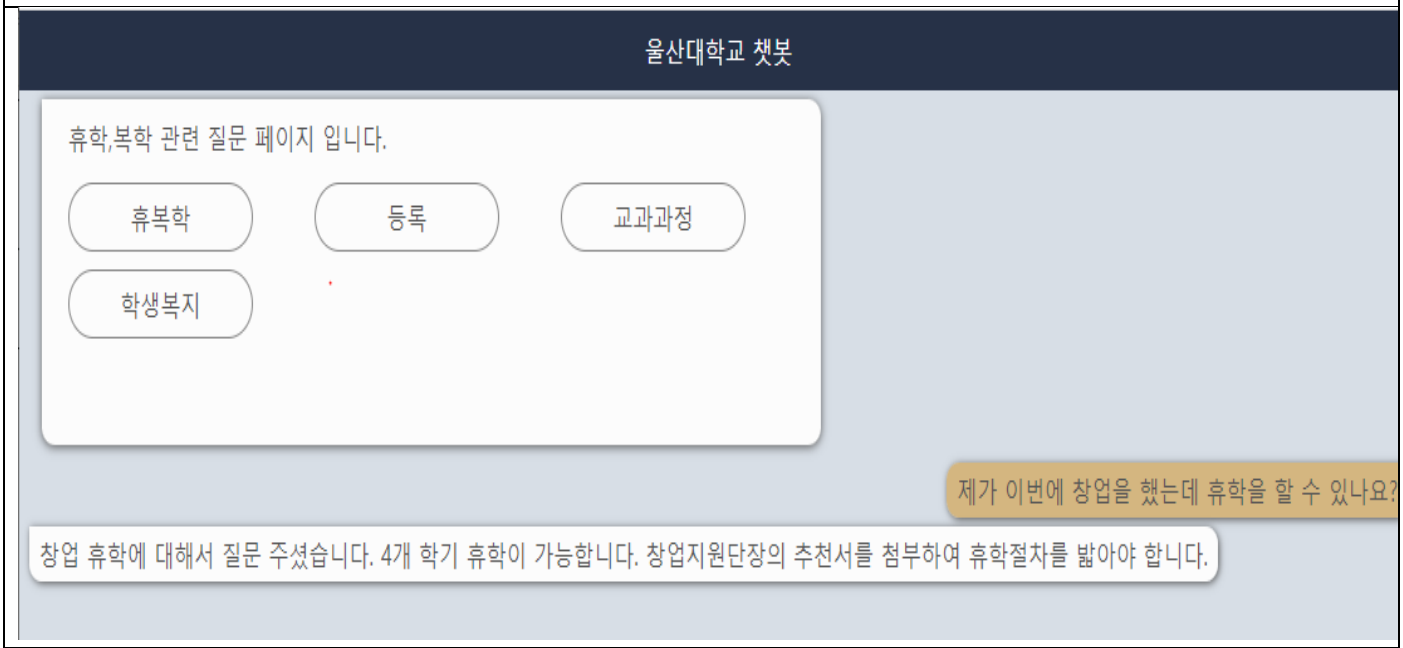
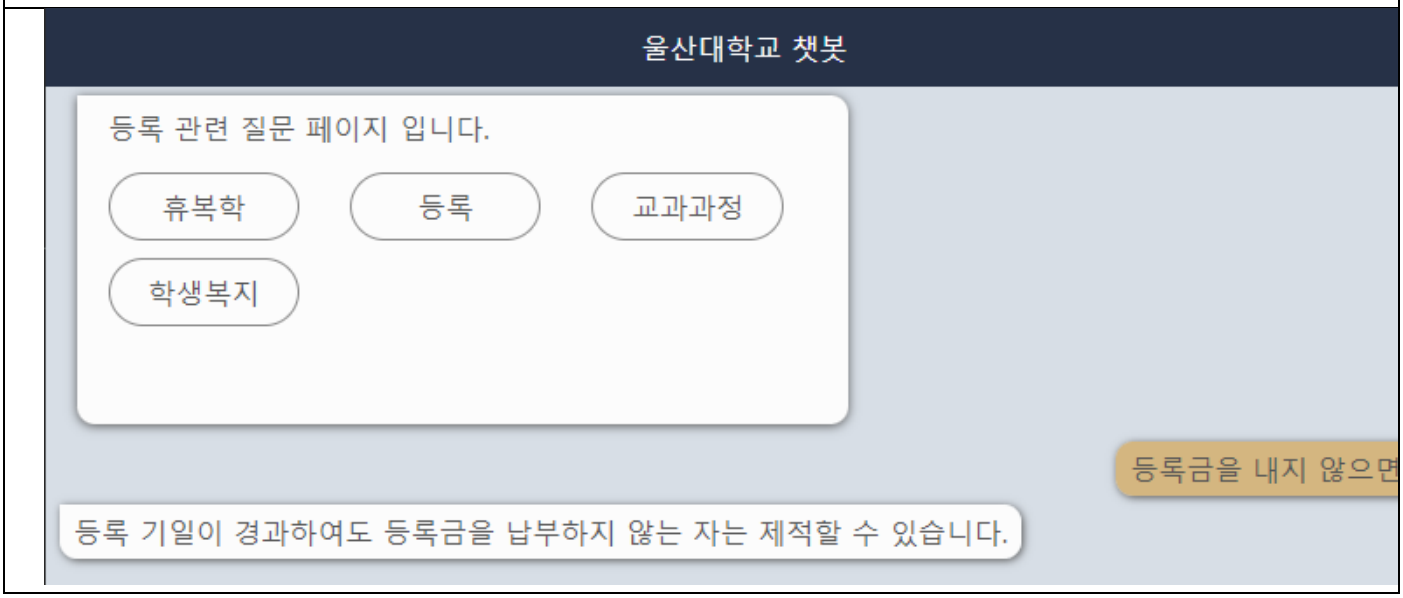
포트폴리오

프로젝트 – 울산대학교 챗봇 시스템 개발

개발 기간	2021.06.30 ~ 2022.06.09
개발 환경	Python3 / hugging face – KoElectra / pycharm – professional -
개발 인원	3명 -> 2명 (도중에 한명 휴학)
요 약	흩어져있는 울산대학교 Q&A를 챗봇을 통해서 한 곳으로 통합
기능	**1) 휴학복학에 관한 Q&A** **2) 등록에 관한 Q&A** **3) 사용자의 질문이 1000개가 차면 자동으로 학습**
실행 화면	





1. 개발 기획

MileStone	21.06	21.07	21.08	21.09	21.10	21.11	21.12	22.01	22.02	22.03	22.04	22.05
분석												
아이디어 회의												
수요 조사												
현재기술 조사												
필요지식 스터디												
설계												
개발 모델 결정												
데이터 확보												
프론트엔드 제작												
백엔드 제작												
AI 모델 설계												
추가 기능 논의												
테스트, 디버그												
배포												

1.1 요구사항 :

기능적 요구사항

- 시스템은 사용자에게 질문을 입력 할 수 있는 영역을 제공
- 시스템은 사용자가 질문을 입력 하면 적절한 대답을 출력
- 사용자가 원하는 대답을 받지 못했을 때 시스템은 다른 답변 또는 솔루션 제시

프로덕트 요구사항 – 유용성

- 사용자에게 질문 입력 완료를 Enter 키 또는 마우스 클릭으로 표현 할 수 있게 제공
- UI의 채팅 박스는 수직적으로 밑으로 계속 쌓음. 이때 화면의 세로 길이는 증가한다
- 사용자의 입력 또는 시스템의 출력시 자동으로 스크롤바가 제일 밑으로 이동

프로덕트 요구사항 – 효율성

- 사용자가 질문 입력 후 답변 받는데 까지의 시간은 2sec 이내

프로덕트 요구사항 – 신뢰성

- 대답가능한 영역을 축소하더라도 답변에 대한 정확도를 높임
- 변동성이 있는 답변은 울산대학교 공식 홈페이지로 연결을 유도하거나 상담원에게 연결을 유도

