Unity에서의 ChatGPT
연동 및 내부 실행 과정

학번: 2019305061

학과: 컴퓨터공학과

이름: 임주형

발단:

2024 졸업작품 전시회 중 작품 평가 발표를 하던 중, chatGPT와 Unity의 연동 및 내부적인 실행 과정에 대해 파악하여 레포트로 제출하라는 과제로 인해 이번 레포트를 작성하게 되었다.

주요 코드 분석:

1. **OpenAIApi.cs**: API와의 상호작용을 관리하는 핵심 클래스

① OpenAI API와의 HTTP 요청을 처리하고 응답을 관리하는 기능을 제공

- DispatchRequest 메서드
  - OpenAI API에 요청을 전송하고, 응답을 처리하는 핵심 함수
  - 비동기로 동작하여 Unity의 메인 쓰레드가 블록되지 않도록 설계됨

```
private async Task<T> DispatchRequest<T>(string path, string method, byte[] payload = null) where T: IResponse
{
    T data;

    // UnityWebRequest를 사용해 HTTP 요청을 생성
    using (var request = UnityWebRequest.Put(path, payload))
    {
        request.method = method;    // 요청의 HTTP 메서드 설정 (POST, GET 등)
        request.SetHeaders(Configuration, ContentType.ApplicationJson); // API 키 및 콘텐츠 타입을 포함한 헤더 설정

        var asyncOperation = request.SendWebRequest();  // 요청 전송 및 비동기 응답 처리

        while (!asyncOperation.isDone) await Task.Yield();

        data = JsonConvert.DeserializeObject<T>(request.downloadHandler.text, jsonSerializerSettings);
    }

    if (data?.Error != null)
    {
        ApiError error = data.Error;
        Debug.LogError($"Error Message: {error.Message}\nError Type: {error.Type}\n");
    }

    if (data?.Warning != null)
    {
        Debug.LogWarning(data.Warning);
    }

    return data;
}
```

- CreateChatCompletion 메서드
  - ChatGPT 모델과 대화를 생성
  - 사용자의 입력 메시지를 기반으로 응답 텍스트를 반환

```
public async Task<CreateChatCompletionResponse> CreateChatCompletion(CreateChatCompletionRequest request)
{
    var path = $"{BASE_PATH}/chat/completions"; // ChatGPT 엔드포인트 경로
    var payload = CreatePayload(request);       // 요청 데이터를 JSON으로 직렬화

    // API에 요청을 보내고 응답 데이터를 반환
    return await DispatchRequest<CreateChatCompletionResponse>(path, UnityWebRequest.kHttpVerbPOST, payload);
}
```

