(플레이 화면에 구현되어야 할 내용만을 담았습니다 / Assets -> Scripts -> Main 폴더 안에 있습니다)

(OOP 개념을 사용하지 않기 위해서 (Method/Field) 대신 (Function/Variable)이라는 어휘를 사용한 점 양해 부탁드립니다)

(Public이 붙은 것은 다른 Script에서 접근할 수 있는 것을 의미하며, Private가 붙은 것은 다른 Script에서 접근할 수 없는 것을 의미합니다)

# 바뀐거(2018/07/24 오후 4시)

1. Headquarter의 최대 에너지 항목을 기획서에 맞게 삭제
2. GameManager에서 HeadQuarter.cs에게 바로 오브젝트 소환 명령을 내리던 구조에서 GameManager는 명령만 내리고 HeadQuarter.cs에서 실제로 오브젝트를 소환하도록 변경
3. GameManager.cs의 Start() 함수에서 게임을 시작할 때 카메라 크기에 맞춰 HQ를 적절한 위치에 소환하는 기능 추가.
4. 주문 쓸 때 소환 막는 기능 제거(모든 주문은 즉발)
5. HQ의 X축 크기를 저장하는 변수 추가 (Start()에서 계산함)
6. GameManager.cs에 플레이어가 선택한 유닛을 저장하는 List 변수를 만듬.

# GameManager.cs

* 게임의 전반적인 수치를 관리하는 매니저 역할입니다.
* 게임 수치에 맞춰 UI 요소를 관리합니다(Private 함수)
* 플레이어가 버튼을 누르면 그걸 감지해서 HeadQuarter.cs의 Unit 소환 함수를 호출해야 합니다.
* 게임 내에서 GameManager Object의 Component로 작용합니다.
* 다음과 같은 기능을 제공하는 public 함수를 구현해야 합니다.

## Public Function:

**Public Vector3 Get\_RGBValue()**

* + Unit 및 UI가 호출하며, 현재 환경의 RGB 값을 Vector3형으로 반환합니다.
  + Input Parameter: None
  + Output Parameter
    - Vector3(int Red, int Green, int Blue): 현재 환경값 수치를 반환합니다.

**Public void Set\_RGBValue(Vector RGB)**

* + Unit 및 UI가 호출하며, 현재 환경의 RGB 값을 입력받은 RGB의 값으로 변환합니다. 바꾼 후 RGB 값이 최대값이나 최소값을 넘어가는 경우 각각 최대값, 최소값으로 조정합니다.
  + Input Parameter: Vector RGB(int R, int G, int B): 바꾸고 싶은 RGB 값입니다.
  + Output Parameter: None

**Public void Change\_RGBValue(Vector dRGB)**

* + Unit 및 UI가 호출하며, 현재 환경의 RGB 값에 입력받은 RGB 값을 더해줍니다. 바꾼 후 RGB 값이 최대값이나 최소값을 넘어가는 경우 각각 최대값, 최소값으로 조정합니다.
  + Input Parameter: Vector dRGB(int R, int G, int B): 현재 환경 변수에 각각 R, G, B만큼 더해줍니다.
  + Output Parameter: None

**Public void Destroyed(int Team)**

* + 호출되면 Team 번호의 HQ가 파괴되었다고 간주하고 승리 판정을 내립니다.
  + Input Parameter
    - int Team: HQ가 파괴된 Team의 번호입니다.
  + Output Parameter: None

## Private Function:

**Private Vector3 Limit\_RGBValue(Vector3 RGB)**

* + RGB값을 입력 받으면 미리 설정된 최대/최솟값을 넘지 않도록 제한을 걸어준 뒤 그 결과를 리턴하는 함수.
  + Input Variable: Vector3 RGB
  + Output Parameter:
    - Vector3: 최대/최솟값을 넘지 않도록 값이 조율된 결과입니다.

**Private void Start()**

* + 게임이 시작될 때 단 한번 호출되는 함수
  + RGB값을 초기화 합니다.
  + HQ의 위치를 카메라의 크기에 맞게 적절히 배치합니다.

