**일렉트릭 블루, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명**

**네트워크 게임프로그래밍  
프로젝트 진행 보고서**

**2017114025 성현석  
2019182003 권순원  
2019182024 엄장헌**

엄장헌

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| 10/29 | 30 | 30 | 11/1 | 2 | 3 | 4 |
|  |  |  |  |  | 시험 | LockQueue |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Client  PacketManager  (연결부분만) | Client-Server  Initialize() | Client-Server  Send(), Recv() | Client-Server  Loop 구조 | Client  PacketManager |  | 졸작 회의 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| ~~Client~~  ~~움직임 방식 변경~~ | 시험 | 시험 | 시험 | ~~Client~~  ~~타 플레이어 관리 추가~~ |  | 졸작 회의 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| Client  움직임 방식 변경 | Client  타 플레이어 관리 추가 | Client  Stage 구조 변경 | InGame  Collision Manager | 플레이 가능 브렌치 생성 | 알파 테스트 | 졸작 회의 |
| 26 | 27 | 28 | 29 | 30 | 12/1 | 2 |
|  | 버그 픽스 |  |  |  | 베타 테스트 | 졸작 회의 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 문서 정리 | 코드 정리 |  | 최종 점검 |  |  |

1주차 (11/4 ~ 11/9) 100%:   
LockQueue  
서버에서 공용으로 사용하는 큐를 동기화하기 위한 LockQueue class 작성 완료  
PacketManager  
클라이언트 패킷 매니저의 초기화와 send, recv 구현 완료  
키 입력시 PacketManager  
서버에서 클라이언트와 패킷을 주고 받을 쓰레드 함수 초기화 및 send, recv 부분 완료  
쓰레드 메인 루프에서 너무 긴 시간 점유하지 않도록 타임아웃 옵션 설정 완료

변경사항:  
기존 계획에서는 난이도 수신을 wait server에서 받고 Client-server 스레드에 전달할 계획 이였는데, Client-server 스레드에서 초기화 작업을 하는 과정에서 난이도를 전달 받는것으로 수정

2주차 (11/11 ~ 11/16):

시험 주간

3주차 (11/18 ~ 11/23): 100%

다른 플레이어 랜더링 구현.  
플레이어가 상하좌우 모든 방향으로 움직일 수 있도록 수정  
PacketManager를 통해 서버와 정보를 주고받는 부분 구체화  
다른 플레이어의 정보 갱신

4주차, 5주차 (11/25 ~ 12/5): 100%

장애물과 플레이어의 충돌 방식을 변셩  
모든 플레이어 뒤에 파티클 생성 추가  
플레이어와 장애물 충돌시에 터지는 파티클 추가  
충돌시 사망 처리 및 관련 패킷 송신  
  
위치 보간 처리에서 버벅이는 문제를 해결하기 위해  
보정 위치와 현재 위치가 너무 크지 않다면 이동되지 않도록 수정  
플레이어 움직임과 패킷 수신관련 버그 수정  
코드 정리 및 테스트 코드 및 주석 정리  
  
바뀐부분: 플레이어는 가속도를 보내지만, 받는 부분은 속도로 변경

성현석

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| 10/29 | 30 | 30 | 11/1 | 2 | 3 | 4 |
|  |  |  |  |  |  | InGame  Initialize() |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 졸작 회의 | 시험 | 시험 | InGame  Loop 구조  Lobby  GameStart() | InGame  PushPaket() |  | InGame  Physics Class 테스트 환경 제작 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 졸작 회의 | InGame  Physics Class 멤버 함수 | InGame  Physics Class 멤버 함수 | InGame  Collision Manager | InGame  Collision Manager |  | InGame  Update() |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 졸작 회의 | InGame  Collision Manager | 시험 | 시험 | 플레이 가능 브렌치 생성 | 알파 테스트 |  |
| 26 | 27 | 28 | 29 | 30 | 12/1 | 2 |
| 졸작 회의 | 버그 픽스 |  |  |  | 베타 테스트 |  |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 졸작 회의 | 문서 정리 | 코드 정리 |  | 최종 점검 |  |  |

1주차 (11/4 ~ 11/9) 100%:   
InGameThread 초기화와 각 클라이언트의 EventQueue의 주소를 담을 새로운 구조체 EventQueues 구현  
InGameThread Loop 구조 구현  
EventQueue의 내용 읽기(pop)를 위해 ToSeverQueueCheck() 구현  
EventQueue에 내용을 보내기(push)를 위한 PushPacket() 구현

변경사항  
InGameThread애서 ClientInfoQueue의 데이터를 읽는게(pop) 아닌 LobbyThread에서 읽는 것(pop)으로 변경

