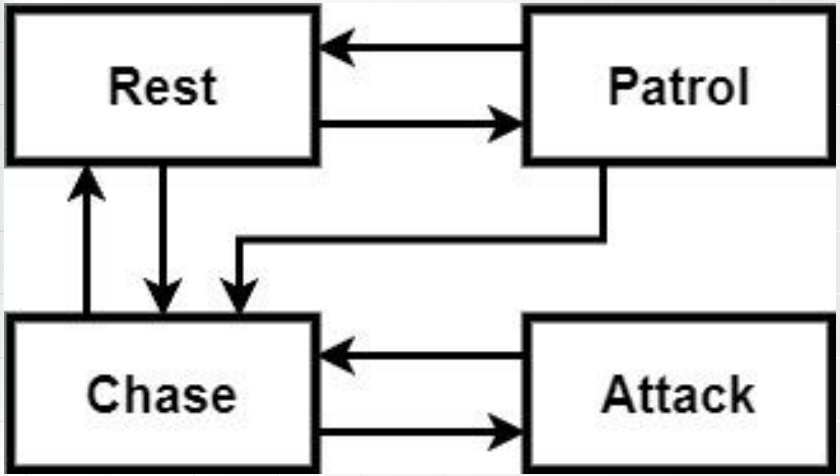
Sztuczna inteligencja - podstawy



Graficzna reprezentacja maszyny stanów dla MeleeSM

- **Maszyna stanów** - model obliczeń zakładający skończoną liczbę stanów i zawsze bycie w dokładnie jednym stanie.
- Możemy definiować akcje, które dzieją się przy:
  - Wejściu
  - Wyjściu
  - W stanie

Sztuczna inteligencja - podstawy

```
3 references
public bool IsGoalInRange()
{
    if(enemySM.GetGoalDynamic() == null)
    {
        return false;
    }
    return enemySM.IsInRangeXZ(detectionRange);
}
```

```
if (mainStateMachine.IsGoalInRange())
{
    mainStateMachine.ChangeState(mainStateMachine.chase);
    return;
}
```

Przykładowe użycie w MeleeChase i MeleeSM

- **EnemySM** - maszyna stanów odpowiadająca za kontrolowanie poruszania się oraz udostępnianie przydatnych metod dla każdej konkretnej maszyny stanów.

Sztuczna inteligencja - podstawy

```
12 references
protected virtual void AwakeInitialize()
{
    enemySM = GetComponent<EnemySM>();
    anim = GetComponent<Animator>();
    attackOngoing = false;

    enemyMagicSystem = GetComponent<EnemyMagicSystem>();
    spellCount = enemyMagicSystem.GetSpellCount();
    dead = new EnemySpecificDead(enemySM, this);

    InitializeSpellChances();

    RandomizeStateTimes();
}
```

```
public virtual void Die()
{
    ScoreManager.AddToScore();
    enemySM.SetAlive(false);
    ChangeState(dead);
    Notify();
}
1 reference
public void DestroyEnemy()
{
    Destroy(gameObject);
}

1 reference
public bool IsAlive()
{
    return enemySM.IsAlive();
}
```

Góra - użyteczne dla wszystkich konkretnych maszyn stanu
Dół - użyteczne dla zewnętrznych elementów systemu

- **EnemySpecific** - abstrakcyjna klasa, która zawiera wspólne cechy potrzebne konkretnym maszynom stanów oraz udostępnia jednolite interfejsy dla zewnętrznych elementów programu

