캡스톤디자인 II 중간보고서(표지)

프로젝트명 : 웹 기반 메타버스 저작 플랫폼 구현

캡스톤 디자인II, 중간보고서

Version 1.0

개발 팀원 명(팀리더):최진아
이혜진
유선아

대표 연락처:010-3268-5527
e-mail: 20191792@edu.hanbat.ac.kr

캡스톤 디자인 II 중간보고서 내용

1. 요구사항 정의서에 명세된 기능에 대하여 현재까지 진척된 결과 및 그 내용을 기술하시오.

1) 링크 오브젝트 구현

```
74    VRModel.find({ region_id: vritem.region_id }, function (err, vrmodels) {
75      sceneName = vrmodels.scene_name;  // scene 이름
76      sceneId = vrmodels._id;          // scene id
77
78      var sceneName = [];
79      var sceneId = [];
80
81      for (i in vrmodels) {
82        sceneName.push(vrmodels[i]['scene_name']);
83        sceneId.push(vrmodels[i]['_id']);
84        // console.log(vrmodels[i]['scene_name']);
85        // console.log(vrmodels[i]['_id']);
86      }
87
88      console.log(sceneName);
89      console.log(sceneId);
90
91      return res.render("vr_item", { vid: req.params.vid, vrimage_id: vritem.image_file,
        arrowList: linkList, objectList: entityList, sceneName: sceneName, sceneId: sceneId });
92    });
```

vr.js에서 region, scene 정보를 저장되어 있는 VRModel에서 같은 region_id를 가지고 있는 scene들의 이름과 아이디를 찾는다. 찾은 scene의 이름과 아이디는 각각 sceneName, sceneId 변수에 저장한 뒤, vr_item.pug에 리스트 형태로 렌더하여 보낸다.

```
126                    a-gui-button.clickable(id="link" class="gui" width="2.5" height="0.75"
                       background-color='#7066e0' object-link=`sceneName: ${sceneName}; sceneId: $
                       {sceneId}` font-size="0.3" value=`LINK`)
```

vr_item.pug에서 vr.js에서 렌더한 sceneName과 sceneId를 이용하여 object-link를 통해서 넘겨준다. 이때, 'key: value' 형태로 형식이 지정된다.

```
49  AFRAME.registerComponent('object-link', {
50      schema: {
51          sceneName: { type: 'string', default: 'none' },
52          sceneId: { type: 'string', default: 'none' }
53      },
```

pressed-gui-button.js에서 object-link로부터 넘겨 받은 sceneName과 sceneId를 불러오기 위해서 스키마를 정의한다. 스키마에서는 문자형(string) 타입으로 sceneName, sceneId를 정의한다.

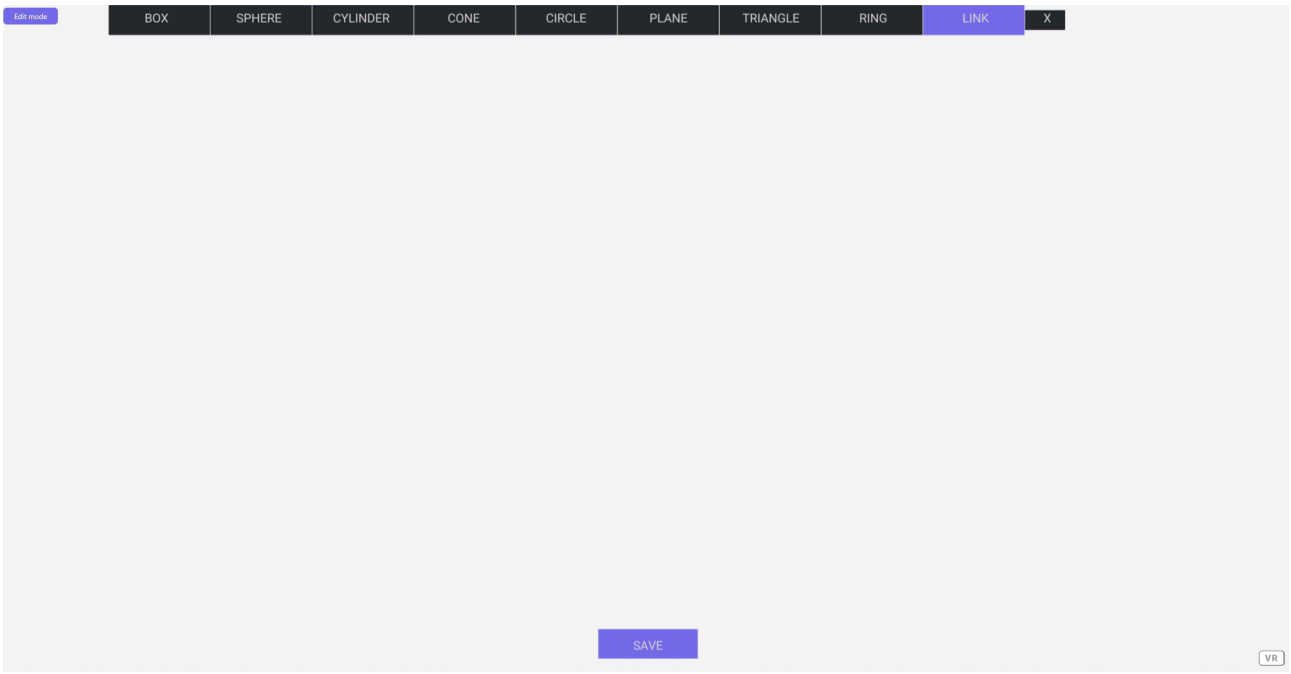
```
54      init: function () {
55          // console.log("========================================");
56          // console.log(this.data.sceneName);
57          // console.log(this.data.sceneId);
58
59          var HREF = 'http://localhost:3000/vr/';
60
61          var nameList = this.data.sceneName.split(',');
62          var idList = this.data.sceneId.split(',');
```

scene을 연결하기 위해서 HREF로 공통으로 사용하는 url의 주소를 var를 통해서 정의한다. object-link에서 key-value 형태로 받은 데이터는 nameList, idList 라는 변수에 콤마를 기준으로 분리하여 저장한다.

