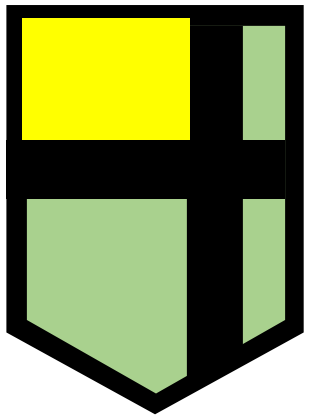
유니티
게임 제작 입문

필드 액션 게임 만들기

길벗

필드 액션 게임 만들기

기획

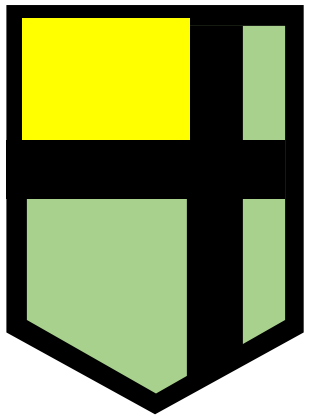


프로그래밍

레벨디자인

필드 액션 게임 만들기

기획



프로그래밍

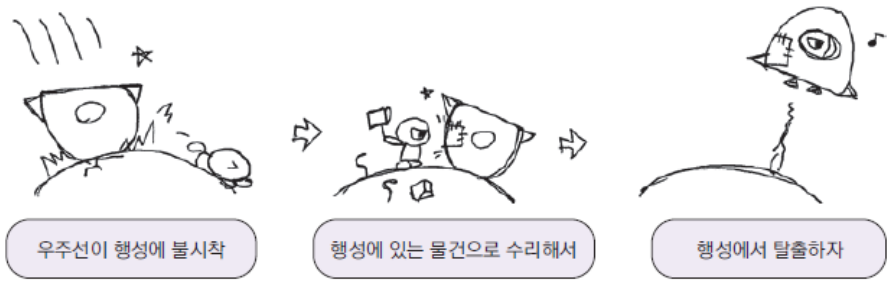
레벨디자인

* 필드 액션 게임 기획

제목 : 플라플라 플래닛

장르 : 필드액션

세계관



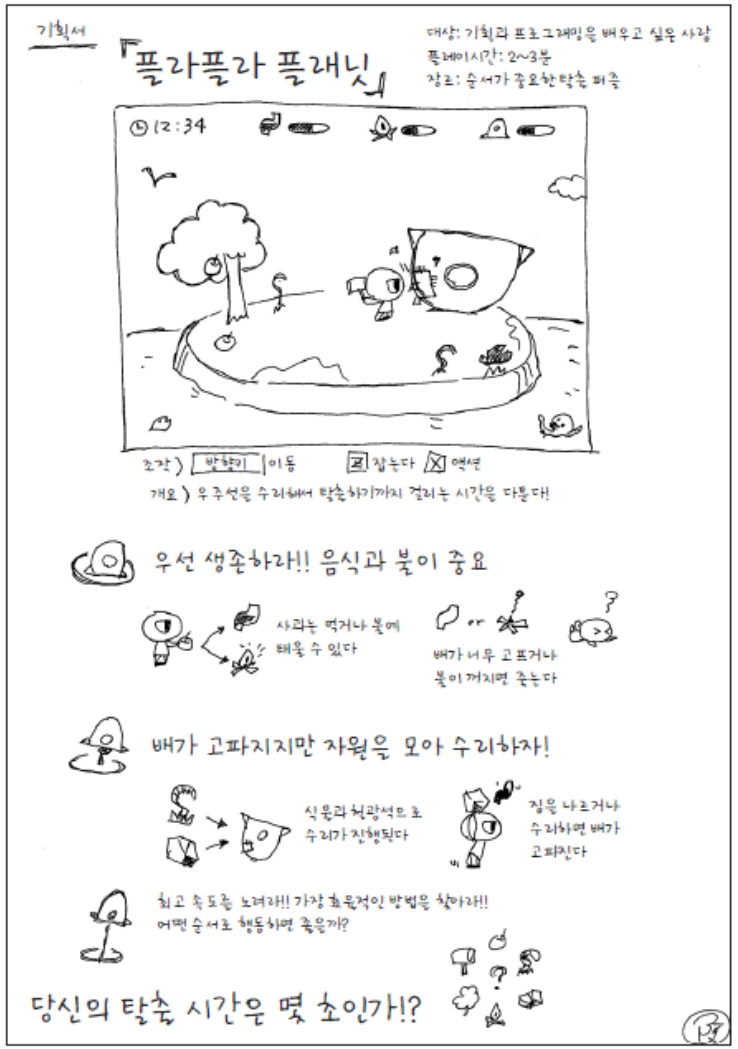
	배	북	수리	움직이면 배고파진다	비고
	↑	↑	X	↓	내버려두면 나무로 성장
	X	↑	↑	↓	어딘가에서 생겨난다
	X	X	↑	↓	

- 게임에 필요한 요소를 설정

  체력, 모닥불의 남은 시간, 우주선의 수리 상태 등

* 필드 액션 게임 기획

한 장짜리 기획서로 만들기



* 사양서 작성

타이틀 화면

게임 화면

* 사양서 작성

그림 12-12 플레이어 캐릭터의 사양서

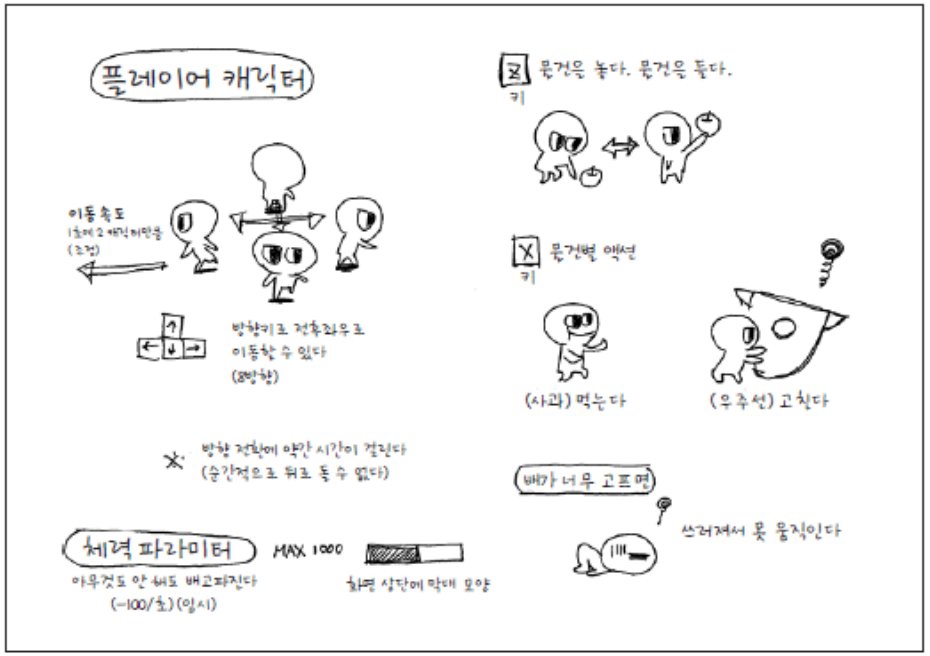
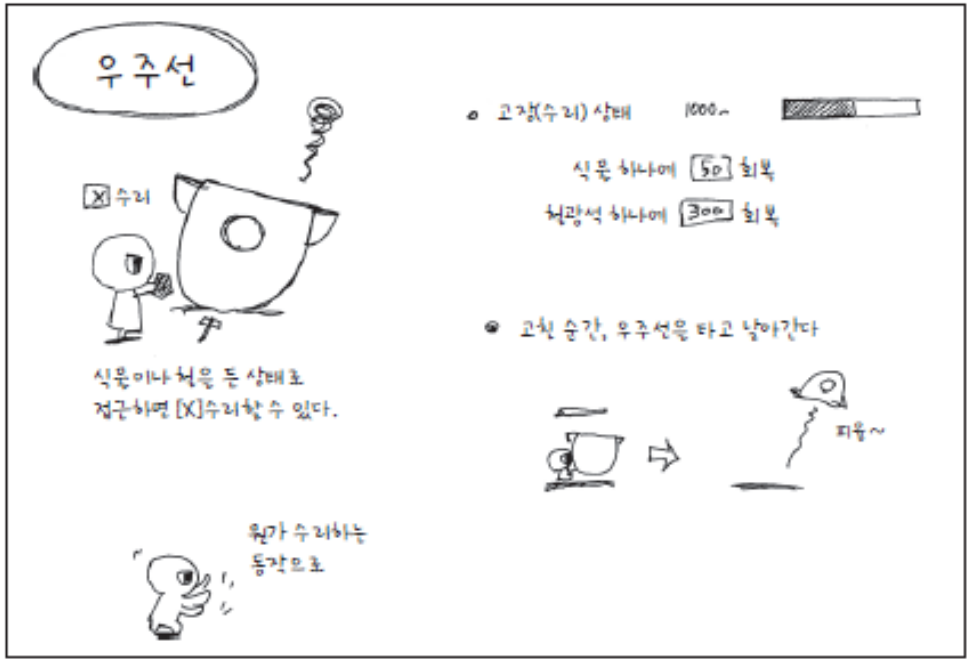


