2D Game Basic

키입력과 충돌처리(1)

박성준

(sjpark.jesus@gmail.com)

학습내용

- 1.키보드 처리
- 2.충돌 판정
- 3.따라오기
- 4.충돌하면 반전
- 5.충돌하면 게임정지
- 6.충돌하면 무언가 표시

키보드 처리 구현 원리

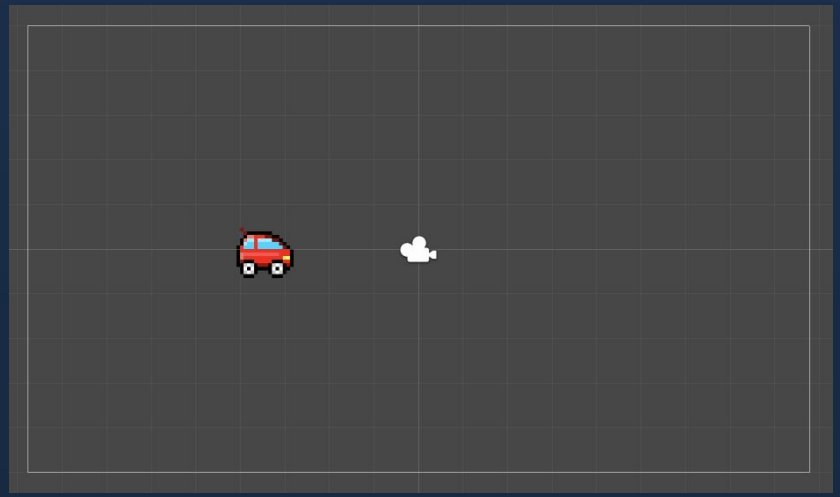
- Update()에서 키 값을 받고 FixedUpdate()에서 실제 이동하는 로직을 구성하는 이유는 불규칙적으로 호출되는 Update()로 가서 키 입력을 확인하여 이동량을 저장해 놓았다가 일정시간마다 호출되는 FixedUpdate()함수에서 실제 이동을 처리함으로써 입력 실수가 적고 매끄러운 움직임을 구현 할 수 있기 때문이다.

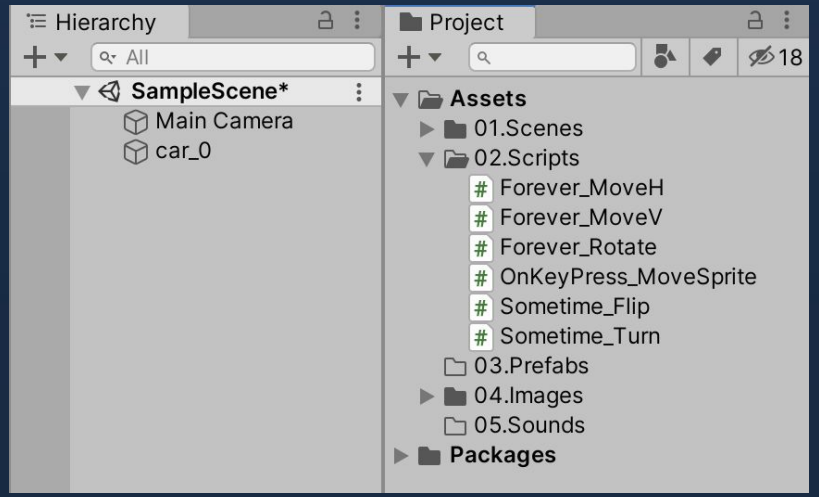
- 코드 구현

```
float vx = 0;
void Update() {   // 키가 눌린것을 계속 조사한다.
    vx = 0;
    if(Input.GetKey(“right”)) {
        vx = 1;   // -> 값을 저장
        leftFlag = false;  // 자동차의 움직이는 방향을 반전
    }
void FixedUpdate() {     // 규칙적으로 호출되는 함수 - 실제 움직임을 구현
    this.transform.Translate(vx/50, 0, 0);
}
```

1. 키보드 처리

- 기존에 만든 프로젝트를 다시 연다 (2DGame)
- 이미지 파일 : Car_0 (이미지 파일을 씬 뷰에 올려 놓는다)
- 스크립트 파일을 하나 만든다 : OnKeyPress_MoveSprite





