

**헨젤런**

**VR 기말 프로젝트**



|  |  |  |
| --- | --- | --- |
| **팀명** | **학번** | **이름** |
| **달리조** | **20181833** | **강성관** |
| **20170727** | **김남하** |
| **20202296** | **전채린** |
| **담당교수** | | **박형수** |

**목차**

1. **개발 배경**
2. **개발 목적**
3. **시나리오**
4. **게임 설계**
5. **UI 및 감독스크립트**
6. **아이템과 장애물 프리팹 및 제너레이터 스크립트**
7. **플랫폼 및 기타 오브젝트**
8. **캐릭터 컨트롤러**
9. **게임 플레이**
10. **조작법**
11. **게임 진행**
12. **게임 오버**
13. **스크립트 설명**
    1. **PlayerController**
    2. **GameManager**
    3. **ItemGenerator**
14. **실행결과 및 게임 화면**
15. **역할분담**
    1. **강성관**
    2. **김남하**
    3. **전채린**
16. **개발 배경**

코로나19 사태 이후 게임 시장이 크게 성장하게 되는데, 그중에서도 특히 집에서 간편히 즐길 수 있는 PC와 모바일 게임 관련 분야가 크게 성장했다. 세계 게임 시장은 보다 더 큰 폭으로 성장했는데, 그중에서도 특히 모바일 게임은 코로나 사태 전에 비해 약 25%, PC와 콘솔 게임은 약 21% 성장했다. 또한, 스마트폰과 개인 PC, 게임기의 대중 보급이 활성화되면서 게임을 처음 접하는 연령층이 낮아졌다. 특히 저 연령 아동의 스마트폰 보유율은 2010년 14%에서 2020년 29.5%로 크게 급등했고, 자연스레 게임을 접할 기회가 많아진 아이들은 게임을 하는 시간 또한 늘게 되었다. 한 설문조사에 따르면, 아이들이 즐겨 하는 게임은 자유도가 높은 게임, 달리거나 점프를 하는 게임 등 다양한 요구조건이 있지만, 그중 아이들이 가장 원하는 게임의 특징은 조작법이 간단하고 흥미로운 스토리라인이 있는 게임이라는 결과가 나왔다. 따라서 저희는 조작법이 간단하고 아이들의 흥미를 유발하는 ‘헨젤과 그레텔’을 모티브로 2d 횡스크롤 게임인 ‘헨젤런’을 개발하게 되었습니다.

1. **개발 목적**

높은 점수를 얻거나 경쟁하는 등 플레이어가 게임을 즐기는 방법은 다양합니다. ‘헨젤런’은 게임 점수를 높이거나 경쟁을 즐기는 높은 연령층의 플레이어뿐 아니라 단순히 게임 하는 것 자체를 즐기는 낮은 연령층의 플레이어도 쉽게 조작하고 몰입하도록 게임을 기획하였습니다. 다양한 타겟층을 확보하고자 몰입도를 높이는 데 중점을 두었고, 조작법이 간단한 러닝 플랫폼 게임 시리즈를 기획하게 되었습니다. 또한, 스토리 라인은 아이들이 즐겨 읽는 ‘헨젤과 그레텔이’라는 동화책 속 스토리라인을 가져와 아이들이 보다 흥미를 느끼고, 게임에 대한 이해와 몰입도를 높이는 데 집중하였습니다.