2주차 (11/11 ~ 11/16): 100%

InGamePhysics 클래스 구조 구현  
CollisionManager 클래스 구조 구현  
추가로 필요한 PlayerInfo 구조체 생성  
Physics 플레이어끼리의 충돌은 운동량 보존 법칙이용  
벽이나 플레이어간의 충돌 여부를 체크하기위한 InGamePhysics 클래스의 멤버함수 구현  
충돌후 가속도를 구현하기 위한 물리법칙을 적용한 CollisionManager 클래스의 멤버함수 구현

<변경 사항>  
시험 일정 변경으로 모든 일정이 2일 당겨짐

<변경할 점>  
오류를 방지하기 위해 충돌 시 오차 구현  
운동량 보존 법칙에서 작용 반작용 법칙을 적용

3주차 (11/18 ~ 11/23): 100%

InGamePhysics 클래스 맴버 함수를 운동량 보존 법칙에서 작용 반작용 법칙으로 변경  
오류를 방지하기 위해 CollisionManager 클래스 보완  
Client move() 추가 구현  
클라이언트의 강제종료 여부 판단, 패킷을 상황에 맞게 전송등 InGameThread 구조 개선

4주차, 5주차 (11/25 ~ 12/5): 100%  
Physics 버그 수정  
패킷 전송 데이터를 가속도에서 속도로 변경  
elapsedTime을 적용시켜 물리 계산 및 일정 시간 마다 좌표 정보 Push 구현  
속도로 변경됨에 따라 충돌에 대한 안정성을 높이기위해 계산식 수정  
작용 반작용에서 충격량 보존 법칙으로 변경  
플레이어들의 충돌에 대한 패킷 정보 처리  
InGameThread 버그 수정 및 보완  
위치정보에 선형 보간법 적용  
전체 코드 리뷰

권순원

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 일 | 월 | 화 | 수 | 목 | 금 | 토 |
| 10/29 | 30 | 30 | 11/1 | 2 | 3 | 4 |
|  |  |  |  | 시험 |  | wait server  Initialize() |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| wait server  AcceptClient() | Lobby  Initialize() | Lobby  Loop 구조 | Lobby  GameStart() | Client  추가 플레이어 모델 로드 |  | 졸작 회의 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| ~~Client~~  ~~움직임 방식 변경~~ | 시험 | 시험 | 시험 | ~~Client~~  ~~타 플레이어 랜더링~~ |  | 졸작 회의 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| Client  움직임 방식 변경 | Client  타 플레이어 랜더링 | Client  장애물 패턴 추가 | Lobby  테스트 및 버그픽스 | 플레이 가능 브렌치 생성 | 알파 테스트 | 졸작 회의 |
| 26 | 27 | 28 | 29 | 30 | 12/1 | 2 |
|  |  | 버그 수정 |  |  | 베타 테스트 | 졸작 회의 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 문서 정리 | 코드 정리 |  | 최종 점검 |  |  |

1주차 (11/4 ~ 11/9): 100%  
MainThread  
서버가 시작되고 LobbyThread 호출  
대기소켓 생성 후 Client 요청이 있을 시 accept하고 client-server thraed 호출

LoobyThread  
MainThread에서 생성 요청을 받아 생성  
ClinetInfoQueue에서 인원을 체크하고 InGame thread를 호출

변경 사항  
원래 계획은 InGame thread를 초기화 하면서 ClientInfoQueue에서 WaitPop을 통해 플레이어와 통신할 수 있는 큐를 가져오는 것이었는데, thread로 동작하기 때문에 초기화를 다 하기 전에 접근해 InGame thread가 두 번 호출되는 문제가 발생해 Lobby thread에서 WaitPop을 해서 전달해주는 것으로 변경  
1주차 계획 내용 전부 구현.

2주차 (11/11 ~ 11/16):

시험주간

3주차 (11/18 ~ 11/23): 100%

다른 플레이어 랜더링 구현.  
플레이어가 상하좌우 모든 방향으로 움직일 수 있도록 수정  
장애물 생성 방식 변경 및 장애물 패턴 추가

4주차, 5주차 (11/25 ~ 12/5): 100%  
elapsed time 적용  
장애물 패턴 추가  
장애물을 서버로부터 seed 값을 받아 모든 플레이어가 동일한 패턴의 장애물을 보도록 변경  
게임 시작 시 대화상자를 통해 IP를 입력해 접속할 수 있도록 변경  
모든 플레이어가 모일 때까지 기다리다 모든 플레이어의 패킷을 받으면 게임이 시작하도록 변경  
장애물 생성 방식 변경(오브젝트 폴링)  
게임 종료 화면 변경 및 추가

버그 수정

- 플레이어 위치가 제대로 초기화 되지 않았던 문제 수정  
- InGame 쓰레드에서 모든 플레이어 데이터가 확인되기 전에 진행이 되어 InGame 쓰레드가 두 번 호출되는 문제 해결(Lobby 쓰레드에서 플레이어에 대한 처리를 해서 넘겨줌)  
최종 버전 생성 전 코드 리뷰

