유니티
게임 제작 입문

게임 제작의 기초

길벗

유니티 구성요소와 특징

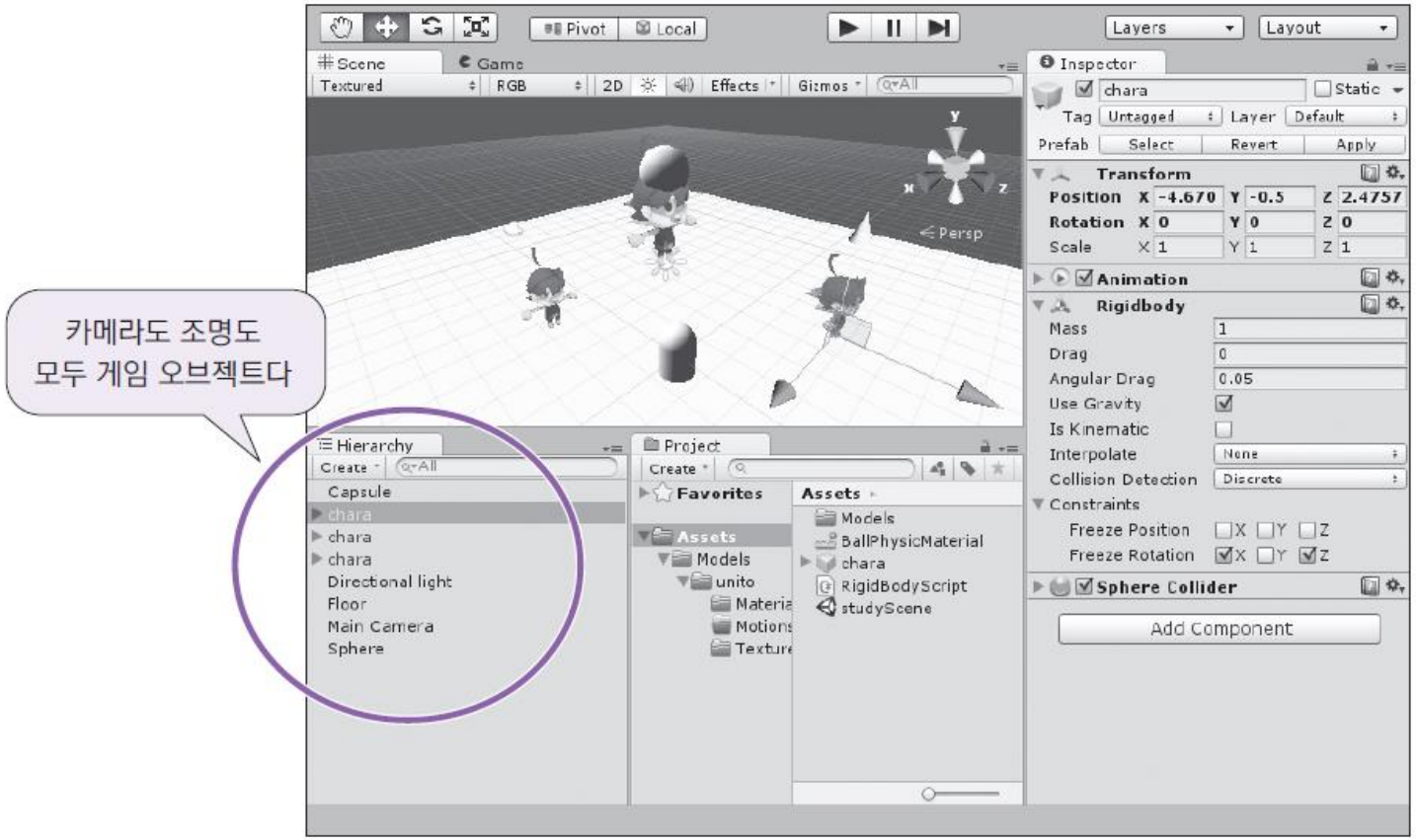
게임 오브젝트

리자드바디

프리팹

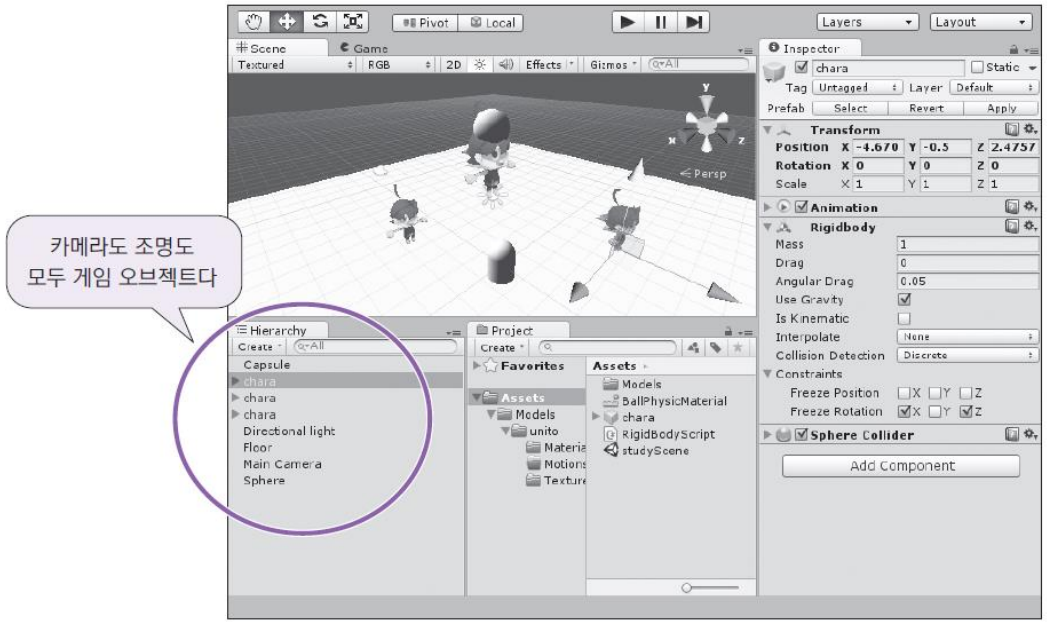
1. 게임 오브젝트

그림 2-1 게임 오브젝트란



1. 게임 오브젝트

그림 2-1 게임 오브젝트란



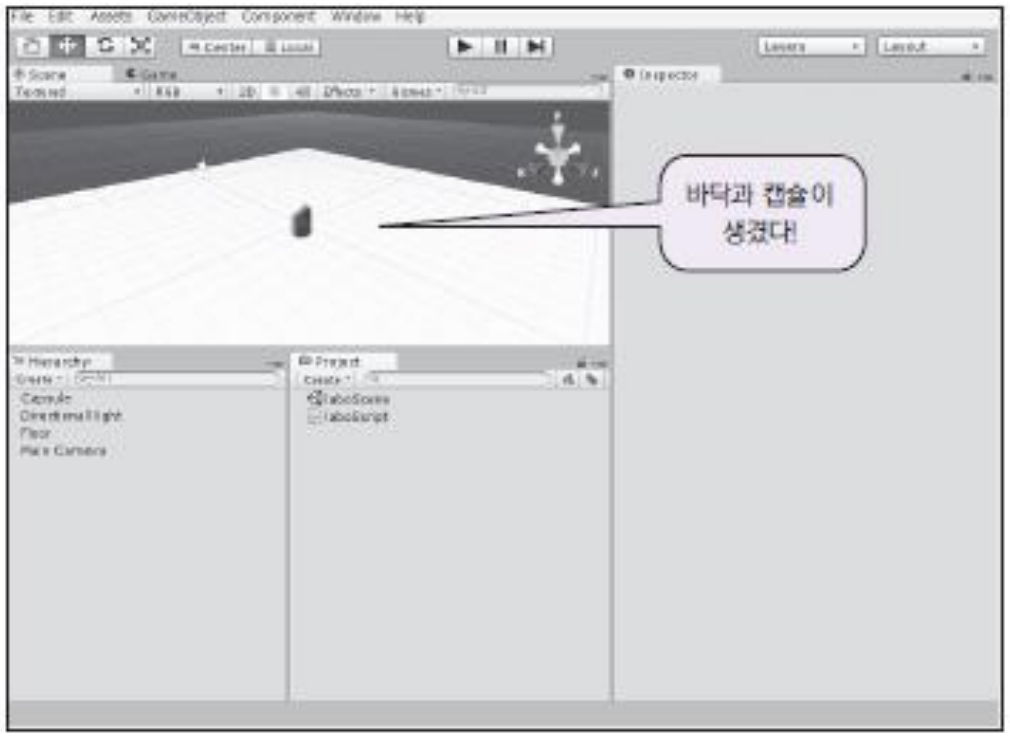
유니티의 씬에 배치된 것은 모두 게임 오브젝트.