1. **시나리오(게임 스토리)**
   1. 스토리의 주요 진행은 기본적으로 동화책 ‘헨젤과 그레텔’ 속의 스토리를 바탕으로 한다.
   2. 헨젤과 그레텔 스토리

헨젤과 그레텔은 매우 가난한 집에서 아버지와 계모와 함께 살고 있었다. 헨젤과 그레텔의 부모님은 자신의 형편으로 아이들을 키우기 힘들다고 생각해 헨젤과 그레텔을 깊은 숲속에 남겨두고 오자는 계획을 세운다. 영리했던 헨젤은 그 대화 내용을 엿듣게 되고, 전날 작은 돌을 주워서 바지 주머니 속에 모아둔다. 다음날 헨젤과 그레텔의 부모님은 아이들에게 소풍을 가자고 하고 깊은 숲속으로 들어가는데, 이때 헨젤은 전날 모아둔 돌을 하나씩 떨어뜨려 집으로 돌아오는 길을 표시하면서 간다. 헨젤과 그레텔의 부모님은 아이들이 잠시 한눈을 판 사이 아이들을 두고 집으로 돌아가 버리고, 여동생과 둘이 남겨진 헨젤은 떨어뜨린 돌을 따라 집으로 돌아오게 된다. 돌아온 헨젤과 그레텔을 본 부모님은 다음엔 더 깊은 숲속으로 가서 아이들을 두고 오자는 계획을 세우게 된다. 헨젤은 그 대화 내용을 또 엿듣게 되고, 이번엔 저녁으로 받은 빵을 잘게 부숴 주머니 속에 모아둔다. 다음날 또 헨젤과 그레텔의 부모님은 아이들을 데리고 더 깊은 숲속으로 가게 되는데, 이때 헨젤은 미리 잘게 부숴둔 빵을 하나씩 흘려 돌아가는 길을 표시해둔다. 밤이 되고 아버지와 계모는 아이들이 한눈을 판 사이 집으로 돌아가 버리고, 아이들은 깊은 숲속에 남게 된다. 헨젤은 미리 남겨둔 빵 조각을 찾기 위해 숲속을 돌아보지만 이미 그 빵 조각은 지나가는 동물들이 주워 먹게 되고 헨젤과 그레텔은 집으로 돌아갈 수 없게 된다. 한참을 방황하던 중 헨젤과 그레텔은 과자로 만든 집을 발견하게 되는데, 배가 너무 고팠던 헨젤과 그레텔은 과자의 집에 들어가 그 집을 먹게 된다. 하지만 그 집은 마녀의 집이었고 아이들을 본 마녀는 아이들을 살을 찌워 잡아 먹어버리겠다고 생각하고, 아이들을 가둬 둔 뒤 과자를 먹이게 된다. 며칠이 지나 마녀가 헨젤을 먼저 먹으려 하는 순간 그레텔이 기지를 발휘해 마녀를 밀어낸 뒤, 헨젤과 함께 도망치게 되면서 게임이 시작된다.

