출처 : Introduction to Game Design, Prototyping and Development 2ed, Jeremy Gibson BOND

Introduction to Game Design 2ed

03 장 Space SHMUP

유니티 엔진 버전 2019.4.20f1

서진석 jsseo@deu.ac.kr

2021-9-25

섹션 01 : 프로젝트 생성 및 씬 설정

1. 새로운 3D 프로젝트 생성 : Space SHMUP
   - 씬 이름 변경 : Scene_0
2. 리소스 다운로드 : 03_Space_SHMUP_Starter_with_Resource.zip
   - 압축해제
   - 03_Space_SHMUP_Starter_with_Resource.unitypackage 임포트
3. 하이어라키 창 / Directional Light 선택
   - 위치 : 0, 20, 0
4. 하이어라키 창 / MainCamera 선택
   - 위치 : 0, 0, -10
   - Clear Flags : Solid Color
   - Background : 0, 0, 0, 255
   - Projection : Orthographic
   - Size : 40
   - Near Clipping Plane : 0.3
   - Far Clipping Plane : 100
5. 게임 창
   - 화면 비율 : 3:4

섹션 02 : Player 게임 오브젝트

1. 프로젝트 창 / _Prefabs 폴더 생성
2. _Prefab 폴더 / HeroPrefab 을 하이어라키 창으로 드래그하여 게임 오브젝트 생성, 이름 변경 : Player
   - Rigidbody 컴포넌트 추가
     - Use Gravity : 체크 해제, Is Kinematic : 체크
     - Constraints / Freeze Position : Z 체크, Freeze Rotation : X, Y, Z 체크
3. 프로젝트 창 / Assets 에 폴더 생성 : __Scripts
4. __Scripts 폴더에 스크립트 생성 : Player, Player 게임오브젝트에 어태치

```
public class Player : MonoBehaviour
{
    static public Player S;

    [Header("Set in Inspector")]
    public float speed = 30;
    public float rollMult = -45;
    public float pitchMult = 30;

    [Header("Set Dynamically")]
    public float shieldLevel = 1;

    void Awake()
    {
        if (S == null)
        {
            S = this;
        }
        else
        {
            Debug.LogError("Player.Awake() - Attempted to assign second Player.S!");
        }
    }

    void Update()
    {
        float xAxis = Input.GetAxis("Horizontal");
        float yAxis = Input.GetAxis("Vertical");

        Vector3 pos = transform.position;
        pos.x += xAxis * speed * Time.deltaTime;
        pos.y += yAxis * speed * Time.deltaTime;
        transform.position = pos;
        transform.rotation = Quaternion.Euler(yAxis * pitchMult, xAxis * rollMult, 0);
    }
}
```

5. 플레이
   - WASD 키나 방향키로 플레이어 움직임 가능
6. 하이어라키 창 / 3D Object / Quad 생성
   - 이름 변경 : Shield
   - Player 의 자식 게임 오브젝트로 이동
   - 위치 : 0, 0, 0
   - 크기 : 8, 8, 8
   - Mesh Collider 컴포넌트 제거
   - Sphere Collider 컴포넌트 추가
7. 프로젝트 창 / _Materials 폴더에 Material 생성
   - 이름 변경 : Mat_Shield
   - 위 5 번에서 만든 Shield 게임 오브젝트에 적용
   - Shader : ProtoTools / UnlitAplha 로 변경
   - None (Texture) 박스에 있는 Select 버튼 클릭 : Shields 선택
   - Main Color : 0, 255, 0, 2555
   - Tiling X : 0.2
   - Offset X : 0.4
8. 스크립트 생성 : Shield

```
public class Shield : MonoBehaviour
{
    [Header("Set in Inspector")]
    public float rotationsPerSecond = 0.1f;

    [Header("Set Dynamically")]
    public int levelShown = 0;

    Material mat;

    void Start()
    {
        mat = GetComponent<Renderer>().material;
    }

    void Update()
    {
        int currLevel = Mathf.FloorToInt(Player.S.shieldLevel);

        if (levelShown != currLevel)
        {
            levelShown = currLevel;
            mat.mainTextureOffset = new Vector2(0.2f * levelShown, 0);
        }
```

```
        float rZ = -(rotationsPerSecond * Time.time * 360) % 360f;
        transform.rotation = Quaternion.Euler(0, 0, rZ);
    }
}
```

- 위 5 번에서 만든 Shield 게임 오브젝트에 어태치

9. 플레이
   - 플레이어 주변에 방어막이 생김

섹션 03 : 스크린 경계 검사

1. 새로운 스크립트 생성 : BoundsCheck
   - Player 게임 오브젝트에 어태치

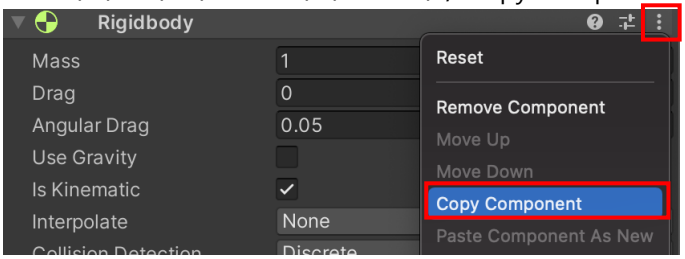
```
public class BoundsCheck : MonoBehaviour
{
    [Header("Set in Inspector")]
    public float radius = 1f;

    [Header("Set Dynamically")]
    public float camWidth;
    public float camHeight;

    void Awake()
    {
        camHeight = Camera.main.orthographicSize;
        camWidth = camHeight * Camera.main.aspect;
    }

    void LateUpdate()
    {
        Vector3 pos = transform.position;

        if (pos.x > camWidth - radius)
        {
            pos.x = camWidth - radius;
        }

        if (pos.x < -camWidth + radius)
        {
            pos.x = -camWidth + radius;
        }

        if (pos.y > camHeight - radius)
        {
            pos.y = camHeight - radius;
        }

        if (pos.y < -camHeight + radius)
        {
            pos.y = -camHeight + radius;
        }

        transform.position = pos;
    }
```

```
    void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Vector3 boundSize = new Vector3(camWidth * 2, camHeight * 2, 0.1f);
        Gizmos.DrawWireCube(Vector3.zero, boundSize);
    }
}
```

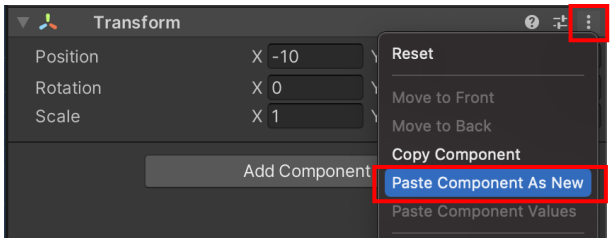
2. 플레이
   - 플레이어가 화면 경계를 벗어나지 못함
   - 씬 창에 Gizmos 함수에 의해 화면 경계가 그려짐
3. Player 게임 오브젝트 / 인스펙터 창 / Bounds Check 컴포넌트
   - Radius : 4 로 변경

섹션 04 : Enemy 프리팹 설정

1. 프로젝트 창 / _Prefabs / EnemyPrefab0 선택
   - 인스펙터 창 / Open Prefab 버튼 클릭
   - Rigidbody 컴포넌트 추가
     - ■ Use Gravity : 체크 해제
     - ■ Is Kinematic : 체크
     - ■ Constraints / Freeze Position : Z 체크
     - ■ Constraints / Freeze Rotation : X, Y, Z 체크
   - Rigidbody 컴포넌트 타이틀의 세로 점 아이콘 클릭 / Copy Component 메뉴 아이템 선택



