TRIAPO (2024)

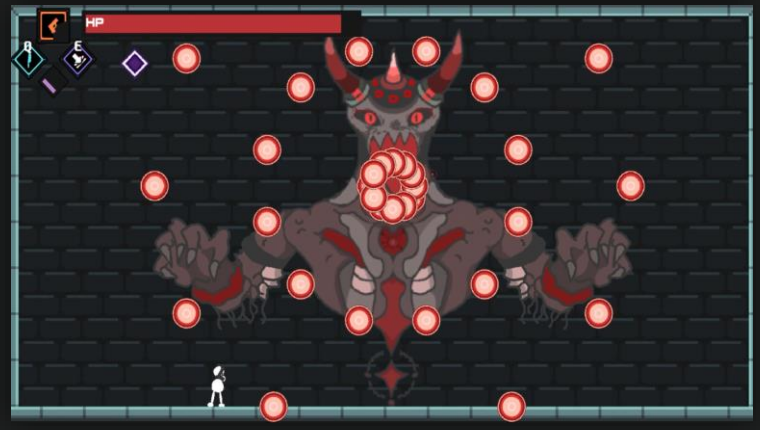
- **프로젝트 내용** : 적들이 훔쳐간 전설의 무기 3개를 되찾기 위해 적들을 물리치는 플랫포머 액션 게임입니다. 적을 쓰러뜨리고 손에 넣은 무기들은 같이 연계해서 사용 시 더 강력한 콤보를 사용할 수 있습니다.
- **사용 툴 및 기술** : C#, Unity
- **제작 기간** : 2024.05.03 ~ 2024.06.05 (1개월)
- **제작 인원** : 4명 (프로그래밍 4명/그래픽 1명/기획 2명(기획 보조 1명)/QA 1명)
- **제작 동기** : 졸업 전에 캐릭터 조작의 재미를 살린 2D 게임을 만들어보고 싶어 제작한 프로젝트입니다.
- **목표** : 정해진 기한 안에 완성해 졸업 프로젝트로서 최종 발표회에서 높은 성과를 거두는 것입니다.

Unity 프로젝트

[맡은 역할]
- 프로그래밍(메인), 그래픽, 기획 보조
    - 플레이어 캐릭터에 대한 규칙 및 이벤트 등을 중점으로 구현하고, 게임 전체적으로 필요한 기능들을 추가적으로 담당함.

[구현 내용]
- 플레이어 캐릭터 조작, 무기 교체 시스템, 무기 고유 능력, UI, 전투 시스템, 자동 저장, 도전 과제, 최종 보스 등

- 스크린샷



GitHub 링크

유튜브 링크

Unity 프로젝트

코드 링크

[게임 구조 : 무기 시스템] - SkillManager (1)

```
private void Start()
{
    owner = GetComponent<Entity>();

    if (hudControl) // 플레이어가 보는 UI에서 currentWeapon 값에 해당하는 무기가 무엇인지 보여줌.
        hudControl.ChangeCurrentWeapon(currentWeapon);

    ChangeWeaponSkill(false, currentWeapon); // currentWeapon 값에 해당하는 무기로 현재 무기 바꿈,
    CheckMPUI(); // Gauge 값을 체크하여 Gauge가 얼마나 모였는지 표시함. (Gauge는 보라색 마름모로 표시함. Gauge가 3이면 보라색 마름모가 3개이다.)
}
```

빨간색 박스에 들어있는 부분은 게이지를 나타내고, 하늘색 박스에 들어있는 부분은 보유한 무기를 나타낸다.

- 무기 교체 기능 및 연계 스킬의 작동 타이밍을 조절하는 기능을 하는 클래스입니다.
- 필살기와 스킬을 쓰기 위한 마나를 관리합니다.
- 자동으로 처음 실행되는 Start 함수에선 캐릭터에 대한 정보(hp, mp 등)가 저장되는 Entity 컴포넌트를 변수 owner에 저장하고, 캐릭터의 현재 무기와 현재 게이지를 UI에 표시하고 있습니다.