게임 상에서 보이지 않는 존재도 포함

게임 오브젝트에 스크립트를 추가하여 동작시킨다

* 실습 준비!

그림 2-3 바닥과 캡슐이 만들어졌다



- 바닥

[GameObject]-[Create Other]-[Cube]

- 조명

[GameObject]-[Create Other]-
[Directional Light]

- 캡슐 + 스크립트

[GameObject]-[Create Other]-[Capsule]

[Assets]-[Create]-[C# Script]

* Transform 변경하기

오브젝트 안에 사용된 Inspector의 각도, 크기, 회전을 변경할 수 있다.

```
void Update() {
    if (Input.GetKeyDown(KeyCode.A)) {
        float rnd = Random.Range (0.0f, 0.5f);
        this.transform.position = new Vector3 (0.0f, 0.0f, rnd);
    }

    if(Input.GetKeyDown(KeyCode.B)) {
        float rnd = Random.Range(0.0f, 360.0f);
        // x축 기준으로 회전값을 임의로 변경.
        this.transform.rotation = Quaternion.Euler(rnd, 0.0f, 0.0f);
    }
}
```

0과 0.5 사이에 난수 발생

A키가 눌리면 거리가 조금씩 변한다.

B키가 눌리면 오브젝트를 회전시킨다.
X축 기준으로 임의의 각도만큼 회전.

* Translate

```
void Update() {
.....

    if(Input.GetKey(KeyCode.UpArrow)) {
        this.tranform.Translate(
            new Vector3(0.0f, 0.0f, 3.0f * Time.deltaTime));
    }
}
```

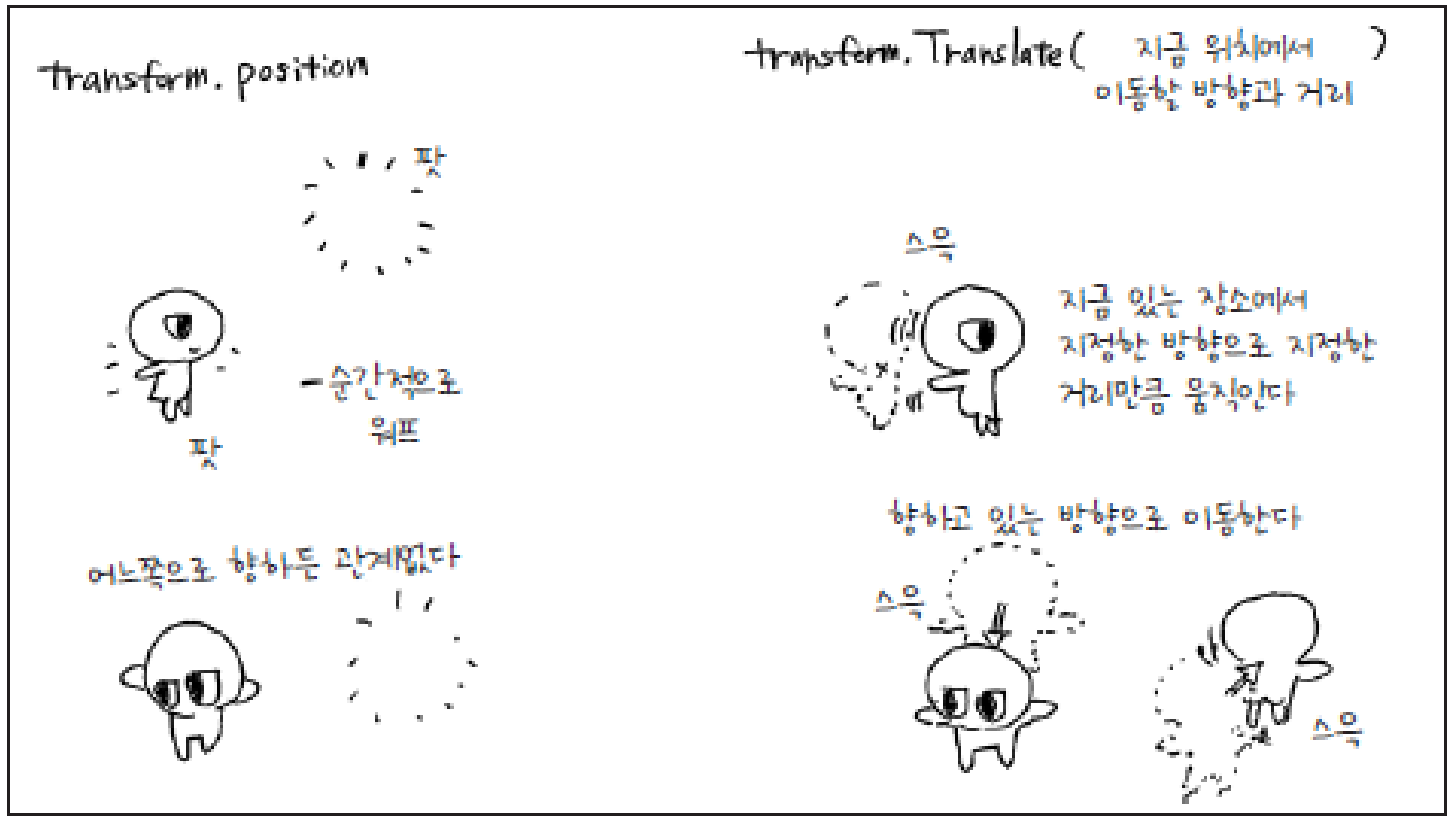
오브젝트 자신이 현재 위치와 각도를 기준으로 조작한다.

Tranform과 다르게 단순히 이동할 거리만 지정. 오브젝트가 향한 방향으로 진행한다.

절대위치를 단순한 표현 가능, 하지만 변하는 방향에 맞춰 위치 변경하려면 재계산 필요

* Transform과 Translate

그림 2-4 Translate



* Time.deltaTime

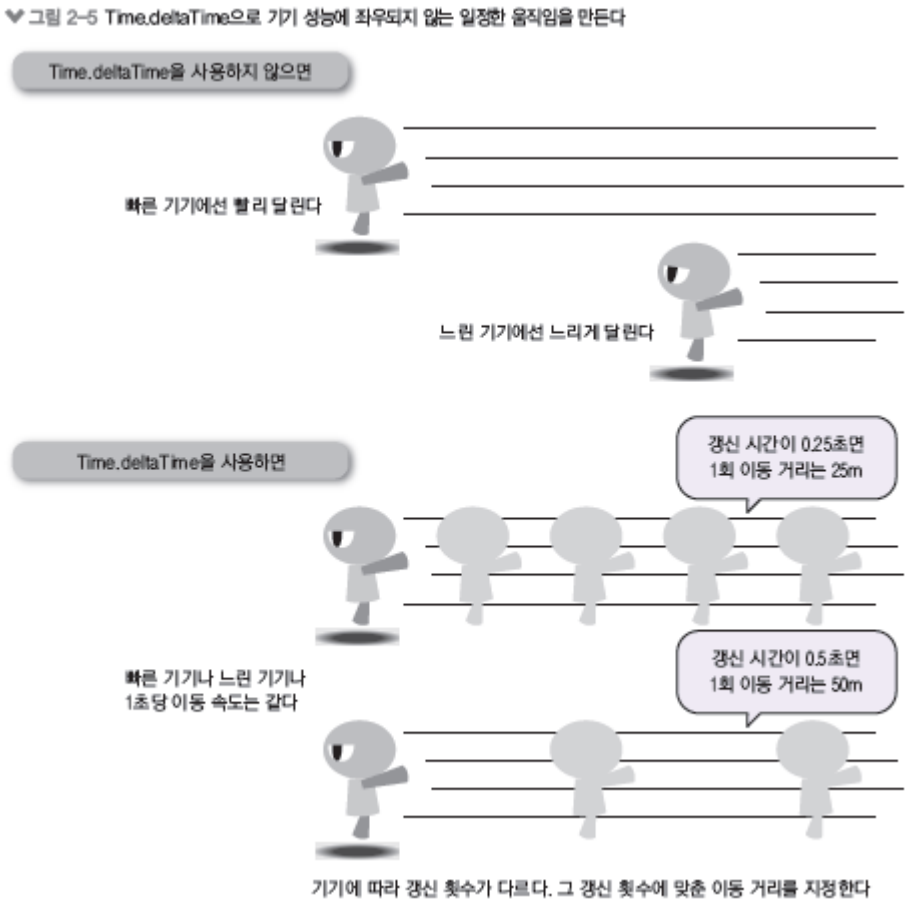


그림 2-5 Time.deltaTime으로 기기 성능에 좌우되지 않는 일정한 움직임을 만든다

기기에 따라 갱신 횟수가 다르다. 그 갱신 횟수에 맞춘 이동 거리를 지정한다

기기의 성능에 따라 게임의 갱신빈도가 다르다.