   - 저장
   - EnemyPrefab1 선택 / 인스펙터 창
     - ■ Transform 컴포넌트 타이틀의 세로 점 아이콘 클릭 / Paste Component As New 선택



   - 다른 모든 EnemyPrefab 도 위와 마찬가지로 Paste Component As New 적용
2. EnemyPrefab0 프리팹을 하이어라키 창으로 드래그하여 게임 오브젝트 생성
   - 이름 변경 : Enemy0
   - 위치 : 0, 20, 0

섹션 05 : Enemy 스크립트

1. 스크립트 생성 : Enemy
   - 하이어라키 창에 있는 Enemy0 게임 오브젝트에 어태치

```
public class Enemy : MonoBehaviour
{
    [Header("Set in Inspedctor: Enemy")]
    public float speed = 10f;
    public float health = 4;
    public int score = 100;

    public Vector3 pos
    {
        get
        {
            return transform.position;
        }
        set
        {
            transform.position = value;
        }
    }

    void Update()
    {
        Move();
    }

    public virtual void Move()
    {
        Vector3 tempPos = pos;
        tempPos.y -= speed * Time.deltaTime;
        pos = tempPos;
    }
}
```

   - 플레이 : Enemy0 가 아래로 내려옴
2. 하이어라키 창 / Enemy0 게임 오브젝트 선택
   - BoundsCheck 컴포넌트 추가
   - 플레이 : Enemy0 가 화면 경계에 가면 멈춤
3. Enmy0 / 인스펙터 창 / Radius : -2.5
   - 플레이 : Enemy0 가 화면을 벗어나자마자 멈춤

섹션 06 : BoundsCheck 수정

1. BoundsCheck 스크립트 수정
   - 변수 추가

```
public float radius = 1f;
public bool keepOnScreen = true;

[Header("Set Dynamically")]
public bool isOnScreen = true;
```

   - LateUpdate 메서드 수정

```
void LateUpdate()
{
    Vector3 pos = transform.position;
    isOnScreen = true;

    if (pos.x > camWidth - radius)
    {
        pos.x = camWidth - radius;
        isOnScreen = false;
    }

    if (pos.x < -camWidth + radius)
    {
        pos.x = -camWidth + radius;
        isOnScreen = false;
    }

    if (pos.y > camHeight - radius)
    {
        pos.y = camHeight - radius;
        isOnScreen = false;
    }

    if (pos.y < -camHeight + radius)
    {
        pos.y = -camHeight + radius;
        isOnScreen = false;
    }

    if (keepOnScreen && !isOnScreen)
    {
        transform.position = pos;
        isOnScreen = true;
    }
}
```

2. 하이어라키 창 / Enemy0 게임 오브젝트 선택 / 인스펙터 창
   - Bounds Check 컴포넌트
     - ■ Keep On Screen 체크 해제
3. Enemy 스크립트 수정
   - 변수 추가

```
public int score = 100;

private BoundsCheck boundsCheck;
```

   - Awake 메서드 추가

```
void Awake()
{
    boundsCheck = GetComponent<BoundsCheck>();
}
```

   - Update 메서드 수정

```
void Update()
{
    Move();

    if (boundsCheck != null && !boundsCheck.isOnScreen)
    {
        if (pos.y < -boundsCheck.camHeight - boundsCheck.radius)
        {
            Destroy(gameObject);
        }
    }
}
```

4. 플레이
   - Enemy0 가 화면을 벗어나면 제거됨

섹션 07 : BoundsCheck 개선

1. BoundsCheck 스크립트 수정
   - 변수 추가

```
public float camHeight;

[HideInInspector]
public bool offRight, offLeft, offUp, offDown;
```

   - LateUpdate 메서드 수정

```
void LateUpdate()
{
    Vector3 pos = transform.position;
    isOnScreen = true;
    offRight = offLeft = offUp = offDown = false;

    if (pos.x > camWidth - radius)
    {
        pos.x = camWidth - radius;
        isOnScreen = false;
        offRight = true;
    }

    if (pos.x < -camWidth + radius)
    {
        pos.x = -camWidth + radius;
        isOnScreen = false;
        offLeft = true;
    }

    if (pos.y > camHeight - radius)
    {
        pos.y = camHeight - radius;
        isOnScreen = false;
        offUp = true;
    }

    if (pos.y < -camHeight + radius)
    {
        pos.y = -camHeight + radius;
        isOnScreen = false;
        offDown = true;
    }

    isOnScreen = !(offRight || offLeft || offUp || offDown);
```

```
        if (keepOnScreen && !isOnScreen)
        {
            transform.position = pos;
            isOnScreen = true;
            offRight = offLeft = offUp = offDown = false;
        }
    }
```

2. Enemy 스크립트의 Update 메서드 수정

```
    void Update()
    {
        Move();

        if (boundsCheck != null && boundsCheck.offDown)
        {
                Destroy(gameObject);
        }
    }
```

3. 플레이
   - 이전과 같이 동작하는지 확인

섹션 08 : Enemy 스폰

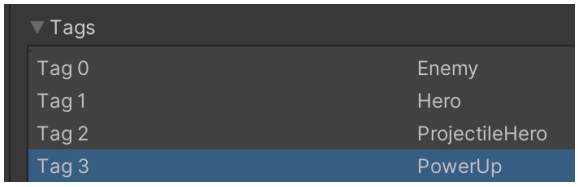
1. MainCamera 선택
   - BoundsCheck 컴포넌트 어태치, Keep On Screen : 체크 해제
   - 새로운 스크립트를 생성하고 어태치 : Main

```
using UnityEngine.SceneManagement;

public class Main : MonoBehaviour
{
    static public Main S;

    [Header("Set in Inspector")]
    public GameObject[] prefabEnemies;
    public float enemySpawnPerSecond = 0.5f;
    public float enemyDefaultPadding = 1.5f;

    private BoundsCheck bndCheck;

    void Awake()
    {
        S = this;
        bndCheck = GetComponent<BoundsCheck>();

        Invoke("SpawnEnemy", 1f / enemySpawnPerSecond);
    }

    public void SpawnEnemy()
    {
        int ndx = Random.Range(0, prefabEnemies.Length);
        GameObject go = Instantiate<GameObject>(prefabEnemies[ndx]);

        float enemyPadding = enemyDefaultPadding;
        if (go.GetComponent<BoundsCheck>() != null)
        {
            enemyPadding = Mathf.Abs(go.GetComponent<BoundsCheck>().radius);
        }

        Vector3 pos = Vector3.zero;
        float xMin = -bndCheck.camWidth + enemyPadding;
        float xMax = bndCheck.camWidth - enemyPadding;

        pos.x = Random.Range(xMin, xMax);
        pos.y = bndCheck.camHeight + enemyPadding;
        go.transform.position = pos;

        Invoke("SpawnEnemy", 1f / enemySpawnPerSecond);
    }
}
```

2. 하이어라키 창에 있는 Enemy0 선택 / 인스펙터 창
   - Prefab / Overrides 버튼 클릭 / Apply All 클릭
3. 하이어라키 창에 있는 Enemy0 삭제
4. MainCamera 게임 오브젝트 선택 / 인스펙터 창
   - Main 컴포넌트 / Prefab Enemies
     - Size : 1
     - Element 0 : EnemyPrefab0
5. 플레이
   - Enemy 가 무작위 위치에서 생성됨