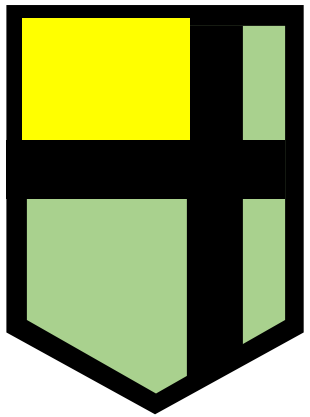
그림 12-15 우주선(로켓)의 사양서



가능한 게임에 필요한 요소들에 대한 사양서를 모두 작성할 것 => 구현이 더 쉬워짐

필드 액션 게임 만들기

기획



프로그래밍

레벨디자인

* 프로그래밍 순서

- 프로그래밍 단계를 정리하는 것은 구현 시 방향을 잡아준다

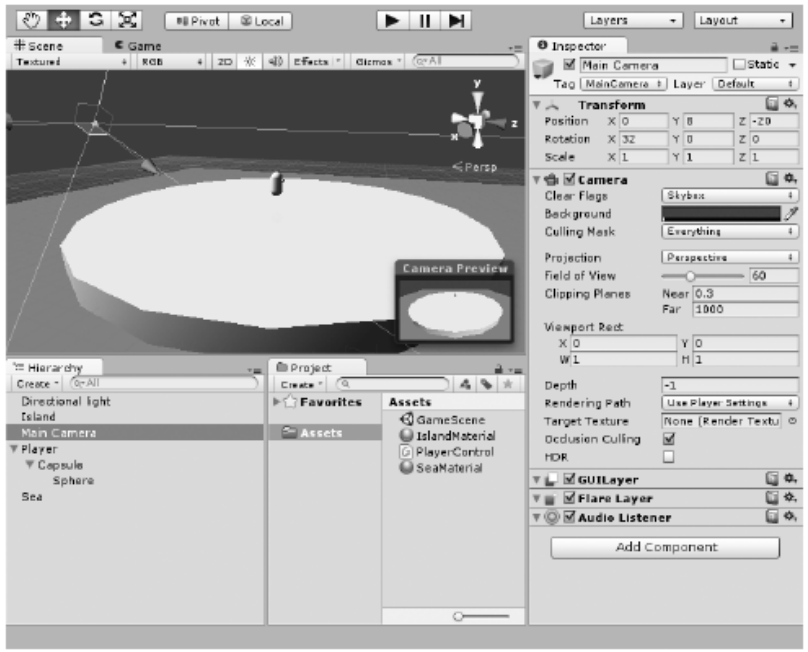
1. 플레이어 캐릭터를 이동시킨다
2. 떨어진 사과를 집어 든다.
3. 사과를 먹는다.
4. 식물과 철광석을 출현시킨다
5. 식물이나 철광석을 사용하여 우주선을 수리한다.

* 준비

- 씬 생성
- 플레이어와 배경
- 카메라 위치 조정

마지막인데, 알아서 척척!

그림 13-3 바다와 섬과 캐릭터가 완성되었다



1. 플레이어 이동

PlayerControl.cs

```
public static float MOVE_AREA_RADIUS = 15.0f;          // 섬의 반지름.
public static float MOVE_SPEED = 5.0f;                 // 이동 속도.

private struct Key {                                   // 키 조작 정보 구조체.
    public bool up;                                    // ↑
    public bool down;                                  // ↓
    public bool right;                                 // →
    public bool left;                                  // ←
    public bool pick;                                  // 줍는다/버린다.
    public bool action;                                // 먹는다/수리한다.
};  // 키 조작 정보를 모아서 다룸

private Key key;                                       // 키 조작 정보를 보관하는 변수.

public enum STEP {                                     // 플레이어의 상태를 나타내는 열거체.
    NONE = -1,                                         // 상태 정보 없음.
    MOVE = 0,                                          // 이동 중.
    REPAIRING,                                         // 수리 중.
    EATING,                                            // 식사 중.
    NUM,                                               // 상태가 몇 종류인지 나타낸다(=3).
};

public STEP step = STEP.NONE;                          // 현재 상태.
public STEP next_step = STEP.NONE;                     // 다음 상태.
public float step_timer = 0.0f;                        // 타이머.


void Start() {
    this.step = STEP.NONE;                             // 현 단계 상태를 초기화.
    this.next_step = STEP.MOVE;                        // 다음 단계 상태를 초기화.
}
```

1. 플레이어 이동

PlayerControl.cs

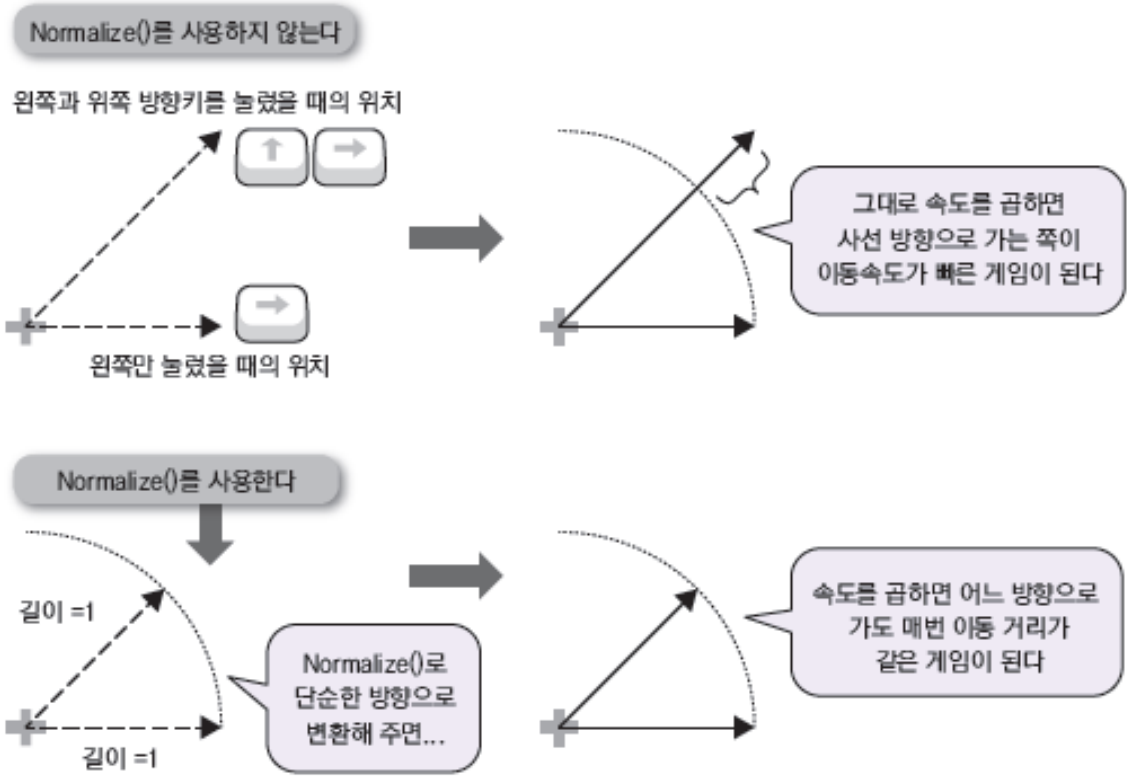
```
private void get_input()        // 키 입력을 받아 key 값 갱신
{
    this.key.up = false;
    this.key.down = false;
    this.key.right = false;
    this.key.left = false;

    // ↑키가 눌렸으면 true를 대입.
    this.key.up |= Input.GetKey(KeyCode.UpArrow);   // 논리연산자를 사용해
    this.key.up |= Input.GetKey(KeyCode.Keypad8);   // 키가 입력 되었는 지 확인

    // ↓키가 눌렸으면 true를 대입.
    this.key.down |= Input.GetKey(KeyCode.DownArrow);
    this.key.down |= Input.GetKey(KeyCode.Keypad2);

    // →키가 눌렸으면 true를 대입.
    this.key.right |= Input.GetKey(KeyCode.RightArrow);
    this.key.right |= Input.GetKey(KeyCode.Keypad6);

    // ←키가 눌렸으면 true를 대입.
    this.key.left |= Input.GetKey(KeyCode.LeftArrow);
    this.key.left |= Input.GetKey(KeyCode.Keypad4);

    // Z키가 눌렸으면 true를 대입.
    this.key.pick = Input.GetKeyDown(KeyCode.Z);
    // X키가 눌렸으면 true를 대입.
    this.key.action = Input.GetKeyDown(KeyCode.X);
}
```

