온라인 게임 프로그래밍

기말프로젝트

202213037 이수연
202013058 허승윤

프로그램 설명

2인 플레이 방식의 알까기 게임

- 플레이어는 회원가입/로그인이 가능하며 서버에서 관리
- 마우스 입력 조작 방식 및 턴제 플레이

작업 분배 방식

- 인게임-서버
- 데이터베이스-서버

Client - Server

허승윤

A. 알 날리기 및 경로 표시

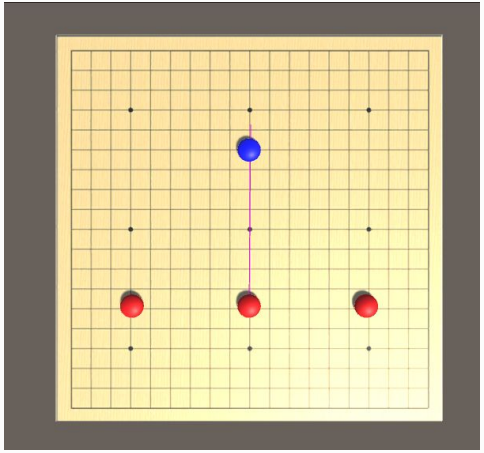
B. 접속한 플레이어 수 가져오기

C. 실시간 처리 기능 (능력 부족)

알 날리기 및 경로 표시

```
private void OnMouseUp()
{
    if (tag == "Red")
    {
        float angle = 0;

        // 마우스 포지션과 오브젝트 거리 지정
        Vector3 mousePosition = Input.mousePosition;
        Vector3 scrSpace = Camera.main.WorldToScreenPoint(transform.position);
        Vector3 offset = new Vector3(scrSpace.x - mousePosition.x, 0, scrSpace.y - mousePosition.y);

        float distance = Mathf.Sqrt(offset.x * offset.x + offset.z * offset.z); // 벡터의 크기 계산

        if (distance > maxDragDistance)
        {
            offset = offset.normalized * maxDragDistance;
            distance = maxDragDistance;
        }

        offset /= distance; // 정규화

        if (offset.z > 0)
            angle += Mathf.Rad2Deg * Mathf.Acos(offset.x);
        else
            angle += 360 - Mathf.Rad2Deg * Mathf.Acos(offset.x);

        offset *= distance;

        rigidbody.AddForce(offset * hitPower / 2, ForceMode.Impulse);

        StartCoroutine(Shoot());
    }
}

IEnumerator Shoot()
{
    yield return new WaitForSeconds(0.1f);

    while (rigidbody.velocity.magnitude > 0.1f && transform.position.y >= 0)
        yield return new WaitForSeconds(0.1f);

    if(GameManager.turn % 2 == 1)
        GameObject.Find("Main Camera").GetComponent<GameManager>().player2Turn();
    else
        GameObject.Find("Main Camera").GetComponent<GameManager>().player1Turn();
}
```

```
public class RenderLine : MonoBehaviour
{
    private bool clicked = false;
    private LineRenderer lineRenderer;

    void Start()
    {
        lineRenderer = gameObject.AddComponent<LineRenderer>();
        lineRenderer.GetComponent<LineRenderer>();

        lineRenderer.SetPosition(0, Vector3.zero);
        lineRenderer.SetPosition(1, Vector3.zero);

        lineRenderer.SetColors(Color.red, Color.yellow);
        lineRenderer.SetWidth(0.1f, 0.1f);
    }

    private void FixedUpdate()
    {
        if (clicked && tag == "Red")
        {
            Vector3 mousePosition = Input.mousePosition;

            Vector3 scrSpace = Camera.main.WorldToScreenPoint(transform.position);

            Vector3 offset = new Vector3(scrSpace.x - mousePosition.x, 0, scrSpace.y - mousePosition.y);

            lineRenderer.SetPosition(0, transform.position);
            lineRenderer.SetPosition(1, transform.position + offset / 5);
        }
    }

    private void OnMouseDown()
    {
        clicked = true;
    }

    private void OnMouseUp()
    {
        clicked = false;
        lineRenderer.SetPosition(0, transform.position);
        lineRenderer.SetPosition(1, transform.position);
    }
}
```



마우스 입력으로 오브젝트를 드래그하고, 드래그한 거리와 오브젝트 사이의 거리를 계산한 후 거리와 가해줄 힘만큼 곱해준다. RenderLine도 똑같은 방식을 사용하여 계산해서 그려준다.

