과제 구현 설명서

김중현

과제내용 : 이동하는 타워 앞에 쌓이는 몬스터 구현

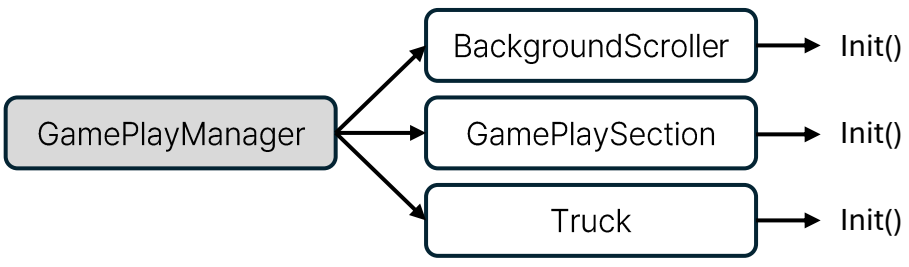
구현내용

1. **게임 플로우**
2. **트럭 이동**
    a. 스크롤하는 배경과 오브젝트
    b. 몬스터에게 밀리면 서서히 정지
    c. 몬스터 생성 지점과 근접시
3. **적 (몬스터)**
    a. 베이스 스크립트
    b. 생성 후 이동
    c. 공격과 플레이어 피해
    d. 쌓이며 순환하는 구조 형성
4. **플레이어 무기**
    a. 적 탐지 및 총알 발사
    b. 적 피해

구현내용 상세

1. 게임플로우

게임을 실행하면 우선적으로 GamePlayManager에서 진행에 필요한 오브젝트들을 초기화합니다.
진행에 필요한 오브젝트들은, 현재 스테이지 (배경), 현재 구역(스테이지 내의 단계), 플레이어 정보가 있습니다.
초기화를 하면 자동으로, 적이 생성되며 트럭이 움직이게 됩니다.
현재는 하나의 구역만 만들어진 상황입니다.



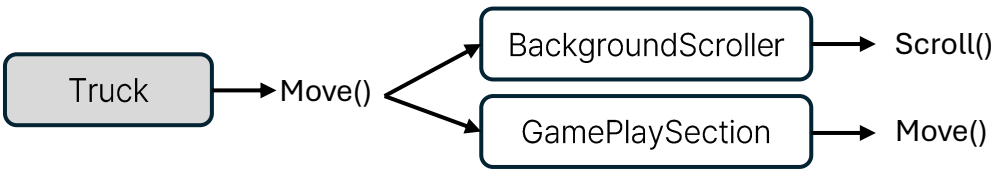

```
void StartGame()
{
    Debug.Log("게임 시작");
    isPlaying = true;

    // 현재 단계 구역 생성. - 지금은 강제로 주입
    GamePlaySection section = FindObjectOfType<GamePlaySection>();
    section.Init();
    playingSection = section;


    List<BackgroundScroller> backgroundScrollers = FindObjectsOfType<BackgroundScroller>().ToList();
    foreach( BackgroundScroller bg in backgroundScrollers)
    {
        bg.Init();
    }

    //  트럭 초기화 - 하위 박스랑 히어로도 초기화됨.
    FindObjectOfType<Truck>().Init(playingSection,backgroundScrollers);
}
```

