네트워크 게임 프로그래밍

Term 프로젝트 최종보고서

2019182025 원정우

2019182029 이원준

2022150039 조진영

목차

1. 애플리케이션 기획 3
1.1 개요 3
1.2 조작법 및 스크린샷 3
2. High-level 디자인 4
3. Low-level 디자인 5
3.1 사용자 정의 데이터 5
3.2 Server.cpp 7
3.3 Client.cpp 8
3.3.1. main 함수 8
3.3.2. GuiAndClientMain 함수 8
3.3.3. DlgProc 함수 9
3.3.4. ReceiveDataThread 함수 10
3.3.5. Update 함수 10
3.3.6. Event 함수 10
3.3.7. 키 입력을 처리하기 위한 KeyState 자료구조 11
3.3.8. 그 외 변경점 11
3.4 InfoCheckThread.cpp 12
3.5 ClientServerThread.cpp 13
3.6 InGameThread.cpp 15
4. 팀원 별 역할분담 및 개발일정 16
5. 개발환경 19

1. 애플리케이션 기획

1.1 개요

2023년도 2학기 컴퓨터 그래픽스 수업에서 원정우 팀원이 제작한 게임입니다.

사방이 벽면으로 막혀 있는 상황에서 앞쪽에서 무작위로 생성되어 날아오는 장애물을 마우스를 이용하여 좌우로 피하는 게임입니다.

기존 소스코드에서는 마우스로 캐릭터를 이동하던 것을 프로젝트에서는 키보드를 통해 이동하는 것으로 변경하기로 했습니다.

1.2 조작법 및 스크린샷

AD 키를 이용해 좌우로 움직이며 날아오는 장애물을 피할 수 있습니다. 이동 경로 상에 다른 플레이어가 있는 경우 더 이상 해당 방향으로 움직일 수 없습니다.

W 키를 이용해 점프할 수 있고, 착지하면서 다른 플레이어를 밟아 데미지를 줄 수 있습니다.

날아오는 아이템과 접촉하여 아이템을 획득할 수 있습니다. C 키를 이용해 아이템을 사용할 수 있고, 아이템 사용 시 다른 플레이어들은 장애물이 반투명하게 보이게 되어 장애물을 피하기 더욱 어려워집니다.

장애물과 부딪히거나 다른 플레이어에 의해 데미지를 입은 경우 사망 처리되며, 마지막 생존자가 우승하게 됩니다.

