네트워크 게임 프로그래밍

텀프로젝트 추진계획서

**<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 기획텍스트이(가) 표시된 사진

자동 생성된 설명

드라마 ‘오징어 게임’의 게임 중 유리 다리 건너기 게임을 간단히 제작해 보도록 한다.

* 키: 방향키(이동), 스페이스바(점프)
* 게임의 흐름: 클라이언트에서 서버에 연결하면 시작 위치에서 대기한다. 일정 수의 플레이어가 접속하면 게임 시작. 제일 먼저 강화유리를 밟으면 50점을 획득한다. 점수는 화면 왼쪽 위에 점수 내림차순으로 게시. 떨어지면 10점 감점 후 시작 위치에서 부활하여 다시 유리 다리를 건널 수 있다. 누구든 도착 지점에 도착한 순간 카운트 다운이 시작되고 그 시간안에 도착지점에 도착하여야 한다.
* 승리 조건: 도착 지점에 도착한 사람 중 점수가 가장 높은 사람이 승리하게 된다.
* 플레이어의 구분: 자신 캐릭터만 초록색, 나머지 플레이어는 빨간색으로 설정한다. 플레이어가 설정한 아이디를 캐릭터의 머리 위에 표시한다.
* 예상 게임 화면(그래픽 요소는 차이가 있음)

실내이(가) 표시된 사진

자동 생성된 설명

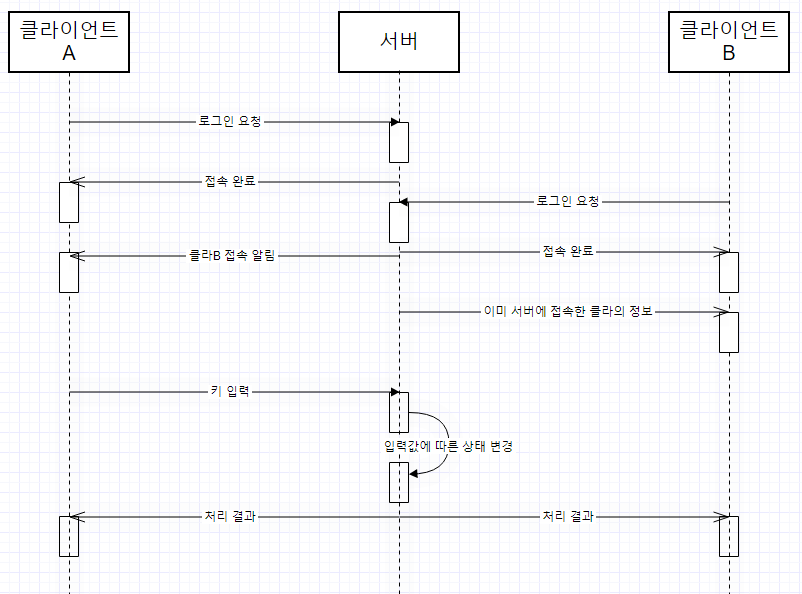
자신: 초록색

다른 플레이어: 빨간색

점수 표시

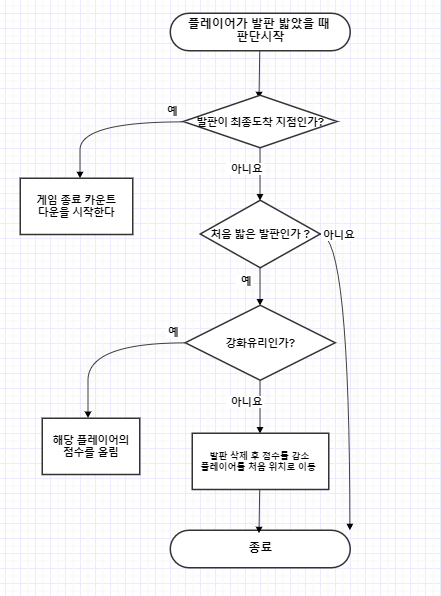
2. High-Level 디자인

- 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. | 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. |

