INTERACTIVE AGENT DESIGN

Behaviour Tree Guard

Mijn **Guard** behaviour tree bestaat uit één selector als root, met daarin twee sequences om heen en weer tussen te gaan en tot slot een fallback task genaamd Patrol.

```
protected override Node SetupTree()
{
    Node root = new Selector(new List<Node>
    {
        new Sequence(new List<Node>
        {
            new CheckSightBlocked(transform, animator, blockedSightLayer, this),
            new CheckCanAttack(this),
            new CheckEnemyInAttackRange(transform, this),
            new TaskAttack(transform, animator, this),
        }),
        new Sequence(new List<Node>
        {
            new CheckSightBlocked(transform, animator, blockedSightLayer, this),
            new CheckEnemyInFOVRange(transform, headTransform, enemyLayer, obstructionLayer, this),
            new CheckForItem(transform, headTransform, pickupableLayer, obstructionLayer, this, pickupableFindRange),
            new TaskGoToItem(transform, animator, this),
            new TaskPickUpItem(animator, this, attackManager),
            new TaskGoToTarget(transform, headTransform, animator, this),
        }),
        new TaskPatrol(transform, animator, waypoints, this),
    });;

    return root;
}
```

De eerste sequence van Nodes bestaat uit het checken en aanvallen van de Guard zijn target. Bij **CheckSightBlocked** wordt er puur gechecked of de Guard momenteel in een rookbom staat of niet, zo ja dan gaat de Guard uit deze sequence, anders dan gaat deze door met de volgende check. De volgende check is **CheckCanAttack** en daarbij wordt er gekeken of er al wordt aangevallen en als er aangevallen is, dan wordt daar de timer bijgehouden totdat de Guard weer kan aanvallen.

Bij **CheckEnemyInAttackRange** wordt er gekeken of de target in de radius is van het huidige wapen, dit kan verschillen tussen de default attack, wat trappen is en tussen de Axe en AK. Vervolgens wordt de **TaskAttack** uitgevoerd waar de correcte animaties en acties worden uitgevoerd.

Bij de tweede sequence van Nodes wordt er gechecked of de Guard geblindeerd is door een rookbom bij **CheckSightBlocked** en vervolgens wordt er gechecked als er targets zijn binnen de range, of deze ook in de FOV van de Guard staan, zo ja dan wordt er nog een laatste check gedaan of er een item in de FOV ligt, wat een wapen of peper kan zijn in dit geval. Tot slot worden de Tasks uitgevoerd, om te beginnen bij **TaskGoToItem**, waarbij er eerst wordt gekeken of er een item is gevonden, zo ja dan gaat de Guard naar dit item toe en pakt hij deze bij **TaskPickUpItem** op. Tot slot gaat de Guard achter de target aan in **TaskGoToTarget**, wat ervoor zorgt dat er een pad gecalculeerd wordt op de NavMesh en dat deze gevolgd wordt met de juiste snelheid.

Onder de twee sequences staat een **TaskPatrol**, dit is de fallback Node voor als beiden sequences falen. Hier wordt gechecked of er niet wordt aangevallen, zo niet dan gaat deze zijn weg voortzetten tussen van te voren ingestelde waypoints.

States: Aanvallen, Naar een Item gaan, Een Item oppakken, Naar target gaan en patrolleren.

Behaviour Tree Ally

Mijn **Ally** behaviour tree bestaat uit één selector als root, met daarin één sequence om doorheen te gaan, met tot slot een fallback task genaamd FollowMaster.

```
protected override Node SetupTree()
{
    Node root = new Selector(new List<Node>
    {
        new Sequence(new List<Node>
        {
            new CheckCanAttackEnemy(this),
            new CheckMasterBeingAttacked(master),
            new TaskFindCover(transform, animator, this, master, hideableLayers),
            new TaskAttackEnemy(transform, animator, this, master),
        }),
        new TaskFollowMaster(transform, animator, this, master),
    });

    return root;
}
```