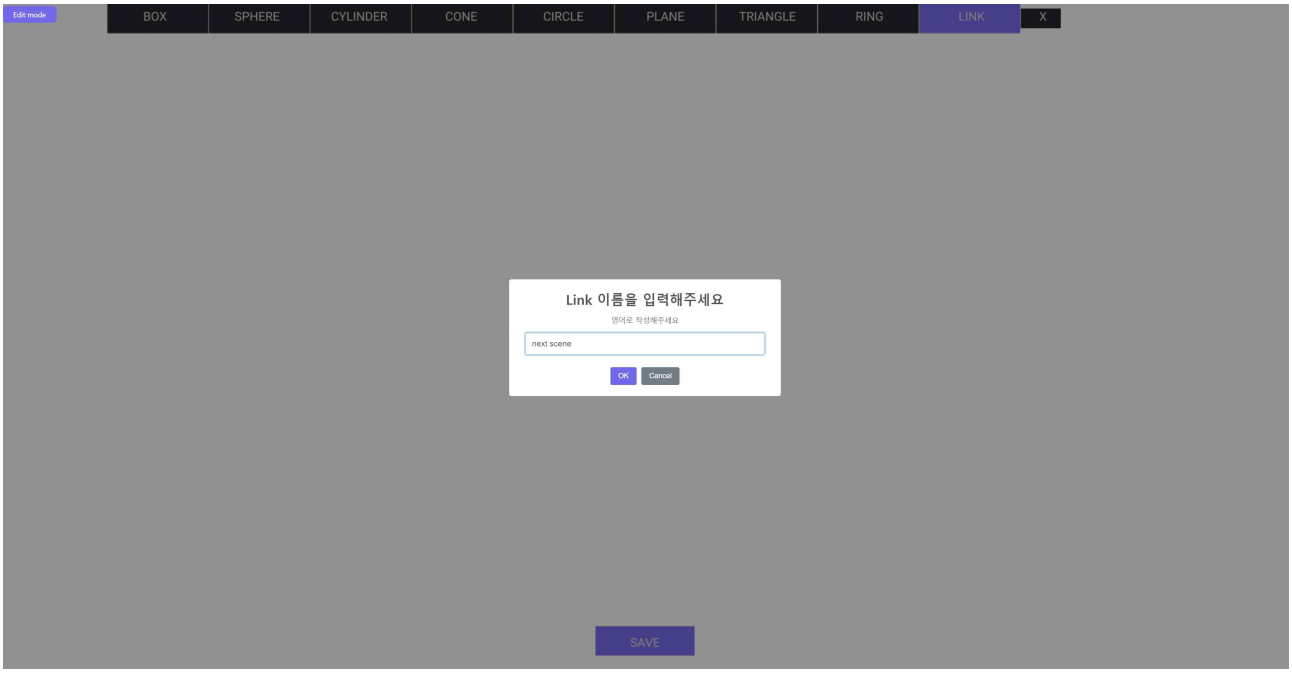
```
78                  const { value: text } = await Swal.fire({
79                      title: 'Link 이름을 입력해주세요',
80                      input: 'text',
81                      inputLabel: '영어로 작성해주세요',
82                      width: 600,
83                      inputPlaceholder: 'Link 이름 작성',
84                      showCancelButton: true
85                  })
86
87                  if (text) {
88                      //list
89                      const { value: scene } = await Swal.fire({
90                          title: 'Scene 선택',
91                          input: 'select',
92                          inputOptions: {
93                              'Scene List': nameList
94                          },
95                          inputPlaceholder: 'Scene 선택',
96                          showCancelButton: true,
97                          inputValidator: (value) => {
98                              return new Promise((resolve) => {
99                                  if (value) {
100                                     sceneHref = HREF + idList[value];
101                                     resolve();
102                                 } else {
103                                     resolve('연결할 Scene을 선택해주세요!');
104                                 }
105                             })
106                         }
107                     })
```

링크 오브젝트를 생성하기 위해서 링크 버튼을 클릭하면 먼저 링크의 이름을 입력하라는 메시지가 나온다. 링크의 이름은 링크 버튼이 생성되었을 때 상단에 표기되는 이름을 말한다. 이후 현재 링크를 생성하고 있는 scene에서 region_id가 같은 다른 scene들의 이름을 리스트 형태로 보여준다. 사용자가 scene 이름을 선택하면 해당 이름을 통해 scene의 id를 찾고 이를 주소로 변환하여 저장한다.

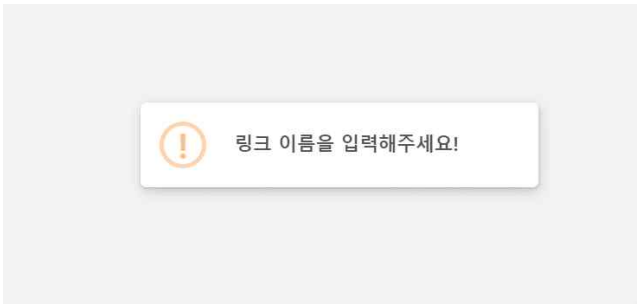
실제 구현한 모습은 다음과 같다.