조직 요구사항 – 배포

- Django를 이용하여 웹페이지를 만든 후 배포 및 서비스
- 배포는 AWS의 EC2를 사용

조직 요구사항 – 구현

- Python을 메인 언어로 사용

- 딥러닝 라이브러리는 pytorch 사용
- Data크롤링은 울산대학교 정보통신원에서 제공 받되 부족한 것은 BeautifulSoup와 Selenium을 혼합해서 해결

1.2 타사 아이템 분석

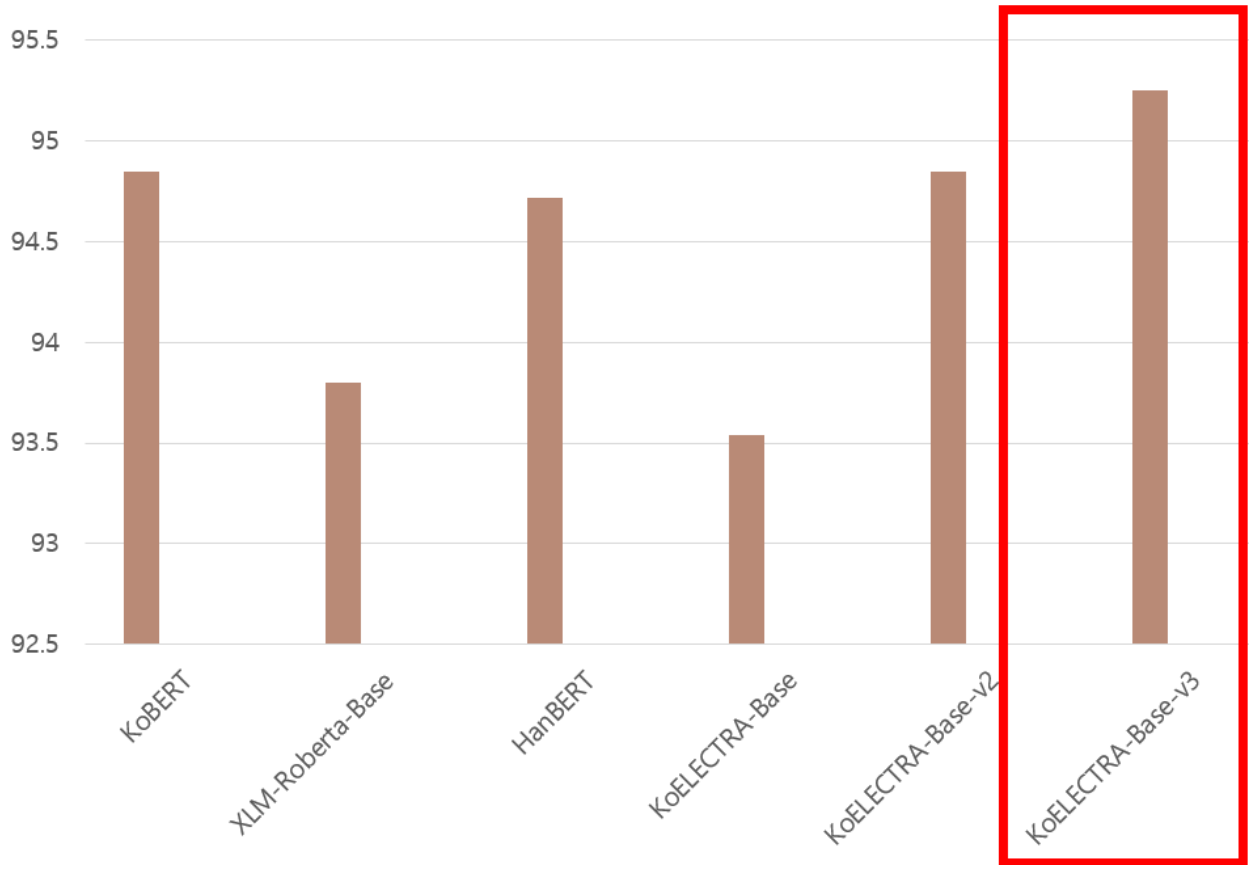
단위 : 1000원

	의뢰 비용	월 지속 비용
채x톡	문의 필요	70
깃x챗	문의 필요	50
단x AI	문의 필요	30
Cloxx xxring	2000	100

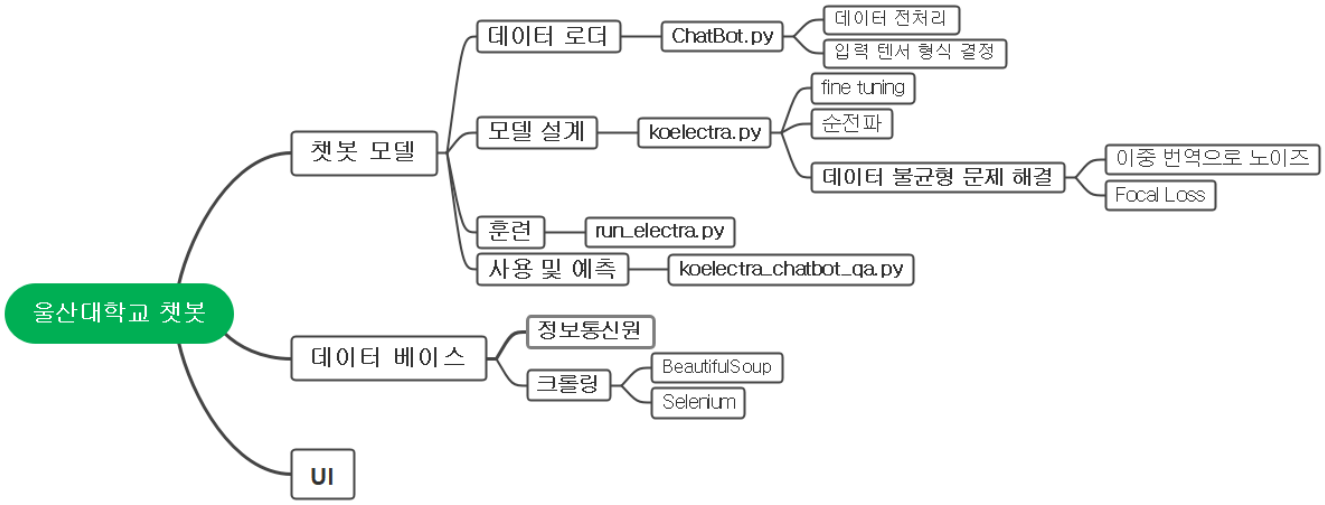
- 초기 설계 비용이 꽤 들어감
- 유지에도 비용이 다소 들어감
- 기존 챗봇 연구는 Bert와 GPT 중심의 연구

차별점 :

- 기존 챗봇은 Bert 모델이 중심이었음 그러나 이 연구에서 사용한 기반기술은 Bert보다 높은 성능을 지니고 있는 최신 기술임
- 학부생이 자체 개발한 기술로서 제작 단가가 저렴
- 데이터가 쌓이면서 유지보수를 하면 무궁무진한 연구 잠재력이 있음



1.3 마인드맵을 이용해 필요한 기능과 로직 분석.



