실전
게임 기초 AI 프로그래밍

예제로 쉽게 배우는 게임 인공지능 프로그래밍

2조: 박소영, 이재현, 임형택, 조창희

3장 | 프로덕션 시스템
4장 | 배경과 AI

프로덕션 시스템 | Production System

인공지능에서 자주 사용되는 지식표현 방법으로 주어진 조건에 대한
권고, 지시, 전략 등을 나타내는 정형화된 지식 표현 방법

이 책 3장에서 4가지 주제에 대해서 다루고 있지만 결론적으로 3가지라고 생각한다.

- **자동 유한 상태 기계** AFSM, Automated finite-state machine
  - **확률 계산**
- **유틸리티 기반 함수**
- **유동적인 게임 AI 밸런스 조절**

하나하나 간단하게 살펴보도록 하자.

자동 유한 상태 기계 | AFSM, Automated finite-state machine

인공지능 캐릭터가 현 상황에서 여러 요소(위치, 캐릭터의 체력, 현재 무기 등)를
고려하고 계산하여 최선의 행동을 선택하도록 하는 방법

2장에서는 주어진 상태에 따라 상황 별 행동을 하게 표를 작성하였다.
자동 유한 상태 기계는 어느 상황이 주어지든 계산하고 행동할 수 있으며
간단하게 설명하자면

기본 목표 설정 ➡ 목표 달성을 위한 행동 구성 ➡ 상황에 따라 행동 실행

위와 같은 순서로 진행된다. 가능기반 표와 많이 유사하다.
하지만 표 작성 방법과 두는 차이점은 어느 위치에 있든 행동하는 **'기본 목표 설정'**에 있다.

확률 계산

바로 전 슬라이드에서 **자동 유한 상태 기계**를 가능기반표에 빗대어 설명했다.
여기서는 Boolean을 이용하여 캐릭터의 주 목적을 정의하고 그것을 활용하여
AI 캐릭터가 스스로 선택할 수 있는 지능적인 캐릭터를 개발할 수 있다.

결국 **확률 계산**은 확률기반 표의 연장선이며,
결론적으론 자동 유한 상태 기계는 가능·확률기반 표를 작성하는 것에 그치지 않고
실행할 수 있게 시스템화 되었다고 볼 수 있다.

유틸리티 기반 함수

자동 유한 상태 기계를 이용하여 목표에 대한 행동을 취할 수 있게 AI 캐릭터를 개발할 수 있게 되었다.
그렇지만 이 방법 또한 **기계(Machine)**란 뜻을 달고 있다는 것은 자연스러운 행동을 유발할 수 없다는 것이다.

그렇기 때문에 더욱 **세세하게 설정**하여 단순히 목표 달성이 아닌 무엇을 우선으로 두고 선택하고 실행하여 목표를 달성해야 하는지, 혹은 목표를 포기하고 다른 목표를 먼저 처리해야 할지 스스로 정할 만큼 자세한 프로그래밍이 동반해야 한다.

즉, **어떤 선택이 AI 스스로에게 유용한지 알아서 판단할 수 있도록 함수를 작성**해야 한다.

유동적인 게임 AI 밸런스 조절

다이나믹 게임 AI 밸런스 Dynamic Game AI Balance 라고도 책에서 표현하고 있다.

게임의 난이도가 일정하다면 난이도의 고저는 온전히 플레이어의 게임 숙련도에 달려 있다.
그렇기 때문에 **각 게이머의 숙련도에 맞춰 난이도 문제를 해결**해야 한다.
일반적으로는 여러 항목의 수치(체력, 파워 등)를 조절하여 난이도를 수정한다.
또한 게임의 장르에 따라 공격 횟수가 줄어들거나, 둔한 반응을 보이게 한다든지 말이다.

이렇게 게임의 난이도가 유동적으로 조절될 수 있지만, 그 **게임만의 특성을 유지**하기 위해
난이도를 조절하지 않고 어렵게 유지하는 방식 등으로 개성을 살릴 수 있다.

결론

그래서 자동 유한 상태 기계로 AI 캐릭터가 **게임 어느 곳이든 스스로 행동**을 하게 하고,
스스로 **확률을 계산**하여 더 좋은 판단을 하고, **유틸리티 기반의 함수**까지 추가되어
더욱 **자연스러운 행동**을 위해 **AI가 스스로 유용한지 판단**하게 끔 하는 방법을 배워보았다.
또한, 마지막으로 **난이도를 유동적으로 조절**하는 법도 배웠다.

Unity에서도 스크립트를 Sprite에 적용하면 작동하고, 또한 그것을 Generator 시켜
스크립트를 적용해주면 같은 캐릭터를 공장화 시킬 수(찍어낼 수) 있다.

그렇기 때문에 귀찮고 오래 걸릴 작업이라고 생각하기보다, 한번 정성을 담아 제작하면
그 뒤로는 큰 틀이 정해지고, 복사도 가능하기 때문에 어렵게만 생각하지 않아도 될 것 같다.

앞에서 봤듯이 게임의 AI를 구현할 때 가장 중요한 요소는 위치다.
4장에서는 배경의 여러 상호작용에 대해서 알아볼 수 있다.

시각적 상호작용

게임 플레이에 직접적인 영향을 끼치지 않는 상호작용이다.
하지만, 게임 몰입에 있어 큰 중요성을 가지고 있다.

기본 배경 상호작용

상호작용을 게임 플레이에 적용하여 게이머의 행동에 영향을 끼치는 상호작용이다.
단순히 시각적으로 배경으로만 치는 것이 아니라 주위 환경을 인식하고 상호작용하게 만들어 게임의 목적을 달성하는 방법이 게임 속에서 중요한 부분으로 자리잡았다.

기본 배경 상호작용

이 책에서는 기본 배경 상호작용에서 3가지 방법을 살펴볼 수 있다.

