네트워크 게임 프로그래밍

텀프로젝트 Progress Report

**<2팀>**

2018180048 박의인

2017184037 홍진선

2017184030 정의범

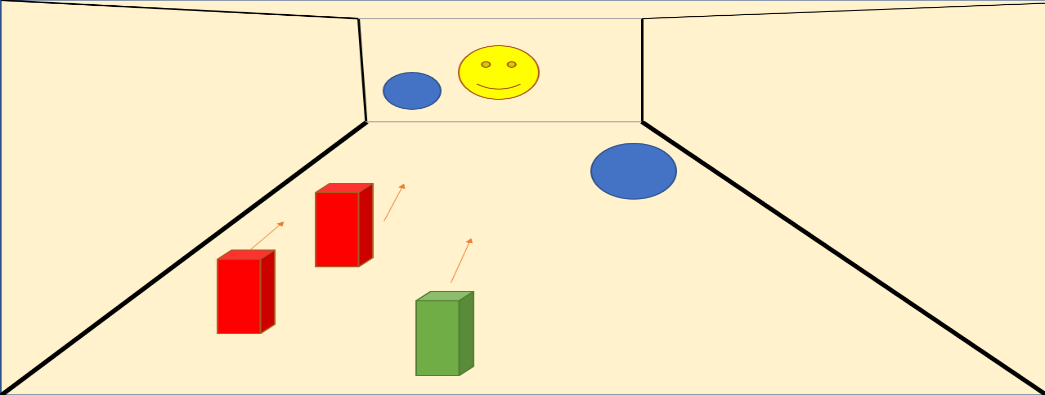
목차

|  |
| --- |
| 1. 애플리케이션 기획  2. High-Level 디자인  3. Low-Level 디자인  4. 팀원 별 역할 분담  5. 개발 환경  6. 개발 일정 |

1. 애플리케이션 기획

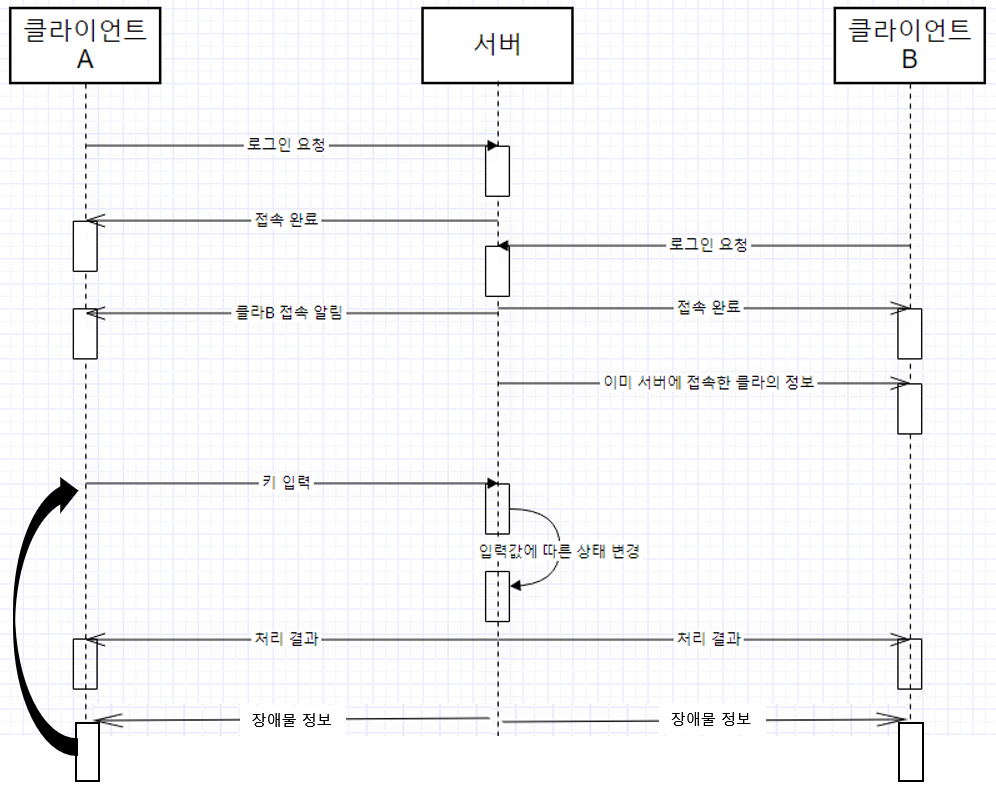
게임 ‘테일즈 런너’의 게임 모드 중 하나인 도마뱀 삼촌의 위기를 간단히 구현하여 네트워크 요소를 추가해 본다.