기기별 성능의 차이를 메워주는 것

=> Time.deltaTime

초당 이동 거리와 회전량 지정

빠른 기기 : 1회 이동거리 ↓ 갱신 빈도 ↑
느린 기기 : 1회 이동거리 ↑ 갱신 빈도 ↓

* Rotate

오브젝트가 향한 방향에서 얼마나 회전시킬 것인가. **=> 상대적인 회전**

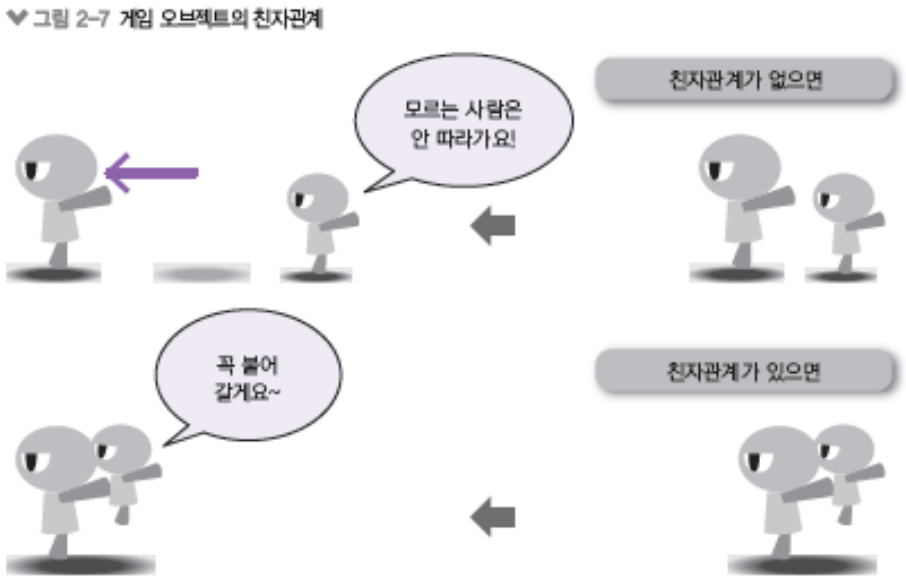
```
void Update() {
...

    if (Input.GetKey(KeyCode.R)) {   // R키로 우회전.
        this.transform.Rotate(90.0f * Time.deltaTime, 0.0f, 0.0f);
    }

    if (Input.GetKey(KeyCode.L)) {   // L키로 좌회전.
        this.transform.Rotate(-90.0f * Time.deltaTime, 0.0f, 0.0f);
    }
}
```

절대적 회전은 rotation. 잊지 말자!

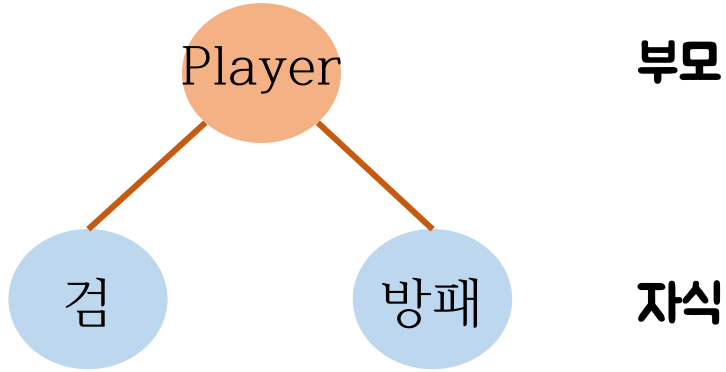
* 오브젝트의 친자 관계

오브젝트는 서로의 부모-자식 관계가 될 수 있다.

친자 관계로 엮어두면 게임 조작에 훨씬 편리하다. parent 사용



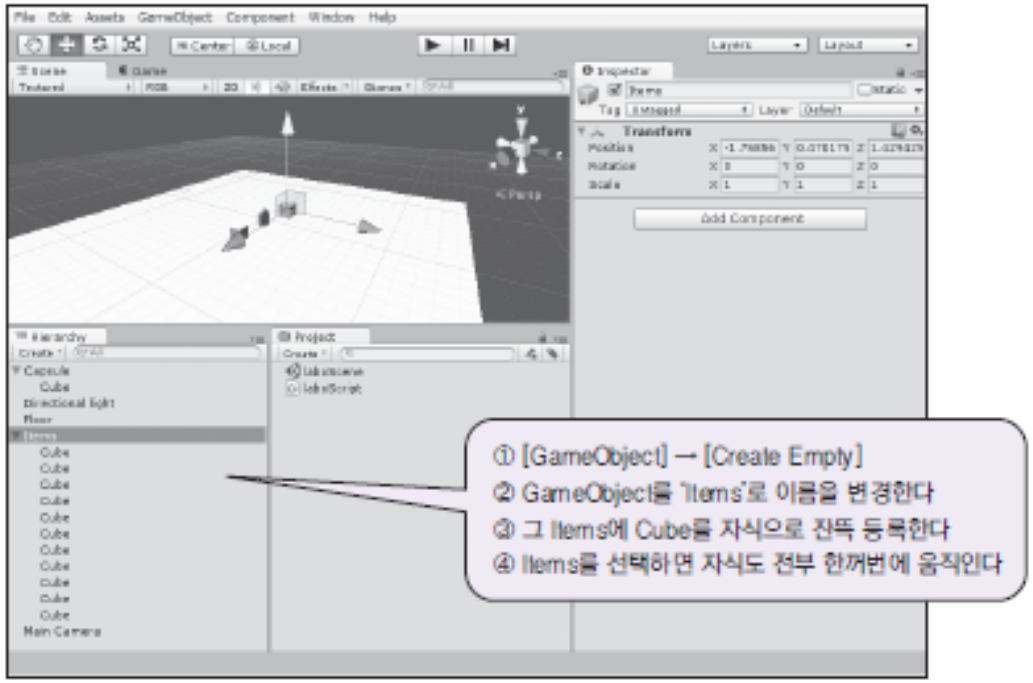
EX) 무기와 방패를 가진 캐릭터



* 오브젝트의 친자 관계

- 빈 게임 오브젝트를 폴더처럼 사용하기 (Item 창 만들 때 유용)

그림 2-9 Hierarchy에 폴더 기능을 넣는다



Item을 이동하면
안에 들어있는 항목을
한꺼번에 이동할 수 있음

* Script 재사용

같은 처리를 여러 번 작성하는 수고를 덜 수 있다.

