Doom-Like 2.5D FPS

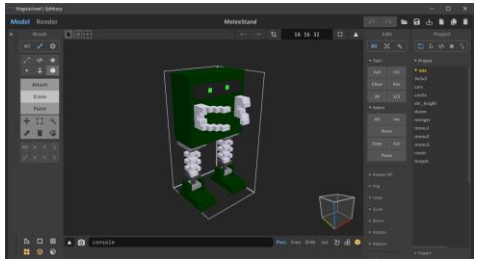
robmaz

201704044 백문하

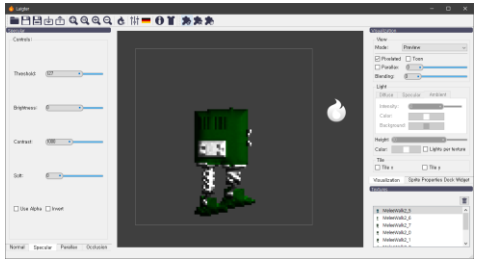
프로젝트 개요

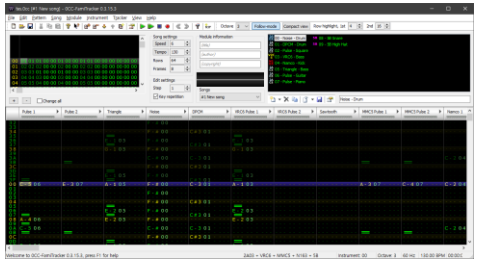
- 2.5D FPS
  - Wolfenstein 3-D, Doom, Duke Nukem 3D 등 초기 FPS에서 시도되던 방식.
  - 레트로한 분위기의 게임을 만들고자 하였음.
- **Robmaz** : Robot Maze, 로봇들을 무찔러야 하는 미로 게임.
- 사용 기술
  - 게임 엔진 : Unity
  - 효과음, 음악 제작 : Famitracker
  - 모델링 : MagicaVoxel
  - 스프라이트 편집 : Laigter, GIMP, 그림판