- **배경 오브젝트 움직이기**
- **방해하는 배경 오브젝트**
- **배경을 여러 지역으로 세분화하기**

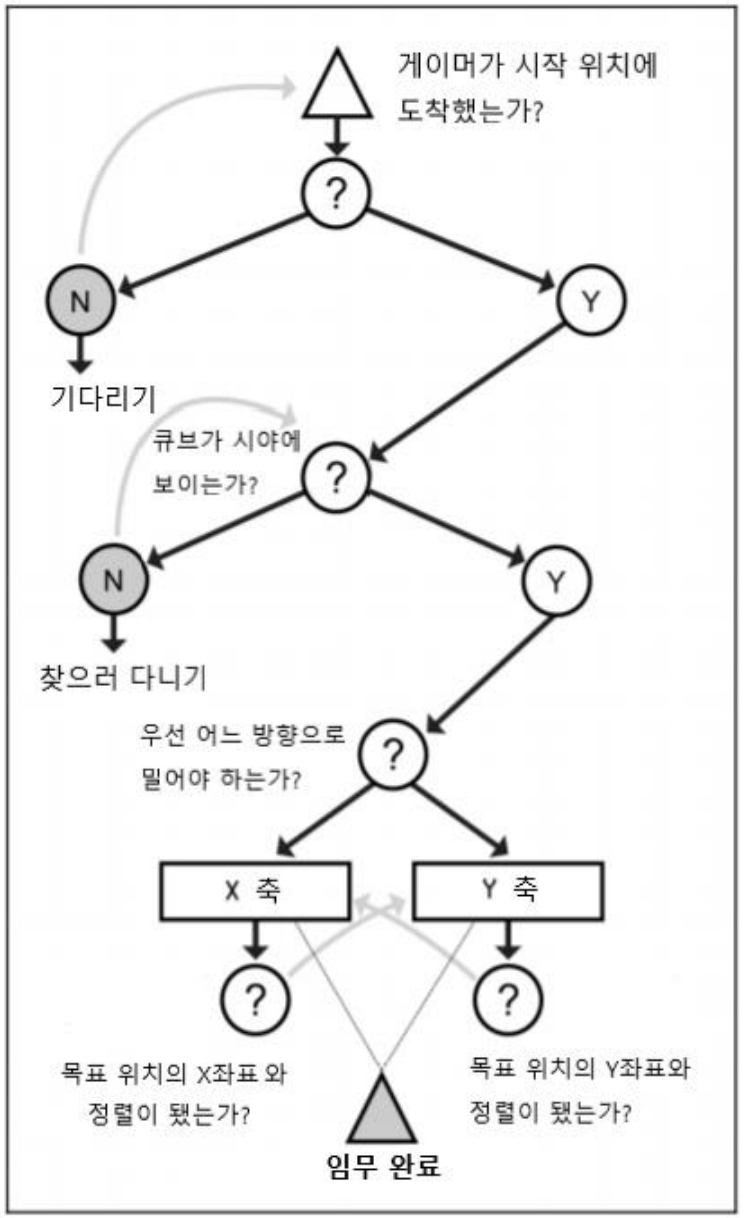
책에 있는 모든 내용을 적용한 것은 아니지만
기본적으로 책의 내용을 따라가는 형식으로 예제를 만들어가면서 학습하였다.

배경 오브젝트 움직이기

배경은 게임의 중요한 일부분이며 배경 오브젝트는
게임 플레이에 직접적인 영향을 끼칠 수 있다.

AI 캐릭터는 플레이어에게 도움을 주기 위해
(혹은 방해) 스스로 오브젝트와 상호작용 할 수 있다.

교재에서는 오른쪽 그림과 같이 AI의 행동을 대략적으로
나타내었다.

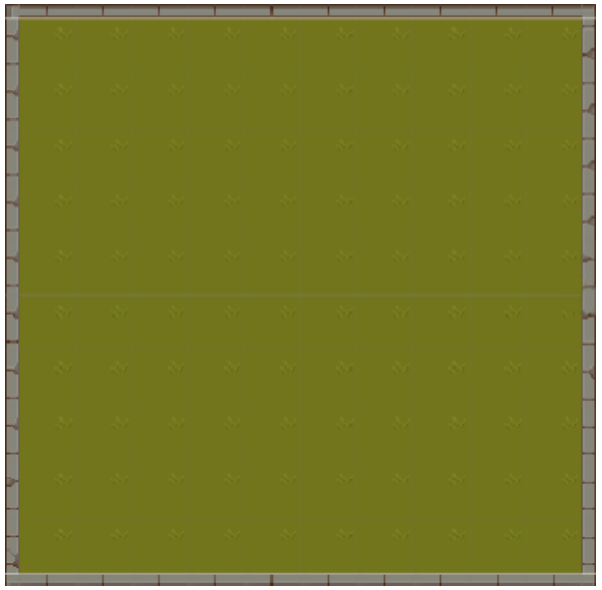


맵 구현하기

시작에 앞서 학습을 위해 어떤 구성으로 진행할 것인지 결정하는 시간을 가졌다.

맵은 **2D** 형식으로 결정하였고,
에셋 스토어의 **Pixel Art Top Down – Basic** 에셋을 이용하였다.

(타일의 장점인 맵을 직접 구성할 수 있는 점을 활용하였다.)



