

포트폴리오 설명문서

지원자 :

작성일 : 2017년 / 01월 / 13일

=====READ ME=====

이 문서는 프로젝트 설명을 위해 작성되었습니다. 프로젝트 소스코드 및 플레이영상 그리고 실행 파일의 경우 동봉되어 있는 폴더에 있으니 참고 부탁드립니다.



참고 사항

1. 이 문서에 포함되어 있는 소스코드는 일부 발췌 했습니다. 전체 소스 코드는 동봉되어 있는 파일을 참고해주시면 됩니다.
2. 이 문서에 설명하는 프로젝트 뿐만 아니라 여러 개인 프로젝트를 진행 중에 있으며 모든 내용은 개인 블로그에 정리하여 올리고 있으며 각 소스코드들은 GitHub를 통해 버전관리를 하고 있습니다. 링크는 아래와 같습니다.
3. 만약, 동봉된 파일(소스코드 및 실행파일)이 없거나 잘못된 경우 아래 구글 드라이브 링크로 가시면 파일이 있으니 다운받아 이용하시면 됩니다.
4. 이 문서에는 일부 소스코드만 설명합니다. 자세한 내용은 동봉 또는 구글 드라이브 링크를 통해 소스코드를 참고하시면 됩니다.

블로그 주소	http://pgrblues.egloos.com/
깃허브 주소	https://github.com/opk4406opk
구글드라이브	https://drive.google.com/file/d/0B0TK9ZPf_INHNIhybDJWT0dMU1U/view?usp=sharing

목차

1. Watch-Out UnityChan

2. HELLO_MY_WORLD

3. OneCoinOnePlay

4. WatchOutCubes

5. ProtectPlanet(가칭)

6. 기타 개인 프로젝트

➔ RunUnityChan.

➔ 모바일 웹크롤링 앱.

➔ 개인용도의 기타 프로그램들.

7. 현재 공부중인 OpenSource 프로젝트.

WatchOut-UnityChan !

게임 장르 : 러닝 게임

사용 엔진 : Unity3D

사용 언어 : C#

사용 라이브러리 : ITween, NGUI2.x

구글 플레이 서비스 유/무 : 무

목표 플랫폼 : 안드로이드 (모바일)

프로젝트 소스 링크 : <https://github.com/opk4406opk/WatchOut>

구글 스토어 링크 :

https://play.google.com/store/apps/details?id=com.KojeomStudio.WatchOut_UnityChan

설명 : 구글 플레이 스토어 입점 및 NUGI2.x, ITween 라이브러리를 테스트하고자 시작했던 프로젝트입니다.
캐릭터는 계속 행성위를 달리며 하늘에서 유성이 떨어진다는 간단한 내용의 러닝 룰을 가진 게임입니다.

```
1 void Start ()
2 {
3
4     defaultNum = 1;
5     //defaultMisArr[0] = defaultMisPrefab0;
6     defaultMisArr[1] = defaultMisPrefab1;
7
8     missileList = new List<GameObject>();
9
10    positionGroup = generatePosGroup.GetComponentInChildren<Transform>();
11    posGroupLength = positionGroup.Length;
12
13    homingPosGroup = generateHomingPosGroup.GetComponentInChildren<Transform>();
14    homingPosGroupLength = homingPosGroup.Length;
15
16    for(int idx = 0; idx < missileMaxNum; ++idx)
17    {
18        missileList.Add(Instantiate(defaultMisArr[defaultNum],
19                                Vector3.zero,
20                                Quaternion.identity) as GameObject);
21        missileList[idx].transform.position = positionGroup[Random.Range(1, posGroupLength)].position;
22        missileList[idx].SetActive(false);
23
24        EffectSettings settings = missileList[idx].GetComponent<EffectSettings>();
25        settings.MoveSpeed = Random.RandomRange(1.0f, 5.0f);
26        settings.Target = targetPlanet;
27        settings.CollisionEnter += (n, e) =>
28        {
29            // to do
30            if(e.Hit.collider.CompareTag("Planet"))
31            {
32                AudioSource sfxAudio = settings.gameObject.GetComponent<AudioSource>();
33                sfxAudio.clip = explosionSfx;
34                sfxAudio.volume = 0.2f;
35                sfxAudio.Play();
36            }
37        };
38    };
39 }
40 }
```

하늘에서 유성이 떨어진다는 컨셉의 게임이기에 유성들을 미리 생성한후에 게임중에는 재사용을 하였습니다. Queue 같은 자료구조를 이용해 메모리풀 기능을 제대로 구현한건 아니지만, 기본적으로 지면에 충돌한 유성은 원래 위치로 복귀하여 다시 Start 를 하는 구조로 되어 있습니다. (아래 코루틴을 참고해주세요.)

```
1 public void StartMisProcess() { StartCoroutine(RePositioningMis()); }
2 IEnumerator RePositioningMis()
3 {
4     while (true)
5     {
6         missileMaxNum = missileList.Count;
7         for (int idx = 0; idx < missileMaxNum; ++idx)
8             if (missileList[idx].activeSelf == false)
9             {
10                 missileList[idx].transform.position = positionGroup[Random.Range(1, posGroupLength)].position;
11                 missileList[idx].SetActive(true);
12                 AudioSource sfx = missileList[idx].GetComponent<AudioSource>();
13                 sfx.clip = frozenMeteoSfx;
14                 sfx.volume = 0.3f;
15                 sfx.PlayDelayed(0.35f);
16             }
17
18         yield return null;
19     }
20 }
```

HELLO_MY_WORLD

게임 장르 : 샌드박스형식의 마인크래프트 모작게임

사용 엔진 : Unity3D

사용 언어 : C#

사용 라이브러리 : ITween, NGUI3.x, JJsonObject, Sqlite3

구글 플레이 서비스 유/무 : 무

목표 플랫폼 : PC - Windows / Linux

프로젝트 소스 링크 : https://github.com/opk4406opk/HELLO_MY_WORLD

프로젝트 실행 파일 링크 :

https://drive.google.com/folderview?id=0B3yL35_ie0dQY0VDWDhqSIFQdU0&usp=sharing

설명 : 해당 게임은 마인크래프트의 복셀형식을 따라하고자 시작한 프로젝트입니다. PC에서 윈도우즈/리눅스를 플랫폼으로 하여 제작중인 게임입니다. 블록을 부수고 만드는 것 외에 RPG 성향도 추가해 가고 있습니다.

코드설명 :

마인크래프트와 같은 복셀형식의 게임의 기본이 되는 알고리즘은 외국의 프로그래머가 자신의 블로그에 남겨놓은 튜토리얼을 참고했습니다.

(주소:<http://studentgamedev.blogspot.kr/2013/08/unity-voxel-tutorial-part-1-generating.html>)

해당 튜토리얼은 유니티엔진에서 제공하는 Physics 관련 API를 활용하도록 되어있습니다. 이에 따라 이 프로젝트도 동일한 방식으로 작성했으나 최근 들어 기존 엔진에서 제공하는 Physics API를 사용하지 않고 직접 제작한 AABB, RayCasting 등을 이용하여 지형충돌을 하고 있으며 추가적으로 게임 내 존재하는 NPC의 길 찾기 알고리즘 적용 그리고 지형 렌더링 및 충돌에 쓰이는 Octree 등 여러 방면에서 새롭게 제작하여 테스트 및 적용을 하고 있습니다.

기존 유니티엔진에서 제공하는 Physics API를 사용하는걸 전제로 한 프로젝트였기에 위에서 언급한 새로운 방법으로 변경을 진행중 입니다. 따라서 현재진행형인 프로젝트이며 이 문서에서 보이는 소스코드들의 경우 미흡한 부분이 있사오니 이점 유념해 주시길 바랍니다. (버그가 있을 수 있습니다.)

모든 개인 프로젝트는 gitHub를 이용하고 있으며, HELLO_MY_WORLD 프로젝트는 2개의 브랜치를 이용하여 나누어 관리하고 있습니다. 1개는 메인 브랜치로 기존에 유니티엔진 Physics API를 이용한 것이며 나머지 브랜치는 앞서 말한 새로운 방법으로 변경 하여 다시 개발중인 프로젝트입니다. 주소는 아래와 같습니다.