2.1 dataloader.Chatbot.ChatbotTextClassicification 클래스

Code	해설
`index_of_words = self.tokenizer.encode(datas[1])` `token_type_ids = [0] * len(index_of_words)` `attention_mask = [1] * len(index_of_words)`	질문 데이터의 길이를 구하고 그 길이만큼 token_type_ids변수와 attention_mask 변수에 넣음 이 것은 후에 input으로 들어감
`padding_length = max_seq_len - len(index_of_words)` `index_of_words += [0] * padding_length` `token_type_ids += [0] * padding_length` `attention_mask += [0] * padding_length`	Input 문장 데이터의 길이를 모두 동일하게 맞춰주기 위한 과정

2.2 model.koelectra.ElectraClassificationHead 클래스

Code	해설
`class ElectraClassificationHead(nn.Module):` `  """Head for sentence-level classification tasks."""` `  def __init__(self, config, num_labels):` `  super().__init__()` `  self.dense = nn.Linear(config.hidden_size, 4*config.hidden_size)` `  self.dropout = nn.Dropout(config.hidden_dropout_prob)` `  self.out_proj = nn.Linear(4*config.hidden_size, num_labels)` `  def forward(self, features, **kwargs):` `  x = features[:, 0, :]  # take <s> token (equiv. to [CLS])` `  x = self.dropout(x)` `  x = self.dense(x)` `  x = get_activation("gelu")(x)  # although BERT uses tanh here, it s` `  x = self.dropout(x)` `  x = self.out_proj(x)` `  return x`	순전파가 하나 돌아갈때 fine tune 구조. 2개의 dropout layer 계층을 줌으로서 overfitting을 방지하고자 함

2.3 model.koelectra.koElectraForSequenceClassification 클래스

Code	해설
`# 분류 레이블이 2개면 다중 분류 mean square err, 그 이상이면 crossentropy` `  if labels is not None:` `  if self.num_labels == 1:` `  #  We are doing regression` `  loss_fct = MSELoss()` `  loss = loss_fct(logits.view(-1), labels.view(-1))` `  else:` `# crossentropy는 데이터 불균형 문제에서 취약. 논문참조 focal loss` `  #loss_fct = CrossEntropyLoss()` `  #loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))` `  loss = focal_loss(logits.view(-1, self.num_labels), labels.view(-1),` `  outputs = (loss,) + outputs`	Loss 값 계산 과정 레이블이 2개면 MSE를 사용 그 이상이면 Focal loss 사용

```python
def koelectra_input(tokenizer, str, device = None, max_seq_len = 512):
  index_of_words = tokenizer.encode(str)
  attention_mask = [1] * len(index_of_words)

  # Padding Length
  padding_length = max_seq_len - len(index_of_words)
  # Zero Padding
  index_of_words += [0] * padding_length
  attention_mask += [0] * padding_length

  data = {\
    'input_ids': torch.tensor([index_of_words]).to(device),
    'attention_mask': torch.tensor([attention_mask]).to(device),
  }
  return data
```

Input 문장을 모델에 집어 넣기 위해 형식을 변경 해주는 함수

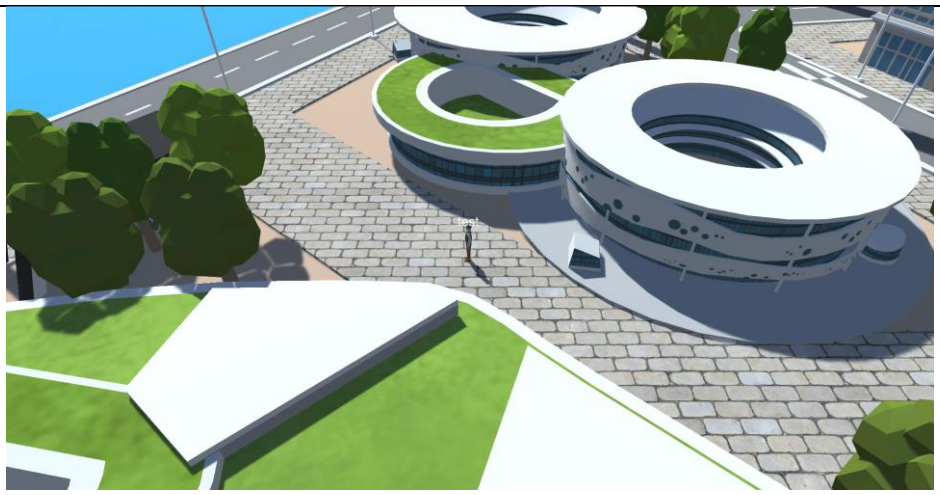
맡은 역할	1. 조장 2. pretrain 모델 코드 분석 3. fine tuning 설계 및 AI모델 제작 4. 챗봇 아이디어 제공 5. 요구사항 설명서 작성 6. 보고서 작성 7. 웹 프로그래밍 8. 데이터 전처리
프로젝트를 진행하면서 어려웠던 점	**1. 팀원 한명의 따라오는 속도가 다소 느렸음.** -> 능력의 문제라 어쩔수 없었기에 빠르게 능력 파악 후 충분히 할 있는 업무를 할당 **2. 팀원 한명이 졸업작품에 투자하는 시간이 매우 적었음.** -> 소통의 부재 또는 의지의 문제라고 판단하였음. 꾸준히 식사 자리와 삶을 물어보면서 소통 길을 열어두고 멘탈을 케어 하고자 하였음. 딱히 다른 이유가 없음에도 불구하고 5개월에 걸친 권유와 기다림과 믿음에도 변화가 없자 강력하게 경고 . **3. 팀원들의 기여도가 줄어들수록 나의 일이 너무 많아짐** -> 버티고 버텼음. 그리고 1년이 지난 지금 인성과 실력면에서 많이 성장하였음. **4. 데이터 불균형 문제** -> 레이블간의 데이터의 개수가 차이가 나서 pytorch라이브러리에서 제공하는 cross entropy 함수로만 계산하면 개수가 적은 레이블의 데이터는 지나치게 적게 훈련이 되어 정확도가 떨어지는 현상이 나타났음. cross entropy의 함수를 변형시켜서 만든 Focal Loss를 자체적으로 작성하여 개수가 적은 레이블에 대해서 가중치를 많이 줌으로서 해결함

깨달은 점	1. 혼자서는 절대 양질의 프로젝트를 완성시킬 수 없음 2. 프로젝트에서 가장 중요한 것은 협업 3. 사업 아이디어 내는 것이 정말 힘듦 4. 듣기 싫은 소리도 리더로서 해야 할 때가 있음 5. 발표자료는 측량 가능한 수치로 할 것 6. 보고서 쓸 때 단위 표기가 매우 중요함

프로젝트 – 유니티 메타버스 시스템 개발

개발 기간	2022.03.23 ~ 2022.05.08
개발 환경	Unity 2020.03.33 / VisualStudio 2019 / C#
개발 인원	3명
요 약	웹기반 메타버스 플랫폼 운영
기능	**1. 임의 위치 동영상 재생** **2. 임의 위치 유튜브 링크 재생** **3. 유니티 내 웹페이지 연동** **4. SNS 연동** **5. 방명록 구현** **6. 게시판 구현** **7. 채팅 구현 (텍스트, 영상)** **8. 아바타 기본 동작 (걷기, 손흔들기, V포즈, 손흔들기)** **9. 특정 아바타랑 나의 아바타랑 사진 찍기**