Unity 프로젝트

코드 링크

[게임 구조 : 무기 시스템] - SkillManager (2)

```
private void Update()
{
    if (owner)
        owner.aManager.ani.SetInteger("Weapon", currentWeapon); // 애니메이터의 파라미터 값을 바꿔 현재 무기에 맞는 애니메이션이 재생되도록 함.

    if (hudControl)
    {
        if (Input.GetKeyDown(KeyCode.Q))
        {
            ChangeWeaponSkill(true); // 왼쪽무기와 현재무기교체 (현재 무기는 왼쪽으로 이동)
        }

        if (Input.GetKeyDown(KeyCode.E))
        {
            ChangeWeaponSkill(false); // 오른쪽무기와 현재무기교체 (현재 무기는 오른쪽으로 이동)
        }
        if (owner.GetHp() > 0 && Input.GetKeyDown(KeyCode.W)) // 체력 회복
        {
            int curGauge = owner.aManager.ani.GetInteger("Gauge"); // 애니메이터에서 Gauge 값을 가져옴. Gauge는 스킬, 필살기 사용에 필요한 값이다.
            if (curGauge > 0) // Gauge 값이 0보다 크면 게이지 값 1을 소모하여 체력 회복이 가능하다.
            {
                MedalControlSystem.hpBeforeChange = (int)owner.GetHp();
                Instantiate(healEffect, owner.transform.position, Quaternion.identity).transform.parent = owner.transform;
                owner.SetHp(owner.GetHp() + owner.maxHP * 0.3f); // 체력 회복 퍼센테이지 0.3
                isHealthEvent = true;
                ReduceGauge();
            }
        }
    }
    if (owner.GetMp() == owner.maxMp) // 현재 MP값이 최대치일 때 처리
    {
        int curGauge = owner.aManager.ani.GetInteger("Gauge");
        if (curGauge < 3) // 값이 3이 아닐 경우 게이지 값이 증가한다.
        {
            owner.aManager.ani.SetInteger("Gauge", curGauge + 1);
            CheckMPUI();
        }
        owner.SetMp(0); // MP값이 0으로 초기화됨.
    }
}
```

- 프레임 단위로 호출되는 Update 함수의 경우엔 아래와 같은 작업을 진행합니다.
  1. 무기에 따라 다른 애니메이션을 재생하도록 애니메이터의 파라미터 값을 조절함.
  2. 무기 교체(Q, E 버튼으로 가능)에 대한 처리와 마나를 사용한 체력 회복, 마나 자동 충전 등을 진행함.

Unity 프로젝트

코드 링크

[게임 구조 : 무기 시스템] - SkillManager (3)

```
public void ChangeWeaponSkill(bool isLeftWeapon, int newWeapon = -1) // 첫번째 매개변수는 왼쪽 무기와 교체하는지 여부, 두번째 매개변수는 바꿀 무기의 번호이다.
{
    var previousWeapon = currentWeapon; // 교체 전 무기를 변수로 저장
    if (newWeapon == -1)  // 무기의 번호를 매개변수로 전달 안 할 경우
    {
        if (isLeftWeapon) // 왼쪽으로 교체하는 버튼을 눌렀을 경우
            currentWeapon = currentLeftWeapon; // 현재 무기를 기본 값으로 설정된 왼쪽무기로 바꿈.
        else
            currentWeapon = currentRightWeapon; // 현재 무기를 기본 값으로 설정된 오른쪽무기로 바꿈.
    }
    else
        currentWeapon = newWeapon; // 무기 번호에 맞는 무기로 바꿈.

    if(haveWeaponNum < currentWeapon) // 바꾼 무기의 번호(무기 번호는 얻는 순서와 같음)가 현재 보유한 무기의 수보다 작을 경우
    {
        currentWeapon = previousWeapon; // 이전 무기로 다시 바꿈.
        return;
    }

    hudControl.ChangeCurrentWeapon(currentWeapon); // 현재 무기를 UI에 반영

    bool left = false;
    for (int i = 0; i < 3; i++) // 다음 왼쪽 무기와 오른쪽 무기를 번호 순서에 따라 정함.
    {
        if (i != (currentWeapon - 1))
        {
            if (!left)
            {
                left = true;
                currentLeftWeapon = i + 1;
            }
            else
                currentRightWeapon = i + 1;
        }
    }
}
```

- ChangeWeaponSkill 함수에선 현재 무기를 왼쪽 또는 오른쪽 무기와 교체하는 작업을 진행하며, 무기 번호를 두번째 매개변수 값으로 전달하면 해당 무기로 바로 전환 가능하도록 구현했습니다.
  - 보유하지 않는 무기로는 바꿀 수 없게 처리함.
  - 무기를 교체한 이후엔 왼쪽 무기와 오른쪽 무기가 자동으로 무기 번호 순서대로 바뀜.