1. **게임 설계**
   1. Unity 2D core이용 화면 구성
   2. 에셋스토어에서 백그라운드, 캐릭터, 체력바, 다양한 아이템, 사운드 에셋을 다운로드
   3. UI 및 감독스크립트
      1. UI(Score, HeartBar, Gameover)를 Scene에 배치한다.
      2. UI(Score, HeartBar, Gameover)를 갱신하는 감독 스크립트를 작성한다.
      3. 빈 오브젝트를 생성하고 감독 스크립트를 적용한다.
   4. 아이템과 장애물 프리팹 및 제너레이터 스크립트
      1. 아이템과 장애물을 Scene에 배치하고 Collider를 씌운다.
      2. 아이템과 장애물을 프리팹화 한다.
      3. 제너레이터 스크립트 작성한다.
      4. 빈 오브젝트를 생성하고 제너레이터 스크립트를 적용한다.
      5. 제너레이터 스크립트에 프리팹(아이템, 장애물)을 전달한다.
   5. 플랫폼 및 기타 오브젝트
      1. 플랫폼 프리팹을 가져온다.
      2. 플랫폼에 Collider를 씌운다.
      3. 빈 오브젝트를 생성하고, 플랫폼 제네레이터 스크립트를 작성하고 적용한다.
      4. 플랫폼 제네레이터 스크립트에 프리팹을 전달한다.
      5. 배경에 각 요소들을 배치한 후 이미지화 한다.
      6. 배경을 배치하고 Collider를 씌운 후 Trigger설정을 완료한다.
      7. 배경을 무한 회전시킬 수 있는 Background Loop 스크립트를 작성한다.
      8. 각 배경에 Background Loop, Scrolling Object 스크립트를 적용한다.
   6. 캐릭터 컨트롤러
      1. Main Character에 collider와 rigidbody를 씌운다.
      2. Main character 애니메이션(Run, Jump, Hurt, Dead)을 제작한다.
      3. 상황에 맞게 작동되는 트리거를 제작한다.
      4. 상황에 맞게 순환하는 애니메이터를 제작한다.
      5. 플레이어 컨트롤러 스크립트를 작성한다.
      6. 플레이어의 상황에 맞게 트리거를 발생시켜 애니메이션을 실행한다.
2. **게임 플레이**
   1. 조작법
3. 스페이스바로 점프를 한다.
   1. 게임 진행
4. 체력 게이지는 프레임 당 0.0002%씩 줄어든다.
5. 플레이어가 달리면서 쿠키를 먹게 되면 100 스코어가 쌓이게 된다.
6. 플레이어가 달리면서 파란색 포션을 먹게 되면 체력 게이지의 5%를 회복한다.
7. 플레이어가 달리면서 보라색 포션에 부딪치면 체력 게이지의 5%를 감소한다.
   1. 게임 오버
8. 플레이어가 플랫폼 밑으로 떨어지게 되면 게임오버가 된다.
9. 플레이어의 체력 게이지가 바닥나게 되면 플레이어가 쓰러지면서 게임오버가 된다.
10. 게임 오버 시 점수 및 재시작 버튼이 나온다.
11. **스크립트 설명**
    1. PlayerController

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class PlayerController : MonoBehaviour

{

    public AudioClip deathClip; // 사망시 재생할 오디오 클립

    public AudioClip hurtClip; // 독극물 복용 시 재생할 오디오 클립

    public AudioClip jumpClip; // 다시 복원할 오디오 클립(점프)

    public AudioClip runClip; // 처음 시작할 때 재생할 오디오 클립

    public AudioClip itemClip; // 아이템(사탕,과자)을 먹었을 때 재생할 오디오 클립

    public AudioClip healClip; // 회복 아이템 복용 시 재생할 오디오 클립

    private AudioSource playerAudio;

    Rigidbody2D rigid2D;

    Animator animator;

    float jumpForce = 430.0f;

    private int jumpCount = 0;

    private bool isGrounded = false; // 땅에 닿았는지 나타내는 상태변수

    private bool isDead = false; // 사망 상태

    // Start is called before the first frame update

    void Start()

    {

        this.rigid2D = GetComponent<Rigidbody2D>();

        this.animator = GetComponent<Animator>();

        this.playerAudio = GetComponent<AudioSource>();

        PlaySound("RUN");

    }

    // Update is called once per frame

    void Update()

    {

        // 사용자 입력을 감지하고 점프하는 처리

        if (isDead)

        {

            return;

        }

        // 점프한다.

        if (Input.GetKeyDown(KeyCode.Space) && jumpCount < 2)

        {

            jumpCount++;

            this.animator.SetTrigger("JumpTrigger");

            this.rigid2D.AddForce(transform.up \* this.jumpForce);

            PlaySound("JUMP");

        }

        animator.SetBool("Grounded", isGrounded);

        // 체력게이지가 0과 동일해 지거나 낮아질 경우 데드트리거 발동

        if (0 >= GameDirector.instance.health)

        {

            this.animator.SetTrigger("DeadTrigger");

            Die();

        }

    }

    private void Die()

    {

        // 사망 처리

        // 애니메이터의 Die 트리거 파라미터를 셋

        this.animator.SetTrigger("DeadTrigger");

        PlaySound("DEAD");

        // 속도를 제로로 변경

        rigid2D.velocity = Vector2.zero;