Ex) cubeScript의 bigsize 메서드를 재사용하려고 할 때!

cubeScript.cs

```
using UnityEngine;
using System.Collections;

public class cubeScript: MonoBehaviour {
    void Start() {
    }
    void Update() {
    }

    public void bigsize() {
        // x, y, z 모든 방향에 대해서 크기를 3배로 한다.
        this.transform.localScale = new Vector3(3.0f, 3.0f, 3.0f);
    }
}
```

studyScript.cs

```
void Update() {
...

        if (Input.GetKey(KeyCode.G)) {
          GameObject go = GameObject.Find("Cube") as GameObject;
          go.GetComponent<cubeScript>().bigsize();
    }
}
```

cubeScript의 bigsize() 메서드 시작

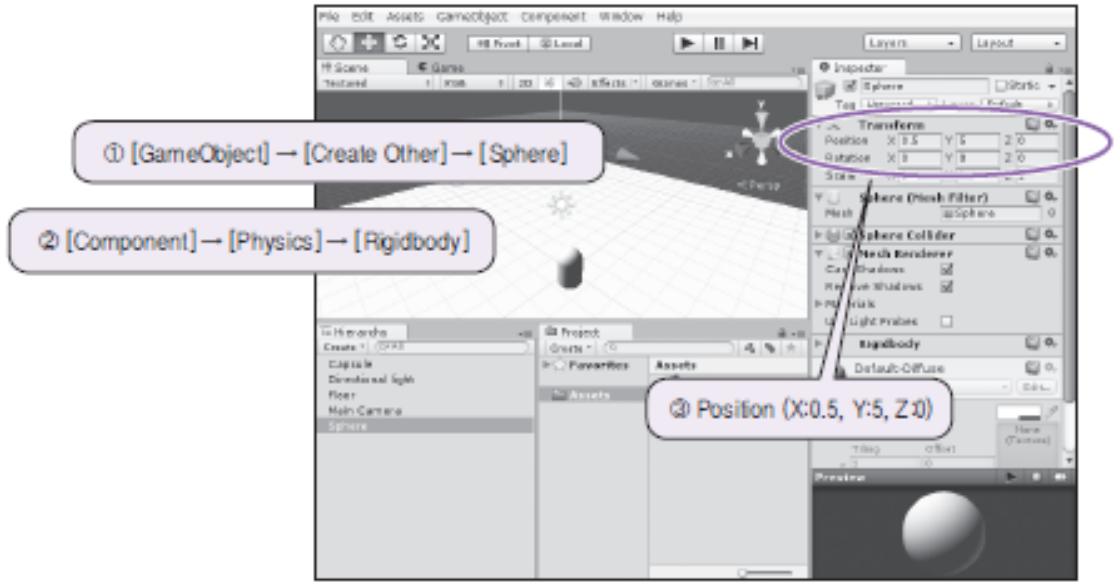
2. 리지드바디

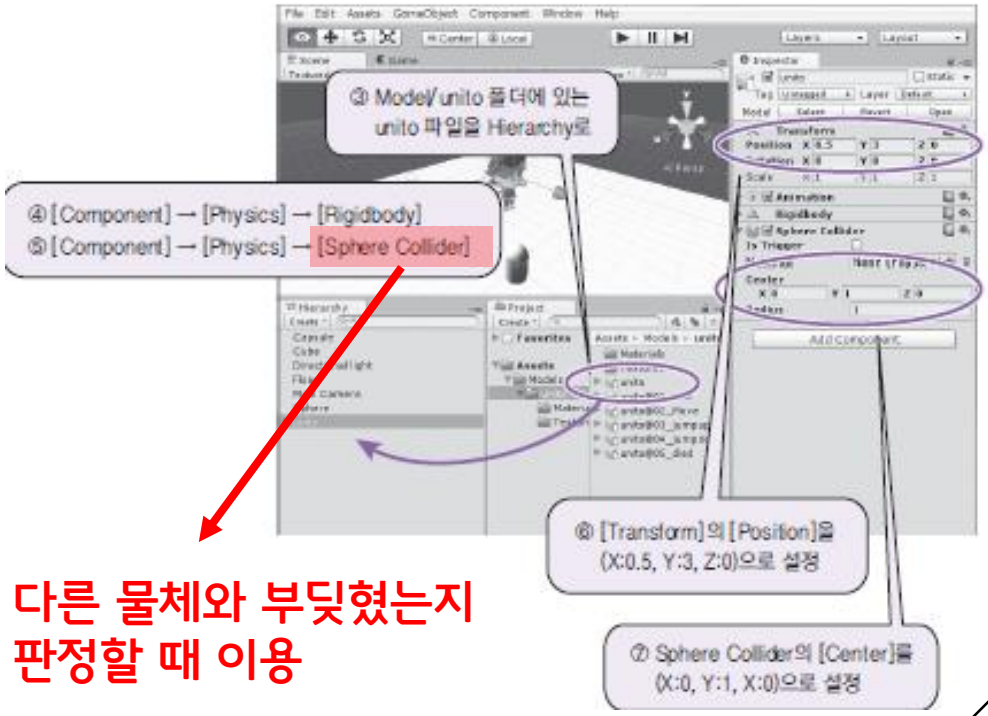
게임 오브젝트에 물리적인 움직임을 부여하는 마법

리지드바디를 적용하면 오브젝트는 중력에 의해 낙하한다. 즉, **중력 효과를 부여**한다.

3D

그림 2-11 리지드바디를 적용하자



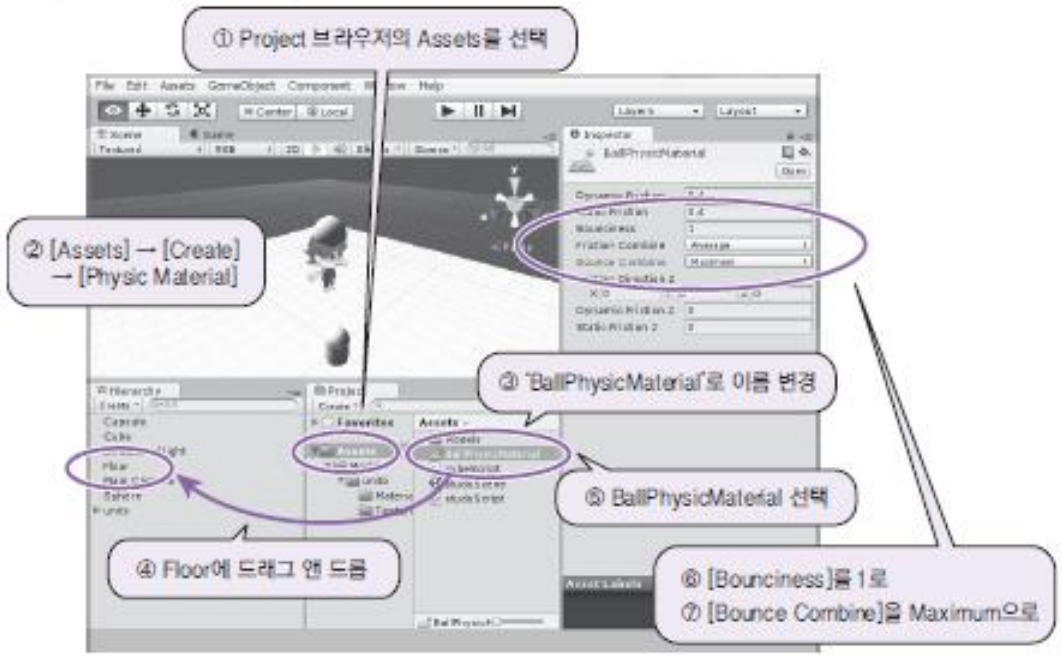


다른 물체와 부딪혔는지 판정할 때 이용

* Physic Material

슈퍼마리오 게임을 생각해보자. 단계마다 배경이 다른데, 벽돌 맵이 있는 가 하면 얼음 맵도 있다.
이를 표현하기 위해 게임 오브젝트에서 소재를 결정할 수 있다.

그림 2-14 Physic Material을 설정한다



* Script에서 리지드바디 조작

메서드 안에서 리지드바디를 조작할 수 있다.
'지정한 방향으로, 지정한 양의 힘을 가하는.'