:: 유니티 엔진 이용(Main Branch) -> https://github.com/opk4406opk/HELLO_MY_WORLD/tree/master

:: 새로운 방법 이용(Sub Branch) -> https://github.com/opk4406opk/HELLO_MY_WORLD/tree/custom-object-collision-detection-%EB%B0%8F-%EA%B8%B0%ED%83%80-%EC%84%B1%EB%8A%A5%EA%B0%9C%EC%84%A0-%ED%85%8C%EC%8A%A4%ED%8A%B8

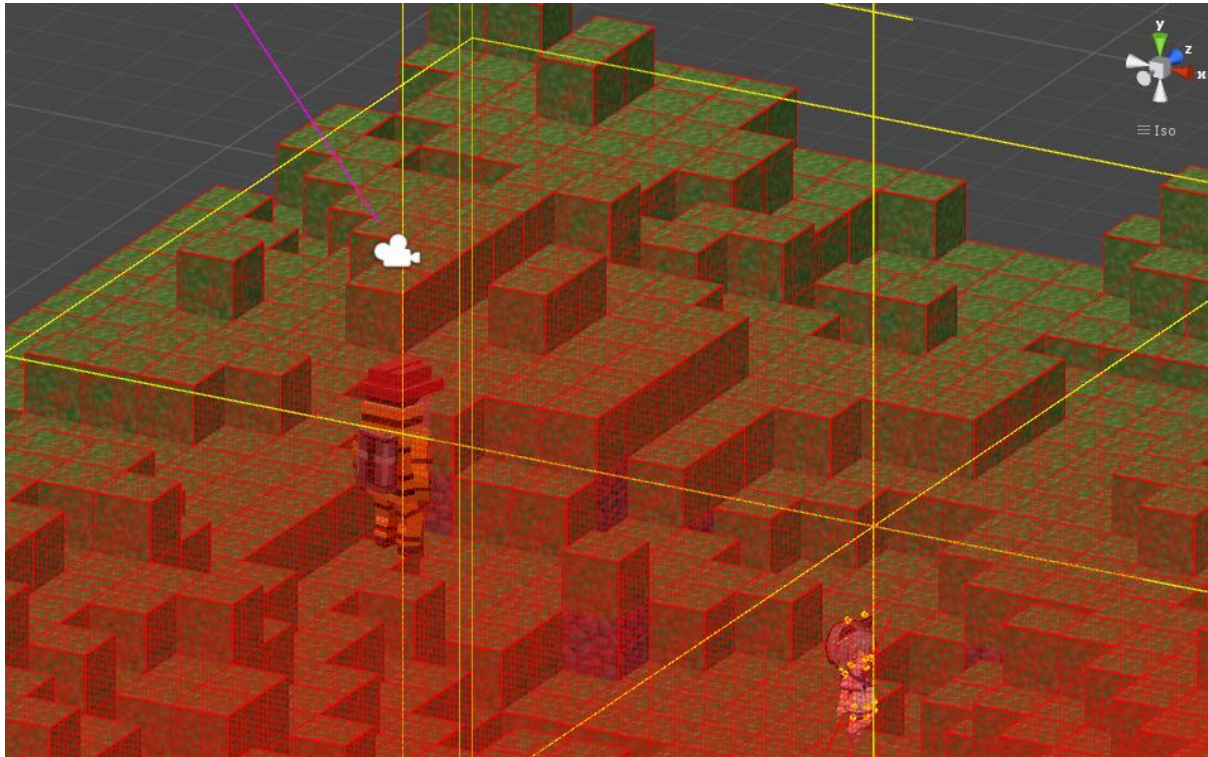
플레이 영상:

<https://www.youtube.com/watch?v=fiSUAhfTsUk>

(설명은 다음 장으로 이어집니다.)

p.s. 이 문서에서 HELLO_MY_WORLD의 설명은 새롭게 변경중인 부분을 중점으로 설명합니다.

p.s.2 현재 HELLO_MY_WORLD에 플레이 영상, 실행파일은 main branch(기존 유니티 물리 API를 이용했던)의 영상입니다. 아직 서버로 빼놓은 Branch 작업이 끝나지 않았기에 따로 플레이 영상이나 실행파일은 만들어서 포함시키지 않았습니다. 다만, 소스코드는 sub branch의 코드로 첨부되어 있습니다.



(* 게임 내에서 지형이 렌더링 되는 모습입니다. 빨간색 큐브는 AABB의 모습을 그려주고 있습니다.)

```

4 public struct CustomAABB {
5
6     private Vector3 _minExtent;
7     public Vector3 minExtent
8     {
9         get { return _minExtent; }
10    }
11    private Vector3 _maxExtent;
12    public Vector3 maxExtent
13    {
14        get { return _maxExtent; }
15    }
16    // in world space coordinate
17    private Vector3 _centerPos;
18    public Vector3 centerPos
19    {
20        get { return _centerPos; }
21    }
22
23    private bool _isEnabled;
24    public bool isEnabled
25    {
26        set { _isEnabled = value; }
27        get { return _isEnabled; }
28    }
29

```

AABB 클래스 입니다.

게임 내 존재하는 지형의 경우, 월드 좌표에서 회전을 하거나 움직이는 일이 없습니다. 따라서, 충돌 계산이 빠르고 구현하기 쉬운 AABB를 이용하여 지형 충돌에 이용하고 있습니다.

이 클래스는 AABB크기를 생성하는 메소드와 AABB와 발생할 수 있는 여러 충돌 상황에 대한 감지 메소드를 가지고 있습니다.

현재는 특정 Point 및 AABB 간의 충돌 감지 메소드 총 2개를 가지고 있습니다.

소스코드는 이어서 다음 그림에 있습니다.

```

1 public void MakeAABB(Vector3[] points)
2 {
3     _minExtent = points[0];
4     _maxExtent = points[0];
5     for(int idx = 1; idx < points.Length; idx++)
6     {
7         if (points[idx].x < _minExtent.x) _minExtent.x = points[idx].x;
8         else if (points[idx].x > _maxExtent.x) _maxExtent.x = points[idx].x;
9         if (points[idx].y < _minExtent.y) _minExtent.y = points[idx].y;
10        else if (points[idx].y > _maxExtent.y) _maxExtent.y = points[idx].y;
11        if (points[idx].z < _minExtent.z) _minExtent.z = points[idx].z;
12        else if (points[idx].z > _maxExtent.z) _maxExtent.z = points[idx].z;
13    }
14
15    _centerPos = (_maxExtent + _minExtent) / 2;
16 }
17 public void MakeAABB(Vector3 minExtent, Vector3 maxExtent)
18 {
19     _minExtent = minExtent;
20     _maxExtent = maxExtent;
21     _centerPos = (_maxExtent + _minExtent) / 2;
22 }
23
24 public bool IsInterSectPoint(Vector3 point)
25 {
26     if ((point.x > _minExtent.x && point.x < _maxExtent.x ) &&
27         (point.y > _minExtent.y && point.y < _maxExtent.y) &&
28         (point.z > _minExtent.z && point.z < _maxExtent.z))
29     {
30         return true;
31     }
32     return false;
33 }
34
35 public bool IsInterSectAABB(CustomAABB other)
36 {
37     if ((_minExtent.x <= other.maxExtent.x && _maxExtent.x >= other.minExtent.x) &&
38         (_minExtent.y <= other.maxExtent.y && _maxExtent.y >= other.minExtent.y) &&
39         (_minExtent.z <= other.maxExtent.z && _maxExtent.z >= other.minExtent.z))
40     {
41         return true;
42     }
43     return false;
44 }
45 }

```

(* AABB 클래스의 Method 선언 부분 입니다)