1. 플레이어 이동

PlayerControl.cs

```
private void move_control()                    // 실제로 이동시킨다
{
    Vector3 move_vector = Vector3.zero;        // 이동용 벡터.
    Vector3 position = this.transform.position;// 현재 위치를 보관.
    bool is_moved = false;

    if(this.key.right) {                       // →키가 눌렸으면.
        move_vector += Vector3.right;          // 이동용 벡터를 오른쪽으로 향한다
        is_moved = true;                       // '이동 중' 플래그.
    }
    if(this.key.left) {
        move_vector += Vector3.left;
        is_moved = true;
    }
    if(this.key.up) {
        move_vector += Vector3.forward;
        is_moved = true;
    }
    if(this.key.down) {
        move_vector += Vector3.back;
        is_moved = true;
    }

    move_vector.Normalize();                   // 길이를 1로.
    move_vector *= MOVE_SPEED * Time.deltaTime;// 속도×시간=거리.
    position += move_vector;                   // 위치를 이동.
    position.y = 0.0f;                         // 높이를 0으로 한다.

    // 세계의 중앙에서 갱신한 위치까지의 거리가 섬의 반지름보다 크면.
    if(position.magnitude > MOVE_AREA_RADIUS) {
        position.Normalize();    // 정규화 : 벡터의 길이를 1로 만듦
        position *= MOVE_AREA_RADIUS;          // 위치를 섬의 끝자락에 머물게 한다.
    }

    // 새로 구한 위치(position)의 높이를 현재 높이로 되돌린다.
    position.y = this.transform.position.y;
    // 실제 위치를 새로 구한 위치로 변경한다.
    this.transform.position = position;
    // 이동 벡터의 길이가 0.01보다 큰 경우.
    // = 어느 정도 이상의 이동한 경우.
    if(move_vector.magnitude > 0.01f) {
        // 캐릭터의 방향을 천천히 바꾼다.
        Quaternion q = Quaternion.LookRotation(move_vector, Vector3.up);
        this.transform.rotation =
            Quaternion.Lerp(this.transform.rotation, q, 0.1f);
    }
}
```

1. 플레이어 이동 - Normalize



1. 플레이어 이동

PlayerControl.cs

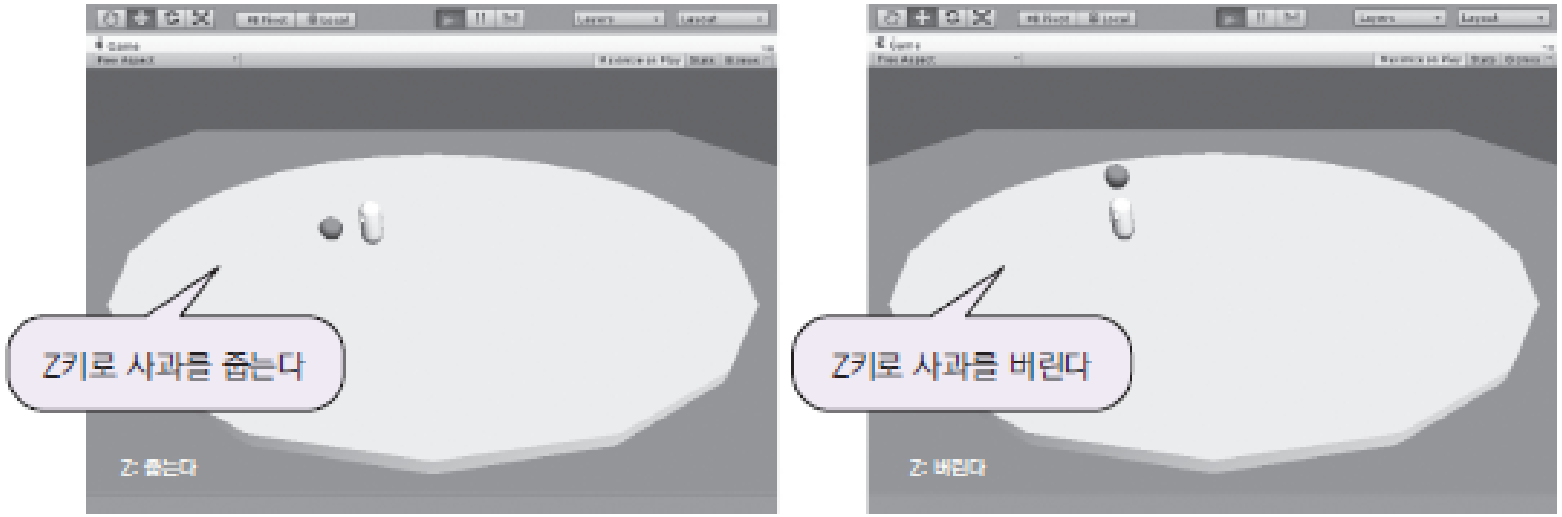
```
void Update() {
    this.get_input();                       // 입력 정보 취득.

    // 상태가 변했을 때----------.
    while(this.next_step != STEP.NONE) {    // 상태가 NONE 이외 = 상태가 변했다.
        this.step = this.next_step;
        this.next_step = STEP.NONE;
        switch(this.step) {
            case STEP.MOVE:
                break;
        }
        this.step_timer = 0.0f;
    }

        // 각 상황에서 반복할 것----------.
        switch(this.step) {     // 아직 플레이어 이동만 구현했으니,
            case STEP.MOVE:     // case도 move만 구현해 놓기로 하자.
                this.move_control();
                break;
        }
}
```

현재 시점에서는 아무것도 하지 않지만 나중에 위치를 알기 쉽도록 작성해둔다

그림 13-7 Player를 조종해보자

2. 사과 집어 들기

그림 13-8 사과를 줍기도 하고 내버려두기도 하자



프로젝트,
게임 오브젝트 준비.

2. 사과 집어 들기

ItemRoot.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;   // List를 사용하기 위해서.

public class Item {
    public enum TYPE {              // 아이템 종류.
        NONE = -1,                  // 없음.
        IRON = 0,                   // 철광석.
        APPLE,                      // 사과.
        PLANT,                      // 식물.
        NUM,                        // 아이템이 몇 종류인지 나타낸다(=3).

    };
};

public class ItemRoot : MonoBehaviour {
    // 아이템의 종류를 Item.TYPE으로 반환하는 메서드.
    public Item.TYPE getItemType(GameObject item_go)
    {
        Item.TYPE type = Item.TYPE.NONE;
        if(item_go != null) {           // 인수로 받은 게임 오브젝트가 비어 있지 않으면.
            switch(item_go.tag) {       // 태그로 분기.
            case "Iron": type = Item.TYPE.IRON; break;
            case "Apple": type = Item.TYPE.APPLE; break;
            case "Plant": type = Item.TYPE.PLANT; break;
            }
        }
        return(type);
    }
}
```

2. 사과 집어 들기

PlayerControl.cs

```
private GameObject closest_item = null;      // 플레이어의 정면에 있는 게임 오브젝트.
private GameObject carried_item = null;      // 플레이어가 들어 올린 게임 오브젝트.
private ItemRoot item_root = null;           // ItemRoot 스크립트를 가짐.
public GUIStyle guistyle;                    // 폰트 스타일.


void Start() {
    this.step = STEP.NONE;                  // 현 단계 상태를 초기화.
    this.next_step = STEP.MOVE;             // 다음 단계 상태를 초기화.
    this.item_root =
        GameObject.Find("GameRoot").GetComponent<ItemRoot>();

    this.guistyle.fontSize = 16;
}


 void Update() {
     ...
     // 각 상황에서 반복할 것----------.
     switch(this.step) {
     case STEP.MOVE:
         this.move_control();
         this.pick_or_drop_control();
         break;
     }
 }
```

2. 사과 집어 들기

PlayerControl.cs

```
void OnTriggerStay(Collider other)
{
    GameObject other_go = other.gameObject;