**Private void Update()**

* + 매 Frame마다 호출되는 기본 함수
  + 플레이어의 키 입력을 감지하고, 입력된 키에 따라서 적절한 행동을 취합니다.
  + Esc: 일시정지
  + Player 1은 Q, W, E를, Player 2는 U, I, O를 누르면 각 HeadQuarter.cs의 Summon\_Unit Function을 호출한다. (인자로 각각 1,2,3을 넘겨주며, 미리 할당된 세 종류의 크리처를 소환한다)
  + Player 1은 1, 2, 3을 Player 2는 7, 8, 9를 누르면 각 HeadQuarter의 SpellManager.cs의 Spell\_Load을 호출합니다. (인자로 각각 1,2,3을 넘겨주며, 미리 할당된 세 종류의 주문을 사용할 준비를 합니다.)

Public Variable

Private Variable

* Vector3 Max\_Env, Min\_Env: 각 환경요소의 최대값/최솟값을 나타냅니다. Inspector에서 편집합니다.
* Vector3 Cur\_Env: 각 환경요소의 현재값을 나타냅니다.
* Float Init\_R,Init\_G,Init\_B: 각 환경요소의 초기값을 나타냅니다, 바뀔 일이 거의 없으므로 Inspector에서 편집하지 않고 코드 상에서 직접 편집합니다.
* private GameObject P1\_HQ, P2\_HQ; 플레이어 1, 2의 HQ 게임 오브젝트를 저장하는 변수입니다. Start()함수에서 HQ의 위치를 초기화 하면서 할당됩니다.

# HeadQuarter.cs

* 각 HQ를 관리하는 서브 매니저 역할입니다. (HQ는 각 플레이어별로 한 개씩 있습니다)
* UI가 아니라 하나의 오브젝트이며, 게임이 시작될 때(Scene이 Load 될때) GameManager가 Prefab에서 HeadQuarter를 적절한 위치에 배치합니다.
* HQ의 체력, 오브젝트 소환, 피격시 체력 감소, 파괴되었을 때 GameManager에게 파괴되었음을 알려야 합니다. (GameManager의 Destroyed 함수를 호출)
* HQ에는 ‘에너지’ 자원이 매 초마다 1씩 생성되며, 이 자원을 소비해서 스킬 사용/ 크리쳐 소환을 수행합니다.
* 모든 종류의 Unit을 Prefab화 하여 HeadQuarter.cs에서 정해진 종류의 Prefab을 소환할 수 있게 합니다.

## Public Function:

* **Public GameObject Summon\_Unit(int ID)**
  + GameManager에 의해 호출되며, 호출되면 ID 번호에 맞는 Unit GameObject를 생성합니다.
  + (Is\_Casting || Is\_Pause) == False이며, 에너지가 충분할 때만 소환을 실행합니다.
  + Input Parameter:
    - Int ID: 소환할 Unit의 미리 지정된 ID를 의미합니다. ID에 해당하는 종류의 Unit을 소환합니다.
  + Output Parameter:
    - 소환한 GameObject를 리턴합니다, 소환에 실패하면 null을 리턴합니다.
* **Public void Hit(float damage)**
  + 호출되면 피격되었다고 간주하고 체력을 damage만큼 경감합니다.
  + Input Parameter
    - Float damage: 얼마의 데미지를 입었는가를 나타냅니다, 체력이 이 수치만큼 감소합니다.
  + Output Parameter: None
* **Public float HQ\_Health()**
  + 현재 HeadQuarter의 체력을 Return 합니다.
  + Input Parameter: None
  + Output Parameter
    - 현재 HeadQuarter의 체력(health)를 리턴합니다.

**Public void Pause\_Toggle(bool Is\_Pause)**

* + GameManager에 의해 호출되며. 호출되면 게임이 일시정지됩니다. TimeScale을 0으로 설정합니다.
  + Input Parameter:
    - Bool Is\_Pause: 일시정지 여부를 나타냅니다, True면 게임을 일시정지하며, False면 일시정지를 해제합니다.
  + Output Parameter: None
* **Public void Finish\_Game()**
* **Public int Get\_Unit\_Count()**
  + 현재 Unit 수를 리턴합니다.
  + Input Parameter: None
  + Output Parameter:
    - Int Unit\_Count: 현재 유닛 수
* **Public void Unit\_Dead(int ID)**
  + Unit.cs에 의해 호출되며, Unit\_Count[ID]를 1 감소시킵니다.
    - Unit의 비활성화는 Unit.cs의 Private 함수가 진행하며, 이 함수는 Unit\_Count를 1 감소시키는 역할만 수행합니다.
* **Public void Toggle\_Is\_Casting()**
  + GameMamager.cs나 같은 Object에 붙어있는 SpellManager.cs 에서 호출하며 Is\_Casting의 값을 현재 값의 반대 값(True 🡨🡪 False)으로 변경합니다.
  + Input / Output Parameter: None