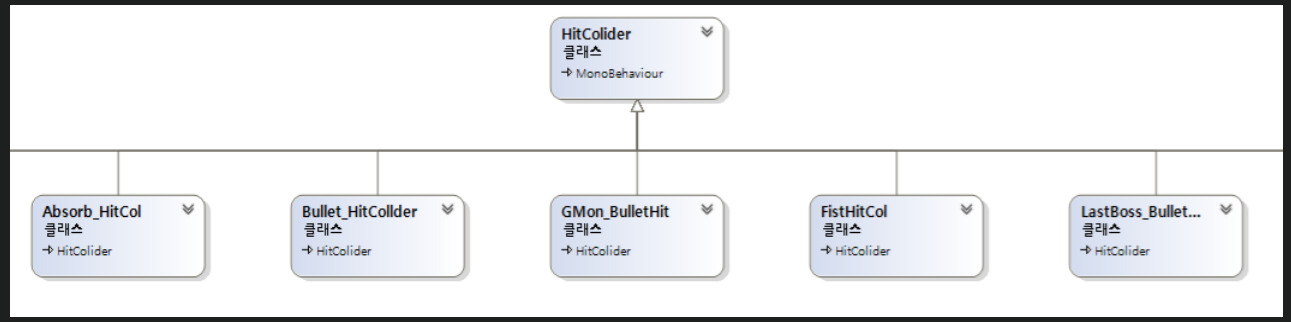
Unity 프로젝트

코드 링크

[게임 구조 : 공격 판정 관리] - HitColider

- HitColider 클래스는 각 Collider가 겹칠 때를 체크해 충돌 대상에게 일정 데미지를 줍니다.

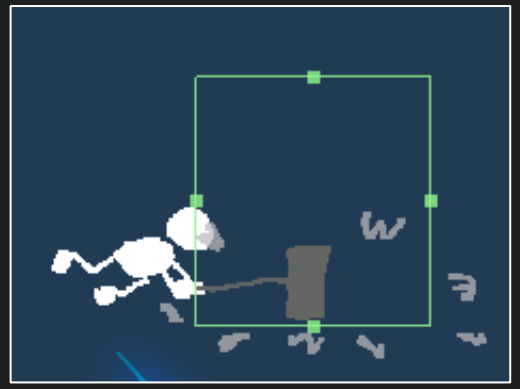
- HitColider를 상속하는 클래스들의 경우 충돌 시 HitColider의 기능 외 특정한 이벤트를 실행하고 있습니다.
  - 예시 1) 총알의 경우 플레이어의 검과 충돌 시 튕겨 남.
  - 예시 2) 흡수 공격의 경우 공격자가 공격을 성공 시 체력을 회복함.




```
참조 10개
public enum AttackType // 공격 타입 : 각 공격의 효과가 다르기에 구분한다.
{
    none,
    Player_SwordAtt,
    Player_GunAtt,
    Player_Hammer,
    Player_FinishdAtt
}
public AttackType attType = AttackType.none;
public bool isAbleDestroy = false;

public GameObject DestroyEffect;

⊙ Unity 메시지 | 참조 1개
protected void OnTriggerEnter2D(Collider2D other)
{
    if (owner == other.gameObject.GetComponent<Entity>())
        return;
    if (other.CompareTag("Camera")) // 카메라의 경우 무시
        return;
    if (playerIsOwn && (other.CompareTag("Player"))) // 플레이어의 공격은 플레이어 무시
        return;
```



```
if (owner.transform.position.x > entity.transform.position.x) // 상대를 공격 방향으로 날린다.
{
    if (!owner || owner.movement.PlayerType || entity.movement.PlayerType)
        entity.Damaged(attackForce, (-attackForce) * thrustValue);
}
else
{
    if (!owner || owner.movement.PlayerType || entity.movement.PlayerType)
        entity.Damaged(attackForce, attackForce * thrustValue);
}
```