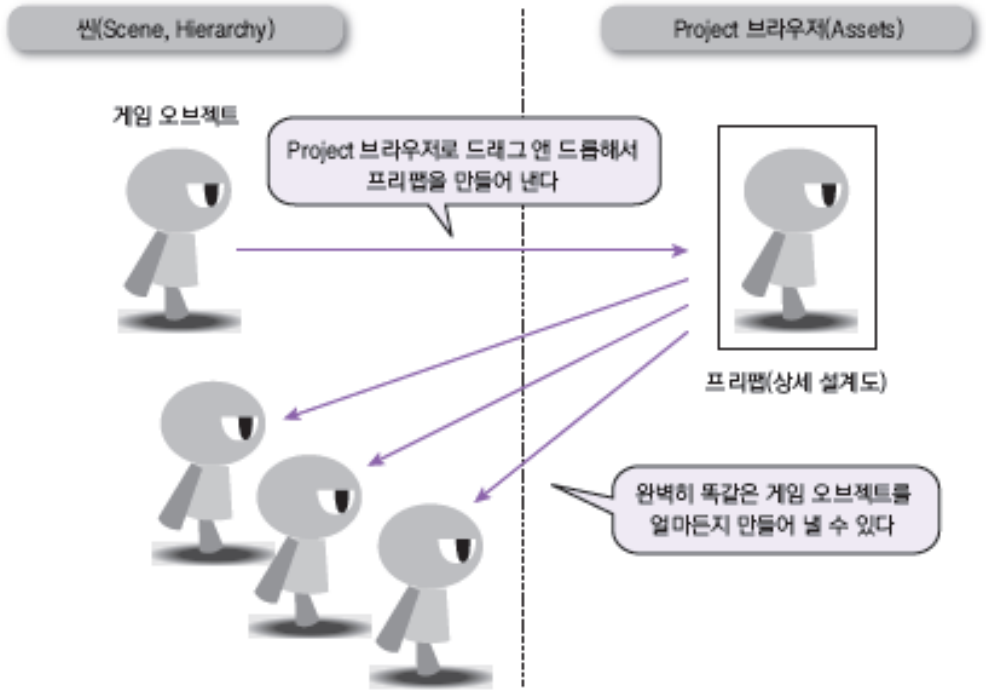
```
void Update() {
    if(Input.GetKey(KeyCode.UpArrow)) {      // ↑키로 안쪽으로.
        this.transform.rigidbody.AddForce(
            Vector3.forward * 300 * Time.deltaTime);
    }
    if(Input.GetKey(KeyCode.DownArrow)) {    // ↓키로 앞쪽으로.
        this.transform.rigidbody.AddForce(
            Vector3.back * 300* Time.deltaTime);
    }
    if(Input.GetKey(KeyCode.LeftArrow)) {    // ←키로 왼쪽으로.
        this.transform.rigidbody.AddForce(
            Vector3.left * 300 * Time.deltaTime);
    }
    if(Input.GetKey(KeyCode.RightArrow)) {   // →키로 오른쪽으로.
        this.transform.rigidbody.AddForce(
            Vector3.right * 300 * Time.deltaTime);
    }
    if(Input.GetKey(KeyCode.U)) {            // U키로 위로.
        this.transform.rigidbody.AddForce(
            Vector3.up * 300 * Time.deltaTime);
    }
    if(Input.GetKey(KeyCode.D)) {            // D키로 아래로.
        this.transform.rigidbody.AddForce(
            Vector3.down * 300 * Time.deltaTime);
    }
}
```

* Script에서 리지드바디 조작

중력 제어 gravity 파라미터 이용

```
void Update() {
...

    if(Input.GetKeyDown(KeyCode.Keypad0)) {      // 텐키(숫자 키패드)의 0.
        Physics.gravity = Vector3.zero;          // 중력을 제로로 한다.
    }
    if(Input.GetKeyDown(KeyCode.Keypad8)) {      // 텐키의 8.
        Physics.gravity = Vector3.up;            // 중력 방향을 위로 한다.
    }
    if(Input.GetKeyDown(KeyCode.Keypad2)) {      // 텐키의 2.
        Physics.gravity = Vector3.down;          // 중력 방향을 아래로 한다.
    }
}
```

3. 프리팹



- 게임 오브젝트의 상세 설계도
- 배치할 때 사용되는 설계도
- Hierarchy에 있는 게임 오브젝트를 Project 브라우저로 drag & drop하여 생성

* 프리팹을 프로그램으로 배치하기

그림 2-21 프리팹 클론을 많이 만든다

교재를 따라 하다 보면 게임 오브젝트가 고무 소재라 통통 튕겨나가는 모션을 만들 수 있다.

GameRootScript.cs

```
oid Update() {
    if(Input.GetMouseButtonDown(0)) {
        // Instantiate(prefab);

        // prefab 변수에서 만들어진 게임 오브젝트 획득.
        GameObject go =
            GameObject.Instantiate(this.prefab) as GameObject;
        // 획득한 게임 오브젝트의 설정 변경.
        go.transform.position =
            new Vector3(Random.Range(-2.0f, 2.0f), 1.0f, 1.0f);
    }
}
```

* 효과음 넣기

```
public GameObject prefab = null;

private AudioSource audio;
public AudioClip jumpSound;

void Start() {
    // GameObject에 〈AudioSource〉 컴포넌트를 추가한다.
    // 이렇게 하면 소리를 다룰 수 있게 된다.
    this.audio = this.gameObject.AddComponent<AudioSource>();

    // jumpSound에 저장한 소리를 내도록 준비한다.
    this.audio.clip = this.jumpSound;

    // 반복 재생하지 않도록 설정한다.
    this.audio.loop = false;
}

void Update() {
    if(Input.GetMouseButtonDown(0)) {
        GameObject go =
            GameObject.Instantiate(this.prefab) as GameObject;
        go.transform.position =
            new Vector3(Random.Range(-2.0f, 2.0f), 1.0f, 1.0f);

        this.audio.Play();    // audio에 들어 있는 소리를 재생한다.
    }
}
```

* OnGUI()

2D 표현 **사용 예)** 체력 게이지, score.

GameRootScript.cs

```
public Texture2D icon = null;
public static string mes_text = "test";

void OnGUI() {
    GUI.DrawTexture(new Rect(Screen.width / 2, 64, 64, 64), icon);
    GUI.Label(new Rect(Screen.width / 2, 128, 128, 32), mes_text);
}
```

사용할 변수를 먼저 설정

유니티에서 GameRoot 선택 - [GameRootScript] 컴포넌트에 새로 생긴 [Icon] 항목에 넣고 싶은 이미지를 드래그 앤 드롭

* 씬의 이동

게임에는 실행화면만 있는 것이 아니다. 타이틀화면이 있고, 결과를 나타내는 화면도 있다.

화면 간의 이동은 꼭 필요하다. 먼저 씬을 저장하고 새로운 씬을 만들자.

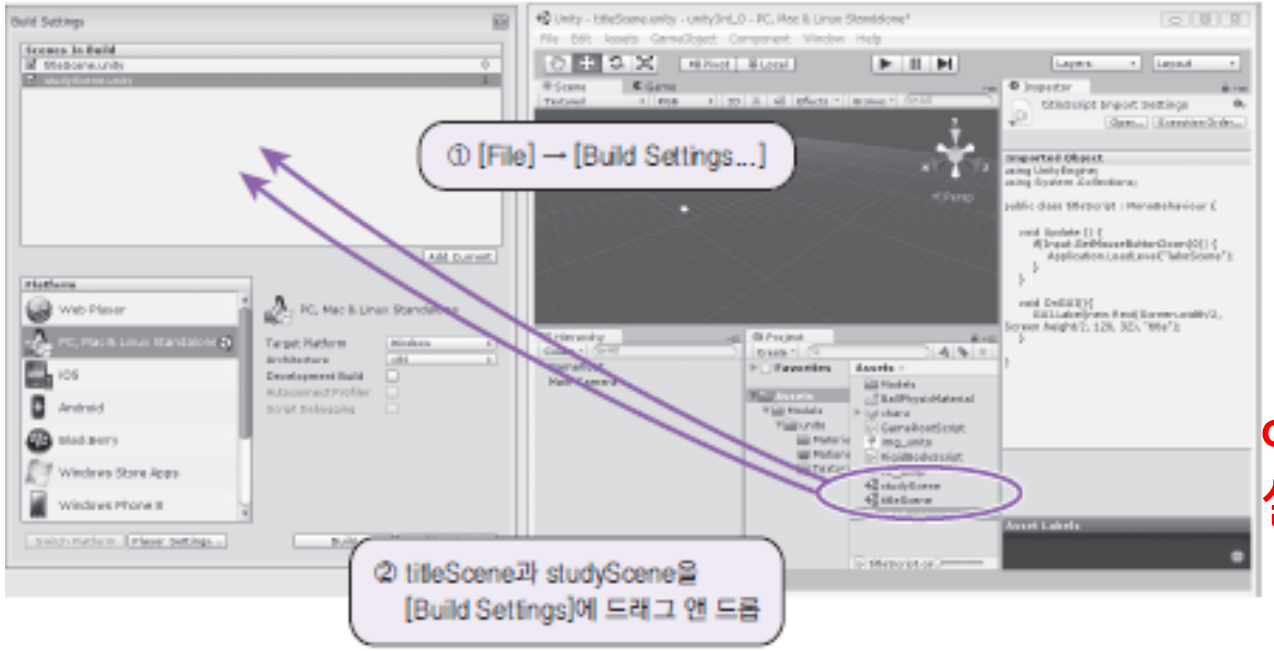
```
void Update() {
    if(Input.GetMouseButtonDown(0)) {          // 마우스 왼쪽 버튼을 누르면.
        Application.LoadLevel("studyScene");   // studyScene으로 이동.
    }
}

void OnGUI() {
    // 화면에 title이라고 표시.
    GUI.Label(new Rect(Screen.width/2, Screen.height/2, 128, 32), "title");
}
```

* 씬의 이동

게임에는 실행화면만 있는 것이 아니다. 타이틀화면이 있고, 결과를 나타내는 화면도 있다.