    // 트리거의 게임 오브젝트 레이어 설정이 Item이라면.
    if(other_go.layer == LayerMask.NameToLayer("Item")) {
        // 아무 것도 주목하고 있지 않으면.
        if(this.closest_item == null) {
            if(this.is_other_in_view(other_go)) {        // 정면에 있으면.
                this.closest_item = other_go;            // 주목한다.
            }
        // 뭔가 주목하고 있으면.
        } else if(this.closest_item == other_go) {
            if(! this.is_other_in_view(other_go)) {      // 정면에 없으면.
                    this.closest_item = null;            // 주목을 그만둔다.
            }
        }
    }

    void OnTriggerExit(Collider other)
    {
        if(this.closest_item == other.gameObject) {
            this.closest_item = null;                    // 주목을 그만둔다.
        }
    }
```

OnTriggerStay( ): 트리거에 걸린 게임 오브젝트가 Item 레이어에 설정되어 있고 플레이어의 정면에 있을 때, 그 게임 오브젝트를 주목하게 한다.

OnTriggerExit( ): 주목을 그만두게 한다.

2. 사과 집어 들기

PlayerControl.cs

```
void OnGUI()
{
    float x = 20.0f;
    float y = Screen.height - 40.0f;

    // 들고 있는 아이템이 있다면.
    if(this.carried_item != null) {
        GUI.Label(new Rect(x, y, 200.0f, 20.0f), "Z:버린다", guistyle);
    } else {
        // 주목하는 아이템이 있다면.
        if(this.closest_item != null) {
            GUI.Label(new Rect(x, y, 200.0f, 20.0f), "Z:줍는다", guistyle);
        }
    }
}
```

```
private void pick_or_drop_control()
{
    do {
        if(! this.key.pick) {                   // '줍기/버리기' 키가 눌리지 않았으면.
            break;                              // 아무것도 하지 않고 메서드 종료.
        }
        if(this.carried_item == null) {         // 들고 있는 아이템이 없고.
            if(this.closest_item == null) {     // 주목 중인 아이템이 없으면.
                break;                          // 아무것도 하지 않고 메서드 종료.
            }
            // 주목 중인 아이템을 들어 올린다.
            this.carried_item = this.closest_item;
            // 들고 있는 아이템을 자신의 자식으로 설정.
            this.carried_item.transform.parent = this.transform;
            // 2.0f 위에 배치(머리 위로 이동).
            this.carried_item.transform.localPosition = Vector3.up * 2.0f;
            // 주목 중인 아이템을 없앤다.
            this.closest_item = null;
        } else {  // 들고 있는 아이템이 있을 경우.
            // 들고 있는 아이템을 약간(1.0f) 앞으로 이동시켜서.
            this.carried_item.transform.localPosition =
                Vector3.forward * 1.0f;
            this.carried_item.transform.parent = null;   // 자식 설정을 해제.
            this.carried_item = null;                    // 들고 있는 아이템을 없앤다.
        }
    } while(false);
}
```

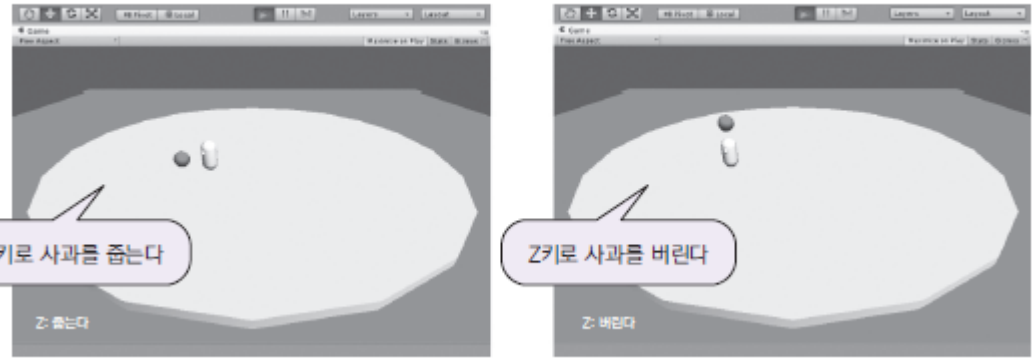
2. 사과 집어 들기

PlayerControl.cs

```
private bool is_other_in_view(GameObject other)
{
    bool ret = false;
    do {
        Vector3 heading =              // 자신이 현재 향하고 있는 방향을 보관.
            this.transform.TransformDirection(Vector3.forward);
        Vector3 to_other =             // 자신 쪽에서 본 아이템의 방향을 보관.
            other.transform.position - this.transform.position;
        heading.y = 0.0f;
        to_other.y = 0.0f;
        heading.Normalize();           // 길이를 1로 하고 방향만 벡터로.
        to_other.Normalize();          // 길이를 1로 하고 방향만 벡터로.
        float dp = Vector3.Dot(heading, to_other);  // 양쪽 벡터의 내적을 취득.
        if(dp < Mathf.Cos(45.0f)) {    // 내적이 45도의 코사인 값 미만이면.
            break;                     // 루프를 빠져나간다.
        }
        ret = true;                    // 내적이 45도의 코사인 값 이상이면 정면에 있다.
    } while(false);
    return(ret);
}
```

```
// 게임 오브젝트가 플레이어 정면에 있는지 판정
// 있으면 true 반환
```

❤ 그림 13-13 사과를 줍거나 버릴 수 있다



3. 사과 먹기

PlayerControl.cs

```
void Update() {          // 굵은 글씨부분 추가
    this.get_input();
    this.step_timer += Time.deltaTime;
    float eat_time = 2.0f;                    // 사과는 2초에 걸쳐 먹는다.

    // 상태를 변화시킨다----------------------.
    if(this.next_step == STEP.NONE) {         // 다음 예정이 없으면.
        switch(this.step) {
            case STEP.MOVE:                   // '이동 중' 상태의 처리.
                do {
                    if(! this.key.action) {   // 액션키가 눌려 있지 않다.
                        break;                // 루프 탈출.
                    }
                    if(this.carried_item != null) {
                        // 가지고 있는 아이템 판별.
                        Item.TYPE carried_item_type =
                            this.item_root.getItemType(this.carried_item);

                        switch(carried_item_type) {
                            case Item.TYPE.APPLE:          // 사과라면.
                            case Item.TYPE.PLANT:          // 식물이라면.
                            // '식사 중' 상태로 이행.
                            this.next_step = STEP.EATING;
                            break;
                        }
                    }
                } while(false);
                break;
            case STEP.EATING:                 // '식사 중' 상태의 처리.
            if(this.step_timer > eat_time) {  // 2초 대기.
                this.next_step = STEP.MOVE;   // '이동' 상태로 이행.
            }
            break;
            }
    }

    // 상태가 변화했을 때------------.
    while(this.next_step != STEP.NONE) {
        this.step = this.next_step;
        this.next_step = STEP.NONE;

        switch(this.step) {
            case STEP.MOVE:
                break;
            case STEP.EATING:  // '식사 중' 상태의 처리.
                if(this.carried_item != null) {
                    // 가지고 있던 아이템을 폐기.
                    GameObject.Destroy(this.carried_item);
                    this.carried_item = null;
                }
                break;
        }
        this.step_timer = 0.0f;
    }
```

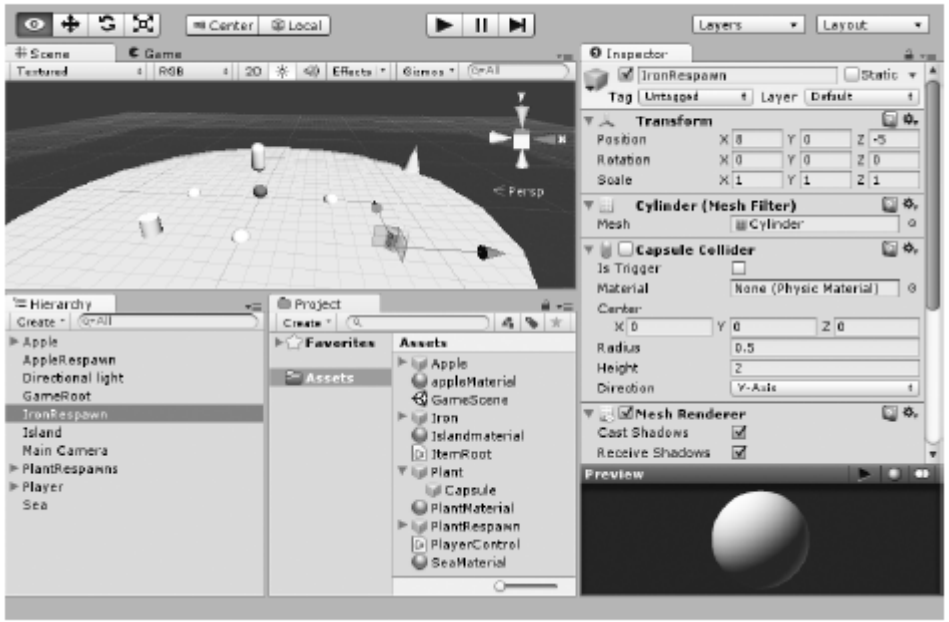
3. 사과 먹기

PlayerControl.cs