10X10 타일 맵

플레이어

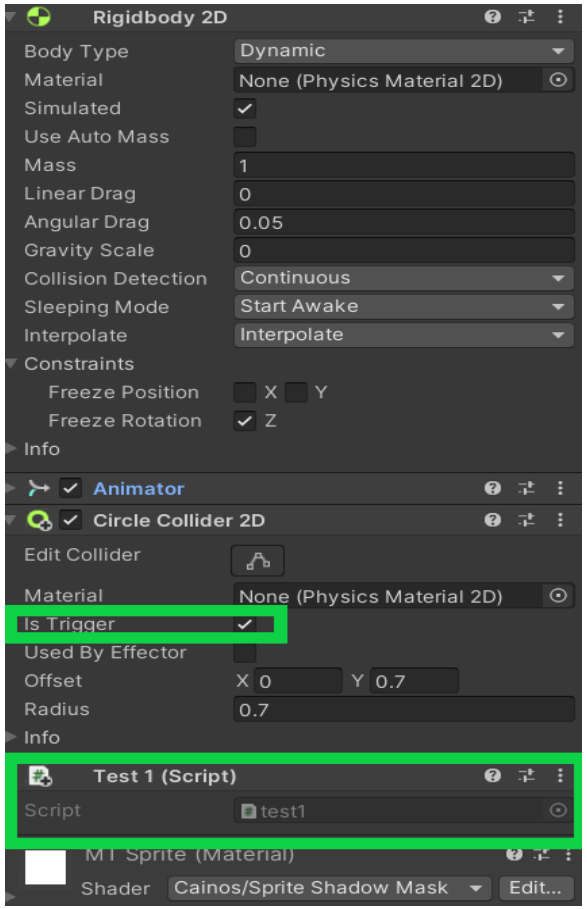
AI

배경 오브젝트 움직이기

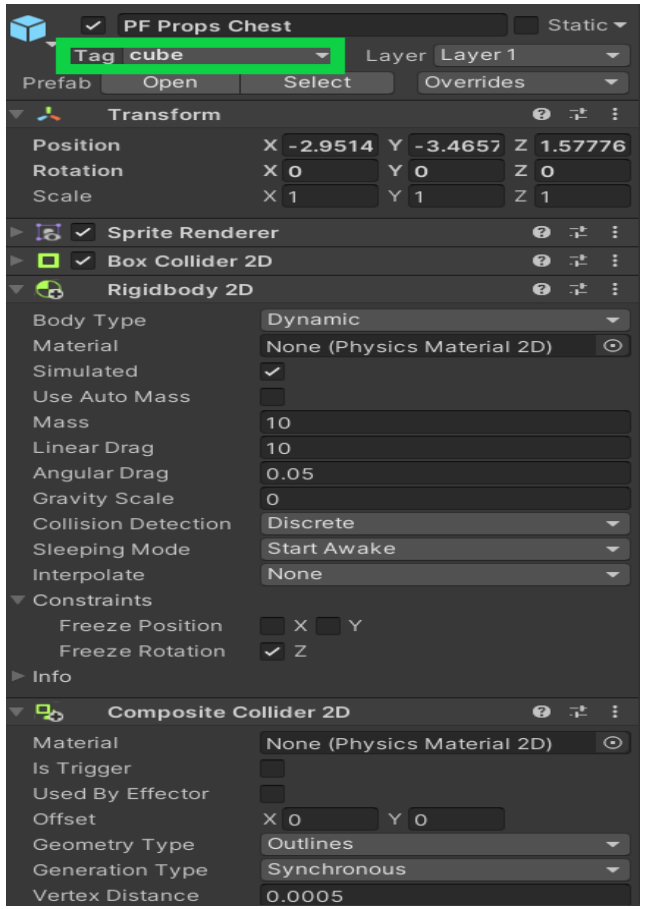
AI 캐릭터가 플레이어를 도와주게 구현해보자.