화면 간의 이동은 꼭 필요하다. 먼저 씬을 저장하고 새로운 씬을 만들자.

그림 2-26 Build Settings로 필요한 씬을 등록



어느 씬을 다룰 것인지
설정해주어야 한다.

재미 만들기

‘재미’란 ?

레벨디자인

기획서 작성요령

* 재미있는 게임이란?

평소에 하고 있는 게임이 있는가?

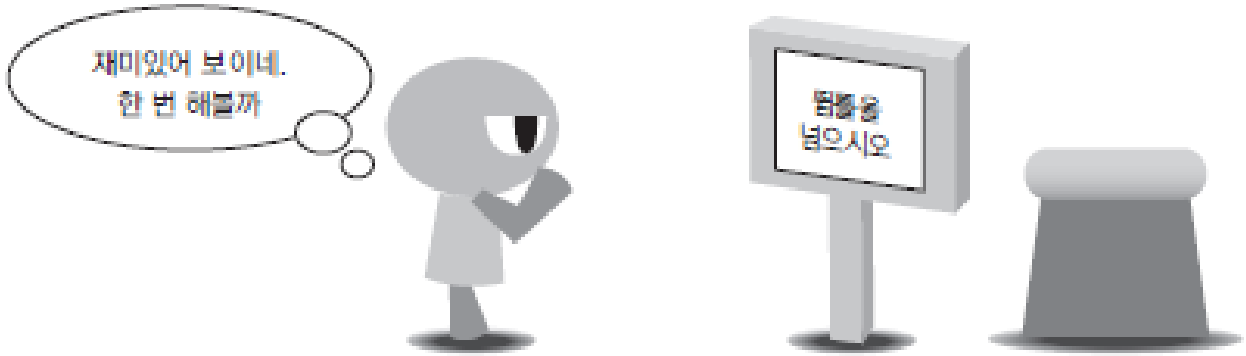
얼마나 즐겨 하는 가?

그 게임을 즐겨 하는 이유는 무엇인가?

* 재미의 요소

게임 개발자의 입장에선 유저가 게임을 즐겨주어야 좋다.

유저입장에서 게임을 즐기려면? **재미가 있어야 한다**



목표 고찰 행동 결과 보상

* 재미있는 게임을 만드는 방법

목표

알맞은 목표를 설정해라. (ex. 보스몹)

캐릭터의 성장이 어려워야 한다.

많은 목표를 주어라. - 게임을 오래 즐길 수 있다.

공통된 목표보단 변화를.

반복되는 행동에 변화를 준다

고찰

게임에 들어갈 한 가지 단순한 포인트도 어떻게 활용할 것인가

장점과 단점을 어떻게 활용할 것인가
Ex) 빠른 대신 공격력이 낮은 무기, 느린 대신 공격력이 쎈 무기

* 재미있는 게임을 만드는 방법

결과

다양하게 변형하여 연출

어느 정도 성공했는 지 알 수 있으면 결과에 대한 기대감을 높일 수 있다

이해하기 쉬운 실패를 만든다

보상

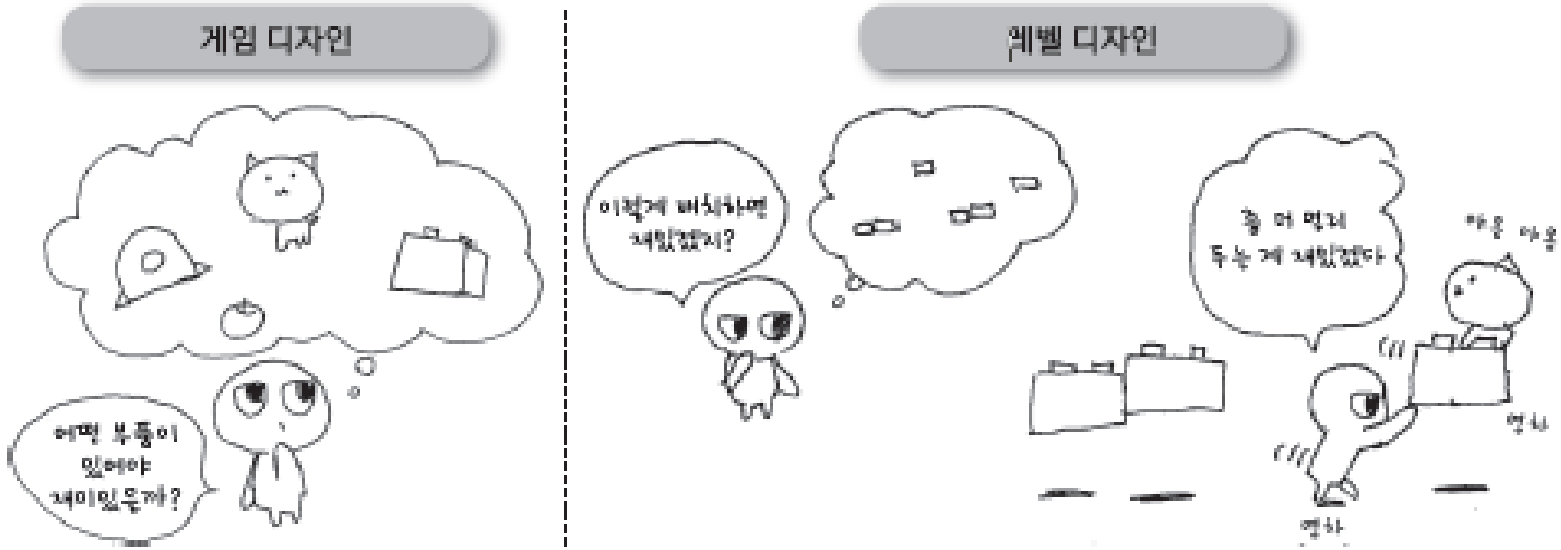
플레이어가 목표를 달성했을 때 보상을 준다

플러스 보상이 말고도 마이너스 보상(리스크)를 넣었을 때 재미는 더 증가할 수 있다

* 레벨 디자인

준비된 부품을 어떻게 재미있게 배치할 수 있느냐

그림 3-15 게임 디자인과 레벨 디자인



* 레벨 디자인

게임을 시작했을 때

게임의 첫 단계,
튜토리얼

코스 요리처럼

완급 조절은 필수

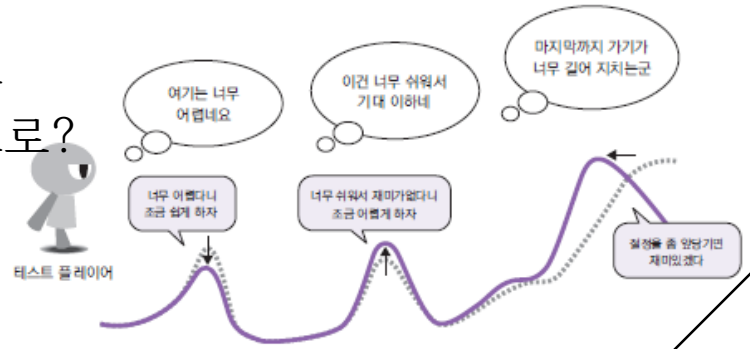
난이도 곡선

게임 시작~끝
어떻게 난이도를
배치할 것인가

가장 어려운 곳은
난도를 어느 정도로?