다음 장에 이어서 게임 내 존재하는 모든 블록이 가진 AABB에 대해 충돌체크를 하는 부분에 대해 설명합니다.


```

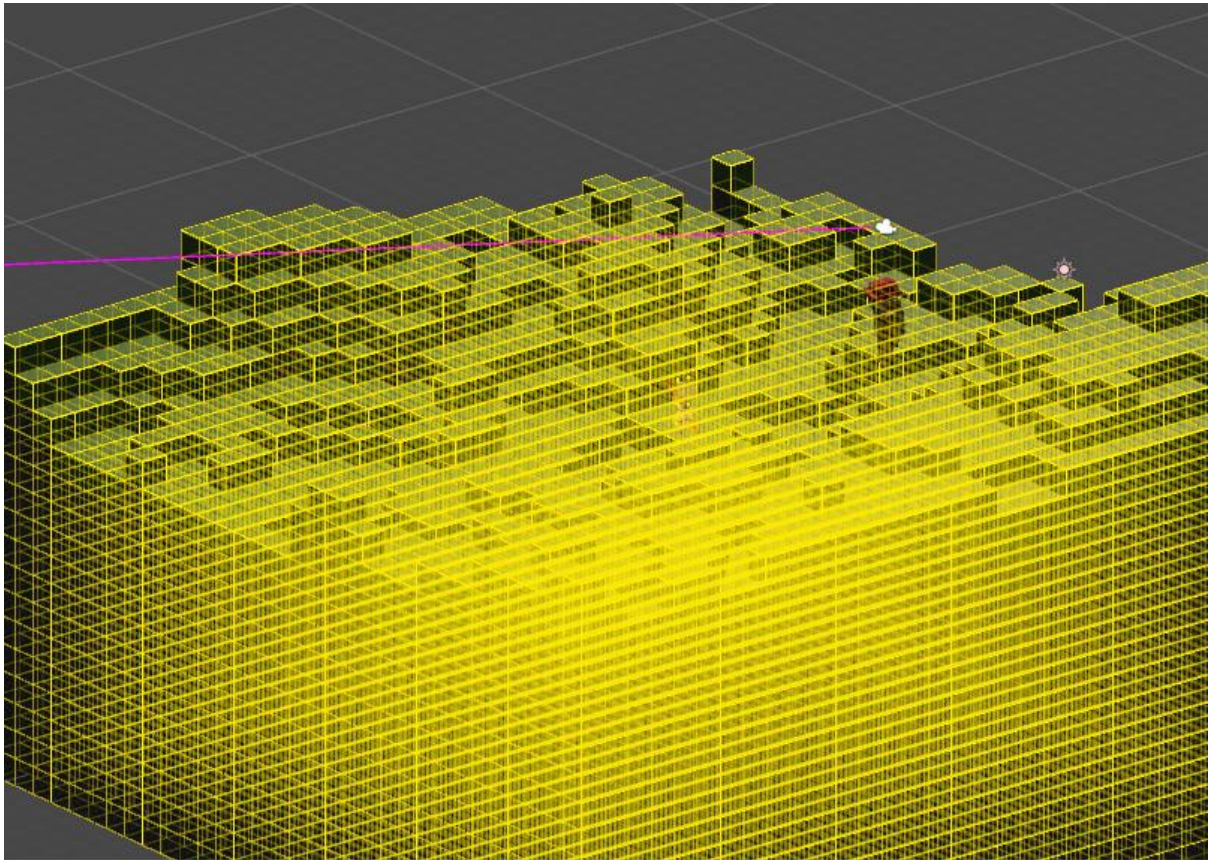
4 // ref #1 : http://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-
5
6 public class CustomRayCast : MonoBehaviour
7 {
8     public static bool IntersectWithAABB(Ray ray, CustomAABB aabb)
9     {
10         if (!aabb.isEnabled) return false;
11
12         float tmin = (aabb.minExtent.x - ray.origin.x) / ray.direction.normalized.x;
13         float tmax = (aabb.maxExtent.x - ray.origin.x) / ray.direction.normalized.x;
14
15         if (tmin > tmax)
16         {
17             float tmp = tmin;
18             tmin = tmax;
19             tmax = tmp;
20         }
21
22         float tymin = (aabb.minExtent.y - ray.origin.y) / ray.direction.normalized.y;
23         float tymax = (aabb.maxExtent.y - ray.origin.y) / ray.direction.normalized.y;
24
25         if (tymin > tymax)
26         {
27             float tmp = tymin;
28             tymin = tymax;
29             tymax = tmp;
30         }
31
32         if ((tmin > tymax) || (tymin > tmax))
33             return false;
34
35         if (tymin > tmin)
36             tmin = tymin;
37
38         if (tymax < tmax)
39             tmax = tymax;
40
41         float tzmin = (aabb.minExtent.z - ray.origin.z) / ray.direction.normalized.z;
42         float tzmax = (aabb.maxExtent.z - ray.origin.z) / ray.direction.normalized.z;
43
44         if (tzmin > tzmax)
45         {
46             float tmp = tzmin;
47             tzmin = tzmax;
48             tzmax = tmp;
49         }
50
51         if ((tmin > tzmax) || (tzmin > tmax))

```

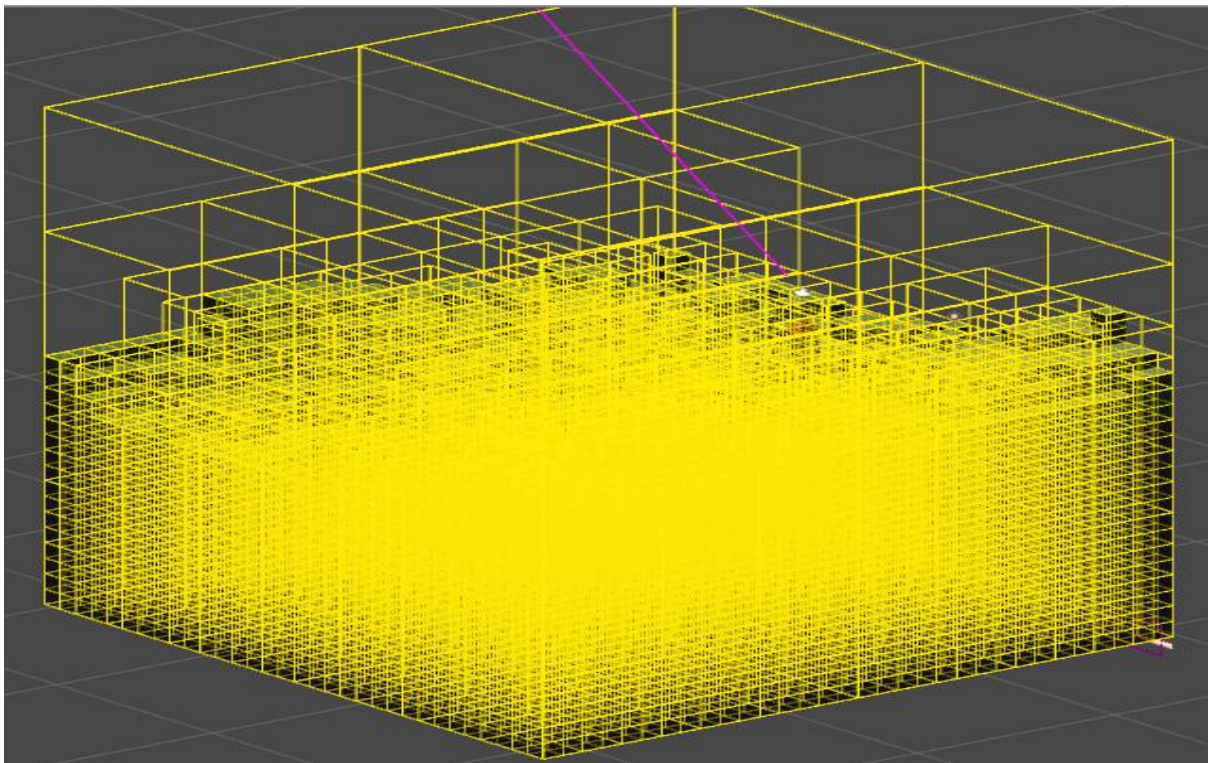
(* 지형에 대해 플레이어가 블록을 삭제/생성 할 수 있도록 픽킹을 위한 RayCast 클래스 입니다.)

앞서 설명했듯이 지형으로 존재하는 블록들은 AABB를 가지고 있습니다. 따라서 플레이어가 특정 지형에 대해 삭제/생성을 하는 경우 충돌 체크를 위해 플레이어가 클릭한 지점을 기준으로 Ray를 생성하여 광선 충돌 테스트를 합니다.

다음으로, Octree를 이용한 지형충돌에 대한 부분을 살펴보겠습니다.



(* Octree중에 최소 단위의 Node만 그려주고 있습니다.)



(* Octree의 모든 Node를 그려 주고 있습니다.)

```

private void CollideNodeWithRay(Ray ray, COTNode root)
{
    if (root == null) return;

    CollideInfo info;
    info.isCollide = false;
    info.hitBlockCenter = new Vector3(0, 0, 0);
    if(root.size == blockMinSize)
    {
        info.isCollide = true;
        info.hitBlockCenter = root.center;
        collideCandidate.Add(info);
    }
    for(int i = 0; i < 8; i++)
    {
        if ((root.childds[i] != null) &&
            (CustomRayCast.InterSectWithAABB(ray, root.childds[i]
        {
            CollideNodeWithRay(ray, root.childds[i]);
        }
    }
}

```

(*Octree의 Collide 메소드 입니다.)