키보드 처리

- 전체 스크립트 작성 (1)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OnKeyPress_MoveSprite : MonoBehaviour
{
    public  float speed = 2; // 속도
    private float vx = 0;
    private float vy = 0;
    private bool  leftFlag = false;

    void Update() // 매 프레임 마다 호출 (계속 수행한다)
    {
        vx = 0;
        vy = 0;

        if(Input.GetKey("right")) // 만약 오른쪽 화살표 키가 눌려 졌다면
        {
            vx = speed;
            leftFlag = false;
        }

        if(Input.GetKey("left")) // 만약 왼쪽 화살표 키가 눌려 졌다면
        {
            vx = -speed;
            leftFlag = true;
        }
```

키보드 처리

- 전체 스크립트 작성 (2)

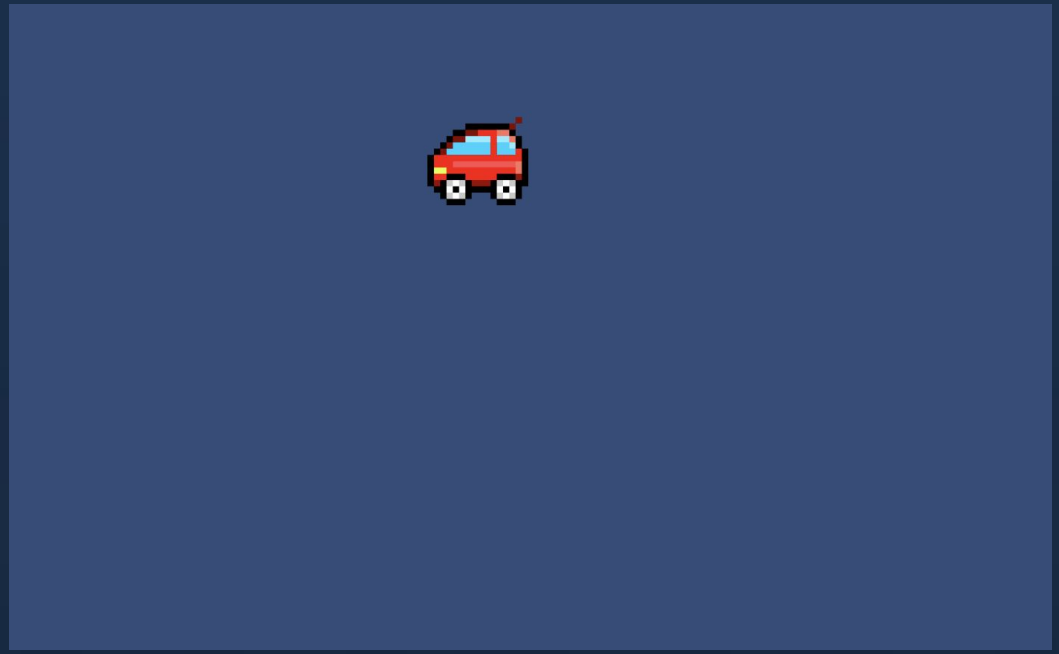
```
        if(Input.GetKey("up")) // 만약 위쪽 화살표 키가 눌려 졌다면
        {
            vy = speed;
        }

        if(Input.GetKey("down")) // 만약 아래쪽 화살표 키가 눌려 졌다면
        {
            vy = -speed;
        }
    }