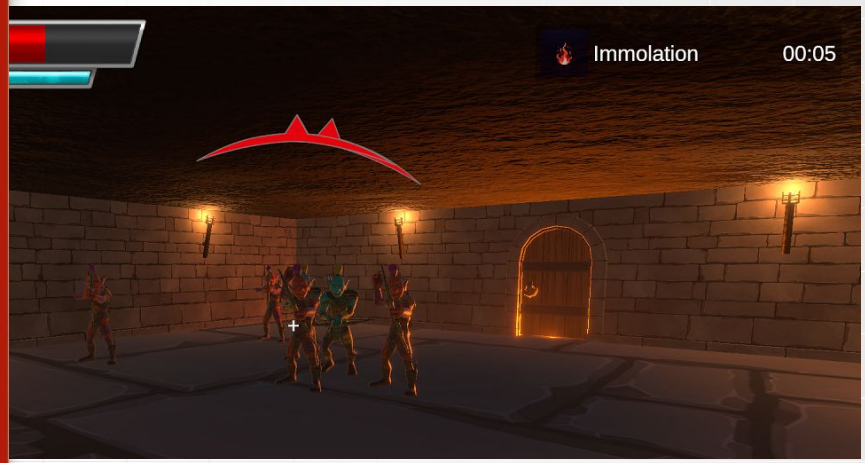
Sztuczna inteligencja - MeleeSM

Grupa szkieletów osaczających gracza

Szkielet:

- prosty kontaktowy przeciwnik
- wrażliwy na ogień, ziemię, światło
- niebezpieczny w dużych grupach

Sztuczna inteligencja - RangedSM



Goblińscy łucznicy strzelają i czasem nakładają efekty

Goblin:

- strzela do gracza
- czasem nakłada efekty
- ucieka, gdy gracz blisko
- różne odporności w zależności od wariantu
- stałe zagrożenie

Sztuczna inteligencja - ShamanSM

Szamani mają różne animacje w zależności od używanego zaklęcia

Szaman:

- korzysta z puli zaklęć
- częstotliwość zależna od mocy zaklęcia
- używa defensywnego lub odrzucającego zaklęcia, gdy gracz jest blisko
- różne odporności w zależności od wariantu
- stałe i poważne zagrożenie

Sztuczna inteligencja - GolemSM

Gdy jest daleko od gracza golem rzuca daleko zasięgowe zaklęcia

Kamienny Golem:

- powolny i wytrzymały
- gdy gracz daleko rzuca zaklęcia ziemi
- silny atak z bliska
- odporny na ziemię, elektryczność, światło
- czyhające zagrożenie

Sztuczna inteligencja - GolemSM

Gdy jest daleko od gracza golem zaczyna szarżować w jego stronę

Rogaty Golem:

- powolny i wytrzymały
- gdy gracz daleko szarżuje
- łatwe do uniknięcia
- silny, szybki atak z bliska
- odporny na powietrze
- źródło niepokoju

Sztuczna inteligencja - dynamiczne patrole

Zielone linie oznaczają najlepsze możliwe kierunki patrolu, czerwone oznaczają te z gorszym priorytetem (bo przeciwnicy na drodze)

Przeciwnik potrzebuje znaleźć wokół siebie punkt, do którego powinien być w stanie dotrzeć bez konieczności omijania ściany, ponadto najlepiej, aby minimalizowane były kolizje między przeciwnikami.

Sztuczna inteligencja - dynamiczne patrole

Zielone linie oznaczają najlepsze możliwe kierunki patrolu, czerwone oznaczają te z gorszym priorytetem (bo przeciwnicy na drodze)

Robimy w różnych kierunkach próbkowanie dostępności navmesha w celu weryfikacji dostępności ścieżki. W tych samych kierunkach Raycast sprawdzający obecność przeciwników na drodze. Losujemy z możliwie najlepszych.

Sztuczna inteligencja - dynamiczne patrole

```
public bool FindPatrolSpot(out Vector3 destination,float rotation=30f, float pathLength=8f, int sampleStep=8, float minimumPathPercent = 0.5f)
{
    destination = stateMachine.GetAgentPosition();
    rotation = Mathf.Abs(rotation);
    if(rotation == 0)
    {
        return false;
    }
    List<Vector3> potentialSpots = new List<Vector3>();
    List<Vector3> potentialBetterSpots = new List<Vector3>();
    Vector3 tempVector = stateMachine.GetForward();
    for( float currentRotation = 0f;  currentRotation <= 360f;  currentRotation += rotation )
    {
        tempVector = stateMachine.GetForward();
        tempVector = stateMachine.RotateVector(tempVector, yRotation: currentRotation);
        Vector3 tempDestination = Vector3.zero;
        if(stateMachine.MakePathDestination(out tempDestination, tempVector, pathLength, sampleStep,minimumPathPercent))
        {
            potentialSpots.Add(tempDestination);
        }
        else
        {
            continue;
        }
        if(!stateMachine.Raycast(tempVector, pathLength*minimumPathPercent))
        {
            potentialBetterSpots.Add(tempDestination);
        }
    }
```

```
if( potentialSpots.Count == 0)
{
    return false;
}
if( potentialBetterSpots.Count != 0 )
{
    destination = GetRandomElement(potentialBetterSpots);
    return true;
}
destination = GetRandomElement(potentialSpots);
return true;
```