도움 AI 영상

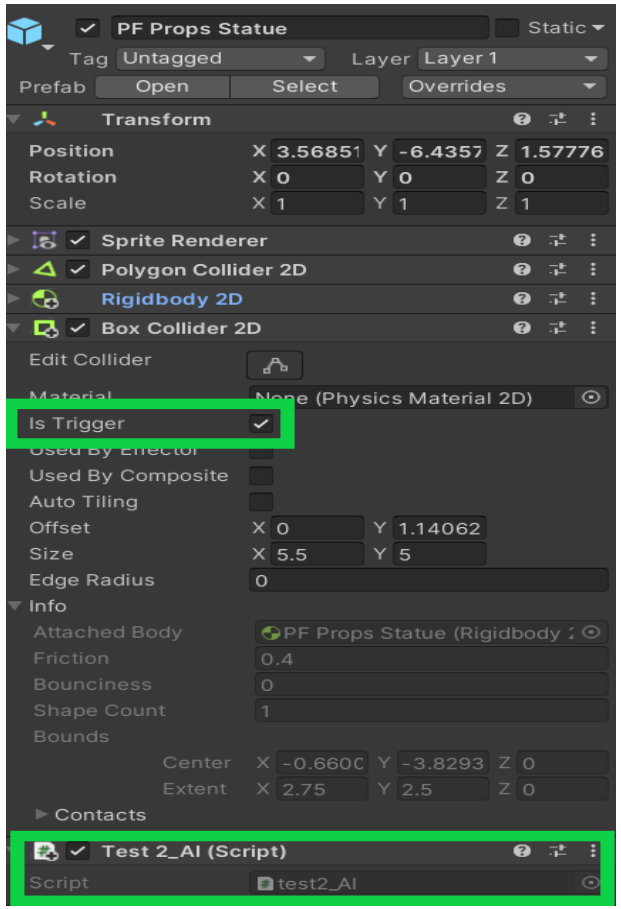
배경 오브젝트 움직이기



플레이어(Player) 설정



상자(cube) 설정



AI(AI) 설정

배경 오브젝트 움직이기

```
private int startCount = 0;
private Transform cube;
private Vector3 cubePosition;
private Transform cubeMark;
private Vector3 cubeMarkPosition;
private Vector3 calPosition;
private List<float> boxArray = new List<float>();
private float c_near = 0;
private float c_nearvalue = 0;
private List<GameObject> boxobj;
private bool state;
private GameObject test;
private Vector3 testposi;


void Start() {
    cube = GameObject.FindWithTag("cube").transform;
    cubePosition = cube.position;
    cubeMark = GameObject.FindWithTag("cubeMark").transform;
    cubeMarkPosition = cubeMark.position;
    test = GameObject.FindWithTag("cubeMark");
    testposi = test.GetComponent<BoxCollider2D>().bounds.center;
    // boxobj= new List<GameObject>(GameObject.FindGameObjectsWithTag("Box"));
    // GetComponent<PolygonCollider2D>()
    // GetComponent<PolygonCollider2D>().bounds.center;
}

void Update() {
    boxobj= new List<GameObject>(GameObject.FindGameObjectsWithTag("Box"));
    cubePosition = cube.position;
    if (startCount < 3) {
        transform.position = Vector3.MoveTowards(transform.position, cubePosition, Time.deltaTime)
    }
}
```

```
void path() {
    boxArray.Clear();
    state = false;
    foreach (GameObject box in boxobj) {
        if (box.transform.position.y - cubePosition.y > -1) state = true;
        if (box.transform.position.y > cubePosition.y && box.transform.position.y - cubePosition.y < 1){
            boxArray.Add(box.transform.position.x);
        }
    }
    if (state) {
        if (boxArray.Count == 0) {
            cube.Translate(Vector2.up * 0.5f);
            transform.Translate(Vector2.down * 0.2f);
        } else {
            c_near = Mathf.Abs(cubePosition.x - boxArray[0]);
            c_nearvalue = boxArray[0];
            for (int i=1; i<boxArray.Count; i++) {
                if (Mathf.Abs(cubePosition.x - boxArray[i]) < c_near) {
                c_near = Mathf.Abs(cubePosition.x - boxArray[i]);
                c_nearvalue = boxArray[i];
                }
            }
            if (c_near > 1.2f) {
                cube.Translate(Vector2.up * 0.5f);
            } else {
                if (cubePosition.x - c_nearvalue > 0.1) {
                    cube.Translate(Vector2.right);
                    transform.Translate(Vector2.left * 0.2f);
                    transform.Translate(Vector2.down * 0.2f);
                }
                if (cubePosition.x - c_nearvalue < 0.1) {
                    cube.Translate(Vector2.left);
                    transform.Translate(Vector2.down * 0.2f);
                    transform.Translate(Vector2.right * 0.2f);

                }
            }
        }
```

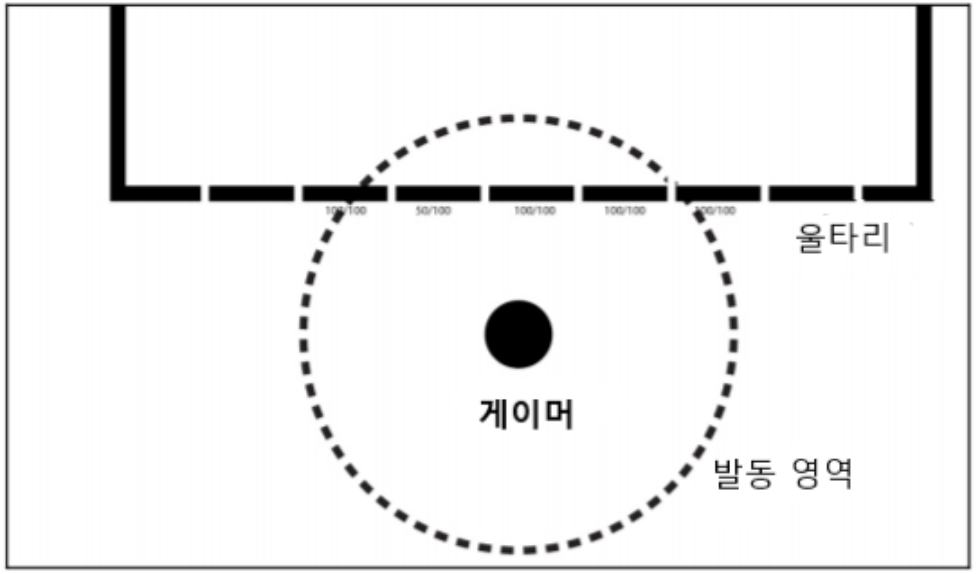
배경 오브젝트 움직이기

```
    } else {
            // cube.transform.position = Vector3.MoveTowards(cube.transform.position, cubeMarkPosition, 0.2f);
            cube.transform.position = Vector3.MoveTowards(cube.transform.position, testposi, 0.2f);
            calPosition = cubeMarkPosition - cube.transform.position;
            calPosition.Normalize();
            transform.Translate(-calPosition * 0.3f);

            // transform.position = -Vector3.MoveTowards(transform.position, cubeMarkPosition, 0.2f);
    }
}

void OnCollisionStay2D(Collision2D other) {
    if (other.gameObject.tag == "cube") {
        // Invoke("path", 0.5f);
        path();
    }
}

void OnTriggerEnter2D(Collider2D other) {
    if (other.gameObject.tag == "startBox") {
        startCount++;
    }
}

void OnTriggerExit2D(Collider2D other) {
    if (other.gameObject.tag == "startBox") {
        startCount--;
    }
}
```

거리를 계산해 이동하고 박스를 미는 AI 스크립트(test2_AI)

방해하는 배경 오브젝트

오브젝트를 이용하거나 움직여서 게임의 목적을 달성할 수도 있지만,
반대로 오브젝트가 캐릭터의 길을 가로막을 수 있다.
도착을 방해하는 울타리를 예로 들면, AI는 울타리의 한 부분을 부숴야 한다.
이때, 울타리와 캐릭터 사이의 거리와 울타리의 체력을 고려해야 한다.

