**네트워크**

**게임 프로그래밍**

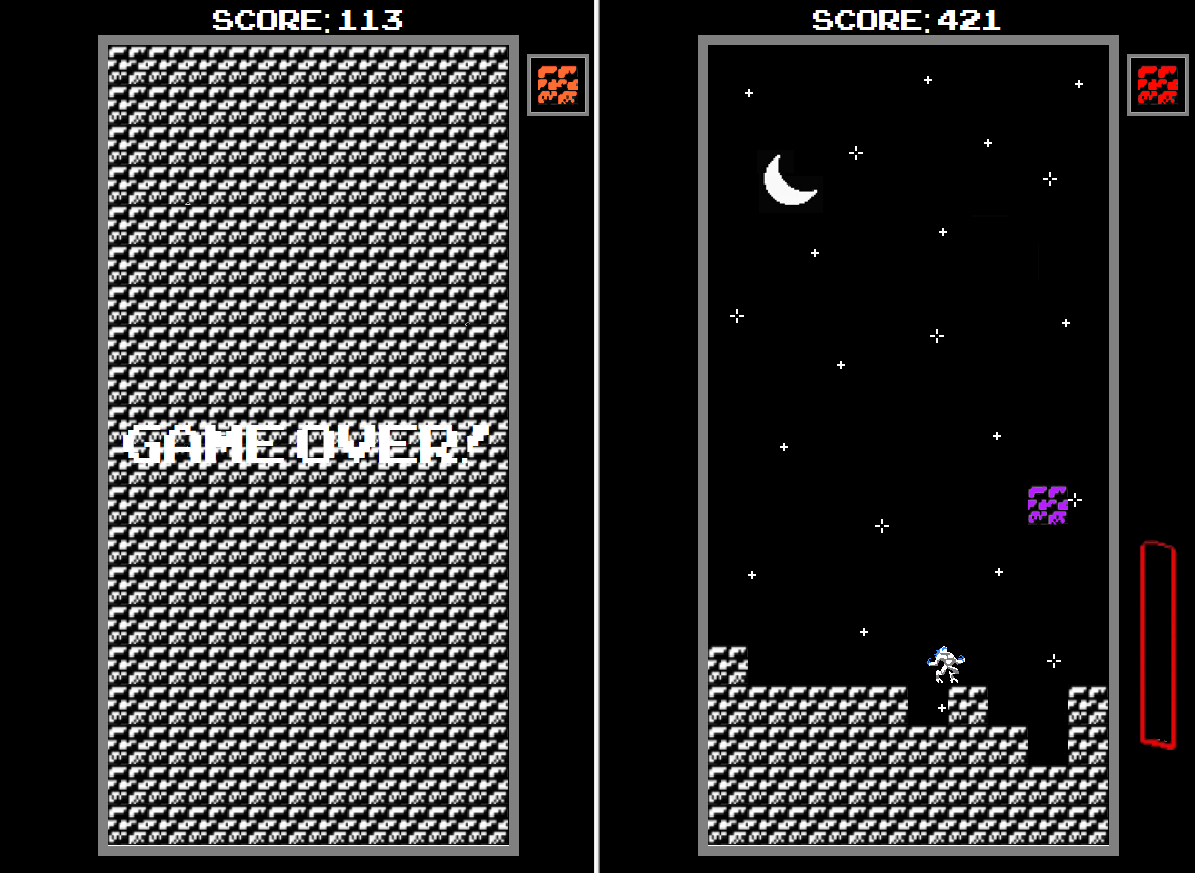
**Project Progress Report**

**-Evaid-**

|  |  |
| --- | --- |
| **분반 / 수업시간** | 02반 / 화78, 목78 |
| **학과** | 게임공학과 |
| **학번/이름** | 2016180006 / 김동석  2016180012 / 김영준 |