GamePlayManager.cs의 게임 초기화 및 시작 로직

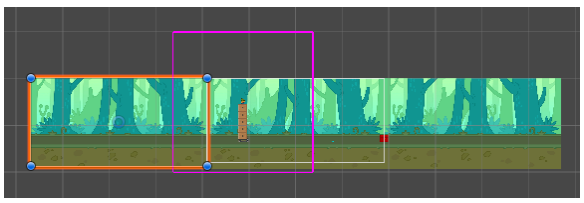
2. 트럭 이동

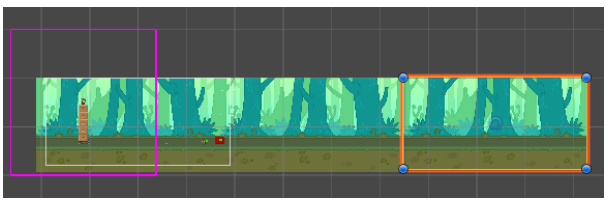
a. 스크롤하는 배경과 오브젝트

트럭이 이동해야 할 때 배경과 구역 자체가 계속 좌측으로 움직입니다.
더 작고 적은 수의 배경 이미지를 사용할 수 있기 때문입니다.
트럭은 FixedUpdate에서 움직일 거리 targetSpeed를 계산하고 위 오브젝트들을 움직입니다.



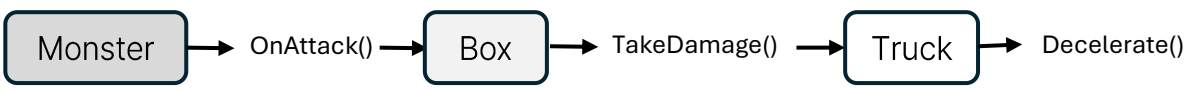
BackgroundScroller는 좌측으로 이동하다가, 임계 지점을 지나면 좌측의 이미지가 우측 끝으로 이동하며 무한 반복되는 배경을 연출하게 됩니다.
임계지점은 지정된 이미지의 크기를 통해 자동으로 계산합니다.





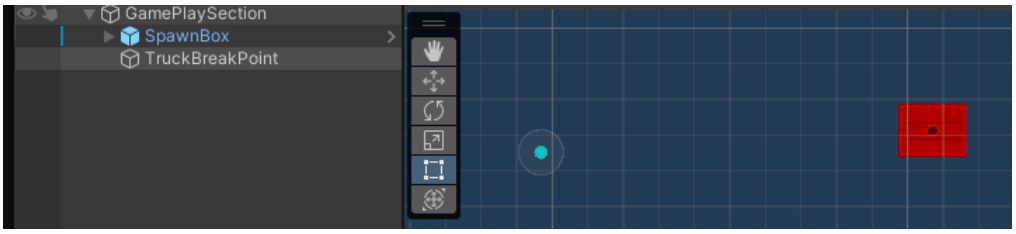
b. 적에게 밀리면 서서히 정지

박스가 적의 공격을 받으면 이동속도가 일정 비율 만큼 감소합니다.
적의 공격을 받고 일정 시간이 지나면 서서히 이동속도를 회복하여 최대속도까지 증가합니다.



c. 몬스터 생성 지점과 근접시

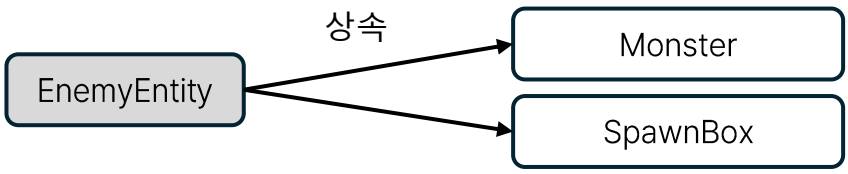
몬스터 생성 지점과 근접했을 때, (브레이크 포인트에 도달했을 때,) 트럭은 정지합니다.
그 후, 생성 지점을 파괴하여 해당 구역을 클리어하면 트럭이 다시 이동하게 됩니다.
아래 이미지의 Spawn Box 가 생성지점, TruckBreakPoint가 트럭 정지 지점입니다.