지형 크기 만큼 Octree의 node를 생성합니다. 이 node를 순회하며 충돌한 Block을 찾게 됩니다. 현재 마우스픽킹으로 생성된 '광선'과 Octree의 node간의 충돌만을 구현하여 사용하고 있습니다. 1개의 광선이 지나가는 길에 여러 개의 node가 있으므로 이 node들 중에 광선의 시작점과 가장 가까운 것을 충돌된 블록으로 판정하고 있습니다. (다음 장에 포함된 소스코드 스크린샷을 참고하시기 바랍니다.)

```

public CollideInfo Collide(Ray ray)
{
    CollideNodeWithRay(ray, root);

    CollideInfo info;
    info.isCollide = false;
    info.hitBlockCenter = new Vector3(0, 0, 0);
    Vector3 hitBlockCenter;
    if(collideCandidate.Count > 0)
    {
        float minDist = Vector3.Distance(ray.origin, collideCandidate[0].hitBlockCenter);
        hitBlockCenter = collideCandidate[0].hitBlockCenter;
        for (int i = 1; i < collideCandidate.Count; i++)
        {
            float d = Vector3.Distance(ray.origin, collideCandidate[i].hitBlockCenter);
            if (minDist > d)
            {
                minDist = d;
                hitBlockCenter = collideCandidate[i].hitBlockCenter;
            }
        }
        info.isCollide = true;
        info.hitBlockCenter = hitBlockCenter;
        collideCandidate.Clear();
    }
    return info;
}

```

추가적으로 광선뿐만 아니라 단일 AABB나 특정 Point간의 충돌 부분도 구현할 예정이며, 지형을 충돌 하는 데에만 Octree를 이용 하는게 아니라 지형을 렌더링 하는 부분에도 이용할 생각입니다. 현재 구현 중에 있습니다.

다음으로 게임 내 NPC들의 특정 지형에서 길을 찾는 알고리즘인 A*에 대한 부분을 보겠습니다.

```
8 public class PathNode
9 {
10     // 길찾기용 맵 데이터 좌표 x,z 이다. 그러나 실제 월드 블록 배열 x,z 평면값과 동일
11     // 이 값을 실제 길을 찾아가는 오브젝트가 월드 좌표 (x, z)값으로 사용 가능하다.
12     private int _pathMapDataX;
13     public int pathMapDataX
14     {
15         set { _pathMapDataX = value; }
16         get { return _pathMapDataX; }
17     }
18     private int _pathMapDataZ;
19     public int pathMapDataZ
20     {
21         set { _pathMapDataZ = value; }
22         get { return _pathMapDataZ; }
23     }
24
25     private int _worldCoordY;
26     public int worldCoordY
27     {
28         set { _worldCoordY = value; }
29         get { return _worldCoordY; }
30     }
31
32     private bool _isJumped;
33     public bool isJumped
34     {
35         set { _isJumped = value; }
36         get { return _isJumped; }
37     }
38
39     private PathNode _parentNode;
40     public PathNode parentNode
41     {
42         set { _parentNode = value; }
43         get { return _parentNode; }
44     }
45
46     private bool _isGoalNode;
47     public bool isGoalNode
48     {
49         set { _isGoalNode = value; }
50         get { return _isGoalNode; }
51     }
52 }
```

(* PathNode 클래스의 일부분의 소스코드 입니다.)


```

97  /// <summary>
98  /// 길찾기에 이용되는 맵정보 배열을 초기화 합니다.
99  /// 실제 월드 블록 배열의 xz평면의 범위값을 이용해 맵 정보 배열을 순회.
100  /// </summary>
101  private void InitPathFindMapData()
102  {
103      for (int x = 0; x < MAP_SIZE_X; x++)
104          for (int z = 0; z < MAP_SIZE_Z; z++)
105          {
106              pathFindMapData[x, z].pathMapDataX = x;
107              pathFindMapData[x, z].pathMapDataZ = z;
108              pathFindMapData[x, z].worldCoordY = 0;
109              pathFindMapData[x, z].parentNode = null;
110              pathFindMapData[x, z].isGoalNode = false;
111              pathFindMapData[x, z].isJumped = false;
112          }
113  }
114  /// <summary>
115  /// 길찾기에 이용되는 맵정보 배열에 실질적인 정보를 저장합니다.
116  /// </summary>
117  private void BuildPathFindMapData()
118  {
119      //길을 찾아 움직이려는 오브젝트의 현재 높이.
120      //실제 밟고 서있는 WorldBlock 배열의 y값을 구한다.
121      // p.s. 게임 내 월드블록 배열에서 xz 평면에서의 데이터만 있으면 된다.
122      int curHeight = Mathf.RoundToInt(moveObject.position.y);
123      for(int x = 0; x < MAP_SIZE_X; x++)
124          for(int z=0; z < MAP_SIZE_Z; z++)
125          {
126              int jumpedHeight = curHeight + 1;
127              if (jumpedHeight < worldBlockData.GetLength(2))
128              {
129                  if (worldBlockData[x, jumpedHeight, z].isRendered){
130                      pathFindMapData[x, z].isJumped = true;
131                      pathFindMapData[x, z].worldCoordY = jumpedHeight;
132                  }
133              }
134              pathFindMapData[x, z].worldCoordY = CalcDepth(curHeight, x, z);
135          }
136  }
137
138  private int CalcDepth(int curHeight, int x, int z)
139  {
140      int height = 0;
141      for(int y = curHeight; y > 0; y--)
142      {
143          if (worldBlockData[x, y, z].isRendered)
144          {
145              height = y;
146              break;
147          }
148      }
149      return height;
150  }
151

```

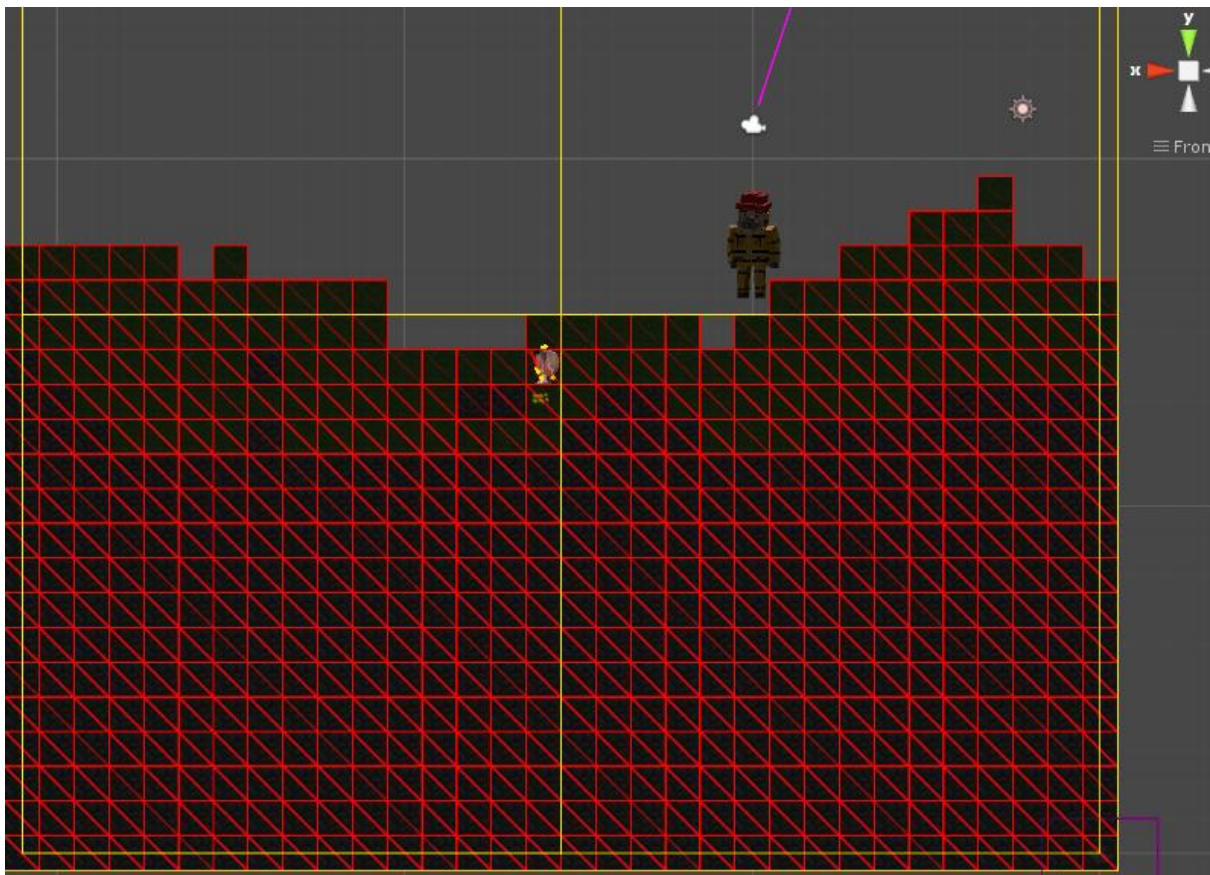
(* Astar 클래스의 일부분 입니다.)