접속한 플레이어 수 가져오기_client

```
[StructLayout(LayoutKind.Sequential, Size = 6)]
struct P_RoomEnterResponse
{
    [MarshalAs(UnmanagedType.I2)]
    public short result;

    [MarshalAs(UnmanagedType.I4)]
    public int PlayerNum;
}
```

```
switch ((E_PACKET)packetId)
{
    case E_PACKET.ROOM_ENTER_RESPONSE:
        P_RoomEnterResponse roomEnterResponse = UnsafeCode.ByteArrayToStructure<P_RoomEnterResponse>(packet.data);
        PlayerNum = roomEnterResponse.PlayerNum;
        Debug.Log($"ROOM_ENTER_RESPONSE result={roomEnterResponse.result}");
        Debug.Log($"ROOM_ENTER_RESPONSE PlayerNum={PlayerNum}");
        break;
```

클라이언트 Packet.cs에서 코드를 추가해준 후 Match.cs에서 값을 변수에 저장하려는 과정에서 문제가 생겨 정상적으로 동작하지 않았다.

접속한 플레이어 수 가져오기_server

```
struct ROOM_ENTER_RESPONSE_PACKET : public PACKET_HEADER
{
    INT16 Result;
    INT32 PlayerNum;
    //char RivaluserID[MAX_USER_ID_LEN + 1] = { 0, };
    ROOM_ENTER_RESPONSE_PACKET() : Result{ 0 }, PlayerNum{ 0 }, PACKET_HEADER(sizeof(*this), PACKET_ID::ROOM_ENTER_RESPONSE) {}
};
```

```
// Room::EnterUser()에서 입장하는 유저에게 방안 유저 리스트를 전송한다
auto enterResult = mRoomManager->EnterUser(roomNumber, pReqUser);
{
    ROOM_ENTER_RESPONSE_PACKET roomEnterResPacket;
    roomEnterResPacket.Result = enterResult;

    roomEnterResPacket.PlayerNum = pRoom->GetCurrentUserCount();
    std::cout << "PlayerNum : " << roomEnterResPacket.PlayerNum << std::endl;
    SendPacketFunc(clientIndex_, sizeof(ROOM_ENTER_RESPONSE_PACKET), (char*)&roomEnterResPacket);
}
```

```
[송신 완료] bytes : 7
[OnReceive] 클라이언트: Index(1), dataSize(9)
PlayerNum : 1
[송신 완료] bytes : 11
Response Packet Sended[송신 완료] bytes : 46
```

Packet.h 파일에 PlayerNum의 수치를 추가해준다. 이후 PacketManager안에 있는 ProcessEnterRoom 함수에서 현재 유저 수를 클라이언트로 넘겨준다. 이제 클라이언트에서 서버로 진입하는 과정에서 문제없이 출력이 되었다.

실시간 처리 구상

1. 모든 플레이어는 항상 빨간색 돌만 사용한다.
2. 모든 플레이어는 접속하였을 경우 자신의 돌이 생성된다.
3. 2번째 플레이어가 접속하였을 때 2번 플레이어 화면에 생성된 돌은 1번 플레이어 화면에서는 대칭되도록 생성되도록 만들어준다. 2번 플레이어는 들어오자마자 대칭되도록 조건을 걸어둔다.
4. 플레이어가 빨간색 돌을 선택하여 돌을 움직였을 경우 그 계산식을 저장해두어 서버로 전달한다.
5. 서버에서 이전 빨간색 돌의 위치에 Vector3 {-1.0f, 0, -1.0f } 곱하여 이전 돌의 위치와 대칭되는 곳의 위치 값을 넘겨준다.
6. 클라이언트에서 위치 값을 전달받고 같은 위치에 있는 돌에 저장해둔 계산식을 사용하여 가해진 힘의 반대편으로 날려 기존 위치와 같은 위치에 위치하도록 해준다.