실행 화면



1. 개발 기획

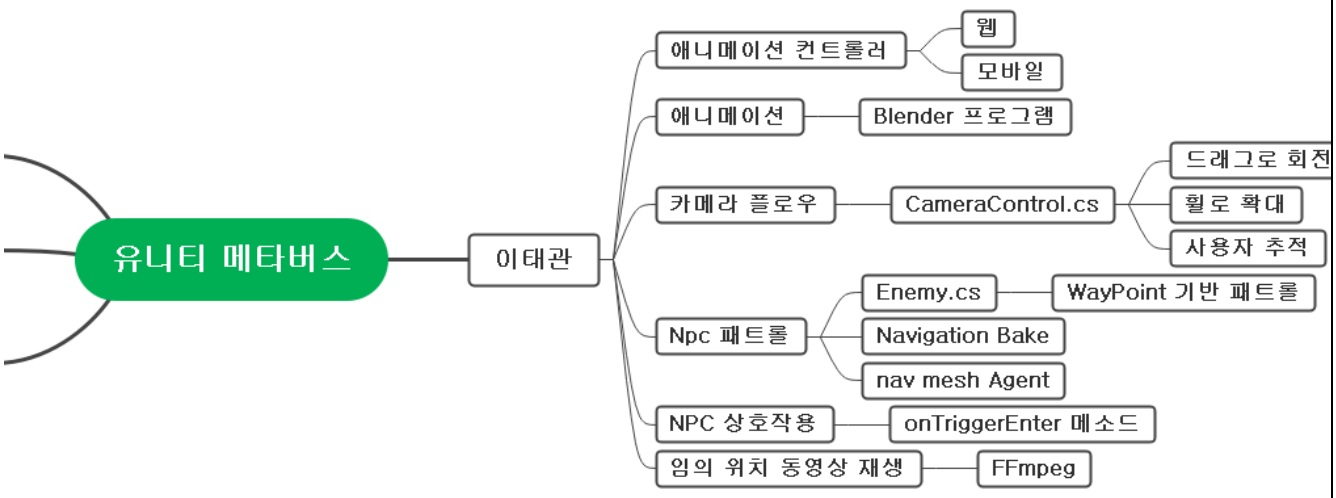
MileStone	03.23	03.30	04.06	04.13	04.20	04.27	05.04	05.09	05.16	05.23	05.30	06.06

기능구현												
동영상 재생												
유튜브 재생												
웹페이지 연동												
SNS 연동												
방명록												
게시판												
채팅												
아바타												
애니메이팅												
상호작용												
카메라 플로우												
Npc 패트롤												
운영테스트												
부하 테스트												
추가 수정 작업												

1.1 요구사항 : (전 말단이라 정확하게는 잘 모릅니다)

1. 디자인을 제외한 흔한 쿼터뷰 게임에서 볼 수 있는 기능들
2. 웹 기반 메타버스 플랫폼 운영
3. Unity 기반 메타버스 플랫폼 구축
4. 메타버스 내 아바타 및 물체 컨트롤 기능 구현
5. 전광판 내 영상 재생(스트리밍도 되게)
6. 채팅 기능 구현
7. 응원 게시판 구현

1.3 마인드맵을 이용해 필요한 기능과 로직 분석.



2.1 CameraControl 클래스 : 카메라의 줌, 휠업시의 확대 휠 다운시 축소 및 카메라 회전 역할

Code	해설
`` // by태관 카메라 줌. 휠업시 확대 휠 다운시 축소 void Zoom() { if (Input.GetAxis("Mouse ScrollWheel") != 0) { Distance += Input.GetAxis("Mouse ScrollWheel") * ZoomSpeed * -1; AxisVec = transform.forward * -1; if (Input.GetAxis("Mouse ScrollWheel") > 0) { if (Distance <= 9.8f) { Distance = 9.8f; } else { AxisVec += (Distance * -1); transform.position = MainCamera.position + AxisVec; } } else { if (Distance >= 10.1f) { Distance = 10.1f; } else { AxisVec += Distance; transform.position = MainCamera.position + AxisVec; } } } } ```	마우스 휠을 돌려서 위로 돌렸을때는 확대(Distance를 줄이기)하고 아래로 돌리면 축소(Distance를 늘리기) 하는 기능. 최대확대(최소 Distance)와 최소 축소(최대 Distance)는 각각 9.8, 10.1로 설정 해놓은 모습

`` // by태환 카메라 회전. void CameraRotation() { if (transform.rotation != TargetRotation) transform.rotation = Quaternion.Slerp(transform.rotation, TargetRotation, RotationSpeed * Time.deltaTime); if (Input.GetMouseButton(1)) { // 값을 축적. Gap.x += Input.GetAxis("Mouse Y") * RotationSpeed * -1; Gap.y += Input.GetAxis("Mouse X") * RotationSpeed; // 카메라 회전범위 제한. Gap.x = Mathf.Clamp(Gap.x, -5f, 85f); // 회전 값을 변수에 저장. TargetRotation = Quaternion.Euler(Gap); // 카메라벡터 객체에 Axis객체의 x,z회전 값을 제외한 y값만을 넘긴다. Quaternion q = TargetRotation; q.x = q.z = 0; CameraVector.transform.rotation = q; } } ```	쿼터뷰 게임의 회전을 구현 마우스 오른쪽 벝튼을 누르면 회전 쿼터니언 값을 받아와서 회전 후 카메라는 target(Player)를 정 중앙에 오게끔 다시 움직인다

2.2 Player 클래스 : Player의 움직임과 애니메이션컨트롤러의 연동

Code	해설
`` void Move() { moveVec = new Vector3(hAxis, 0, vAxis).normalized; if (isDodge) { moveVec = dodgeVec; } if (wDown) { transform.position += moveVec * speed * 0.3f * Time.deltaTime; } else { transform.position += moveVec * speed * Time.deltaTime; } anim.SetBool("isRun", moveVec != Vector3.zero); anim.SetBool("isWalk", wDown); } ```	캐릭터가 움직이는 것을 구현. wDown은 걷기임
`` void Jump() { if (jDown && !isJump && moveVec == Vector3.zero) { rigid.AddForce(Vector3.up * 15, ForceMode.Impulse); isJump = true; anim.SetBool("isJump", true); anim.SetTrigger("doJump"); } } ```	캐릭터가 점프 하는 것을 구현. 두번째 메소드는 캐릭터가 바닥과 닿았을때 애니메이션 컨트롤러에게 '점프 끝났다' 라고 알려줌

```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "Floor")
    {
        anim.SetBool("isJump", false);
        isJump = false;
    }
}
```

```
void Dodge()
{
    if (jDown && !isJump && moveVec != Vector3.zero && !isDodge)
    {
        dodgeVec = moveVec;
        speed *= 2;
        anim.SetTrigger("doDodge");
        isDodge = true;

        Invoke("DodgeOut", 0.4f);
    }
}

void DodgeOut()
{
    speed *= 0.5f;
    isDodge = false;
}
```

캐릭터가 대쉬를 했 했을 때 구현.

2.3 GameManager 클래스 : Agora asset 사용하여 유니티 내 영상통화 기능

Code	해설
`void onClickAdd() // by태관  참여자가 추가됐을때 화면 오른쪽 리스트에 추가 시키는 함수` `{` `GameObject temp = Instantiate(test, new Vector3(0, 0, 0), Quaternion.identity);` `temp.name = "user" + userCount.ToString();` `userCount++;` `temp?.GetComponent<Button>()?.onClick.AddListener(changeObject);` `GameObject faceContent = GameObject.Find("faceContent");` `temp.transform.SetParent(faceContent.transform);` `}`	1:1의 상황일때는 상대방의 화면을 큰 화면에 위치. 후에 추가로 사람이 추가되면 오른쪽 리스트 뷰에 작은 화면에 상대방들의 화면을 위치 temp객체에 user(숫자) 이름을 붙이고 faceContent 객체 밑으로 옮김

```
void changeObject() // by 태관 / 오른쪽 화면 리스트를 클릭시 실행되는 큰 화면과 교체하는 이벤트 리스너 함수
{

    GameObject clickObject = EventSystem.current.currentSelectedGameObject;
    if (clickObject.name != "RemoteView"){ // 큰화면 클릭시 반응 x
        GameObject panel = GameObject.Find("Panel");
        RectTransform clickObject_T = clickObject.GetComponent<RectTransform>();

        // 클릭한 화면을 화면밖으로 빼돌림
        clickObject.transform.SetParent(panel.transform);
        GameObject changeRemoteObject = GameObject.Find("RemoteView");
        RectTransform remoteView_T = changeRemoteObject.GetComponent<RectTransform>();


        clickObject_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Horizontal, remoteView_T.rect.width);
        clickObject_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical, remoteView_T.rect.height);
        clickObject.transform.position = changeRemoteObject.transform.position;

        changeRemoteObject.transform.SetParent(GameObject.Find("faceContent").transform);
        //remoteView_T.transform.position = new Vector3(0, 0, 0);
        remoteView_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Horizontal, 100);
        remoteView_T.SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical, 100);


        string temp = changeRemoteObject.name;
        changeRemoteObject.name = clickObject.name;
        clickObject.name = temp;
    }
}
```