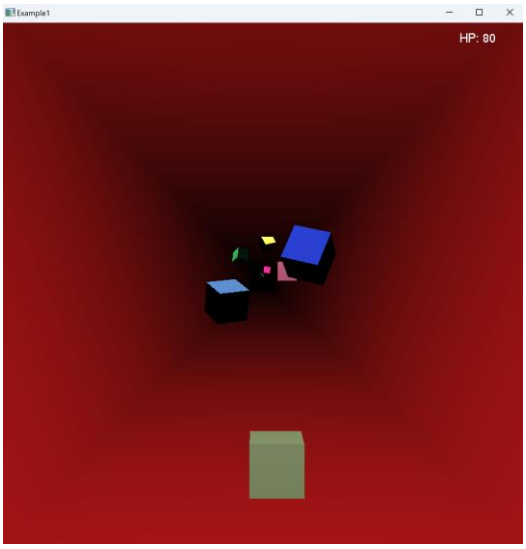
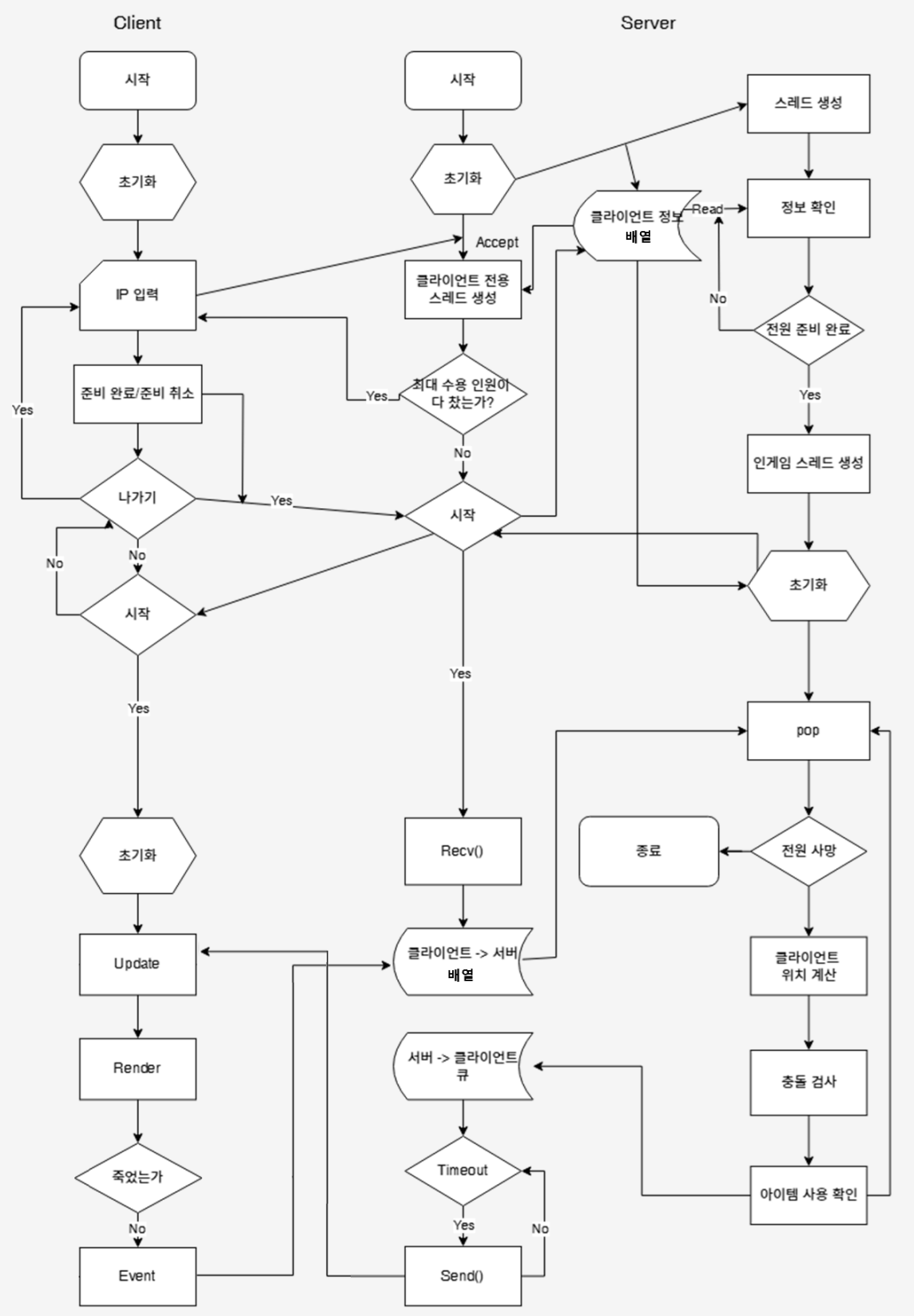


그림 1 : 게임 플레이 화면

2. High-level 디자인



3. Low-level 디자인

3.1 사용자 정의 데이터

■ 패킷 : 모든 통신에서 사용

➔ 기존 : 서버가 장애물의 최초 위치와 이동 방향에 대한 랜덤 시드를 전달하면 클라이언트가 이를 바탕으로 매 프레임마다 장애물의 위치를 계산

변경 : 서버가 랜덤 시드를 바탕으로 매 루프마다 장애물의 위치를 계산하고 이를 클라이언트에게 전송

```
class Packet {
        float x, y;       // 플레이어 위치
        BYTE state[2];   // 플레이어 상태 정보
        → 준비 완료 여부 0
          게임 시작 여부 0
          플레이어 번호 000 : Valid bit 0 + Number bit 00

          아이템 보유 여부 0
          아이템 적용 여부 0
          생존 여부 0

          현재 어떤 면이 바닥인가? 00 : 00(아랫면), 01(오른면), 10(윗면), 11(왼면)
          어떤 키가 눌려있는가? 0000 : WADC
          ~~장애물 시드 00~~ : 사용 X
};
```

■ 배열, 큐 : 서버 전역 변수

➔ 기존 : 클라이언트 정보 배열, 클라이언트 서버 큐, 서버 클라이언트 배열

변경 : 장애물 위치 전송을 위한 장애물 정보 배열 추가

➔ 기존 : 임계 영역을 이용한 동기화

변경 : 클라이언트 정보 배열에 대해 WriteEvent(초기값 : false)와 ReadEvent(초기값 : true)를 생성 → 클라이언트 전용 스레드는 클라이언트 정보 수신 시 ReadEvent를 기다린 뒤