```
void OnGUI() {
    float x = 20.0f;          // 굵은 글씨부분 추가
    float y = Screen.height - 40.0f;
    if(this.carried_item != null) {
        GUI.Label(new Rect(x, y, 200.0f, 20.0f), "Z:버린다", guistyle);
        GUI.Label(new Rect(x+100.0f, y, 200.0f, 20.0f),
            "x:먹는다", guistyle);
    } else {
        if(this.closest_item != null) {
            GUI.Label(new Rect(x, y, 200.0f, 20.0f), "Z:줍는다", guistyle);
        }

        …

   switch(this.step) {
       case STEP.EATING:
           GUI.Label(new Rect(x, y, 200.0f, 20.0f),
               "우걱우걱우물우물...", guistyle);
           break;
   }
}
```

4. 철광석과 식물 출현

그림 13-15 철광석, 식물, 사과의 출현 지점을 만들었다



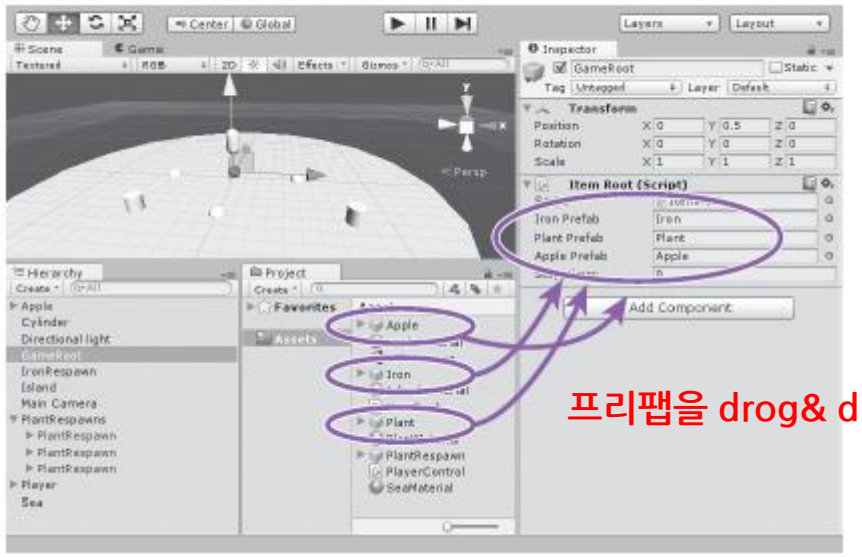
오브젝트 메뉴를 이용해 식물과 철광석을 만든다.

4. 철광석과 식물 출현

ItemRoot.cs

```
public GameObject ironPrefab = null;                    // 프리팹 Iron.
public GameObject plantPrefab = null;                   // 프리팹 Plant.
public GameObject applePrefab = null;                   // 프리팹 Apple.
protected List<Vector3> respawn_points;                 // 출현 지점 List.

public float step_timer = 0.0f;
public static float RESPAWN_TIME_APPLE = 20.0f;         // 사과 출현 시간 상수.
public static float RESPAWN_TIME_IRON = 12.0f;          // 철광석 출현 시간 상수.
public static float RESPAWN_TIME_PLANT = 6.0f;          // 식물 출현 시간 상수.
private float respawn_timer_apple = 0.0f;               // 사과의 출현 시간.
private float respawn_timer_iron = 0.0f;                // 철광석의 출현 시간.
private float respawn_timer_plant = 0.0f;               // 식물의 출현 시간.
```

그림 13-16 프로그램에서 사용할 프리팹을 설정하자



4. 철광석과 식물 출현

ItemRoot.cs

```
public void respawnIron()
{
    // 철광석 프리팹을 인스턴스화.
    GameObject go =
        GameObject.Instantiate(this.ironPrefab) as GameObject;
    // 철광석의 출현 지점을 취득.
     Vector3 pos = GameObject.Find("IronRespawn").transform.position;
     // 출현 위치를 조정.
     pos.y = 1.0f;
     pos.x += Random.Range(-1.0f, 1.0f);
     pos.z += Random.Range(-1.0f, 1.0f);
     // 철광석의 위치를 이동.
     go.transform.position = pos;
 }
```

```
public void respawnApple()
{
    // 사과 프리팹을 인스턴스화.
    GameObject go =
        GameObject.Instantiate(this.applePrefab) as GameObject;
    // 사과의 출현 지점을 취득.
    Vector3 pos = GameObject.Find("AppleRespawn").transform.position;
    // 출현 위치를 조정.
    pos.y = 1.0f;
    pos.x += Random.Range(-1.0f, 1.0f);
    pos.z += Random.Range(-1.0f, 1.0f);
    // 사과의 위치를 이동.
    go.transform.position = pos;
}
```

4. 철광석과 식물 출현

ItemRoot.cs

```
public void respawnIron()
{
    // 철광석 프리팹을 인스턴스화.
    GameObject go =
        GameObject.Instantiate(this.ironPrefab) as GameObject;
    // 철광석의 출현 지점을 취득.
     Vector3 pos = GameObject.Find("IronRespawn").transform.position;
     // 출현 위치를 조정.
     pos.y = 1.0f;
     pos.x += Random.Range(-1.0f, 1.0f);
     pos.z += Random.Range(-1.0f, 1.0f);
     // 철광석의 위치를 이동.
     go.transform.position = pos;
 }
```

```
public void respawnApple()
{
    // 사과 프리팹을 인스턴스화.
    GameObject go =
        GameObject.Instantiate(this.applePrefab) as GameObject;
    // 사과의 출현 지점을 취득.
    Vector3 pos = GameObject.Find("AppleRespawn").transform.position;
    // 출현 위치를 조정.
    pos.y = 1.0f;
    pos.x += Random.Range(-1.0f, 1.0f);
    pos.z += Random.Range(-1.0f, 1.0f);
    // 사과의 위치를 이동.
    go.transform.position = pos;
}
```

4. 철광석과 식물 출현

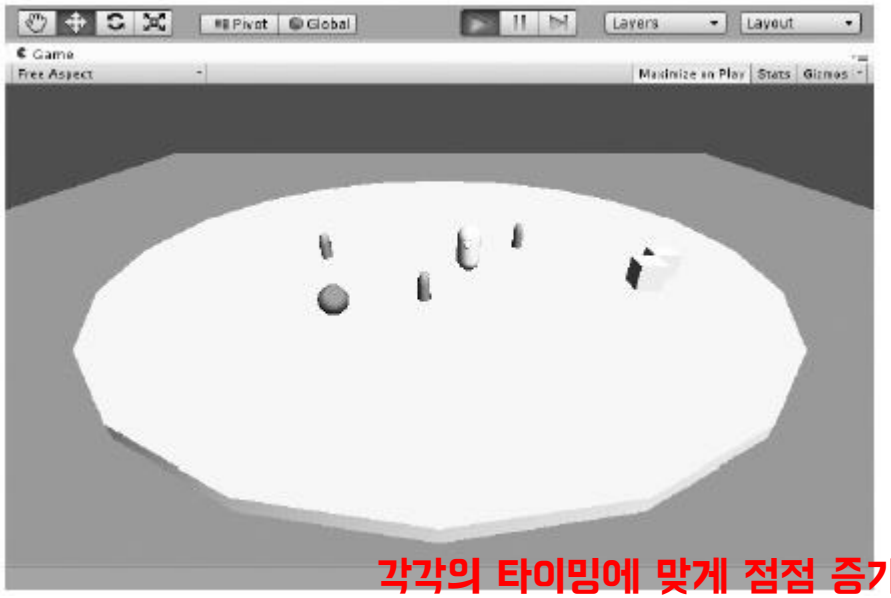
ItemRoot.cs