2. Configuration.cs: 사용자 인증 관리

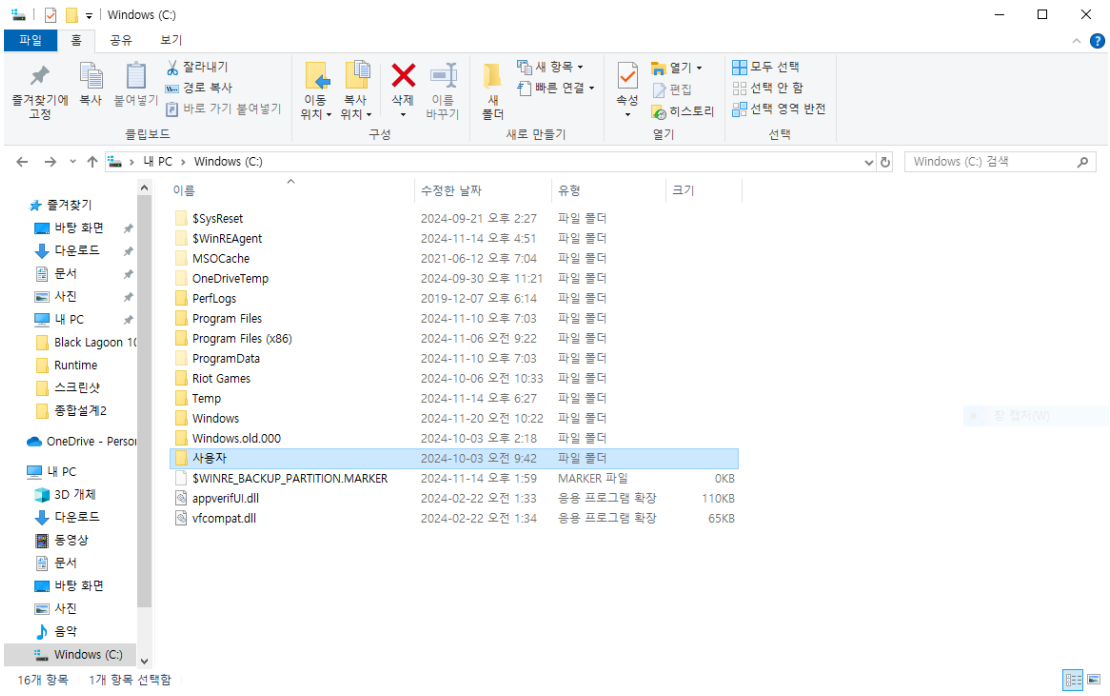
① API 키와 조직 정보를 저장하고 관리

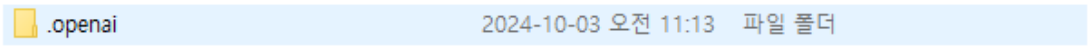
② API 키가 누락된 경우, 로컬의 **auth.json**에서 정보를 호출
(이 방법을 사용하여 게임을 제작함)

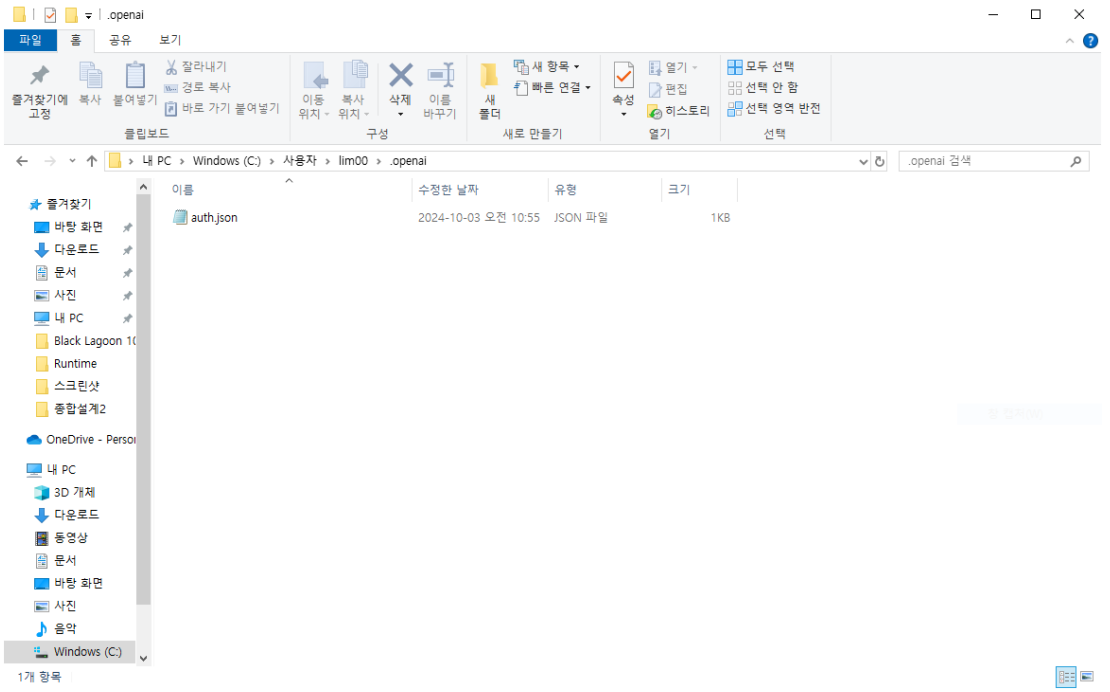
```
public Configuration(string apiKey = null, string organization = null)
{
    if (apiKey == null)// API 키가 제공되지 않은 경우
    {
        var userPath = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);    // 사용자 홈 디렉토리 경로
        var authPath = $"{userPath}/.openai/auth.json"; // 인증 파일 경로

        if (File.Exists(authPath))  // 인증 파일이 존재하면
        {
            var json = File.ReadAllText(authPath);  // 파일 내용을 읽어옴
            Auth = JsonConvert.DeserializeObject<Auth>(json, jsonSerializerSettings);   // JSON 데이터를 Auth 객체로 변환
        }
        else
        {
            Debug.LogError("API Key is null and auth.json does not exist. " +
                "Please check https://github.com/srcnalt/OpenAI-Unity#saving-your-credentials");
        }
    }
    else    // 이 방식을 사용하여 Unity에 인증
    {
        Auth = new Auth()
        {
            ApiKey = apiKey,            // 전달받은 API 키를 설정
            Organization = organization // 전달받은 조직 ID를 설정
        };
    }
}
```

③ 인증 방법







C드라이브 > 사용자(user) > 사용중인 계정 폴더 > .openai 폴더 생성 > auth.json 생성 > api_key 및 organization 입력