Realizacja FindPatrolSpot w kodzie

Sztuczna inteligencja - dynamiczne patrole

```
public bool MakePathDestination(out Vector3 destination, Vector3 direction, float pathLength = 3f, int sampleStepCount = 4,
    float minimumPathLengthPercent = 0.5f)
{
    destination = Vector3.zero;
    direction = direction.normalized;
    Vector3 bestDestination = GetAgentPosition();
    for (int i = 1; i <= sampleStepCount; i++)
    {
        destination = GetAgentPosition() + direction * pathLength * ((float)i / sampleStepCount);
        if (SampleDestination(destination))
        {
            bestDestination = destination;
        }
        else if ((float)i >= sampleStepCount*minimumPathLengthPercent)
        {
            break;
        }
        else
        {
            destination = GetAgentPosition();
            return false;
        }
    }
    destination = bestDestination;
    destination = bestDestination;
    if (!IsDestinationValid(destination))
    {
        return false;
    }
    return true;
```

```
public bool SampleDestination(Vector3 destination, float? maxRadius = null)
{
    if (maxRadius == null)
    {
        maxRadius = GetDefaultSampleRadius(destination);
    }
    return NavMesh.SamplePosition(destination, out NavMeshHit hit, maxRadius.Value, 1);
}
```

Realizacja MakePathDestination w kodzie

System statystyk - Skalowanie obrażeń

- **Skalowanie obrażeń** - umożliwia wpływ statystyk na skuteczność zaklęć, zarówno podczas ich rzucania, jak i podczas bycia przez nie zranionym.
-

Pożądane cechy:

- Podobna użyteczność statystyk podstawowych
- Zauważalny wpływ
- Miejsce na elastyczność
- Balans

Skalowanie obrażeń

Statystyki gracza i przeciwników wpływają na otrzymywane oraz zadawane obrażenia. Dzielą się one na podstawowe i elementowe.

- Statystyki podstawowe zmieniają bazową wartość obrażeń poprzez dodawanie
- Statystyki elementowe zmieniają wartość obrażeń przez mnożenie

Skalowanie obrażeń

Statys
otrzyr
one n

- Sta
  ob
- Sta
  pr

ość

żeń

Słownik

- skalowanie – wpływ grupy parametrów na wartość innego parametru
- element – typ ataku, żywioł, np. ogień, woda, fizyczny, ciemność
- statystyki podstawowe – statystyki niepowiązane jednoznacznie z żadnym elementem t.j. siła, zręczność, inteligencja, obrona
- statystyki elementowe – statystyki dotyczące konkretnego elementu, np. FireDamage, AirResistance, PhysicalDamage
- statystyki ofensywne – statystyki elementowe wpływające na zwiększenie zadawanych obrażeń danego typu, np. WaterDamage, DarknessDamage, PhysicalDamage
- statystyki defensywne – statystyki elementowe wpływające na zmniejszenie otrzymywanych obrażeń danego typu, np. FireResistance, PhysicalResistance, ElectricityResistance (nie defense!)
- Entity – jednostka, istota, również nazwa abstrakcyjnego komponentu, po którym dziedziczy Enemy i Player

Skalowanie obrażeń

Statysty ıty
(pierws
- **Ogi**
- **Wo**
- **Zie**
- **Pov**
- **Ele**
- **Cie**
- **Jas**
- **Fizy**

Warto z ‘0.1)