## Private Function:

**Private void Start()**

* + 오브젝트가 생성이 완료된 후 단 한번 호출되는 함수
  + Is\_Casting의 값을 True로 변경합니다.
  + 동적 배열인 Unit\_Count[MAX\_UNIT\_VARIATION], Unit\_Cost[MAX\_UNIT\_VARIATION], Unit\_Summon\_Time[MAX\_UNIT\_VARIATION], GameObject Unit\_Template[MAX\_UNIT\_VARIATION]의 크기를 초기화 해줍니다.

**Private void Update()**

* + 매 Frame마다 호출되는 기본 함수
  + 매초마다 에너지를 1씩 추가합니다. 에너지가 최대치만큼 있으면 추가하지 않습니다.
    - Update가 호출되고 다음 Update가 호출될 때까지 걸린 시간에 1을 곱한 값만큼 에너지를 추가합니다. (결과적으로 초당 1씩 에너지가 추가됩니다).

**Private void Object\_Pooling()**

* + 게임을 시작할 때 단 한번 호출하며, 플레이어가 선택한 Unit을 미리 만들어 두고 비활성화 합니다.
  + 이때 각 Unit의 Cost와 Summon Time을 받아와서 저장합니다.

## Public Variable

**public int MAX\_UNIT\_VARIATION**; //플레이어가 가질 수 있는 유닛의 총 개수를 의미합니다, Inspector에서 값을 조정합니다. 기본값은 3입니다.

## Private Variable

**Bool Is\_Casting**: 주문을 시전하고 있는지 여부를 나타냅니다. Default값은 False입니다.

**Bool Is\_Paused**: 현재 게임이 일시정지 되었는지를 나타냅니다. Default값은 False입니다.

**Float Health:** HQ의 현재 체력을 나타냅니다.

**float Resource**: HQ가 현재 소유한 자원의 수를 나타냅니다.

**Int Unit\_Count[MAX\_UNIT\_VARIATION**]: 종류별 현재 Unit의 수를 나타냅니다. Summon\_Unit() 함수를 호출할 때 마다 각 종류에 해당하는 값이 1씩 증가하며, Removed() 함수가 호출될 때 마다 각 Unit 종류에 해당하는 값이 1씩 감소합니다.

**Float Unit\_Cost[MAX\_UNIT\_VARIATION], Unit\_Summon\_Time[MAX\_UNIT\_VARIATION**]: 플레이어가 선택한 Unit의 Cost와 Summon\_Time을 저장합니다. Object Pooling 과정에서 저장합니다.

**GameObject Unit\_Template[MAX\_UNIT\_VARIATION**]: 플레이어가 선택한 Unit의 Prefab을 저장합니다, Object Pooling 과정에서 이 Prefab을 활용하여 진행합니다.

**Int Unit\_Queue[5**]: 소환 대기열에 들어있는 Unit의 번호를 나타냅니다.

# Unit.cs

* 모든 유닛의 AI 역할을 하는 코드입니다.
* 이동/공격/피격을 관장합니다.
* 각 Unit별로 고유의 공격Collider/피격Collider가 있으며, 공격Collider와 충돌한 Unit 중 특정 Unit의 Hit 함수를 호출합니다. Collider는 Unity 편집기에서 직접 조율합니다.