* 키: 방향키(앞뒤좌우 이동)
* 게임의 흐름: 클라이언트에서 서버에 연결하면 시작 위치에서 대기한다. 일정 수의 플레이어가 접속하면 게임 시작. 멀어지는 적을 향해 다가가야 한다. 이 때 앞으로 가는 키를 누르고 있으면 일정 수준까지 속도가 점점 증가한다. 뒤로 가는 키를 누를 시 속도가 감소한다. 앞에서 날아오는 장애물들을 피해 앞으로 전진해야 한다. 장애물에 맞을 경우, 속도가 느려져 뒤에서 따라오는 장애물에 부딪히는 경우 사망하게 된다.
* 승리 조건: 먼저 도착지점에 도착한 플레이어가 승리한다. 만약 도착지점에 도착하지 못한 상태에서 플레이어가 모두 사망하게 되면, 가장 앞에서 죽은 플레이어가 승리한다.
* 플레이어의 구분: 자신 캐릭터만 초록색, 나머지 플레이어는 빨간색으로 설정한다.
* 예상 게임 화면(그래픽 요소는 차이가 있음)



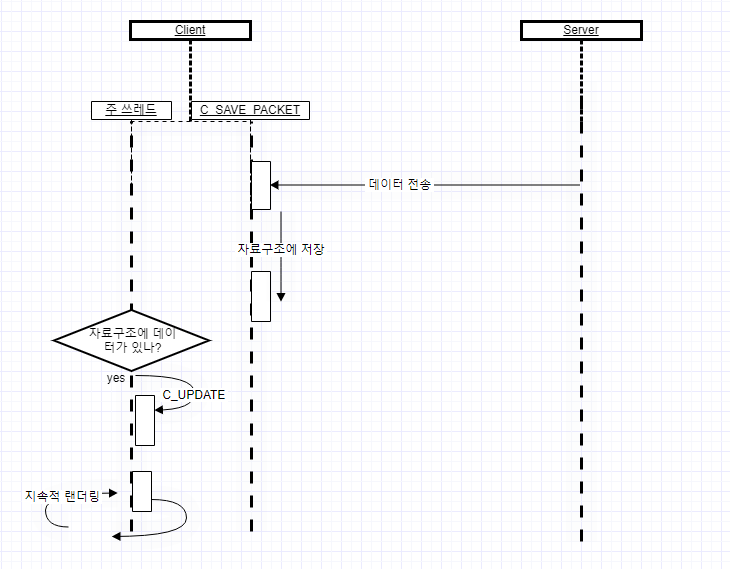
2. High-Level 디자인

- 클라-서버 간 데이터 전송 과정



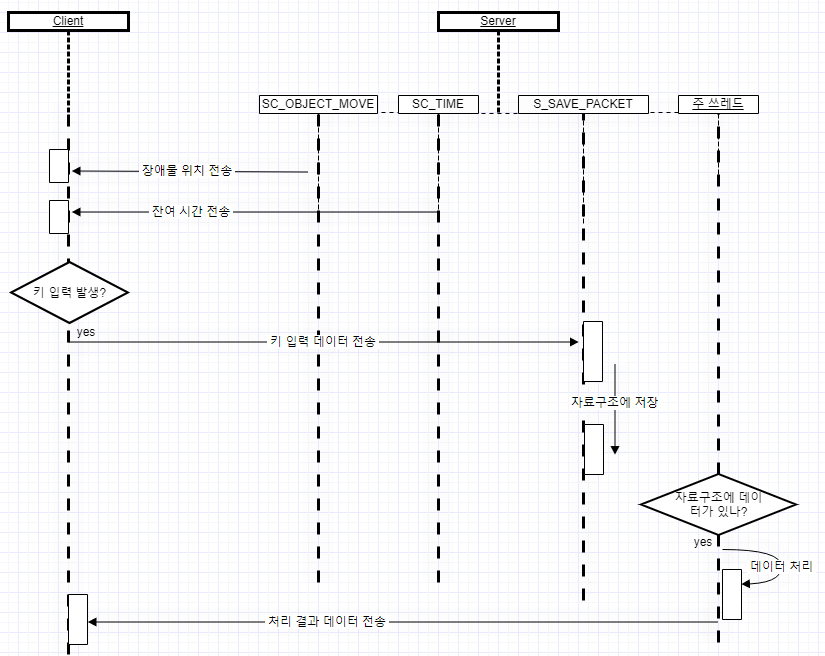
|  |  |
| --- | --- |
| 클라이언트 | 서버 |