        // 사망 상태를 true로 변경

        isDead = true;

        // 게임 매니저의 게임오버 처리 실행

        GameDirector.instance.OnPlayerDead();

    }

    private void OnTriggerEnter2D(Collider2D other)

    {

        // 파란색 포션을 먹었을 경우 체력을 회복하는 처리

        if (other.gameObject.tag.Equals("healPotion"))

        {

            GameDirector.instance.AddHP(0.05f);

            PlaySound("HEAL");

            other.gameObject.SetActive(false);

        }

        // 장애물에 부딪힐 경우 Hurt Animation 발동

        if (other.gameObject.tag.Equals("poisonPotion"))

        {

            this.animator.SetTrigger("HurtTrigger");

            PlaySound("HURT");

            GameDirector.instance.DamagedHP(0.05f);

            other.gameObject.SetActive(false);

        }

        // 디저트를 먹었을 경우 점수를 추가하는 처리

        if (other.gameObject.tag.Equals("Item"))

        {

            GameDirector.instance.AddScore(100);

            PlaySound("ITEM");

            other.gameObject.SetActive(false);

        }

        // 사망 오브젝트 충돌 시 사망

        if (other.gameObject.tag.Equals("DeathObject") || other.gameObject.tag.Equals("DeathObjectSide"))

        {

            this.animator.SetTrigger("DeadTrigger");

            Die();

        }

    }

    private void OnCollisionEnter2D(Collision2D collision)

    {

        //바닥에 닿았을 때 점프 초기화를 위한 메소드

        if(collision.contacts[0].normal.y > 0.7f)

        {

            isGrounded = true;

            jumpCount = 0;

        }

    }

    private void OnCollisionExit2D(Collision2D collision)

    {

        // 바닥에서 벗어났음을 감지하는 처리

        isGrounded = false;

    }

    void PlaySound(string action)

    {

        switch(action)

        {

            case "JUMP":

                playerAudio.clip = jumpClip;

                break;

            case "HURT":

                playerAudio.clip = hurtClip;

                break;

            case "DEAD":

                playerAudio.clip = deathClip;

                break;

            case "RUN":

                playerAudio.clip = runClip;

                break;

            case "ITEM":

                playerAudio.clip = itemClip;

                break;

            case "HEAL":

                playerAudio.clip = healClip;

                break;

        }

        playerAudio.Play();

    }

}

1. GameManager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

*// 게임 오버 상태를 표현하고, 게임 점수와 UI를 관리하는 게임 매니저*

*// 씬에는 단 하나의 게임 매니저만 존재할 수 있다.*

public class GameManager : MonoBehaviour {

    public static GameManager instance; *// 싱글톤을 할당할 전역 변수*

    public bool isGameover = false; *// 게임 오버 상태*

    public Text scoreText; *// 점수를 출력할 UI 텍스트*

    public Text healthText; *// 체력을 출력할 UI 텍스트*

    public GameObject healthBar; *// 체력바를 표시할 UI 게임 오브젝트*

    public GameObject gameoverUI; *// 게임 오버시 활성화 할 UI 게임 오브젝트*

    private int score = 0; *// 게임 점수*

    private float health = 100.0f; *// 플레이어 체력*

*// 게임 시작과 동시에 싱글톤을 구성*

    void Awake() {

*// 싱글톤 변수 instance가 비어있는가?*

*if* (instance == null)

        {

*// instance가 비어있다면(null) 그곳에 자기 자신을 할당*

            instance = this;

        }

*else*

        {

*// instance에 이미 다른 GameManager 오브젝트가 할당되어 있는 경우*

*// 씬에 두개 이상의 GameManager 오브젝트가 존재한다는 의미.*

*// 싱글톤 오브젝트는 하나만 존재해야 하므로 자신의 게임 오브젝트를 파괴*

            Debug.LogWarning("씬에 두개 이상의 게임 매니저가 존재합니다!");