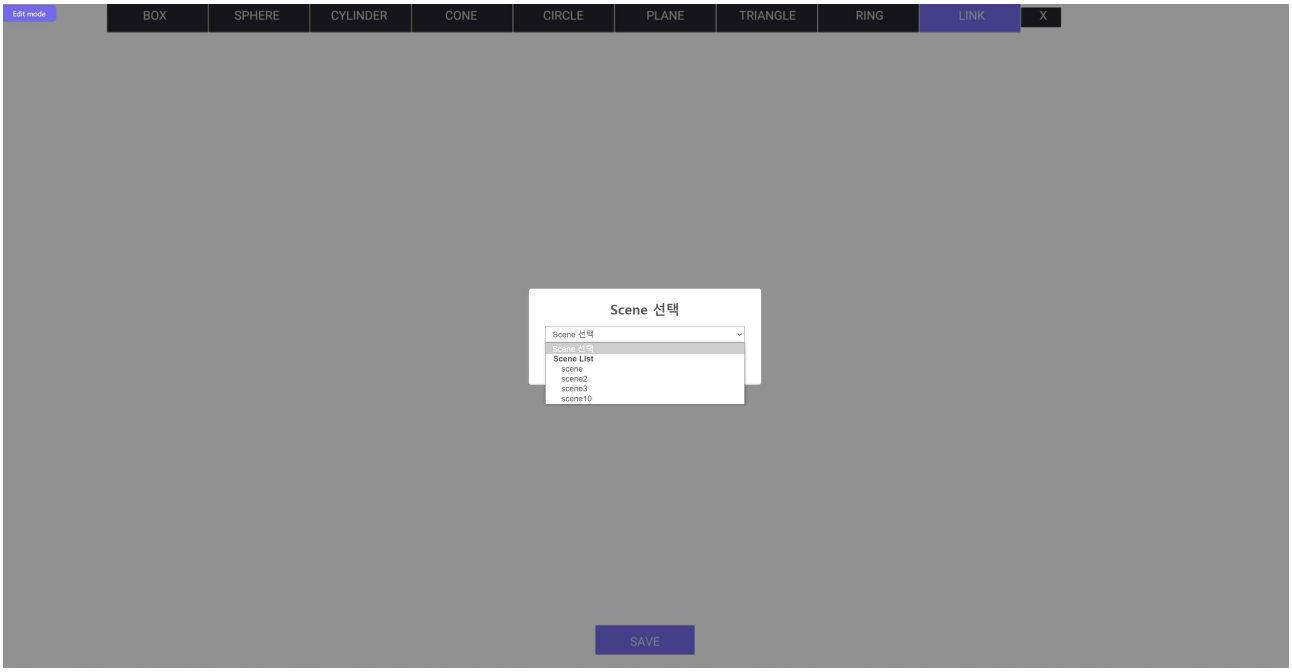


먼저 링크 오브젝트를 생성하기 위해서 LINK 버튼을 클릭한다.

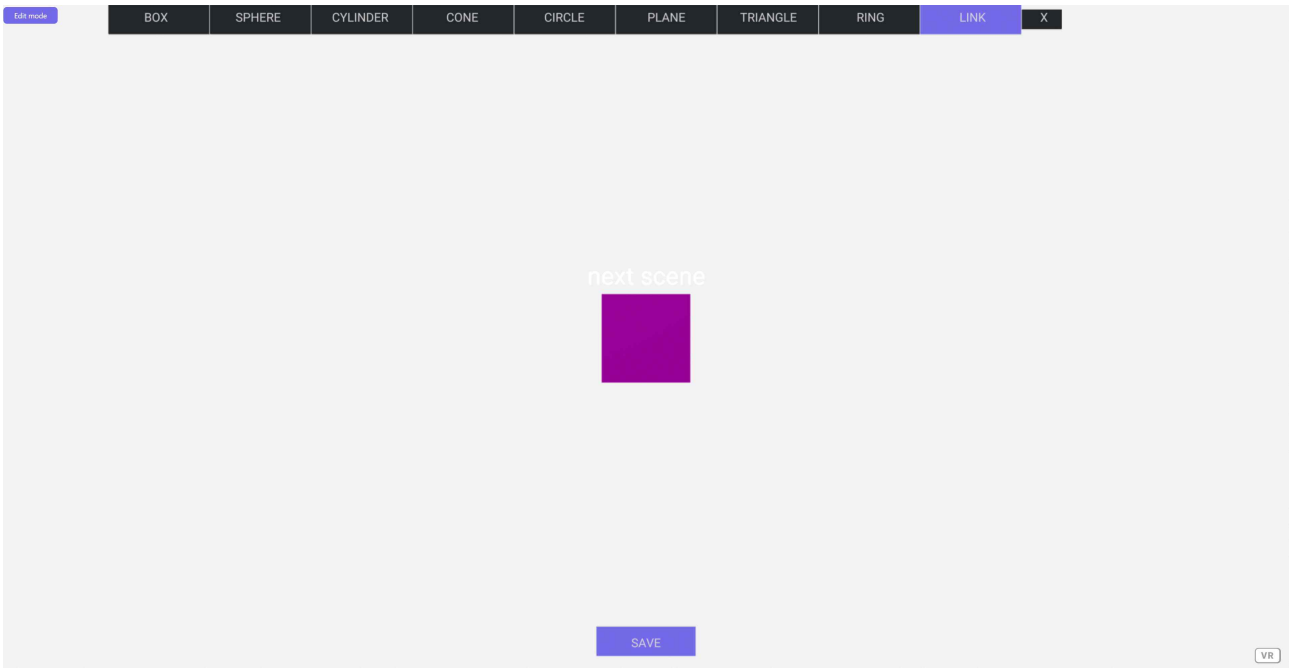


링크 오브젝트의 이름을 입력한 뒤 OK 버튼을 누른다.
링크 오브젝트의 이름을 입력하지 않으면 아래와 같이 안내 문구가 뜬다.





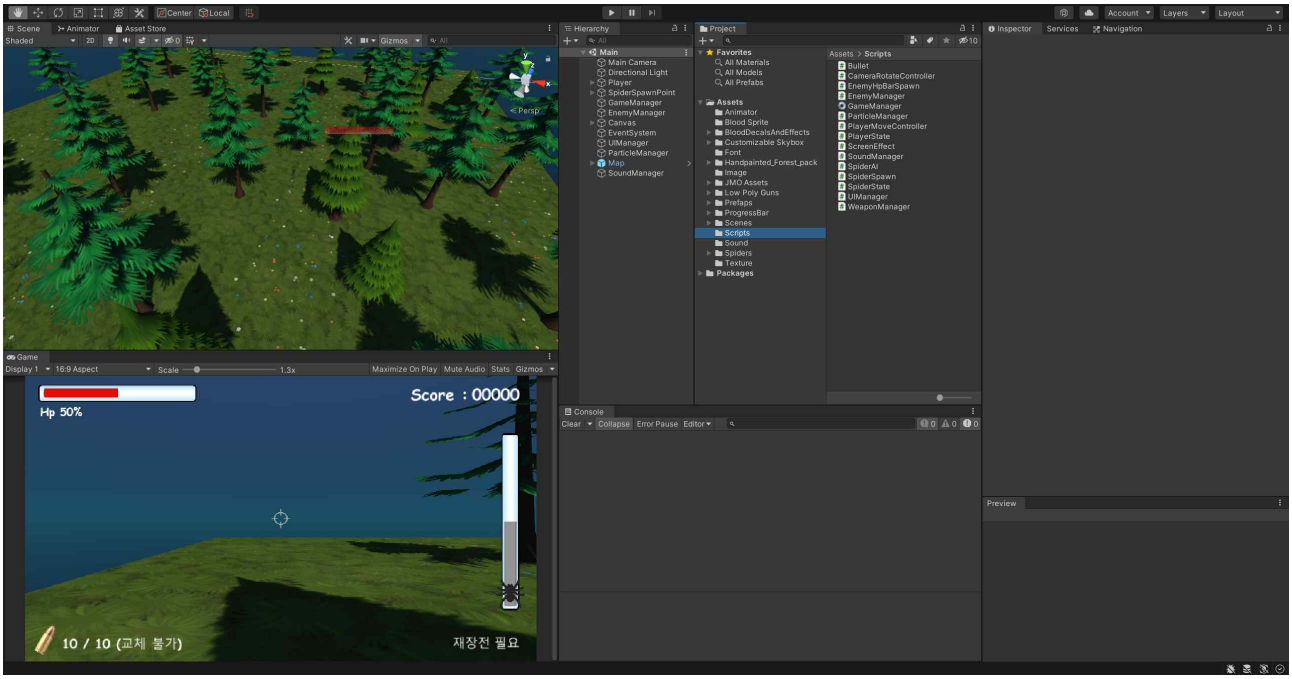
여러 Scene List 중에 연결하고 싶은 scene을 선택한다.