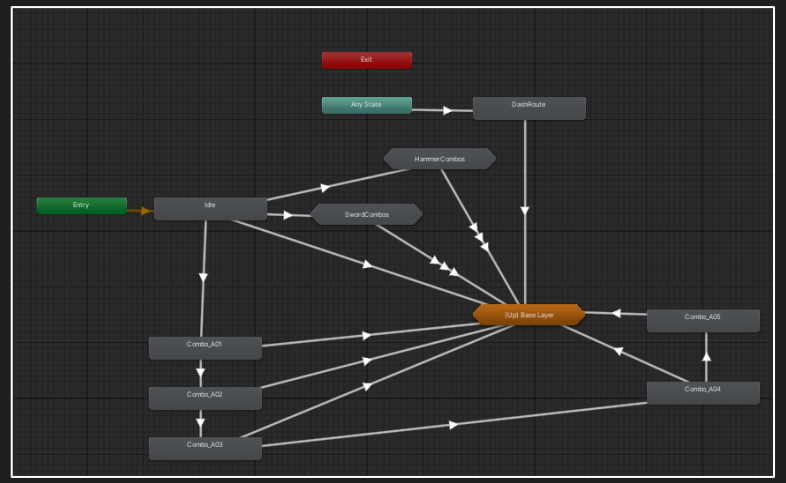
Unity 프로젝트

코드 링크

[게임 구조 : 애니메이션] - AnimationManager

- 특정 키에 반응하여 애니메이션을 재생하는 클래스입니다.

- PlayerAnimation 함수 부분은 플레이어의 입력을 받아 무기를 이용한 공격이나 스킬을 사용할 수 있게 합니다.
  - 연계 스킬에 대한 부분도 구현됨.
    - 코루틴 WaitLink를 실행해 특정 시간 동안만 waitLinkAttack 값이 true 이도록 만들어 연계 스킬이 적절한 타이밍에만 가능하도록 했음.




```
void PlayerAnimation() // 조종하는 플레이어 캐릭터의 애니메이션 관리 -> 입력에 반응
{
    if (Input.GetKeyDown(Punch))
    {
        ani.SetTrigger("Punch");
        ani.SetBool("waitLinkAttack", true);
        StartCoroutine(WaitLink(1f));
    }
    if (ani.GetInteger("Weapon") != 2 &&
            (groundCheck.GetOnGround || !additionalJump) && Input.GetKeyDown(Jump) && !Input.GetKey(DownArrow))
    {
        if (!onGround)
        {
            additionalJump = true;
            if (Ec && Ec.jumpDustEffectPrefab)
            {
                GameObject eff = GameObject.Instantiate(Ec.jumpDustEffectPrefab);
                eff.transform.position = gameObject.transform.position - new Vector3(0,0.5f,0);
            }
        }

        if (ani.GetInteger("Weapon") != 2)
        {
            State = AnimationState.Jump;
            ani.SetTrigger("Jump");
        }


참조 1개
IEnumerator WaitLink(float delay)
{
    yield return new WaitForSeconds(delay);
    ani.SetBool("waitLinkAttack", false);
}
```

```
// Update is called once per frame
void Update()
{
    if(owner && owner.movement.PlayerType)
    {
        isPlayer = true;
    }
    else
    {
        isPlayer = false;
    }
    if (groundCheck)
    {
        if (!groundCheck.owner)
        {
            groundCheck.owner = owner;
        }
        if (groundCheck.GetOnGround)
        {
            if (onGround && State != AnimationState.Normal)
            {
                ani.SetTrigger("Landing");
                State = AnimationState.Normal;
            }

            if (State == AnimationState.Fall)
            {
                onGround = true;
                if (isHuman)
                {
                    ani.ResetTrigger("Jump");
                    ani.SetTrigger("Landing");
                    owner.aManager.ani.ResetTrigger("fall");
                }
            }
        }
        else
        {
            if (onGround)
            {
                onGround = false;
                airDash = false;
                additionalJump = false;
                if (isHuman)
                {
                    ani.ResetTrigger("Landing");
                }
            }
            else
            {
                if (isPlayer)
                {
                    if (Input.GetKeyDown(Punch))
                    {
                        ani.SetTrigger("Punch");
                    }
                    if (Input.GetKeyDown(Dash) && !airDash && !owner.movement.BlockDash)
                    {
                        airDash = true;
                        ani.SetTrigger("Dash");
                    }
                }
            }
        }
    }
    if (isPlayer)
        PlayerAnimation();

    transform.localEulerAngles = new Vector3(transform.localEulerAngles.x, transform.localEulerAngles.y, rotationZ);
}
```