Statystyki elementowe

- Przy wartości statystyk elementowych 0 nie wywierają one żadnego wpływu na obrażenia.
- Przy wartości statystyk elementowych -200 zmniejszają one obrażenia o około 80% (mnożone są przez 0.2)
- Przy wartości statystyk elementowych 200 zwiększają one obrażenia o około 150% (mnożone są przez 2.5)
- Przy wartości statystyk elementowych -100 zmniejszają one obrażenia o 50% (mnożone są przez 0.5)
- Przy wartości statystyk elementowych 100 zwiększają one obrażenia o 75% (mnożone są przez 1.75)
- Przy wartości statystyk elementowych -700 zmniejszają one obrażenia o 90% (mnożone są przez 0.1)
- Przy wartości statystyk elementowych 700 zwiększają one obrażenia o 300% (mnożone są przez 4.0)

03

Skalowanie obrażeń

Statystyki podstawowe niejednakowo wpływają na różne elementy (pierwsza wymieniona statystyka ma większy wpływ):

- **Ogień** - inteligencja*0.35, siła*0.25
- **Woda** - zręczność*0.4, siła*0.2
- **Ziemia** - siła*0.35, obrona*0.2
- **Powietrze** - zręczność*0.35, inteligencja*0.25
- **Elektryczność** - siła*0.4, zręczność*0.2
- **Ciemność** - siła*0.35, suma statystyk ofensywnych *0.03
- **Jasność** - zręczność*0.35, suma statystyk defensywnych *0.03
- **Fizyczne** - siła*0.4, zręczność*0.2

Warto zaznaczyć, że inteligencja wpływa na każdy rodzaj magii (*0.1)

Sztuczna inteligencja - większa naturalność

```
2 references
public float GetStateTimeVariancePercent()
{
    return stateTimeVariancePercent;
}
0 references
public void SetStateTimeVariancePercent(float percent)
{
    stateTimeVariancePercent = Mathf.Clamp(percent, -maxStateTimeVariancePercent, maxStateTimeVariancePercent);
    stateTimeVariancePercent = Mathf.Abs(stateTimeVariancePercent);
}
27 references
public float RandomizeStateTime(ref float stateTime)
{
    float statePercent = 1f + UnityEngine.Random.Range(-GetStateTimeVariancePercent(), GetStateTimeVariancePercent());
    stateTime = statePercent * stateTime;
    return stateTime;
}
5 references
protected abstract void RandomizeStateTimes();
```

```
protected override void RandomizeStateTimes()
{
    RandomizeStateTime(ref restTime);
    RandomizeStateTime(ref patrolTime);
    RandomizeStateTime(ref chaseTime);
    RandomizeStateTime(ref hurtTime);
    RandomizeStateTime(ref reactionTime);
    RandomizeStateTime(ref focusTime);
    RandomizeStateTime(ref attackCooldown);
}
```

Zmiana wartości czasów stanów o pewien losowy procent, w odpowiednim zakresie

Sztuczna inteligencja - większa naturalność

```
[SerializeField, Range(0f, maxStateTimeVariancePercent)]
protected float stateTimeVariancePercent = 0.1f;
protected const float maxStateTimeVariancePercent = 0.5f;
[SerializeField]
protected int[] spellChanceWeights;
[HideInInspector]
protected int spellWeightSum = 0;

12 references
protected virtual void AwakeInitialize()
{
    enemySM = GetComponent<EnemySM>();
    anim = GetComponent<Animator>();
    attackOngoing = false;

    enemyMagicSystem = GetComponent<EnemyMagicSystem>();
    spellCount = enemyMagicSystem.GetSpellCount();
    dead = new EnemySpecificDead(enemySM, this);

    InitializeSpellChances();

    RandomizeStateTimes();
}
```

```
Percent, maxStateTimeVariancePercent);

ncePercent(), GetStateTimeVariancePercent());
ed override void RandomizeStateTimes()

domizeStateTime(ref restTime);
domizeStateTime(ref patrolTime);
domizeStateTime(ref chaseTime);
domizeStateTime(ref hurtTime);
domizeStateTime(ref reactionTime);
domizeStateTime(ref focusTime);
domizeStateTime(ref attackCooldown);
```