링크 오브젝트의 이름과 함께 링크 오브젝트가 생성되고, 링크 오브젝트를 클릭하면 선택한 scene으로 이동한다.

2) 미니게임 구현

- 실제 게임화면



- 소스 코드

```
6   public class GameManager : MonoBehaviour
7   {
8       // 싱글턴 패턴
9       public static GameManager instance;
10
11      public float playerMaxHp = 100;     // 플레이어의 최대 체력
12      public float playerCurrentHp = 100; // 플레이어의 현재 체력
13
14      public int weaponIndex;             // 현재 장착한 무기의 인덱스
15
16      public int[] bulletMaxCount;        // 최대 총알 개수 저장 배열
17      public int[] bulletCurrentCount;    // 현재 총알 개수 저장 배열
18
19      public float bulletDamage = 1.0f;   // 총알이 주는 데미지
20
21      public int score;                   // 점수
22
23      public bool isGameOver = false;     // 게임 오버인지 확인하는 플래그
24
25      public PlayerState.State playerState = PlayerState.State.IDLE; // 현재 플레이어의 상태
26
27      // 싱글턴 패턴
        Unity 메시지 | 참조 0개
28      private void Awake()
29      {
30          instance = this;
31      }
```

GameManager.cs : 게임 세부 설정에 중심이 되는 코드

```
5   public class EnemyManager : MonoBehaviour
6   {
7       // 싱글턴 패턴
8       public static EnemyManager instance;
9
10      public int enemyMaxSpawnCount = 10;    // 최대 생성될 수 있는 적의 마리수
11      public int enemyCurrentSpawnCount = 0; // 현재 생성된 적의 마리수
12
13      public float enemyMaxHp = 5.0f;        // 적의 최대 체력
14
15      public float enemyChaseDelay = 2.0f;   // 적이 추적을 시작하기 까지의 딜레이
16
17      public int enemyKillScore = 100;       // 적을 죽였을 때 얻을 수 있는 점수
18
19      // 싱글턴 패턴
        Unity 메시지 | 0 참조
20      private void Awake()
21      {
22          instance = this;
23      }
```

EnemyManager.cs : 적의 생성과 적의 설정을 담당하는 코드

```
6   public class SpiderState : MonoBehaviour
7   {
8       Transform tr;
9
10      Rigidbody rbody;
11      Animator animator;
12      Transform player;
13
        참조 15개
14      enum State
15      {
16          IDLE = 0,
17          WALK,
18          ATTACK,
19          DAMAGE,
20          DEAD
21      };
22
23      State enemyState = State.IDLE;
24
25      public float enemyCurrentHp = 100.0f;
26
27      float moveDelay = 2.0f;
28      bool isMoveDelay = true;
29
30      float chaseDistance = 2.0f;
31
32      bool isDead = false;
```

```
58      void Update()
59      {
60          switch (enemyState)
61          {
62              case State.IDLE:
63                  {
64                      if(isMoveDelay)
65                      {
66                          StartCoroutine(WaitMoveDelay());
67                      }
68                      else
69                      {
70                          rbody.constraints = RigidbodyConstraints.FreezeAll;
71
72                          animator.SetBool("isWalk", true);
73                          enemyState = State.WALK;
74                      }
75                  }
76                  break;
77              case State.WALK:
78                  {
79                      this.GetComponent<SpiderAI>().ChaseTarget();
80
81                      if (Vector3.Distance(tr.position, player.position) < chaseDistance)
82                      {
83                          animator.SetBool("isAttack", true);
84                          enemyState = State.ATTACK;
85                      }
86                  }
87                  break;
```

SpiderState.cs : 적 개체의 상태를 담당하는 코드, 각 상태 별로 switch문을 통해 지정된 동작을 수행함

```
6   public class SpiderAI : MonoBehaviour
7   {
8       Transform target;
9
10      Vector3 destination;
11      Vector3 targetDestination;
12
13      NavMeshAgent navMeshAgent;
14
15      float chaseDelay = 2.0f;
16      bool isChase = false;
```

```
37      public void ChaseTarget()
38      {
39          if (!isChase)
40          {
41              StartCoroutine(WaitChaseDelay());
42          }
43          else
44          {
45              targetDestination = target.position;
46
47              if (destination != targetDestination)
48              {
49                  navMeshAgent.enabled = true;
50                  navMeshAgent.SetDestination(targetDestination);
51              }
52              else
53              {
54                  navMeshAgent.enabled = false;
55              }
56          }
57      }
```