오른쪽 리스트뷰 작은 화면을 클릭하면 그 작은 화면이 큰 화면으로 옮겨지고 큰 화면은 작은 화면으로 옮겨짐

2.4 Enemy 클래스 : 자동차, 사람, 세그웨이 패트롤 기능

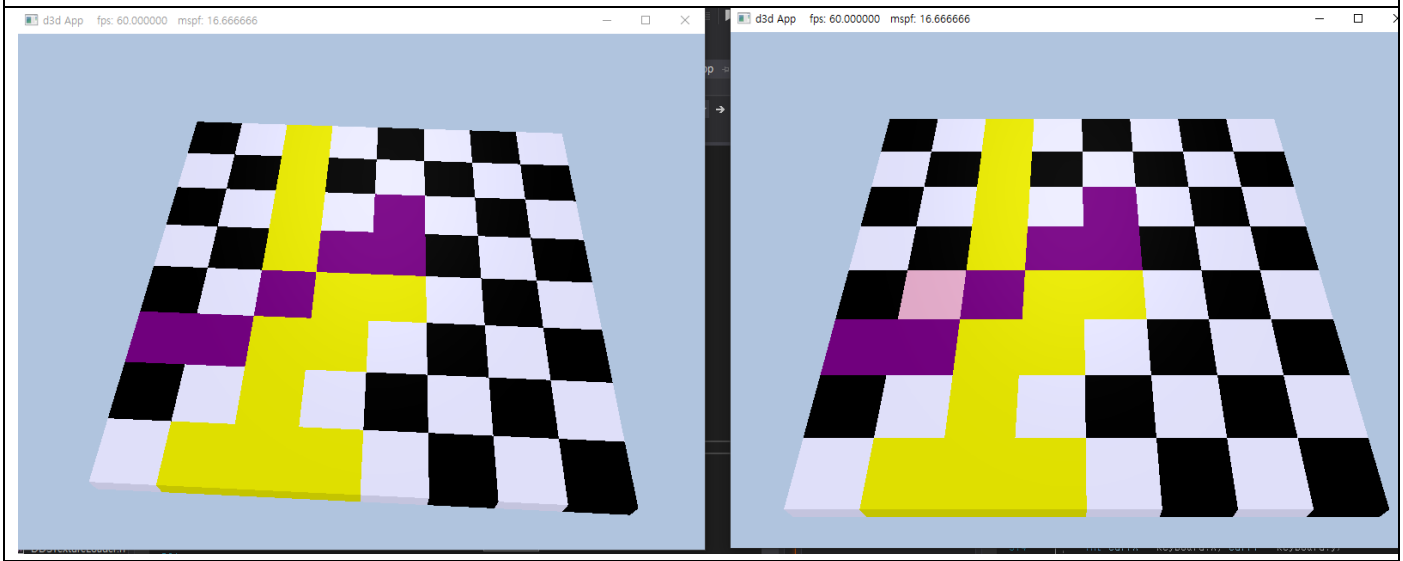
Code	해설
void Update() { //Debug.Log(target); if (Vector2.Distance(new Vector2(transform.position.x,transform.position.z), new Vector2(target.x, target.z)) < 1) { IterateWaypointIndex(); UpdateDestination(); } }	WayPoint와 가까워졌을 때 ItererateWayPointIndex()를 호출 하여 다음 waypoint를 찾고 UpdateDestination()을 호출 하여 다음 waypoint로 이동한다
public void UpdateDestination() { target =waypoints[waypointIndex].position; agent.SetDestination(target); }	Waypoint의 해당 인덱스로 이동한다
public void IterateWaypointIndex() { waypointIndex++; if (waypointIndex == waypoints.Length) { waypointIndex = 0; } }	Waypoint 배열의 다음 index를 가져온다

맡은 역할	1. 대량 트래픽을 한번에 감당할수 있는 서버 – 클라이언트 구조 개발 2. 애니메이션 컨트롤러 로직 설계 3. 간단한 애니메이션 제작 4. 쿼터뷰 게임에서 볼 수 있는 카메라 플로우 로직 설계 5. 자동차와 세그웨이, 지하철의 패트롤 설계 6. npc와 부딪혔을때의 상호작용 개발 7. 전광판에서 동영상 재생 8. Agora asset 을 사용한 영상채팅.
프로젝트를 진행하면서 어려웠던 점	**1. photon asset을 사용하면 한 채널에 최대 16명까지 수용 가능하지만 쉽게 구현이 가능하고 연구실 서버 컴퓨터를 활용해서 자체적으로 서버를 돌릴 수 만 있다면 100명이고 200명이고 수용이 가능하다고 교수님이 그렇게 해보라고 요구 하셔서 이틀 정도 집에도 못들어가고 연구실에서 하루종일 해커톤을 했음. 이틀째 밤새고 다음날 오전 9시 교수님 출근하시고 나서 그냥 하던거 폐기하고 photon으로 쉽게 쉽게 가자고 하셨음. 이틀의 노력이 물거품이 되어서 멘탈적으로 힘들었지만 그래도 어디가서 1억짜리 자체 서버 컴퓨터로 서버개발 처음부터 할 기회 없을텐데 귀한 경험했다고 자기암시를 하며 멘탈을 지켜내며 극복.** 2. 우리는 메타버스 플랫폼을 개발만 하면 되었음. 메타버스 장소와 사람 디자인과 애니메이션은 모두 다른 회사가 만들어서 전달해주기로 했는데 너무 늦게 4월 28일에 전달받았음. 때문에 개발일정이 배로 빠듯해졌었음 3. 미리 요구를 하지못한 간단한 애니메이션 (손흔들기, v사진포즈)들이 필요했었는데 그나마 할일이 적었던 내가 구현했음. 아예 생전 처음보는 툴을 갑작스럽게 공부해서 성과를 내야하는게 부담되었지만 진득하게 의자에 하루종일 앉아서 공부함으로서 극복 .
깨달은 점	**1. 개발 계획대로 착착 진행되는 경우는 거의 없다** **2. 연구개발실 분위기 축 처지게 안되게 만들고 집중할 수 있는 환경을 만드는 것도 리더의 능력이다. 그래서 팀장님이 존경스러웠다.** **3. 실력 향상엔 모르더라도 일단 일이 맡겨지면 코피터져가면서 데드라인 지키기위해 발버둥 치는 것이 최고다** **4. 작업을 의뢰한 사람은 본인이 무엇을 원하는지 잘 모른다** **5. 뭘 물어볼땐 바로 앞기수 대학원생에게 물어봐야한다** **6. 개발과 야근은 관련성이 많다. 미리 마음의 준비를 해야한다** **7. 라이브 서비스 할 때 가장 중요한 것은 안정성**

미니프로젝트 – 오셀로 게임 구현

개발 기간	2022.09.27 ~ 2022.10.03
개발 환경	C++ / Visual Studio 2019
개발 인원	1명
요 약	DirectX12를 활용한 오셀로 게임 제작
기능	**1. 키보드 조작을 통한 오셀로 게임 기능** **2. tcp 프로토콜 기반 1:1 기능**

실행 화면



1.1 요구사항

1. 메이플스토리의 미니게임 배틀리버스를 최대한 구현하는 것을 목표
2. TCP 소켓 프로그래밍으로 1:1 대결 구현
3. 최종은 사용자가 마우스로 클릭함으로서 조작할 수 있게 하는 것을 목표.
4. DirectX12 책의 예제를 적절히 활용