MagicaVoxel



Laigter



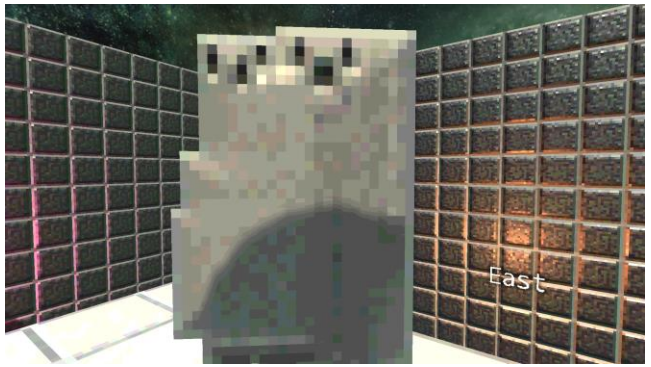


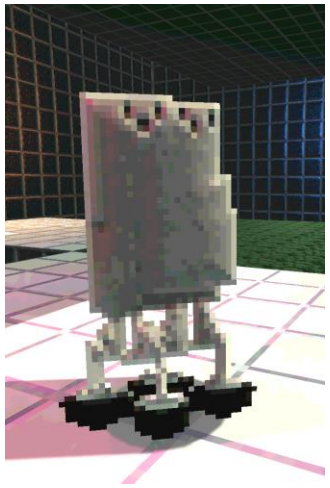
주요 기능

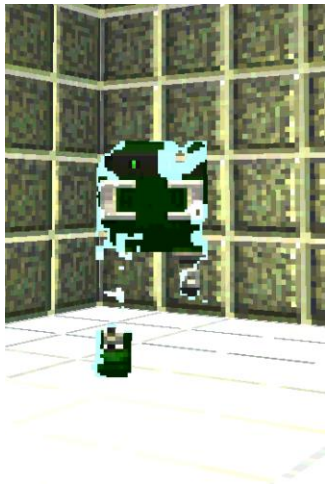
1. **2.5D 스프라이트 오브젝트**
   - 플레이어와의 방향에 따라 스프라이트 변경, 별도 메시와 렌더텍스처로 그림자 관련 기능 구현.
2. **셰이더 & 액체 효과**
   - 레트로 분위기로 색상 제한. 딱딱하게 끊어지는 텍스처 애니메이션.
   - 액체 속에서 카메라에 울렁거리는 필터 및 음향에 저역 필터 통과 적용.
3. **물리적 이동 효과 :** 부드러운 이동, 공기 저항, 경사로 대응.
4. **무기 구현 :** 히트스캔 및 투사체 무기 구현, 2중 레이캐스트로 정확한 조준 구현
5. **인공지능**
   - 거리와 시야에 따른 상태 기계로 작동.
   - 네비게이션 메시 기반.
   - 플레이어의 다음 위치 예측하여 투사체 발사 각도 결정
6. **맵 무작위 생성**
   - 미로 알고리즘으로 스테이지 순서 구성, 각 스테이지 난이도에 따라 무작위 생성
   - 동적 네비게이션 메시 및 링크 생성
7. **게임 시스템 :** 이벤트 시스템, 자막, 메뉴, 옵션과 키 매핑

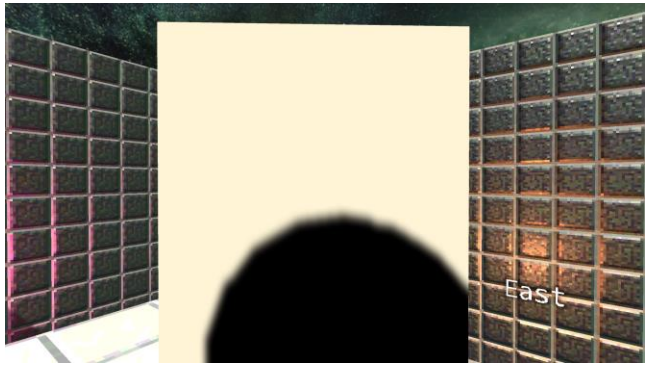
2.5D 스프라이트 구현

- 복셀 방식으로 제작한 모델을 8 방향으로 저해상도 추출,
  스프라이트 일괄 처리 툴을 통해 반사 맵과 노멀 맵 생성.
- 플레이어와의 방향에 따라 각 방향 별 스프라이트가 플레이어를 바라보도록 구현.
- 기존 셰이더를 기반으로 투명 컷아웃 방식으로 개조.
- 스프라이트는 납작하여 그림자가 불가능하므로,
  보이지 않는 실린더 메시를 추가하여 그림자를 드리우도록 제작.
- 드리워지는 그림자는 별도의 카메라와 렌더텍스처를 이용하여 구현.
- 적 제거 시 Particle System 이용한 폭발 Prefab 생성,
  노이즈 텍스처 활용한 소멸 애니메이션 구현.







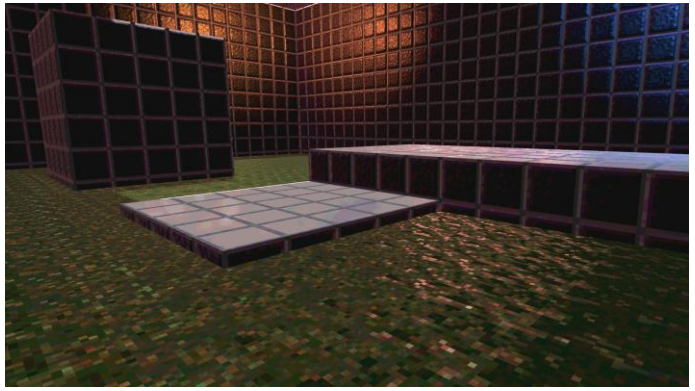


셰이더 & 액체 효과

- 기본적인 유니티 셰이더를 개조, 고전 게임의 제한된 색상을 구현.
  - 16으로 곱한 뒤 floor 연산 후 다시 16으로 나눠주면 채널 당 256단계의 색상이 $\frac{16}{256}$=16 단계가 됨.
- 액체의 경우 노멀 맵과 노이즈를 이용하여 흘러가는 애니메이션.
  - 부드럽게 움직이는 것이 아니라 픽셀의 크기에 맞춰 딱딱하게 움직이도록 구현.
    시간에 floor 연산 후 다시 픽셀 수로 나눠주면 픽셀 수에 맞춰 움직임. 시간 자체에 계수를 곱해 속도 조절 가능.

```
fixed time = floor(_Time.x * 64) / 16.0f;
```

- 카메라가 액체 속에 있다면, 셰이더를 통해 울렁거리는 효과 구현,
  AudioMixer를 이용, 모든 효과음에 Low Pass Filter 적용.
- 카메라가 액체 속으로 들어가거나 나올 때, 각 객체들이 액체 속으로 빠질 때 효과음 재생.