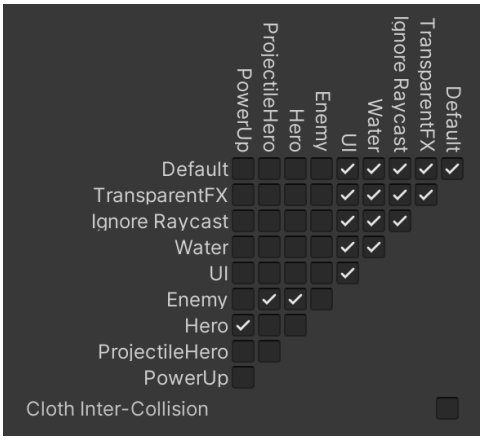
섹션 09 : 태그와 레이어

1. 메뉴 / Edit / Project Settigns...
   - Tags and Layers 섹션
     - Tags 설정



     - Layers 설정

   - Physics 섹션
     - Layer Collision Matrix 설정



2. 하이어라키 창 / Player 게임 오브젝트 선택 / 인스펙터 창
   - 레이어 설정 : Hero
     - Yes, change children 선택
   - 태그 설정 : Hero
3. 프로젝트 창 / 모든 Enemy 프리팹 선택
   - 레이어 설정 : Enemy
     - Yes, change children 선택
   - 태그 설정 : Enemy

섹션 10 : Enemy 와 Player 의 충돌 처리

1. Player 의 자식 오브젝트인 Shield 게임 오브젝트 선택
   - Sphere Collider / Is Trigger : 체크
2. Player 스크립트에 OnTriggerEnter 메서드 추가

```
void OnTriggerEnter(Collider other)
{
    print("Triggered: " + other.gameObject.name);
}
```

3. 플레이
   - Enemy 와 충돌하면 충돌하는 대상의 이름이 2 개가 출력됨(Cockpit 과 Wing)
4. Player 스크립트의 OnTriggerEnter 메서드 수정

```
void OnTriggerEnter(Collider other)
{
    Transform rootT = other.gameObject.transform.root;
    GameObject go = rootT.gameObject;
    print("Triggered: " + go);
}
```

5. 플레이
   - Enemy 게임 오브젝트 이름이 출력됨
   - 한 Enemy 와 충돌해도 메시지는 2 번 출력됨
6. Player 스크립트 수정
   - 변수 추가

```
public float shieldLevel = 1;

private GameObject lastTriggerGo = null;
```

   - OnTriggerEnter 메서드 수정

```
void OnTriggerEnter(Collider other)
{
    Transform rootT = other.gameObject.transform.root;
    GameObject go = rootT.gameObject;
    //print("Triggered: " + go);

    if (go == lastTriggerGo)
    {
        return;
    }

    lastTriggerGo = go;
```

```
        if (go.tag == "Enemy")
        {
            shieldLevel--;
            Destroy(go);
        }
        else
        {
            print("Triggered by non-Enemy: " + go.name);
        }
    }
```

7. 플레이
   - Enemy 가 Plyaer 와 충돌하면 사라지고, Player 의 Shield 가 변함
8. Player 스크립트 수정
   - 변수 추가

```
    [Header("Set Dynamically")]
    [SerializeField]
    private float _shieldLevel = 1;
    //public float shieldLevel = 1;
```

   - shieldLevel 프로퍼티 추가

```
    public float shieldLevel
    {
        get
        {
            return _shieldLevel;
        }
        set
        {
            _shieldLevel = Mathf.Min(value, 4);
            if (value < 0)
            {
                Destroy(this.gameObject);
            }
        }
    }
```

9. 플레이
   - Player 의 shieldLevel 이 0 인 상태에서 Enemy 와 충돌하면 Player 가 사라짐

섹션 11 : 게임 재시작

1. Player 스크립트 수정
   - 변수 추가

```
public float pitchMult = 30;
public float gameRestartDelay = 2f;
```

   - shieldLevel 프로퍼티 수정

```
if (value < 0)
{
    Destroy(this.gameObject);
    Main.S.DelayedRestart(gameRestartDelay);
}
```

2. Main 스크립트 수정
   - 다음 2 개의 메서드 추가

```
public void DelayedRestart(float delay)
{
    Invoke("Restart", delay);
}

public void Restart()
{
    SceneManager.LoadScene("Scene_0");
}
```

3. 플레이
   - Player 가 사망하면 게임이 다시 시작됨
4. 재시작시 어두운 조명 문제
   - 메뉴 / Window / Rendering / Lighting Settings
   - Auto Generates : 체크 해제
   - Generate Lighting 버튼 클릭

섹션 12 : 발사체 프리팹

1. 하이어라키 창 / 3D Object / Cube 생성
   - 이름 변경 : ProjectileHero
   - 크기 : 0.25, 1. 0.5
   - 태그와 레이어 설정 : ProjectilleHero
   - Rigidbody 컴포넌트 추가
     - Use Gravity : 체크 해제
     - Collision Detection : Continuous
     - Constraints Freeze Position : Z 체크
     - Constraints Freeze Rotation : X, Y, Z 체크
   - Box Collider
     - Size Z : 10
   - BoundsCheck 컴포넌트 어태치
     - Keep On Screen : 체크 해제
     - Radius : -1
2. 프로젝트 창 / Material 생성 : Mat_Projectile
   - Shader 변경 : ProtoTools / UnlitAlpha
   - ProjectileHero 게임 오브젝트에 적용
3. 스크립트 생성 : Projectile
   - ProjectileHero 게임 오브젝트에 어태치
4. ProjectileHero 게임 오브젝트를 프로젝트 창 / _Prefabs 폴더로 드래그 하여 프리팹 생성
5. 하이어라키 창에 있는 ProjectileHero 게임 오브젝트 삭제

섹션 13 : Player 발사

1. Player 스크립트 수정
   - 변수 추가

```
public float gameRestartDelay = 2f;
public GameObject projectilePrefab;
public float projectileSpeed = 40;
```

   - TempFire 메서드 추가

```
void TempFire()
{
    GameObject projGo = Instantiate<GameObject>(projectilePrefab);
    projGo.transform.position = transform.position;
    Rigidbody rigidB = projGo.GetComponent<Rigidbody>();
    rigidB.velocity = Vector3.up * projectileSpeed;
}
```

   - Update 메서드 수정

```
    transform.rotation = Quaternion.Euler(yAxis * pitchMult, xAxis * rollMult, 0);

    if (Input.GetKeyDown(KeyCode.Space))
    {
        TempFire();
    }
}
```

2. Player 게임 오브젝트 / 인스펙터 창
   - Player 컴포넌트 / Projectile Prefab : ProjectileHero
3. 플레이 : 스페이스바 키를 누르면 Projectile 이 발사됨
4. Projectile 스크립트 수정

```
public class Projectile : MonoBehaviour
{
    private BoundsCheck boundsCheck;

    void Awake()
    {
        boundsCheck = GetComponent<BoundsCheck>();
    }

    void Update()
    {
        if (boundsCheck.offUp)
        {
            Destroy(gameObject);
        }
    }
}
```

5. 플레이
   - 화면 위로 벗어난 Projectile 이 소멸됨
6. Enemy 스크립트에 OnCollisionEnter 메서드 추가

```
void OnCollisionEnter(Collision collision)
{
    GameObject otherGO = collision.gameObject;
    if (otherGO.tag == "ProjectileHero")
    {
        Destroy(otherGO);
        Destroy(gameObject);
    } else {
        print("Enemy hit by non-ProjectileHero: " + otherGO.name);
    }
}
```