2.1 : ochello_host.sln 주요 코드

Code	해설

```
//스레드 함수. 나중에 class 안에 static으로 해볼것
void fn()
{
    int retval;

    // 윈속 초기화
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return;

    // socket()
    SOCKET listen_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_sock == INVALID_SOCKET) err_quit("socket()");

    // bind()
    SOCKADDR_IN serveraddr;
    ZeroMemory(&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons(SERVERPORT);
    retval = bind(listen_sock, (SOCKADDR*)&serveraddr, sizeof(serveraddr));
    if (retval == SOCKET_ERROR) err_quit("bind()");

    // listen()
    retval = listen(listen_sock, SOMAXCONN);
    if (retval == SOCKET_ERROR) err_quit("listen()");

    // 데이터 통신에 사용할 변수
    SOCKET client_sock;
    SOCKADDR_IN clientaddr;
    int addrlen;
    char buf[BUFSIZE + 1]; // 2로 줄여볼것
```

TCP 통신 기능을 수행하는 스레드 함수

```
while (1) {
    // accept()
    addrlen = sizeof(clientaddr);
    client_sock = accept(listen_sock, (SOCKADDR*)&clientaddr, &addrlen);
    if (client_sock == INVALID_SOCKET) {
        err_display("accept()");
        break;
    }

    // 접속한 클라이언트 정보 출력
    // 클라이언트와 데이터 통신
    while (1) {
        // 데이터 받기
        retval = recvn(client_sock, buf, 2, 0);
        if (retval == SOCKET_ERROR) {
            err_display("recv()");
            break;
        }
        else if (retval == 0)
            break;

        // 받은 데이터 출력
        theApp->receivedPoint(buf[0], buf[1]);
        theApp->checkChangeBlock(buf[0], buf[1]);
        // 대기했다가
        while (!theApp->received) // 비효율적. 다른 방법 생각해볼것
        {

        }
        theApp->received = false;
        // 데이터 보내기
        buf[0] = theApp->getX() + '0';
        buf[1] = theApp->getY() + '0';
        theApp->checkChangeBlock(buf[0], buf[1]);

        retval = send(client_sock, buf, retval, 0);
        if (retval == SOCKET_ERROR) {
            err_display("send()");
            break;
            break;
        }
    }

    // closesocket()
    closesocket(client_sock);
}

// closesocket()
closesocket(listen_sock);

// 윈속 종료
WSACleanup();
return;
```

프레임 n에서

```cpp
void LitColumnsApp::Update(const GameTimer& gt)
{
    OnKeyboardInput(gt);
    UpdateCamera(gt);

    // 순환적으로 자원 프레임의 다음 원소에 접근한다
    mCurrFrameResourceIndex = (mCurrFrameResourceIndex + 1) % gNumFrameResources;
    mCurrFrameResource = mFrameResources[mCurrFrameResourceIndex].get();

    /*
    GPU가 현재 프레임 자원의 명령들을 다 처리했는지 확인.
    아직 다 처리하지 않았으면 GPU가 이 울타리 지점까지의 명령들을 처리할때까지 기다린다
    */
    if (mCurrFrameResource->Fence != 0 && mFence->GetCompletedValue() < mCurrFrameResource
    {
        HANDLE eventHandle = CreateEventEx(nullptr, false, false, EVENT_ALL_ACCESS);
        ThrowIfFailed(mFence->SetEventOnCompletion(mCurrFrameResource->Fence, eventHandle)
        WaitForSingleObject(eventHandle, INFINITE);
        CloseHandle(eventHandle);
    }

    AnimateMaterials(gt);
    UpdateObjectCBs(gt);
    UpdateMaterialCBs(gt);
    UpdateMainPassCB(gt);
}
```

CPU가 하는 일들.
순환배열을이용하여CPU와 GPU의 균형을 맞춰준다.

```cpp
void LitColumnsApp::Draw(const GameTimer& gt)
{
    auto cmdListAlloc = mCurrFrameResource->CmdListAlloc;

    // 명령 기록에 관련된 메모리의 재활용을 위해 명령 할당자를 재설정.
        // 재설정은 gpu가 관련 명령 목록들을 모두 처리한 후에 일어남
    ThrowIfFailed(cmdListAlloc->Reset());

    //명령목록을 executeCommandList를 통해서 명령 대기열에 추가했다면 명령목록재설정 가능
    // 명령 목록을 재설정하면 메모리가 재활용
    ThrowIfFailed(mCommandList->Reset(cmdListAlloc.Get(), mOpaquePSO.Get()));

    mCommandList->RSSetViewports(1, &mScreenViewport);
    mCommandList->RSSetScissorRects(1, &mScissorRect);

    //자원 용도에 관련된 상태 전이를 direct3d에 통지
    mCommandList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::Transition(CurrentBackBuffer(),
        D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    //후면 버퍼와 깊이 버퍼를 지운다
    mCommandList->ClearRenderTargetView(CurrentBackBufferView(), Colors::LightSteelBlue, 0, nul
    mCommandList->ClearDepthStencilView(DepthStencilView(), D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEA

    // 렌더링 결과가 기록될 렌더대상 버퍼들을 지정
    mCommandList->OMSetRenderTargets(1, &CurrentBackBufferView(), true, &DepthStencilView());
```

update() 함수와 연계해서 프레임 자원들을 CPU와 GPU를 모두 효율적이게 사용한다.
GPU에게 계속해서 일거리를 줌으로서 효율적이게 사용할 수 있다

```
    mCommandList->SetGraphicsRootSignature(mRootSignature.Get());

    auto passCB = mCurrFrameResource->PassCB->Resource();
    mCommandList->SetGraphicsRootConstantBufferView(2, passCB->GetGPUVirtualAddress());

    DrawRenderItems(mCommandList.Get(), mOpaqueRitems);

    // Indicate a state transition on the resource usage.
    mCommandList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::Transition(CurrentBackBuffer(),
        D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(mCommandList->Close());

    // 명령큐 실행
    ID3D12CommandList* cmdsLists[] = { mCommandList.Get() };
    mCommandQueue->ExecuteCommandLists(_countof(cmdsLists), cmdsLists);

    // 스왑체인 스왑
    ThrowIfFailed(mSwapChain->Present(0, 0));
    mCurrBackBuffer = (mCurrBackBuffer + 1) % SwapChainBufferCount;

    // 현재 fence 지점까지의 명령들을 표시하도록 fence 멤버를 전진
    mCurrFrameResource->Fence = ++mCurrentFence;
    /*
    새 fence 지점을 설정하는 signal(명령)을 명령 대기열에 추가.
    지금 우리는 GPU 시간선 (timeline) 위에 있으므로 새 fence 지점은
    GPU가 이 signal() 명령 이전까지의 모든 명령을 처리하기 전까지는 설정되지 않음
    */
    mCommandQueue->Signal(mFence.Get(), mCurrentFence);
}
```

```
struct RenderItem
{
    RenderItem() = default;

    // 뭐 그 늘 보던 세계공간 기준으로 물체의 국소공간 서술하는 세계행렬
    // 이 행렬은 세계 공간 안에서 물체의 위치와 방향 크기를 결정
    XMFLOAT4X4 World = MathHelper::Identity4x4();

    /*
    물체의 자료가 변해서 상수 버퍼를 갱신해야하는 지의 여부를 뜻하는 '더러움' 플래그
    FrameResource마다 물체의 cbuffer가 있으므로 FrameResource마다 갱신을 적용해야함
    따라서 물체의 자료를 수정할땐 반드시 NumFramesDirty = gNumFrameResources로 설정해야함
    그래야 각각의 프레임 자원이 갱신된다
    */
    int NumFramesDirty = gNumFrameResources;

    // 이 렌더항목의 물체 상수에 해당하는 GPU 상수 버퍼의 인덱스
    UINT ObjCBIndex = -1;

    MeshGeometry* Geo = nullptr;

    // Primitive topology.
    D3D12_PRIMITIVE_TOPOLOGY PrimitiveType = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;

    // DrawIndexedInstances 매개변수들
    UINT IndexCount = 0;
    UINT StartIndexLocation = 0;
    int BaseVertexLocation = 0;
};
```