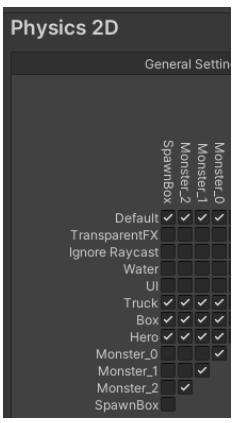
3. 적 (몬스터)

a. 베이스 스크립트

적 개체 (Monster, SpawnBox) 는 EnemyEntity를 상속받습니다.
EnemyEntity는 Enemy 태그를 사용하여, 플레이어가 타겟 탐색에 감지됩니다.
EnemyEntity는 체력과 피해적용 함수가 있어 체력이 0이 되면 파괴됩니다.

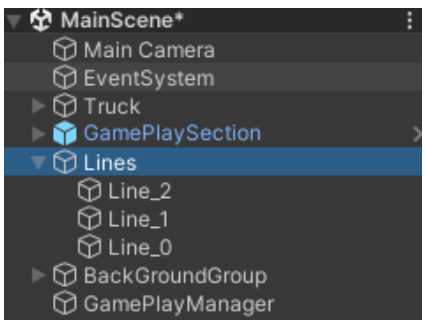


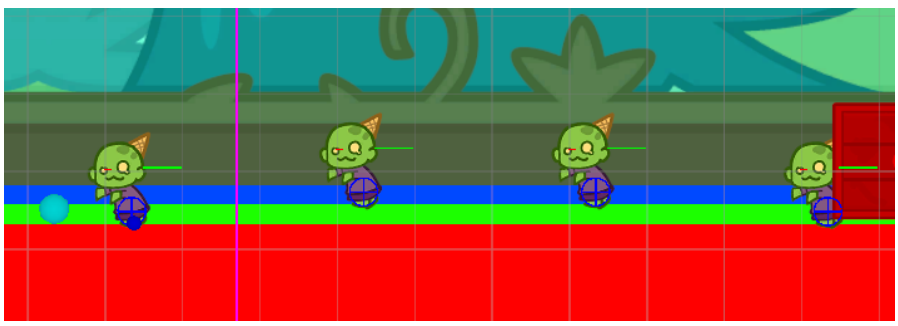
Monster는 애니메이션, 인공지능, 움직임, 상태머신 기능이 있어 플레이어를 추적하며 공격할 수 있습니다.
이때, Monster_0, Monster_1, Monster_2 중 하나의 레이어를 지정 받아, 같은 레이어의 몬스터 개체들끼리만 충돌하도록 하였습니다.



b. 생성 후 이동

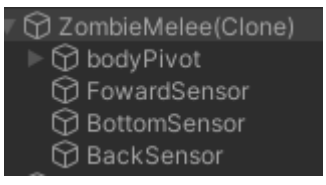
SpawnBox 는 GamePlayManager가 GamePlaySection을 초기화할 때 초기화됩니다.
그 후, spawnInterval 마다 하나의 몬스터를 3개의 공격로 중 하나에 생성합니다.
동시에 몬스터는 해당 공격로와 일치하는 레이어를 지정 받습니다.
몬스터는 계속 트럭방향(좌측)으로 이동합니다. '
트럭은 밀리지 않기 때문에, 몬스터는 트럭이 파괴되기 전까지 트럭 좌측으로 이동할 수 없습니다.

