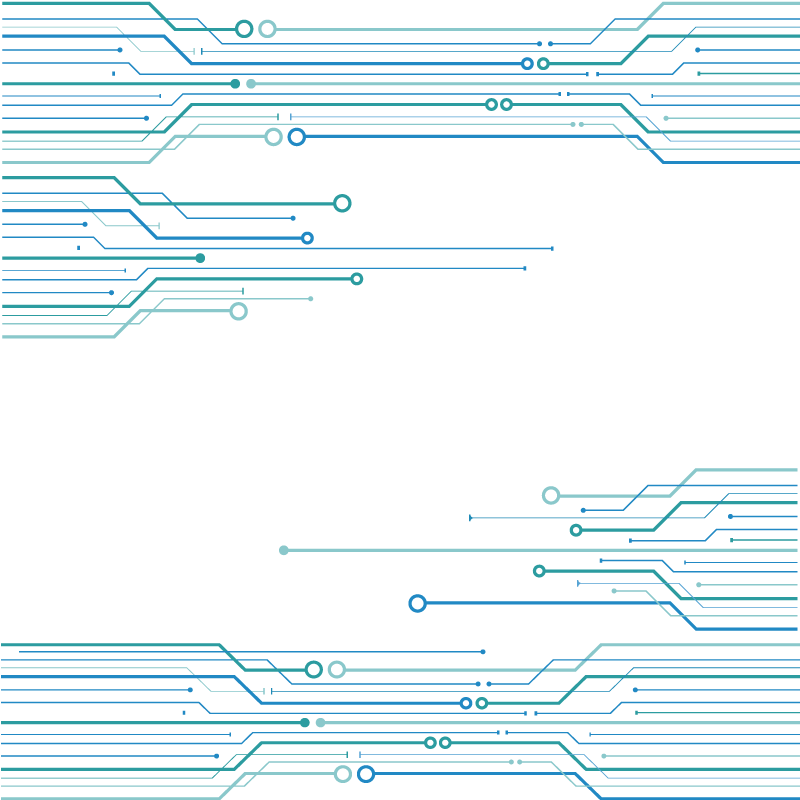
|  |  |
| --- | --- |
| **학번** | **이름** |
| **2015180033** | **이태훈** |
| **2015180036** | **장영진** |
| **2015182034** | **임종현** |

****

**네트워크 게임 프로그래밍**

**추진 계획서**

**[ 목 차 ]**

**1. 애플리케이션 기획 3**

**2. High-Level Design 6**

**2.1 클라이언트 구현 내용 6**

**2.2 서버 구현 내용 8**

**3. Low-Level Design 11**

**3.1 클라이언트 구현 내용 11**

**3.2 서버 구현 내용 16**

**4. 팀원 별 역할 분담 20**

**5. 개발 일정 21**

**6. 개발 환경 23**

1. **애플리케이션 기획**

* 구현 목표: 크레이지 아케이드 모작에 네트워크 통신 프로그래밍을 접목하여 2~3인의 플레이어들 간의 경쟁이 가능한 게임을 구현한다.
* 게임 이름: 크레이지 아케이드
* 게임 장르: 2D 캐주얼 아케이드 게임
* 게임 컨셉: 크레이지 아케이드는 물풍선을 이용해 모든 적 캐릭터들을 가두어서 터뜨리면 승리하게 되는 방식의 아케이드 게임이다.
* 게임 조작
  + →: 플레이어 오른쪽 방향 이동
  + ←: 플레이어 왼쪽 이동
  + ↑: 플레이어 위쪽 이동
  + ↓: 플레이어 아래쪽 이동
  + SPACE BAR: 물풍선 생성
* 게임 아이템:
  + 신발: 캐릭터의 이동 속도 증가
  + 물약: 캐릭터의 공격력 증가
  + 물풍선: 캐릭터가 생성가능한 물풍선 개수 증가
* 플레이어블 캐릭터:
  + 배찌
  + 우니
* 예시 스크린샷



<대기화면 예시>



<게임화면 예시>



<결과화면 예시>

**2. HIGH LEVEL 디자인**

* 1. **클라이언트 구현 내용**

1. **서버 연결**

* 콘솔창에 접속할 서버 IP주소를 입력해주면 접속할 서버를 결정하고, 윈도우 소켓을 초기화한 뒤 해당 서버에 connect함수로 연결을 요청한다.