**목차**

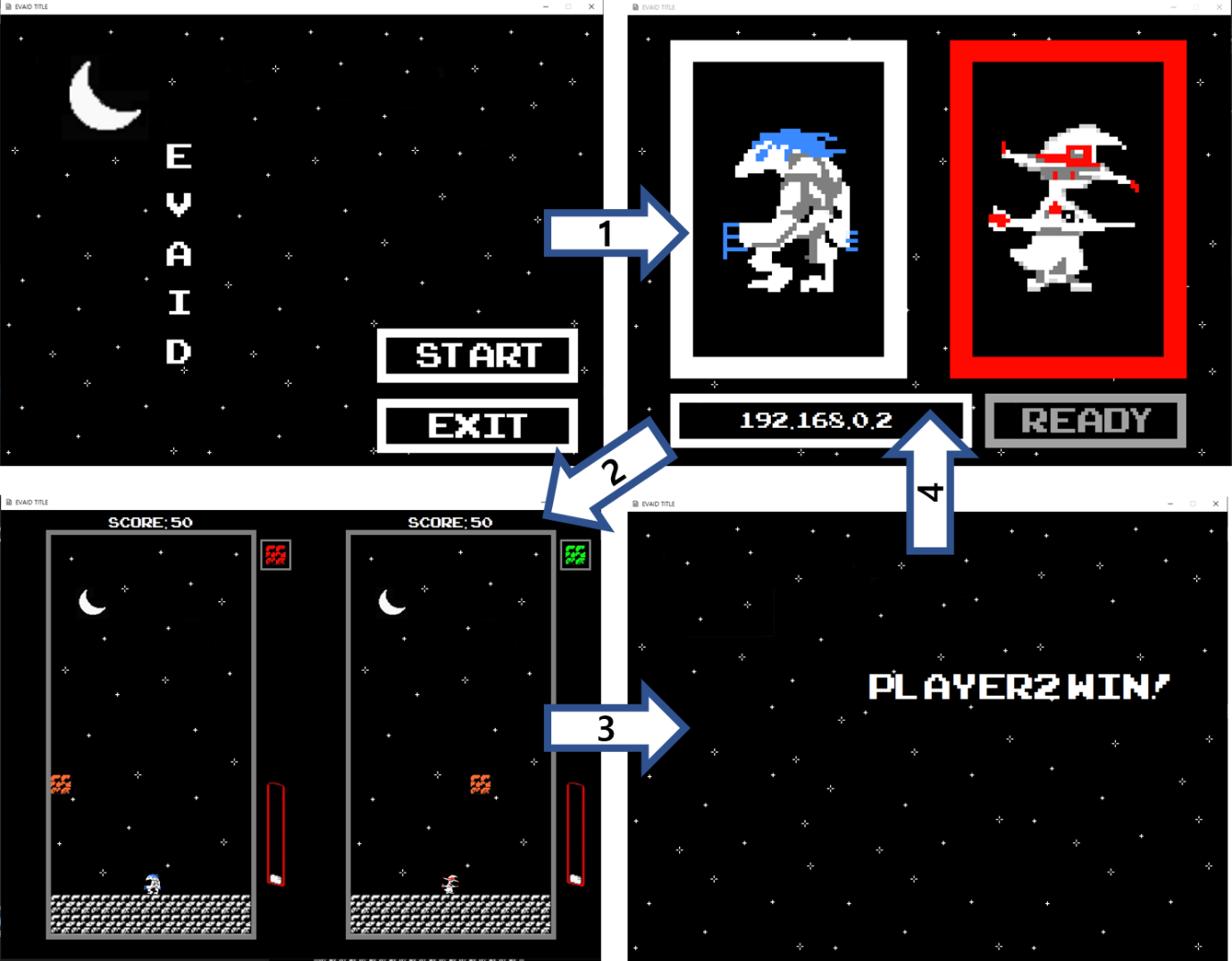
1. **개발 개요**
   1. 개발 목적
   2. 개발 환경
   3. 개발 도구
2. **게임 개요**
   1. 게임 소개
   2. 게임 흐름
3. **High Level Design**
4. **Low Level Design**
   1. 서버 설계
   2. 클라이언트 설계
   3. Application 프로토콜
5. **동기화**
6. **역할 분담 및 개발 일정**
7. **개발 개요**
   1. 개발 목적  
      2인 대전 멀티플레이 게임 개발  
      Multi Thread 활용 능력 배양  
      Server/Client 동기화 기법 습득
   2. 개발 환경
      1. 플랫폼: Windows PC
      2. 사용 API: Window Socket API, Window API
      3. 프로토콜: TCP/IP
   3. 개발 도구  
      Visual studio 19  
      C++
8. **게임 개요**
   1. 게임 소개