    void FixedUpdate()
    {
        this.transform.Translate(vx / 50, vy / 50, 0);
        this.GetComponent<SpriteRenderer>().flipX = leftFlag;
    }
}
```

키보드 처리

- 결과 화면
  - 화살표 키의 이동에 따라 자동차가 움직인다.

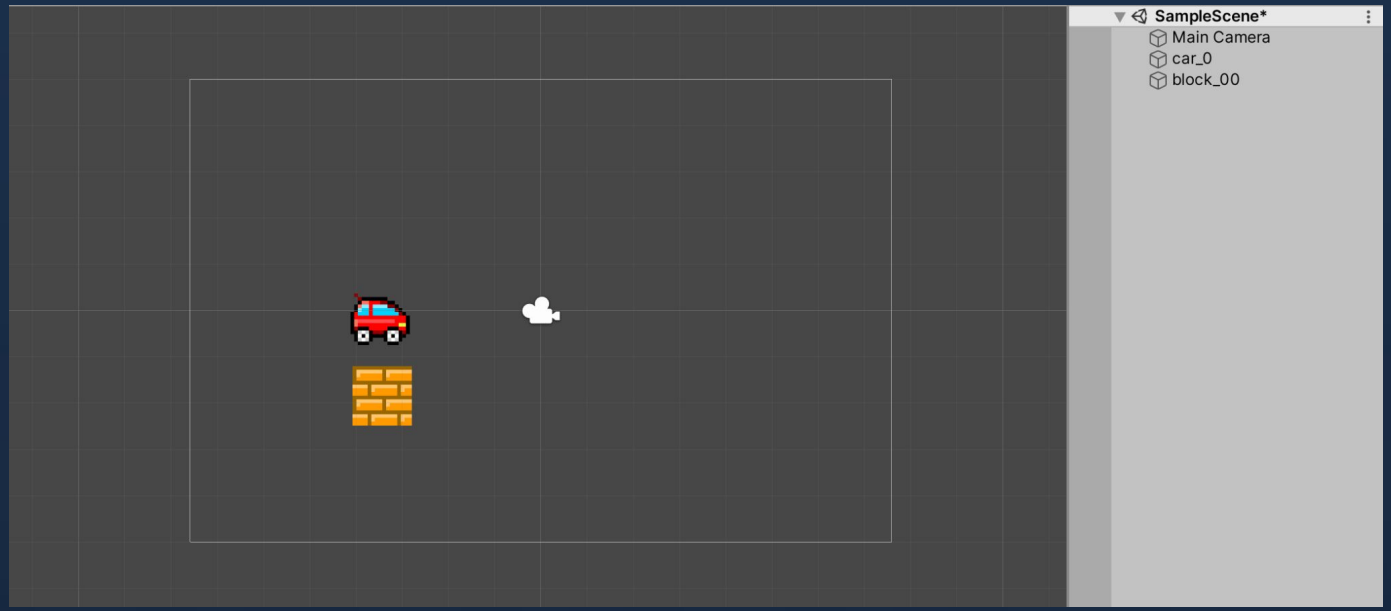


2.충돌 판정

- 충돌 판정
  - 유니티 게임엔진에서는 **[리지드바디]**와 **[콜라이더]** 2개의 콤포넌트를 사용하여 물리적인 충돌을 쉽게 구현할 수 있음
- **리지드 바디**
  - 게임 오브젝트에 물리적인 움직임을 시키는 콤포넌트
  - 예) 총알과 벽이 충돌할 경우 리지드 바디 콤포넌트는 벽이 아니라 총알에 들어가 있어야 함
  - 3D용 Rigidbody / 2D용 Rigidbody2D 가 있음
- **콜라이더**
  - 게임오브젝트에 충돌을 행하는 형태를 설정하는 콤포넌트
  - 2D 게임이면 여러가지 형태의 이미지가 게임오브젝트가 될텐데"이 그림의 어디가 충돌할 것인가"를 지정하는 것이 콜라이더임
  - 총알과 벽이 모두 콜라이더 콤포넌트를 포함하고 있어야만 충돌 판정을 할 수 있음
  - Box Collider 2D, Circle Collider 2D, Capsule Collider 2D, Polygon Collider 2D 등이 있음