Unity 프로젝트

코드 링크

[게임 구조 : 사격 시스템] - ShootingControl (1)

- 총의 기능을 담당하는 클래스로 마우스 커서의 모양을 바꾸고 총 무기의 사격 및 폭탄 투척 스킬 등을 구현하였습니다.
  - 발사체를 생성하는 기술의 경우엔 모두 해당 클래스를 사용함.

- Update 함수에서 마우스 커서를 바꾸고, 총의 조준 방향 조절 및 총알 발사, 총의 애니메이션 재생 등에 대한 처리 또한 같이 진행합니다.
  - 총은 무한으로 발사할 수 없도록 발사 횟수의 제한을 두어 횟수 초과 시 재장전됨.
  - 총의 조준 방향은 마우스 위치에 따라 달라지며, 총 오브젝트도 마우스에 따라 회전함.

```
void Update()
{
    if(owner.movement.PlayerType)
    if(!owner.isDie)
    if (WhenTargeting) // 플레이어가 총을 무기로 사용하고 있으며 사격 가능한 상태
    {
        Cursor.SetCursor(targetingImage, new Vector2(targetingImage.width/3f, targetingImage.height / 3f), CursorMode.Auto); // 커서를 사격 조준경으로 바꿈.
        AtargetingPos = Camera.main.ScreenToWorldPoint(Input.mousePosition + new Vector3(20, -20, 0)); // 마우스 위치를 변수에 저장함.

        // gun은 총 오브젝트를 의미하며 아래 변수들은 마우스의 방향에 따라 총의 회전값이 변하도록 하였다.
        var dir = Input.mousePosition - Camera.main.WorldToScreenPoint(transform.position);
        var angle = MathF.Atan2(dir.x, dir.y) * Mathf.Rad2Deg;

        gun.transform.rotation = Quaternion.AngleAxis(-angle, Vector3.forward);

        // 총을 사용 가능한 상태에선 gunAnimator라는 애니메이터를 이용해 총의 발사 애니메이션을 보여준다.
        // 아래는 gunAnimator를 가진 게임 오브젝트의 회전 값을 바꾸어 총의 위치가 제대로 위치하도록 하는 코드.
        gunAnimator.transform.localEulerAngles = new Vector3(gunAnimator.transform.localEulerAngles.x, owner.transform.localEulerAngles.y, gunAnimator.transform.localEulerAngles.z);

        if (bulletCount < bulletMaxCount) // 총의 발사 횟수 제한이 있고, 해당 횟수를 넘어서면 일정 시간 동안 재장전이 이뤄진다.
        {
            if (Input.GetKeyUp(KeyCode.Mouse0))
            {
                // 아래는 총을 발사하는 과정을 나타낸 코드이다.

                bulletCount++;
                if (GunGauge.fillAmount > 0.0f)
                {
                    GunGauge.fillAmount -= 0.15f;
                }
                else
                {
                    GunGauge.fillAmount -= 0.17f;
                }
                var obj = Instantiate(bullet, fireTr.position, Quaternion.identity);
                obj.GetComponent<HitCollider>().owner = owner;
                Destroyer d = obj.GetComponent<Destroyer>();

                obj.transform.LookAt(AtargetingPos);
                d.moveSpeed = bulletSpeed;
                d.haveTarget = true;

                if (gunAnimator != null)
                {
                    gunAnimator.SetTrigger("Shoot");
                }
                owner.aManager.ResetAttackTriggerEvent();
            }
            if (bulletCount == bulletMaxCount)
                StartCoroutine(Reload()); // 재장전을 담당하는 코루틴 호출
        }
    }
```