## Public Function:

**Public void Init(int \_Team, int \_ID, GameObject HeadQuarter)**

* + Unit이 생성된 직후 HeadQuarter에 의해 호출되며, Unit이 속한 팀의 번호를 할당합니다.
  + Input Parameter
    - Int Team: Unit이 소속된 팀의 번호입니다. 피아식별 할때 사용됩니다.
    - GameObject HeadQuarter: Unit을 소환한 HeadQuarter GameObject를 나타냅니다. Unit이 죽을 때 사용됩니다.
    - Int \_ID: 이 Unit이 플레이어의 몇번째 슬롯에 할당되어 있는지를 나타냅니다. 유닛이 죽을 때 사용됩니다.
  + Output Parameter: None

**Public void Hit(float damage)**

* + 특정 Collider와 충돌했을 때 그 Collider가 붙어있는 Object의 Script가 호출합니다, 호출되면 피격되었다고 간주하고 체력을 damage만큼 경감합니다.
  + Input Parameter
    - Float damage: 얼마의 데미지를 입었는가를 나타냅니다, 체력이 이 수치만큼 감소합니다.
  + Output Parameter: None

**Public Float Get\_Cost()**

* + 이 Unit을 소환하는데 필요한 자원의 양을 Return 합니다.
  + HeadQuarter.cs에서 Object Pooling을 진행할 때 한번씩 호출되며, 게임이 시작되고 나서는 호출되지 않습니다.
  + Input Parameter: None
  + Output Parameter: Float Cost

**Public Float Get\_SummonTime()**

* + 이 Unit을 소환하는데 걸리는 시간을 Return 합니다.
  + HeadQuarter.cs에서 Object Pooling을 진행할 때 한번씩 호출되며, 게임이 시작되고 나서는 호출되지 않습니다.
  + Input Parameter: None
  + Output Parameter: Float Summon\_Time

**public void Add\_Enemy(Transform Enemy)**

* + Unit Object의 Child로 지정된 공격 범위를 나타내는 Collider2D가 다른 Unit과 충돌했을 때 호출됩니다. (HitBox.cs의 OnTriggerEnter2D에서 호출합니다)
  + 범위 안에 들어온 적을 저장하는 Enemies List에 Enemy를 저장합니다.
  + Input Parameter: Transform Enemy: 공격범위에 들어온 Unit의 Transform
  + Output Parameter: Float Summon\_Time

**public void Remove\_Enemy(Transform Enemy)**

* + Unit Object의 Child로 지정된 공격 범위를 나타내는 Collider2D가 다른 Unit과 이 Collider를 빠져나갈 때 호출됩니다. (HitBox.cs의 OnTriggerExit2D에서 호출합니다)
  + 범위 안에 들어온 적을 저장하는 Enemies List에서 Enemy를 제거합니다.
  + Input Parameter: Transform Enemy: 공격범위에 들어온 Unit의 Transform
  + Output Parameter: Float Summon\_Time

## Private Function:

**Private void Start()**

* + 오브젝트가 생성이 완료된 후 단 한번 호출되는 함수. 편집기에서 초기화되지 않은 변수를 초기화합니다.
  + 각 Unit에 할당된 Sprite

**Private void Update()**

* + 매 Frame마다 호출되는 기본 함수
  + 현재 체력을 확인하고, 체력이 0보다 낮으면 Is\_Moveable을 False로 변경한다.
  + 공격 가능한 Object가 사정거리 내에 있는지 확인한 후, 없으면 Is\_Moveable을 True로 변경한 뒤 Move()를 호출합니다. (있으면 False로 변경합니다)

**Private void Move()**

* + 호출될 때 마다 정해진 양식의 움직임을 나타내는 함수.
  + Private void Update에서 호출되며, Is\_Moveable이 True일때만 Unit Object를 이동시킵니다.

**Private void Check\_Dead()**

* + Update()에서 호출하며, 체력이 0 이하로 떨어졌는지 확인합니다. 0 이하로 떨어졌으면 GameObject HQ의 HeadQuarter.cs Component의 Unit\_Dead(ID)를 호출합니다.
  + 유닛이 죽은게 확인되면 Coroutine을 활용하여 사망 애니메이션을 재생합니다.

**IEnumerator Dead\_Anim()**

* + Check\_Dead()에서 Coroutine으로써 호출하며, 사망 애니메이션 재생 후 유닛을 비활성화 합니다.

**Private void Get\_RGBValue()**

* + GameManager.cs에서 RGB값을 받아옵니다. Update()가 호출합니다.