| 1. 서버에 로그인 요청  2. 전달받은 데이터를 바탕으로 화면에 오브젝트 생성.  3. 키 입력 시 서버에게 클라이언트 데이터구조체 전송.  4. 전달받은 처리 결과 데이터를 반영.  5. 장애물 정보를 받아서 처리. | 1. 클라이언트의 로그인 요청 수락.  2. 클라이언트에 ID 부여.  3. 접속 중인 모든 클라이언트에게 새로 접속한 클라이언트의 정보를 전달.  4. 전달받은 데이터 처리 후 결과를 모든 클라이언트에게 전송.  5. 장애물 정보를 모든 클라이언트에게 지속적인 전송. |

- 클라이언트 내의 스레드와 서버 간 데이터 전송



|  |  |
| --- | --- |
| 클라이언트 | 서버 |
| 1. C\_SAVE\_PACKET 쓰레드에서 서버에서 받은 데이터를 자료구조에 저장.  2. 주 쓰레드에서 자료구조 내에 데이터가 존할 경우 C\_UPDATE 함수 호출해서 데이터를 업데이트한다.  3. 주 쓰레드에서 지속적으로 랜더링이 일어난다. | 1. 서버에서 처리한 데이터 전송 |

- 서버 내의 스레드와 클라이언트 간 데이터 전송



|  |  |
| --- | --- |
| 클라이언트 | 서버 |
| 1. 키 입력 발생 시 데이터를 서버로 전송 | 1. SC\_OBJECT\_MOVE 쓰레드와 SC\_TIME 쓰레드에서 장애물 위치와 잔여시간을 각각 클라이언트로 전송  2. S\_SAVE\_PACKET 쓰레드에서 클라이언트에서 받은 데이터를 자료구조에 저장  3. 자료구조에 데이터가 있다면, 데이터를 처리하고 결과를 클라이언트에 전송 |