게임 내 월드는 3차원 배열로 이루어져 있습니다. (이 부분에 대해서는 뒤에 가서 다시 한번 다루겠습니다.)

따라서, 3차원 공간은 XZ 평면을 Y만큼 쌓아 올렸다고 생각을 하고 이 부분에 맞춰 A* 알고리즘을 적용했습니다. 어떤 NPC가 동적으로 변하는 3차원 지형에서 길을 찾기 위해 NPC의 현재 위치한 높이가 변할 때마다 길을 찾는 메소드를 호출합니다. 이 메소드를 호출하기 앞서 위 사진에서 보이는 2개의 메소드를 차례로 불러줍니다.

BuildPathFindMapData() 메소드의 경우 길 찾기 알고리즘을 실행하기 전 XZ평면에서 존재하는 모든 장애물을 검색하여 closeList에 추가합니다. 그러나 현재 장애물 선정을 하는데 있어 여러 가지 방법을 테스트 중에 있어 현재 실려있는 소스코드는 수정 중에 있는 상태입니다. 따라서 미흡한 점이 있으니 이 점 참고바랍니다.

다음으로 월드 생성 (= 게임 내 존재하는 모든 블록) 코드를 살펴보겠습니다.



(* 3차원 배열 [x, y, z]로 이루어지는 월드의 XY평면에서 바라보는 모습입니다.)

(다음 장에 이어 계속 설명합니다.)

```

public class Block {

    private byte _type;
    public byte type
    {
        set { _type = value; }
        get { return _type; }
    }

    public CustomAABB aabb;

    private string _belongWorld;
    public string belongWorld
    {
        set { _belongWorld = value; }
        get { return _belongWorld; }
    }

    private Vector3 _worldPos;
    public Vector3 worldPos
    {
        set { _worldPos = value; }
        get { return _worldPos; }
    }

    private int _blockDataPosX;
    public int blockDataPosX

```

게임 내 존재하는 월드의 경우 3차원 배열로 이루어지며 각 배열 원소로는 다음과 같은 Blcok 클래스가 사용되고 있습니다.

각 블록은 AABB를 가지고 있습니다. 또한 각 블록은 월드좌표에서 가지는 실제 좌표값 및 월드 배열을 참조할 때 쓰이는

BlcokDataX,Y,Z 변수가 있습니다. 이 변수들의 경우 기존 프로젝트를 여러 부분에서 변경하면서 생긴 불필요한 코드이나 현재는 동작을 하기 위해 쓰이고 있습니다.

추후, 계획한 기능으로 새롭게 변경되고 테스트가 완료되면 해당 변수는 삭제될 예정입니다.

(ex: Octree 추가 후 지형 및 충돌생성에 대한 테스트 완료가 되면..)

(* Block 클래스 입니다. 일부 소스코드만 보여주고 있습니다.)

-> 다음 장에 이어서 설명합니다.

```

private void GenColumn(int x, int z)
{
    for (int y = 0; y < _chunkGroup.GetLength(1); y++)
    {
        // 유니티엔진에서 제공되는 씬에서 존재하는 모든 게임 오브젝트들의 중점은
        // 실제 게임 오브젝트의 정중앙이 된다.
        // 따라서, 유니티엔진에 맞춰서 오브젝트의 중점을 정중앙으로 하려면, 아래와
        // p.s. 이 프로젝트에서 1개의 block의 기준점(block을 생성할 때 쓰이는)은
        float coordX = x * chunkSize - 0.5f;
        float coordY = y * chunkSize + 0.5f;
        float coordZ = z * chunkSize - 0.5f;
        GameObject newChunk = Instantiate(_chunkPrefab, new Vector3(0, 0, 0),
                                           new Quaternion(0, 0, 0, 0)) as Chunk;

        newChunk.transform.parent = gameObject.transform;
        newChunk.transform.name = "Chunk_" + chunkNumber++;
        _chunkGroup[x, y, z] = newChunk.GetComponent("Chunk") as Chunk;
        _chunkGroup[x, y, z].world = this;
        _chunkGroup[x, y, z].worldDataIdxX = x * chunkSize;
        _chunkGroup[x, y, z].worldDataIdxY = y * chunkSize;
        _chunkGroup[x, y, z].worldDataIdxZ = z * chunkSize;
        _chunkGroup[x, y, z].worldCoordX = coordX;
        _chunkGroup[x, y, z].worldCoordY = coordY;
        _chunkGroup[x, y, z].worldCoordZ = coordZ;
        _chunkGroup[x, y, z].Init(worldTileDataFile);
    }
}

```

(* 지형생성 부분에 대한 일부 코드 입니다.)

3차원 배열에 담겨있는 모든 블록을 게임 내 오브젝트로 만들면 실제 메모리 증가와 퍼포먼스 저하가 발생합니다. 따라서, Chunk 라는 구조를 도입하여 이 부분을 개선했습니다. 원리는 다음과 같습니다.

N개의 Block은 1개의 Chunk로 이루어집니다. 그리고 Chunk 클래스는 하나의 커다란 Mesh 덩어리 입니다. 이 클래스에는 Mesh를 구성하는 여러 개의 Vertex 와 텍스처 좌표를 가지고 있습니다.

만약 월드가 X:128, Y:128, Z:128의 크기에 1개의 Chunk size 가 8이라면 Chunk는 8 / 128 하여 16개가 존재하게 됩니다. (Chunk 크기 8 * 8 * 8) 결국 게임 내에 실제 존재하는 게임 오브젝트의 수가 줄어드는 효과가 생깁니다.

(다음 장에 이어서 설명합니다.)


```

/// <summary>
/// 게임내 블록 덩어리를 의미하는 Chunk 클래스.
/// -> 1개의 Chunk는 N개의 면으로 구성되어 하나의 메쉬로 생성된다.
/// </summary>
public class Chunk : MonoBehaviour
{
    private World _world;
    public World world
    {
        set { _world = value; }
        get { return _world; }
    }

    private List<Vector3> newVertices = new List<Vector3>();
    private List<int> newTriangles = new List<int>();
    private List<Vector2> newUV = new List<Vector2>();
    // 256 x 256 tile 기준. 1tile(16pixel) 이 차지하는 텍스처 좌표값.
    private float tUnit = 0.0625f;
    private Vector2 texturePos;
    private Mesh mesh;
    private int faceCount;

    private int chunkSize = 0;

```

(* Chunk 클래스 입니다. 일부만 발췌했습니다.)

Chunk 클래스에서는 이제 실제 지형을 생성하기 위한 동작을 하게 됩니다. 간단하게 주어진 Chunk size를 기준으로 3차원 순회를 하게 됩니다. (아래 그림 입니다.)

```

private void GenerateMesh()
{
    for (int relativeX = 0; relativeX < chunkSize; relativeX++)
    {
        for (int relativeY = 0; relativeY < chunkSize; relativeY++)
        {
            for (int relativeZ = 0; relativeZ < chunkSize; relativeZ++)
            {
                int blockIdxX, blockIdxY, blockIdxZ;
                blockIdxX = relativeX + _worldDataIdxX;
                blockIdxY = relativeY + _worldDataIdxY;
                blockIdxZ = relativeZ + _worldDataIdxZ;
                //This code will run for every block in the chunk
                if (CheckBlock(blockIdxX, blockIdxY, blockIdxZ) != 0)
                {

