네트워크 게임 프로그래밍

텀프로젝트 추진계획서

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, 스페이스바
* 게임의 흐름: 클라에서 서버에 연결하면 시작 위치에서 대기한다. 일정 수의 플레이어가 접속하면 게임 시작. 제일 먼저 강화유리를 밟으면 1점을 획득한다. 점수는 화면 왼쪽 위에 점수 내림차순으로 게시. 떨어지면 -1점 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

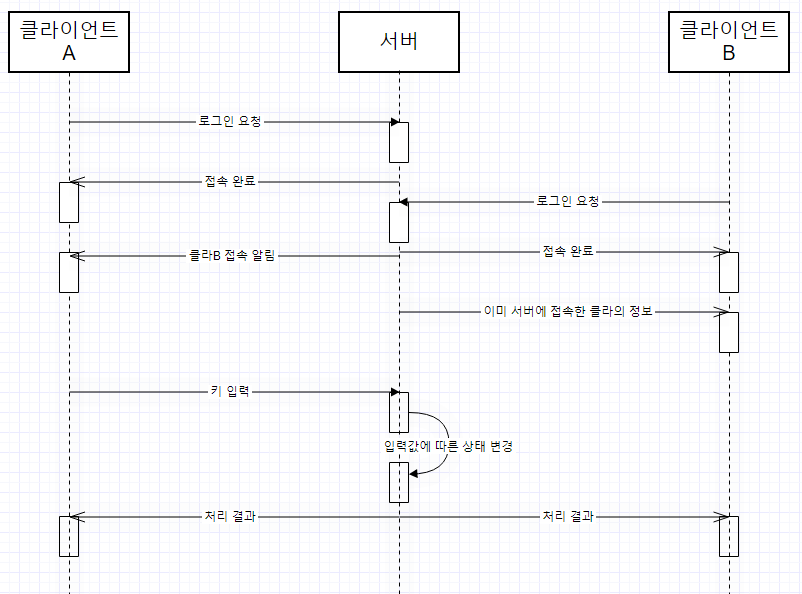
자신: 초록색

다른 플레이어: 빨간색

점수 표시

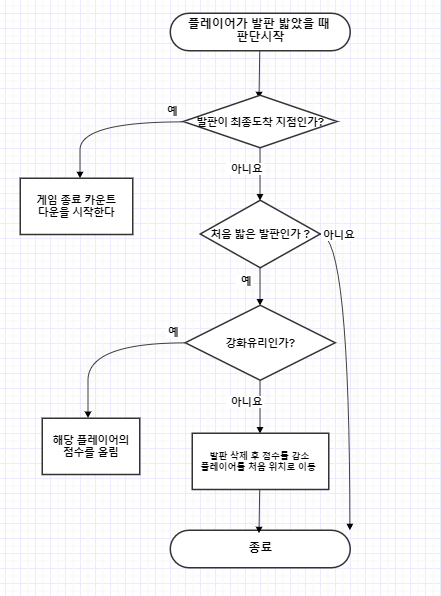
2. High-Level 디자인

- 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  // TYPE  #define MOVE\_FRONT 0x01;  #define MOVE\_BACK 0x02;  #define MOVE\_RIGHT 0x03;  #define MOVE\_LEFT 0x04;  #define MOVE\_JUMP 0x05;  #define LOCATION 0x11;  #define SCORE 0x12;  #define DESTROY 0x13;  #define LOGIN 0x14;  #define GAME\_START 0x15;  #define TIME 0x16;  #pragma pack(push,1)  // DATA 형식  struct SERVER\_DATA {  unsigned char type;  unsigned short id;  unsigned short score;  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;  unsigned short score;  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.  MOVE\_JUMP: 점프  **서버 데이터 타입**  **(서버->클라이언트)**  LOCATION: 캐릭터의 위치를 변경  SCORE: 점수  DESTROY: 일반 유리 발판 파괴  LOGIN: 클라이언트 로그인  GAME\_START: 게임 시작  TIME: 남은 시간  **서버 데이터 구조체**  **(서버->클라이언트)**  type: 위에 정의한 서버 데이터 타입  id: 서버가 클라이언트에 부여한 ID  score: 클라이언트의 점수  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  score: 점수  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 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 |
| void TopCheck(int max, int id)  {  for (int i = 0; i < clients.size(); ++i)  {  if (max < clients[i].score) {  max = clients[i].score;  id = clients[i].id;  }  }  string = "CLIENT\_INFO[id].id 유저가 CLIENT\_INFO[id].score로 1등중";  } | 서버로부터 SCORE 타입의 데이터를 전달받았을 때 점수를 업데이트하고 호출하는 함수.  함수 실행을 통해 현재 1등인 유저를 띄워준다. |
| void Destroy\_Object(int id)  {  Scaffolding[id].destroy;  } | 서버로부터 DESTROY 타입의 데이터를 전달받았을 때 호출하는 함수.  함수 실행을 통해 깨진 발판을 삭제한다. |
| void Update(SERVER\_DATA server\_data)  {  switch (server\_data.type)  {  case LOCATION:  clients[server\_data.id].x = server\_data.x;  clients[server\_data.id].y = server\_data.y;  clients[server\_data.id].z = server\_data.z;  break;  case SCORE:  clients[server\_data.id].score  += server\_data.score;  TopCheck(current\_max\_value, current\_top\_id);  break;  case DESTROY:  Destroy\_Object(server\_data.id);  break;  case LOGIN:  clients[server\_data.id].id = server\_data.id;  break;  case GAME\_START:  break;  case TIME:  break;  }  } | RecvThread()에서 자료구조 queue에 저장한 데이터가 있을 경우 호출되는 함수.  Queue에 저장된 데이터를 인자로 받아 데이터의 타입에 따라 전달 받은 처리 결과 반영.  LOCATION: 플레이어의 위치 변경  SCORE: 플레이어의 점수 업데이트  DESTROY: 발판 파괴  LOGIN: 클라이언트 로그인  GAME\_START: 게임 시작  TIME: 남은 시간 출력 |

