**네트워크 게임 프로그래밍**

**AVOID**

**추진 계획서**

**2018182025 이승준**

**2017180008 김혁동**

**Index**

1. 애플리케이션 기획

* AVOID(2019) 게임 소개
* AVOID(network) 게임 소개
* 예상 게임 흐름도
* 조작 방법

1. High-level 디자인
2. Low-level 디자인

* 코딩 규칙
* 프로토콜(Protocol)
* 서버 함수
* 클라이언트 함수

1. 팀원 별 역할분담
2. 개발 환경
3. 개발 일정
4. **애플리케이션 기획**

* AVOID(2019) 게임 소개

텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |
| --- | --- |
| 게임 이름 | AVOID |
| 플레이 시간 | 곡 당 약 2분 |
| 조작 | 키보드, 마우스 |
| 개발 언어 | C++, 윈도우API |
| 개발 교과목 | 2019년 1학기 윈도우 프로그래밍 |
| 개발자 | 2018182025 이승준, 2018182024 이동현 |

*(타이틀 화면) (곡 선택 화면)*

AVOID는 윈도우 API를 사용하여 개발한 2D 리듬 게임입니다.

장치, 엔진이(가) 표시된 사진

자동 생성된 설명

*(인게임 화면) (스킬 사용)*

플레이어는 원하는 곡을 선택하여 게임에 입장할 수 있으며, 곡이 시작되면 음악에 맞춰 적이 회전하며 총알을 발사합니다.

플레이어는 인게임에서 스페이스 바를 눌러 마나를 소모하여 총알의 속도를 줄일 수 있습니다.

음악이 끝날 때까지 일정 체력 이상으로 버티게 된다면 플레이어의 승리입니다.

* AVOID(network) 게임 소개

기존 AVOID를 이용하여 최대 3인까지 이용 가능한 멀티플레이 경쟁 리듬 게임을 제작하고자 합니다.

플레이어가 곡을 선택하면 바로 인게임으로 넘어가지 않고, Player Waiting Scene에서 다른 플레이어들의 접속을 기다립니다.

엔터 버튼을 통해 모든 플레이어들의 접속을 알리거나, 3인의 플레이어가 모이면 인게임으로 넘어갑니다. 인게임에는 3명의 플레이어가 모두 표시됩니다.

각자 플레이어는 음악에 맞춰 등장하는 적 총알을 피하며 최대한 많은 체력으로 버텨야 합니다. 기존 AVOID에서는 일정 체력 이상으로 버티면 클리어로 판정했지만, AVOID(network)는 경쟁 게임이니만큼 곡 종료 시 더 많은 체력을 가지고 있는 플레이어를 승자로 판정하고자 합니다.

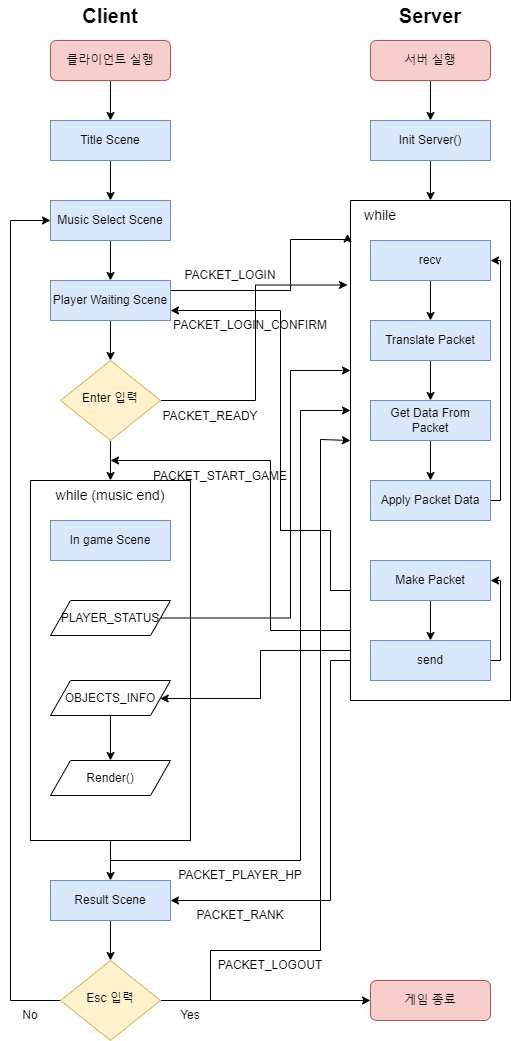
스페이스 바를 이용해 총알을 느리게 하는 스킬 역시 그대로 사용할 수 있습니다. 이 경우 상대에게도 총알이 느려지기 때문에, 상대의 예측에 혼란을 줘 전략적인 플레이가 가능하게 됩니다.