-플레이어가 발판을 밟았을 때, 작동하는 알고리즘



3. Low-Level 디자인

|  |  |
| --- | --- |
| **헤더 파일** | |
| #pragma once  // CLIENT DATA TYPE  #define MOVE\_FRONT 0x01;  #define MOVE\_BACK 0x02;  #define MOVE\_RIGHT 0x03;  #define MOVE\_LEFT 0x04;  // SERVER DATA TYPE  #define LOCATION 0x11;  #define LOGIN 0x12;  #define GAME\_START 0x13;  #define GAME\_OVER 0x14;  #define GAME\_RESULT 0x15;  #define TIME 0x16;  // LOCATION 하위 TYPE  #define PLAYER 0x11;  #define OBJECT 0x12;  #pragma pack(push,1)  // DATA 형식  struct SERVER\_DATA {  unsigned char dataType;  unsigned char objectType;  unsigned short id;  unsigned short rank;  int x, y, z;  int time;  };  // CLIENT가 보내는 DATA 형식  struct CLIENT\_DATA {  unsigned char type;  unsigned short id;  };  // CLIENT INFO  struct CLIENT\_INFO {  unsigned short id;  bool alive;  int 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.  **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  (LOCATION의 주체)  PLAYER: 플레이어  OBJECT: 장애물  **서버 데이터 구조체**  **(서버->클라이언트)**  dataType: 위에 정의한 서버 데이터 타입  objectType: location의 대상  id: 서버가 클라이언트에 부여한 ID  rank: 클라이언트들의 순위  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**  id: 클라이언트의 ID  alive: 생존여부  x, y, z: 캐릭터의 위치 값. |

|  |  |
| --- | --- |
| **클라이언트** | |
| 함수 정의 | 함수 설명 |
| void RecvThread()  {  SERVER\_DATA server\_data;  while (true) {  recv(server\_data);  lock();  queue.input(server\_data);  unlock();  }  } | 서버에서 전송한 데이터를 전달받아 queue자료구조에 저장하는 스레드 함수.  자료구조에 저장 시 mutex를 동해 스레드 동기화. |
| void KeyInput()  {  CLIENT\_DATA client\_data;  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;  }  } | 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. |
| void update\_login(SERVER\_DATA server\_data)  {  clients[server\_data.id].alive = true;  clients[server\_data.id].x = 0;  clients[server\_data.id].y = 0;  clients[server\_data.id].z = 0;  } | 클라이언트 접속 시 서버에게서 접속한 유저의 정보를 받아 처리하는 함수 |
| void update\_location(SERVER\_DATA server\_data)  {  clients[server\_data.id].x = server\_data.x;  clients[server\_data.id].y = server\_data.y;  clients[server\_data.id].z = server\_data.z;  } | 서버로부터 각 클라이언트의 좌표를 받아 클라이언트의 위치를 변경하는 함수 |
| void update\_gameover(SERVER\_DATA server\_data)  {  clients[server\_data.id].alive = false;  } | 장애물과 부딪힌 클라이언트 정보를 받아 alive를 false로 설정하는 함수 |
| void update\_time(SERVER\_DATA server\_data)  {  time = server\_data.time;  } | 서버와 시간을 동기화하는 함수 |
| void update\_result(SERVER\_DATA server\_data)  {  switch (server\_data.rank)  {  case 1:  openFirstScene();  break;  case 2:  openSecondScene();  break;  case 3:  openThirdScene();  break;  }  } | 서버로부터 게임의 결과를 전달받아 각 클라이언트에게 순위에 맞는 씬을 보여주는 함수 |
| void update\_gamestart()  {  bool start = true;  } | 클라이언트 3명이 모두 접속 시, 서버가 보낸 데이터를 받아 key입력이 가능하게 하는 함수. |

|  |  |
| --- | --- |
| **서버** | |
| 함수 정의 | 함수 설명 |
| void RecvThread()  {  CLIENT\_DATA client\_data;  while (true) {  recv(client\_data);  lock();  queue.input(client\_data);  unlock();  }  } | 클라이언트에서 전송한 데이터를 전달받아 queue자료구조에 저장하는 스레드 함수.  자료구조에 저장 시 mutex를 동해 스레드 동기화. |
| void Coll\_check(clients user, OBJ obj)  {  if (true == Coll(user, obj) && user.alive) {  SERVER\_DATA server\_data;  server\_data.dataType = GAME\_OVER;  for (int i = 0; i < size(clients); ++i)  send(server\_data);  }  } | 클라이언트와 장애물 간의 충돌을 체크해서 각 클라이언트에게 충돌 여부를 전달하는 함수 |
| void send\_gameover()  {  int cnt = 0;  for (int i = 0; i < size(clients); ++i)  if (false == clients[i].alive)  cnt++;  if (size(clients) == cnt) {  SERVER\_DATA server\_data;  server\_data.dataType = GAME\_OVER;  for (int i = 0; i < size(clients); ++i)  send(server\_data);  send\_rank();  }  } | Alive == false 카운트가 클라이언트 수와 같아지면, 클라이언트에게 게임종료를 전송하고  Send\_rank()함수를 호출하는 함수 |
| void send\_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 Goal\_Check(CLIENT\_DATA client\_data)  {  if (clients[id].z < GOAL.Z && first\_goal == true)  time = 10;  } | 해당 클라이언트가 목적지에 도달했는지 검사하는 함수  클라이언트의 z값이 목적지에 도달하면, 카운트 시간을 정한다. |
| void send\_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 send\_rank()  {  int ranked[3] = sort(clients.x);  int rank = 1;  SERVER\_DATA server\_data;  server\_data.dataType = GAME\_RESULT;  for (int i = 0; i < size(clients); ++i) {  if (ranked[rank - 1] == clients[i].x) {  server\_data.rank = rank++;  send(server\_data);  i = 0;  }  }  } | 게임 종료 시, 클라이언트의 순위를 측정해서 게임 결과를 각 클라이언트에게 전달하는 함수 |
| void send\_object\_move(int o\_id)  {  SERVER\_DATA server\_data;  server\_data.dataType = LOCATION;  server\_data.id = o\_id;  server\_data.z += dz;  for (int i = 0; i < size(clients); ++i)  send(server\_data);  } | 장애물의 종류와 변경 위치 정보를 각 클라이언트에게 전달하는 함수 |
| void send\_player\_move(CLIENT\_DATA client\_data)  {  SERVER\_DATA server\_data;  server\_data.dataType = LOCATION;  switch (client\_data.type)  {  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);  } | 클라이언트가 입력한 key type에 따라 위치를 변경시켜 모든 클라이언트에게 변경된 좌표를 전달하는 함수 |
| void TIme\_Thread()  {  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 함수 |