1. 제목: Evaid
2. 장르: 대전
3. 게임 방식:

2명의 플레이어가 떨어지는 블럭을 피함과 동시에 에너지를 모아 스킬을 사용하여 상대 플레이어가 떨어지는 블럭을 피하지 못하게 만들어 상대 플레이어보다 더 오래 생존하면 승리하는 게임

1. 조작법  
   방향키: 좌, 우 이동  
   Z: 점프   
   X: 스킬 사용



* 1. 게임 흐름

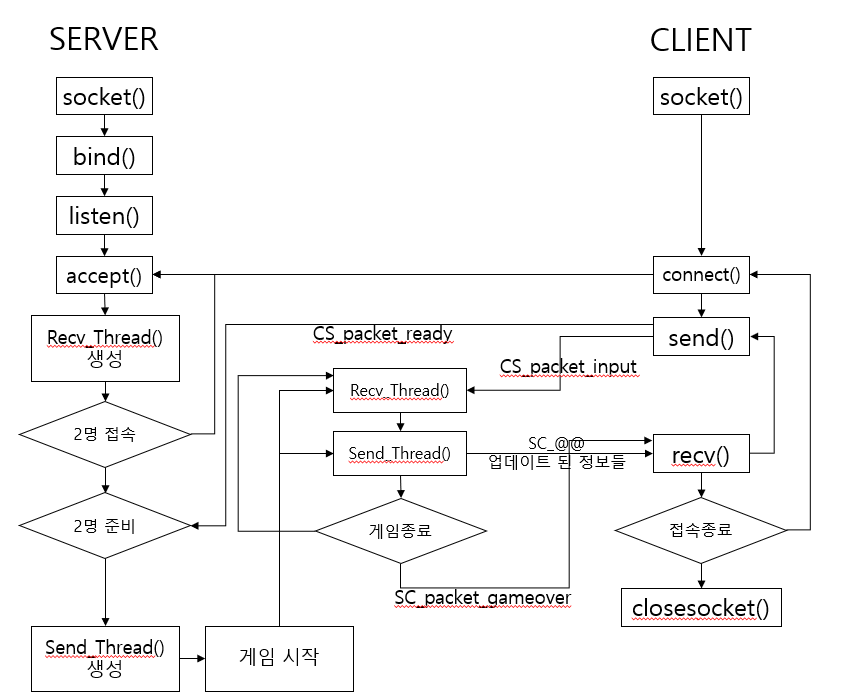
1. ~~Title Scene에서 IP주소 입력~~

**→** Title Scene에서 Start 버튼을 눌러 Lobby Scene으로 이동

1. Lobby Scene에서 ~~SPACE BAR 키로 READY,  
   플레이어가 모두 READY 시 게임 시작~~

**→** 키보드로 IP 입력 후 모든 플레이어가 Ready 시 게임시작

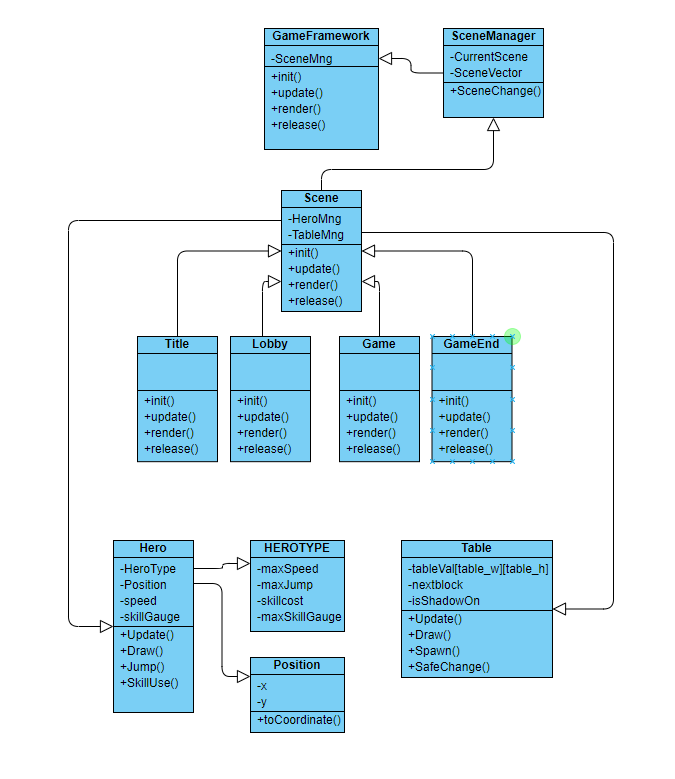
1. 게임 플레이
2. 게임 종료 시 승리 플레이어 출력 후 Lobby Scene으로 전환
3. **High Level Design**

****

**[FLOW CHART]**

* 1. **Client**

1. **클라이언트 실행 후, 서버에 connect 한다.**
2. **서버에서 요청이 받아지면 서버로부터 ID를 할당 받는다.**
3. **로비에 진입하고, 준비되었다면 준비 상태로 바꾼다.**
4. **게임 시작 메시지(초기화된 월드 상태)를 받는다.**