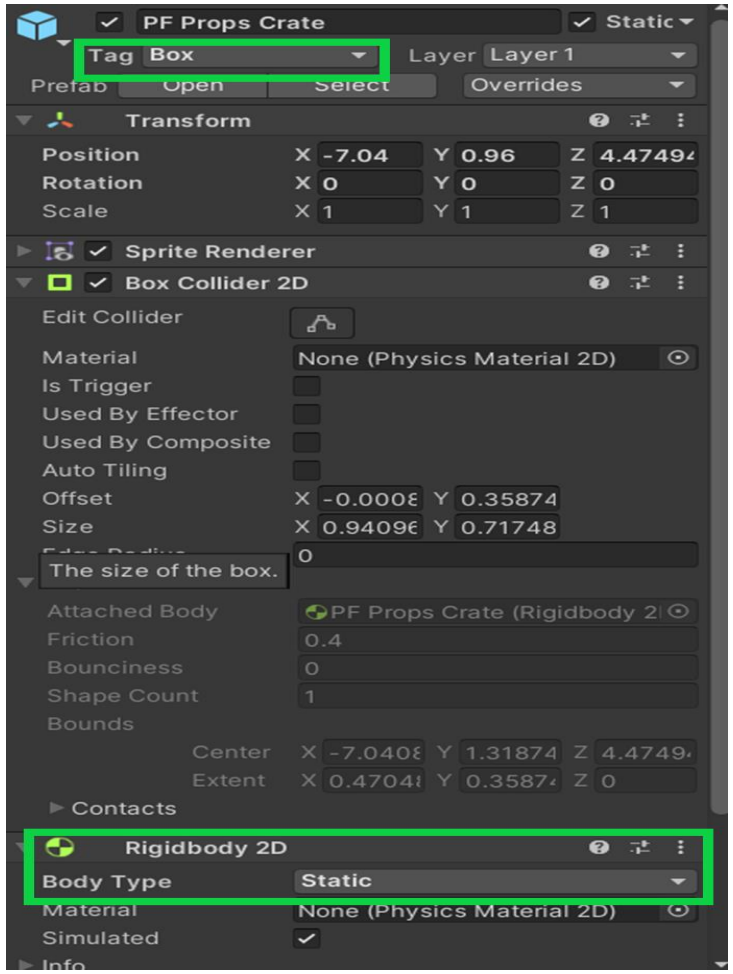


방해하는 배경 오브젝트

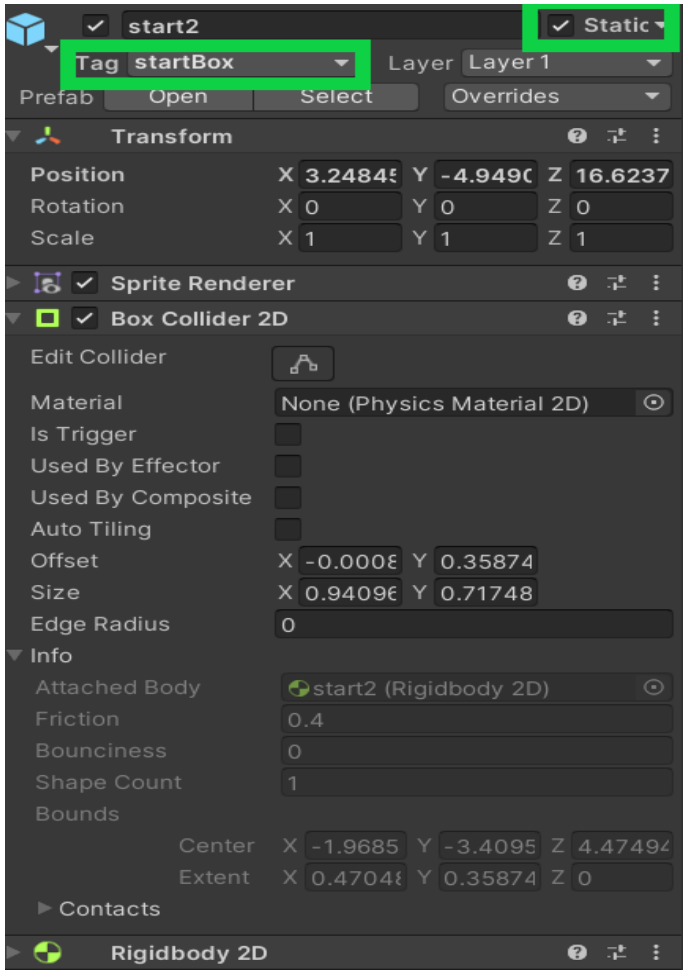
방해 오브젝트를 이용하여 캐릭터의 목적을 방해하는 경우를 해결하도록 구현해보자.
발동 영역을 지정하여 구현해 보았다.

방해하는 오브젝트 영상

방해하는 배경 오브젝트



방해물(Box) 설정



방해물(startBox) 설정

배경을 여러 지역으로 세분화 하기

바다, 사막, 동굴 등 AI 캐릭터가 다양한 환경에 활용되려면
여러 지역을 인식할 수 있게 만들어야 한다.
이는 캐릭터가 현재 있는 장소에서 어떻게 행동해야 하고 다른 장소로 어떻게 이동해야 하는지 등의 **여러 정보를 캐릭터에 입력**해야 한다는 것을 의미한다.

배경을 여러 지역으로 세분화 하기

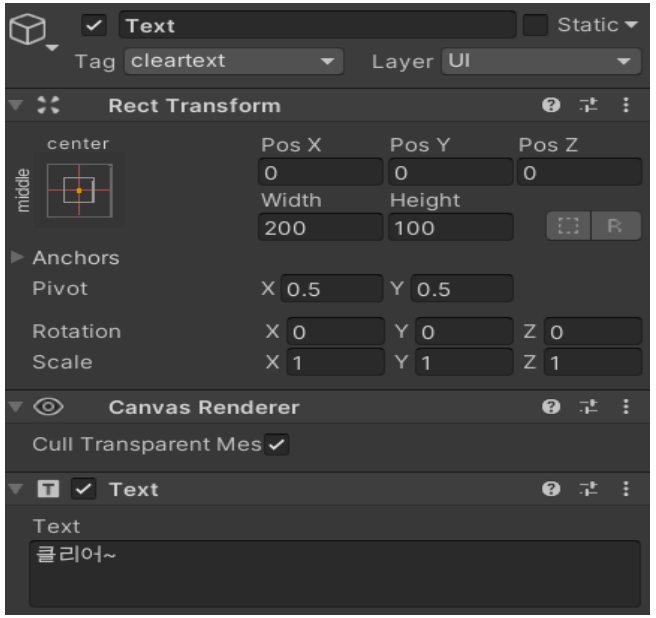
우리는 **A맵**과 **B맵**, **C맵** 세 가지로 **포탈을 통해** 다음 스테이지로 갈 수 있도록 구성하였다.