렌더링 하는 물체의 정보들을 담은 구조체.
물체의 세계 행렬과 정점버퍼 정보, 인덱스버퍼 정보 위상 기본정보 등을 담는다

```cpp
//상수 버퍼 업데이트
void LitColumnsApp::UpdateObjectCBs(const GameTimer& gt)
{
    auto currObjectCB = mCurrFrameResource->ObjectCB.get();
    for (auto& e : mAllRitems)
    {
        /*
        상수들이 바뀌었을때만 cubuffer 자료 갱신
        이러한 갱신은 저기 매개변수 오듯 프레임 자원마다 갱신 해야함 당연한거
        */
        if (e->NumFramesDirty > 0)
        {
            XMMATRIX world = XMLoadFloat4x4(&e->World);
            XMMATRIX texTransform = XMLoadFloat4x4(&e->TexTransform);

            ObjectConstants objConstants;
            XMStoreFloat4x4(&objConstants.World, XMMatrixTranspose(world));
            XMStoreFloat4x4(&objConstants.TexTransform, XMMatrixTranspose(texTransform));

            currObjectCB->CopyData(e->ObjCBIndex, objConstants);

            // 다음 프레임 자원으로 넘어감
            e->NumFramesDirty--;
        }
    }
}
```

update 함수가 프레임당 한번씩 호출. 상수 버퍼들이 만약 수정해야한다면 수정 기능을 수행. 아니면 그냥 넘어감

```cpp
/*
    passCB == FrameResource.h에서 변하지 않는 상수 자료를 저장.
    ex) 시점 위치, 시야 행렬, 투영행렬 등등..
*/
void LitColumnsApp::UpdateMainPassCB(const GameTimer& gt)
{
    XMMATRIX view = XMLoadFloat4x4(&mView);
    XMMATRIX proj = XMLoadFloat4x4(&mProj);

    XMMATRIX viewProj = XMMatrixMultiply(view, proj);
    XMMATRIX invView = XMMatrixInverse(&XMMatrixDeterminant(view), view);
    XMMATRIX invProj = XMMatrixInverse(&XMMatrixDeterminant(proj), proj);
    XMMATRIX invViewProj = XMMatrixInverse(&XMMatrixDeterminant(viewProj), viewProj);

    XMStoreFloat4x4(&mMainPassCB.View, XMMatrixTranspose(view));
    XMStoreFloat4x4(&mMainPassCB.InvView, XMMatrixTranspose(invView));
    XMStoreFloat4x4(&mMainPassCB.Proj, XMMatrixTranspose(proj));
    XMStoreFloat4x4(&mMainPassCB.InvProj, XMMatrixTranspose(invProj));
    XMStoreFloat4x4(&mMainPassCB.ViewProj, XMMatrixTranspose(viewProj));
    XMStoreFloat4x4(&mMainPassCB.InvViewProj, XMMatrixTranspose(invViewProj));
    mMainPassCB.EyePosW = mEyePos;
    mMainPassCB.RenderTargetSize = XMFLOAT2((float)mClientWidth, (float)mClientHeight);
    mMainPassCB.InvRenderTargetSize = XMFLOAT2(1.0f / mClientWidth, 1.0f / mClientHeight);
    mMainPassCB.NearZ = 1.0f;
    mMainPassCB.FarZ = 1000.0f;
    mMainPassCB.TotalTime = gt.TotalTime();
    mMainPassCB.DeltaTime = gt.DeltaTime();
    mMainPassCB.AmbientLight = { 0.25f, 0.25f, 0.35f, 1.0f };
    mMainPassCB.Lights[0].Direction = { 0.57735f, -0.57735f, 0.57735f };
    mMainPassCB.Lights[0].Strength = { 0.6f, 0.6f, 0.6f };
    mMainPassCB.Lights[1].Direction = { -0.57735f, -0.57735f, 0.57735f };
    mMainPassCB.Lights[1].Strength = { 0.3f, 0.3f, 0.3f };
    mMainPassCB.Lights[2].Direction = { 0.0f, -0.707f, -0.707f };
    mMainPassCB.Lights[2].Strength = { 0.15f, 0.15f, 0.15f };

    auto currPassCB = mCurrFrameResource->PassCB.get();
    currPassCB->CopyData(0, mMainPassCB);
}
```

update() 함수에서 프레임당 한번 호출 passCB 별 상수 버퍼 갱신. 상수 버퍼들이 만약 수정해야한다면 수정 기능을 수행. 아니면 그냥 넘어감

```cpp
void LitColumnsApp::BuildRootSignature()
{
    /*
    일반적으로 셰이더 프로그램은 특정 자원 (상수 버퍼, 텍스처, 표본 추출기 등)이 입력된다고 기대함
    루트 서명은 셰이더 프로그램이 기대하는 자원들을 정의한다
    셰이더 프로그램은 본질적으로 하나의 함수이고 셰이더에 입력되는 자원들은 함수의 매개변수에 해당한다
    따라서 루트서명은 곧 함수 수명을 정의하는 수단이라고 볼 수 있음
    */

    // 루트 매개변수는 서술자 테이블이거나 루트 서술자 또는 루트 상수.
    CD3DX12_ROOT_PARAMETER slotRootParameter[3];

    // cbv 하나를 담는 서술자 테이블을 생성한다
    slotRootParameter[0].InitAsConstantBufferView(0);
    slotRootParameter[1].InitAsConstantBufferView(1);
    slotRootParameter[2].InitAsConstantBufferView(2);

    //루트 서명은 루트 매개변수들의 배열.
    CD3DX12_ROOT_SIGNATURE_DESC rootSigDesc(3, slotRootParameter, 0, nullptr, D3D12_ROOT_SIGNATURE_FLAG_AL

    //상수 버퍼 하나로 구성된 서술자 구간을 가리키는 슬롯하나로 이루어진 루트 서명 생성
    ComPtr<ID3DBlob> serializedRootSig = nullptr;
    ComPtr<ID3DBlob> errorBlob = nullptr;
    HRESULT hr = D3D12SerializeRootSignature(&rootSigDesc, D3D_ROOT_SIGNATURE_VERSION_1,
        serializedRootSig.GetAddressOf(), errorBlob.GetAddressOf());

    if (errorBlob != nullptr)
    {
        ::OutputDebugStringA((char*)errorBlob->GetBufferPointer());
    }
    ThrowIfFailed(hr);

    ThrowIfFailed(md3dDevice->CreateRootSignature(
        0,
        serializedRootSig->GetBufferPointer(),
        serializedRootSig->GetBufferSize(),
        IID_PPV_ARGS(mRootSignature.GetAddressOf())));
}
```

루트 서명 설정.