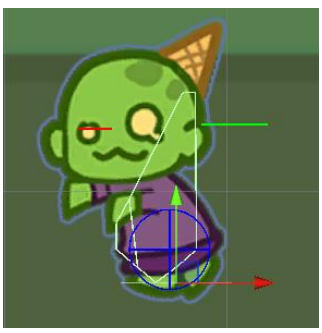




c. 공격과 플레이어 피해

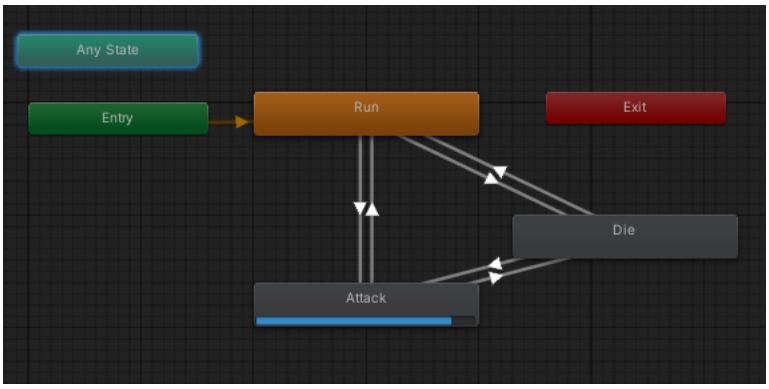
몬스터는 상태를 결정하는 인공지능이 있습니다. (MonsterAI.cs)
이 인공지능은 일정 시간(aiUpdateTime) 마다
전면센서, 후방센서, 바닥센서의 상황에 맞춰 다음 행동을 계산합니다.



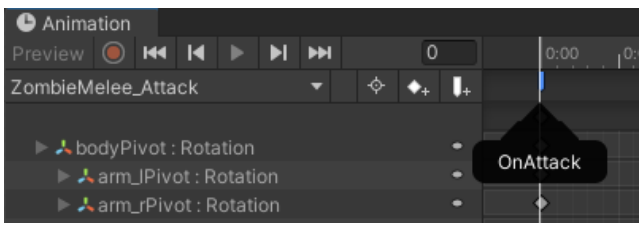


전면센서에 플레이어 레이어가 감지되면 곧바로 attack state 로 전환되며,
공격 애니메이션을 실행하게됩니다.

```
// 몬스터 앞에 있는 장애물 검사
1 reference
MonsterPositionState GetNewPositionState()
{
    //
    var hit = Physics2D.Raycast(t_forwardSensor.position - new Vector3(float.Epsilon,0),
     Vector2.left, _forwardSensorRange, 1<<GameConstants.truckLayer| 1<<layerNum);

    MonsterPositionState ret = MonsterPositionState.Default;
    if( hit.collider == null)
    {
        return ret;
    }

    //
    int hitLayer = hit.collider.gameObject.layer;
    if( hitLayer  == GameConstants.truckLayer)
    {
        ret = MonsterPositionState.PlayerInAttackRange;
    }
    else if (hitLayer  == layerNum)
    {
        ret = MonsterPositionState.BlockedByMonster;
    }

    return ret;
}
```

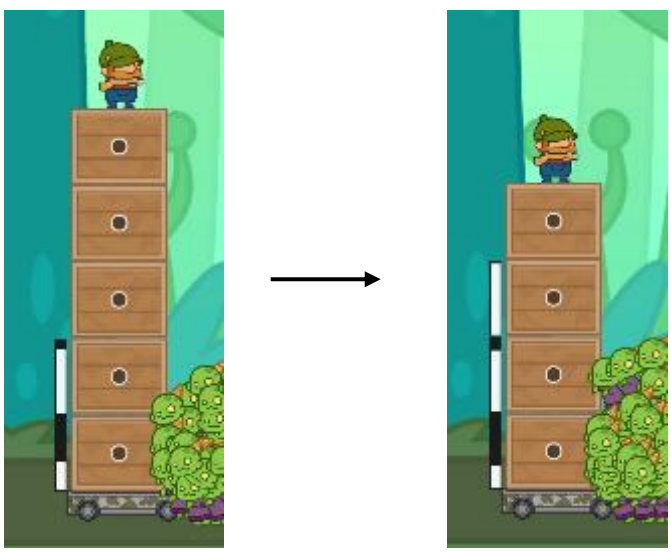


재생되는 애니메이션에는 AnimationEvent에 할당되어 있어,
Monster.OnAttack(AnimationEvent animationEvent)함수를 실행합니다.
이때 현재 감지된 플레이어의 오브젝트의 피해적용 함수를 실행합니다.
단, 현재 Hero 는 피해를 입지 않습니다.