3. Low-Level 디자인

수정한 내용: 파란색

삭제한 내용: ~~빨간색 줄~~

|  |  |
| --- | --- |
| **헤더 파일** | |
| #pragma once  // CLIENT DATA TYPE  ~~#define~~ -> constexpr unsigned char  MOVE\_FRONT = 0x01;  MOVE\_BACK = 0x02;  MOVE\_RIGHT = 0x03;  MOVE\_LEFT = 0x04;  LOGOUT = 0x05;    // SERVER DATA TYPE  ~~#define~~ -> constexpr unsigned char  LOCATION = 0x11;  LOGIN = 0x12;  GAME\_START = 0x13;  GAME\_OVER = 0x14;  GAME\_RESULT = 0x15;  TIME = 0x16;  REJECT = 0x17;  RESTART = 0x18;  // LOCATION 하위 TYPE  ~~#define~~ -> constexpr unsigned char  PLAYER = 0x21;  OBJECT = 0x22;  //LOGIN 하위 SUBDATA TYPE  constexpr unsigned char  SELF = 0x31;  OTHER = 0x32  //OBJECT DATA TYPE  constexpr unsigned char  BOSS = 0x41;  TRACKER = 0x42;  BALL = 0x43;  BULLDOZER = 0x44;  enum STATE { NONE, FULL };  #pragma pack(push,1)  // DATA 형식  struct SERVER\_DATA {  unsigned char dataType;  unsigned char subDataType;  OBJECT\_INFO objectInfo;  unsigned short id;  ~~bool mission\_result;~~  float x, y, z;  int time;  };  // CLIENT가 보내는 DATA 형식  struct CLIENT\_DATA {  unsigned char type;  unsigned short id;  };  // CLIENT INFO  struct CLIENT\_INFO {  SOCKET sock;  STATE state;  unsigned short id;  bool alive;  float x, y, z;  };  struct OBJECT\_INFO {  unsigned char objectType;  bool moving;  unsigned short line;  unsigned short id;  float x, y, z;  };  #pragma pack(pop) | **키 입력 타입**  **(클라이언트->서버)**  MOVE\_FRONT: 앞으로 이동  MOVE\_BACK: 뒤로 이동  MOVE\_RIGHT: 오른쪽으로 이동  MOVE\_LEFT: 왼쪽으로 이동.  LOGOUT: 클라이언트의 로그아웃  **서버 데이터 타입**  **(서버->클라이언트)**  LOCATION: 위치를 변경  LOGIN: 클라이언트 로그인  GAME\_START: 게임 시작  GAME\_OVER: 게임 종료  GAME\_RESULT: 게임 결과  TIME: 남은 시간  REJECT: 로그인 거절  RESTART: 게임 재시작  (LOCATION의 주체)  PLAYER: 플레이어  OBJECT: 장애물  SELF: 받은 데이터가 자신의 것  OTHER: 받은 데이터가 다른 유저의 것  BOSS: 전방의 몬스터  TRACKER: 후방에서 쫓아오는 장애물  BALL: 전방에서 날아오는 장애물  BULLDOZER: 전방에서 날아오는 장애물  각 ID가 사용중인지, 아닌지 서버에서 판단  **서버 데이터 구조체**  **(서버->클라이언트)**  dataType: 위에 정의한 서버 데이터 타입  subDataType: location의 대상  objectInfo: 오브젝트 전달  id: 서버가 클라이언트에 부여한 ID  ~~mission\_result: 클라이언트들이 도착지점에 도착여부~~  x, y, z: 캐릭터의 위치 값  time: 시간  **클라이언트 데이터 구조체**  **(클라이언트->서버)**  Type: 위에 정의한 키 입력 타입  id: 서버가 부여해준 ID  **클라이언트 데이터**  SOCK: 클라이언트의 소켓  STATE: ID를 할당 여부  id: 클라이언트의 ID  alive: 생존여부  x, y, z: 캐릭터의 위치 값.  **오브젝트 데이터**  objectType: 오브젝트 타입  moving: 움직이는 중인지 여부  line: 오브젝트의 생성 위치  id: 오브젝트의 id  x, y, z: 오브젝트의 위치 값 |