실시간 처리_client

```
case E_PACKET.UPDATE_BALL_POSITION:
    P_UpdateBallPosition updateBallPosition = UnsafeCode.ByteArrayToStructure<P_UpdateBallPosition>(packet.data);
    MoveBall(updateBallPosition.child_index, updateBallPosition.ballPos);
    Debug.Log("Update Ball Position : " + updateBallPosition.ballPos);
    break;
```

```
if (local)
{
    GameObject teamSet = Instantiate(prefabTeamSet);
```

```
private void MoveBall(int index, Vector3 ballPos)
{
    // 파란색 오브젝트 생성 후 위치에 옮겨준다.
    GameObject EnemySet = Instantiate(prefabBlueBall);
    EnemySet.transform.position = ballPos;
}
```

```
public class BallPositionInfo : MonoBehaviour
{
    GameObject myBall;
    int childCount;
    Vector3[] myPosition;

    public void SendBallPosition()
    {
        Debug.Log("SendBallPosition Function Active");

        myBall = GameObject.Find("MyTeam");
        childCount = myBall.transform.childCount;
        myPosition = new Vector3[childCount];

        // 자식 객체의 위치값을 저장
        for (int i = 0; i < childCount; i++)
        {
            Transform childTransform = transform.GetChild(i);
            myPosition[i] = childTransform.position;
        }

        P_BallPosition ballPosition = default;

        for(int i = 0; i < childCount; i++)
        {
            ballPosition.player_id = LocalPlayerInfo.ID;
            ballPosition.child_index = i;
            ballPosition.ballPos = myPosition[i];

            Client.TCP.SendPacket2(E_PACKET.BALL_POSITION, ballPosition);
        }
    }
}
```

클라이언트에서 처리하는 플레이어가 접속한 경우 돌을 생성해주었으며, 상대가 접속한 경우 모든 돌의 위치를 대칭 위치에 보이도록 하는 기능을 만드려다 실패하였다.

실시간 처리_server

```
// Ball Position
BALL_POSITION,
UPDATE_BALL_POSITION,
```

```
struct BALL_POSITION : public PACKET_HEADER
{
    INT64 userUUID;
    INT32 childIndex;
    Vector3 ballPos;

    BALL_POSITION() : PACKET_HEADER(sizeof(*this), PACKET_ID::BALL_POSITION) {}
};

struct UPDATE_BALL_POSITION : public PACKET_HEADER
{
    INT64 userUUID;
    INT32 childIndex;
    Vector3 ballPos;

    UPDATE_BALL_POSITION() : PACKET_HEADER(sizeof(*this), PACKET_ID::UPDATE_BALL_POSITION) {}
};
```

```
Vector3 UpdateBallPosition(Vector3 ballPos)
{
    Vector3 v = { -1.0f, 0.0f, -1.0f };
    Vector3 result = { ballPos.x * v.z,ballPos.y,ballPos.z * v.z };
    return result;
}
```

```
void PacketManager::ProcessBallPosition(UINT32 clientIndex_, UINT16 packetSize_, char* pPacket_)
{
    UNREFERENCED_PARAMETER(packetSize_);
    UNREFERENCED_PARAMETER(pPacket_);

    auto BallPosition = reinterpret_cast<BALL_POSITION*>(pPacket_);

    if (BallPosition->userUUID != clientIndex_)
    {
        printf("[ProcessPlayerMovement] userUUID(%lld) != clientIndex_(%ld)\n", BallPosition->userUUID, clientIndex_);
        return;
    }

    printf("[ProcessPlayerMovement] userUUID(%lld), index=%d dx=%f, dy=%f \n",
        BallPosition->userUUID, BallPosition->childIndex, BallPosition->ballPos.x, BallPosition->ballPos.z);

    auto reqUser = mUserManager->GetUserByConnIdx(clientIndex_);
    auto roomNum = reqUser->GetCurrentRoom();

    auto pRoom = mRoomManager->GetRoomByNumber(roomNum);
    if (pRoom == nullptr)
    {
        printf("[ProcessPlayerMovement] pRoom == nullptr userUUID(%lld), roomNum(%d)\n", BallPosition->userUUID, roomNum);
        return;
    }

    UPDATE_BALL_POSITION updateBall;
    updateBall.childIndex = BallPosition->childIndex;
    updateBall.ballPos = reqUser->UpdateBallPosition(BallPosition->ballPos);

    pRoom->SendToAllUser(updateBall.PacketLength, (char*)&updateBall, clientIndex_, false);
}
```