    - 벽->Box, 사람->Capsule, 동그란 공 ->Circle 등으로 설정할 수 있음

충돌 판정

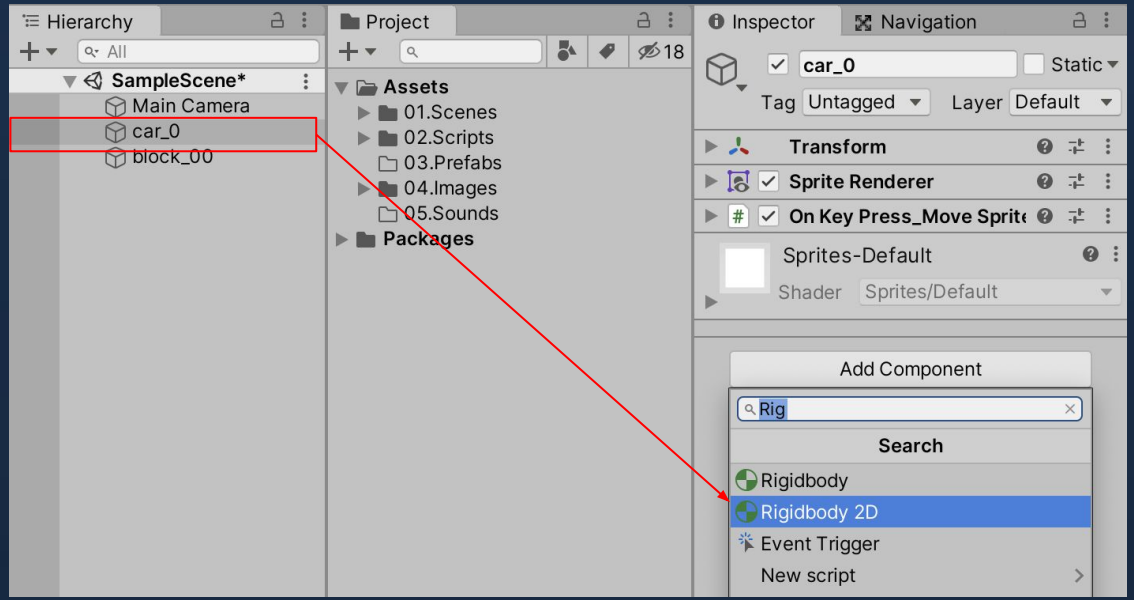
- 공중에서 자동차를 떨어뜨려서 블록으로 만든 지면으로 받아내겠습니다.
- (자동차와 블럭이 충돌이 됨)
- 이미지 파일 : Car_0 (자동차 이미지 파일) / block_00 (블록 이미지 파일)



충돌 판정

- 콤포넌트 추가
  - 자동차 오브젝트에 2개의 컴포넌트를 추가한다.
    - Rigidbody 2D
    - Box Collider 2D
  - 벽돌 오브젝트에게는 1개의 컴포넌트만 추가합니다.
    - Box Collider 2D

- 콤포넌트 추가 방법
  - 씬뷰에서 오브젝트 선택
  - 인스펙트 창에서
  - Add Component

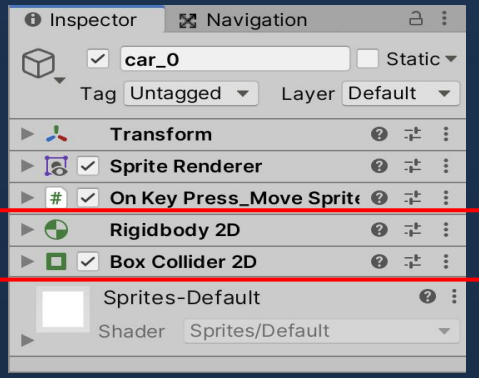


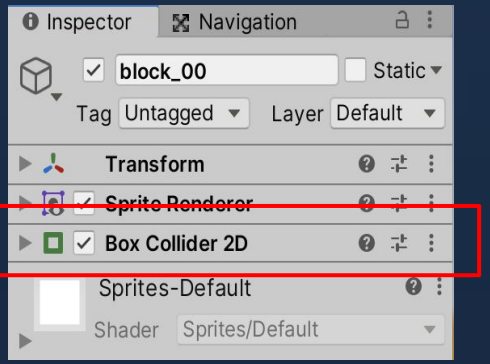
충돌 판정

- 콤포넌트 구성 내용

- Rigidbody 2D
- Box Collider 2D

- Box Collider 2D





충돌 판정

- (실험 1) 자동차를 놓으면 블록위에서 멈춘다.
- (실험 2) 자동차를 블록의 반정도에 위치한 후 실행을 누르면 물리 법칙을 적용 받아 회전해서 떨어진다.

충돌 판정

- 떨어지지 않도록 만든다. (중력 무시)
- 충돌해도 회전하지 않도록 한다
- OnKeyPress_MoveSprite(수정)