|  |  |
| --- | --- |
| **클라이언트** | |
| 함수 정의 | 함수 설명 |
| DWORD WINAPI C\_SAVE\_PACKET()  {  SERVER\_DATA server\_data;  while (true) {  recv(server\_data);  ~~lock();~~  ~~queue.input(server\_data);~~  ~~unlock();~~  network->C\_UPDATE(server\_data);  }  } | 서버에서 전송한 데이터를 전달받아 ~~queue자료구조에 저장하는 스레드 함수.~~  ~~자료구조에 저장 시 mutex를 동해 스레드 동기화.~~  C\_UPDATE함수에 전달 |
| void CS\_MOVE()  {  CLIENT\_DATA client\_data;  client\_data.id = m\_id;  ~~switch (key) {~~  ~~case VK\_UP:~~  ~~client\_data.id = id;~~  ~~client\_data.type = MOVE\_FRONT;~~  ~~send(client\_data);~~  ~~break;~~  ~~case VK\_DOWN:~~  ~~client\_data.id = id;~~  ~~client\_data.type = MOVE\_BACK;~~  ~~send(client\_data);~~  ~~break;~~  ~~case VK\_RIGHT:~~  ~~client\_data.id = id;~~  ~~client\_data.type = MOVE\_RIGHT;~~  ~~send(client\_data);~~  ~~break;~~  ~~case VK\_LEFT:~~  ~~client\_data.id = id;~~  ~~client\_data.type = MOVE\_LEFT;~~  ~~send(client\_data);~~  ~~break;~~  ~~default:~~  ~~break;~~  ~~}~~  If(GetAsyncKeyState( )){  Send(client\_data);  }  } | 입력한 키의 값이 무엇인지 판단하고, 그에 맞는 id, type값을 설정하여 서버에 전송하는 함수  클라이언트가 입력한 키의 값에 따라서 클라이언트의 Id와 입력한 key type을 서버에 전송한다.  이동키의 동시 입력을 위해서 if문을 사용 |
| void C\_UPDATE(SERVER\_DATA server\_data)  {  switch (server\_data.type)  {  case LOCATION:  switch(server\_data.subDatatype){  case PLAYER:  case OBJECT:  }  case LOGIN:  switch(server\_data.subDatatype){  case SELF:  case OTHER:  }  case LOGOUT:  case GAME\_START:  case TIME:  case GAME\_OVER:  case GAME\_CLEAR:  case REJECT:  case RESTART:  ~~case RESULT:~~    }  } | SC\_RECV 쓰레드에서 큐에 저장한  서버에서 보내준 데이터가 존재할 경우, 실행되는 함수.  LOCATION 클라이언트의 위치 갱신  PLAYER: 플레이어의 위치  OBJECT: 장애물의 위치  LOGIN 각 클라이언트의 초기위치 지정  SELF: 나 자신  OTHER: 다른 유저  LOGOUT 각 클라이언트의 로그아웃  GAME\_START 3명의 클라이언트 접속시 게임 시작  TIME 남은 시간 갱신  GAME\_OVER 게임 종료  GAME\_CLEAR 게임 클리어  REJECT 접속 거부  RESTART 게임 재시작 |
| ~~void OPEN\_RESULT\_SCENE(bool mission\_result)~~  ~~{~~  ~~If(mission\_result){~~  ~~}~~  ~~else{~~  ~~}~~  ~~}~~ | ~~도착지점에 도착 여부를 확인해 결과창을 띄우는 함수~~ |
| void DrawScene()  {  CS\_MOVE();  } | 주 쓰레드에서 실행되는 그리기 함수  플레이어의 키 입력을 받아 서버로 보낸다. |
| void CS\_START()  {  CLIENT\_DATA client\_data;  client\_data.id = m\_id;  client\_data.type = GAME\_START;  send(client\_data);  } | 스타트 버튼 클릭 시 서버에 GAME\_START를 전송하는 함수 |

