**일렉트릭 블루, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명**

**네트워크 게임프로그래밍  
프로젝트 추진계획서**

**2017114025 성현석  
2019182003 권순원  
2019182024 엄장헌**

**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

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. **High-level 디자인**

**2**

**텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

**54**

**4**

**3**

**1**

1. **Low-level 디자인**
2. **공유 class 및 struct와 define**

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

                [00] 목숨

                    [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

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

private:

    socket sock;

    shared\_ptr<queue<Packet>> toClientEventQue;

}

키 입력에 따른 패킷 전송

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

{

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

    PacketManager.SendPacket(acc);

    glutPostRedisplay();

}

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

{

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

    PacketManager.SendPacket(acc);

    glutPostRedisplay();

}

클라이언트 업데이트

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

    void GameStart(ClientInfo, GAME\_LEVEL)

        dword void GameThread(ClientInfo, GAME\_LEVEL)

        // 소켓 정보와 난이도 정보를 전달해 인게임 쓰레드를 생성

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]

    // 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)

* 1. **팀원 별 일정표**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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  Stage 구조 변경 | InGame  Collision Manager | InGame  Collision Manager | 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 |  |  |

엄장헌

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 | 시험 | 시험 | InGame  Physics Class 멤버 함수 |  | InGame  Update() |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 졸작 회의 | InGame  Collision Manager | InGame  Collision Manager | InGame  Collision Manager | 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 |  |  |

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  패턴 추가 | Lobby  서버 테스트 환경 구축 | Lobby  테스트 및 버그픽스 | Lobby  테스트 및 버그픽스 | 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 |  |  |

**\* 매주 금요일은 코드리뷰 및 점검, Progress Report 작성**