7. 플레이
   - Projectile 과 Enemy 가 충돌하면 두 게임 오브젝트 모두 사라짐

섹션 14 : Enemy 클래스 추가 : Enemy_1

1. 새로운 스크립트 생성 : Enemy_1
   - EnemyPrefab1 프리팹에 어태치

```
public class Enemy_1 : Enemy
{
    [Header("Set in Inspector: Enemy_1")]
    public float waveFrequency = 2;
    public float waveWidth = 4;
    public float waveRotY = 45;

    private float x0;
    private float birthTime;

    void Start()
    {
        x0 = pos.x;
        birthTime = Time.time;
    }

    public override void Move()
    {
        Vector3 tempPos = pos;
        float age = Time.time - birthTime;
        float theta = Mathf.PI * 2 * age / waveFrequency;
        float sin = Mathf.Sin(theta);
        tempPos.x = x0 + waveWidth * sin;
        pos = tempPos;

        Vector3 rot = new Vector3(0, sin * waveRotY, 0);
        this.transform.rotation = Quaternion.Euler(rot);

        base.Move();
    }
}
```

2. EnemyPrefab0 로부터 BoundsCheck 컴포넌를 복사하여(Copy Component) EnemyPrefab1 에도 어태치(Paste Component As New)
3. MainCamera / 인스펙터 창 / Main 컴포넌트
   - Prefab Enemies / Element 0 : EnemyPrefab1 으로 변경
4. 플레이
   - EnemyPrefab1 의 인스턴스가 생성됨

섹션 15 : Enemy 클래스 추가 : Enemy_2

1. Enemy 스크립트의 boundsCheck 변수 선언 수정

```
protected BoundsCheck boundsCheck;
```

2. 새로운 스크립트 생성 : Enemy_2
   - EnemyPrefab2 프리팹에 어태치

```
public class Enemy_2 : Enemy
{
    [Header("Set in Inspector: Enemy_2")]
    public float sinEccentricity = 0.6f;
    public float lifeTime = 10;

    [Header("Set Dynamically: Enemy_2")]
    public Vector3 p0;
    public Vector3 p1;
    public float birthTime;

    void Start()
    {
        p0 = Vector3.zero;
        p0.x = -boundsCheck.camWidth - boundsCheck.radius;
        p0.y = Random.Range(-boundsCheck.camHeight, boundsCheck.camHeight);

        p1 = Vector3.zero;
        p1.x = boundsCheck.camWidth + boundsCheck.radius;
        p1.y = Random.Range(-boundsCheck.camHeight, boundsCheck.camHeight);

        if (Random.value > 0.5f)
        {
            p0.x *= -1;
            p1.x *= -1;
        }

        birthTime = Time.time;
    }

    public override void Move()
    {
        float u = (Time.time - birthTime) / lifeTime;
        if (u > 1)
        {
            Destroy(this.gameObject);
            return;
        }

        u = u + sinEccentricity * Mathf.Sin(u * Mathf.PI * 2);
        pos = (1 - u) * p0 + u * p1;
```

```
    }
}
```

3. EnemyPrefab0 로부터 BoundsCheck 컴포넌를 복사하여(Copy Component) EnemyPrefab2 에도 어태치(Paste Component As New)
   - Radius = 3
4. MainCamera / 인스펙터 창 / Main 컴포넌트
   - Prefab Enemies / Element 0 : EnemyPrefab2 으로 변경
5. 플레이
   - EnemyPrefab2 의 인스턴스가 생성됨

섹션 16 : Enemy 클래스 추가 : Enemy_3

1. 새로운 스크립트 생성 : Enemy_3
   - EnemyPrefab3 프리팹에 어태치

```
public class Enemy_3 : Enemy
{
    [Header("Set in Inspector: Enemy_3")]
    public float lifeTime = 5;

    [Header("Set Dynamically: Enemy_3")]
    public Vector3[] points;
    public float birthTime;

    void Start()
    {
        points = new Vector3[3];
        points[0] = pos;

        float xMin = -boundsCheck.camWidth + boundsCheck.radius;
        float xMax = boundsCheck.camWidth - boundsCheck.radius;

        Vector3 v;
        v = Vector3.zero;
        v.x = Random.Range(xMin, xMax);
        v.y = -boundsCheck.camHeight * Random.Range(2f, 2.75f);
        points[1] = v;

        v = Vector3.zero;
        v.y = pos.y;
        v.x = Random.Range(xMax, xMax);
        points[2] = v;

        birthTime = Time.time;
      }

    public override void Move()
    {
        float u = (Time.time - birthTime) / lifeTime;

        if (u > 1)
        {
            Destroy(this.gameObject);
            return;
        }

        Vector3 p01, p12;
```

```
        p01 = (1 - u) * points[0] + u * points[1];
        p12 = (1 - u) * points[1] + u * points[2];
        pos = (1 - u) * p01 + u * p12;
    }
}
```

2. EnemyPrefab0 로부터 BoundsCheck 컴포넌를 복사하여(Copy Component) EnemyPrefab3 에도 어태치(Paste Component As New)
   - Radius = 2.5
3. MainCamera / 인스펙터 창 / Main 컴포넌트
   - Prefab Enemies / Element 0 : EnemyPrefab3 으로 변경
4. 플레이
   - EnemyPrefab3 의 인스턴스가 생성됨
5. Enemy_3 스크립트의 Move 메서드에 코드 추가

```
        Vector3 p01, p12;
        u = u - 0.2f * Mathf.Sin(u * Mathf.PI * 2);
        p01 = (1 - u) * points[0] + u * points[1];
        p12 = (1 - u) * points[1] + u * points[2];
        pos = (1 - u) * p01 + u * p12;
    }
```

6. 플레이
   - 완화(Easing)된 속도로 움직임

섹션 17 : 무기 타입과 정의

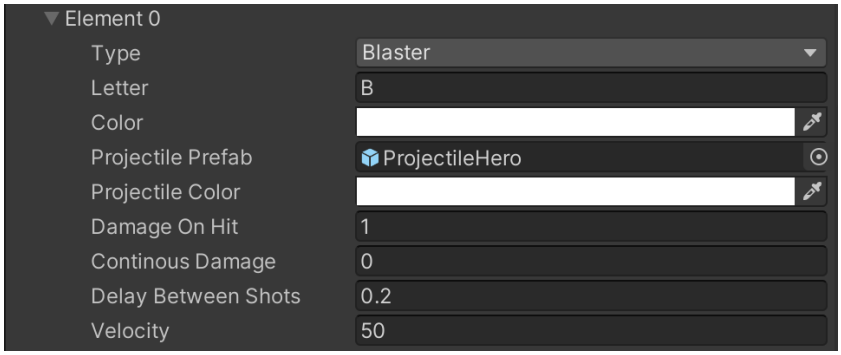
1. 새로운 스크립트 생성 : Weapon

```
public enum WeaponType
{
    none,
    blaster,
    spread,
    shield
}

[System.Serializable]
public class WeaponDefinition
{
    public WeaponType type = WeaponType.none;
    public string letter;
    public Color color = Color.white;
    public GameObject projectilePrefab;
    public Color projectileColor = Color.white;
    public float damageOnHit = 1;
    public float continousDamage = 0;
    public float delayBetweenShots = 0;
    public float velocity = 20;
}

public class Weapon : MonoBehaviour
{
```