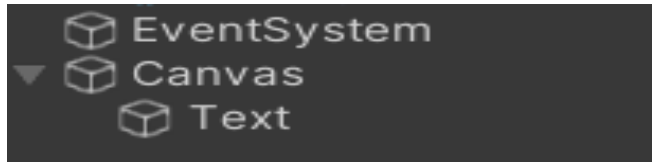
배경을 여러 지역으로 세분화 하기

통합 설정 구현하기

```
public class test_claer : MonoBehaviour
{
    private GameObject portal;
    private GameObject textui;
    void Start() {
        portal = GameObject.FindWithTag("portal");
        textui = GameObject.FindWithTag("cleartext");
        portal.SetActive(false);
        textui.SetActive(false);
    }

    void OnTriggerEnter2D(Collider2D other) {
        if (other.gameObject.tag == "cube") {
            Destroy(other.gameObject);
            portal.SetActive(true);
            textui.SetActive(true);
        }
    }
}
```

박스가 위치에 도달하면
포탈이 나타나는 스크립트(test_clear)



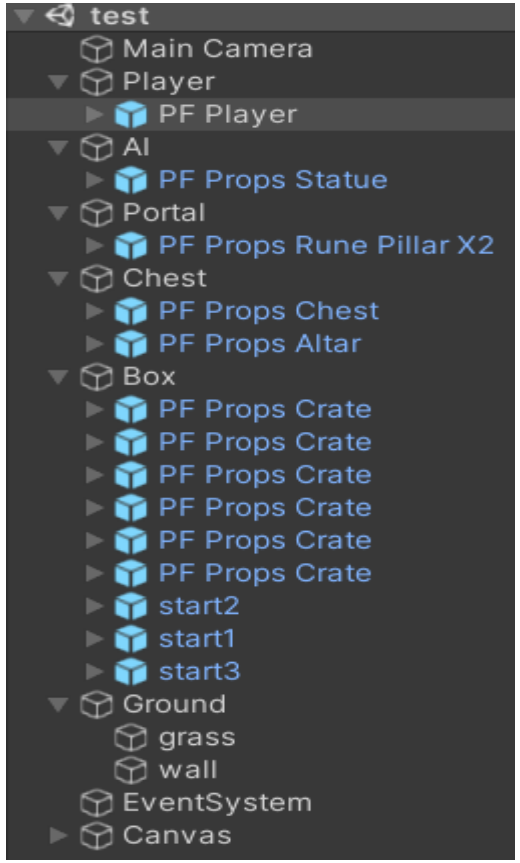


텍스트 UI 구현

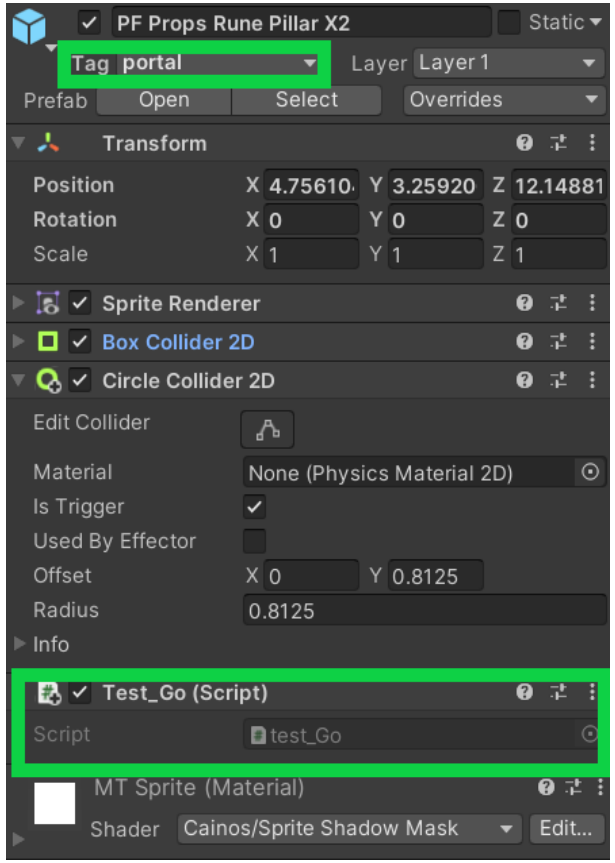
```
// 카메라는 매 프레임마다 업데이트 되어야하기 때문에 여기서
void Update()
{
    if (target.gameObject != null)
    {
        //this는 생략이 가능하고 카메라를 의미한다. 카메라의 z값은 타겟보다 멀리있어야 타겟이 화면에 나올 수 있다.
        targetPosition.Set(target.transform.position.x, target.transform.position.y, this.transform.position.z);
        //Time.deltaTime은 1초에 실행되는 프레임의 역수이며 1초에 moveSpeed만큼 이동하게 해준다.
        //카메라의 위치를 변화시킨다.
        this.transform.position = Vector3.Lerp(this.transform.position, targetPosition, moveSpeed * Time.deltaTime);
    }
```

카메라 이동 스크립트(CameraManager)

배경을 여러 지역으로 세분화 하기 – A맵



A맵(test) 제작



포탈(portal) 설정

```
void OnCollisionEnter2D(Collision2D other) {
    if (other.gameObject.tag == "Box" || other.gameObject.tag == "startBox" ) {
        Destroy(other.gameObject);
    }
```

Player가 장애물을 부수는 스크립트 (test)추가

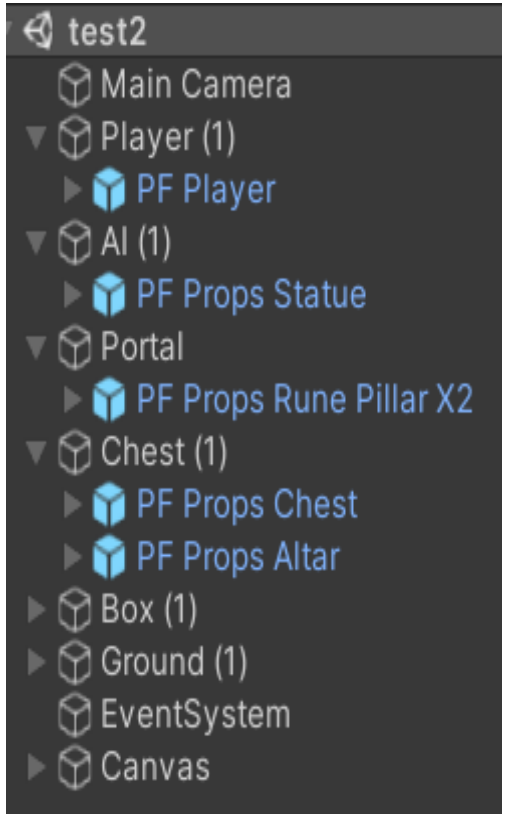
```
void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag.Equals("portal"))
        SceneManager.LoadScene("test");

    else
    {
        SceneManager.LoadScene("test2");
    }
}
```