```
public void respawnPlant()
{
    if(this.respawn_points.Count > 0) {  // List가 비어 있지 않으면.
        // 식물 프리팹을 인스턴스화.
        GameObject go =
            GameObject.Instantiate(this.plantPrefab) as GameObject;
        // 식물의 출현 지점을 랜덤하게 취득.
        int n = Random.Range(0, this.respawn_points.Count);
        Vector3 pos = this.respawn_points[n];
        // 출현 위치를 조정.
        pos.y = 1.0f;
        pos.x += Random.Range(-1.0f, 1.0f);
        pos.z += Random.Range(-1.0f, 1.0f);
        // 식물의 위치를 이동.
        go.transform.position = pos;
    }
}

void Start() // 초기화
{
    // 메모리 영역 확보.
    this.respawn_points = new List<Vector3>();
    // "PlantRespawn" 태그가 붙은 모든 오브젝트를 배열에 저장.
    GameObject[] respawns =
        GameObject.FindGameObjectsWithTag("PlantRespawn");
    // 배열 respawns 내의 각 GameObject를 순서대로 처리.
    foreach(GameObject go in respawns) {
        // 렌더러 획득.
        MeshRenderer renderer = go.GetComponentInChildren<MeshRenderer>();
        if(renderer != null) {          // 렌더러가 존재하면.
            renderer.enabled = false;  // 그 렌더러를 보이지 않게.
        }
        // 출현 지점 List에 위치 정보를 추가.
        this.respawn_points.Add(go.transform.position);
    }
    // 사과의 출현 지점을 취득하고 렌더러를 보이지 않게.
    GameObject applerespawn = GameObject.Find("AppleRespawn");
    applerespawn.GetComponent<MeshRenderer>().enabled = false;
    // 철광석의 출현 지점을 취득하고 렌더러를 보이지 않게.
    GameObject ironrespawn = GameObject.Find("IronRespawn");
    ironrespawn.GetComponent<MeshRenderer>().enabled = false;

    this.respawnIron();                 // 철광석을 하나 생성.
    this.respawnPlant();                // 식물을 하나 생성.
}
```

4. 철광석과 식물 출현

ItemRoot.cs

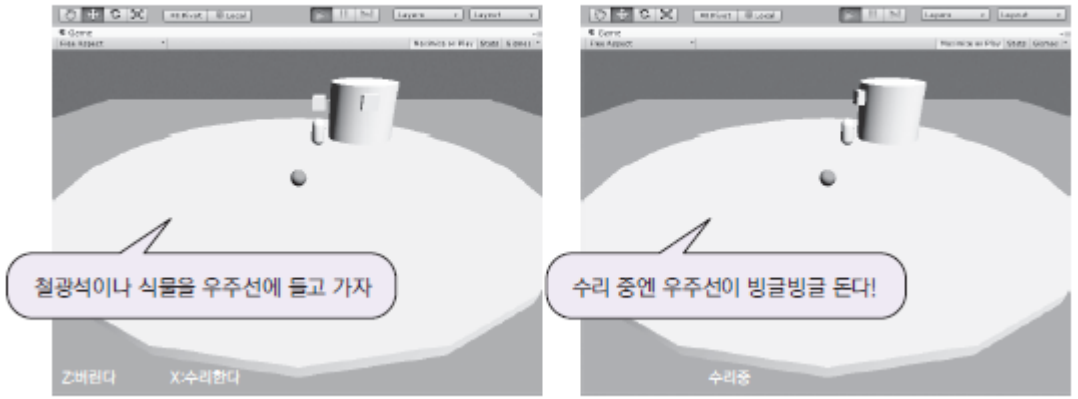
```
void Update() {
    respawn_timer_apple += Time.deltaTime;
    respawn_timer_iron += Time.deltaTime;
    respawn_timer_plant += Time.deltaTime;

    if(respawn_timer_apple > RESPAWN_TIME_APPLE) {
        respawn_timer_apple = 0.0f;
        this.respawnApple();        // 사과를 출현시킨다.
    }
    if(respawn_timer_iron > RESPAWN_TIME_IRON) {
        respawn_timer_iron = 0.0f;
        this.respawnIron();         // 철광석을 출현시킨다.
    }
    if(respawn_timer_plant > RESPAWN_TIME_PLANT) {
        respawn_timer_plant = 0.0f;
        this.respawnPlant();        // 식물을 출현시킨다.
    }
}
```



각각의 타이밍에 맞게 점점 증가할 것이다.
확인했나? (Y/N)

5. 우주선 수리

그림 13-18 우주선을 수리할 수 있게 해보자.



EventRoot.cs

```
public class Event {        // 이벤트의 종류.
    public enum TYPE {
        NONE = -1,          // 없음.
        ROCKET = 0,         // 우주선 수리.
        NUM,                // 이벤트가 몇 종류 있는지 나타낸다(=1).
    };
};


public Event.TYPE getEventType(GameObject event_go)
{
    Event.TYPE type = Event.TYPE.NONE;

    if(event_go != null) {      // 인수의 게임 오브젝트가 비어 있지 않으면.
       if(event_go.tag == "Rocket") {
           type = Event.TYPE.ROCKET;
       }
    }
    return(type); // 이벤트 값을 반환
}
```

5. 우주선 수리

EventRoot.cs

```
public bool isEventIgnitable(Item.TYPE carried_item, GameObject event_go)
{
    bool ret = false;                          // 철광석이나 식물을 든 상태에서
    Event.TYPE type = Event.TYPE.NONE;         // 우주선에 접촉했는지 확인

    if(event_go != null) {
        type = this.getEventType(event_go);          // 이벤트 타입을 가져온다.
    }

    switch(type) {
        case Event.TYPE.ROCKET:
            if(carried_item == Item.TYPE.IRON) {     // 가지고 있는 것이 철광석이라면.
                ret = true;                          // '이벤트할 수 있어요!'라고 응답한다.
            }
            if(carried_item == Item.TYPE.PLANT) {    // 가지고 있는 것이 식물이라면.
                ret = true;                          // '이벤트할 수 있어요!'라고 응답한다.
            }
            break;
    }
    return(ret);
}
```

```
public string getIgnitableMessage(GameObject event_go)
{
    string message = "";
    Event.TYPE type = Event.TYPE.NONE;
    if(event_go != null) {
        type = this.getEventType(event_go);
    }
    switch(type) {
        case Event.TYPE.ROCKET:
            message = "수리한다";
            break;
    }
    return(message);
}
```

5. 우주선 수리

playerControl.cs

```
private GameObject closest_event = null;      // 주목하고 있는 이벤트를 저장.
private EventRoot event_root = null;          // EventRoot 클래스를 사용하기 위한 변수.
private GameObject rocket_model = null;       // 우주선의 모델을 사용하기 위한 변수.

void Start() {
    this.step = STEP.NONE;              // 현 단계 상태를 초기화.
    this.next_step = STEP.MOVE;         // 다음 단계 상태를 초기화.
    this.item_root = GameObject.Find("GameRoot").GetComponent<ItemRoot>();
    this.guistyle.fontSize = 16;

    this.event_root =
        GameObject.Find("GameRoot").GetComponent<EventRoot>();

    this.rocket_model = GameObject.Find("rocket").transform.FindChild(
        "rocket_model").gameObject;
}
```

수정!

```
void OnTriggerStay(Collider other)
{
    GameObject other_go = other.gameObject;

    if(other_go.layer == LayerMask.NameToLayer("Item")) {

            ...


    // 트리거의 게임 오브젝트 레이어 설정이 Event라면.
    } else if(other_go.layer == LayerMask.NameToLayer("Event")) {
        // 아무것도 주목하고 있지 않으면.
        if(this.closest_event == null) {
            if(this.is_other_in_view(other_go)) {        // 정면에 있으면.
                this.closest_event = other_go;           // 주목한다.
            }

        // 무엇인가 주목하고 있으면.
        } else if(this.closest_event == other_go) {
            if(!this.is_other_in_view(other_go)) {       // 정면에 없으면.
                this.closest_event = null;               // 주목을 그만둔다.
            }
        }
    }
}
```

5. 우주선 수리

playerControl.cs

```
private bool is_event_ignitable()
{
    bool ret = false;
    do {
        if(this.closest_event == null) {        // 주목 이벤트가 없으면.
            break;                              // false를 반환한다.
        }
        // 들고 있는 아이템 종류를 가져온다.
        Item.TYPE carried_item_type =
            this.item_root.getItemType(this.carried_item);

        // 들고 있는 아이템 종류와 주목하는 이벤트의 종류에서.
        // 이벤트가 가능한지 판정하고 이벤트 불가라면 false를 반환한다.
        if(! this.event_root.isEventIgnitable(
            carried_item_type, this.closest_event)) {
            break;
        }

        ret = true; // 여기까지 오면 이벤트를 시작할 수 있다고 판정!.
    } while(false);
    return(ret);
}
```