2. Main 스크립트에 변수 추가

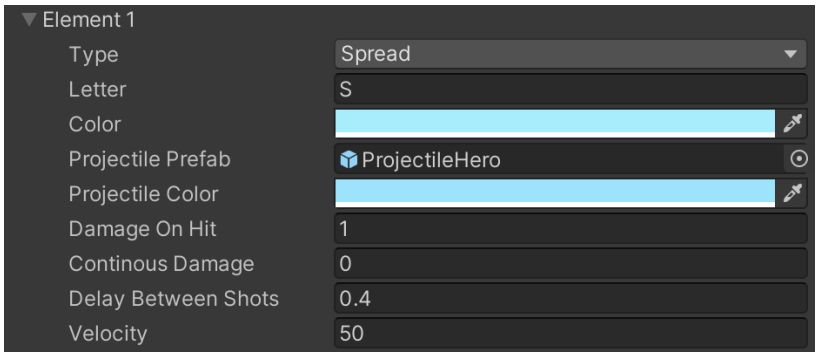
```
    public float enemyDefaultPadding = 1.5f;
    public WeaponDefinition[] weaponDefinitions;
```

3. MainCamera 선택 / 인스펙터 창 / Main 컴포넌트 / Weapon Definitions
   - Size : 3
   - Element 0 설정

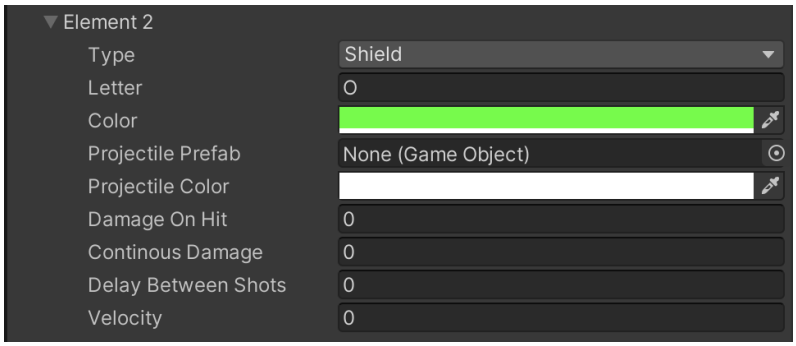


(색상 선택시 alpha 는 255 가 되도록 설정)

- Element 1 설정



- Element 2 설정



4. Main 스크립트 수정
   - 변수 추가

```
public class Main : MonoBehaviour
{
    static public Main S;
    static Dictionary<WeaponType, WeaponDefinition> WEAP_DICT;
```

   - Awake 메서드에 코드 추가

```
        Invoke("SpawnEnemy", 1f / enemySpawnPerSecond);

        WEAP_DICT = new Dictionary<WeaponType, WeaponDefinition>();
        foreach (var def in weaponDefinitions)
        {
            WEAP_DICT[def.type] = def;
        }
    }
```

   - GetWeaponDefinition 메서드 추가

```
    static public WeaponDefinition GetWeaponDefinition(WeaponType wt)
    {
        if (WEAP_DICT.ContainsKey(wt))
        {
            return WEAP_DICT[wt];
        }
        return new WeaponDefinition();
    }
```

5. Projectile 스크립트 수정
   - 변수 추가, WeaponType 프로퍼티 추가, SetType 메서드 추가, Awake 메서드 수정

```
public class Projectile : MonoBehaviour
{
    private BoundsCheck boundsCheck;
    private Renderer rand;

    [Header("Set Dynamically")]
    public Rigidbody rigid;
    [SerializeField]
    private WeaponType _type;

    public WeaponType type
    {
        get
        {
            return _type;
        }
        set
        {
            SetType(value);
        }
    }

    public void SetType(WeaponType eType)
    {
        _type = eType;
        WeaponDefinition definition = Main.GetWeaponDefinition(_type);
        rand.material.color = definition.projectileColor;
    }

    void Awake()
    {
        boundsCheck = GetComponent<BoundsCheck>();
        rand = GetComponent<Renderer>();
        rigid = GetComponent<Rigidbody>();
    }
```

6. Player 스크립트 수정
   - 변수 추가

```
    private GameObject lastTriggerGo = null;

    public delegate void WeaponFireDelegate();
    public WeaponFireDelegate fireDelegate;
```

   - Awake 메서드 수정 (마지막 줄)

```
        fireDelegate += TempFire;
    }
```

- Update 메서드 수정 (마지막 부분)

```
        // if (Input.GetKeyDown(KeyCode.Space))
        // {
        //     TempFire();
        // }

        if (Input.GetAxis("Jump") > 0.9f && fireDelegate != null)
        {
            fireDelegate();
        }
    }
```

- TempFire 메서드 수정

```
        Rigidbody rigidB = projGo.GetComponent<Rigidbody>();
        //rigidB.velocity = Vector3.up * projectileSpeed;

        Projectile projectile = projGo.GetComponent<Projectile>();
        projectile.type = WeaponType.blaster;
        float tSpeed = Main.GetWeaponDefinition(projectile.type).velocity;
        rigidB.velocity = Vector3.up * tSpeed;
    }
```

7. 플레이
   - 스페이스바를 누르면 매우 빠른 속도로 Projectile 이 발사됨

섹션 18 : Weapon 객체 생성

1. 하이어라키 창 / 빈 게임오브젝트 생성
   - 이름 : Weapon
   - Weapon 스크립트 어태치
   - 위치 : 0, 2, 0
2. Weapon 의 자식 오브젝트로 Cube 생성
   - 이름 : Barrel
   - Box Collider 컴포넌트 제거
   - 위치 : 0, 0.5, 0
   - 크기 : 0.25, 1, 0.1
3. Weapon 의 자식 오브젝트로 Cube 생성
   - 이름 : Collar
   - Box Collider 컴포넌트 제거
   - 위치 : 0, 1, 0
   - 크기 : 0.375, 0.5, 0.2
4. 새로운 머티리얼 생성 : Mat_Collar
   - Shader : Proto Tools / UnlitAlpha
   - Collar 에 적용
5. Weapon 게임 오브젝트를 프로젝트 창 / _Prefabs 폴더로 드래그하여 프리팹 생성
6. 하이어라키 창 / Weapon 게임 오브젝트를 Player 게임 오브젝트의 자식으로 이동
   - Weapon 의 위치 변경 : 0, 2, 0

섹션 19 : Weapon 스크립트에서 발사 처리

1. Player 스크립트 Awake 메서드에서 마지막 문장 주석 처리

```
        //fireDelegate += TempFire;
    }
```

2. Weapon 스크립트의 Weapon 클래스에 코드 추가

```
public class Weapon : MonoBehaviour
{
    static public Transform PROJECTION_ANCHOR;

    [Header("Set Dynamically")]
    [SerializeField]
    private WeaponType _type = WeaponType.none;
    public WeaponDefinition def;
    public GameObject collar;
    public float lastShotTime;

    private Renderer collarRend;

    private void Start()
    {
        collar = transform.Find("Collar").gameObject;
        collarRend = collar.GetComponent<Renderer>();

        SetType(_type);

        if (PROJECTION_ANCHOR == null)
        {
            GameObject go = new GameObject("_ProjectileAnchor");
            PROJECTION_ANCHOR = go.transform;
        }

        GameObject rootGO = transform.root.gameObject;
        if (rootGO.GetComponent<Player>() != null)
        {
            rootGO.GetComponent<Player>().fireDelegate += Fire;
        }
    }

    public WeaponType type
    {
        get
        {
            return _type;
        {
```

```
        set
        {
            _type = value;
        }
    }

    public void SetType(WeaponType wt)
    {
        type = wt;
        if (type == WeaponType.none)
        {
            this.gameObject.SetActive(false);
            return;
        }
        else
        {
            this.gameObject.SetActive(true);
        }
        def = Main.GetWeaponDefinition(type);
        collarRend.material.color = def.color;
        lastShotTime = 0;
    }

    public void Fire()
    {
        if (!gameObject.activeInHierarchy) return;
        if (Time.time - lastShotTime < def.delayBetweenShots) return;

        Projectile p;
        Vector3 vel = Vector3.up * def.velocity;
        if (transform.up.y < 0)
        {
            vel.y = -vel.y;
        }

        switch (type)
        {
        case WeaponType.blaster:
            p = MakeProjectile();
            p.rigid.velocity = vel;
            break;
        case WeaponType.spread:
            p = MakeProjectile();
            p.rigid.velocity = vel;
            p = MakeProjectile();
            p.transform.rotation = Quaternion.AngleAxis(10, Vector3.back);
            p.rigid.velocity = p.transform.rotation * vel;
            p = MakeProjectile();
```

```
            p.transform.rotation = Quaternion.AngleAxis(-10, Vector3.back);
            p.rigid.velocity = p.transform.rotation * vel;
            break;
        }

    public Projectile MakeProjectile()
    {
        GameObject go = Instantiate<GameObject>(def.projectilePrefab);
        if (transform.parent.gameObject.tag == "Hero")
        {
            go.tag = "ProjectileHero";
            go.layer = LayerMask.NameToLayer("ProjectileHero");
        }
        else
        {
            go.tag = "ProjectileEnemy";
            go.layer = LayerMask.NameToLayer("ProjectileEnemy");
        }
        go.transform.position = collar.transform.position;
        go.transform.SetParent(PROJECTION_ANCHOR, true);
        Projectile p = go.GetComponent<Projectile>();
        p.type = type;
        lastShotTime = Time.time;
        return p;
    }
}
```