1. **타이틀 화면**

* 클라이언트에서 아이디 및 패스워드를 입력하여 send함수를 통해 서버 측에 로그인 요청을 한다.
* 등록이 되어있지 않다면 계정을 등록한 뒤 로그인을 한다.
* 로그인에 성공한다면 로비화면으로 넘어간다.

1. **로비 화면**

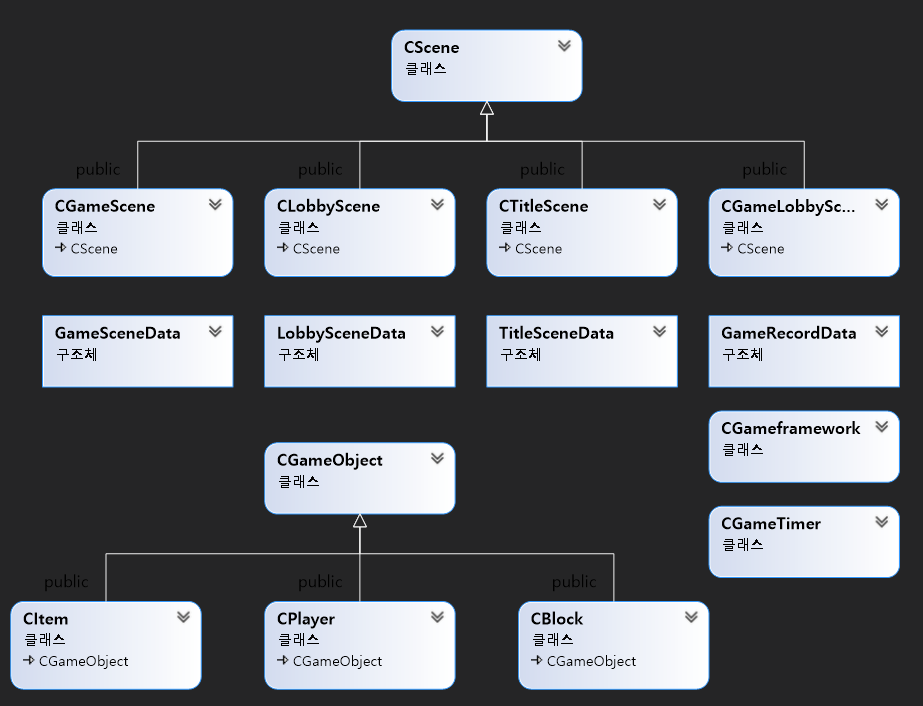
* 첫번째로 접속한 플레이어는 방장이 되어 접속한 유저들이 전부 준비완료 상태라면 게임을 시작할 수 있다.
* 게임플레이어들은 자신의 준비 상태를 서버에 전송한다.
* 로비화면에서 플레이어들은 채팅창을 통해 채팅정보를 send 함수를 통해 서버에 전송한다.
* 플레이어의 준비가 모두 완료되면 첫번째로 접속한 플레이어가 게임 시작 요청을 서버에 전송한다
* 각 클라이언트는 서버로부터 게임 시작 정보를 받으면 게임플레이 화면으로 넘어간다.

1. **게임 플레이 화면**

* 게임이 시작되어 입장한 각 플레이어는 매 업데이트 함수 호출 시마다 자신의 정보를 서버에 전달한다.
* 서버에서 처리한 정보를 받아 플레이어 및 게임 맵 등의 정보를 갱신하여 화면에 그린다.
* 게임 종료 조건을 만족한다면 게임 결과 화면으로 넘어간다.

1. **결과 화면**

* 각 클라이언트에서 자신의 결과에 대해서 종합하여 서버에 전송한다.
* 서버에서 받은 결과값에 대한 정보를 결과창에 출력한다.
* 게임 결과 출력이 끝나면 3번의 로비화면으로 넘어간다.