SpiderAI : 적 개체가 플레이어를 추적하도록 만드는 코드, NavMeshAgent를 통해 플레이어와의 최단 경로를 계산하여 추적한다.

```
5   public class PlayerState : MonoBehaviour
6   {
        참조 17개
7       public enum State
8       {
9           IDLE = 0,
10          WALK,
11          ATTACK,
12          DAMAGE,
13          DEAD
14      };
```

```
25      void Update()
26      {
27          switch (GameManager.instance.playerState)
28          {
29              case State.IDLE:
30                  {
31                      if (Input.GetAxis("Horizontal") != 0 || Input.GetAxis("Vertical") != 0)
32                      {
33                          GameManager.instance.playerState = State.WALK;
34                      }
35
36                      if(Input.GetMouseButton(0))
37                      {
38                          GameManager.instance.playerState = State.ATTACK;
39                      }
40                  }
41                  break;
42              case State.WALK:
43                  {
44                      if(Input.GetMouseButton(0))
45                      {
46                          GameManager.instance.playerState = State.ATTACK;
47                      }
```

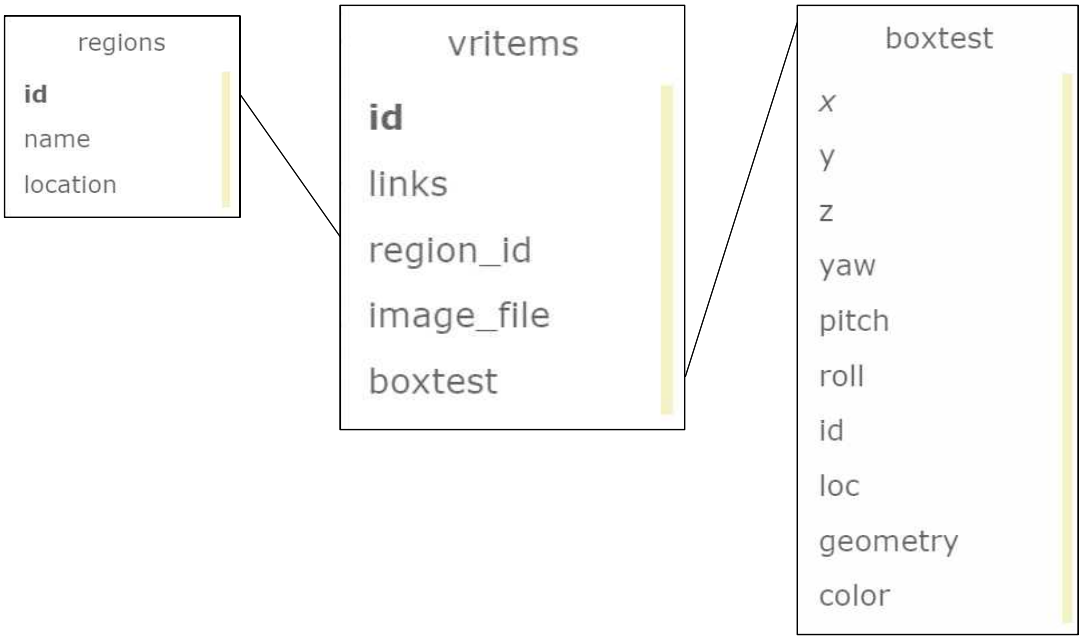
PlayerState.cs : 플레이어의 상태를 담당하는 코드, 각 상태 별로 switch문을 통해 지정된 동작을 수행함

```
5   public class WeaponManager : MonoBehaviour
6   {
7       public GameObject[] weapons;       // 무기 종류 저장 배열
8       public GameObject weaponPosition;  // 무기가 생성되는 위치
9
10      GameObject currentWeapon;          // 현재 장착한 무기
11
12      public float changeDelay = 5.0f;   // 무기 변경시, 다음 변경까지의 딜레이
13      bool isChangeDelay = false;        // 무기 변경 딜레이인지 체크하는 플래그
14
15      public GameObject bullet;          // 총알 오브젝트
16      Transform bulletSpawnPoint;        // 총알이 생성되는 위치
17
18      public float[] fireDelay;          // 총알 발사 딜레이 (무기 종류에 따라 달라짐)
19      bool isFireDelay = false;          // 총알 발사 딜레이인지 체크하는 플래그
20
21      public GameObject muzzleEffect;    // 머즐 이펙트 오브젝트
22      Transform muzzlePoint;             // 머즐 이펙트가 생성되는 위치
23
24      public Transform rayPoint;         // 광선이 생성되는 위치
25
26      float reloadWarningDelay = 1.0f;   // 재장전 경고 메세지가 떠 있는 딜레이
27      bool activeReloadWarning = false;  // 재장전 메세지 활성화 플래그
```

```
38      void Update()
39      {
40          ChangeWeapon(); // 현재 장착한 무기 변경
41          Fire();         // 총알 발사
42          Reload();       // 재장전
43      }
```

WeaponManager.cs : 무기 생성 및 변경, 총알 발사, 재장전을 담당하는 코드

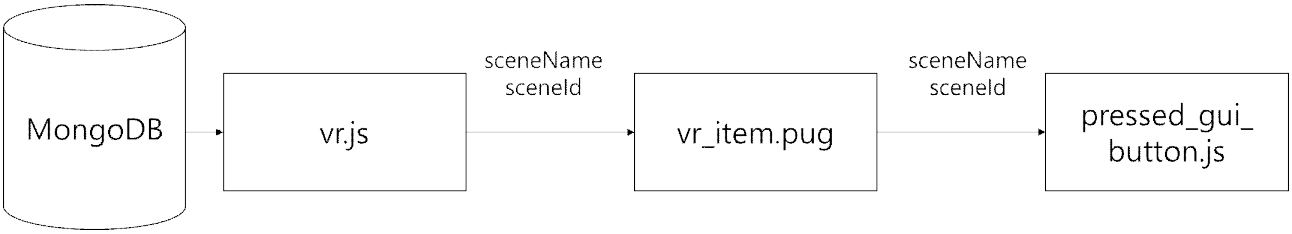
3) 데이터베이스 구조


regions
id
name
location
vritems
id
links
region_id
image_file
boxtest
boxtest
x
y
z
yaw
pitch
roll
id
loc
geometry
color

2. 프로젝트 수행을 위해 적용된 추진전략, 수행 방법의 결과를 작성하고, 만일 적용과정에서 문제점이 도출되었다면 그 문제를 분석하고 해결방안을 기술하시오.