포탈에 맵 이동 1 스크립트(test_go)추가

배경을 여러 지역으로 세분화 하기 – A맵

A맵 구현 영상

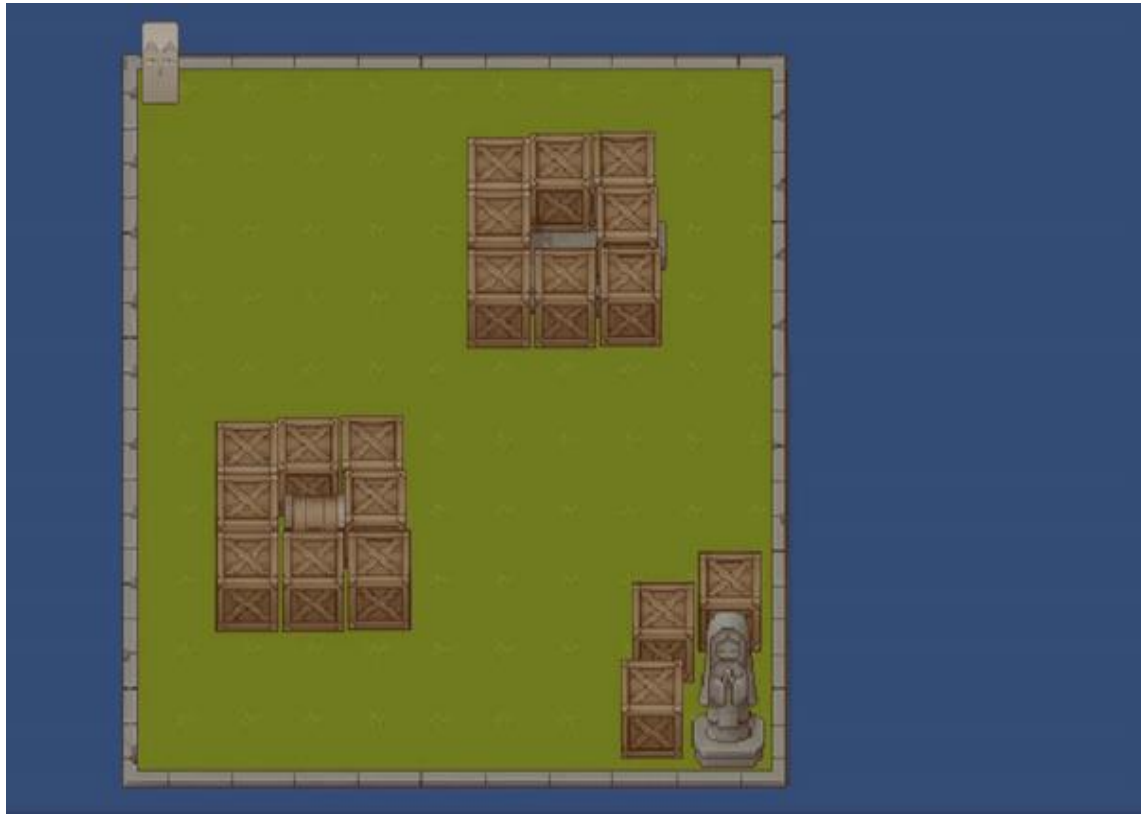
배경을 여러 지역으로 세분화 하기 – B맵



B맵(test2) 제작

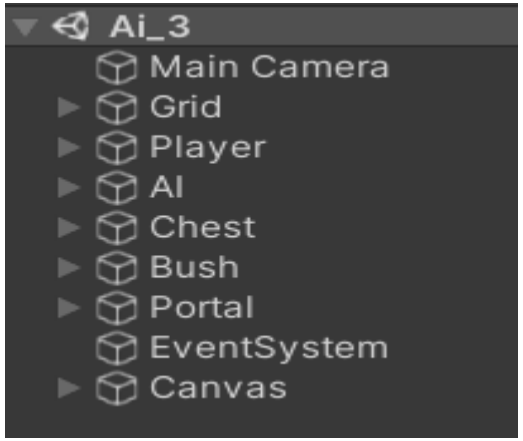
```
void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag.Equals("portal"))
        SceneManager.LoadScene("test2");

    else
    {
        SceneManager.LoadScene("Ai_3");
    }
}
```