**프로젝트 추진계획서  
(변경사항)**

**목차**

1. **게임 소개**
   1. **개요**
   2. **프로젝트를 위한 변경점**
   3. **스크린샷**
   4. **플레이 방법 및 규칙**
2. **개발 환경**
   1. **개발 환경**
   2. **코드 스타일**
3. **High-level 디자인**
4. **Low-level 디자인**
5. **개발 일정**
   1. **역할 분담**
   2. **팀원 별 일정표**
6. **게임 소개**
   1. **개요**

2022년도 2학기 컴퓨터 그래픽스 팀 프로젝트 때 개발한 게임입니다. 팀원 권순원과 엄장헌이 참여해서 제작했습니다.  
 플레이어가 앞에서 날아오는 장애물을 피하면서 날아가는 게임입니다. 기존 게임에 멀티플레이 요소를 더하기 위해 기존 게임과 비슷한 『마리오 파티』 미니게임 중 하나를 참고하여 변경했습니다.

* 1. **프로젝트를 위한 변경점**

플레이어가 원 둘레 이동만 가능에서 평면상을 자유롭게 이동할 수 있는 방식으로 변경.  
장애물 또한 원 둘레를 따라 나오던 것을 평명 상에 랜덤하게 나올 수 있도록 변경.

* 1. **우주, 천체, 대기권 밖, 어둠이(가) 표시된 사진

     자동 생성된 설명스크린샷**

그림 2. 『마리오 파티』 미니게임 플레이 화면

그림 . 『Voyage in Space』 플레이 화면

그림 . 『Voyage in Space』 플레이 화면

* 1. **플레이 방법 및 규칙**

WASD 키를 이용해 상하좌우로 이동하며 날아오는 장애물을 회피.  
장애물에 부딪히면 ~~목숨이 감소하며, 3번 부딪히면~~ 해당 플레이어는 탈락(접속 종료).  
플레이어들은 서로 충돌을 하며 밀어내며 방해할 수 있다(가속도와 질량에 따라 충돌 처리).  
마지막까지 살아남은 플레이어가 최종 우승

**+수정사항: 3번에서 1번으로 수정**

1. **개발 환경**
   1. **개발 환경**

IDE: Visual Studio 2022  
버전관리: Git, GitHub  
그래픽 API: OpenGL  
개발언어: C++  
네트워크 프로토콜: TCP/IP  
네트워크 API: Winsock

* 1. **코드 스타일**

변수명: camelCase, global 변수 및 member 변수 g\_, m\_ 표기  
함수명: PascalCase, 설명 주석 필수  
중괄호 표기: 함수/클래스 등 아래 표시, 반복/분기문 옆에 표시

1. 텍스트, 도표, 평면도, 개략도이(가) 표시된 사진

   자동 생성된 설명**High-level 디자인**
2. **Low-level 디자인**
3. **공유 class 및 struct와 define**

**수정: 게임 난이도 기능 삭제(기존 플레이는 화면이 돌아가는 기능이 추가되어 난이도 구분이 있었는데 사방향으로 움직이는 것으로 변경해 더 이상 난이도 요소가 없어져 삭제)**

~~게임 난이도~~

~~enum~~ *~~class~~* ~~GAME\_LEVEL : byte{~~

~~EASY;~~

~~NORMAL;~~

~~HARD;~~

~~}~~

패킷 정의 – 모든 통신에 공통적으로 사용

struct Packet{

*float* x, y;

    byte stateMask;

}

stateMask[BBBBBBBB]

     [0] 게임시작용 비트

        [00] 플레이어 번호

            [0] pos a **(수정사항: 클라이언트에서 받을땐 a대신 speed의 의미)**

                [00] 목숨**(수정사항: 마지막 4비트는 처음 init 패킷에선 seed의 의미)**

                    [0] 진행중인가

                        [0] 우승인가

LockQueue – 공유 Event 큐 동기화

*class* LockQueue{

public:

    LockQueue();

    ~LockQueue();

    void Push(Packet){

        lock(); // mutex lock, 스코프를 벗어나면 자동으로 unlock

        que.push();

        notify(); // mutex 사용이 끝났다는 것을 기다리고 있는 스레드에게 알림

    }

*bool* TryPop(Packet&){

        lock();

        if(que.empty()){

            unlock();

            return false;

        }

        packet = que.front();

        que.pop();

        return true;

    }

    void WaitPop(Packet&){

        lock();

        wait(); // notify가 호출될 때까지 대기

        packet = que.front();

        que.pop();

    }

private:

    queue<Packet> que;

    mutex mtx;

    condition\_variable cv;

}

Client 매칭 대기열을 위한 구조체

struct ClientInfo{

    shared\_ptr<queue<Packet>> packetQuePtr;