            Destroy(gameObject);

        }

    }

    void Start() {

        this.healthBar = GameObject.Find("Health");

        this.gameoverUI = GameObject.Find("gameoverUI");

    }

*// Update is called once per frame*

    void Update() {

        DecreaseHealth();

    }

*// 점수를 증가시키는 메서드*

    public void AddScore(int newScore) {

*// 게임 오버가 아니라면*

*if* (!isGameover)

        {

*// 점수를 증가*

            score += newScore;

            scoreText.text = "Score : " + score.ToString();

        }

    }

*//  체력을 증가시키는 메서드*

    public void AddHP(float newHP) {

*// 게임 오버가 아니라면*

*if* (!isGameover && health <= 100.0f)

        {

*if*(health > 100.0f)

                health = 100.0f;

*else*

*// 체력을 증가*

                health += newHP;

*// 현재 체력을 체력바에 표시*

            this.healthBar.GetComponent<Image>().fillAmount += health;

*// 텍스트로도 표시 하기 위한 UI text*

            this.healthText.text = string.Format("HP {0}/100", health);

        }

    }

*// 부상 시 체력을 감소시키는 메서드*

    public void DamagedHP(float newHP) {

*// 게임 오버가 아니라면*

*if* (!isGameover && health > 0.0f)

        {

*// 체력을 감소*

            health -= newHP;

*// 현재 체력을 체력바에 표시*

            this.healthBar.GetComponent<Image>().fillAmount -= health;

*// 텍스트로도 표시 하기 위한 UI text*

            this.healthText.text = string.Format("HP {0}/100", health);

        }

*else* {

*// 체력이 0이 되면 플레이어의 죽음 함수를 호출*

            OnPlayerDead();

        }

    }

*// 점진적으로 체력을 감소시키는 메서드*

    public void DecreaseHealth()

    {

*if*(!isGameover && health > 0.0f) {

*// 체력 감소*

            health -= 0.02f;

*// 현재 체력을 체력바에 표시*

            this.healthBar.GetComponent<Image>().fillAmount -= 0.0002f;

;

*// 텍스트로도 표시 하기 위한 UI text*

            this.healthText.text = string.Format("HP {0}/100", health);

        }

*else* {

*// 체력이 0이 되면 플레이어의 죽음 함수를 호출*

            OnPlayerDead();

        }

    }

*// 게임 오버시 게임을 재시작 하는 메서드*

    public void Restart()

    {

*// 게임 오버 상태에서 gameoverUI 버튼을 클릭하면 현재 씬 재시작*

        SceneManager.LoadScene(SceneManager.GetActiveScene().name);

    }

*// 플레이어 캐릭터가 사망시 게임 오버를 실행하는 메서드*

    public void OnPlayerDead() {

*// 현재 상태를 게임 오버 상태로 변경*

        isGameover = true;

*// 게임 오버 UI를 활성화*

        gameoverUI.SetActive(true);

    }

}

1. Destroyer

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Destroyer : MonoBehaviour

{

    float posX = -100.0f;

    void Update()

    {

        if (transform.position.x < posX)

        {

            Destroy(gameObject);

            posX = posX + 10.0f;

        }

    }

}

1. ScrollingObjects

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

// 게임 오브젝트를 계속 왼쪽으로 움직이는 스크립트

public class ScrollingObject : MonoBehaviour {

    public float speed = 10f; // 이동 속도

    // Update is called once per frame

    private void Update() {

        // 게임 오버가 아니라면

        if (!GameDirector.instance.isGameover)

        {

            // 초당 speed의 속도로 왼쪽으로 평행 이동

            transform.Translate(Vector3.left \* speed \* Time.deltaTime);