* 예상 게임 흐름도



* 조작
  + 플레이어 이동 – 마우스
  + 스킬 사용 – 스페이스 바
  + 곡 선택 – 키보드 ←, →
  + 선택 – 엔터

1. **High-level 디자인**



게임에는 Title Scene, Music Select Scene, Player Waiting Scene, In Game Scene, Result Scene의 5가지 Scene이 있습니다.

게임을 시작하면 Title Scene이 뜨고, 키를 눌러 조작을 시작하면 Music Select Scene으로 이동하여 음악을 선택하게 됩니다.

음악을 선택하면 Player Waiting Scene에서 서버에 접속하여 같이 플레이를 할 플레이어를 기다리게 됩니다. 기본적으로 총 3명이 접속하게 되면 In Game Scene으로 넘어가 게임을 플레이할 수 있도록 하나, 3명이 접속하기 전 Enter를 통해 플레이어의 수가 부족한 경우에도 플레이를 할 수 있도록 합니다.

In Game Scene에서는 클라이언트의 데이터를 서버에 넘겨주어 서버에서 충돌체크를 포함한 클라이언트 간의 상호작용이나 클라이언트 각각의 움직임을 지정해 준 뒤, 다시 클라이언트로 보내 화면에 표시하도록 합니다.

게임이 끝난 후 서버가 각각의 클라이언트에게 플레이어의 HP 정보를 받아오고, 해당 HP 정보를 바탕으로 순위를 계산해 승자를 결정한 뒤, 클라이언트에게 알려줍니다.

1. **Low-level 디자인**

* **코딩 규칙**
  + 클래스: 첫번째 문자가 대문자(Pascal Case)
  + 함수: 첫번째 문자가 대문자(Pascal Case)
  + 함수 인자: 첫 단어만 소문자(Camel Case)
  + 변수: 첫 단어만 소문자(Camel Case), 클래스 멤버 변수일 경우 맨 앞에 ‘m\_’ 추가로 붙임
  + 전역 변수: 모든 문자가 대문자
* **프로토콜(Protocol)**
  + **서버 정보**

constexpr short SERVER\_PORT = 9000

constexpr char SERVER\_ADDR = “127.0.0.1”

constexpr int BUF\_SIZE = 2048

constexpr int MAX\_USER = 3

* + **패킷 타입 (Client to Server)**

constexpr char CS\_PACKET\_LOGIN = 1

constexpr char CS\_PACKET\_READY = 2

constexpr char CS\_PACKET\_PLAYER\_STATUS = 3

constexpr char CS\_PLAYER\_HP = 4

constexpr char CS\_PACKET\_LOGOUT = 100

* + **패킷 (Client to Server)**

#pragma pack(1)

struct packet {

unsigned char size;

char type;

}

모든 packet은 이 구조체를 상속받아 사용합니다. 서버는 패킷이 도착하면 맨 앞 2바이트를 읽어 패킷의 크기와 패킷의 타입을 확인하고, 패킷의 크기 - 2바이트를 추가적으로 읽어 필요한 패킷의 정보를 확보합니다.

struct cs\_packet\_login : public packet {

char selectedMusic;

}

서버에 클라이언트가 현재 접속 중이며, 게임 시작을 위해 대기중임을 알리기 위해 보내는 패킷입니다.

struct cs\_packet\_ready : public packet {

}

최대 플레이어 수인 3인이 채워지기 전에 게임을 시작하고자 할 때 클라이언트에서 서버로 보내는 패킷입니다.

struct cs\_packet\_player\_status : public packet {

char x, y;

char playerID;

bool isSkill;

}

플레이어의 현재 위치와 스킬 사용 여부를 서버에 전송하기 위해 사용하는 패킷입니다. 매 프레임마다 전송합니다.

struct cs\_packet\_player\_hp : public packet {

char hp;

}

플레이어의 현 체력을 서버에 알립니다.

struct cs\_packet\_logout : public packet {

char playerID;

}

플레이어가 접속을 종료했음을 서버에 알립니다.