5. **키 입력 시 ~~해당 키에 해당하는 명령 메시지를 서버에 보낸다.~~   
   → 해당 키의 명령(이동, 충돌처리 등)을 처리한 후 업데이트된 내용을 서버  
    에 보낸다.**
6. **서버에서 업데이트 된 월드 상태(플레이어, 상대의 위치, 소환된 블록의 위치)를 받는다. 업데이트 된 내용으로 그린다.**
7. **5~6을 반복하고, 게임 결과를 받으면 로비로 이동한다.**
8. **사용자가 접속종료 시 게임을 종료한다.**
   1. **Server**
9. **서버 실행 후, listen 상태에서 클라이언트의 connect 요청이 오면 accept 후 클라이언트에 ID부여, Recv\_Thread를 만든다.**
10. **로비에서 2개의 클라이언트의 준비상태를 확인한다. 모두 준비상태이면 SendThread를 만들고, 초기화된 월드 상태를 보낸다.**
11. **RecvThread는 클라이언트의 메시지를 받는다. 그리고 메시지의 타입을 구분하여 공유 자원에 넣는다. (로비에서의 메시지 처리는 RecvThread에서 한다.)**
12. **SendThread는 공유 자원의 값을 읽어 월드 상태를 업데이트한다. ~~블럭과 플레이어 간의 충돌처리를 하고 업데이트된 월드 상태를 클라이언트에 전송한다.~~→ 업데이트 된 월드 상태를 각각의 클라이언트에게 전송한다.**
13. **3~4번을 반복한다. 생존자가 1명이 되면 모든 클라이언트에게 게임 결과를 보내준다.**
14. **Low Level Design**
    1. **Client**