|  |  |
| --- | --- |
| **서버** | |
| 함수 정의 | 함수 설명 |
| DWORD WINAPI S\_RECV\_PACKET()  {  CLIENT\_DATA client\_data;  while (true) {  recv(client\_data);  ~~lock();~~  ~~queue.input(client\_data);~~  ~~unlock();~~  SC\_SEND(client\_data);  }  } | 클라이언트에서 전송한 데이터를 전달받아 ~~queue자료구조에 저장하는 스레드 함수.~~  ~~자료구조에 저장 시 mutex를 동해 스레드 동기화.~~  SC\_SEND함수에 전달 |
| DWORD WINAPI SC\_TIME()  {  server\_Data.type = TIME;  server\_Data.time = 100;  while (true) {  for (모든 유저)  send(server\_Data);  server\_Data.time--;  }  } | 모든 클라이언트에게 현재 남은 시간을 전송하는 쓰레드 함수 |
| DWORD WINAPI SC\_OBJECT\_MOVE()  {  SERVER\_DATA server\_data;  server\_data.dataType = LOCATION;  while(true){  for(모든 오브젝트){  server\_data.id = i;  server\_data.x += dx;  server\_data.y += dy;  server\_data.z += dz;  for (int i = 0; i < size(clients); ++i)  send(server\_data);  COLL\_CHECK();  If(RESET\_OBJECT() == true){  for(모든 유저){  send(server\_data);  }  }  }  }  } | 장애물의 종류와 변경 위치 정보를 각 클라이언트에게 전달하는 함수  장애물의 위치 정보를 전달 후 모든 클라이언트와 장애물의 충돌 여부를 확인하는 COLL\_CHECK함수를 호출.  장애물 위치의 초기화 여부를 확인하는 RESET\_OBJECT함수를 호출해 장애물 변경 위치 정보를 각 클라이언트에게 전달 |
| void GOAL\_CHECK(CLIENT\_DATA client\_data)  {  if (clients[id].z < GOAL.Z){  for(모든 클라이언트){  send(server\_data);  }  SC\_INIT();  }  } | 해당 클라이언트가 목적지에 도달했는지 검사하는 함수  클라이언트의 z값이 목적지에 도달하면, ~~카운트 시간을 정한다.~~  모든 클라이언트와 장애물의 위치, 타이머를 초기화하는 SC\_INIT함수 호출 |
| void COLL\_CHECK(clients user, OBJ obj)  {  if (user.z > MIN(obj.z) && user.z < MAX(obj.z) && user.x > MIN(obj.x) && user.x < MAX(obj.x) ) {  ~~SERVER\_DATA server\_data;~~  ~~server\_data.dataType = GAME\_OVER;~~  ~~for (int i = 0; i < size(clients); ++i)~~  ~~send(server\_data);~~  SC\_COLL(user);  }  } | 클라이언트와 장애물 간의 충돌을 체크해서 각 클라이언트에게 충돌 여부를 전달하는 SC\_COLL함수를 호출한다. |
| void SC\_COLL(int num)  {  SERVER\_DATA server\_data;  server\_data.dataType = GAME\_OVER;  server\_data.id = num;  for (모든 클라이언트) {  send(server\_data);  }  } | 장애물과 충돌한 클라이언트의 정보를 모든 클라이언트에게 전달하는 함수 |
| void SC\_SEND(CLIENT\_DATA client\_data)  {  SERVER\_DATA server\_data;  if(LOGOUT){  for(모든 클라이언트)  send(server\_data);  }  else if (GAME\_ START){  SC\_GAMESTART();  }  else if (LOCATION){  switch(client\_data){  case MOVE\_FRONT:  server\_data.z += dz;  break;  case MOVE\_BACK:  server\_data.z -= dz;  break;  case MOVE\_RIGHT:  server\_data.x += dx;  break;  case MOVE\_LEFT:  server\_data.x -= dx;  break;  }  server\_data.id = client\_data.id;  for (int i = 0; i < size(clients); ++i)  send(server\_data);  }  ~~if(true == Goal\_Check){~~  ~~SERVER\_DATA server\_data;~~  ~~server\_data.dataType = GAME\_RESULT;~~  ~~server\_data.mission\_result= true;~~  ~~send(server\_data);~~  ~~}~~  ~~if(true == Coll\_check){~~  ~~SERVER\_DATA server\_data;~~  ~~server\_data.dataType = GAME\_RESULT;~~  ~~server\_data.mission\_result= false;~~  ~~send(server\_data);~~  ~~}~~  GOAL\_CHECK(user);  } | 서버에서 클라이언트로 클라이언트의 로그아웃 정보, 게임 시작 정보, 위치 이동 정보를 전송하고  위치 이동 후 목적지 도착 확인 함수 호출 |
| void SC\_LOGIN()  {  SERVER\_DATA server\_data;  server\_data.dataType = LOGIN;  for (int i = 0; i < size(clients); ++i) {  send(server\_data);  }  } | 클라이언트가 접속하면, 모든 유저에게 접속한 클라이언트 정보를 전달하는 함수 |
| void SC\_GAMESTART()  {  SERVER\_DATA server\_data;  server\_data.dataType = GAME\_START;  for (int i = 0; i < size(clients); ++i) {  send(server\_data);  }  } | 3명의 클라이언트가 모두 접속했을 때, 모든 클라이언트에게 게임 시작 정보를 전달하는 함수 |
| void SC\_INIT()  {  } | 모든 클라이언트의 위치 값, 모든 장애물의 위치 값, 타이머 수치를 초기화하여 모든 클라이언트에게 전송하는 함수 |
| bool RESET\_OBJECT(OBJECT\_INFO object)  {  if (objectInfo[1].z < object.z && true == object.moving) {  return true;  }  return false;  } | 장애물의 z좌표를 TRACKER의 z좌표와 비교해서 좌표의 초기화 여부를 판단하는 함수 |