앞의 구상에서 나온 5번 돌의 위치를 받아온 후 Vector3 {-1.0f, 0, -1.0f } 곱하여 이전 돌의 위치와 대칭되는 곳의 위치 값을 넘겨주는 과정을 진행하려 하였다.

Server - DB

이수연

A. Redis 응용

B. 회원가입 처리

C. 회원가입 / 로그인 실패시 클라이언트에 출력

D. Protobuf로 유저 DB 관리

E. 유저 본인 정보 불러오기 (구현 실패)

Redis 응용

이수연

기존 Redis Manager, Redis Conn 의 구조를 변경하지 않고

확장하여 관리 기능 및 set() 구현

```
bool set(const std::string& key, const std::string& value) {
    if (!_connected || !_redCtx)
    {
        _errStr = _errDes[ERR_NO_CONNECT];
        return false;
    }

    bool ret = false;
    redisReply* reply = redisCmd("SET %s %s", key.c_str(), value.c_str());

    if (_getError(reply))
    {
        ret = false;
    }
    else
    {
        ret = true;
    }


    return ret;
}
```

회원가입

이수연

확장시킨 Redis 함수를 이용하여 작업 수행

DB에 없는 아이디 => Redis에 저장
DB에 있는 아이디 => 추가 작업 수행 X

로그인/회원가입 클라이언트 처리

이수연

로그인 오류 시, 서버에서 오류코드를 패킷에 저장하여 전달

이를 클라이언트에서 분석하여 화면에 출력

```
void PacketManager::ProcessLoginDBResult(UINT32 clientIndex_, UINT16 packetSize_, char* pPacket_)
{
    printf("ProcessLoginDBResult. UserIndex: %d\n", clientIndex_);

    auto pBody = (RedisLoginRes*)pPacket_;

    LOGIN_RESPONSE_PACKET loginResPacket;

    if (pBody->Result == (UINT16)ERROR_CODE::NONE)
    {
        //로그인 완료로 변경한다
        auto pUser = mUserManager->GetUserByConnIdx(clientIndex_);
        pUser->SetLogin(pBody->UserID);

        loginResPacket.Result = clientIndex_;
        loginResPacket.IsSucceed = 1;
    }
    else {
        loginResPacket.Result = pBody->Result;
        loginResPacket.IsSucceed = 0;
    }

    SendPacketFunc(clientIndex_, sizeof(LOGIN_RESPONSE_PACKET), (char*)&loginResPacket);
}
```

Protobuf로 유저 정보 관리

이수연

유저 정보 관리 체계 (Redis)

기존 : key(아이디) - value(비밀번호)
변경 : key(아이디) - value(protobuf)

ProtoBuf를 이용하여 더 많은 정보 저장

- 아이디, 비밀번호, 닉네임, 플레이 횟수등

유저 정보 확인 (구현 실패)

이수연

1. 유저 정보를 요청하는 패킷 서버에 전송
2. uid를 이용해 Redis에서 유저 정보 받아오기
3. 해당 정보 protobuf를 이용해 직렬화
4. 패킷 전송
5. 클라이언트에서 protobuf로 역직렬화 => 유니티 내에서 오류 발생
6. 받아온 정보 화면에 출력

Thanks