1) 기능 구현 과정

- 링크 오브젝트 구현 중 문제 상황과 해결 방법



같은 region을 가진 scene의 이름과 아이디를 가져오기 위해서 vr.js에서 데이터베이스로부터 데이터를 찾은 다음 vr_item.pug로 렌더링을 통해 전송하고, vr_item.pug에서는 인자를 통해 pressed_gui_button.js로 데이터를 전송하여 링크 오브젝트를 생성할 때 sceneList를 불러오도록 구현했다.

- 미니게임 구현 중 문제 상황과 해결 방법

화면 이펙트를 담당하는 ScreenEffect.cs에서 GameManager.cs에서의 playerState 변수 내용을 Update()에서 실시간으로 인식하지 못하는 문제점이 발생하였다. 따라서 ScreenEffect.cs에서 처리할 내용을 함수로 만들고, 필요한 부분에서 해당 함수를 호출하는 식으로 문제를 해결하였다.

```
28      // 플레이어 피격 이펙트 보여주기
        참조 1개
29      public void ShowPlayerDamage()
30      {
31          // 플레이어 상태가 피격상태인 경우
32          if (GameManager.instance.playerState == PlayerState.State.DAMAGE)
33          {
34              // 플레이어 피격 이펙트 나타냈다가 숨기기
35              StartCoroutine(FadeDamageEffect());
36          }
37      }
```

ScreenEffect.cs 의 코드 일부

```
67                  case State.DAMAGE:
68                      {
69                          GameManager.instance.ChangePlayerHp(5.0f);
70
71                          UIManager.instance.gameObject.GetComponent<ScreenEffect>().ShowPlayerDamage();
72
73                          if (GameManager.instance.playerCurrentHp <= 0)
74                          {
75                              GameManager.instance.playerState = State.DEAD;
76                          }
77                          else
78                          {
79                              GameManager.instance.playerState = State.IDLE;
80                          }
81                      }
82                      break;
```

PlayerState.cs 의 코드 일부

2) 팀원의 책임 및 역할 분배

최진아 학생은 프로젝트 팀장으로서 프론트엔드를 개발하고, 이혜진 학생은 데이터베이스 개발 및 관리를 맡고, 유선아 학생은 유니티를 이용한 미니게임 개발을 맡았습니다. 하지만 적은 인원으로 모든 영역을 개발할 때, 각자가 맡은 역할만 수행하는 것은 프로젝트 진행

속도에 무리가 있다고 판단하였습니다. 따라서 한 사람씩 맡은 파트를 책임자로 정하고, 책임자 중심으로 다른 학생들이 개발을 지원하는 방식으로 프로젝트를 운영했습니다. 예를 들어, 최진아 학생이 백엔드 개발의 책임자가 되어 백엔드 개발에서 문제가 발생하거나 개발이 지연되는 경우 이혜진, 유선아 학생이 백엔드 개발에 함께 참여하여 개발의 속도와 질을 높였습니다. 각 파트마다 책임자가 존재하기 때문에 의견 충돌이 있거나 문제가 발생한 경우 책임자를 중심으로 의사 결정을 할 수 있었고, 프로젝트를 수행하면서 팀원들의 책임과 역할을 명확하게 분배하여 프로젝트를 수행하고 있습니다.

3) 프로젝트 수행 내용 및 앞으로의 계획

프로젝트 일정은 아래 표와 같습니다.

	7월	8월	9월	10월	11월
프로젝트 계획 및 논의					
프로토타입에 대한 연구 논문 작성					
프로젝트 수행(개발)					
프로젝트 유지보수 및 마무리					

프로젝트 계획 및 논의를 시작으로 캡스톤디자인2를 시작했습니다. 프로토타입에 대한 연구 논문을 작성하여 한국공학교육학회에서 진행하는 2022 공학교육학술대회에 ‘온오프라인 하이브리드형 공학 진로프로그램의 설계 및 운영’, ‘K-12에서 공학 교육 도구를 활용한 진로 탐색 프로그램 연구’, ‘융합인재교육을 위한 인-월드 에디터와 교육 콘텐츠 설계 및 운영’ 총 3편의 논문을 제출했습니다. 8월부터 10월까지 프로젝트에 대한 개발을 진행할 예정이며, 11월에 프로젝트 유지보수 및 마무리를 할 계획입니다.

[별첨1]