4. 팀원 별 역할 분담

박의인: 서버 구조, 클라이언트에서 보낸 요청 처리 후 클라이언트로 패킷 결과 전송, 타이머

S\_RECV\_PACKET(), SC\_SEND(), SC\_LOGIN(), SC\_GAMESTART(), SC\_TIME(),

홍진선: 서버에서 보낸 데이터를 클라이언트에서 처리, 클라이언트에서 패킷 전송

CS\_MOVE(), C\_UPDATE(), C\_SAVE\_PACKET(), CS\_START()

정의범: 게임제작, 충돌 판정, 장애물 움직임, 목적지 도달 네트워크 처리

COLL\_CHECK(), GOAL\_CHECK(), SC\_OBJECT\_MOVE(), SC\_COLL(), RESET\_OBJECT(), SC\_INIT()

5. 개발 환경

* 운영 체제: 윈도우
* 개발 환경: 비쥬얼 스튜디오 2019 (mode: Release, x86
* API: Open GL
* 버전 관리: GitHub (git pork를 활용)

**6. 개발 일정**

**개별 개발일정**

박의인

11월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
|  | 11/1 | 11/2 | 11/3 | 11/4 | 11/5 | 11/6 |
|  |  |  |  |  |  |  |
| 11/7 | 11/8 | 11/9 | 11/10 | 11/11 | 11/12 | 11/13 |
|  |  | 계획서 재검사 |  | **졸작 회의** | 서버 소켓 생성 및 기본 구조 작성 | **백신 2차 1일차** |
| 11/14 | 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 |
| **백신 2차 2일차** | ~~C\_SAVE\_PACKET() 작성~~ | 1주차 점검 및 회의 | S\_SAVE\_PACKET() 작성 | **졸작 회의** |  | SC\_LOGIN() 작성 |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
|  | SC\_GAMESTART() 작성 | 2주차 점검 및 회의(완성된 클라이언트에 서버 연결 후 테스트) | SC\_SEND() - MOVE 패킷 처리 작성 | **졸작 회의** |  | SC\_SEND() - 충돌 시 패킷 전송 처리 작성 |

12월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
| 11/28 | 11/29 | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 |
|  |  | 3주차 점검 및 회의 |  |  |  | SC\_SEND() 도착 시 패킷 전송 처리 작성 |
| 12/5 | 12/6 | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 |
| SC\_TIME() – 타이머 보내기 작성 |  | 4주차 점검 및 회의 |  |  |  |  |
| 12/12 | 12/13 | 12/14 |  |  |  |  |
|  |  | 과제 종료 |  |  |  |  |