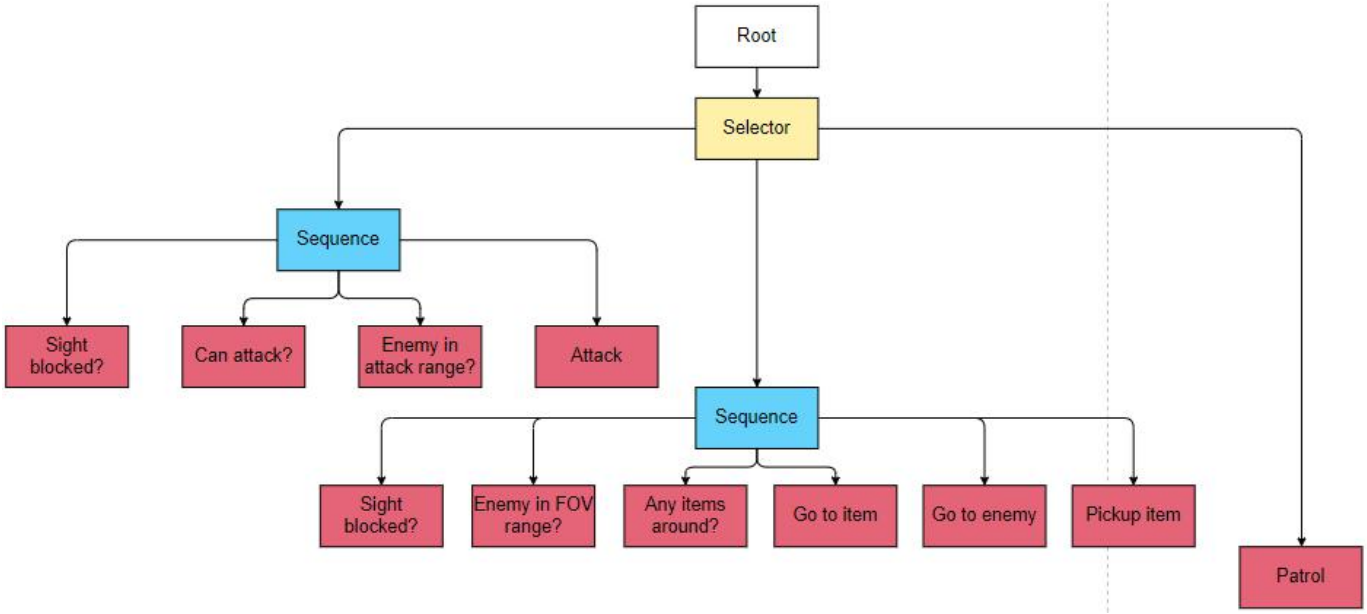
De sequences van Nodes bestaat uit het checken of er aangevallen kan worden, zo ja dan wordt er gechecked of zijn Master wordt aangevallen. Als deze wordt aangevallen dan zoekt de Ally cover. Dit doet hij op basis van colliders binnen layers. Als er colliders zijn wordt er gechecked hoe deze in relatie staat tot het target, de guard in dit geval, en als deze van de guard af staat, dan gaat de ally daar naartoe. Tot slot gaat de Ally smoke granaten werpen op de target met een simpele berekening die niet al te accuraat is. Als de guard dichtbij staat is hij vrij inaccuraat en als de guard heel ver weg staat ook, dus daar moet je als speler een balans tussen vinden.

Tot slot is er een **TaskFollowMaster** die wordt uitgevoerd als de sequence faalt. Hierbij gaat de ally zijn Master volgen tot een bepaalde afstand van de speler.