<예상 클라이언트 클래스 구조>

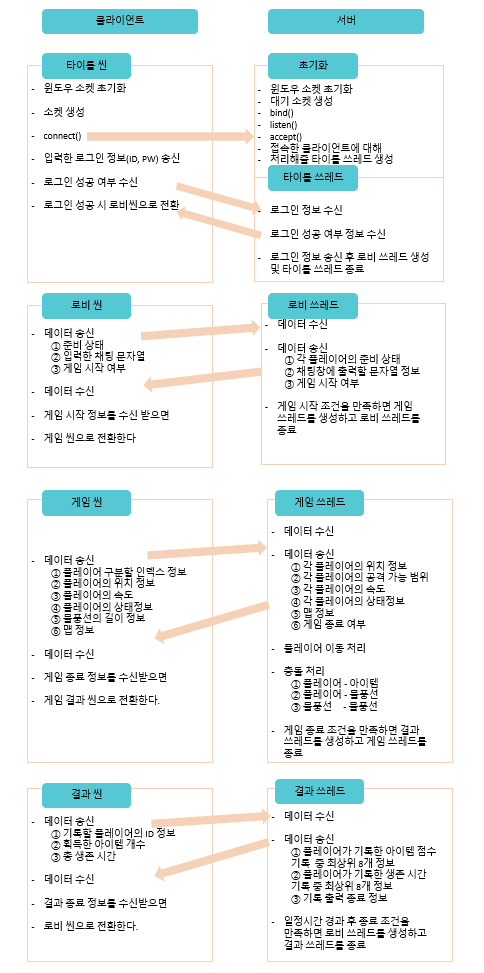
* 1. **서버 구현 내용**

**1. 서버 실행**

* 서버를 실행 시 윈도우 소켓 초기화 및 클라이언트의 접속을 처리해 줄 대기용 소켓 소켓을 생성하여 서버의 IP주소와 포트번호를 bind함수로 결정한다.
* 클라이언트의 접속을 기다리기 위한 대기용 소켓의 TCP포트 상태를 listen함수를 이용하여 Listening상태로 설정한다.
* 클라이언트가 실행되어 connect함수를 통해 접속 요청을 하면 이를 처리할 소켓을 생성한다.
* 이를 처리할 쓰레드를 만들기 위해서 전달할 함수, 전달해준 인자 값을 결정하여 쓰레드를 생성한다.

1. **플레이어 별 스레드를 이용한 데이터 통신**

* **타이틀 화면**
  + 클라이언트가 send함수로 송신한 ID 및 PW를 recv함수로 수신한다.
  + 로그인정보가 등록되어 있다면 true값, 아니라면 false값을 클라이언트에 send함수로 송신한다.
  + 클라이언트가 계정등록 전송한 ID 및 PW값으로 계정등록을 요청하면 계정을 등록한다.
* **로비 화면**
  + 각 클라이언트로부터 자신의 준비완료, 대기 상태를 recv함수로 수신한다.
  + 서버에서는 이를 종합하여 클라이언트에 각 플레이어의 준비 상태를 send함수로 송신한다.
  + 각 클라이언트로부터 채팅 정보를 recv함수로 수신하여 결과값을 종합한 뒤 채팅창에 출력할 문자열을 send함수로 송신한다.
  + 첫번째로 접속한 클라이언트가 게임 시작 요청을 하면 서버에서 각 클라이언트에 게임 시작을 알리는 정보를 send함수를 통해 송신한다.
* **게임플레이 화면**
  + 각 클라이언트가 Update마다 보내는 게임 정보를 recv함수로 수신한다.
  + update함수의 균등한 처리를 위하여 event를 이용한다.
  + 처리할 내용
    1. 아이템 획득 정보
    2. 공격 정보
    3. 이동 정보
  + 하나의 플레이어만 남은 경우 클라이언트들에 게임 종료를 알리는 정보를 send함수를 통해 송신한다.
* **게임 결과 화면**
  + 각 클라이언트에서 send함수를 통해 수시한 결과 정보를 집계한다.
  + 갱신된 최고 기록(아이템 획득 기록, 살아있었던 시간)이 있으면 기록한다.
  + 각 클라이언트에 집계한 기록 결과를 send함수로 송신한다.