* + **패킷 타입 (Server to Client)**

constexpr char SC\_PACKET\_LOGIN\_CONFIRM = 1

constexpr char SC\_PACKET\_START\_GAME = 2

constexpr char SC\_PACKET\_OBJECTS\_INFO = 3

constexpr char SC\_PACKET\_MUSIC\_END = 4

constexpr char SC\_PACKET\_RANK = 5

constexpr char SC\_PACKET\_LOGOUT\_OK = 100

* + **패킷 (Server to Client)**

struct sc\_packet\_login\_confirm : public packet {

char playerID;

}

클라이언트가 접속 사실을 서버에 알리면 서버는 클라이언트마다 서로 다른 ID를 전송합니다. 이는 플레이어의 초기 시작 위치 등을 정하는데 사용됩니다.

struct sc\_packet\_start\_game : public packet {

char playerNum;

}

3명의 플레이어가 접속하거나 cs\_packet\_ready 패킷을 송신할 경우 서버는 클라이언트에 게임 시작을 알립니다.

struct coord {

short x, y;

}

struct sc\_packet\_objects\_info : public packet {

char playerNum;

coord playerCoord[playerNum];

char bulletNum;

coord bulletCoord[bulletNum];

}

서버는 접속한 클라이언트의 cs\_packet\_player\_status 패킷을 통해 플레이어의 위치와 스킬 사용 여부를 파악합니다. 이를 바탕으로 총알 오브젝트의 현 위치를 계산하고 클라이언트에 다시 전송해 줍니다.

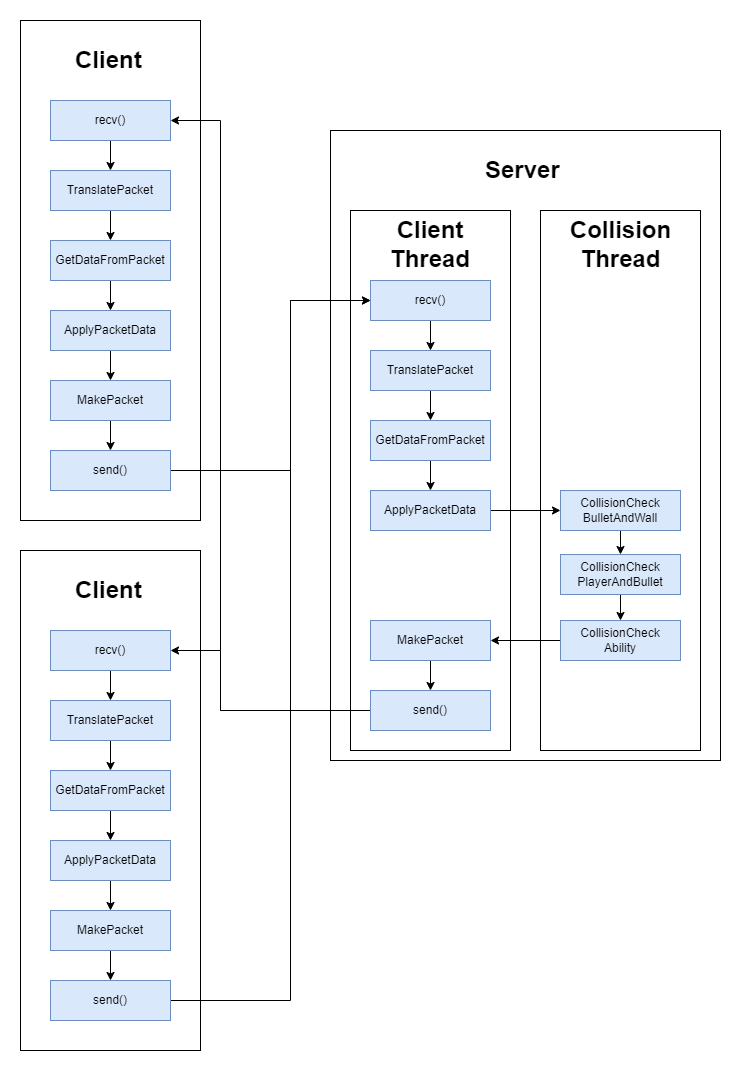
struct sc\_packet\_rank : public packet {

char rank;

}

음악 종료 시 클라이언트로부터 전송받은 체력 정보를 바탕으로 클라이언트에 순위를 전송합니다.