테스트 플레이

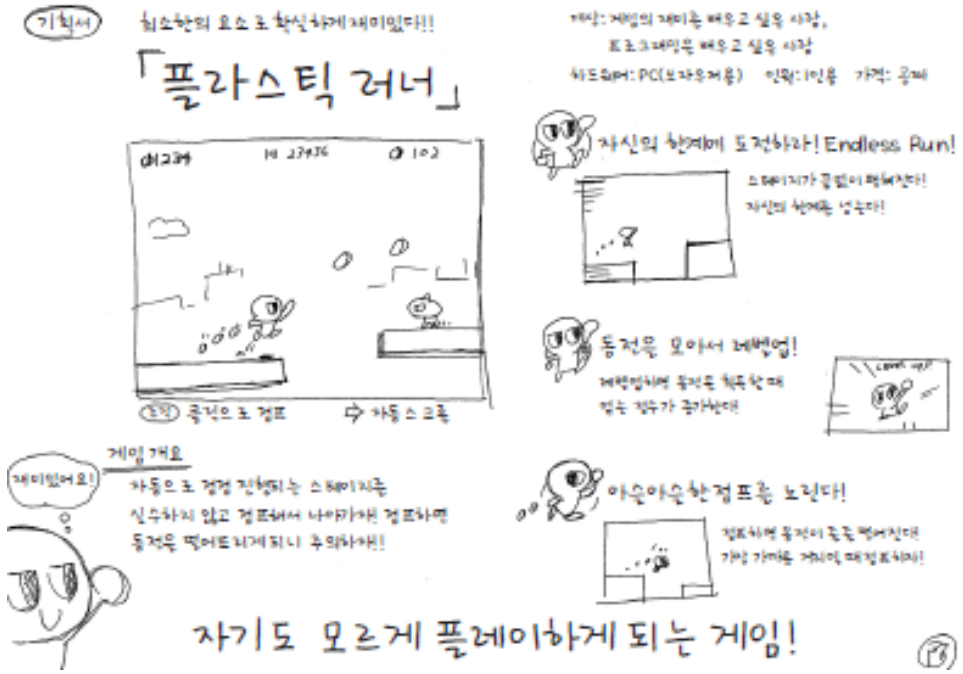
테스트 플레이어를 찾아
지금까지 디자인한 것들이
잘 맞는지 확인



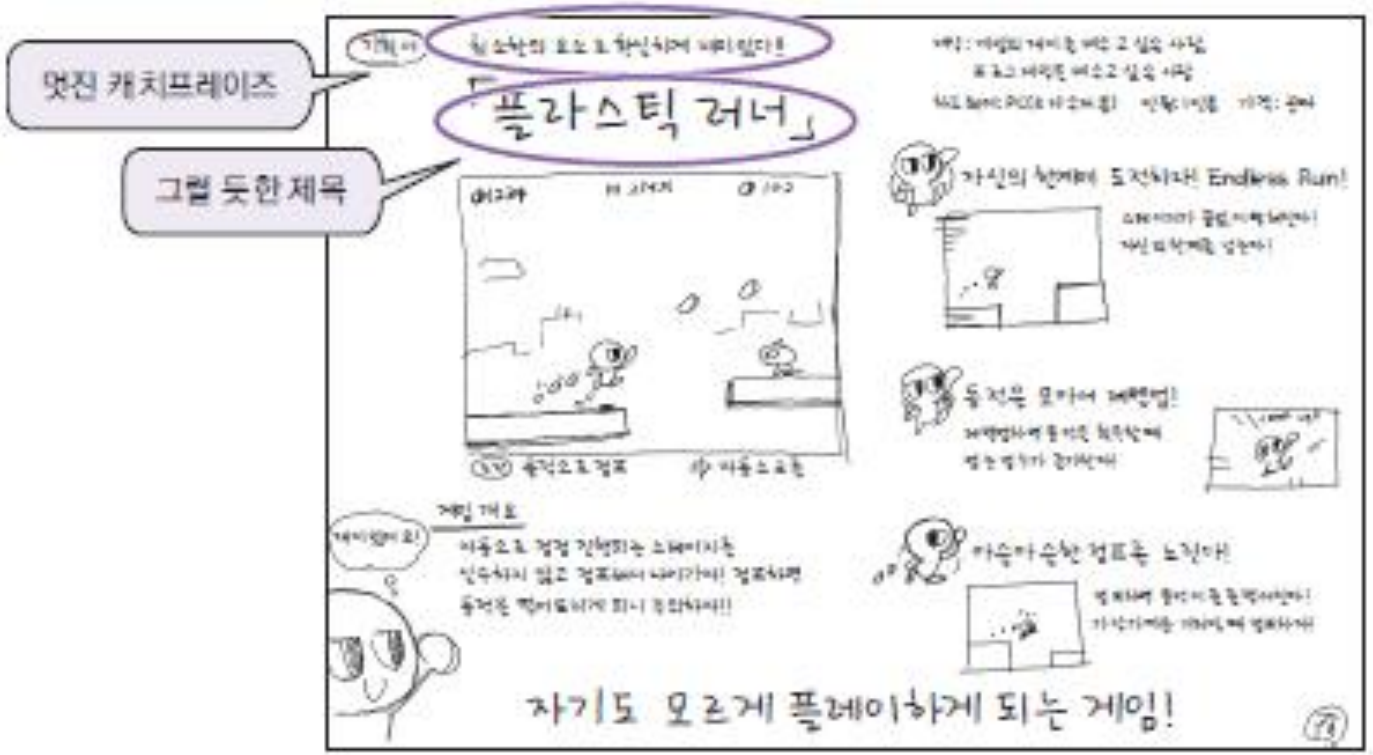
* 기획서

처음부터 너무 거창하게 만들 필요는 없다.
한 페이지짜리 기획서를 작성해보자.

기획개요

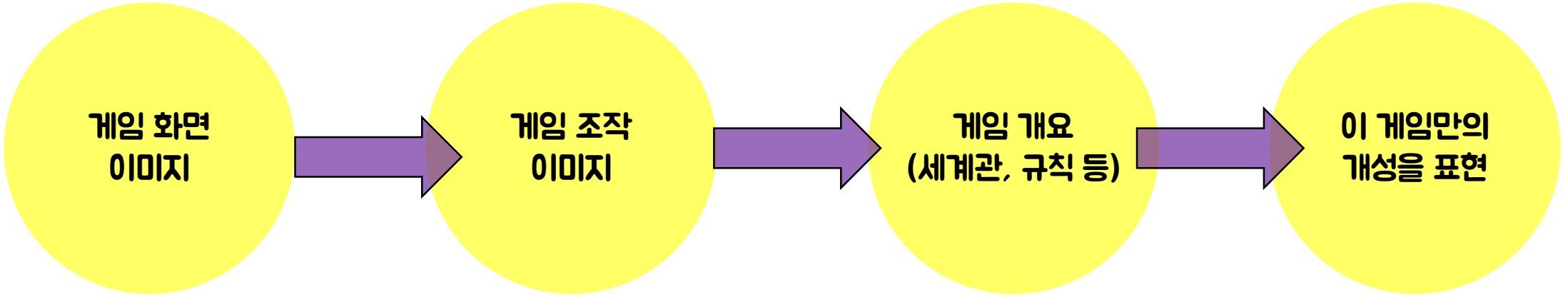


* 기획서 - 제목과 캐치프레이즈



그럴싸한 제목으로
기획서의 분위기를
대강 알 수 있게 한다.

* 기획서 - 제작 순서



Ex) 이동, 점프, 아이템 획득 등
전반적인 조작방법을 표현

게임의 재미요소를 표현
=> 왜 기획해야 하는 지!

* 기획서 - 깔끔하게 정리하기

그림 3-27(a) 글로만 가득 채운 기획서

기획서

최소한의 요소로 확실하게 재미있다! "플라스틱 러너"

대상: 게임의 재미를 배우고 싶은 사람, 프로그래밍을 배우고 싶은 사람
하드웨어: PC(브라우저용) **대상인원:** 1인용 **가격:** 무료

게임 개요: 자동으로 점점 진행되는 스테이지를 구멍에 빠지거나 고양이에게 닿지 않도록 점프해서 진행한다. 점프하면 동전(점수)을 떨어뜨리게 되니 가능한 한 아슬아슬하게 뛰어 넘어야 한다. 최고 점수를 갱신해 가는 게임이다.