```
void Update() {
    this.get_input();
    this.step_timer += Time.deltaTime;
    float eat_time = 2.0f;
    float repair_time = 2.0f;       // 수리에 걸리는 시간도 2초.
    // 상태를 변화시킨다----------------------.
    if(this.next_step == STEP.NONE) {
        switch(this.step) {
        case STEP.MOVE:
            do {
                if(! this.key.action) {
                    break;
                }
                // 주목하는 이벤트가 있을 때.
                if(this.closest_event != null) {
                    if(! this.is_event_ignitable()) {       // 이벤트를 시작할 수 없으면.
                        break;                              // 아무 것도 하지 않는다.
                    }
                    // 이벤트 종류를 가져온다.
                    Event.TYPE ignitable_event =
                        this.event_root.getEventType(this.closest_event);
```

5. 우주선 수리

playerControl.cs

```
            switch(ignitable_event) {
                case Event.TYPE.ROCKET:
                    // 이벤트의 종류가 ROCKET이면.
                    // REPAIRING(수리) 상태로 이행.
                    this.next_step = STEP.REPAIRING;
                    break;
            }
            break;
        }
        if(this.carried_item != null) {
            Item.TYPE carried_item_type =
                this.item_root.getItemType(this.carried_item);
            switch(carried_item_type) {
                case Item.TYPE.APPLE:
                    this.next_step = STEP.EATING;
                    break;
            }
        }
    } while(false);
    break;
```

```
        case STEP.EATING:
            if(this.step_timer > eat_time) {
                this.next_step = STEP.MOVE;
            }
            break;
        case STEP.REPAIRING:                        // '수리 중' 상태의 처리.
            if(this.step_timer > repair_time) {     // 2초 대기.
                this.next_step = STEP.MOVE;         // '이동' 상태로 이행.
            }
            break;
        }
    }

    // 상태가 변화했을 때------------.
    while(this.next_step != STEP.NONE) {    // 상태가 NONE 이외 = 상태가 변했다.
        this.step = this.next_step;
        this.next_step = STEP.NONE;
        switch(this.step) {
            case STEP.MOVE:
                break;
            case STEP.EATING:               // '식사 중' 상태의 처리.
```

// 아이템을 사용했다는 뜻으로 간주, 아이템 삭제

5. 우주선 수리

playerControl.cs

```
            switch(ignitable_event) {
                case Event.TYPE.ROCKET:
                    // 이벤트의 종류가 ROCKET이면.
                    // REPAIRING(수리) 상태로 이행.
                    this.next_step = STEP.REPAIRING;
                    break;
            }
            break;
        }
        if(this.carried_item != null) {
            Item.TYPE carried_item_type =
                this.item_root.getItemType(this.carried_item);
            switch(carried_item_type) {
                case Item.TYPE.APPLE:
                    this.next_step = STEP.EATING;
                    break;
            }
        }
    } while(false);
    break;
```

```
        case STEP.EATING:
            if(this.step_timer > eat_time) {
                this.next_step = STEP.MOVE;
            }
            break;
        case STEP.REPAIRING:                        // '수리 중' 상태의 처리.
            if(this.step_timer > repair_time) {     // 2초 대기.
                this.next_step = STEP.MOVE;         // '이동' 상태로 이행.
            }
            break;
        }
    }
```

// 아이템을 사용했다는 뜻으로 간주, 아이템 삭제

```
    case STEP.REPAIRING:                // '수리 중'이 되면.
        if(this.carried_item != null) {
            // 가지고 있는 아이템 삭제.
            GameObject.Destroy(this.carried_item);
            this.carried_item = null;
            this.closest_item = null;
        }
        break;
```

5. 우주선 수리

playerControl.cs

```
void  Update() {
    ...

        // 각 상황에서 반복할 것----------.
        switch(this.step) {
        case STEP.MOVE:
            this.move_control();
            this.pick_or_drop_control();
            break;
        case STEP.REPAIRING:
            // 우주선을 회전시킨다.
            this.rocket_model.transform.localRotation *=
                Quaternion.AngleAxis(
                    360.0f / 10.0f * Time.deltaTime, Vector3.up);
            break;
        }
    }
```

```
void OnGUI() {          // 화면에 표시할 내용 추가
    float x = 20.0f;
    float y = Screen.height - 40.0f;

    if(this.carried_item != null) {
        GUI.Label(new Rect(x, y, 200.0f, 20.0f), "Z:버린다", guistyle);
        do {
            if(this.is_event_ignitable()) {
                break;
            }
            if(item_root.getItemType(this.carried_item) == Item.TYPE.IRON) {
                break;
            }
            GUI.Label(new Rect(x+100.0f, y, 200.0f, 20.0f), "x:먹는다", guistyle);
        }while(false);
    } else {
        if(this.closest_item != null) {
            GUI.Label(new Rect(x, y, 200.0f, 20.0f), "Z:줍는다", guistyle);
        }
    }
```

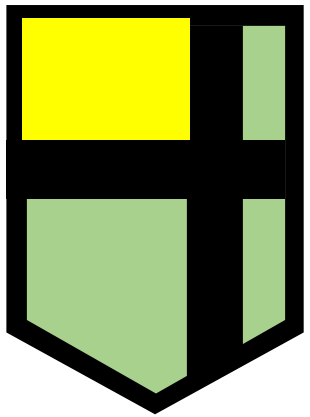
5. 우주선 수리

playerControl.cs

```
    switch(this.step) {
        case STEP.EATING:
            GUI.Label(new Rect(x, y, 200.0f, 20.0f), "우걱우걱우물우물...", guistyle);
            break;
        case STEP.REPAIRING:
            GUI.Label(new Rect(x+200.0f, y, 200.0f, 20.0f), "수리중", guistyle);
            break;
    }

    if(this.is_event_ignitable()) {  // 이벤트가 시작 가능한 경우.
        // 이벤트용 메시지를 취득.
        string message =
            this.event_root.getIgnitableMessage(this.closest_event);
        GUI.Label(new Rect(x+200.0f, y, 200.0f, 20.0f), "X:" + message, guistyle);
    }
}
```

필드 액션 게임 만들기

기획



프로그래밍

레벨디자인

1. 경과시간 만들기

GameStatus.cs

```
public bool isGameClear()
{
    bool is_clear = false;
    if(this.repairment >= 1.0f) {       // 수리 정도가 100% 이상이면.
        is_clear = true;                // 클리어!.
    }
    return(is_clear);
}

public bool isGameOver()
{
    bool is_over = false;
    if(this.satiety <= 0.0f) {          // 체력이 0 이하라면.
        is_over = true;                 // 게임 오버.
    }
    return(is_over);
}
```

1. 경과시간 만들기

SceneControl.cs

```
private GameStatus game_status = null;
private PlayerControl player_control = null;

public enum STEP {              // 게임 상태.
        NONE = -1,              // 상태 정보 없음.
        PLAY = 0,               // 플레이 중.
        CLEAR,                  // 클리어 상태.
        GAMEOVER,               // 게임 오버 상태.
        NUM,                    // 상태가 몇 종류인지 나타낸다(=3).
};

public STEP step = STEP.NONE;           // 현대 단계.
public STEP next_step = STEP.NONE;      // 다음 단계.
public float step_timer = 0.0f;         // 타이머.
private float clear_time = 0.0f;        // 클리어 시간.
public GUIStyle guistyle;               // 폰트 스타일.

void Start()
{
    this.game_status = this.gameObject.GetComponent<GameStatus>();
    this.player_control =
    GameObject.Find("Player").GetComponent<PlayerControl>();
this.step = STEP.PLAY;
this.next_step = STEP.PLAY;
this.guistyle.fontSize = 64;
}
```

```
void Update()  // 게임오버인지 판정해서 게임 상태 전환
{
    this.step_timer += Time.deltaTime;
    if(this.next_step == STEP.NONE) {
        switch(this.step) {
        case STEP.PLAY:
            if(this.game_status.isGameClear()) {
                // 클리어 상태로 이동.
                this.next_step = STEP.CLEAR;
            }
            if(this.game_status.isGameOver()) {
                // 게임 오버 상태로 이동.
                this.next_step = STEP.GAMEOVER;
            }
            break;