**Private void Attack()**

* + 매 공격 주기마다 호출합니다(Update()에서 Last\_AttackTime과 현재 시간을 비교해서 공격 주기보다 더 크면 호출합니다)
  + 공격 범위 내에 있는 Unit Object(Enemies List에 저장됨)를 검사해서 공격 대상의 Hit(ATK) 함수를 호출합니다

## Public Variable

* float Max\_Health, Health: 최대 체력수치/현재 체력수치를 나타냅니다, Unity 편집기에서 각 Unit별로 고유의 값을 지정합니다.
* Float Cost: 이 Unit을 소환하는데 필요한 자원(에너지)의 양입니다. Unity 편집기에서 각 Unit별로 별도의 값을 지정한 뒤 Prefab화 합니다.
* Float Speed: 각 Unit의 고유 속도입니다, 단위는 Unity Meter입니다. Unity 편집기에서 각 Unit별로 별도의 값을 지정한 뒤 Prefab화 합니다.
* Float Summon\_Time: 각 Unit의 고유 소환시간입니다. 단위는 초입니다. Unity 편집기에서 각 Unit별로 별도의 값을 지정한 뒤 Prefab화 합니다.
* Float Attack\_Speed: 각 Unit이 공격을 1회 하는데 걸리는 시간을 의미합니다. Unity 편집기에서 각 Unit별로 별도의 값을 지정한 뒤 Prefab화 합니다.
* Int team: 소속된 Team의 번호를 나타냅니다.
* GameManager GM\_Script: GameManager 오브젝트에 붙어있는 GameManager.cs를 저장합니다.

## Private Variable

* ~~Int MAX\_R, MAX\_G, MAX\_B: 각 환경요소의 최대값을 나타냅니다. GameManager에서 받아옵니다.~~
* ~~Int MIN\_R, MIN\_G, MIN\_B: 각 환경요소의 최소값을 나타냅니다. GameManager에서 받아옵니다.~~
* Bool Is\_Moveable: 현재 Unit.cs가 붙어 있는 Object가 움직일 수 있는지를 나타냅니다. 기본값은 True이며, 앞에 공격할 수 있는 다른 Object가 있거나 health가 0보다 작아지면 False로 변경됩니다.
* GameObject HQ: Unit을 소환한 HeadQuarter GameObject를 저장합니다.
* Int ID: 이 Unit의 번호를 나타냅니다. (할당된 슬롯 순서)
* Vector3 Cur\_Env: 각 환경요소의 현재값을 나타냅니다, GameManager에서 주기적으로 받아오는 환경변수를 저장하는 임시 변수입니다.
* float ATK, DEF; //공격력과 방어력을 나타냅니다, Inspector에서 편집합니다.
* List<Transform> Enemies = new List<Transform>(); 히트박스에 들어온 적들을 나타냅니다.
* private float Last\_AttackTime; //마지막으로 공격한 시간을 저장합니다

# UI\_Enviroment.cs

* UI 요소 중 환경 수치를 나타내기 위한 스크립트입니다. RGB 속성에 따라 UI의 모양을 바꾸며, UI의 모양이 **확정되면** 관련 함수를 작성합니다.
* GameManager.cs의 Get\_RGBValue() 함수를 통해 RGB값을 받아와서 UI에 반영합니다.

## Public Function

## Private Function:

### **Private void Start()**

* + 오브젝트가 생성이 완료된 후 단 한번 호출되는 함수
  + GameManager의 public int MAX\_R, MAX\_G, MAX\_B, MIN\_R, MIN\_G, MIN\_B를 받아옵니다.

### **Private void Update()**

* + 매 Frame마다 호출되는 기본 함수

## Public Variable

## Private Variable

* Int MAX\_R, MAX\_G, MAX\_B: 각 환경요소의 최대값을 나타냅니다. GameManager에서 받아옵니다.
* Int MIN\_R, MIN\_G, MIN\_B: 각 환경요소의 최소값을 나타냅니다. GameManager에서 받아옵니다.

# SpellManager.cs

* 각 HQ에서 시전되는 주문을 관리하는 서브 매니저 역할입니다.
* 플레이어가 주문을 사용하고자 할 때 SpellManager.cs에서 플레이어의 입력을 받고 주문을 시전합니다.