프로젝트명 : 웹 기반 메타버스 저작 플랫폼 구현

소프트웨어 요구사항 정의서

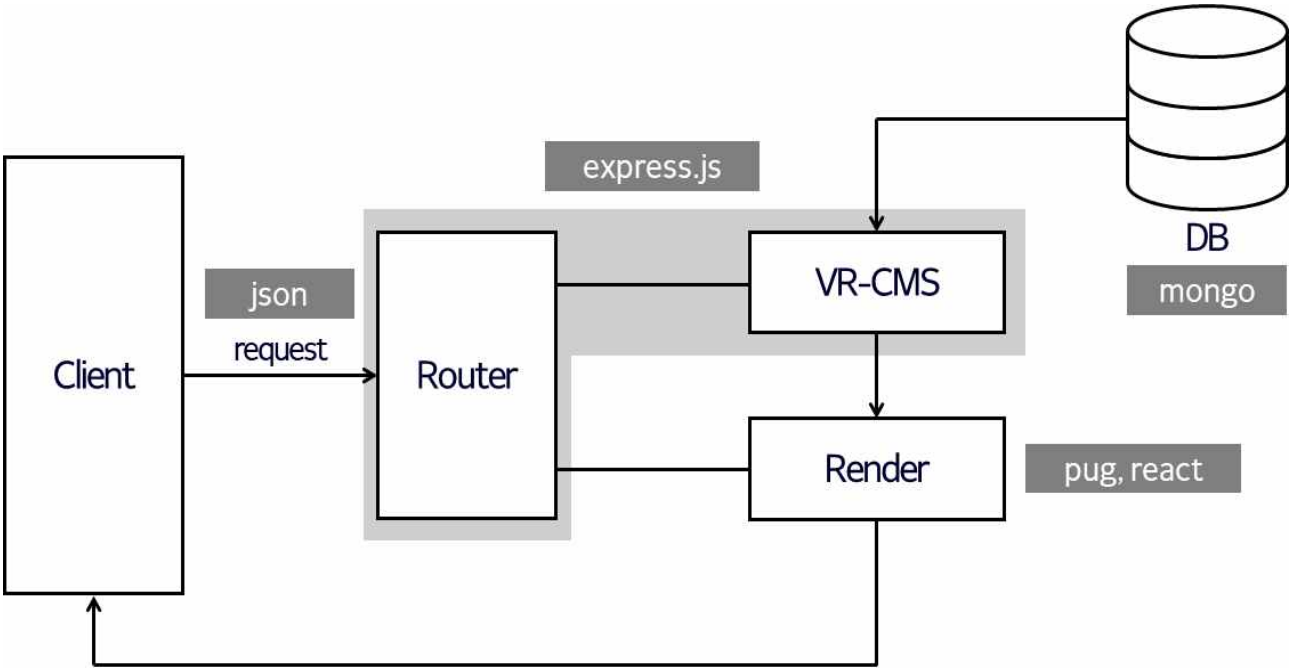
Version 1.1

개발 팀원 명(팀리더):최진아
이혜진
유선아

대표 연락처:010-3268-5527
e-mail: 20191792@edu.hanbat.ac.kr

목차

1. 시스템 개요


express.js
Client
json
request
Router
VR-CMS
Render
pug, react
DB
mongo

2. 시스템 장비 구성요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
통신 서버	ECR-01	정의 : 실시간 데이터 전송을 위한 TCP-IP 통신서버 • 해당 라즈베리파이4를 웹 페이지를 운영하기 위한 서버로 사용함 • 수량 : 1EA • 라즈베리파이4 2GB • Broadcom BCM2711, 쿼드 코어 Cortex-A72 (ARM v8) 64 비트 SoC @ 1.5GHz • 메모리 : 2GB • 연결성 : 2.4GHz 및 5.0GHz IEEE 802.11b / g / n / ac 무선 LAN, Bluetooth 5.0, BLE		
O/S 소프트웨어	ECR-02	정의 : O/S 소프트웨어 • 인터넷에 접속이 가능한 모든 소프트웨어 지원 가능(Window, Linux, Max OS, Ubuntu 등)		
DB 소프트웨어	ECR-03	정의 : DB 소프트웨어 • 소프트웨어 : MongoDB • 기능 : DB 관리 • 성능 및 특징 : 수평적 확장이 가능한 Document 지향 데이터베이스		

3. 기능 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
Region 생성 및 삭제	SFR-01	정의 : 사용자만의 Region을 생성하여 Region 내의 원하는 씬을 구성함 • 12 bytes binary 숫자에 대한 MongoDB 24자리 hex 문자열 ID를 자동으로 부여하여 Region을 생성함 • 추가(ADD), 삭제(DELETE) 버튼을 눌러 생성한 Region 내의 각 씬을 추가하거나 삭제하고 자율적으로 구성함		
씬(Scene) 생성/읽기/ 삭제	SFR-02	정의 : 360° 4K 이미지를 배경으로 가상공간인 Scene을 생성/삭제함 • 사용자가 원하는 이미지를 씬의 배경으로 하여 Scene을 생성(Create)하여 DB에 저장함 • DB에 저장된 Scene을 불러와 읽음(Read) • Scene에 대해 수정(Update)을 가한 뒤 수정된 데이터를 DB에 반영함 • DB에 저장되어있는 Scene 데이터를 삭제(Delete)함		
씬(Scene) 편집	SFR-03	정의 : 사용자가 생성한 씬을 주어진 오브젝트를 이용하여 꾸밀 수 있음 • 3D Object의 CRUD 기능을 구현함 • 3D Object를 생성(Create)하여 DB에 저장함 • DB에 저장된 3D Object를 불러와 읽음(Read) • 3D Object에 대해 수정(Update)을 가한 뒤 수정된 데이터를 DB에 반영함 • DB에 저장되어있는 데이터를 삭제(Delete)함		
씬(Scene) 연결	SFR-04	정의 : 각 씬들을 link 오브젝트를 통해 연결할 수 있음 • 플랫폼 내팝업 창을 통해 사용자가 원하는 이름으로 링크를 설정할 수 있음 • 사용자가 사용중인 region내의 씬 리스트에서 연결하고자 하는 씬으로 선택함		
3D object Repository 구축	SFR-05	정의 : 사용자가 사용할 수 있는 3D Object를 불러옴 • 상자(box), 원기둥(cylinder) 등 3D Object를 불러옴		
Region 공유	SFR-06	정의 : 생성한 Region을 URL로 공유하여 다수의 사용자가 접속하여 봄 단축 아이콘을 통해 URL을 공유함		
외부 3D object 삽입	SFR-07	정의 : 외부에서 다운로드 받은 3D objectf를 플랫폼과 연결시켜 사용자가 씬 내부에 삽입할 수 있음 • 다운로드받은 3D object를 사용자가 원하는 위치에 삽입할 수 있음 • 씬에 추가된 3D object를 사용자가 원하는 대로 편집할 수 있음 • 추가된 3D object를 DB에 저장함		

미니게임	SFR-08	정의 : 씬 내에 3D object의 추가 기능으로 미니게임 기능을 추가할 수 있음		

4. 성능 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
웹 페이지 로딩 시간	PER-01	• 플랫폼 이용 사용자의 화면에 가상 세계가 로딩되는 시간 • 각 가상 세계 내에 배치된 3D object들을 포함한 전체 웹 페이지가 사용자가 요구한 시점으로부터 10초 이내에 완벽하게 로딩되어야 함		
동시 사용자 접속 수	PER-02	• 웹페이지 당 동시 사용자 수 50명 이상을 지원해야 하고 성능이 저하되지 않아야 함		
데이터 형식 오류 응답 시간	PER-03	• 사용자로부터 발생한 모든 오류에 대해서 5초 이내에 오류 웹페이지 또는 메시지를 전송해야 함		