        }

    }

}

1. PlatformSpawner

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

// 발판을 생성하고 주기적으로 재배치하는 스크립트

public class PlatformSpawner : MonoBehaviour {

    public GameObject platformPrefab; // 생성할 발판의 원본 프리팹

    public int count = 3; // 생성할 발판의 개수

    public float timeBetSpawnMin = 8f; // 다음 배치까지의 시간 간격 최솟값

    public float timeBetSpawnMax = 9f; // 다음 배치까지의 시간 간격 최댓값

    private float timeBetSpawn; // 다음 배치까지의 시간 간격

    public float yMin = -3.5f; // 배치할 위치의 최소 y값

    public float yMax = 1.5f; // 배치할 위치의 최대 y값

    private float xPos = 33f; // 배치할 위치의 x 값

    private GameObject[] platforms; // 미리 생성한 발판들

    private int currentIndex = 0; // 사용할 현재 순번의 발판

    // 초반에 생성된 발판들을 화면 밖에 숨겨둘 위치

    private Vector2 poolPosition = new Vector2(0, -25);

    private float lastSpawnTime; // 마지막 배치 시점

    // Start is called before the first frame update

    // 변수들을 초기화하고 사용할 발판들을 미리 생성

    void Start() {

        // count 만큼의 공간을 가지는 새로운 발판 배열 생성

        platforms = new GameObject[count];

        // count 만큼 루프하면서 발판을 생성

        for (int i = 0; i < count; i++)

        {

            // platformPrefab을 원본으로 새 발판을 poolPosition 위치에 복제 생성

            // 생성된 발판을 platform 배열에 할당

            platforms[i] = Instantiate(platformPrefab, poolPosition, Quaternion.identity);

        }

        // 마지막 배치 시점 초기화

        lastSpawnTime = 0f;

        // 다음번 배치 까지의 시간 간격을 0으로 초기화

        timeBetSpawn = 0f;

    }

    // Update is called once per frame

    void Update() {

        // 게임 오버 상태에서는 동작하지 않는다.

        if (GameDirector.instance.isGameover)

        {

            return;

        }

        // 마지막 배치 시점에서 timeBetSpawn 이상 시간이 흘렀다면

        if (Time.time >= lastSpawnTime + timeBetSpawn)

        {

            // 기록된 마지막 배치 시점을 현재 시점으로 갱신

            lastSpawnTime = Time.time;

            // 다음 배치까지의 시간 간격을 timeBetSpawnMin, timeBetSpawnMax 사이에서 랜덤 설정

            timeBetSpawn = Random.Range(timeBetSpawnMin, timeBetSpawnMax);

            // 배치할 위치의 높이를 yMin과 yMax 사이에서 랜덤 설정

            float yPos = Random.Range(yMin, yMax);

            // 발판 버그 방지

            platforms[currentIndex].SetActive(false);

            platforms[currentIndex].SetActive(true);

            // 현재 순번의 발판을 화면 오른쪽으로 재배치

            platforms[currentIndex].transform.position = new Vector2(xPos, yPos);

            // 순번 넘기기

            currentIndex++;

            // 마지막 순번에 도달했다면 순번을 리셋

            if (currentIndex >= count)

            {

                currentIndex = 0;

            }

        }

    }

}

1. BackgroundLoop

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

// 왼쪽 끝으로 이동한 배경을 오른쪽 끝으로 재배치하는 스크립트

public class BackgroundLoop : MonoBehaviour {

    private MeshRenderer render;

    private float offset;

    public float speed;

    void Start()

    {

        render = GetComponent<MeshRenderer>();

    }

    void Update()

    {

        if(!GameDirector.instance.isGameover)

        {

            offset += Time.deltaTime \* speed;

            render.material.mainTextureOffset = new Vector2(offset, 0);

        }

    }

}

1. PoisonSpawner

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PoisonSpawner : MonoBehaviour

{

    public GameObject poisonPrefab; // 생성할 독극물의 원본 프리팹

    float span = 4.0f;

    float delta = 0;

    // Update is called once per frame

    void Update()

    {

        // 게임 오버 상태에서는 동작하지 않는다.