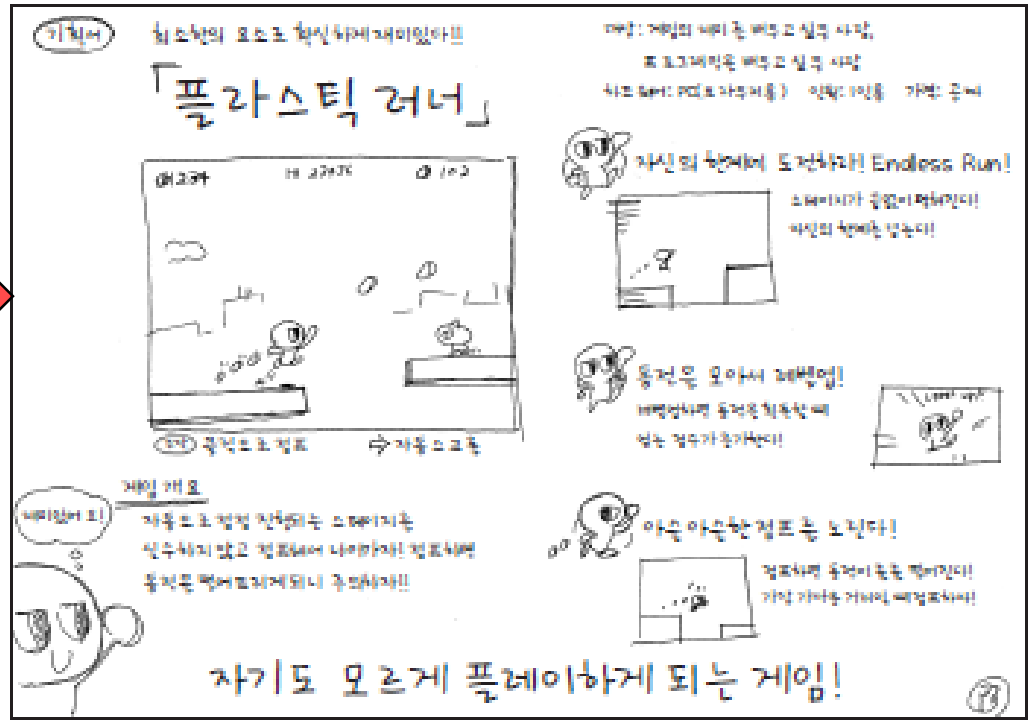
특징 1: 자신의 한계에 도전하라! Endless Run!
스테이지는 끝없이 펼쳐진다! 구멍을 뛰어 넘고 고양이를 뛰어 넘어 자신의 한계를 돌파하라! 플레이어의 속도도 점점 빨라진다.

특징 2: 동전을 모아서 레벨업!
레벨업하면 동전을 획득할 때 점수가 조금씩 증가한다! 계속 동전을 획득해서 최고 점수를 갱신하자!

특징 3: 아슬아슬한 점프를 노린다!
점프하면 모처럼 모은 동전(점수)을 떨어뜨리게 된다! 구멍은 아슬아슬한 간격으로 넘는 게 제일 좋다! 하지만 구멍에 빠질 위험도 높아진다!!

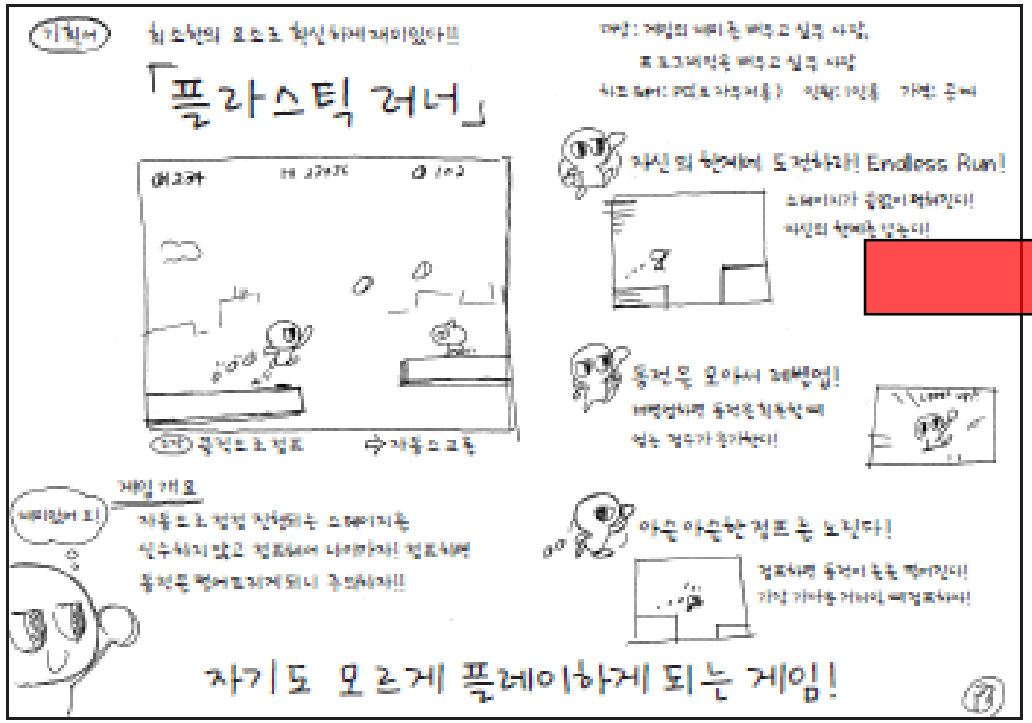
자기도 모르게 플레이하게 되는 게임!!

그림 3-27(b) 손으로 쓴 기획서

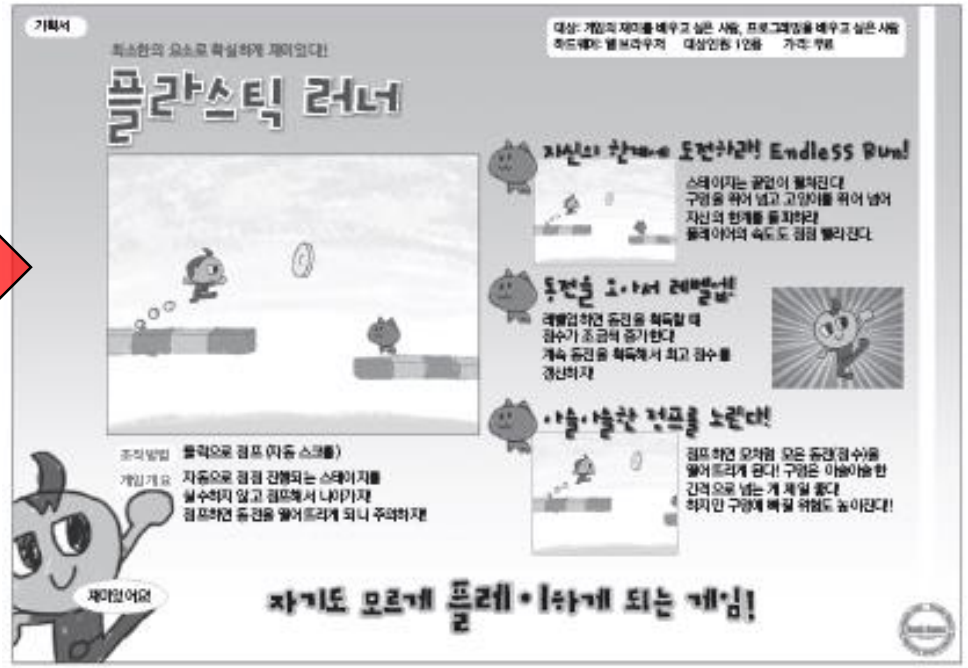


* 기획서 - 깔끔하게 정리하기

♥ 그림 3-27(b) 손으로 쓴 기획서



♥ 그림 3-27(c) 예쁘게 다시 작성한 기획서



유니티
게임 제작 입문

다음 페이지에서 보아요

길벗