3. DataTypes.cs: 데이터 구조 정의

① 요청(Request)과 응답(Response)을 처리

- CreateChatCompletionRequest 클래스
  - ChatGPT와 대화를 생성하기 위한 요청 데이터 구조

```
public sealed class CreateChatCompletionRequest
{
    public string Model { get; set; }                   // 사용할 모델 이름 (예: "gpt-3.5-turbo")
    public List<ChatMessage> Messages { get; set; }     // 대화 메시지의 목록
    public float? Temperature { get; set; } = 1;        // 응답의 창의성 조정 (기본값: 1)
    public int N { get; set; } = 1;                     // 생성할 응답의 개수 (기본값: 1).
    public bool Stream { get; set; } = false;           // 스트리밍 모드 활성화 여부.
    public string Stop { get; set; }                    // 응답 생성을 멈추는 조건.
    public int? MaxTokens { get; set; }                 // 생성할 응답의 최대 토큰 수.
    public float? PresencePenalty { get; set; } = 0;    // 주제의 다양성을 증가시키는 값.
    public float? FrequencyPenalty { get; set; } = 0;   // 동일 단어 반복을 억제하는 값.
    public Dictionary<string, string> LogitBias { get; set; }   // 특정 단어의 우선순위 조정.
    public string User { get; set; }                    // 요청을 보낸 사용자 식별자.
    public string SystemFingerprint { get; set; }
    public ResponseFormat ResponseFormat { get; set; }
}
```

  - 사용 예시

```
var request = new CreateChatCompletionRequest
{
    Model = "gpt-3.5-turbo",
    Messages = new List<ChatMessage>
    {
        new ChatMessage { Role = "user", Content = "Hello, ChatGPT!" }
    },
    Temperature = 0.7f,
    MaxTokens = 100
};
```

- ChatMessage 구조체
  - ChatGPT 대화에서 개별 메시지를 나타내는 구조
  - **Role -> user:**
    사용자가 ChatGPT에게 질문하거나 요청하는 메시지
  - **Role -> assistant:**
    OpenAI 모델이 사용자에게 응답으로 생성한 메시지
  - **Role -> system:**
    모델의 초기 지침 또는 행동 방식을 설정하는 메시지

```
public struct ChatMessage
{
    public string Role { get; set; }        // 메시지의 역할 ("user", "assistant", "system")
    public string Content { get; set; }     // 메시지 내용
}
```