        if (GameDirector.instance.isGameover)

        {

            return;

        }

        this.delta += Time.deltaTime;

        if (this.delta > this.span)

        {

            this.delta = 0;

            GameObject po = Instantiate(poisonPrefab);

            int px = Random.Range(7, 15);

            int py = Random.Range(-2, 3);

            po.transform.position = new Vector3(px, py, 0);

        }

    }

}

1. ScoreSpawner

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ScoreSpawner : MonoBehaviour

{

    public GameObject scorePrefab; // 생성할 독극물의 원본 프리팹

    float span = 4.0f;

    float delta = 0;

    // Update is called once per frame

    void Update()

    {

        // 게임 오버 상태에서는 동작하지 않는다.

        if (GameDirector.instance.isGameover)

        {

            return;

        }

        this.delta += Time.deltaTime;

        if (this.delta > this.span)

        {

            this.delta = 0;

            GameObject po = Instantiate(scorePrefab);

            int px = Random.Range(7, 15);

            int py = Random.Range(-2, 3);

            po.transform.position = new Vector3(px, py, 0);

        }

    }

}

1. ItemGenerator

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

*/\* 아이템을 생성하는 제너레이터 스크립트 \*/*

public class ItemGenerator : MonoBehaviour

{

    public GameObject[] candyPrefabs;

    float span = 3.0f;

    float delta = 0;

    void Start()

    {

        candyPrefabs = GameObject.FindGameObjectsWithTag("Item");

    }

*// Update is called once per frame*

    void Update()

    {

*// 3초마다 아이템 생성*

        this.delta += Time.deltaTime;

*if* (this.delta > this.span) {

            this.delta = 0;

*// 0부터 25 사이의 수를 랜덤하게 생성*

            int i = Random.Range(0, 26);

*// i번째 아이템 생성*

            GameObject item = Instantiate(candyPrefabs[i]) as GameObject;

*// 아이템의 X 좌표는 0부터 6 사이의 수를 랜덤하게 생성*

            int x = Random.Range(0, 7);

*// 아이템의 Y 좌표는 0부터 2 사이의 수를 랜덤하게 생성*

            int y = Random.Range(0, 3);

            item.transform.position = new Vector3(x, y, 0);

        }

    }

}

1. **실행 결과 및 게임 화면**
   1. 유니티 실행 결과

텍스트, 모니터, 전자기기, 화면이(가) 표시된 사진

자동 생성된 설명

* 1. 헨젤런 게임 화면

1) 게임을 시작하면 플레이어가 앞으로 달려나간다

텍스트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

2) 케이크, 쿠키, 사탕, 초콜릿 등 아이템 오브젝트를 먹으면 점수가 100씩 누적된다.

텍스트, 나무, 모니터, 화면이(가) 표시된 사진

자동 생성된 설명

3) 파란포션을 먹으면 체력게이지가 5% 증가하고 보라색 포션을 먹으면 체력게이지가 5% 감소한다.

텍스트, 모니터, 화면, 옅은이(가) 표시된 사진

자동 생성된 설명

4) 플랫폼 밖으로 떨어지거나 체력게이지가 바닥 나게 될 경우 게임이 종료되고 재시작 버튼이 나온다

텍스트, 모니터, 화면, 어두운이(가) 표시된 사진

자동 생성된 설명

1. **역할분담**
   1. 강성관
      1. Platform 및 Object
         1. 플랫폼 및 장애물 생성하는 제너레이터
         2. 카메라 컨트롤러
         3. 배경 조작 스크립트
         4. 게임 개발 PPT 작성
   2. 김남하
      1. Character Controller
         1. Player 조작 스크립트
         2. 플레이어 애니메이션
         3. 게임 개발 보고서 작성
   3. 전채린
      1. UI 및 Scene Manager
         1. 게임 매니저 스크립트
         2. 아이템 관련 제너레이터
         3. 게임 시연 및 발표 담당