```
public class OnKeyPress_MoveSprite : MonoBehaviour
{
    public  float speed = 2; // 속도
    private float vx = 0;
    private float vy = 0;
    private bool  leftFlag = false;

    Rigidbody2D rbody;

    void Start()
    {
        rbody = GetComponent<Rigidbody2D>();
        rbody.gravityScale = 0;
        rbody.constraints = RigidbodyConstraints2D.FreezeRotation;
    }

    void Update() // 매 프레임 마다 호출 (계속 수행한다)...

    void FixedUpdate()...
}
```

충돌 판정

- 이동처리 방법 변경(Translate()함수 가 아닌 물리엔진 사용)
  - 어느 정도의 속도로 이동할지를 지정하고 실제로 움직이는 처리는 물리엔진에게 맡기는 방법
  - **Rigidbody2D.velocity = new Vector2(vx, vy);**
  - transform.Translate() 메소드를 사용하면 위치가 갑자기 강제적으로 바뀌는 현상이 있어서 물리적으로 부자연스러운 움직임을 보일 수 있기 때문에 이러한 처리를 한 것임

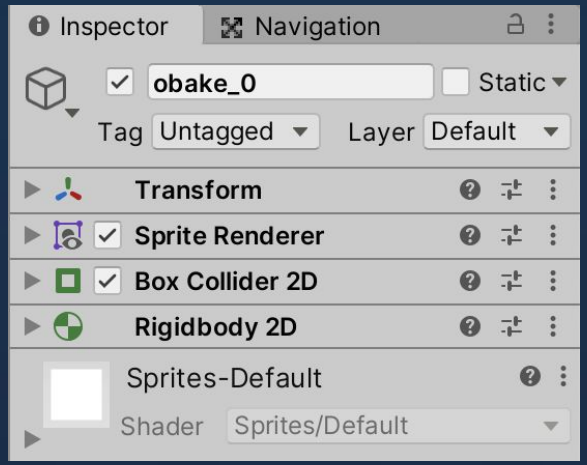
```
void FixedUpdate()
{
    //this.transform.Translate(vx/50, vy/50, 0);
    rbody.velocity = new Vector2( x: vx,  y: vy);
    this.GetComponent<SpriteRenderer>().flipX = leftFlag;
}
```

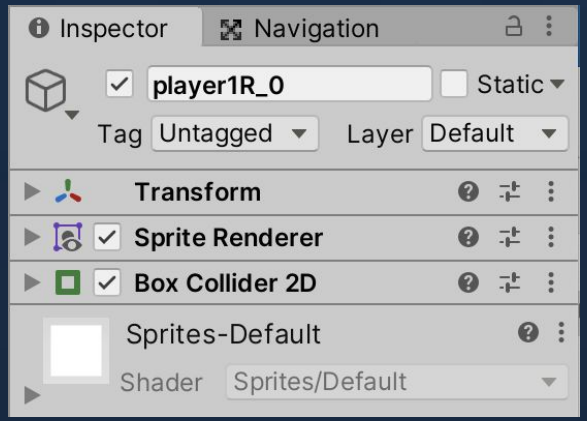
3.따라오기 (계속 뒤쫓아 간다)

- 적이 “뒤쫓아 계속 따라오는” 알고리즘
  - (1) 플레이어 **오브젝트를 찾아낸다**.
    - GameObject Player
    - **Player = GameObject.Find(PlayerName)**
  - (2) **플레이어의 방향을 조사**한다.
    - 점 – 점 = 벡터 (기하학 원리)
    - **Vector3 dir = (플레이어.transform.position – 몬스터.transform.position).normalized;**
    - normalized
      - normalized를 붙이면 방향은 유지한 채 길이가 1.0인 벡터가 만들어 진다.
      - 붙이지 않을때는 2개 오브젝트간의 거리가 된다. 거리가 멀때는 값이커지고, 가까울때는 값이 작아진다. 만약 이 값을 이용해서 뒤쫓아 가면 오브젝트 간의 거리가 멀때는 빠르게, 가까울때는 느리게 좇아가게 된다. normalized를 붙이면 오브젝트간의 거리에 상관없이 언제나 일정한 속도로 뒤쫓아 가게 된다.

따라오기 (계속 뒤쫓아 간다)

- 이미지 파일 준비 : 유령(obake_0), 플레이어(player1R_0)
  - 씬 뷰에 올려 놓는다.
- 유령 게임 오브젝트 : Add Component
  - Rigidbody2D
  - Box Collider 2D
- 플레이어 게임 오브젝트 : Add Component
  - Box Collider 2D





따라오기

- 새로운 스크립트 만들기 : Forever_Chase

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Forever_Chase : MonoBehaviour
{
    public string targetObjectName; //목표 오브젝트 이름 : Inspector에서 지정
    public float speed = 1; // 속도 : Inspector에서 지정

    GameObject targetObject;
    Rigidbody2D rbody;
```