****

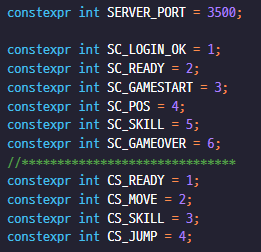
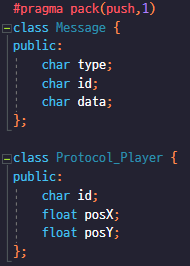
* GameFramework: main에서 호출되며 클라이언트의 코어, Scene을 관리한다.
* init(): SceneMng 생성, TitleScene 추가
* update(): SceneMng의 CurrentScene->update() 호출
* render(): SceneMng의 CurrentScene->render() 호출
* release(): Singletone 객체인 SceneMng 삭제 후 ctor 호출
* SceneManager: Scene 관리(Current변경, 업데이트, 렌더링)
  + ~~SceneVector: 추가된 Scene의 목록~~
  + CurrentScene: 선택된(화면에 출력되는) Scene
  + NextScene: CurrentScene다음으로 화면에 출력될 Scene
* Scene: 인터페이스 함수, 가상함수 init(), update(), render(), release() 관리
* TitleScene: ~~키보드를 통해 서버 IP주소를 입력하여 서버에 접속, 접속에 성공하면 LobbyScene으로 CurrnetScene을 바꾼다.~~
  + ~~Init(): non-block-socket 생성~~
  + ~~Update(): IP주소를 입력하면 해당 IP주소의 서버에 Connect, 성공하여 자신의 ID을 받으면 LobbyScene으로 CurrnetScene 전환~~

**→** Start버튼을 클릭하면 LobbyScene으로 전환

* LobbyScene: 키보드를 통해 서버IP주소를 입력하여 서버에 접속, 접속한 플레이어가 모두 준비상태가 되면 GameScene으로 CurrnetScene 전환
  + Update(): IP를 키보드 입력을 통해 입력 받아 서버에 연결하고, 서버에서 제공받은 나의 ID와 다른 클라이언트의 ID의 준비상태를 받음, Ready버튼을 누르면 서버에 cs\_packet\_ready를 전송, 모든 클라이언트가 Ready가 되면 GameScene으로 CurrnetScene 전환
  + Render(): 접속한 클라이언트 ID, Connect Button, Ready Button을 그림
* GameScene: 서버와 클라이언트간 통신을 통해 게임 오브젝트를 그리고 지움, 플레이어가 한 명 죽으면 시간이 지나고 LobbyScene으로 전환
  + Update(): 키 입력에 따라서 연산(충돌처리, 이동, 점프 등)을 통해 게임 정보를 업데이트한 후 정보를 서버에 전달, 한 명이 죽으면 게임 종료상태 이후 ResultScene으로 CurrentScene 전환
  + Render(): 받아온 정보를 화면에 출력함
* ResultScene: 승리 플레이어를 화면에 출력해 준 후 일정시간이 지나고 CurrentScene을 LobbyScene으로 전환
* Hero: 플레이어가 상호작용하는 객체
  + ~~addForce(vector2 force, float elapsedTime)~~
    - ~~입력한 힘의 방향과 HEROTYPE의 능력치를 통해 움직인다~~.