포탈에 맵 이동 스크립트
(test_Go2) 추가

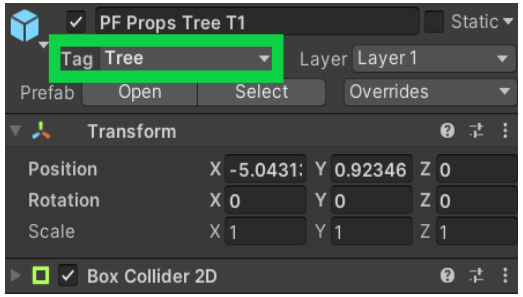


B맵 구현 영상

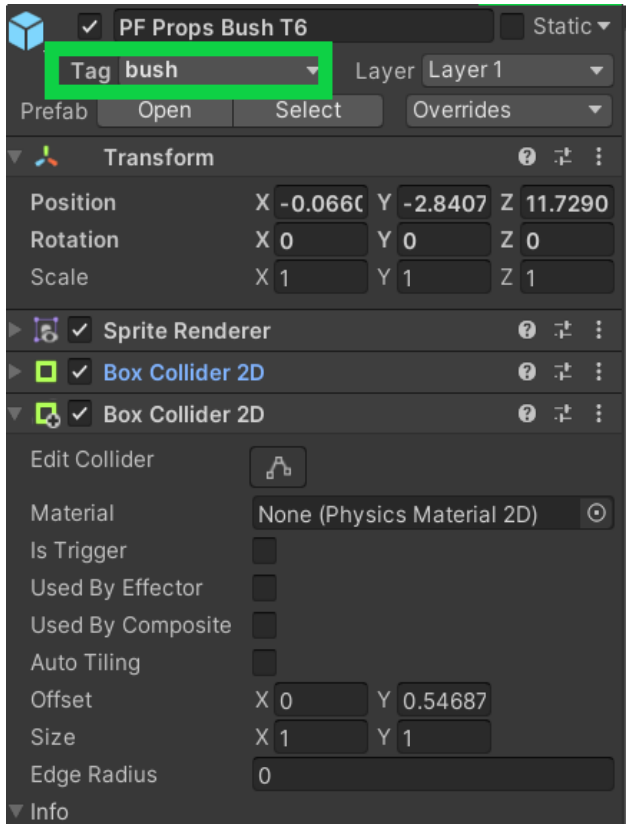
배경을 여러 지역으로 세분화 하기 – C맵



C맵(Ai_3) 제작



나무(Tree) 설정



풀숲(bush) 설정

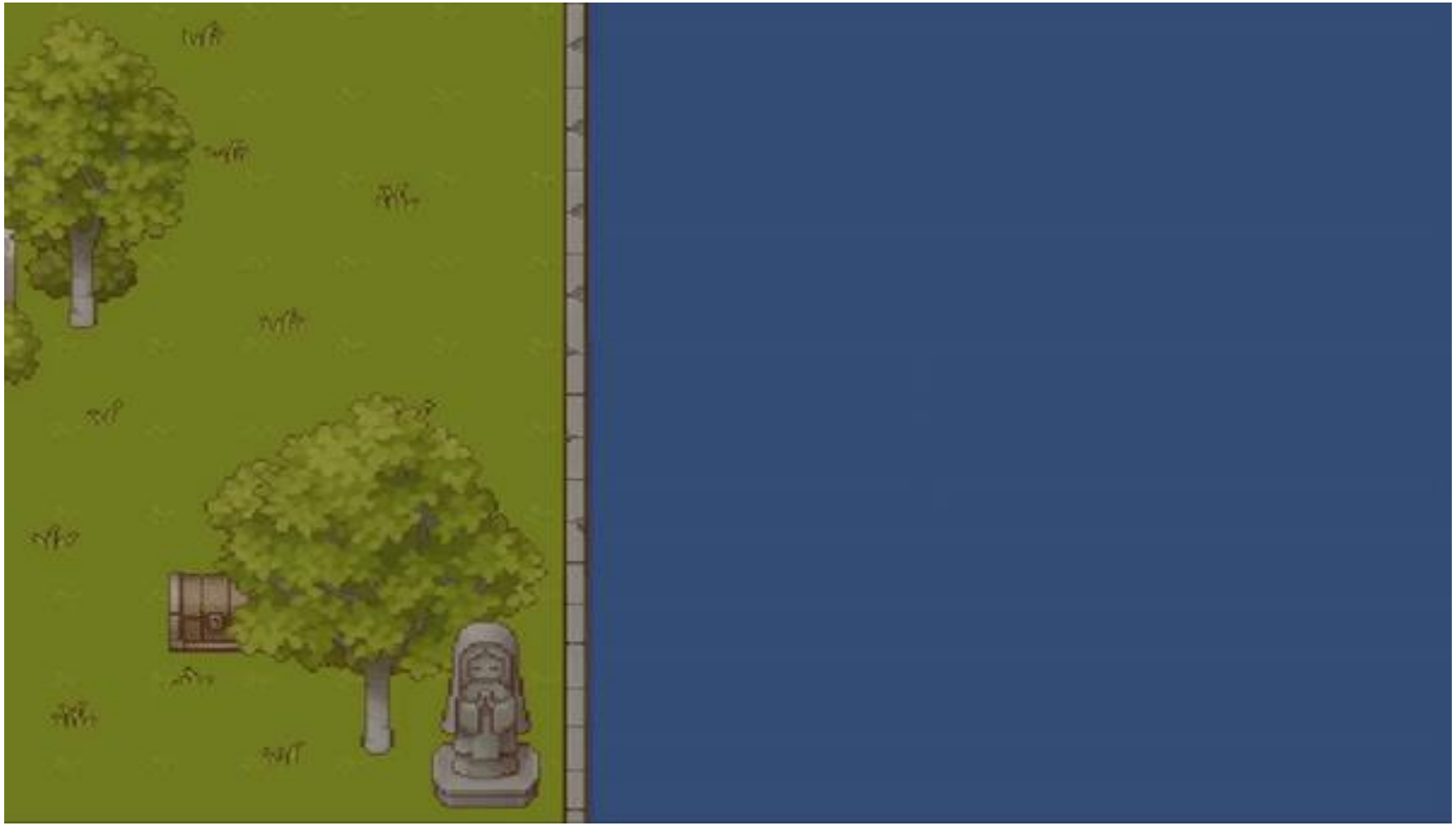
```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.tag.Equals("bush"))
    {
        Destroy(other.gameObject);
        //적을 파괴합니다.
    }
}
```

AI에 bush 스크립트 추가

```
void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag.Equals("portal"))
        SceneManager.LoadScene("Ai_3");

    else
    {
        SceneManager.LoadScene("test");
    }
}
```

포탈에 맵 이동 3 스크립트(Go3)추가

배경을 여러 지역으로 세분화 하기 – C맵



C맵 구현 영상

결론

배경의 중요성을 알게 되었고, 배경 오브젝트 **상호작용**을 통해 다양한 기술을 여러 장르의 게임에 적용할 수 있게 되었다.

캐릭터와 배경 사이의 기본적인 상호작용뿐만 아니라 다양한 방법의 상호작용을 살펴볼 수 있었고, 특히 기본적인 플레이어의 조종으로 인한 상호작용 뿐만 아니라 **AI캐릭터가 배경 오브젝트와 상호작용을 통해 새롭고 특별한 방법으로 게임을 구성할 수 있는 방법을 배워보았다.**