클라이언트 정보 배열을 갱신하고 WriteEvent를 Set, 정보 확인 스레드는 WriteEvent를 기다린 뒤 클라이언트 정보를 검사하고 게임이 시작되지 않았다면 ReadEvent를 Set

std::array<Packet, 4> ClientInfoArray; : 클라이언트 정보 배열

std::queue<Packet> ClientServerQueue; : 클라이언트 → 서버 큐

std::array<Packet, 4> ServerClientArray; : 서버 → 클라이언트 배열 (클라이언트 정보)

std::array<Position, 10> ObstacleArray; : 서버 → 클라이언트 배열 (장애물 위치)

3.2 Server.cpp

➔ 기존 : 대기 소켓을 블로킹 소켓으로 유지 → accept 함수에서 블로킹되어 while 루프를 탈출하지 못하는 문제 발생
변경 : 대기 소켓을 논블로킹 소켓으로 전환하여 매 루프마다 게임 종료 여부 검사 → 검사 결과가 true일 경우 break하여 while 루프 탈출

```
서버 전역 변수, 동기화 객체 선언

int main(int argc, char* argv[]) {
        윈속 초기화
        대기 소켓을 논블로킹 소켓으로 전환
        동기화 객체 생성

        // 정보 확인 스레드 생성
        CreateThread(InfoCheckThread, (LPVOID)&InfoCheckThreadArg);

        while (1) {
                accept()
                if (게임이 종료되었다면) { break; }

                // 클라이언트 전용 스레드 생성
                클라이언트 전용 소켓을 블로킹 소켓으로 전환
                CreateThread(ClientServerThread, (LPVOID)ClientServerThreadArg);
}

동기화 객체 제거
윈속 종료
}
```

3.3 Client.cpp

3.3.1. main 함수

```
int main() {
        while (1) {
            if (연결 되어있지 않다면) {
                윈속 초기화
                connect();
                Receive 스레드 생성
            }
            if (게임이 시작되었다면) {
                while (1) {
                    Update & Render
                }
            }
        }

        CreateThread(GuiAndClientMain);   // GUI 및 소켓 통신 스레드 생성

        오브젝트 생성
        glutTimerFunc(16, update, 1);   // 인게임 업데이트
}
```

3.3.2. GuiAndClientMain 함수

```
DWORD WINAPI GuiAndClientMain(LPVOID arg) {
        DialogBox(g_inst, MAKEINTRESOURCE(IDD_DIALOG1), NULL, DlgProc);
}
```

3.3.3. DlgProc 함수

```
INT_PTR DlgProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam) {
        switch (uMsg) {
        case WM_INITDIALOG:
                GUI 생성

        case WM_COMMAND:
                switch (LOWORD(wParam)) {
                case ID_CONNECT:    // ip를 입력 받은 경우
                        소켓 생성 및 연결
                        recv()
                        if (자리가 있다면) {
                                준비완료 버튼 활성화}
                        }

                case ID_READY:  // 준비 완료 버튼을 누른 경우
                        준비 완료 bit를 true로 수정 후 send()
                        ReceiveDataThread 생성

                case ID_CANCEL:  // 준비 취소 버튼을 누른 경우
                        준비 완료 bit를 false로 수정 후 send()
                        데이터 수신을 중지하고 스레드를 종료

                case ID_EXIT:  // 나가기 버튼을 누른 경우
                        유효 bit를 false로 수정 후 send()
                        closesocket()
        }
}
```

3.3.4. ReceiveDataThread 함수

```
DWORD WINAPI ReceiveDataThread(LPVOID arg) {
        논블로킹 소켓으로 전환

        while (준비 완료 버튼이 눌려있다면) {
                recv()

                if (게임이 시작했다면) {
                        GUI 종료 및 스레드 종료
                        블로킹 소켓으로 전환
                }
        }
}
```

3.3.5. Update 함수

```
GLvoid update(int value) {
        플레이어 정보 recv()
        장애물 정보 recv()

        플레이어 Update
        장애물 Update

        게임 종료 확인
}
```

3.3.6. Event 함수

```
void Event(int key) {
        if (게임이 시작되지 않았다면) {
```

~~준비완료, 준비취소, 나가기 send()~~

~~}~~

if (죽지 않았다면) {

KeyState 갱신 후 send()

}

}

3.3.7. 키 입력을 처리하기 위한 KeyState 자료구조

```
std::unordered_map<char, bool> keyStates = {
    {'w', false},    {'a', false},    {'d', false},    {'c', false}
};
```