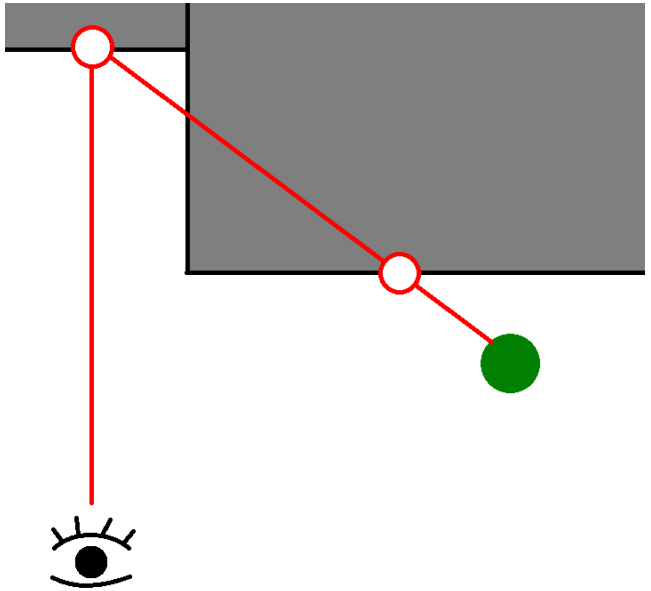
물리적 이동 효과

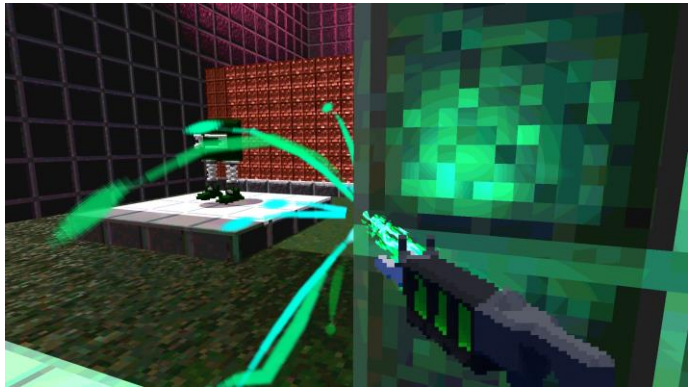
- 이동을 부드럽게 하고, 충돌에 대응하기 위해 이동과 점프에 Rigidbody.addForce를 이용.
- 지면 체크 및 수중 체크는 플레이어의 발을 기준으로 Physics.CheckSphere를 통해,
  해당하는 마스크의 오브젝트가 닿았는지를 인식.
- 지면일 때와 공중일 때의 drag값을 다르게 하여 공중에선 저항을 덜 받도록 구현.
  - 추가로 액체 속에 있다면 drag를 높여 액체의 저항을 표현.
- Physical Material을 이용했다면 벽에 붙었을 때 천천히 미끄러지기 때문에, Friction을 0으로 설정.
- Update 대신 FixedUpdate에서 수행.
  - Update는 불규칙적이어서 물리 검사에 방해가 되고, 과도하게 호출될 수도 있음.
    반면 FixedUpdate는 일정하게 적은 필요한 횟수만 호출됨.
- 경사로에 있을 경우 무중력으로 설정.
  - Friction이 0이라 경사로 상에서 미끄러지기 때문.
  - 바닥에 레이캐스트를 실시했을 때, 표면의 법선이 위를 향하지 않았을 경우, 경사로에 있는 판정.