<데이터 통신 플로우 차트>

**3. LOW LEVEL 디자인**

**3-1. 클라이언트 구현 내용**

**1. [클라이언트] 서버 연결 공통 관련 함수**

* + **void error\_quit(const char\* msg)**
    1. const char\* msg : 받을 메시지
    2. 통신 중 에러가 발생한 경우 이를 출력하고 프로그램을 종료한다.
  + **void error\_display(const char\* msg)**
    1. const char\* msg: 받을 메시지
    2. 통신 중 에러가 발생한 경우 이를 출력하되, 프로그램을 종료하지 않는다.
  + **void recvn(SOCKET s, const char\* buf, int len, int flag)**
    1. TCP프로토콜의 데이터 경계를 구분하지 않는 특성으로 인해 인자로 전달된 len값 만큼 recv함수가 읽어오지 못할 수 있다. 이를 방지하기 위한 사용자 정의 함수
  + **void prepareCommunicate()**
    1. 사용할 변수
       - * WSADATA wsa: 클라이언트에서 사용할 윈도우 소켓
         * SOCKET sock: 서버와 통신할 소켓
         * SOCKADDR serverAddr: 통신할 서버의 IP주소 및 포트번호를 담은 소켓 주소 구조체
    2. 클라이언트가 서버와 통신하기 위한 기초 작업들을 수행하는 함수

**2. [클라이언트] 화면 별 공통 함수 인터페이스**

* + void processInput()
  + void update(): 필요한 업데이트 작업을 수행 후, sendData함수와 recvData함수를 호출하여 서버와 통신한다.
  + void collisionCheck(): GameObject간 충돌 처리를 위한 함수
  + void draw(): 각 화면을 그리는 함수
  + void sendData(): 각 화면클래스에서 정의한 데이터 구조체를 송신하기 위한 함수
  + void recvData(): 각 화면클래스에서 정의한 데이터 구조체를 수신하기 위한 함수
  + void sceneChange(): 다른 화면으로 넘어가기 위한 함수

**3. [클라이언트] 타이틀 화면**

* + **송신할 데이터 구조체[TitleSceneData]**
    1. string id, pw: ID, PW 문자열
    2. bool isNewID: 계정 신규 등록 여부
  + **중심 함수**
    1. void Draw(): 타이틀 화면을 출력한다.
    2. void update(): 서버와 통신한 데이터를 통해 로그인에 성공하였으면 sceneChange함수를 통해 로비화면으로 넘어간다.
    3. void processInput()
       - * Point2D mouseInput: 마우스 입력값을 받아 저장한다.
         * String ID, PW: 키보드 입력값을 받아 저장한다.
         * 마우스 입력값이 로그인 버튼 좌표이면 입력한 ID, PW정보를 서버에 송신한다.
         * 키보드 입력값을 통해 ID, PW값을 기록한다.

**4. [클라이언트] 로비화면**

* + **송신할 데이터 구조체[LobbySceneData]**
    1. bool isReady: 플레이어의 준비 상태
    2. string chatData: 플레이어가 입력한 채팅 문자열
    3. bool isGameStart: 게임의 시작 여부
  + **중심 함수**
    1. void draw(): 로비화면과 각 플레이어들의 채팅을 출력한다.
    2. void update():
       - * 서버와 통신하여 채팅창에 출력할 내용을 받는다.
         * 서버와 통신하여 각 플레이어들의 대기 상태 정보를 받는다.
         * 서버와 통신하여 게임이 시작했다면 sceneChange함수를 통해 GameScene으로 넘어간다.
    3. void processInput()
       - * Point2D mouseInput: 마우스 입력값을 받아 저장한다.
         * string chatData: 키보드 입력값을 받아 저장한다.
         * 마우스 입력값이 게임 시작버튼 일 때, 서버에 송신할 정보의 isGameStart값을 true값으로 설정한다.
         * 키보드 입력값을 채팅 메시지를 기록한다.