* **플로우 차트**



* **서버 함수**

기본적으로 TranslatePacket 함수를 반복하여 패킷이 어떤 타입인지 확인한 후, GetDataFromPacket 함수를 통해 패킷 데이터에 따라서 실제 데이터를 받아온 뒤, ApplyPacketData 함수를 통해 데이터를 패킷 타입에 따른 실제 데이터 타입으로 형변환을 시켜 실제 데이터에 적용시킵니다.

|  |
| --- |
| void InitServer() |
| 윈도우 소켓 초기화, 소켓 생성, 소켓 주소 할당 |
| char TranslatePacket(char\* packetBuf) |
| 첫 두 바이트 버퍼를 읽어 패킷 데이터의 정보를 알아오기 위해 사용합니다. |
| void GetDataFromPacket(char\* dataBuf, char packetType) |
| 실제로 데이터를 받아오기 위한 함수입니다. 이 단계에서는 패킷 데이터를 받아오기만 합니다. |
| void ApplyPacketData(char\* dataBuf, char packetType) |
| 받은 데이터를 활용해 서버 상의 데이터들에 적용시킵니다. 이 단계에서 데이터들에 대한 동기화도 진행됩니다. |
| void MakePacket(char\* dataBuf, char packetType) |
| 클라이언트로 보낼 데이터를 패킷 데이터로 만들기 위한 함수입니다. data\_buf에 데이터가 저장됩니다. |
| void CollisionCheckBulletAndWall() |
| 총알들이 벽면과 충돌했는지를 검사합니다. 충돌하였거나, 생성된 이후 0.5초 이상 지났다면 총알을 파괴합니다. |
| void CollisionCheckPlayerAndBullet() |
| 플레이어가 총알과 충돌했는지를 검사합니다. 충돌한 경우, 플레이어의 hp를 감소시키고 2초간 무적 상태로 만듭니다. |
| void CollisionCheckAbility() |
| 총알이 플레이어의 능력과 충돌했는지를 검사합니다. 충돌한 경우, 충돌한 총알의 속도를 감소시킵니다. |
| DWORD WINAPI Client\_Thread(LPVOID arg) |
| 클라이언트에서 서버와 연결한 뒤 데이터를 주고받을 때에 사용할 스레드 함수입니다.  최대 3개의 스레드가 생성되어 실행이 되며, 기본적으로 데이터를 받아 저장한 뒤, 충돌 판정을 실행하고, 다시 데이터를 받아오는 순서로 진행됩니다.  이 함수에서는 기본적으로 임계 영역을 설정하여 동기화를 실행합니다.  임계영역은 구현이 쉽고 속도가 빠르기 때문에 순서에 상관없는 공유 데이터들의 처리는 임계영역을 이용할 계획입니다. |
| DWORD WINAPI Collision\_Thread(LPVOID arg) |
| 플레이어와 총알, 스킬과 총알 사이 간의 충돌 처리를 담당합니다. 이 함수에서는 이벤트를 사용하여 동기화를 실행합니다.  최대 3개의 클라이언트에서 데이터를 받아온 뒤, 충돌 처리를 하고, 다시 클라이언트로 보내는 흐름의 순서가 있으므로 이벤트를 사용할 계획입니다. |

* **클라이언트 함수**

클라이언트도 서버와 같은 방식으로 전송받은 패킷을 해석하고, 다시 서버로 전송합니다.

|  |
| --- |
| void InitClient() |
| 윈도우 소켓 초기화, 소켓 생성, 소켓 주소 할당 등을 담당하는 함수입니다. |
| char TranslatePacket(char\* packetBuf) |
| 첫 두 바이트 버퍼를 읽어 패킷 데이터의 정보를 알아오기 위해 사용합니다. |
| void GetDataFromPacket(char\* dataBuf, char packetType) |
| 실제로 데이터를 받아오기 위한 함수입니다. 이 단계에서는 패킷 데이터를 받아오기만 합니다. |
| void ApplyPacketData(char\* dataBuf, char packetType) |
| 받은 데이터를 활용해 클라이언트 상의 데이터에 적용시킵니다. |
| void MakePacket(char\* dataBuf, char packetType) |
| 클라이언트로 보낼 데이터를 패킷 데이터로 만들기 위한 함수입니다. dataBuf에 데이터가 저장됩니다. |

