);  delete bullet;  }  void AddEnemyDrone()  {  EnemyDrone\* drone= new Drone();  vector.push\_back(drone);  }  void EraseEnemyDrone(int ID)  {  EnemyDrone\* drone= &vector[ID];  vector.erease(ID);  delete drone;  } |

일정 정리를 위한 TODO

| TODO 처리 순서 |
| --- |
| 서버/클라 구분이 **없는** 작업: **빨간색**  서버/클라 구분이 있는 작업: **검은색**   1. WSAManager(네트워크 통신 작업) 클래스 작업 - 서버/클라에 따라서 동작이 다름    1. WSAManager는 명시적 이름 - 서버/클라는 각자의 클래스로 나눠 작업할 것    2. 서버는 4바이트 송신, 1바이트 수신이 발생    3. 클라이언트는 1바이트 송신, 4바이트 수신이 발생 2. player 클래스 작업    1. **클라이언트** **키입력만** 네트워크 통신기능 사용 3. **enemyPlayer 클래스 작업** - WSAManager의 요청에 맞추어 동작    1. **서버**는 1바이트 **클라이언트**는 4바이트 사용 - 로우레벨 참조 4. **적/아군 Bullet/Drone 작업 - 클래스 4개** 5. ObjectManager 클래스 작업 - 생성/소멸 + 충돌처리 담당    1. 충돌처리 작업 - 서버ONLY + 네트워크 통신 작업 필요    2. **생성/소멸 작업 - 생성/소멸 함수는 공통**    3. **서버 송신작업** - 충돌처리가 일어난 뒤, 생성/소멸 함수호출 후 송신발생    4. **클라이언트 수신작업** - 서버에서 받은 정보를 바탕으로 생성/소멸 함수 호출 |
| 모든 작업에 쓰이는 클래스와 함수 - 초기화, 업데이트, 렌더링 관련함수는 생략 |
| 초록새글자 : 안쓸거같음   1. WSAManager - 송/수신 되는 정보들을 관리하는 클래스    1. 서버 - 논블로킹 모드로 recv()는 업데이트 중에       1. *char GetClientRequset() - 가장 최근 클라이언트 요청정보 반환*       2. void GameInfoSend(ActValue) - 클라이언트로 인자를 송신       3. void DecordClientRequest(); - 수신내용을 바탕으로 정보변경    2. 클라 - recv작업은 thread를 사용       1. *ActValue GetActValue() - 가장 최근 서버에서 보낸 정보 반환*       2. void DecordServerData(); - 수신내용을 바탕으로 정보변경 2. Player - 사용자가 조종하는 대상    1. 공통       1. void ChangeState()       2. void loseHP()    2. 서버       1. void KeySetting() - Player 클래스 직접 조종       2. RequestSpawnBullet() - ObjectManager에 요청       3. RequestSpawnDrone()- ObjectManager에 요청    3. 클라이언트       1. void KeySetting() - send() 발생       2. RequestSpawnBullet() - send() 발생       3. RequestSpawnDrone() - send() 발생 3. EnemyPlayer - 사용자의 상대방    1. 공통       1. void ChangeState()       2. void loseHP()   ----------------------------ObjectManager에서 관리되는 객체들------------------------------------   1. Bullet - 사용자의 총알 2. EnemyBullet - 상대방의 총알 3. Drone - 사용자의 드론 4. EnemyDrone - 상대방의 드론   -----------------------------------------------------------------------------------------------------------------   1. ObjectManager - 적/아군의 총알/드론을 관리하는 클래스    1. 공통 - 생성/소멸 관련 함수       1. void AddBullet()       2. void EraseBullet(int ID)       3. void AddDrone()       4. void EraseDrone(int ID)       5. void AddEnemyBullet()       6. void EraseEnemyBullet(int ID)       7. void AddEnemyDrone()       8. void EraseEnemyDrone(int ID)    2. 서버       1. void CollisionInfoSend() - 충돌 발생시 클라이언트에 데이터 송신       2. bool CollisionCheck(obj, obj) - 충돌처리 함수 |

팀원 별 역할분담

최치송 - 서버 송/수신 처리

박의인 - 클라이언트 송/수신 처리

개발환경

| 운영체제: 윈도우  개발환경: 비주얼 스튜디오 2019  API: Window API  버전관리: github |
| --- |

개발일정 (일별/개인별)