## Public Function:

### **Public void Spell\_Load(bool Is\_AttackSpell, int Spell\_ID)**

* + GameManager에 의해 호출되며, 호출되면 ID 번호에 맞는 Spell을 사용할 준비를 합니다. 그와 동시에 HeadQuarter.cs의 Toggle\_Is\_Casting()을 호출하여 크리처 소환을 막습니다.
  + Is\_AttackSpell의 값이 True이면 Attack\_Spell\_ID의 값을 변경허고, False이면 Spell\_ID의 값을 변경합니다.
  + Input Parameter:
    - Int ID: 시전할 주문의 미리 지정된 ID를 의미합니다. (Unit 소환과 달리 할당된 슬롯의 번호가 아니라 전체 주문에 할당된 고유 ID를 나타냅니다)
    - Bool Is\_AttackSpell: 시전할 주문이 공격주문인지를 나타낸다.

## Private Function:

### **Private void Start()**

* + 오브젝트가 생성이 완료된 후 단 한번 호출되는 함수

### **Private void Update()**

* + 매 Frame마다 호출되는 기본 함수
    - Spell\_ID의 값을 체크하여 패시브 주문이 대기중인지 확인합니다.
      * 일반 주문이 대기중이면 즉시 **발동합니다**. 그리고 Spell\_UnLoad()를 호출합니다.
    - Attack\_Spell\_ID의 값을 체크하여 공격 주문이 대기중인지 확인합니다.
      * Active 주문이 대기 중이면 Player 1은 A, D를, Player 2는 J, L를 눌렀을 때(누르는 중에) 주문의 위치를 좌우로 이동합니다.
      * Player 1와 Player 2는 각각 S, K를 눌렀을 때 주문을 발동합니다.
        + 정확한 발동 메커니즘은 현재 구상중입니다.
      * 주문 발동이 끝나면 같은 Script의 Spell\_UnLoad()를 호출합니다.
  + Input / Output Parameter: None

### **Private void Spell\_UnLoad()**

* + Update()에서 주문 시전이 완료되면 호출되며, HeadQuarter.cs의 Toggle\_Is\_Casting()을 호출하여 크리처 소환을 다시 허용합니다.
  + 주문 시전이 완료된 후 취할 액션을 여기에 넣을 예정이며, 기획서가 완성되었을 때 기능이 매우 적다면 이 함수를 Update 함수와 병합할 예정입니다.
  + Input / Output Parameter: None

## Public Variable

## Private Variable

* Bool Is\_Toggled: 주문 사용 준비 중인지를 나타냅니다. Default는 False입니다.
* Int Spell\_ID, Attack\_Spell\_ID: 현재 사용 대기중인 주문 / 공격주문의 식별번호를 저장합니다. Default는 0입니다.

# TimeManager.cs

* Unit을 비롯하여 시스템 일시정지에 영향을 받는 모든 오브젝트의 시간을 관리합니다.
* 가속/감속/일시정지/Unit 고유 FrameRate 설정을 지원합니다. (게임의 본래 FrameRate보다 더 높게 설정할 수는 없습니다)

## Public Function:

### **Public void Spell\_Load(bool Is\_AttackSpell, int Spell\_ID)**

* + GameManager에 의해 호출되며, 호출되면 ID 번호에 맞는 Spell을 사용할 준비를 합니다. 그와 동시에 HeadQuarter.cs의 Toggle\_Is\_Casting()을 호출하여 크리처 소환을 막습니다.
  + Is\_AttackSpell의 값이 True이면 Attack\_Spell\_ID의 값을 변경허고, False이면 Spell\_ID의 값을 변경합니다.
  + Input Parameter:
    - Int ID: 시전할 주문의 미리 지정된 ID를 의미합니다. (Unit 소환과 달리 할당된 슬롯의 번호가 아니라 전체 주문에 할당된 고유 ID를 나타냅니다)
    - Bool Is\_AttackSpell: 시전할 주문이 공격주문인지를 나타낸다.

## Protected Variable

* protected float FrameRate;//Unit의 Update()가 1초에 몇번 호출되는지를 나타냅니다. Inspector에서 값을 지정합니다.
* float LastUpdateTime; //마지막 업데이트가 이루어진 시간을 저장합니다.
* float Unit\_TimeScale;//Unit 고유의 TimeScale을 나타냅니다, 일시정지 및 감속 효과에 사용됩니다.