3. 플레이 : 게임이 시작되면 Weapon 의 타입이 none 이 되어, Wepon 이 비활성화됨
4. Weapon 게임 오브젝트 선택 / 인스펙터 창 / Type : Blaster 로 변경
5. 플레이
   - Weapon 의 Type 을 Spread 로 변경하여 테스트

섹션 20 : Enemy 피해 계산

1. Enemy 스크립트의 OnCollisionEnter 메서드 수정

```
if (otherGO.tag == "ProjectileHero")
{
    Projectile p = otherGO.GetComponent<Projectile>();
    if (!boundsCheck.isOnScreen)
    {
        Destroy(otherGO);
        return;
    }

    health -= Main.GetWeaponDefinition(p.type).damageOnHit;
    if (health <= 0)
    {
        Destroy(this.gameObject);
    }
    Destroy(otherGO);
} else {
```

2. MainCamera 게임 오브젝트 / 인스펙터 창
   - Main 컴포넌트 / Prefab Enemies / Element 0 : EnemyPrefab0 로 변경
3. 플레이

섹션 21 : Enemy 피해 피드백

1. 새로운 스크립트 생성 : Utils

```
public class Utils : MonoBehaviour
{
    static public Material[] GetAllMaterials(GameObject gameObject)
    {
        Renderer[] rends = gameObject.GetComponentsInChildren<Renderer>();

        List<Material> mats = new List<Material>();
        foreach (var rend in rends)
        {
            mats.Add(rend.material);
        }
        return mats.ToArray();
    }
}
```

2. Enemy 스크립트 수정
   - 변수 추가

```
    public int score = 100;
    public float showDamageDuration = 0.1f;

    [Header("Set Dynamically: Enemy")]
    public Color[] originalColors;
    public Material[] materials;
    public bool showingDamage = false;
    public float damageDoneTime;
    public bool notifiedOfDestruction = false;

    protected BoundsCheck boundsCheck;
```

   - Awake 메서드 수정

```
    void Awake()
    {
        boundsCheck = GetComponent<BoundsCheck>();
        materials = Utils.GetAllMaterials(gameObject);
        originalColors = new Color[materials.Length];
        for (int i = 0; i < materials.Length; i++)
        {
            originalColors[i] = materials[i].color;
        }
    }
```

- ShowDamage, UnShowDamage 메서드 추가

```
void ShowDamage()
{
    foreach (var m in materials)
    {
        m.color = Color.red;
    }
    showingDamage = true;
    damageDoneTime = Time.time + showDamageDuration;
}

void UnShowDamage()
{
    for (int i = 0; i < materials.Length; i++) {
        materials[i].color = originalColors[i];
    }
    showingDamage = false;
}
```

- Update 메서드 수정

```
    Move();

    if (showingDamage && Time.time > damageDoneTime)
    {
        UnShowDamage();
    }
```

- OnColllisionEnter 메서드 수정

```
        if (health <= 0)
        {
            Destroy(this.gameObject);
        }
        ShowDamage();
        Destroy(otherGO);
    }
    else
    {
        print("Enemy hit by non-ProjectileHero: " + otherGO.name);
    }
```

3. 플레이

- 플레이어의 공격을 받은 Enemy 는 빨간색으로 깜빡임

섹션 22 : 아이템(Power-Up) 애셋

1. 하이어라키 창 / 3D Object / 3D Text 생성
   - 이름 변경 : PowerUp
     - ■ 인스펙터 창 / Text Mesh 컴포넌트
       - ◆ Text : M
       - ◆ Offset Z : -2
       - ◆ Character Size : 0.5
       - ◆ Anchor : Middle center
       - ◆ Alignment : Center
       - ◆ Font Size : 38
       - ◆ Font Style : Bold
     - ■ Rigidbody 컴포넌트 추가
       - ◆ Use Gravity : 체크 해제
       - ◆ Constraints / Freeze Position : Z 체크
       - ◆ Constraints / Freeze Rotation : X, Y, Z 체크
   - 자식 오브젝트로 3D Object / Cube 생성
     - ■ 크기 : 2, 2, 2
     - ■ Box Collider / Is Trigger : 체크
   - PowerUp 게임 오브젝트 선택
     - ■ 태그와 레이어 설정 : PowerUp (자식 오브젝트도 적용)
2. 머티리얼 생성 : Mat_PowerUp
   - Shader : ProtoTools / UnlitAlpha
   - Texture : PowerUp 으로 설정
   - Main Color : 0, 255, 255, 255
   - PowerUp 게임 오브젝트의 자식인 Cube 에 적용
3. PowerUp 게임 오브젝트 선택
   - BoundsCheck 스크립트 어태치
   - Keep On Screen : 체크 해제

섹션 23 : PowerUp 코드

1. 새로운 스크립트 생성 : PowerUp
   - PowerUp 게임 오브젝트에 어태치

```
public class PowerUp : MonoBehaviour
{
    [Header("Set in Inspector")]
    public Vector2 rotMinMax = new Vector2(15, 90);
    public Vector2 driftMinMax = new Vector2(.25f, 2);
    public float lifeTime = 6f;
    public float fadeTime = 4f;

    [Header("Set Dynamically")]
    public WeaponType type;
    public GameObject cube;
    public TextMesh letter;
    public Vector3 rotPerSecond;
    public float birthTime;

    private Rigidbody rigid;
    private BoundsCheck bndCheck;
    private Renderer cubeRend;

    void Awake()
    {
        cube = transform.Find("Cube").gameObject;
        letter = GetComponent<TextMesh>();
        rigid = GetComponent<Rigidbody>();
        bndCheck = GetComponent<BoundsCheck>();
        cubeRend = cube.GetComponent<Renderer>();

        Vector3 vel = Random.onUnitSphere;
        vel.z = 0;
        vel.Normalize();
        vel *= Random.Range(driftMinMax.x, driftMinMax.y);
        rigid.velocity = vel;

        transform.rotation = Quaternion.identity;

        rotPerSecond = new Vector3(Random.Range(rotMinMax.x, rotMinMax.y),
            Random.Range(rotMinMax.x, rotMinMax.y),
            Random.Range(rotMinMax.x, rotMinMax.y));

        birthTime = Time.time;
    }
```

```
    void Update()
    {
        cube.transform.rotation = Quaternion.Euler(rotPerSecond * Time.time);

        float u = (Time.time - (birthTime + lifeTime)) / fadeTime;

        if (u >= 1)
        {
            Destroy(this.gameObject);
            return;
        }

        if (u > 0)
        {
            Color color = cubeRend.material.color;
            color.a = 1f - u;
            cubeRend.material.color = color;
            color = letter.color;
            color.a = 1f - (u * 0.5f);
            letter.color = color;
        }

        if (!bndCheck.isOnScreen)
        {
            Destroy(gameObject);
        }
    }

    public void SetType(WeaponType weaponType)
    {
        WeaponDefinition definition = Main.GetWeaponDefinition(weaponType);
        cubeRend.material.color = definition.color;
        letter.text = definition.letter;
        type = weaponType;
    }

    public void AbsorbedBy(GameObject target)
    {
        Destroy(this.gameObject);
    }
}
```