따라오기

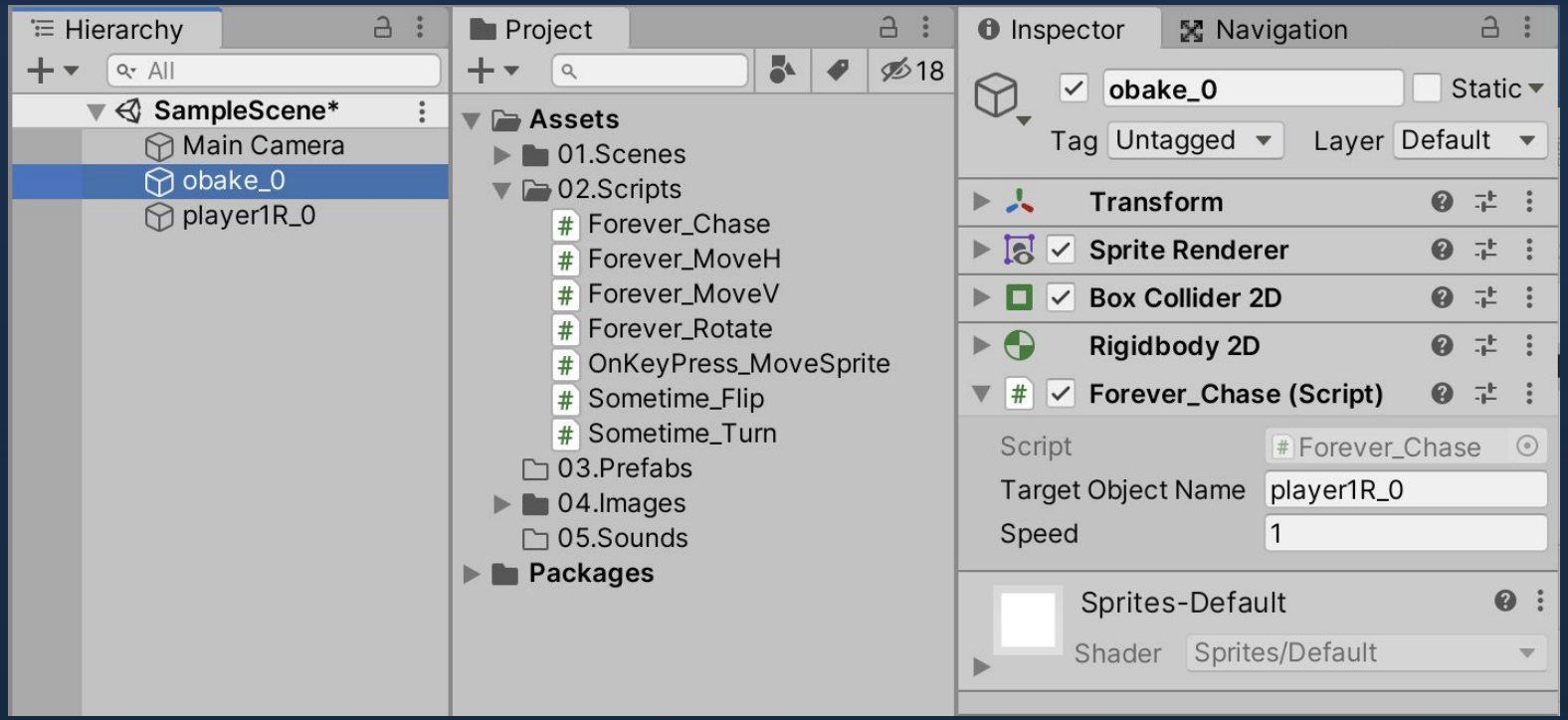
- 스크립트 작성 : Forever_Chase

```
    void Start()
    {
        // 목표 오브젝트를 찾아낸다
        targetObject = GameObject.Find(targetObjectName);
        rbody = GetComponent<Rigidbody2D>();
        rbody.gravityScale = 0; // 중력을 0으로 셋팅
        rbody.constraints = RigidbodyConstraints2D.FreezeRotation;
    }

    void FixedUpdate()
    {
        Vector3 dir = (targetObject.transform.position - this.transform.position).normalized;
        float vx = dir.x * speed;
        float vy = dir.y * speed;
        rbody.velocity = new Vector2(vx, vy);
        this.GetComponent<SpriteRenderer>().flipX = (vx < 0);
    }
}
```

따라오기

- TargetObjectName 에다 플레이어 이름을 셋팅 : player1R_0



생각 해 볼 문제

- 따라오기를 달아나기(회피하기)로 바꿀려면 어떻게 하면 될까요?