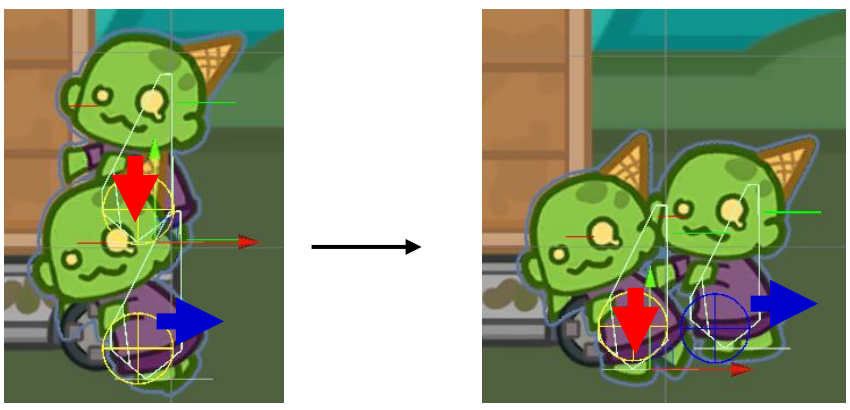

```
// 공격적용.
0 references
void OnAttack(AnimationEvent animationEvent)
{
    // Debug.Log("공격");
    Collider2D collider = mAi.GetColliderInAttackRange();
    if( collider!=null &&collider.TryGetComponent(out Box box))
    {
        box.TakeDamage(attackPower);
    }
}
```

Box는 피해를 입으면 현재 체력이 감소하며, 박스 좌측에 hpSlider가 표시됩니다.
해당 슬라이드는 2초가 지나면 다시 비활성화됩니다.
또한, Box는 현재 체력이 0이 되면 파괴되어, 그 위의 Box와 Hero가 아래로 내려오게 됩니다.



d. 쌓이며 순환하는 구조 형성

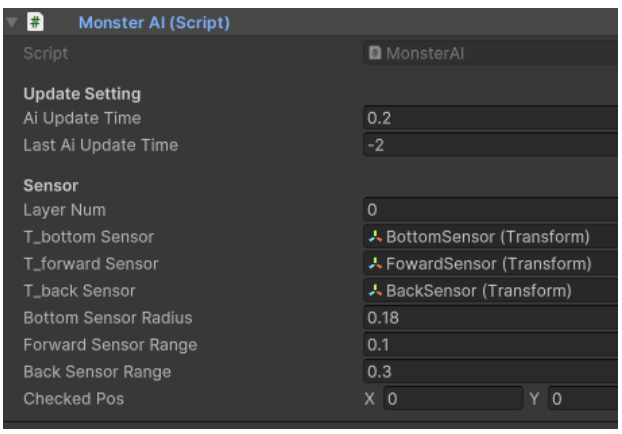
타워에 의해 몬스터의 진행방향이 막히게 되면, 자연스럽게 일렬로 정렬됩니다.
이때, 전면 센서를 이용하여, 앞에 몬스터가 있는 경우에는 점프를 하여 앞의 몬스터를 뛰어넘습니다.
그리고 타워 앞으로 이동하면 중력과 아래 몬스터의 콜라이더 형태에 의해 자연스럽게 아래로 내려오며
순환하는 구조가 만들어 집니다.



이때, 후방센서에 같은 공격로의 몬스터가 감지되지 않아야 점프를 할 수 있습니다.
그리고 공중에 있는 상황(바닥센서에 아무 것도 감지되지 않는 상황)에는 중력 값을 0으로 설정하여,
앞으로 이동 중에 몬스터 사이에 빠지지 않도록 처리하였습니다.

이로써 맨 뒤의 몬스터가 점프하여 맨 앞으로 이동하며, 아래로 떨어지고,
위에서 떨어지는 다른 몬스터에 의해 우측으로 밀리는 순환구조를 구현하였습니다 .