2. 플레이
   - 플레이어와 PowerUp 이 충돌하면 "Triggered by non-Enemy: PowerUp"이라는 메시지가 출력됨
3. PowerUp 을 프로젝트 창 / _Prefabs 폴더로 드래그하여 프리팹 생성 (하이어라키 창에 있는 PowerUp 은 유지)

섹션 24 : PowerUp 충돌 처리

1. Player 스크립트 수정
   - AbsorbPowerUp 메서드 추가

```
public void AbsorbPowerUp(GameObject gameObject)
{
    PowerUp powerUp = gameObject.GetComponent<PowerUp>();
    switch (powerUp.type)
    {

    }
    powerUp.AbsorbedBy(this.gameObject);
}
```

   - OnTriggerEnter 메서드 수정

```
    if (go.tag == "Enemy")
    {
        shieldLevel--;
        Destroy(go);
    }
    else if (go.tag == "PowerUp")
    {
        AbsorbPowerUp(go);
    }
    else
    {
        print("Triggered by non-Enemy: " + go.name);
    }
```

2. 플레이
   - 플레이어와 PowerUp 이 충돌하면 PowerUp 이 사라짐

섹션 25 : 플레이어 무기 추가

1. Player 스크립트에 변수 추가

```
public float projectileSpeed = 40;
public Weapon[] weapons;
```

2. 하이어라키 창 / Player 게임 오브젝트의 자식은 Weapon 선택
   - 이름 변경 : Weapon_0
   - Weapon_0 를 4 개 복제하여 복제된 게임 오브젝트의 이름 변경 : Weapon_1, ..., Weapon_4
   - Weapon_1 의 위치 변경 : -2, 1, 0
   - Weapon_2 의 위치 변경 : 2, -1, 0
   - Weapon_3 의 위치 변경 : -1.25, -0.25, 0
   - Weapon_4 의 위치 변경 : 1.25, -0.25, 0
3. Player 게임 오브젝트 / 인스펙터 창 / Player 컴포넌트
   - Weapons 에 위 2 번에서 만든 Weapon_0 부터 Weapon_4 까지 할당
4. Player 스크립트 수정
   - GetEmptyWeaponSlot, ClearWeapons 메서드 추가

```
Weapon GetEmptyWeaponSlot()
{
    for (int i = 0; i < weapons.Length; i++)
    {
        if (weapons[i].type == WeaponType.none)
        {
            return weapons[i];
        }
    }
    return null;
}

void ClearWeapons()
{
    foreach (Weapon w in weapons)
    {
        w.SetType(WeaponType.none);
    }
}
```

- AbsorbPowerUp 메서드 수정

```
switch (powerUp.type)
{
    case WeaponType.shield:
        shieldLevel++;
        break;
    default:
        if (powerUp.type == weapons[0].type) {
            Weapon weapon = GetEmptyWeaponSlot();
            if (weapon != null) {
                weapon.SetType(powerUp.type);
            }
        } else {
            ClearWeapons();
            weapons[0].SetType(powerUp.type);
        }
        break;
}
```

5. 플레이
   - 5 개의 Weapon 에서 모두 발사됨
6. 하이어라키 창에 있는 PowerUp 선택
   - 인스펙터 창 / Power Up 컴포넌트
     - Type : Spread 로 변경
7. 플레이
   - PowerUp 을 획득하면 무기가 변경됨
8. Player 스크립트에 Start 메서드 추가

```
void Start()
{
    ClearWeapons();
    weapons[0].SetType(WeaponType.blaster);
}
```

9. Weapon 스크립트에 Awake 메서드 추가 (메서드 안에 있는 코드는 Start 메서드에서 잘라내기)

```
void Awake()
{
    collar = transform.Find("Collar").gameObject;
    collarRend = collar.GetComponent<Renderer>();
}
```

10. 하이어라키 창에 있는 PowerUp 의 Type 변경 : Blaster
11. 플레이
    - PowerUp 을 획득하면 2 개의 Weapon 에서 blaster 가 발사됨

섹션 26 : PowerUp 드랍

1. Main 스크립트 수정
   - 변수 추가

```
public WeaponDefinition[] weaponDefinitions;
public GameObject   prefabPowerUp;
public WeaponType[] powerUpFreqeuncy = new WeaponType[] {
    WeaponType.blaster,
    WeaponType.blaster,
    WeaponType.spread,
    WeaponType.shield
};
```

   - ShipDestroyed 메서드 추가

```
public void ShipDestroyed(Enemy enemy)
{
    if (Random.value <= enemy.powerUpDropChance)
    {
        int ndx = Random.Range(0, powerUpFreqeuncy.Length);
        WeaponType puType = powerUpFreqeuncy[ndx];

        GameObject gameObject = Instantiate(prefabPowerUp) as GameObject;
        PowerUp powerUp = gameObject.GetComponent<PowerUp>();
        powerUp.SetType(puType);

        powerUp.transform.position = enemy.transform.position;
    }
}
```

2. Enemy 스크립트 수정
   - 변수 추가

```
public float showDamageDuration = 0.1f;
public float powerUpDropChance = 1f;
```

   - OnCollisionEnter 메서드 수정

```
        if (health <= 0)
        {
            if (!notifiedOfDestruction) {
                Main.S.ShipDestroyed(this);
            }
            notifiedOfDestruction = true;
            Destroy(this.gameObject);
        }
```

3. MainCamera 게임 오브젝트 선택 / 인스펙터 창
   - Main 컴포넌트 / Prefab PowerUp : PowerUp 으로 설정
   - 하이어라키 창에 있는 PowerUp 은 삭제
4. 플레이

섹션 27 : 더 복잡한 Enemy – Enemy_4

1. EnemyPrefab4 수정
   - BoundsCheck 스크립트 컴포넌트 추가
     - Radius : 3.5
     - Keep On Screen : 체크 해제
   - Enemy_4 스크립트 컴포넌트 추가
2. Enemy_4 스크립트 수정

```
public class Enemy_4 : Enemy
{
    private Vector3 p0, p1;
    private float timeStart;
    private float duration = 4;

    void Start ()
    {
        p0 = p1 = pos;
        InitMovement();
    }

    void InitMovement()
    {
        p0 = p1;

        float widMinRad = boundsCheck.camWidth - boundsCheck.radius;
        float hgtMinRad = boundsCheck.camHeight - boundsCheck.radius;
        p1.x = Random.Range(-widMinRad, widMinRad);
        p1.y = Random.Range(-hgtMinRad, hgtMinRad);

        timeStart = Time.time;
    }

    public override void Move()
    {
        float u = (Time.time - timeStart) / duration;

        if (u >= 1) {
            InitMovement();
            u = 0;
        }

        u = 1 - Mathf.Pow(1 - u, 2);
        pos = (1 - u) * p0 + u * p1;
    }
}
```