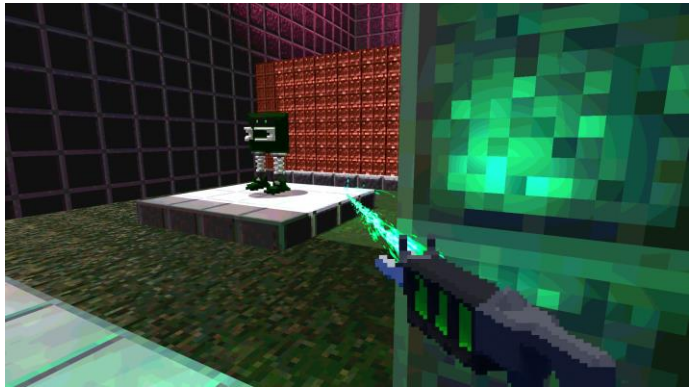
- 폭발로 인한 넉백은 Rigidbody.addExplosionForce로, 거리에 따른 힘을 부여.
- 플레이어 외에도 적이나 NPC, 가스통 등에도 동일한 물리 법칙을 적용.

무기 구현

- 무기 스프라이트는 1인칭 시점으로 미리 만들어 두어 사용.
  무기 교체 시 스프라이트가 아래로 내려갔다가 다시 올라오는 애니메이션 구현.
- 두 번의 RayCast를 통해 정확한 피격지점을 찾도록 구현.
  - 카메라가 가리키는 조준점은 총구 기준으로는 가려질 수 있기 때문.
- 레이저건은 LineRenderer로, 로켓 런처는 Prefab 발사하는 방식으로 구현.
- 가스통과 로켓의 폭발은 거리에 비례하여 넉백과 피해를 부여.

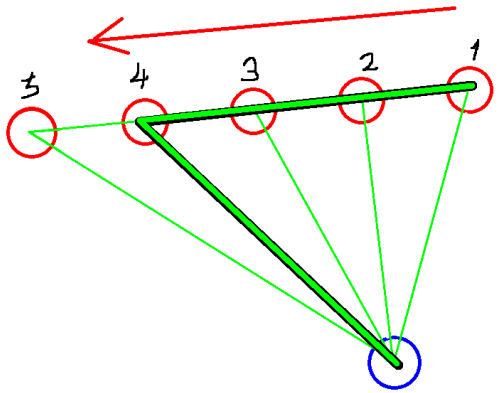






인공지능

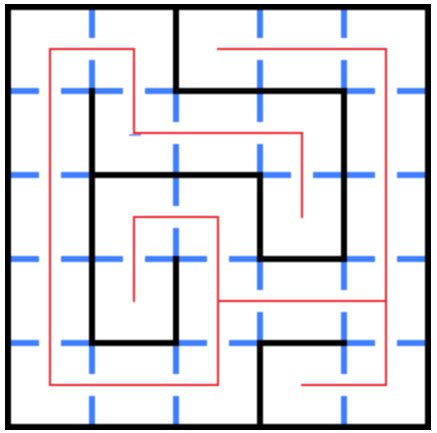
- 네비게이션 메시 기반으로 작동하지만,
  Rigidbody와 NavMeshAgent는 동시 사용이 불가능한 문제 발생
  - 물리 효과를 받는 동안에는 네비게이션 메시 기능 해제하여 해결.
- 적은 플레이어와의 거리 및 시야의 여부에 따라 탐색, 이동, 공격 세 가지의 상태기계로 작동.
- 플레이어와의 거리가 가까우면 플레이어를 추적 이동.
  플레이어가 공격 사정거리 이내에 있고 적의 시야 안에 있다면 공격 상태로 전환.
- 로켓 발사형 적은 플레이어가 등속 직선 운동을 한다고
  가정했을 때의 위치를 예상하고,
  그 부근을 무작위로 발사하도록 구현.