Monster prefab의 계층구조에서 각 센서의 위치를 조정하고,
인스펙터에서 몬스터 콜라이더의 형태와 jumpPower, 각 센서 감지 범위, 인공지능 갱신 시간 등을
조절해서 이 순환 구조의 디테일을 수정할 수 있습니다.



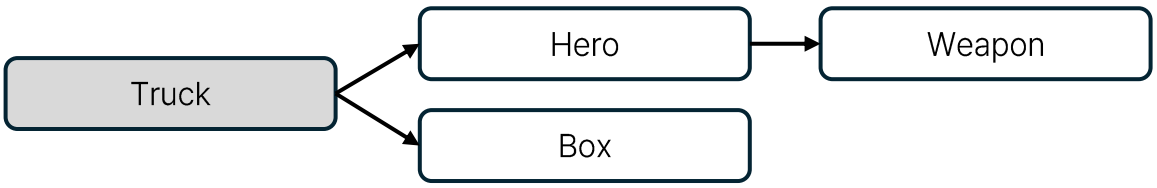
TDS

본 과제

4. 플레이어 무기

a. 적 탐지 및 탄환 발사

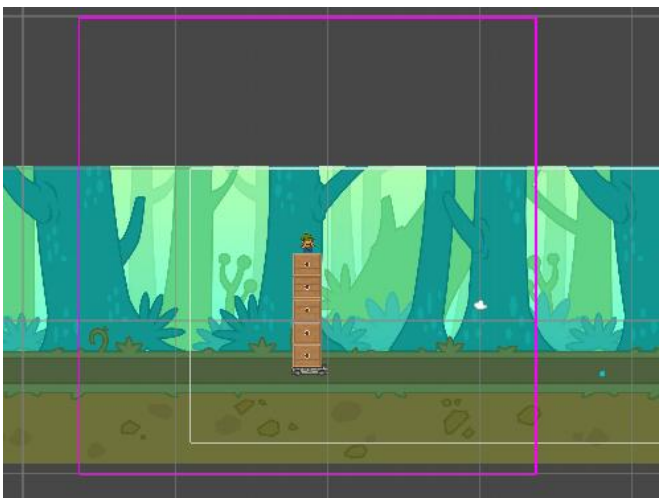
게임 시작 시, Truck -> Hero -> Weapon 연쇄적으로 초기화가 이뤄집니다.
Hero가 살아있는 동안, 이 무기(Weapon)의 TryUse 함수로 적을 향해 탄환을 발사합니다.



무기는 사용간격 (useCooltime) 마다 사용할 수 있으며,
사용 시 공격 대상 탐색 -> 회전 각도 설정 -> 탄환 발사를 진행하게 됩니다.

공격 대상 탐색시, Physics2D. OverlapBoxAll 로 _range 크기에 비례하는 직사각형의 영역내의 EnemyEntity를 탐지합니다.
그러면 해당 영역 내의 모든 Monster 와 SpawnBox가 탐지되며, 이 중에서 플레이어와 가장 가까운 개체를 공격 대상으로 지정합니다.

```
// 무기 사용
1 reference
void Use()
{
    // 타겟 탐색
    if(currTarget == null|| currTarget.isAlive ==false)
    {
        EnemyEntity monster = FindTarget();
        currTarget = monster;
    }


    // 무기 방향 지정
    CalRotation(currTarget);

    // 발사
    GenerateBullets();

    //
    lastUseTime = Time.time;

}
```



그리고 공격 대상의 방향을 회전 지점으로 설정합니다.
지정된 회전은 Update에서 진행됩니다.
이때, 마우스 왼쪽 버튼을 누르고 있으면, 목표 회전 방향이 커서 방향으로 고정됩니다.
추가로, SmoothDamp를 통해 부드럽게 회전하고, 최대로 회전 각도를 지정하였습니다.