1. **팀원 별 역할 분담**

|  |  |  |
| --- | --- | --- |
| 함수 | 이승준 | 김혁동 |
| Server - void InitServer() |  | O |
| Server - char TranslatePacket(char\* packetBuf) | O |  |
| Server - void GetDataFromPacket(char\* dataBuf, char packetType) |  | O |
| Server - void ApplyPacketData(char\* dataBuf, char packetType) | O | O |
| Server - void MakePacket(char\* dataBuf, char packetType) | O | O |
| Server - void CollisionCheckBulletAndWall() |  | O |
| Server - void CollisionCheckPlayerAndBullet() | O |  |
| Server - void CollisionCheckAbility() |  | O |
| Server - DWORD WINAPI Client\_Thread(LPVOID arg) |  | O |
| Server - DWORD WINAPI Collision\_Thread(LPVOID arg) |  | O |
| Client - void InitClient() | O |  |
| Client - char TranslatePacket(char\* packetBuf) | O |  |
| Client - void GetDataFromPacket(char\* dataBuf, char packetType) |  | O |
| Client - void ApplyPacketData(char\* dataBuf, char packetType) | O | O |
| Client - void MakePacket(char\* dataBuf, char packetType) | O | O |

1. 개발 환경

* 운영체제
  + Windows 10
* 개발 도구
  + Visual Studio 2019
  + Photoshop
  + Microsoft Word
  + diagram.net
* 개발 언어
  + C++
  + WinAPI
* VCS
  + GitHub

1. **개발 일정**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 |
|  | 10월 31일 | 11월 1일 | 2일 | 3일 | 4일 | 5일 |
| 이승준 | 추진 계획서 작성 | | | | 네트워킹을 위해  클라이언트 수정 | |
| 김혁동 |
|  | 7일 | 8일 | 9일 | 10일 | 11일 | 12일 |
| 이승준 | 프로토콜 추가  서버 프로그램 제작 | | 클라이언트 recv(), send() 작성 | 클라이언트 Translate Packet() 작성 | 서버 Translate Packet() 작성 | 미비사항 보강 |
| 김혁동 | 서버 recv(), send() 작성 | 클라이언트 Get Data From Packet() 작성 | 서버 Get Data From Packet() 작성 |
|  | 14일 | 15일 | 16일 | 17일 | 18일 | 19일 |
| 이승준 | 서버, 클라이언트 Apply Packet Data() 작성 | | 클라이언트 MakePacket()  함수 작성.  패킷 정상 동작 여부 확인 | Player Waiting Scene 작성 | In Game Scene클라이언트 동기화 | 미비사항 보강 |
| 김혁동 | 서버에서 Bullet 정보를 파일로부터 읽어 들이도록 수정 | |
|  | 21일 | 22일 | 23일 | 24일 | 25일 | 26일 |
| 이승준 | Objects Info Packet 수신 시 처리 | Object 수신 데이터 Render() | CollisionCheck  PlayerAndBullet()  함수 작성 | 충돌 Packet 수신 시 클라이언트 처리 | In Game Scene 정상 작동 확인 | 미비사항 보강 |
| 김혁동 | Objects Info Packet 송신 처리 | | CollisionCheck  BulletAndWall() 함수 작성 | CollisionCheck  Ability()  함수 작성 |
|  | 28일 | 29일 | 30일 | 12월 1일 | 2일 | 3일 |
| 이승준 | 클라이언트 순위 표시 | Result Scene -> Music Select Scene 변환 처리 | | 자연스러운 프로그램 동작 검수 | 여러 상황에서의 버그 검수 | 미비사항 보강 |
| 김혁동 | Rank Packet 전송 | 서버에서 클라이언트 종료 시 처리 | |
|  | 5일 | 6일 | 7일 | 8일 |  |  |
| 이승준 | 리포트 검수 및 동작 테스트 | | | 프로젝트 발표 |  |  |
| 김혁동 |  |  |