```

Chunk는 월드 배열과 위치의 차이가 있으므로 offset 계산하여 더해줍니다.

```

CubeTopFace(worldCoordX, worldCoordY, worldCoordZ, CheckBlock(blockIdxX, blockIdxY, blockIdxZ));
//if (CheckBlock(blockIdxX, blockIdxY - 1, blockIdxZ) == 0)
CubeBotFace(worldCoordX, worldCoordY, worldCoordZ, CheckBlock(blockIdxX, blockIdxY, blockIdxZ));
CubeNorthFace(worldCoordX, worldCoordY, worldCoordZ, CheckBlock(blockIdxX, blockIdxY, blockIdxZ));
CubeSouthFace(worldCoordX, worldCoordY, worldCoordZ, CheckBlock(blockIdxX, blockIdxY, blockIdxZ));
CubeEastFace(worldCoordX, worldCoordY, worldCoordZ, CheckBlock(blockIdxX, blockIdxY, blockIdxZ));
CubeWestFace(worldCoordX, worldCoordY, worldCoordZ, CheckBlock(blockIdxX, blockIdxY, blockIdxZ));

// points 배열은 실제 블록을 생성할 때 쓰이는 8개의 포인트로 실제 월드 좌표값이다.
// 따라서, 이를 이용해 블록의 AABB의 Min, Max Extent 값을 정한다.
Vector3[] points = new Vector3[8];
points[0] = new Vector3(worldCoordX, worldCoordY, worldCoordZ);
points[1] = new Vector3(worldCoordX + 1, worldCoordY, worldCoordZ);
points[2] = new Vector3(worldCoordX + 1, worldCoordY, worldCoordZ + 1);
points[3] = new Vector3(worldCoordX, worldCoordY, worldCoordZ + 1);
points[4] = new Vector3(worldCoordX, worldCoordY - 1, worldCoordZ);
points[5] = new Vector3(worldCoordX + 1, worldCoordY - 1, worldCoordZ);
points[6] = new Vector3(worldCoordX + 1, worldCoordY - 1, worldCoordZ + 1);
points[7] = new Vector3(worldCoordX, worldCoordY - 1, worldCoordZ + 1);
// 블록 생성시 정중앙을 맞추기 위해 추가했던 offset 값을 제거한다.
_world.worldBlockData[blockIdxX, blockIdxY, blockIdxZ].worldPos = new Vector3(worldCoordX, worldCoordY, worldCoordZ);
_world.worldBlockData[blockIdxX, blockIdxY, blockIdxZ].blockDataPosX = worldCoordX;
_world.worldBlockData[blockIdxX, blockIdxY, blockIdxZ].blockDataPosY = worldCoordY;
_world.worldBlockData[blockIdxX, blockIdxY, blockIdxZ].blockDataPosZ = worldCoordZ;
_world.worldBlockData[blockIdxX, blockIdxY, blockIdxZ].belongWorld = world.worldCoordX;
_world.worldBlockData[blockIdxX, blockIdxY, blockIdxZ].aabb.MakeAABB(points);

```

순회를 하면서 각 부분에 블록을 생성하게 됩니다. 사이즈 1를 가지는 정육면체 이기에 6개의 면을 생성합니다. 그리고 생성한 블록 사이즈에 맞춰 AABB 도 크기를 맞추어 생성합니다.

여기까지 게임을 구현하는데 있어 핵심적인 기능인 지형 생성 및 충돌에 대해 다뤄봤습니다.

이외에도 게임을 구성하는 여러 기능이 있습니다만, 내용이 너무 길어져서 지루해 질 수 있기에 간단하게 글로 설명만 하겠습니다.

1. 게임 내 플레이어는 블록을 파괴하여 Element 아이템을 얻습니다. 이 아이템을 모아 특정 아이템을 조합하여 만들어 낼 수 있습니다. 이 기능은 포함된 플레이 영상에서 확인 가능합니다.
2. 게임 내 모든 아이템 데이터, 캐릭터 데이터는 Json 포맷의 파일로 저장하고 있습니다. 이를 JsonObject 라는 라이브러리를 이용해 읽어 들여 게임 내 데이터로 활용하고 있습니다.
3. 플레이어가 획득한 아이템은 게임 내 존재하는 DB가 있습니다. 이에 저장되며 DB를 이용하기 위해 Sqlite3를 사용하고 있습니다.
4. 게임 내 월드 데이터를 저장하고 불러오는 기능이 있습니다. 게임 내 배열은 3차원 으로 이루어져 있으며 byte 타입을 가지고 있습니다. 따라서, 이 배열을 시리얼라이즈 하여 외부 파일로 저장 및

불러오기를 하고 있습니다. 그리고 이 맵 데이터가 커지면 저장하려는 외부파일의 크기도 비례해서 커지기 때문에, 저장 및 불러오기 시에 지형 데이터를 Lzf4 라는 압축 알고리즘을 이용하여 크기에 대한 이점을 보고 있습니다.

OneCoinOnePlay

게임 장르 : 논 타겟팅 액션 게임

사용 엔진 : Unity3D

사용 언어 : C#

사용 라이브러리 : ITween, NGUI, JObject

구글 플레이 서비스 유/무 : 유

목표 플랫폼 : 안드로이드 (모바일)

프로젝트 소스 링크 : <https://github.com/opk4406opk/OneCoinOnePlay>

구글 스토어 링크 :

<https://play.google.com/store/apps/details?id=com.KojeomStudio.OneCoinOnePlay&hl=ko>

설명 : 몰려드는 몬스터를 해치우며 오랜 시간을 버텨야 하는 룰을 가진 논 타겟팅 디펜스 게임입니다.

시중에 있는 여러 디펜스 게임과는 다르게 액션성향을 위주로 제작하였습니다. 사실 이 프로젝트를 시작할 때는 단순히 구글 플레이 서비스 SDK 를 테스트하고자 하는 프로젝트였으나 진행 중에 정이 많이 생겨서 지속적인 플레이가 가능한 게임으로 제작했습니다.

코드설명(일부) :

게임의 가장 핵심인 난이도 설정에 있어서 등장하는 몬스터의 숫자는 프로젝트 내부에 Json 포맷의 파일이 있습니다. 이 파일을 참조하여 흘러가는 시간에 따라 등장하는 숫자가 증가함에 따라 난이도를 조절합니다. (Data Driven Development) 난이도를 증가하는 일부 다른 요소들은 굳이 외부 파일로 두지 않아도 될 만큼 간단한 요소였기에 하드코딩을 한 부분이 있습니다.

```

1 public void Init()
2 {
3     jsonDataSheet = new List<Dictionary<string, string>>();
4     gameDifftextFile = Resources.Load("textAssets/gameDiff") as TextAsset;
5
6     gameDiffJsonObj = new JSONObject(gameDifftextFile.text);
7     AccessData(gameDiffJsonObj);
8
9     int maxGameLevel = jsonDataSheet.Count;
10    _gameLevelData = new GameLevel[maxGameLevel];
11    for(int idx = 0; idx < jsonDataSheet.Count; ++idx)
12    {
13        _gameLevelData[idx] = new GameLevel();
14        string levelName;
15        jsonDataSheet[idx].TryGetValue("levelName", out levelName);
16        string level;
17        jsonDataSheet[idx].TryGetValue("level", out level);
18        string mobMaxNum;
19        jsonDataSheet[idx].TryGetValue("mobMaxNum", out mobMaxNum);
20
21        _gameLevelData[idx].gameLevelName = levelName;
22        _gameLevelData[idx].gameLevel = int.Parse(level);
23        _gameLevelData[idx].gameMobMaxNum = int.Parse(mobMaxNum);
24    }
25 }

```

(※ 게임 난이도 설정을 읽어들이는 부분)

위 코드에서 보듯이 TextAsset으로 리소스 로딩을 하여 사용하고 있습니다. Json 파일을 Encode/Decode 하는건 JSONObject 라는 라이브러리 예셋을 활용했습니다.

모바일 기기 특성상 메모리와 CPU 의 한계가 분명합니다. 따라서 Unity 엔진에서 제공하는 오브젝트 동적 생성인 Instantiate 같은 메소드는 게임중간에 쓰지않고 Start() 메소드나 별도의 Init() 메소드를 통해서만 사용했습니다. 그리고 제네릭 컬렉션을 반복해야 하는 경우에는 foreach 의 경우 반복시마다 Object 생성이 되기에 가비지컬렉터의 성능에 부하를 줍니다. 따라서 크게 가독성을 해치지 않는선에서는 for 문을 사용했습니다.