11/11 ClientInitialize

11/12 DecordServerData

11/13 Request 큐에 데이터 삽입

11/14 Player class 초기화, 업데이트, 렌더

11/16 KeySetting

11/17 송수신 점검

11/18 RequestSpawnBullet

11/19 RequestSpawnDrone

11/20 ChangeState

11/21 loseHP

11/23

EnemyBullet 클래스 초기화, 업데이트, 렌더

11/24

Drone 클래스 초기화, 업데이트, 렌더

11/25

EnemyDrone 클래스 초기화, 업데이트, 렌더

11/26

DecordServerData

11/27

최종점검

작업 리스트에 따라 추가된 것 - 조정할 것임

클래스 작업 - 초기화, 업데이트, 렌더관련 작업을 포함함

|  |  | 11/10 | 11 | 12 | 13 | 14 |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | WSAManager 클래스 작업  void ServerInitialize() | WSAManager 클래스 작업  void DecordClientRequest(); | Player 클래스 작업  void KeySetting() | EnemyPlayer 클래스 작업  void ChangeState()  loseHP() | 오브젝트 클래스들 작업  Bullet  EnemyBullet  Drone  EnemyDrone |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| ObjectManager 클래스 작업  오브젝트 생서/소멸 함수 | ObjectManager 클래스 작업  void CollisionInfoSend() | 네트워크 통신을 통하여 플레이어 이동 확인 및 수정 | WSAManager 클래스 작업  void DecordClientRequest() - ObjectManager 생성/소멸 통신작업 | WSAManager 클래스 작업  void DecordClientRequest() - 충돌처리 네트워크 통신작업 | 네트워크 통신을 통하여 충돌처리 데이터 처리 확인 및 수정 | 씬구분 작업 |

| 최치송 | 박의인 |
| --- | --- |
| ServerManager - socket()~closesocket() 1일 | ClientManager - socket()~closesocket()  1일 |
| nonblocking buffer  5일 | player class  5일 |
| ServerManager - nonbloking buff test  1일 | ClientManager - nonbloking buff test  1일 |
| 중간점검 7일 | |
| EnemyControler  6일 | Drone, bullet  6일 |
| Scene  5일 | collision Manager  5일 |
| 최종점검 7일 | |

| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 11/4 | 11/5 | 11/6 | 11/7 |
|  |  |  | Player class 작성   * 초기화관련 멤버함수와 private 멤버변수 접근을 위한 GetSet함수 작성 | | Player class  - 키 입력에 따른 상태 변화 추가 | |
| 11/8 | 11/9 | 11/10 | 11/11 | 11/12 | 11/13 | 11/14 |
| Drone class  - 위치 정보 생성 후 이미지 표시 | | Bullet class  - 위치 정보 생성 후 이미지 표시 | | Player, Drone, Bullet class 점검  - 이미지는 원하는 위치에 생성이 되는가?  - 키 입력에 따른 상태 변화는 원하는 대로 적용 되는가? | | |

| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| --- | --- | --- | --- | --- | --- | --- |
| 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 | 11/21 |
| WSAManager  - 소켓 생성 후 소켓 소멸 | | WSAManager  - 데이터 수신과 송신 기능 | | - WSAManager(Sever)에서 받은 데이터가 Player, Drone, Bullet class 에서 적용 되는가?  - 적용 된다면 WSAManager(Sever) 에서 보낸 데이터와 WSAManager(Client)가 수신한 데이터의 값은 일치하는 가? | | |
| 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 | 11/28 |
| WSAManager가 잘 작동하는 지 중간점검  - WSAManager(Sever)와 WSAManager(Client) 송수신 점검  - 데이터 손실 유무 점검 | | | | | | |

| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| --- | --- | --- | --- | --- | --- | --- |
| 11/29 | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 | 12/5 |
| CollisionManager  - 플레이어와 드론 충돌 | | CollisionManager  - 플레이어와 총알 충돌 | | CollisionManager  - 총알과 드론 충돌 | | |
| 12/6 | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 | 12/12 |
| CollisionManager가 잘 작동하는 지 점검  - 정확한 위치에서 충돌을 하는가?  - 충돌 무시 현상은 발생하는가? | | | CollisionManager에서 도출 된 결과를 WSAManager가 잘 송수신 하는지 점검 | | | |

| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| --- | --- | --- | --- | --- | --- | --- |
| 11/13 | 11/14 | 12/15 | 12/16 | 12/17 | 12/18 | 12/19 |
| 최종점검 | | | | | | |
| 12/20 | 12/21 | 12/22 | 12/23 | 12/24 | 12/25 | 12/26 |
|  |  |  |  |  |  |  |