5. 인터페이스 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
3D Object GUI	SIR-01	정의 : 사용자가 3D Object와 관련된 기능 사용 시의 인터페이스 • 사용자는 구체적인 사용법 없이도 인터페이스만을 보고 3D Object를 생성, 추가, 편집할 수 있어야 함		
사용자/편집자 구분	SIR-02	정의 : 사용자/편집자 구분 • 사용자는 새로운 가상세계를 생성 또는 삭제하고 싶을 경우 편집자 모드를 통해서 원하는 대로 편집할 수 있어야 함		
오류 웹페이지 안내	SIR-03	정의 : 사용자의 잘못된 입력으로 인한 오류 발생 시의 오류 웹페이지 • 해당 웹 플랫폼을 사용하면서 사용자가 잘못된 입력을 할 경우 발생한 오류에 관한 설명을 포함한 오류 웹페이지를 사용자에게 제시해야 함		

6. 데이터 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
데이터베이스 구축	DAR-01	정의 : 사용자가 서비스를 이용하면서 생성된 데이터를 보관하기 위한 데이터베이스를 구축함 • 내부 데이터베이스는 MongoDB를 사용함 • 데이터베이스는 관리자 외에 접근할 수 없음		
데이터 저장	DAR-02	정의 : 사용자가 서비스를 이용하면서 생성된 데이터를 내부 데이터베이스에 저장함 • 데이터베이스에 저장된 정보는 사용자가 삭제하지 않는 이상 보관함		
데이터 수정	DAR-03	정의 : 사용자가 서비스를 이용하면서 생성된 데이터를 수정할 때 내부 데이터베이스에도 반영되어야 함 • 데이터베이스에 저장된 정보는 사용자가 수정하는 즉시 반영되어야 함		

7. 테스트 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
테스트 계획 수립	TER-01	정의 : 요구사항이 제대로 반영되었는지를 테스트하기 위해서 테스트 계획을 수립함 • 테스트 단계별 수행할 테스트와 목표, 활동, 보완 절차를 수립함 • 테스트 대상, 기간, 성능 목표, 범위, 방법, 시나리오, 정량화된 측정지표 등 상세 계획을 수립함		
테스트 수행	TER-02	정의 : 테스트 계획을 바탕으로 테스트를 실시함 • 설계된 단위에 맞추어서 수시로 단위 테스트를 진행함 • 개발된 시스템이 시나리오에 맞게 완벽하게 수행되는지에 맞춰서 알파 테스트를 진행함		
단위 테스트	TER-03	정의 : 단위 테스트 계획과 예상결과를 포함하는 계획서 작성 • 개발 단계에 따라 개발자 수준에서 단위 테스트를 수행함		
알파 테스트	TER-04	정의 : 알파 테스트 계획과 예상결과를 포함하는 계획서 작성 • 시스템의 기능이 정상적으로 동작하는지 확인하기 위해서 계획 및 상세 시나리오를 작성하고 테스트를 수행함		

8. 보안 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
장비 보안 요구사항	SER-01	정의 : 시스템을 운영하는 통신 서버의 물리적 · 시스템적 보안에 대한 요구사항 • 통신 서버에 물리적으로 접근할 수 없도록 분리된 공간에 위치해야 함 • 통신 서버에 접속하기 한 계정의 아이디와 패스워드는 관리자 외 알 수 없도록 관리해야 함		
네트워크 보안 요구사항	SER-02	정의 : 네트워크 접근 통제, 네트워크 장비의 취약성 및 구성 설정에 대한 보안 요구사항 등 통신을 위해 사용하는 장비 및 접근과 관련한 요구사항 • 유선 네트워크를 사용하여 무선 네트워크 사용으로 발생할 수 있는 취약점을 예방함		

9. 품질 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
장애 복구 대책	QUR-01	• 장애가 발생하지 않도록 시스템을 구축하여야 하며 장애 발생 시 신속하게 조치하여야 한다. • 시스템의 최적 운용 방안 및 응급처리 방안 등 상세한 장애대책을 제출하여야 한다. • 단계별 장애를 분류하여 체계적이고 효과적인 백업방안과 복구지침을 마련하여야 한다.		
웹 호환성	QUR-02	• 특정 브라우저에 적용되는 기술은 가급적 배제하고, 웹브라우저에 대한 호환성이 확보될 수 있도록 설계함		
정보제공	QUR-03	• 시스템은 온라인 에러 메시지 기능을 제공해야 함 • 시스템은 콘텐츠의 모양이나 배치를 논리적으로 이해하기 쉽게 구성하여 설계·개발해야 함 • 온라인 서식과 관련된 콘텐츠는 입력항목에 대한 설명을 설계시 포함해야 함		

10. 제약 사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
프레임워크	COR-01	• JavaScript, Aframe, Node.js 등에서 기본으로 제공되는 라이브러리 사용		
프로그래밍 언어	COR-02	• 프론트엔드에는 JavaScript, HTML, CSS, Pug, Aframe을 사용함 • 백엔드에는 Node.js, JavaScript를 사용함		
개발 방법론	COR-03	• 알파 테스트와 애자일 방법론 사용 • 알파 테스트에서 받은 피드백 기반으로 애자일 방법론을 적용함 • 일정 주기로 실시하는 알파 테스트마다 기능들을 추가하며 프로토타입을 제작함		
업무 모듈화 및 자원 활용 방안	COR-04	• 유연성, 확장성을 확보할 수 있도록 모듈화 개발전략을 반영함 • 현재 보유하여 활용 가능한 H/W, S/W를 최대한 재활용하며, 추가 도입이 필요한 솔루션의 경우 본 용역에 포함하여 수행함		
시스템 설계	COR-05	• 기존 구축되어 있는 현행 시스템 구조 및 전체 표준화 호환성, 시스템 분산설계, 데이터 유형, 프로세스 환경 유형, 사용자 유형, 시스템 토폴로지가 고려되어 구조 설계가 이루어져야함		

11. 프로젝트 관리 요구사항

요구 사항 명칭	고유 번호	요구사항 정의 및 세부내용	산출 정보	관련 요구 사항
프로젝트 수행조직	PMR-01	• 프론트엔드 개발, 백엔드 개발은 데이터베이스의 데이터를 이용해 가상공간의 입출력을 처리함 • 데이터베이스 개발 및 관리는 데이터베이스를 개발하고 데이터를 관리함		
프로젝트 일정계획	PMR-02	• 주기적으로 알파테스트를 진행하며 개발을 진행함		
단계별 산출물	PMR-03	• 계획서, 중간보고서, 개발 및 커스터마이징 결과물 확인함		