몬스터 생성, 플레이어 공격 미사일, 함정등 게임중간에 쓰이는 여러 오브젝트들은 모두 게임 시작시 생성 및 불러오며 이후 게임진행중에는 메모리풀 같은 기법으로 재사용을 원칙으로 했습니다.

```

1 IEnumerator MonSpawnProcess()
2 {
3     int idx = 0;
4     while(true)
5     {
6         yield return new WaitForSeconds(spwanTime);
7         if (idx >= _maxMonsterNum) idx = 0;
8         if (monsterList[_curGameLevel][idx].activeSelf == true)
9         {
10             idx++;
11             continue;
12         }
13         monsterList[_curGameLevel][idx].SetActive(true);
14         monsterList[_curGameLevel][idx].transform.position = spwanPositions[Random.Range(0, 3)].position;
15         monsterList[_curGameLevel][idx].GetComponent<Zombie>().ReSet();
16         idx++;
17     }
18 }
19
20 private void CreateMonster(int gameLevel, int maxCreateNum)
21 {
22     for (int curMonsterNum = 0; curMonsterNum < maxCreateNum; curMonsterNum++)
23     {
24         int randomNum = Random.Range(0, 3);
25         GameObject prefab = monsterPrefabGroup[gameLevel].GetMonster((MonstersGroup.MONSTER_TYPE)randomNum);
26         GameObject monster = Instantiate(prefab,
27             new Vector3(0, 0, 0), new Quaternion(0, 0, 0, 0)) as GameObject;
28         monster.GetComponent<Zombie>().Init();
29         monster.SetActive(false);
30         monster.transform.parent = inGameMonsters;
31         monsterList[gameLevel].Add(monster);
32     }
33 }

```

(※ 몬스터 생성 관련 소스, 플레이어 미사일 및 기타 오브젝트 생성 및 사용은 위와 비슷합니다.)

유니티엔진의 경우 C# 스크립트 자체가 하나의 컴포넌트로 동작하기에 프로그램의 유일한 시작점을 제어하기가 불편합니다. 따라서, 게임 전반적으로 시작을 제어하는 스크립트를 별도 제작 후 이를 시점으로 활용했습니다. 아래 소스를 참고해주세요.

```

1 private void GameInit()
2 {
3     gameTimer.InitGameTimer();
4
5     gameDiffLevel.Init();
6     arr_gameLevel = gameDiffLevel.gameLevelData;
7     curGameDiffLevel = arr_gameLevel[0].gameLevel;
8
9     gameMonGenerator.Init(arr_gameLevel);
10
11     meteoGenerator.Init();
12     meteoGenerator.genTime = 3.0f;
13     meteoGenerator.genMeteoNum = 1;
14     meteoGenerator.StartGenMeteoProcess();
15
16     for (int idx = 0; idx < 12; idx++)
17         misTrapManager.GetTrap(idx).Init();
18
19     StartCoroutine(GameLevelController());
20 }

```

(※ 게임의 시작 및 초기화. - GameManager.cs 스크립트 발췌.)

#WatchOutCubes

게임 장르 : 간단한 피하기 게임

사용 엔진 : Unity3D

사용 언어 : C#

사용 라이브러리 : ITween, NGUI3.x, JObject

구글 플레이 서비스 유/무 : 무

목표 플랫폼 : 안드로이드

프로젝트 소스 링크 : <https://github.com/opk4406opk/AvoidFallingCubes>

구글 스토어 링크 : <https://play.google.com/store/apps/details?id=com.kojeomstudio.watchoutCubes>

설명 : 간단한 피하기 게임 입니다. WatchOut-UnityChan 과 비슷하게 하늘에서 떨어지는 물체를 피하며 도망다니는 게임 입니다.

코드설명 :

```
1 using UnityEngine;
2 using System.Collections;
3 using System.Collections.Generic;
4
5 public class FallingPattern
6 {
7     public string name;
8     public int[] pattern;
9 }
10
11 public class FallingPatternManager : MonoBehaviour {
12
13     private JSONObject fallingPatternJsonObj;
14     private TextAsset fallingPatternText;
15     public Queue<FallingPattern> patterns
16     {
17         get { return _patterns; }
18     }
19     private Queue<FallingPattern> _patterns = new Queue<FallingPattern>();
20
21     public void Init()
22     {
23         fallingPatternText = Resources.Load("textAssets/fallingPattern") as TextAsset;
24         fallingPatternJsonObj = new JSONObject(fallingPatternText.text);
25         AccessData(fallingPatternJsonObj);
26     }
27
28     private void AccessData(JSONObject jsonObj)
29     {
30         switch (jsonObj.type)
31         {
32             case JSONObject.Type.OBJECT:
33                 int listCount = jsonObj.list[0].Count;
34
35                 FallingPattern patternObj = new FallingPattern();
36                 patternObj.name = jsonObj.keys[0];
37                 patternObj.pattern = new int[listCount];
38                 for(int idx = 0; idx < listCount; ++idx)
39                 {
40                     patternObj.pattern[idx] = (int)jsonObj.list[0].list[idx].n;
41                 }
42                 _patterns.Enqueue(patternObj);
43                 break;
44             case JSONObject.Type.ARRAY:
45                 for (int idx = 0; idx < jsonObj.Count; ++idx)
46                 {
47                     AccessData(jsonObj.list[idx]);
48                 }
49                 break;
50             default:
51                 Debug.Log("Json Level Data Sheet Access ERROR");
52                 break;
53         }
54     }
55 }
```

피하기 게임의 가장 중요한 건 피해야 하는 물체들의 등장 패턴 입니다. 이를 위해 Json 포맷의 파일로 패턴을 만든 후에 이 파일을 읽어 들여 떨어지는 물체들의 패턴을 만들어내고 있습니다.

게임 내에서 쓰이는 패턴은 N개의 블록으로 이루어져 있으며, 모바일 디바이스의 제한적인 성능을 위해 블록을 위한 메모리풀을 만들어 이용하고 있습니다. 간단하게 큐를 이용했습니다. 생성된 특정 패턴의 블록들이 모두 파괴되면 자동으로 다음 패턴으로 넘어가게끔 C#에서 제공하는 event를 이용하고 있습니다.

```
33     private void GenerateFallingCubes()
34     {
35         for (int x = minX; x < maxX; x++)
36         {
37             for (int z = minZ; z < maxZ; z++)
38             {
39                 Vector3 cubePos = new Vector3(x, fixedY, z);
40                 GameObject newCube = Instantiate(fallingCubePrefab, cubePos, new Quaternion());
41                 newCube.name = "fallingCube" + cubeIdx;
42                 newCube.GetComponent<FallingCube>().Init(cubePos, cubeIdx);
43                 newCube.GetComponent<FallingCube>().afterHit += FallingNextPattern;
44                 newCube.transform.parent = gameObject.transform;
45                 newCube.SetActive(false);
46                 _fallingCubePool.Add(newCube.GetComponent<FallingCube>());
47                 cubeIdx++;
48             }
49         }
50     }
51
52     public void FallingNextPattern()
53     {
54         curFalledCubes++;
55         if (curFalledCubes >= curPatternMaxCubes) NextPattern();
56     }
57     private void NextPattern()
58     {
59         FallingPattern newPattern = patternManager.patterns.Dequeue();
60         curPatternMaxCubes = newPattern.pattern.Length;
61         for (int idx = 0; idx < newPattern.pattern.Length; idx++)
62         {
63             _fallingCubePool[newPattern.pattern[idx]].gameObject.SetActive(true);
64             _fallingCubePool[newPattern.pattern[idx]].StartFalling();
65         }
66         curFalledCubes = 0;
67         patternManager.patterns.Enqueue(newPattern);
68     }
69
70 }
```

ProtectPlanet (가칭)

게임 장르 : 터치 기반 디펜스 게임

사용 엔진 : Unity3D

사용 언어 : C#

사용 라이브러리 : ITween, NGUI3.x, JObject

구글 플레이 서비스 유/무 : 유 (예정)

목표 플랫폼 : 안드로이드

프로젝트 소스 링크 : <https://github.com/opk4406opk/ProtectCastle>

설명 : 행성을 침공하는 드로이드(=몬스터)를 파괴하는 디펜스류 게임입니다. 기본적으로 터치입력을 통해 드로이드를 파괴합니다.

현재 개발중에 있으며 기본적인 프로토타입까지 완성했습니다. 구글 스토어 출시를 목표로 하고 있습니다.