3.3.8. 그 외 변경점

➔ Recv 함수 : 사용 X, GUI와 Update 함수에서 recv()를 수행하도록 변경

```
void Recv() {
    if (!isGameStart) {
        플레이어 번호, 게임 시작 여부 recv();

        if (플레이어 번호가 Invalid) {
            isConnected = false;
            소켓과 스레드 닫고 다시 IP 입력 받기;
        }
    }
    else {
        플레이어 n명의 정보 recv();
    }
}
```

➔ CreateGUIComponents 함수 : 사용 X, GuiAndClientMain 함수로 대체

```
// IP 입력, 준비 완료, 준비 취소 버튼을 나타낼 GUI 생성
void CreateGUIComponents(HWND hwnd);
```

3.4 InfoCheckThread.cpp

```
DWORD __stdcall InfoCheckThread(LPVOID arg) {
        while (1) {
                ~~ClientInfoArray에서 Read;~~
                WaitForSingleObject(WriteEvent, INFINITE);   // ClientInfoArray 갱신 대기

                if (플레이어 수 == 준비 완료된 플레이어 수) {
                        break;
                }

                SetEvent(ReadEvent);
        }

// 인게임 스레드 생성
CreateThread(InGameThread, arg);
}
```

3.5 ClientServerThread.cpp

```
void init(arg) {
        WaitForSingleObject(ReadEvent, INFINITE);

        ClientInfoArray의 비어있는 자리를 기준으로 플레이어 번호 부여

        if (게임이 시작하지 않았다면) {
                if (ClientInfoArray에 비어있는 자리가 있다면) {
                        플레이어 번호 send()
                        CreateThread(RecvThread)
                        SetEvent(WriteEvent);
                }
                else {  // ClientInfoArray에 비어있는 자리가 없다면
                        비어있는 자리가 없다는 정보 send()
                }
        }
        else {  // 게임이 시작했다면
                게임이 이미 시작되었다는 정보 send()
        }
}


DWORD WINAPI RecvThread(arg) {
        while (1) {
                패킷 recv()

                if (클라이언트가 연결을 종료했다면) {
                        WaitForSingleObject(ReadEvent, INFINITE);
                        ClientInfoArray 갱신
                        SetEvent(WriteEvent);
                }
```

```
                    if (게임이 시작하지 않았다면) {
                            WaitForSingleObject(ReadEvent, INFINITE);
                            ClientInfoArray 갱신
                            SetEvent(WriteEvent);
                    }
                    else {
                            받은 패킷을 ClientServerQueue에 push
                    }
            }
    }


    void ClientServerThread(arg) {
            init(arg);

            while (1) {
                    if (게임이 시작되었다면) {
                            게임이 시작되었다는 정보 send()

                            while (1) {
                                    if (elapsedTime > Timeout) {
                                            플레이어 정보 send()
                                            장애물 정보 send()
                                    }
                            }
                    }
            }
    }
```

3.6 InGameThread.cpp

```
DWORD __stdcall InGameThread(LPVOID arg) {
        // 플레이어 정보와 장애물 정보를 저장할 배열
        std::array<Object, 4> Players;
        std::array<Object, 10> Obstacles;

        // 플레이어 정보와 장애물 정보를 만들어 서버 → 클라이언트 배열에 삽입
        Init();

        // 게임 루프
        while (1) {
                // ClientServerQueue의 첫 번째 요소를 pop하여 ClientInfo 갱신
                PopFromClientServerQueue();

                if (모든 플레이어가 사망했다면) {
                        break;
                }

                // 클리아인트 이동, 장애물 이동
                MovePlayer(elapsedTime);
                MoveObstacle(elapsedTime);

                // 충돌 검사
                CheckPlayerObjectCollision();

                // 60분의 1초 경과 시 서버 → 클라이언트 배열 갱신
                if (totalElapsedTime > 1.0 / 60.0) {
                        RenewalServerClientArray();
                }
        }
}
```

4. 팀원 별 역할분담 및 개발일정

■ 조진영 : 5번(ClientServerThread)