Sztuczna inteligencja - większa naturalność

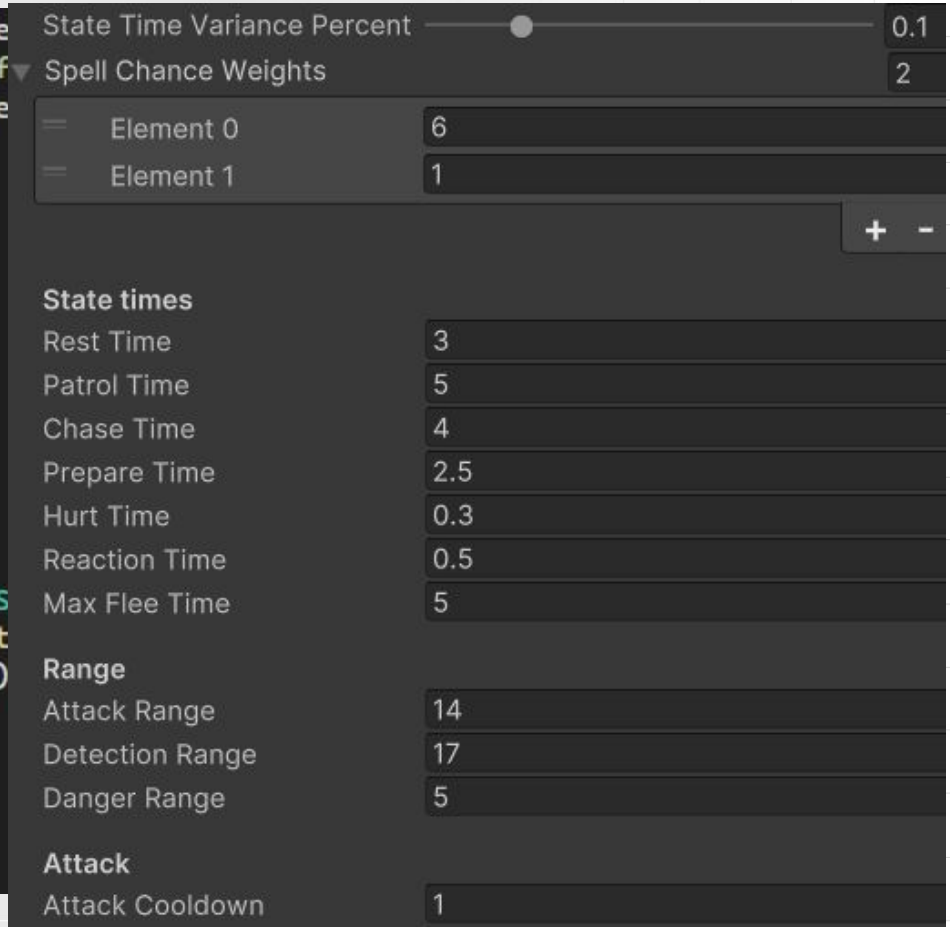
```
[SerializeField, Range(0f, maxStateTimeVariance
protected float stateTimeVariancePercent = 0.1f
protected const float maxStateTimeVariancePerce
[SerializeField]
protected int[] spellChanceWeights;
[HideInInspector]
protected int spellWeightSum = 0;

12 references
protected virtual void AwakeInitialize()
{
    enemySM = GetComponent<EnemySM>();
    anim = GetComponent<Animator>();
    attackOngoing = false;

    enemyMagicSystem = GetComponent<EnemyMagicS
    spellCount = enemyMagicSystem.GetSpellCount
    dead = new EnemySpecificDead(enemySM, this)

    InitializeSpellChances();

    RandomizeStateTimes();
}
```



Sztuczna inteligencja - większa naturalność

```
[SerializeField, Range(0f, maxStateTimeVariance
protected float stateTimeVariancePercent = 0.1f
protected const float maxStateTimeVariancePerce
[SerializeField]
protected int[] spellChanceWeights;
[HideInInspector]
protected int spellWeightSum = 0;

12 references
protected virtual void AwakeInitialize()
{
    enemySM = GetComponent<EnemySM>();
    anim = GetComponent<Animator>();
    attackOngoing = false;

    enemyMagicSystem = GetComponent<EnemyMagicS
    spellCount = enemyMagicSystem.GetSpellCount
    dead = new EnemySpecificDead(enemySM, this)

    InitializeSpellChances();

    RandomizeStateTimes();
}
```