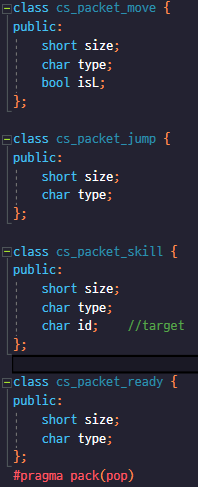
→ - Move(): 플레이어의 이동을 관리

* + Jump(): ~~cs\_packet\_jump를 전송하여 서버에서~~ 점프관련 처리.
  + skillUse(): ~~cs\_packet\_skill를 전송하여 서버에서~~ 스킬관련 처리
  + SetPacketToHero(): 서버로부터 받아온 sc\_packet\_user로부터 Hero의 상태를 설정
  + GetHeroToPacket(): 클라이언트에서 업데이트한 Hero의 정보를 cs\_packet\_user에 저장
* Table: Hero가 상호작용하는 맵
  + tableVal[ ][ ]: coordi 기반으로 만들어진 Table의 정보를 담는 변수
  + nextblock: 다음에 떨어질 블록의 정보~~(서버를 통해 받아옴)~~
  + isShadowOn: 블록이 추가로 생성되는 위험범위 표시여부
  + SafeChange(): 블록의 위치를 바꿀 때 바꿀 수 있는 위치인지 확인하고 바꾸는 함수
  + Spawn(): 블록을 정해진 위치에서 소환
  + SetPacketToTable(): 서버로부터 받아온 sc\_packet\_user로부터 Table의 상태를 설정
  + GetTableToPacket(): 클라이언트에서 업데이트한 Table의 정보를 cs\_packet\_user에 저장
  1. **Server**
* ~~MsgQueue~~
  + ~~Recv 받은 모든 Message을 담고있는 Queue~~
* Recv\_Thread()
  + client(socket)별로 하나씩 생성, recv받은 packet의 type별로 작업 수행
  + Send\_Thread()가 생성되기 이전, 전송해야 하는 Ready, Start 소켓은 Recv\_Thread()안에서 처리한다
  + ~~Void sendReadyPacket(char id, bool ready): id client에서 변경된 ready 상태를 모든 client에게 전~~송
  + Client::getReadyAll()을 통해 각 클라이언트의 Ready을 확인하고 상태를 전달
  + Int sendAll(char\*, int, int): 모든 client에게 buffer의 내용을 send
* Send\_Thread()
  + ~~MsgQueue을 통해 전달받은 Message을 통해 게임 로직에 관련된 연산을 하고, 결과값을 각각의 Client에 Send~~
  + ~~Coordi changePosition(Position pos): recv 받은 position을 Table 안의 좌표(Coordi)로 변환하는 함수~~
  + ~~Void checkBlockCollision(): 모든 Table의 Block과 모든 Hero의 Coordi가 충돌했는지 여부를 확인, 충돌했다면 sc\_packet\_gameover 전송~~
  + ~~Void update(float elapsedTime):~~ ~~모든 Hero, Table의 Block의 사용자 입력, 물리적인 처리를 진~~행
  + Client::Scene값을 기준으로 Recv\_Thread()에서 설정한 각각의 패킷에 대한 처리를 실행
  1. **Application 프로토콜(수정 전)**

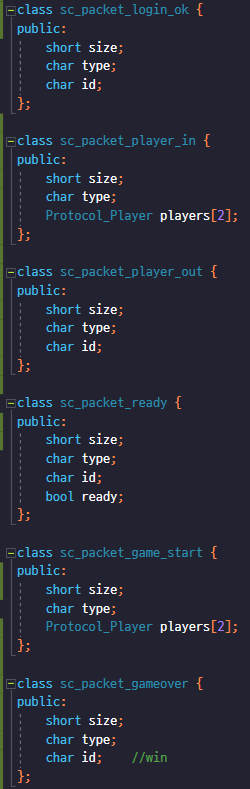


[메시지 클래스와 플레이어 프로토콜]

[메시지 Type]

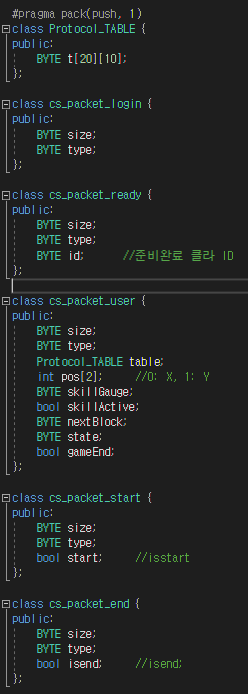
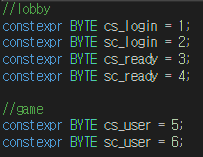