일	월	화	수	목	금	토
					11/1 기획서 검수	11/2 클라이언트 코드 정리
11/3 클라이언트 코드 정리	11/4 클라이언트 코드 정리	11/5	11/6 ClientServerThread Init()	11/7 ClientServerThread Init() 계속	11/8 코드 검수	11/9 버그 픽스
11/10 졸작 회의	11/11 ClientServerThread Recv()	11/12 ClientServerThread Recv() 계속	11/13 ClientServerThread Send()	11/14 ClientServerThread Send() 계속	~~11/15~~ ~~코드 검수~~	~~11/16~~ ~~버그 픽스~~
11/17 졸작 회의	11/18 ClientServerThread ClientServerThread()	11/19 ClientServerThread ClientServerThread()	11/20 ClientServerThread ClientServerThread()	11/21	11/22 코드 검수	11/23 버그 픽스
11/24 졸작 회의	11/25 보조 및 코드 정리	11/26 보조 및 코드 정리	11/27 종설기 발표 준비	11/28 종설기 발표	11/29 코드 종합 및 복기 & 버그 픽스	11/30 버그 픽스
12/1 ~~최종 검수~~	12/2 테스트 및 버그 수정	12/3 테스트 및 버그 수정	12/4 테스트 및 버그 수정	12/5 최종 검수		

■ 원정우 : 2번(Client)

일	월	화	수	목	금	토
					11/1	11/2 클라이언트 코드 정리
11/3 클라이언트 코드 정리	11/4 클라이언트 코드 정리	11/5 Client Packet, 패킷관리 구현	11/6 Client-Server Send(), Recv()	11/7 Client-Server Loop 구조 구현	11/8 Client-Server Loop 구조 구현	11/9 졸작회의
11/10 ~~ConnectToServer 구현~~ GUICompo nents 구현	11/11 ~~ConnectToServer 구현~~ 시험	11/12 ~~ConnectToServer 구현~~ 시험	11/13 ~~SendReadySt atus 구현~~ GUICompo nents 구현	11/14 ~~SendReadySt atus 구현~~ GUICompo nents 구현	11/15 ~~Client 움직임 변경 마우스 → 키보드~~ 로비 recv(), Send() 구현	11/16 졸작회의
11/17 ~~Client 움직임 변경 마우스 → 키보드~~ 로비 recv(), Send() 구현	11/18 Client 움직임 변경 마우스 -> 키보드	11/19 ~~Client 타 플레이어 추가~~ Client 움직임 변경 마우스 -> 키보드	11/20 ~~Client 타 플레이어 추가~~ Client 움직임 변경 마우스 -> 키보드	11/21 ~~로비 recv(), Send() 구현~~ Client 타 플레이어 추가	11/22 ~~로비 recv(), Send() 구현~~ –Client 타 플레이어 추가	11/23 졸작회의
11/24 인게임 recv(), Send() 구현	11/25 인게임 recv(), Send() 구현	11/26 인게임 recv(), Send() 구현	11/27 ~~CreateGUIC omponents 구현~~ 오브젝트 이동,파괴 구현	11/28 ~~CreateGUIC omponents 구현~~ 오브젝트 이동,파괴 구현	11/29 ~~CreateGUIC omponents 구현~~ 오브젝트 이동,파괴 구현	11/30 졸작회의
12/1 ~~최종 검수~~	12/2 테스트 및 버그 수정	12/3 테스트 및 버그 수정	12/4 테스트 및 버그 수정	12/5 최종 검수		

■ 이원준 : 3번(Server), 4번(InfoCheckThread), 6번(InGameThread)

일	월	화	수	목	금	토
					11/1	11/2 졸작회의
11/3 클라이언트 코드 정리	11/4 클라이언트 코드 정리	11/5	11/6 Server.cpp	11/7 CreateInfoThread()	11/8	11/9 졸작회의
11/10 CreateClientServerThread()	11/11 InfoCheckThread.cpp	11/12	11/13 CreateInGameThread()	11/14 InGameThread.cpp	11/15	11/16 졸작회의
11/17 ~~InitInGameThread()~~ Server.cpp, InfoCheckThread.cpp 수정	11/18 ~~PopFromClientServerQueue()~~ InitInGameThread()	11/19	11/20 ~~RenewalPlayerInfo()~~ PopFromClientServerQueue()	11/21 CheckWinner()	11/22	11/23 졸작회의
11/24 ~~MovePlayer()~~ RenewalServerClientArray()	11/25 ~~BoundCheck()~~	11/26	11/27 ~~RenewalServerClientArray()~~ MovePlayer()	11/28 ~~버그 픽스~~ BoundCheck()	11/29	11/30 졸작회의
12/1 ~~최종 검수~~	12/2 테스트 및 버그 수정	12/3 테스트 및 버그 수정	12/4 테스트 및 버그 수정	12/5 최종 검수		

5. 개발환경

개발 환경 : Visual Studio 2022

개발 언어 : C++

버전 관리 : Git, Github

그래픽 라이브러리 : OpenGL

네트워크 프로토콜 : TCP/IP