**5. [클라이언트] 게임 플레이 화면**

* + **송신할 데이터 구조체[GameSceneData]**
    1. int playerIndex: 플레이어들을 구분해줄 정보
    2. Point2D position: 플레이어의 위치 정보
    3. int waterRange: 플레이어의 공격 가능 범위
    4. int speed: 플레이어의 속도
    5. int state: 플레이어의 상태정보
    6. int mapData[width][height]: 맵 정보
  + **중심 함수**
    1. void draw(): 화면을 렌더링한다.
    2. void update():
       - * 플레이어들의 캐릭터를 애니메이션 한다.
         * 서버와 통신하여 각 플레이어들의 정보를 받는다.
         * 서버와 통신하여 한 명 이하의 플레이어가 생존하였다면 sceneChange함수를 호출하여 GameRecordScene으로 넘어간다.
    3. void processInput()
       - * String input: 입력값을 받아 저장한다.
         * ↓↑←→: 플레이어의 이동 방향을 설정한다.
         * Space bar: 플레이어의 현재 좌표에 위치한 맵에 물풍선을 설치한다.

**6. [클라이언트] 게임 결과 화면**

* + **송신할 데이터 구조체[GameRecordData]**
    1. string ID: 자신의 ID정보
    2. int itemScore: 획득한 아이템 개수
    3. float survivedScore: 총 생존 시간
  + **중요 함수**
    1. void draw(): 게임 결과를 화면에 출력한다.
    2. void update()
       - * 서버와 통신하여 각 클라이언트의 기록을 송신하고, 전체 기록을 수신 받는다.

**3-2. 서버 구현 내용**

**1. [서버] 서버 연결 공통 관련 함수**

* + **void error\_quit(const char\* msg)**
    1. const char\* msg: 받을 메시지
    2. 통신 중 에러가 발생한 경우 이를 출력하고 프로그램을 종료한다.
  + **void error\_display(const char\* msg)**
    1. const char\* msg: 받을 메시지
    2. 통신 중 에러가 발생한 경우 이를 출력하되 프로그램을 종료하지 않는다.
  + **void recvn(SOCKET s, const char\* buf, int len, int flag)**
    1. TCP프로토콜의 데이터 경계를 구분하지 않는 특성으로 인해 인자로 전달된 len값 만큼 recv함수가 읽어오지 못할 수 있다. 이를 방지하기 위한 사용자 정의 함수
  + **void saveFile(string filename, vector<string>fileData)**
    1. 인자로 전달받은 fileName을 통해 파일을 열고, fileData값을 저장한다.
    2. 사용자 ID, PW 관리와 플레이어의 점수 기록을 이 함수를 통해 수행한다.

**2. [서버] 타이틀 화면**

* + **DWORD WinAPI TitleThread(LPVOID arg)**
    1. 클라이언트와 수신한 ID, PW정보가 계정 정보파일에 존재한다면 로그인 성공 값 송신
    2. 클라이언트가 ID, PW정보로 새로운 계정 등록 요청을 한다면 계정 정보 파일에 해당 정보를 기록한다.
  + **송신할 데이터 구조체[TitleSceneData]**
    1. bool isCanLogin: 로그인 성공 여부 정보

**3. [서버] 로비** **화면**

* + **DWORD WinAPI LobbyThread(LPVOID arg)**
    1. 각 클라이언트가 송신한 데이터의 isReady값이 모두 true값이라면 isGameStart를 true로 설정한다.
    2. 각 클라이언트가 전송한 chatData를 통합하여 전송한다.
  + **송신할 데이터 구조체[LobbySceneData]**
    1. bool isReady: 플레이어의 준비 상태
    2. string chatData: 채팅창에 출력할 문자열 정보
    3. bool isGameStart: 게임의 시작 여부

**4. [서버] 게임 플레이 화면**

* + **DWORD WinAPI GameSceneThread(LPVOID arg)**
    1. void playerUpdate()
       - * 클라이언트가 송신한 상태 정보와 플레이어의 위치 정보를 이용하여 플레이어의 위치정보를 갱신한다.
    2. void collisionCheck()
       - * 플레이어와 아이템:

플레이어와 아이템 간의 충돌처리를 구현한다. 여러 플레이어가 한 아이템과 동시에 겹친다면 아이템과 더 가까운 플레이어가 아이템을 획득한 것으로 처리한다.

플레이어가 획득한 아이템이 능력치에 변동을 준다면 해당 사항을 기록한다.

* + - * + 플레이어와 물풍선:

플레이어와 물풍선 간의 충돌처리를 구현한다.