[Client → Server - Packet]

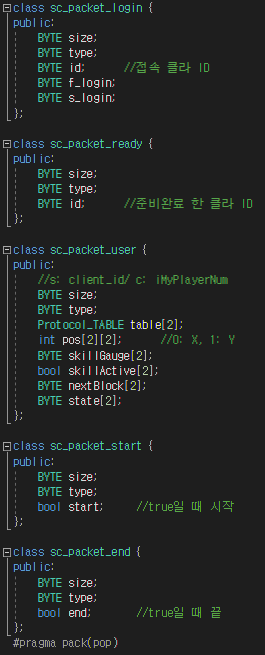


[Server → Client - Packet]

1. **Application 프로토콜(수정 후)**



[메시지 Type]



[Client → Server - Packet]

[Server → Client - Packet]

1. 동기화

* ~~TableVal[ ][ ]는 공유자원으로써 임계영역을 통해 동기화 진행(순서에 상관없이 하나의 Thread만 사용해야 한다)~~

**→** 각각 다른 Table을 사용하므로 Table에 대한 동기화가 필요하지 않다.

1. 역할 분담 및 개발 일정

|  |  |  |  |
| --- | --- | --- | --- |
|  | 내용 | 김영준 | 김동석 |
| 기획 | 기획서 작성 | **O** | **O** |
| Protocol | Protocol.h | **O** | **O** |
| Server | Recv\_Thread() | **O** | **X** |
|  | Send\_Thread() | **O** | **O** |
| Client | Gameframework | **X** | **O** |
|  | SceneManager | **O** | **X** |
|  | ~~AddForce~~ | **~~O~~** | **~~O~~** |
|  | blockDrop | **X** | **O** |
|  | HeroMove | **O** | **X** |

|  |  |  |
| --- | --- | --- |
| **날짜** | **김동석** | **김영준** |
| 11/03 | GameFrameWork 제작 | SceneManager 제작 |
| 11/04 |
| 11/05 | GameFrameWork 점검 | Messege, Packet 관련 제작 |
| 11/06 |
| 11/07 | 개인 학습 | |
| 11/08 | 1주차 점검 및 회의 | |
| 11/09 | ServerFrameWork 제작 | ClientFrameWork 제작 |
| 11/10 |
| 11/11 | 개인 학습 | |
| 11/12 | AddForce() | |
| 11/13 | BlockDrop() | HeroMove() |
| 11/14 | 개인 학습 | |
| 11/15 | 2주차 점검 및 회의 | |
| 11/16 | SendThread() – ColisionLogic() | |
| 11/17 |
| 11/18 | 개인 학습 | |
| 11/19 | SendThread() – jump(), move() | SendThread() – ColisionLogic() |
| 11/20 |
| 11/21 | 개인 학습 | |
| 11/22 | 3주차 점검 및 회의 | |
| 11/23 | SendThread() – Skill() | RecvThread() – Ready |
| 11/24 | RecvThread() – GameStart |
| 11/25 | 개인 학습 | |
| 11/26 | SendThread() 최종 점검 | RecvThread() 최종 점검 |
| 11/27 |
| 11/28 | 개인 학습 | |
| 11/29 | 4주차 점검 및 회의 | |
| 11/30 | 동기화 작업 – (플레이어, 블록) | |
| 12/01 | 동기화 작업 – (플레이어, 플레이어) | |
| 12/02 | 개인 학습 | |
| 12/03 | 최종 코드 점검 | |
| 12/04 |
| 12/05 | 개인 학습 + 테스트 | |
| 12/06 | 5주자 점검 및 발표준비 | |
| 12/07 | 테스트 및 발표준비 | |
| 12/08 | 최종 발표 | |