Unity 프로젝트

코드 링크

[게임 구조 : 사격 시스템] - ShootingControl (2)

- Update 함수 이외에는 호출할 수 있는 여러가지 총과 관련된 함수들을 만들었다.
  - SimpleShoot 과 ThrowBomb 모두 마우스가 총의 발사 방향을 결정함.
  - ThrowBomb의 경우 총알이 아닌 폭탄이 생성되도록 했음.

- 총 이외에도 특정 오브젝트를 생성해 발사하는 함수들이 포함되어 있습니다.
  - ThrowStone, ShockWave 등

```
public void SimpleShoot(float speed)
{
    Vector2 len = this.gameObject.GetComponent<Movement>().mousePos - this.gameObject.transform.position;
    float z = Mathf.Atan2(len.y, len.x) * Mathf.Rad2Deg;

    var obj = Instantiate(bullet, GunfireTr.position, Quaternion.Euler(0f, 0f, z));
    obj.GetComponent<HitColider>().owner = owner;
    Destroyer d = obj.GetComponent<Destroyer>();
    d.moveSpeed = speed;
}
참조 0개
public void ThrowBomb()
{
    var obj = Instantiate(bombPrefab, fireTr.position, Quaternion.identity);
    Destroyer d = obj.GetComponent<Destroyer>();
    obj.transform.LookAt(AtargetingPos);
    d.moveSpeed = bulletSpeed;
    d.haveTarget = true;
}
```

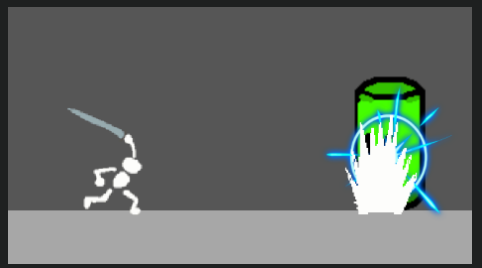
```
public void ThrowStone()
{
    float tempPosX;
    if (owner.transform.rotation.y == 0) tempPosX = 1.5f;
    else tempPosX = -1.5f;
    // 바위를 생성해 앞쪽으로 날린다.
    var stone = Instantiate(StonePrefab, owner.transform.position + new Vector3(tempPosX, 0.3f),
        Quaternion.identity);
    stone.GetComponent<Rigidbody2D>().AddForce(owner.transform.right * 10000);

    // 바위는 플레이어가 맞지 않도록 만든 오브젝트라 별도 처리가 필요 없다.
}

참조 0개
public void ShockWave()
{
    float tempPosX;
    if (owner.transform.rotation.y == 0) tempPosX = 0.5f;
    else tempPosX = -0.5f;

    // 참격을 생성해 앞쪽으로 날린다.
    var shockWave = Instantiate(ShockWavePrefab, owner.transform.position + new Vector3(tempPosX, -0.5f),
        Quaternion.identity);

    // 참격의 경우 플레이어가 맞지 않게 설정함.
    shockWave.GetComponent<HitColider>().owner = owner;
    shockWave.GetComponent<Rigidbody2D>().AddForce(owner.transform.right * 10000);
}
```



Unity 프로젝트

코드 링크

[게임 구조 : 데이터 테이블] - CsvReader

- csv 파일에서 데이터를 읽어오는 클래스입니다.
- 도전 과제 데이터를 표시하는 Achievement 클래스와 말풍선 대사를 표시하는 FancySpeechBubble 클래스에서 호출됩니다.
- 게임 내 텍스트를 csv 파일로 관리하기 위해 제작했습니다.
  - csv 파일로 데이터를 관리하므로 각 데이터를 다른 언어로 쉽게 변경할 수 있음.