홍진선

11월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
|  | 11/1 | 11/2 | 11/3 | 11/4 | 11/5 | 11/6 |
|  | 추진 계획서 제출 | 계획서 검수 |  |  |  |  |
| 11/7 | 11/8 | 11/9 | 11/10 | 11/11 | 11/12 | 11/13 |
|  |  | 계획서 재검사 | **데이터베이스 시험** | OpenGL 개발환경 세팅 |  |  |
| 11/14 | 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 |
|  | **졸업작품 면담** | 1주차 점검 및 회의 | C\_UPDATE() -  case: LOGIN, GAMESTART 작성 | ~~C\_UPDATE() -~~  ~~case: TIME 작성~~  ~~(12/4일로 연기)~~ |  | ~~CS\_MOVE() -~~  ~~case: VK\_UP, VK\_DOWN 작성~~ |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| ~~CS\_MOVE() -~~  ~~case: VK\_RIGHT, VK\_LEFT 작성~~  CS\_MOVE()  case: ‘w’,  ‘s’ 작성 |  | 2주차 점검 및 회의(완성된 클라이언트에 서버 연결 후 테스트) | ~~C\_UPDATE() -~~  ~~case: LOCATION 작성~~  CS\_MOVE() -  case: ‘a’, ‘d’ 작성 | C\_UPDATE() -  case: LOCATION 작성 |  | ~~C\_UPDATE() -~~  ~~case: RESULT 작성~~  ~~(12/5일로 연기)~~ |

12월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
| 11/28 | 11/29 | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 |
|  |  | 3주차 점검 및 회의 |  |  |  | C\_UPDATE() -  case: TIME 작성 |
| 12/5 | 12/6 | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 |
| C\_UPDATE() -  case: RESULT 작성 |  | 4주차 점검 및 회의 | CS\_START()  작성 |  |  |  |
| 12/12 | 12/13 | 12/14 |  |  |  |  |
|  |  | 과제 종료 |  |  |  |  |

정의범

11월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
|  | 11/1 | 11/2 | 11/3 | 11/4 | 11/5 | 11/6 |
|  | 추진 계획서 제출 | 계획서 검수 |  |  |  |  |
| 11/7 | 11/8 | 11/9 | 11/10 | 11/11 | 11/12 | 11/13 |
|  |  | 계획서 재검사 | **데이터베이스 시험** | OpenGL 개발환경 세팅 | 게임 맵 제작 | 괴물, 캐릭터 제작 |
| 11/14 | 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 |
|  | **졸업작품 면담**  Front 장애물 제작,  behind장애물 제작 | 1주차 점검 및 회의 |  | 유저와 장애물 속도 조정 | ~~장애물과 유저간 충돌 계산( COLL\_CHECK()) 작성~~  ~~(11/30으로 연기)~~ |  |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| ~~SC\_OBJECT\_MOVE(),~~  ~~함수 작성~~  ~~(11/27로 연기)~~ |  | 2주차 점검 및 회의(완성된 클라이언트에 서버 연결 후 테스트) |  | ~~GOAL\_CHECK()함수 작성~~  ~~(12/4로 연기)~~ | ~~타이머(SC\_TIME() ) 작성~~  ~~(12/5로 연기)~~ | SC\_OBJECT\_MOVE(),  함수 작성 |

12월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
| 11/28 | 11/29 | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 |
|  | **졸업작품 면담** | 장애물과 유저간 충돌계산( COLL\_CHECK()) 작성  3주차 점검 및 회의 |  |  |  | GOAL\_CHECK()함수 작성 |
| 12/5 | 12/6 | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 |
| ~~타이머(SC\_TIME() ) 작성(박의인 학생이 작성)~~ |  | 4주차 점검 및 회의 | SC\_COLL(),  RESET\_OBJECT() 작성 | SC\_INIT() 작성 |  |  |
| 12/12 | 12/13 | 12/14 |  |  |  |  |
|  |  | 과제 종료 |  |  |  |  |