* + - * + 물풍선과 물풍선: 물풍선의 공격 범위 안에 다른 물풍선이 하는 지 확인하고 연쇄적으로 터지도록 값을 변경한다.
  + **송신할 데이터 구조체[GameSceneData]**
    1. Point2D position: 플레이어의 위치 정보
    2. int waterRange: 플레이어의 공격 가능 범위
    3. int speed: 플레이어의 속도
    4. int state: 플레이어의 상태정보
    5. int mapData[width][height]: 맵 정보
    6. bool isGameEnd: 게임이 끝났는지 여부에 대한 정보

**5. [서버] 게임 결과 화면**

* + **DWORD WinAPI GameRecordThread(LPVOID arg)**
    1. 각 클라이언트가 송신한 기록 값들과 이전에 파일에 저장한 기록값들과 비교하여 최상위 값이 변하였다면 이를 갱신한다.
    2. 각 클라이언트에 기록 값을 전송한다.
    3. 일정시간이 경과하면 송신할 데이터 구조체의 isNewGame 변수 값을 true값으로 설정한다.
  + **송신할 데이터 구조체[GameRecordData]**
    1. String itemScore: 플레이어가 기록한 아이템 점수 중 최상위 8개
    2. string survivedScore: 플레이어가 기록한 생존 시간 기록 중 최상위 8개
    3. bool isNewGame: 기록 출력을 종료하고 로비 화면으로 돌아갈지에 대한 정보

**4. 팀원 별 역할 분담**

* 클라이언트와 서버에서 각 화면(Scene)처리 부분을 각자 맡아서 구현한다.
* 난이도 적으로 어려운 LobbyScene, GameScene을 한 명씩 맡아서 구현하고,
* 난이도 적으로 쉬운 TitleScene, RecordScene을 한 명이 맡아서 구현한다.
* 공통함수를 LobbyScene, GameScene 처리를 맡은 사람에게 분배하여 구현한다.

|  |  |  |
| --- | --- | --- |
| 이태훈 | 장영진 | 임종현 |
| 게임 구현  [CObject Class] | 게임 구현  [CScene class] | 게임 구현  [CPlayer class] |
| 게임 구현  [CFramework class] | 게임 구현  [CBomb class] | 게임 구현  [CItem Class] |
| 클라이언트 구현  [CTitleScene] | 클라이언트 구현  [CLobbyScene class] | 클라이언트 구현  [CGameScene class] |
| 서버 구현  [CTitleScene class] | 서버 구현  [CLobbyScene class] | 서버 구현  [CGameScene class] |
| 클라이언트 구현  [CRecordScene class] | 클라이언트 구현  [공통함수들] | 서버 구현  [공통함수들] |
| 서버 구현  [CRecordScene class] |  |  |

1. **개발 일정**

**5-1) 이태훈 개발일정**

|  |  |  |
| --- | --- | --- |
| 1주차 | 11/2 | 게임 구조 설계 및 기획서 작성 |
| 11/3 ~ 11/4 | 게임 구현 [CObject Class] |
| 11/5 ~ 11/6 | 게임 구현 [CFramework Class] |
| 2주차 | 11/10 | 게임 테스트 및 버그 수정 |
| 11/11 ~ 11/12 | [CTitleScene] 클라이언트 키보드/마우스 입력 구현 |
| 11/13 ~ 11/14 | 클라이언트 ID/PW 송신 및 신규 등록 요청 구현 |
| 3주차 | 11/17 ~ 11/18 | [CTitleScene class] 서버 ID/PW 수신 기능 구현 |
| 11/19 ~ 11/20 | 수신 받은 계정 정보 파일로 저장 및 신규 등록 구현 |
| 11/21 | 서버 로그인 허용 여부 전송 기능 구현 |
| 4주차 | 11/23 ~ 11/24 | [CRecordScene class] 유저 점수 기록 데이터 구조체 구현 |
| 11/25 ~ 11/26 | 클라이언트 유저 점수 데이터 송신 구현 |
| 11/27 ~ 11/28 | 서버 유저 점수 데이터 파일 저장 기능 구현 |
| 5주차 | 12/1 ~ 12/2 | 서버 유저 점수 데이터 기록 갱신 구현 |
| 12/3 ~ 12/4 | 서버 유저 점수 데이터 송신 기능 및 로비로 복귀 구현 |
| 12/5 ~ 12/6 | 재검토 및 다른 팀원 서포트 |
| 6주차 | 12/7 | 코드 일체화 작업 |
| 12/9 ~ 12/11 | 최종 디버깅 작업 |
| 12/12 | 보고서 준비 |