코드설명 :

```
6 public abstract class Node
7 {
8     public abstract bool Invoke();
9 }
10
11 public class CompositeNode : Node
12 {
13     public override bool Invoke()
14     {
15         throw new NotImplementedException();
16     }
17
18     public void AddChild(Node node)
19     {
20         childrens.Push(node);
21     }
22
23     public Stack<Node> GetChildrens()
24     {
25         return childrens;
26     }
27     private Stack<Node> childrens = new Stack<Node>();
28 }
29
30 public class Selector : CompositeNode
31 {
32     public override bool Invoke()
33     {
34         foreach (var node in GetChildrens())
35         {
36             if (node.Invoke())
37             {
38                 return true;
39             }
40         }
41         return false;
42     }
43 }
44
45 public class Sequence : CompositeNode
46 {
47     public override bool Invoke()
48     {
49         foreach (var node in GetChildrens())
50         {
51             if (!node.Invoke())
52             {
53                 return false;
54             }
55         }
56         return true;
57     }
58 }
```


행성을 침공하는 드로이드들을 파괴하는게 주 목적인 게임이기에, 등장하는 드로이드의 행동패턴이 중요합니다. 또한 재사용성이 높아야 여러 패턴을 제작하기에 편리합니다. 따라서, 기존 FSM 을 이용 하는게 아닌 Behavior Tree를 사용해 개발의 편의성을 높이고자 했습니다.

```
130 public class DeadProcess : Node
131 {
132     public MonsterController monController
133     {
134         set { _monController = value; }
135     }
136     private MonsterController _monController;
137     public override bool Invoke()
138     {
139         _monController.DeadPrcoess();
140         return true;
141     }
142 }
143
144 public class IsDead : Node
145 {
146     public MonsterController monController
147     {
148         set { _monController = value; }
149     }
150     private MonsterController _monController;
151     public override bool Invoke()
152     {
153         if (_monController.IsDead())
154         {
155             return true;
156         }
157         else
158             return false;
159     }
160 }
161
162 public abstract class BT_base : MonoBehaviour
163 {
164     public abstract void Init();
165     public abstract void StartBT();
166     public abstract void StopBT();
167     public abstract IEnumerator BehaviorProcess();
168 }
```

그리고 BT_base 라는 추상클래스를 상속받아 타입별 행동트리를 제작하고 있습니다. 현재 기본타입의 행동 트리만 작성하여 테스트 중에 있습니다. 그리고 실제 드로이드의 움직임을 제어하려면 유니티엔진 에서 제공하는 클래스를 이용해야 합니다. 따라서 행동트리 클래스에 이 부분을 모두 작성하면 코드가 복잡해진다 판단하여 해당 부분을 분리하여 작성했습니다. 다음은 드로이드의 실제 행동을 컨트롤 하는 클래스이며 이 행동 트리에서는 이 클래스에 접근하여 오브젝트의 움직임을 제어합니다.

```

42     public bool IsDead()
43     {
44         return isDead;
45     }
46
47     public void DeadPrcoess()
48     {
49         OnDeadEffect();
50     }
51
52     public void MoveForTarget()
53     {
54         Vector3 dir = target.position - trans.position;
55         dir.Normalize();
56         trans.position += dir * Time.deltaTime;
57     }
58
59     public void RotAroundTarget()
60     {
61         trans.RotateAround(target.position, target.up, 1.0f);
62     }
63
64     public bool IsTooCloseTarget()
65     {
66         float dist = Vector3.Distance(trans.position, target.position);
67         if (dist <= 5.0f) return true;
68         else return false;
69     }
70
71     public void StartAttack()
72     {
73         attackController.StartAttack();
74     }
75     public void StopAttack()
76     {
77         attackController.StopAttack();
78     }
79
80 }

```

위 코드는 앞서 얘기한 실제 드로이드의 움직임을 제어하는 클래스입니다. 다만, 공격하는 클래스는 따로 빼놓아서 코루틴을 통해 제어 하고 있습니다. 다음에 등장하는 이미지는 드로이드가 미사일을 발사해 공격하는 클래스 이며, 미사일은 메모리풀을 통해 관리하고 있습니다.

```

16     public int maxQuantity = 5;
17     [Range(1.0f, 5.0f)]
18     public float attackIntervalTime;
19
20     public void Init () {
21         attackProcess = AttackProcess();
22         parentTrans = GameObject.FindGameObjectWithTag("missileBucket").transform;
23         target = GameObject.FindGameObjectWithTag("target");
24
25         for(int idx = 0; idx < maxQuantity; idx++)
26         {
27             GameObject mis = Instantiate(missilePrefab, attacker.position,
28                 new Quaternion(0, 0, 0, 0)) as GameObject;
29             mis.transform.parent = parentTrans;
30             EffectSettings settings = mis.GetComponent<EffectSettings>();
31             settings.Target = target;
32             settings.EffectDeactivated += (n, e) =>
33             {
34                 mis.SetActive(false);
35             };
36
37             mis.SetActive(false);
38             missiles.Enqueue(mis);
39         }
40     }
41
42     public void StartAttack()
43     {
44         if (isAttackNow == false)
45         {
46             StartCoroutine(attackProcess);
47             isAttackNow = true;
48         }
49     }
50
51     public void StopAttack()
52     {
53         StopCoroutine(attackProcess);
54         isAttackNow = false;
55     }
56
57     private IEnumerator AttackProcess()
58     {
59         while (true)
60         {
61             if(missiles.Count > 0)
62             {
63                 GameObject mis = missiles.Dequeue();
64                 if ((mis != null) && (!mis.activeSelf))
65                 {
66                     mis.transform.position = attacker.transform.position;
67                     mis.SetActive(true);
68                     missiles.Enqueue(mis);
69                 }
70             }
71             yield return new WaitForSeconds(attackIntervalTime);
72         }
73     }
74 }

```

게임 내 등장 하는 드로이드(=몬스터), 미사일등 가능하면 메모리풀을 이용해 관리하고 있습니다. 또한 이펙트도 마찬가지로 메모리풀을 이용하고 있습니다.

기타 개인 프로젝트

위에서 소개한 5개의 프로젝트는 직접 개발한 게임이며 그 중 3개는 현재 구글 스토어에 출시 중입니다. 이제부터 소개하는 기타 개인 프로젝트는 게임 뿐만 아니라 개인적인 용도에 맞추어 만들어진 프로그램도 있습니다.

1) RunUnityChan

- 구글 플레이 서비스 및 페이스북 SDK 두 개를 연동하여 쓰고자 했던 테스트 프로젝트입니다.
- 현재 구글 스토어에 출시되어 있는 상태입니다.



2) 모바일용 웹크롤러 앱

- 모바일 장치에서 웹크롤링을 할 수 없을까 하는 호기심에서 출발한 프로젝트입니다. 현재 입력 받은 최상위 URL을 통해 이미지 파일만을 골라 추출하는 기능을 가진 APP을 개발한 상태입니다. 지속적으로 업데이트 중입니다.

언어	Java 1.8
IDE	Android Studio
라이브러리	Jsoup
GitHub 링크	https://github.com/opk4406opk/KojeomWebImageCrawler

- 아래 그림은 간단한 구조를 보여주고 있습니다. 추후 온라인상에 있는 웹크롤러 아키텍처 문서를 참고하여 특정 웹페이지의 모든 이미지를 추출하여 계층구조로 저장하는 부분을 만들생각입니다.

[KojeomWebImageCrawler](#) / [app](#) / [src](#) / [main](#) / [java](#) / [com](#) / [kojeomwebimagecrawler](#) / [kojeoms](#)

 opk4406opk # 모바일 및 PC용 URL 추출 소스 변경. ...	
..	
 DataOfJob	# 멀티스레드 테스트 프로젝트 시작.
 Files	# URL을 통해 비트맵 생성 및 Logger 기능 테스트.
 Network	# 모바일 및 PC용 URL 추출 소스 변경.
 URLExtract	# 모바일 및 PC용 URL 추출 소스 변경.
 Utils	# URL 추출&정규식 수정 및 비트맵 생성 크기 수정.
 WorkerThread	# URL을 통해 비트맵 생성 및 Logger 기능 테스트.
 MainActivity.java	# URL 추출&정규식 수정 및 비트맵 생성 크기 수정.
 WebImageCrawlManager.java	# URL 추출&정규식 수정 및 비트맵 생성 크기 수정.

3) 개인적인 용도로 쓰이는 기타 프로그램들.

- 파일 이름 변환기, 마비노기 아르바이트 타이머 등이 있으며 평소에도 간단하게 필요한 기능들은 만들어서 쓰고 있습니다. 정말 간단하게 제작했기에 따로 GitHub 링크는 적어놓지 않겠습니다.

공부중인 OpenSource 프로젝트.

1. Box2D

:: 물리엔진의 구성과 많은 오브젝트를 동시다발적으로 다루어야 하는 만큼 메모리 관리에 대해 궁금하여 살펴보고 있습니다.

2. Cralwer4j

:: 자바로 작성된 오픈소스 웹크롤러 입니다. 웹크롤러 아키텍처와 기능들을 어떻게 구현했나 살펴보기 위해 살펴보는 프로젝트입니다.