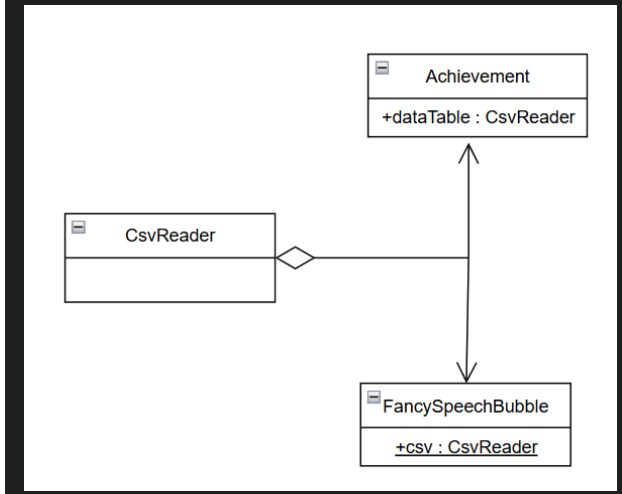
```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.IO;
using UnityEngine;

Unity 스크립트(자산 참조 13개)|참조 4개
public class CsvReader : MonoBehaviour
{
    [Tooltip("csv 파일 경로 Assets 폴더가 시작점")]
    public string filename;
    public List<List<string>> lines = new List<List<string>>();

    Unity 메시지|참조 0개
    private void Awake()
    {
        if(filename == "DialogueData.csv")
            FancySpeechBubble.csv = this;
        StreamReader sr = new StreamReader(Path.Combine(Application.streamingAssetsPath, filename));
        bool eof = false;

        while (!eof) // 끝 도달까지 계속 진행
        {
            string data = sr.ReadLine();
            if(data == null)
            {
                eof = true;
                break;
            }
            var values = data.Split(',');
            List<string> valueList = new List<string>();
            for (int i = 0; i < values.Length; i++)
            {
                valueList.Add(values[i].ToString());
            }
            lines.Add(valueList);

        }
    }
}
```



	A	
1	Dialogue	Dialogue (en)
2	가서 무기	Go! steal all the WEAPONS!!!
3	수리 중인	All the equipment being repaired is GOOOOOOOONE~~
4	내 장비가	My equipment is gone...
5	난 이제 끝	I'm finished...
6	내 모든 마	All my magical tools have disappeared;;
7	상점가가	It seems the market was completely robbed by thieves!
8	오랜만이다	It's been a while... APO
9	뭐여!?	What!?
10	가게 꼬라	Why does my store look like this?
11	하하.. 내	Check out my 3 weapons!!
12	말도 안돼!	What the..!?
13	나의 최강	My strongest 3 weapons are gone...
14	숲과 사막	Pass the forest and the desert

	A	B	C	D	E	F	G
1	M_Code	Name (kr)	Name (en	Descriptio	Description (en)		
2	1	놀라운 컨	Surprising	체력 절반	Knock down the armed boss		
3	2	폭력은 나	Violence i	평화주의	Attack to pacifist slime!		
4	3	우주에서	Hammer t	망치 필살	Use ultimate skill with hamm		
5	4	굉장한 에	Miracle Er	총 필살기	Use ultimate skill with gun.		
6	5	서서히 가	Slowly dro	모래 늪에	Try to die in sand swamp.		
7	6	다 부숴버	Break it al	가게 안의	Break the box in the shop.		
8	7	불쌍한 친	Poor my fi	마지막 보	if you lose when the final bo		
9	8	땅이 없어.	Ground is	낙사	death from a fall		
10	9	집요한 까	Tenacious	까마귀 쓰	Knock down the crow.		
11	10	진정한 소	True Swor	총알 튕겨	You hit the bullet with the sw		

Unity 프로젝트

- **성과** : 기획서의 요구사항을 모두 구현하였으며 최종적으로 프로젝트를 무사히 완성하여 학과 교수 심사 통과 및 졸업 프로젝트 최종 발표회에서 장려상을 수상하였습니다.
  - 또한, 재사용 가능한 게임 프레임워크를 구현하여 이후의 게임 제작에도 코드 일부를 이용할 수 있을 것으로 기대합니다.