**5-2) 장영진 개발일정**

|  |  |  |
| --- | --- | --- |
| 1주차 | 11/2 | 게임 구조 설계 및 기획서 작성 |
| 11/5 ~ 11/6 | 게임 구현[CScene class] |
| 11/7 ~ 11/8 | 게임 구현[CBomb class] |
| 2주차 | 11/9 | 게임 테스트 및 시연 |
| 11/11 ~ 11/12 | 클라이언트 및 서버의 연결 관련 공통 함수 구현 |
| 11/13 ~ 11/15 | Scene별 공통 함수의 인터페이스 및 기능 구현 |
| 3주차 | 11/16 ~ 11/18 | 클라이언트 데이터 구조체 및 로비화면 구현 |
| 11/19 ~ 11/20 | 서버의 데이터 구조체 및 해당 구조체의 송수신 구현 |
| 11/21 ~ 11/22 | 클라이언트와 서버의 채팅 기능 구현 |
| 4주차 | 11/23 ~ 11/25 | 클라이언트와 서버의 채팅 기능 구현 |
| 11/27 | 마우스 입력 기능 처리 및 해당 데이터의 송수신 구현 |
| 11/28 ~ 11/29 | 플레이어 상태 별 데이터 송수신 및 처리 구현 |
| 5주차 | 11/30 | 플레이어 상태 별 데이터 송수신 및 처리 구현 |
| 12/2 ~ 12/3 | 플레이어 상태 별 데이터 송수신 및 처리 구현 |
| 12/4 ~ 12/5 | 재검토 및 다른 팀원 서포트 |
| 6주차 | 12/7 | 코드 일체화 작업 |
| 12/9 ~ 12/11 | 최종 디버깅 작업 |
| 12/12 | 보고서 준비 |

**5-3) 임종현 개발일정**

|  |  |  |
| --- | --- | --- |
| 1주차 | 11/2 | 게임 구조 설계 및 기획서 작성 |
| 11/4 ~ 11/5 | 게임 구현[CPlayer class], |
| 11/7 ~ 11/8 | 게임 구현[CItem class] |
| 2주차 | 11/9 | 1주차 종합, 변경 내용 기획서 기록 |
| 11/11 ~ 11/12 | 클라이언트 및 서버의 연결 관련 공통 함수 구현 |
| 11/14 ~ 11/15 | Scene별 공통 함수의 인터페이스 및 기능 구현 |
| 3주차 | 11/16 | 2 주차 종합, 변경 내용 기획서 기록 |
| 11/18 ~ 11/19 | [GameScene class]클라이언트에서 서버로의 데이터 전송 구현 |
| 11/21 ~ 11/22 | [GameScene class]서버에서 클라이언트 위치정보 갱신 구현 |
| 4주차 | 11/23 | 3주차 종합, 변경 내용 기획서 기록 |
| 11/25 ~ 11/26 | [GameScene class]서버에서 플레이어-아이템 충돌처리 구현 |
| 11/28 ~ 11/29 | [GameScene class]서버에서 플레이어-물풍선 충돌처리 구현 |
| 5주차 | 11/30 | 4주차 종합, 변경 내용 기획서 기록 |
| 12/2 ~ 12/3 | [GameScene class] 서버 – 클라이언트 데이터 전송 구현 점검 |
| 12/5 ~ 12/6 | 미 구현 사항 점검 및 다른 팀원 서포트 |
| 6주차 | 12/7 | 코드 일체화 작업 |
| 12/9 ~ 12/10 | 최종 디버깅 작업 |
| 12/11 ~ 12/12 | 5 ~ 6주차 종합, 변경 내용 기획서 기록 |

1. **개발 환경**

- 통합개발환경 IDE: Visual Studio 2019

- 사용하는 언어: C/C++

- 이미지 처리: Photoshop

- 사운드처리: FMOD

- 네트워크 처리: TCP/IP

- 버전관리: Git