    GAME\_LEVEL level

    SOCK sock

}

1. **Client**

클라이언트에서의 모든 통신 관리

*class* PacketManager{

public:

    PacketManager();

    ~PacketManager();

    void Initialize();

    void SendPacket(Packet); // key event에서 호출

*bool* RecvPacket(); // toClientEventQue에 push

**//수정사항: 아래 함수는 사용하지 않아서 제거**

~~shared\_ptr<queue<Packet>> GetPacketQueue();~~

**//추가적으로 플레이어를 기다리거나**

**//서버의 ip주소를 set,get하는 함수추가**

private:

    socket sock;

    shared\_ptr<queue<Packet>> toClientEventQue;

}

키 입력에 따른 패킷 전송

**//수정: 키보드 입력 쪽에선 누르고 있을 때 계속 패킷이 송신되어**

**//player 쪽 함수쪽으로 이동**

GLvoid keyboard(unsigned char key, *int* x, *int* y)

{

*float* acc = gameManager.handleEvent(key, true);

~~PacketManager.SendPacket(acc);~~

    glutPostRedisplay();

}

**//수정: 위와 동일**

GLvoid keyUp(unsigned char key, *int* x, *int* y)

{

*float* acc = gameManager.handleEvent(key, false);

~~PacketManager.SendPacket(acc);~~

    glutPostRedisplay();

}

클라이언트 업데이트

**//수정: 구조변경에 의해 삭제**

~~GLvoid updateTimer(~~*~~int~~* ~~value)~~

~~{~~

~~PacketManager.RecvPacket();~~

~~// elapsedTime update~~

~~gameManager.update(elapsedTime);~~

~~glutTimerFunc(1000 / gameSpeed, updateTimer, 0);~~

~~}~~

1. **Wait Server**

void Initialize()

// lobby thread 생성, client info queue 초기화, 대기 소켓 생성

    dword void LobbyThread()

void acceptClient()

    dword void ClientThread(ClientInfo)

void recv()

//클라이언트에게 선택한 난이도 정보를 받음, info queue 에 넣어주고 다시 대기 상태로

1. **Lobby Thread**

void Initialize()

wait server에서 호출

while

// client info queue 에 난이도에 따른 큐가 가득 찼는지 확인

    - 가득찬 경우

**// 수정: 만약 취소한 플레이어가 있는 경우 인원이 다 모일 때까지 다시 대기하는 것으로 변경**

    void GameStart(ClientInfo, GAME\_LEVEL)

        dword void GameThread(ClientInfo, GAME\_LEVEL)

        // 소켓 정보와 ~~난이도 정보~~를 전달해 인게임 쓰레드를 생성

**//수정: 난이도 삭제**

1. **In Game Thread**

여기서 처리되는 모든 데이터가 게임 플레이의 기준  
각 게임 세션마다 하나씩 생성

void Initialize(GAME\_LEVEL, sock)

    // 초기 패킷을 toClientEventQue에 push

    // 초기 패킷 - [1][00 플레이어 번호][1 pos][0000 랜덤 패턴 seed 값], x, y : 플레이어 시작 위치

while(true){

    if (numOfPlayer == 0)

        break;

    if (numOfPlayer == 1)

        // 해당 플레이어에게 승리 비트 전송

        PushWinPacket(); // [*0\_\_*\* \*\*0*1*]

        // 게임 종료

        GameEnd();

    // 게임 로직

    void GameLoop()

        // 모든 플레이어의 toServerQue 확인 후 처리

        toServerQueueCheck(); // 모든 클라이언트 이벤트 확인

        void Update();

        // 충돌이 발생하면 toClientQue에 정해진 비트 규칙에따라 push

        PushPacket(); // [*0\_\_1* \_\_10]

**//수정: 속도 정보를 보내도록 변경**

    // ~~모든 플레이어의 가속도 정보~~ toClientQue push

    PushPacket(); // [*0\_\_1* \_\_10]

    if (time <= 0.3)

        // 모든 플레이어 위치 정보 toClientQue push (클라이언트 보정 용도)

        PushPacket() // [*0\_\_0* \_\_10]

}

1. **Client-Server 통신 Thread**

void Initialize() // 이벤트 큐를 동적으로 할당, 이벤트 큐의 주소와 난이도를 client info push

while(){

    if(recv(struct Packet) == 0)

        0000 0000 x MAX\_FLOAT y MIN\_FLOAT 을 toServerEventQue에 push

        break; // client 접속 끊김

    else

        toServerEventQue.push();

    if(*!*toClientEventQue.empty())

        send(struct Packet)

    yield();

}

1. **개발 일정**
   1. **역할 분담**

엄장헌: **1번**, **5번**, 4번(보조)

성현석: **4번**, 2, 3번(보조)

권순원: **2번**, **3번**, 1번(보조)