네트워크 게임프로그래밍

프로젝트 추진계획서

2017114025 성현석

2019182003 권순원

2019182024 엄장헌

목차

1. 게임 소개

A. 개요

2022 년도 2 학기 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.

B. 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.

C. 스크린샷

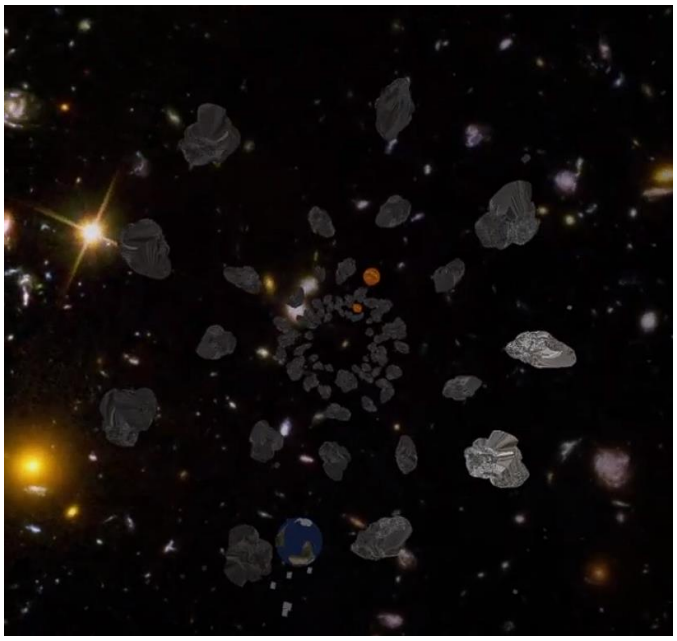


그림 1.『Voyage in Space』플레이 화면

그림 2.『마리오 파티』미니게임 플레이 화면

D. 플레이 방법 및 규칙

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

2. 개발 환경

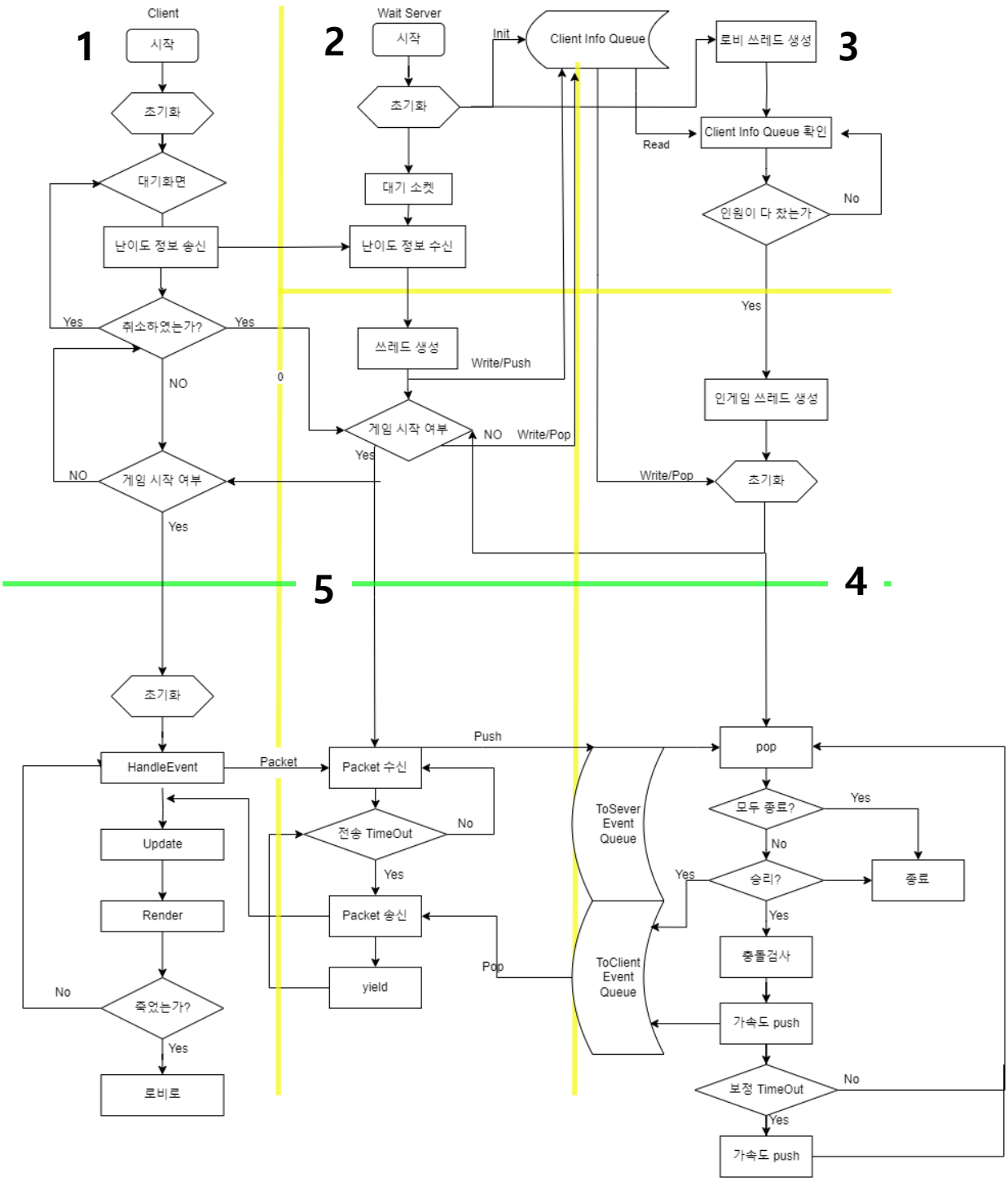
A. 개발 환경

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

B. 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

3. High-level 디자인



4. Low-level 디자인

0. 공유 class 및 struct 와 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(); // notify 가 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 event 에서 호출
    bool RecvPacket(); // toClientEventQue 에 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);
}
```

2. 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
```

3. Lobby Thread

```
void Initialize()
wait server 에서 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
```

4. In Game Thread

여기서 처리되는 모든 데이터가 게임 플레이의 기준

각 게임 세션마다 하나씩 생성

```
void Initialize(GAME_LEVEL, sock)
  // 초기 패킷을 toClientEventQue 에 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__* **01]
    // 게임 종료
    GameEnd();
  // 게임 로직
  void GameLoop()
    // 모든 플레이어의 toServerQue 확인 후 처리
    toServerQueueCheck(); // 모든 클라이언트 이벤트 확인
    void Update();
    // 충돌이 발생하면 toClientQue 에 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]
}
```

5. 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 toServerEventQue 에 push
      break; // client 접속 끊김
   else
      toServerEventQue.push();

   if(!toClientEventQue.empty())
      send(struct Packet)

   yield();
}
```

5. 개발 일정

A. 역할 분담

엄장헌: **1 번**, **5 번**, 4 번(보조)

성현석: **4 번**, 2, 3 번(보조)

권순원: **2 번**, **3 번**, 1 번(보조)

B. 팀원 별 일정표

엄장헌

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

성현석

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

권순원

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

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