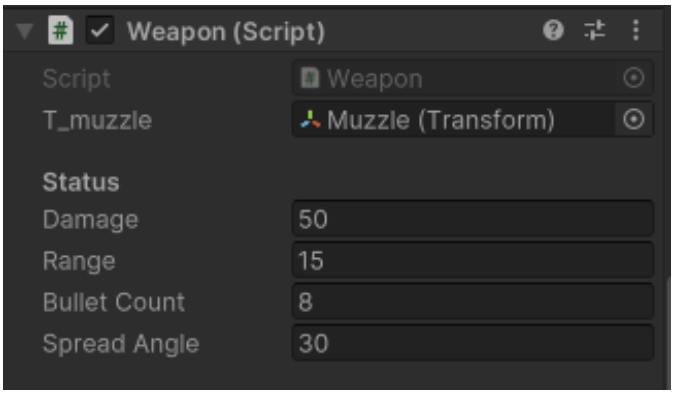
```
0 references
void Update()
{
    autoTarget = !Input.GetMouseButton(0);   // 마우스 클릭시 자동공격

    CalRotation(currTarget);
    Rotate();
}
```

```
// 자연스러운 회전
1 reference
void Rotate()
{
    currAngle = Mathf.SmoothDampAngle(currAngle, targetAngle, ref angularVelocity, smoothTime);
    transform.localRotation = Quaternion.Euler(0, 0, currAngle + defaultRotation);
}
```

자동조준

직접조준

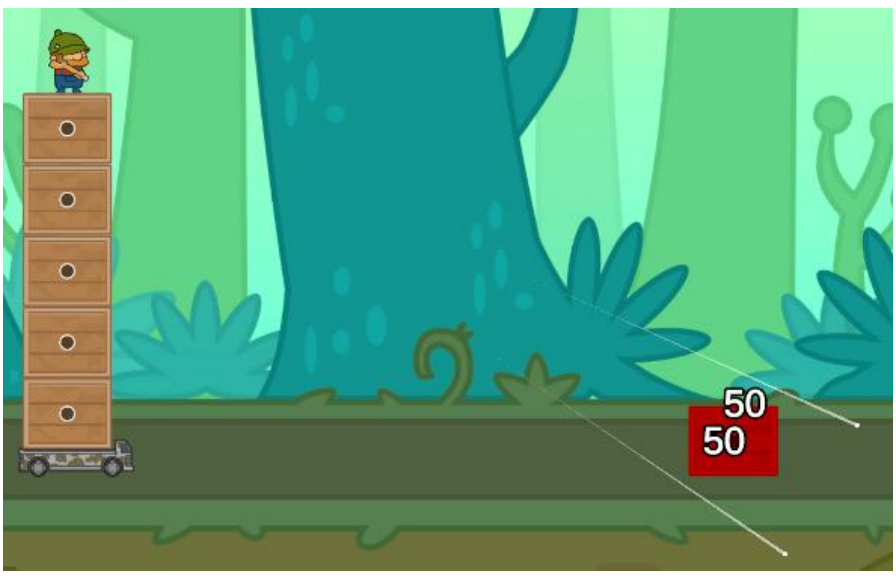
탄환은 회전한 방향으로 bulletCount 개수 만큼 발사합니다.
이때, spreadAngle로 탄환이 퍼지는 각도를 조절할 수 있습니다.



b. 적 피해

발사된 탄환은 OnTriggerEnter에서 EnemyEntity와의 충돌을 감지합니다.
TDS에서는 동일한 적에게 동시에 한 번의 피해만 입힐 수 있지만,
본 과제에서는 동일한 적에게 동시에 여러 피해를 입힐 수 있습니다.

적은 피해를 받은 경우, 피해량 텍스트를 생성합니다.
생성된 텍스트는 오른쪽으로 포물선을 그리며 투명해지면서 사라집니다.



감사합니다

게임 클라이언트 김중현