        // 클리어 시 및 게임 오버 시의 처리.
        case STEP.CLEAR:
        case STEP.GAMEOVER:
            if(Input.GetMouseButtonDown(0)) {
                // 마우스 버튼이 눌렸으면 GameScene을 다시 읽는다.
                Application.LoadLevel("GameScene");
            }
            break;
        }
    }
```

1. 경과시간 만들기

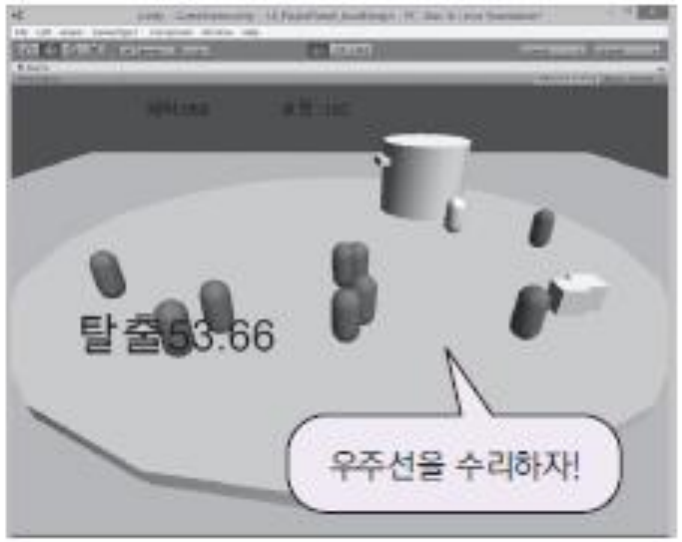
SceneControl.cs

```
        while(this.next_step != STEP.NONE) {
            this.step = this.next_step;
            this.next_step = STEP.NONE;
            switch(this.step) {
            case STEP.CLEAR:
            // PlayerControl을 제어 불가로.
                this.player_control.enabled = false;
            // 현재의 경과 시간으로 클리어 시간을 갱신.
                this.clear_time = this.step_timer;
                break;

          case STEP.GAMEOVER:
              // PlayerControl을 제어 불가로.
              this.player_control.enabled = false;
              break;
          }
          this.step_timer = 0.0f;
      }
  }
```

```
void OnGUI()
{
    float pos_x = Screen.width * 0.1f;
    float pos_y = Screen.height * 0.5f;
    switch(this.step) {
    case STEP.PLAY:
        GUI.color = Color.black;
        GUI.Label(new Rect(pos_x, pos_y, 200, 20),   // 경과 시간을 표시.
            this.step_timer.ToString("0.00"), guistyle);
        break;
    case STEP.CLEAR:
        GUI.color = Color.black;
        // 클리어 메시지와 클리어 시간 표시.
        GUI.Label(new Rect(pos_x, pos_y, 200, 20),
            "탈출" + this.clear_time.ToString("0.00"), guistyle);
        break;
    case STEP.GAMEOVER:
        GUI.color = Color.black;
        // 게임 오버 메시지를 표시.
        GUI.Label(new Rect(pos_x, pos_y, 200, 20),
            "게임 오버", guistyle);
        break;
    }
}
```

1. 경과시간 만들기

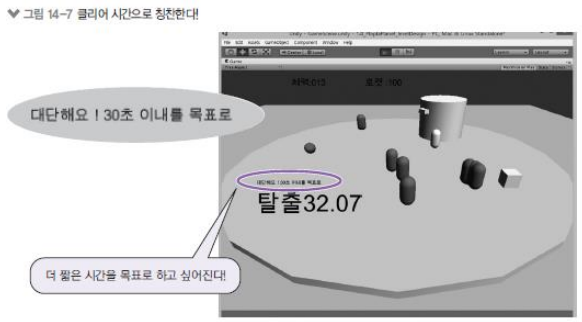
그림 14-3 게임을 클리어할 수 있다!



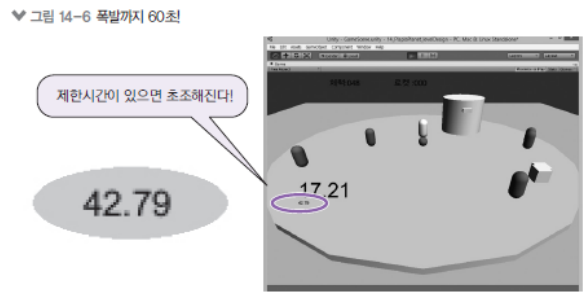
조작감을 위해 playControl 스크립트에 속도를 추가해도 좋다.

2. 게임 요소 추가

보상



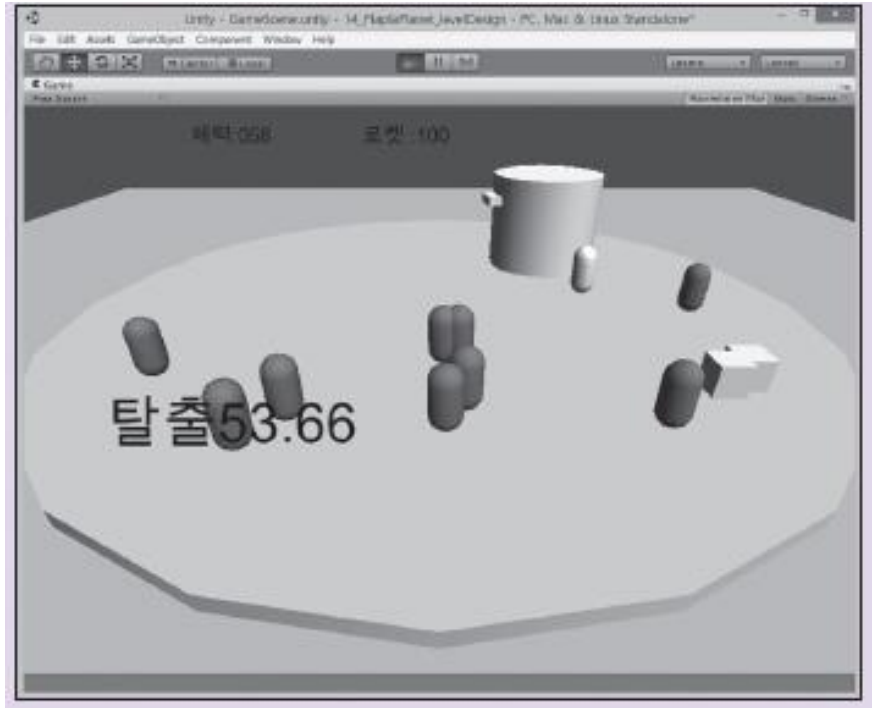
제한시간

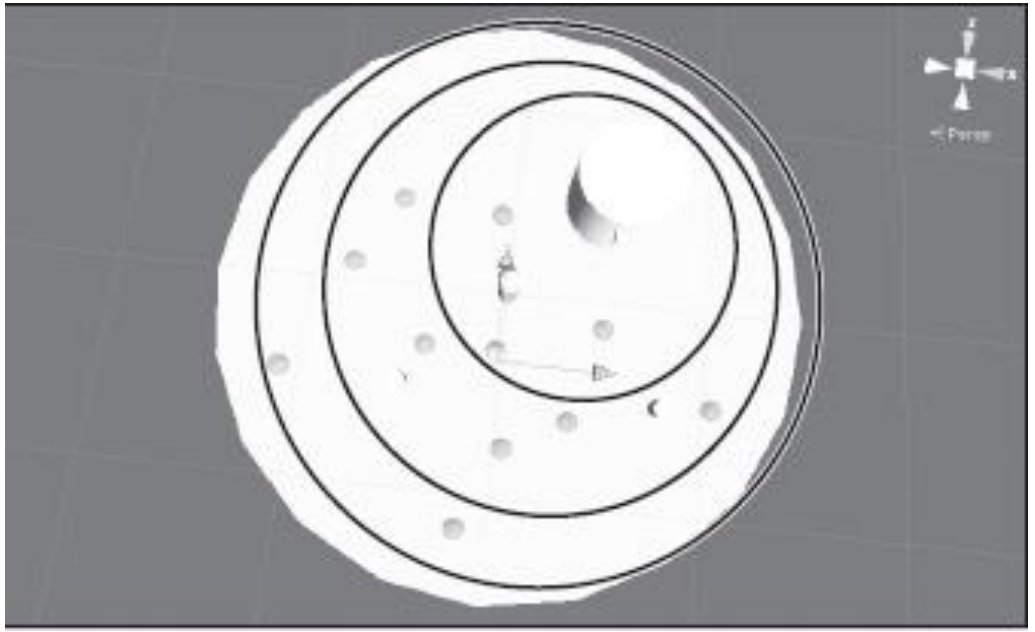


게임의 재미

요소들을 추가해보자.

* 완성





교재 내용 말고도 아이디어를 하나씩 추가해서
자신만의 플라플라 플래닛을 만들어 보세요.

유니티 게임 제작 입문

지금까지 수고하셨습니다.

길벗