셰이더 프로그램이 기대하는 자원들을 정의

```cpp
// 박스를 조합해서 체스판을 만듦
void LitColumnsApp::BuildRenderItems()
{
    UINT ObjCBind = 0;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            auto boxRitem = std::make_unique<RenderItem>();
            // 세계 행렬 정의.
            XMStoreFloat4x4(&boxRitem->World, XMMatrixScaling(2.0f, 2.0f, 2.0f) * XMMatrixTranslation(-10.0f +(
            // 크기 행렬 정의
            XMStoreFloat4x4(&boxRitem->TexTransform, XMMatrixScaling(1.0f, 1.0f, 1.0f));
            boxRitem->ObjCBIndex = ObjCBind;
            ObjCBind++;

            //흰색 검정, 가운데 4타일 대로 체스판 색을 초기화
            if (initColor[i][j]==1)
            {
                boxRitem->Mat = mMaterials["stone1"].get();
            }
            else if (initColor[i][j] == 0)
            {
                boxRitem->Mat = mMaterials["stone0"].get();
            }
            else if (initColor[i][j] == -1)
            {
                boxRitem->Mat = mMaterials["user1"].get();
            }
            else if (initColor[i][j] == -2)
            {
                boxRitem->Mat = mMaterials["user2"].get();
            }
```

박스 Render와 material을 조합하여 8x8의 체스판을 만들고 색깔을 초기화한다

```
        else if (initColor[i][j] == -2)
        {
            boxRitem->Mat = mMaterials["user2"].get();
        }
        boxRitem->Geo = mGeometries["shapeGeo"].get();
        boxRitem->PrimitiveType = D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
        boxRitem->IndexCount = boxRitem->Geo->DrawArgs["box"].IndexCount;
        boxRitem->StartIndexLocation = boxRitem->Geo->DrawArgs["box"].StartIndexLocation;
        boxRitem->BaseVertexLocation = boxRitem->Geo->DrawArgs["box"].BaseVertexLocation;
        mAllRitems.push_back(std::move(boxRitem));

    }
}
for (auto& e : mAllRitems)
{
    mOpaqueRitems.push_back(e.get());
}
```

```
// 사용자 입력. 돌을 놓거나 놓을 위치 이동
void LitColumnsApp::OnKeyboardDown(WPARAM btnState)
{
    TCHAR pressedChar = static_cast<TCHAR>(btnState);
    // 사용자가 space를 눌렀을때 그게 가능한지 판단 후 돌을 놓는다
    if (btnState == VK_SPACE && haveControl && initColor[keyboard.x][keyboard.y] >= 0)
    {
        mOpaqueRitems[keyboard.x + keyboard.y + 8]->Mat = mMaterials["user1"].get();
        initColor[keyboard.x][keyboard.y] = -1;
        received = true;
        haveControl = false;
        return;
    }

    int currMat = initColor[keyboard.x][keyboard.y];
    int currX = keyboard.x, currY = keyboard.y;

    // 키보드 커서 이동
    if (pressedChar == 'w')
    {
        keyboard.x++;
    }
    else if (pressedChar == 'd')
    {
        keyboard.y++;
    }
    else if (pressedChar == 'a')
    {
        keyboard.y--;
    }
    else if (pressedChar == 's')
    {
        keyboard.x--;
    }
```

```
    if (isRange(keyboard.x, keyboard.y))
    {
        mOpaqueRitems[keyboard.x + keyboard.y + 8]->Mat = mMaterials["stone2"].get();
        if (currMat == 1)
        {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["stone1"].get();
        }
        else if (currMat == 0) {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["stone0"].get();
        }
        else if (currMat == -1) {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["user1"].get();
        }
        else if (currMat == -2) {
            mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["user2"].get();
        }
    }
    else
    {
        mOpaqueRitems[currX + currY + 8]->Mat = mMaterials["stone2"].get();
        keyboard.x = currX;
        keyboard.y = currY;
    }

}
```

사용자 입력 처리. space는 자신의 돌을 체스판에 놓음

wasd는 위, 아래 이동

```
//시간복잡도 주의. 돌이 놓여졌을때 작동. 변경되어야할 색깔들을 변경함
void LitColumnsApp::checkChangeBlock(char x, char y)
{
    int cx = x - '0';
    int cy = y - '0';

    int xp = cx, yp = cy, xm = cx, ym = cy;

    Material* colorMat;
    int color = initColor[cx][cy];

    if (color == -1) { ... }
    else { ... }

    for (int i = 1; i < 8; i++)
    {

        if (++xp < 8)
        {
            // 한번이라도 검정, 흰타일 나오면 끝
            if (initColor[xp][cy] < 0)
            {

                // 검색하는 동안에 흰타일 검정타일 안나왔고, 범위 내 일인데 원래 내 색이 나왔을 때
                if (initColor[xp][cy] == color)
                {

                    // cx+1부터 xp-1 까지 쭉 color의 색으로로 변경
                    for (int j = cx + 1; j < xp; j++)
                    {
                        mOpaqueRitems[j + cy * 8]->Mat = colorMat;
                        initColor[j][cy] = color;
                    }
                    xp += 100;
                }
```

상대방에게 상대방이 어디뒀는지 통신을 받거나 내가 돌을 뒀을때 호출. 색깔이 변경되어야할 위치의 블록의 색을 변경함 일직선, 대각선 전부 탐색

2.2 Ochello_Client2.sln 주요 코드 : host.sln과 중복되는 것은 제외

Code	해설

```
void fn()
{
    int retval;

    // 윈속 초기화
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return ;

    // socket()
    SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == INVALID_SOCKET) err_quit("socket()");

    // connect()
    SOCKADDR_IN serveraddr;
    ZeroMemory(&serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = inet_addr(SERVERIP);
    serveraddr.sin_port = htons(SERVERPORT);
    retval = connect(sock, (SOCKADDR*)&serveraddr, sizeof(serveraddr));
    if (retval == SOCKET_ERROR) err_quit("connect()");

    // 데이터 통신에 사용할 변수
    char buf[BUFSIZE];
    int len;

    // 서버와 데이터 통신
    while (1) {
        // 데이터 입력
        while (!theApp->sended) //비효율적이므로 다른거 생각해보기
        {

        }
        buf[0] = theApp->getX() + '0';
```

host.sln과 tcp 통신을 하는 스레드 함수

```
        buf[0] = theApp->getX() + '0';
        buf[1] = theApp->getY() + '0';

        theApp->checkChangeBlock(buf[0], buf[1]);
        // 데이터 보내기
        retval = send(sock, buf, 2, 0);
        if (retval == SOCKET_ERROR) {
            err_display("send()");
            break;
        }

        // 데이터 받기
        retval = recvn(sock, buf, 2, 0);

        theApp->receivedPoint(buf[0], buf[1]);
        theApp->checkChangeBlock(buf[0], buf[1]);
        if (retval == SOCKET_ERROR) {
            err_display("recv()");
            break;
        }
        else if (retval == 0)
            break;

        // 받은 데이터 처리
    }

    // closesocket()
    closesocket(sock);

    // 윈속 종료
    WSACleanup();
    return ;
}
```

프로젝트를 진행하면서 어려웠던 점	1. 언리얼과 유니티로 편하게 작업하다가 아예 맨땅에 렌더링 하려고 하니까 어색해서 적응기간이 생각보다 오려 걸렸음 2. 책에서도 안나오고 검색해도 잘 안나오는 경험적인 실력이 많이 부족해서 비효율적인 구현을 많이 하게 되었음 3. 비동기 스레드 실전 프로그래밍 실력이 부족함을 깨달았음. 4. CommandList를 멀티스레드로 여러 개를 생성하여 멀티스레드 렌더링이 필수인데 몇 개가 가장 최적화 된 개수인지 알기가 쉽지 않았음. (현재 진행, 아직 해결못함) 5. Vertex Buffer View와 Index Buffer View의 Alignment가 64kb라서 무조건 64kb의 용량으로 생성이된다.(상수버퍼도 256kb단위) 이때 index buffer의 크기가 4kb라면 60kb의 용량을 낭비하게 된다. 이 현상으로 인하여 GPU 메모리 낭비가 심한 상태이다. 이것을 해결하기 위해 여러곳에 서칭을 해봤지만 서칭이 쉽지 않았고, 힌트를 얻기 위해 현재 DirectX12 게임을 서비스 중인 문명6 개발팀에게 문의를 해보았으나 긍정적인 답변을 받지 못하고 해결을 못하고 있음. 6. 정점 셰이딩 단계에서 절두체 생성 부분 이해하는데 시간이 오래 걸렸음
깨달은 점	**1. 난 모르는게 많다. 항상 정진하자** **2. 상용엔진 사용법도 물론 중요하지만 그래픽스 기본과 기초cs 지식이 상당히 중요하다.**