3. MainCamera 게임 오브젝트 선택 / 인스펙터 창
   - Prefab Enemies / Element 0 : EnemyPrefab4 로 설정
4. 플레이

섹션 28 : Enemy_4 를 다중 부품으로 분리

1. Enemy_4 스크립트 수정
   - Enemy_4 클래스 정의 위에 새로운 클래스 Part 정의 추가

```
[System.Serializable]
public class Part
{
    public string name;
    public float health;
    public string[] protectedBy;

    [HideInInspector]
    public GameObject gameObject;
    [HideInInspector]
    public Material material;
}
```

   - Enemy_4 클래스에 변수 추가

```
public class Enemy_4 : Enemy
{
    [Header("Set in Inspector: Enemy_4")]
    public Part[] parts;
```

   - Start 메서드에 코드 추가

```
        InitMovement();

        Transform t;
        foreach (var prt in parts)
        {
            t = transform.Find(prt.name);
            if (t != null)
            {
                prt.gameObject = t.gameObject;
                prt.material = t.gameObject.GetComponent<Renderer>().material;
            }
        }
    }
```

2. EnemyPrefab4 수정
   - Parts : Size : 4
   - Element 0
     - Name : Cockpit
     - Health : 10
     - Protected By / Size : 1
     - Protected By / Element 0 : Fuselage
   - Element 1

- Name : Fuselage
- Health : 10
- Protected By / Size : 2
- Protected By / Element 0 : WingL
- Protected By / Element 0 : WingR

- Element 2
  - Name : WingL
  - Health : 10
- Element 3
  - Name : WingR
  - Health : 10

3. Enemy_4 스크립트의 Enemy_4 클래스에 다음 메서드 코드 추가

```
Part FindPart(string n)
{
    foreach (var prt in parts)
    {
        if (prt.name == n)
        {
            return prt;
        }
    }
    return null;
}

Part FindPart(GameObject gameObject)
{
    foreach (var prt in parts)
    {
        if (prt.gameObject == gameObject)
        {
            return prt;
        }
    }
    return null;
}

bool Destroyed(GameObject gameObject)
{
    return Destroyed(FindPart(gameObject));
}

bool Destroyed(string n)
{
    return Destroyed(FindPart(n));
}
```

```
bool Destroyed(Part prt)
{
    if (prt == null)
    {
        return true;
    }
    return prt.health <= 0;
}

void ShowLocalizedDamage(Material material)
{
    material.color = Color.red;
    damageDoneTime = Time.time + showDamageDuration;
    showingDamage = true;
}

private void OnCollisionEnter(Collision collision)
{
    GameObject other = collision.gameObject;
    switch (other.tag)
    {
        case "ProjectileHero":
            Projectile projectile = other.GetComponent<Projectile>();
            if (!boundsCheck.isOnScreen)
            {
                Destroy(other);
                break;
            }

            GameObject goHit = collision.contacts[0].thisCollider.gameObject;
            Part prtHit = FindPart(goHit);
            if (prtHit == null)
            {
                goHit = collision.contacts[0].otherCollider.gameObject;
                prtHit = FindPart(goHit);
            }

            if (prtHit.protectedBy != null)
            {
                foreach (var s in prtHit.protectedBy)
                {
                    if (!Destroyed(s))
                    {
                        Destroy(other);
                        return;
                    }
                }
            }
```

```
                prtHit.health -= Main.GetWeaponDefinition(projectile.type).damageOnHit;
                ShowLocalizedDamage(prtHit.material);
                if (prtHit.health <= 0)
                {
                    prtHit.gameObject.SetActive(false);
                }

                bool allDestroyed = true;
                foreach (var prt in parts)
                {
                    if (!Destroyed(prt))
                    {
                        allDestroyed = false;
                        break;
                    }
                }
                if (allDestroyed)
                {
                    Main.S.ShipDestroyed(this);
                    Destroy(this.gameObject);
                }
                Destroy(other);
                break;
            }
        }
```

4. 플레이
5. MainCamera 게임 오브젝트 선택 / 인스펙터 창 / Main 컴포넌트
   - Prefab Enemies / Size 10
   - Element 0, Element 1, Element 2 : EnemyPrefab0 으로 설정
   - Element 3, Element 4 : EnemyPrefab1 로 설정
   - Element 5, Element 6 : EnemyPrefab2 로 설정
   - Element 6, Element 8 : EnemyPrefab3 으로 설정
   - Element 9 : EnemyPrefab4 로 설정
6. 프로젝트 창 / _Prefabs 폴더
   - EnemyPrefab0 선택 / Power Up Drop Chance : 0.25
   - EnemyPrefab1 선택 / Power Up Drop Chance : 0.5
   - EnemyPrefab2 선택 / Power Up Drop Chance : 0.5
   - EnemyPrefab3 선택 / Power Up Drop Chance : 0.75
   - EnemyPrefab4 선택 / Power Up Drop Chance : 1
7. 플레이

섹션 29 : 배경 스크롤

1. 하이어라키 창 / 3D Object / Quad 생성
   - 이름 변경 : StarfieldBG
   - 위치 : 0, 0, 10
   - 크기 : 80, 80, 1
2. 프로젝트 창 / _Materials 폴더 / Material 생성
   - 이름 변경 : Mat_Starfield
   - Shader : ProtoTools / UnlitAlpha
   - Main(Texture) : Space
   - 1 번에서 만든 StarfieldBG 게임오브젝트에 적용
3. 프로젝트 창 / _Materials 폴더 / Mat_Starfield 복제
   - 이름 변경 : Mat_Starfield_Transparent
   - Main(Texture) : Space_Transparent
4. 하이어라키 창 / StarfieldBG 복제
   - 이름 변경 : StarfieldFG_0
   - StarfieldFG_0 복제
     - ■ 이름 변경 : StarfieldFG_1
5. 스크립트 생성 : Parallax

```
public class Parallax : MonoBehaviour
{
    [Header("Set in Inspecgtor")]
    public GameObject poi;
    public GameObject[] panels;
    public float scrollSpeed = -30f;
    public float motionMult = 0.25f;

    private float panelHt;
    private float depth;

    void Start()
    {
        panelHt = panels[0].transform.localScale.y;
        depth = panels[0].transform.position.z;

        panels[0].transform.position = new Vector3(0, 0, depth);
        panels[1].transform.position = new Vector3(0, panelHt, depth);
    }
```

(다음 페이지에서 계속)

```
    void Update ()
    {
        float tY, tX = 0;
        tY = Time.time * scrollSpeed % panelHt + (panelHt * 0.5f);

        if (poi != null)
        {
            tX = -poi.transform.position.x * motionMult;
        }

        panels[0].transform.position = new Vector3(tX, tY, depth);

        if (tY >= 0)
        {
            panels[1].transform.position = new Vector3(tX, tY - panelHt, depth);
        } else
        {
            panels[1].transform.position = new Vector3(tX, tY + panelHt, depth);
        }
    }
}
```

6. 하이어라키 창 / StarfieldBG 선택
   - Parallax 스크립트 컴포넌트 어태치
   - Parallax 컴포넌트 설정
     - Poi : 하이어라키 창에 있는 Player 로 설정
     - Panels / Size : 2
     - Panels / Element 0 : StarfieldFG_0
     - Panels / Element 1 : StarfieldFG_1
7. 플레이

(03 장 Space SHMUP 끝)