**[개발 일정 수정 전]**

|  |  |  |
| --- | --- | --- |
| **날짜** | **김동석** | **김영준** |
| 11/03 | GameFrameWork 제작 | SceneManager 제작 |
| 11/04 |
| 11/05 | Messege, Packet 관련 설계 |
| 11/06 |
| 11/07 | 개인 학습 | |
| 11/08 | 1주차 점검 및 회의 | |
| 11/09 | ClinetFrameWork 제작 | ServerFrameWork 제작 |
| 11/10 |
| 11/11 | 개인 학습 | |
| 11/12 | Framework 종합 | |
| 11/13 | GameScene 수정 | 원격 IP 접근 구현 |
| 11/14 | 개인 학습 | |
| 11/15 | 2주차 점검 및 회의 | |
| 11/16 | 기획내용 변경 및 구조 변경 | |
| 11/17 |
| 11/18 | 개인 학습 | |
| 11/19 | GameFramework 점검 | SendThread() |
| 11/20 |
| 11/21 | 개인 학습 | |
| 11/22 | 3주차 점검 및 회의 | |
| 11/23 | SendThread()  ResultScene 제작 | RecvThread() 패킷 분류 |
| 11/24 | RecvThread() 구현 |
| 11/25 | 개인 학습 | |
| 11/26 | SendThread() 최종 점검 | RecvThread() 최종 점검 |
| 11/27 |
| 11/28 | 개인 학습 | |
| 11/29 | 4주차 점검 및 회의 | |
| 11/30 | 동기화 작업 – (플레이어, 블록) | |
| 12/01 | ReGame 구현 | |
| 12/02 | 개인 학습 | |
| 12/03 | 최종 코드 점검 | |
| 12/04 |
| 12/05 | 개인 학습 + 테스트 | |
| 12/06 | 5주자 점검 및 발표준비 | |
| 12/07 | 테스트 및 발표준비 | |
| 12/08 | 최종 발표 | |

**[개발 일정 수정 후]**

1. 프로젝트를 마무리하며 느낀 점

**- 김동석**  
처음 추진계획서를 작성할 때 충돌처리와 게임오버 처리 등을 서버에서 구현하는 것으로 계획하였는데 2학년때 팀프로젝트로 작업한 게임을 이용해서 작업하다 보니 프레임워크나 게임구조에서 하드코딩이 되어있는 곳이 많아서 어려움을 겪었습니다. 프로젝트를 시작한 후 몇 주간 서버와 클라이언트에서 처리할 패킷에 넣을 흩어져 있는 정보들을 하나로 모으는 작업을 했지만 이미 하드코딩 되어 있는 코드이다 보니 변수나 함수 간에 얽혀 있는 부분들이 많아서 분리가 어려웠습니다. 이번 프로젝트를 진행하면서 가장 크게 느낀 점은 게임을 기획할 때 클라이언트와 서버가 주고받을 패킷을 고려해서 설계하는게 중요하다는 것 입니다. 기획단계에서 패킷이나 구조들을 설계를 잘해 놓으면 나중에 작업하기 훨씬 편하고 코드를 수정하는데 시간이 오래 걸리지 않기 때문입니다.

**- 김영준** 외부 IP 포트 개방에서 많은 시간을 사용하면서 서버 프로그래밍의 개념에 대해서 다시 생각해 볼 수 있는 기회가 있었습니다, 기존의 클라이언트 구조를 크게 바꾸지 않으려는 생각에 쓰레드를 하나만 사용한 클라이언트와 통신하는 서버를 구현해야 했는데 많은 애로사항이 있었습니다.

기존에 공부를 하거나 실습을 할 때 패킷을 switch-case문으로 처리했던 경험 때문에 익숙함을 잊지 못하고 최종 프로젝트에도 switch-case문으로 패킷을 처리했는데 결국 코드의 직관성을 해치는 문제가 있었습니다. 다른 프로젝트를 진행하게 된다면 클래스와 함수를 잘 사용하여 직관성이 뛰어난 코드를 만들겠습니다.