|  |  |
| --- | --- |
| **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(OBJ user, OBJ scaf)  {  SERVER\_DATA server\_data;  if (true == Coll(user, scaf) && 발판처음 == true) {  if (강화유리 == scaf) {  server\_data.score = 50;  }  else {  server\_data.type = DESTROY;  server\_data.score = -10;  server\_data.id = scaf.id;  for (모든 유저)  send(server\_data);  }  server\_data.type = SCORE;  server\_data.id = user.id;  for (모든 유저)  send(server\_data);  }  } | 플레이어와 발판간 충돌체크 함수  플레이어가 발판을 밟았을 때,  처음 밟는 발판이면,  그 발판이 강화유리인지 일반 유리인지 판단.  발판이 강화유리일 때,  점수 올리고  발판이 일반 유리일 때,  발판을 삭제하고  플레이어 점수를 감소시킨다.  그 후 바뀐 점수 정보를 모든 클라이언트에게 전송 |
| void Send\_Update(CLIENT\_DATA client\_data)  {  unsigned short id = client\_data.id;  SERVER\_DATA server\_Data;  server\_Data.id = client\_data.id;;  server\_Data.type = LOCATION;  server\_Data.x = clients[id].x;  server\_Data.y = clients[id].y;  server\_Data.z = clients[id].z;  switch (client\_data.type)  {  case MOVE\_FRONT:  clients[id].z -= dz;  server\_Data.z = clients[id].z;  break;  case MOVE\_BACK:  clients[id].z += dz;  server\_Data.z = clients[id].z;  break;  case MOVE\_RIGHT:  clients[id].x += dx;  server\_Data.x = clients[id].x;  break;  case MOVE\_LEFT:  clients[id].x -= dx;  server\_Data.x = clients[id].x;  break;  case MOVE\_JUMP:  JumpThread();  break;  }  if(client\_data.type != MOVE\_JUMP){  for (모든 유저)  send(server\_Data);  Goal\_Check(client\_data);  }  } | RecvThread()에서 queue에 저장한 데이터가 있을 경우, 실행되는 함수  기존 클라이언트 위치에 저장한 데이터 확인 후 위치를 변경.  Client 데이터 타입이  MOVE\_FRONT일 경우, 해당 플레이어의 z 값을 (-)  MOVE\_ BACK 일 경우, 해당 플레이어의 z 값을 (+),  MOVE\_ RIGHT 일 경우, 해당 플레이어의 x 값을 (+),  MOVE\_ LEFT 일 경우, 해당 플레이어의 x 값을 (-),  하여 변경된 값을 전송.  MOVE\_ JUMP 일 경우, JumpThread()를 호출  좌표 변경 후 목적지에 도달했는지 검사. |
| void JumpThread(CLIENT\_DATA client\_data)  {  SERVER\_DATA server\_Data;  unsigned short id = client\_data.id;  server\_Data.type = LOCATION;  server\_Data.x = clients[id].x;  server\_Data.y = clients[id].y;  server\_Data.z = clients[id].z;  for (int i = 0; i < NUM; ++i) {  clients[id].y += dy;  if (i == NUM/2) dy \*= -1;  for (모든 유저)  send(server\_Data);  }  ExitThread( );  } | JUMP키 입력 시 호출되는 쓰레드  일정 횟수만큼, 플레이어의 y값을 (+)  증가한 횟수만큼, 플레이어의, y값을 (-)  하고 모든 유저에게 변경된 데이터 전송하고 쓰레드 종료한다. |
| 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 TIme\_Thread()  {  server\_Data.type = TIME;  server\_Data.time = 100;  while (true) {  for (모든 유저)  send(server\_Data);  server\_Data.time--;  }  } | 남은 제한 시간을 알려주는 쓰레드  모든 클라이언트에게 제한 시간 데이터 정보를 전송한다. |

4. 팀원 별 역할 분담

박의인: 선생님

홍진선: 반장

정의범: 모범생

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

- 팀

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
|  |  |  | 10/27 |  | 10/29 | 10/30 |
|  |  |  | 회의 |  | 회의 | 회의 |
| 10/31 | 11/1 |  |  |  |  |  |
| 회의 | 계획서 제출 |  |  |  |  |  |
|  |  |  |  |  |  |  |

- 개별

박의인

홍진선

정의범