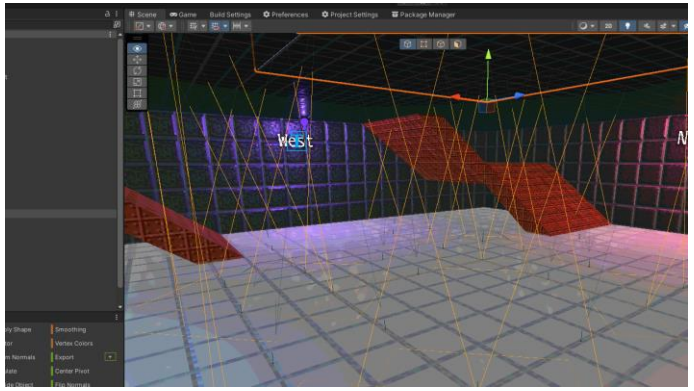



```
public Vector3 next_position(Vector3 launcher_pos, float projectile_speed) {
    Vector3 next_pos = transform.position;
    float distance = Vector3.Distance(next_pos, launcher_pos);
    float time = 0.0f;
    float collision_time = distance / projectile_speed;
    int count = 0;
    while ((time < collision_time) && count < 100) {
        next_pos += rigid.velocity * Time.fixedDeltaTime;
        distance = Vector3.Distance(next_pos, launcher_pos);
        time += Time.fixedDeltaTime;
        collision_time = distance / projectile_speed;
        count++;
    }
    next_pos.y = transform.position.y;
    next_pos = Vector3.Lerp(next_pos, transform.position, Random.Range(0.0f,
0.5f));
    return next_pos;
}
```

맵 무작위 생성

- **본 게임은 미로로 구성하며, 미로 생성은 Recursive Backtracking 알고리즘을 이용.**
  - 2^n으로 Enum을 구성하면 비트연산을 이용해서 플래그처럼 사용 가능.
    각 Enum을 동서남북으로 취급하고 비트 AND연산을 통해 상태를 겹칠 수 있음.
  - 정수의 2차원 배열에서 미로 생성 알고리즘을 적용.
- **미로를 생성하면 각 노드에 대해 각 요소를 랜덤으로 구성.**
  - 원본 맵에는 각각 9개의 지붕과 층, 바닥을 놓고, 각 구석에 4개의 계단을 배치한 뒤,
    미로 생성시에 모든 지형을 서로 이동 가능하게끔 무작위로 배치.
  - 미로의 연결된 방향에 맞게 각 방향에 출구 생성.
- **미로 플레이 시 난이도에 맞게 적과 아이템 무작위 생성.**
  - 난이도는 게임이 진행될수록 어려워지고, 적의 체력 및 공격력, 적과 아이템의 수를 결정.
  - 각 객체들은 9개로 나눠진 각 판의 무작위 위치에 놓임.
- **동적 네비게이션 메시 생성.**
  - NavMeshSurface 컴포넌트를 이용하면 동적 네비게이션 메시 생성이 가능.
  - 단, Off Mesh Links까지 자동 생성해주지는 않으므로,
    NavMeshLinks 컴포넌트를 이용.
    미리 만들어놓은 NavMeshLinks 요소들을 생성된 지형에 따라 토글하여 구현.





게임 시스템

- **UnityEvent 활용한 이벤트 시스템**
  - 객체 소멸, 아이템 획득, 상호작용, 자막 완료 시 등, 자막을 추가하거나 문을 여닫는 이벤트 구현.
- **자막 구현**
  - 자막 큐에 텍스트를 추가하면 한 글자씩 입력되는 애니메이션 구현.
  - <> 로 둘러싸인 Rich Text 요소는 효과를 바로 적용, 스킵 키를 누르면 바로 다음 자막으로 넘어갈 수 있음.
- **메뉴 구성**
  - 각 메뉴 상태를 Enum으로 만들어 GameManager에서 캔버스의 요소들을 토글하는 것으로 구현.
- **옵션과 키 매핑**
  - 각종 설정은 JSON으로 관리.
  - 유니티 엔진의 기본 InputManager는 런타임 중 키 변경이 불가하기 때문에, 자체 제작한 스크립트 사용.

```
public void laser_ammo_get_event() {
    gm.caption_addtext(
        "이것은 <color=#3fff7f>레이저</color>의 탄환입니다.",
        $"[{InputManager.get_button_key_names("weapon 1")}] 키를 눌러 <color=#3fff7f>레이저</color>를 선택할 수 있습니다.",
        $"[{InputManager.get_button_key_names("fire")}] 키를 눌러 발사할 수 있습니다."
    );
    gm.caption_addtext(
        $"<color=#5fff5f>함정</color>에 빠지면 지속적으로 체력이 감소합니다",
        $"[{InputManager.get_button_key_names("jump")}] 키를 눌러 점프하여 건너가세요."
    );
    gm.caption_addtext(
        $"레이저로 앞의 <color=#ff6f6f>적</color>을 죽이세요."
    );
}
```