- ChatMessage 구조체를 통한 NPC(Non Player Character) 학습과정

▼ 게임 시작과 함께 해당 내용을 json 파일로 저장

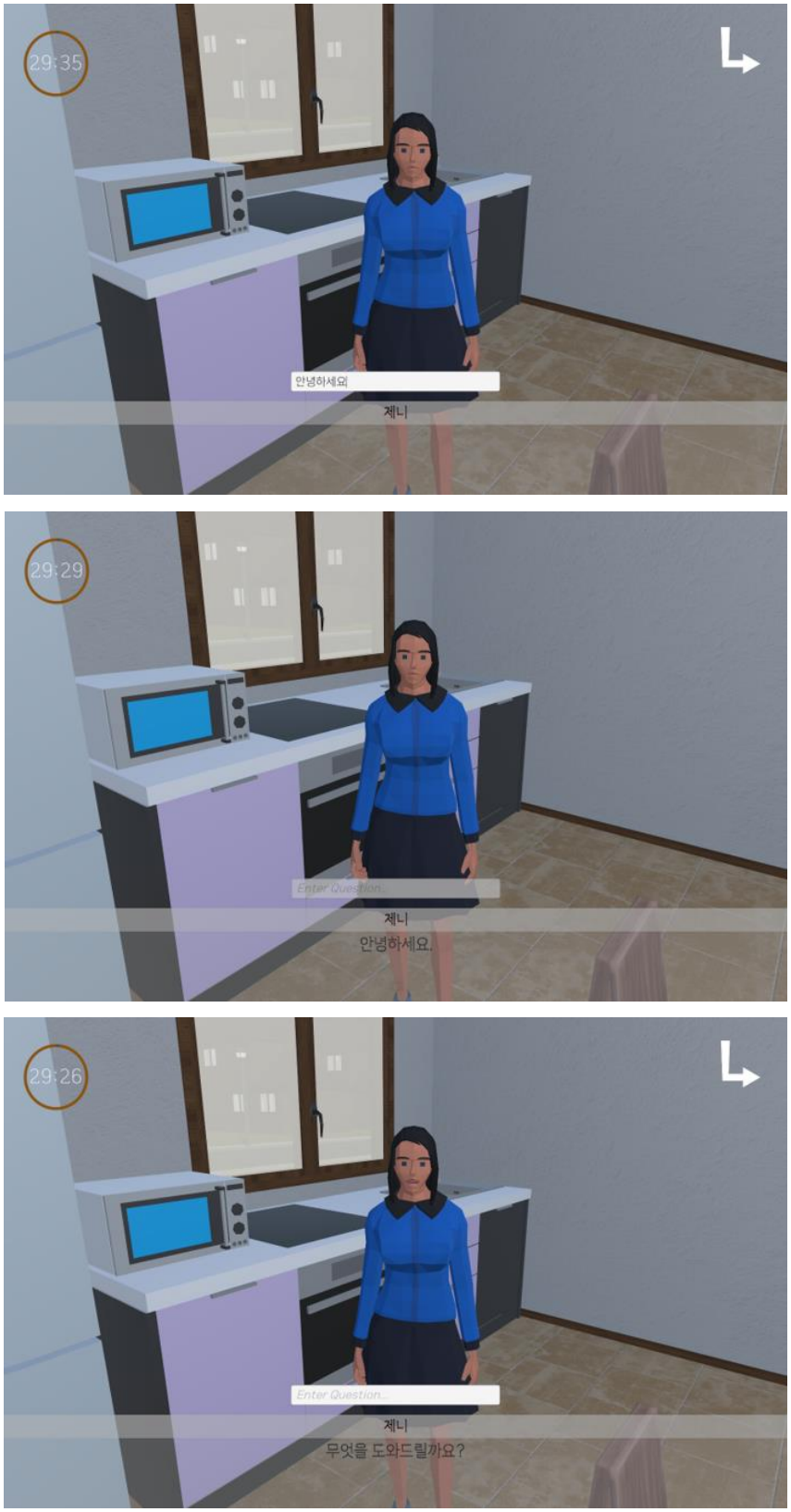
```
private void SaveNPCRoleJson()
{
    List<NPCRoleInfo> npcRoleInfo = new List<NPCRoleInfo>
    {
        new NPCRoleInfo {
            role = "Nason, a male lawyer.",

            instructions =
            "Instructions : " +
            "1. Always refer to NPC names in Korean (e.g., 네이슨, 앨런, 제니, 미나)." +
            "2. Speak in the tone and style that matches your character's personality and role." +
            "For example, as 네이슨, respond with a professional and composed tone appropriate for a lawyer." +
            "3. Be aware that when the player finds evidence, you will receive information about that evidence, " +
            "which may affect your responses." +
            "4. Remember, you are not investigating the incident yourself;" +
            "instead, you are being questioned by the player, who is the investigator in this situation.",

            background =
            "background : " +
            " - 앨런, CEO of a pharmaceutical company, hosted a party on May 7th. " +
            "He invited three friends from university: 네이슨, 제니, and 미나. " +
            "Although they became distant after graduation, 앨런 reunited them at his house.\n" +
            "- The party began at 8 PM and continued into the night. " +
            "At around 2 AM, 네이슨 found 앨런 dead in his room after noticing he was missing. " +
            "It was raining heavily, and 네이슨 immediately called the police. " +
            "Now it's 3 AM, the rain has stopped, and the three friends are being questioned in 앨런's house",
```

▼ 저장된 json 파일을 읽어 각 NPC에게 학습을 시킴

```
public void SetRole()
{
    string npcName = currentCharacter.ToString();
    chatMessages = new List<ChatMessage>(); // 각 NPC마다 새 리스트 생성
    ChatMessage systemMessage = new ChatMessage
    {
        Role = "system",
        Content = GetRole(npcName)
        + GetInstructions(npcName)
        + GetBackground(npcName)
        + GetFriends(npcName)
        + GetAlibi(npcName)
        + GetResponseGuidelines(npcName)
    };
    chatMessages.Add(systemMessage);
}
```



학습된 데이터를 기반으로 응답을 받는 모습을 확인할 수 있음

후기:

졸업작품 전시회를 준비하면서 게임 제작 과정의 여러 측면을 깊이 경험할 수 있었다. 'API를 활용한 게임을 만들어 보자'는 가벼운 다짐으로 시작한 프로젝트였지만, 예상보다 창의적이고 차별화된 게임을 완성하게 되어 큰 보람을 느낀다. 특히 OpenAI API를 활용하여 기존 게임과 차별화된 대화형 시스템을 구현한 점은 이번 프로젝트의 중요한 성과 중 하나로 생각된다.

여름방학부터 프로젝트를 시작하면서 일정 관리와 기술 구현에 대한 불안감도 있었지만, 나름 최선을 다한 결과 유의미한 작품을 완성할 수 있었다. 이번 경험을 통해 게임 제작에 대한 자신감과 더불어, 기술적 도전을 극복하는 과정에서 많은 교훈을 얻었다. 앞으로도 이번 프로젝트에서 얻은 경험을 바탕으로 더욱 독창적인 게임을 만들어 나갈 수 있을 것이라 기대된다.