4. 팀원 별 역할 분담

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

서버/클라 RecvThread, void send\_login(), void send\_gamestart(), void send\_object\_move()

void send\_player\_move()

홍진선: 서버에서 보낸 데이터를 클라이언트에서 처리, 클라이언트에서 패킷 전송

void KeyInput(), void update\_login(), void update\_location(),

void update\_gameover(), void update\_time(),

void update\_result(), void update\_gamestart()

정의범: 게임제작, 충돌 판정 네트워크 처리

게임제작, void Coll\_check(), void Goal\_Check(), void TIme\_Thread(), void send\_gameover(),

void send\_rank()

5. 개발 환경

* 운영 체제: 윈도우
* 개발 환경: 비쥬얼 스튜디오 2019 (mode: Release, x64
* 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 | 헤더파일 작성 | **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** | 서버 RecvThread() 작성 | 1주차 점검 및 회의 | 클라이언트 RecvThread() 작성 | **졸작 회의** | send\_login() 작성  - 로그인한 유저 처리 | send\_login() 작성  - 이미 로그인한 유저 처리 |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| send\_gamestart() 작성  - 게임 시작 데이터 전송 | send\_object\_move() 작성  - 정면 및 후방 장애물 | 2주차 점검 및 회의 | send\_object\_move() 작성  - 정면 발사 장애물 | **졸작 회의** | send\_player\_move() 작성  MOVE\_FRONT,  MOVE\_BACK | send\_player\_move() 작성  MOVE\_RIGHT,  MOVE\_LEFT |
| 11/28 | 11/29 | 11/30 |  |  |  |  |
| 데모 플레이 |  | 3주차 점검 및 회의(데모 플레이 후 오류 없을 시 추가 구현 구상) |  |  |  |  |

12월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
|  |  |  | 12/1 | 12/2 | 12/3 | 12/4 |
|  |  |  |  |  |  |  |
| 12/5 | 12/6 | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 |
|  | 최종 점검 및 회의 | 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,  update\_login() | update\_gamestart() |  |
| 11/14 | 11/15 | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 |
|  | **졸업작품 면담** | 1주차 점검 및 회의 | update\_gamestart() | update\_gameover() |  | update\_gameover() |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| KeyInput()  case: VK\_UP, VK\_DOWN |  | 2주차 점검 및 회의 | KeyInput()  case: VK\_RIGHT, VK\_LEFT |  |  |  |
| 11/28 | 11/29 | 11/30 |  |  |  |  |
| 데모 플레이 |  | 3주차 점검 및 회의(데모 플레이 후 오류 없을 시 추가 구현 구상) |  |  |  |  |

12월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
|  |  |  | 12/1 | 12/2 | 12/3 | 12/4 |
|  |  |  |  |  |  |  |
| 12/5 | 12/6 | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 |
|  | 최종 점검 및 회의 | 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 | 타이머(void TIme\_Thread()) 작성 |  |
| 11/21 | 11/22 | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 |
| 장애물과 유저간 충돌 계산(void Coll\_check()) 작성 |  | 2주차 점검 및 회의 | Goal\_Check()  함수 작성 | void send\_gameover() 함수 작성 |  | void send\_rank()  함수 작성 |
| 11/28 | 11/29 | 11/30 |  |  |  |  |
| 데모 플레이 |  | 3주차 점검 및 회의(데모 플레이 후 오류 없을 시 추가 구현 구상) |  |  |  |  |

12월 개발일정

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **일** | **월** | **화** | **수** | **목** | **금** | **토** |
|  |  |  | 12/1 | 12/2 | 12/3 | 12/4 |
|  |  |  |  |  |  |  |
| 12/5 | 12/6 | 12/7 | 12/8 | 12/9 | 12/10 | 12/11 |
|  | 최종 점검 및 회의 | 과제 종료 |  |  |  |  |