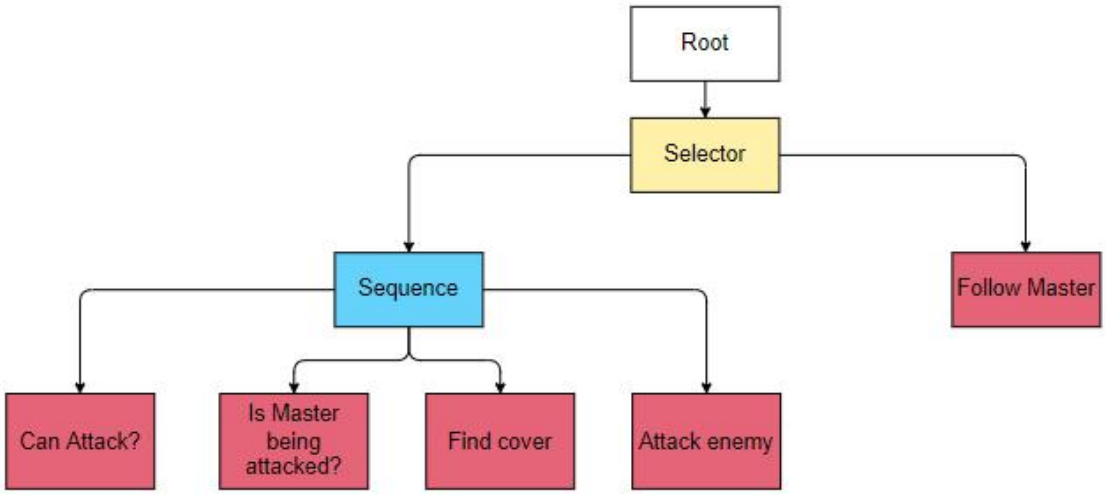
States: Cover vinden, Aanvallen en zijn Master volgen.

UML Diagram

Om het overzichtelijk te maken heb ik de vormgeving in een schuine trap naar beneden gedaan. Hieronder is de UML Diagram van de Guard te zien.



Hieronder is de UML Diagram van de Ally te zien.



Ontwerpkeuzes

Ik heb voor beiden AI's de keuze gemaakt om de Behaviour Tree toe te passen, omdat ik het moeilijk vond om ergens mee te beginnen en snel een goede tutorial vond over een Behaviour Tree maken. Veel tutorials die ik vond waren lastig te begrijpen en ingewikkeld in elkaar gezet. Vandaar dat ik ook niet de scripts van het Starter Project heb gebruikt, omdat dit er te ingewikkeld uitzag. Ondanks dat de Behaviour Tree die ik heb gemaakt misschien niet een typische Behaviour Tree is, werkt het wel hetzelfde als een typische Behaviour Tree, maar dan wat simpeler.

Daarnaast vond ik het ook niet nodig om voor beiden AI's een ander systeem dan een Behaviour Tree te gebruiken, een Utility systeem of GOAP systeem zou leuk zijn geweest, maar ik ben tevreden met de keuze die ik heb gemaakt. De andere systemen kan ik in de toekomst toepassen.

Ik heb ervoor gekozen om gebruik te maken van de NavMesh van Unity, omdat deze goed werkt met het ontwijken van obstakels, wel heb ik de rotatie handmatig gedaan, omdat er anders rare artefacten ontstaan, zoals glijdende agents.

Ik heb wel de models van de Guard en Ninja van het starter project gebruikt, omdat deze goed aansloten bij de opdracht. Daarnaast heb ik wel alle animaties opnieuw bij elkaar gezocht op Mixamo en heb ik alle Animators opnieuw gedaan, zodat alles goed en soepel liep.

PMI

Pluspunten:

• Het is een gemakkelijk te gebruiken systeem, waardoor ik snel meerdere states kon maken, zonder dat ik in de war raakte.
• Het is gemakkelijk om bepaalde condities te maken en door de checks en tasks uit elkaar te houden blijft het systeem overzichtelijk.

Minpunten:

• Het is even puzzelen om interrupties te maken, maar door slim gebruik te maken van checks kun je hier aardig mee door komen.

• Ik heb veel variabelen aangemaakt om ervoor te zorgen dat alle checks en tasks goed liepen, hierdoor voelt het toch een beetje als cheaten.

Interessante punten:

• Het is een enorm krachtig systeem waar veel mogelijk mee is, het ligt er alleen aan hoe je dit aanpakt.
• Nadat ik klaar was met bepaalde tasks coderen, kwam ik erachter dat deze veel te specifiek waren, waardoor het me aan het denken zette om dit modulair te maken.

Conclusie

Ik vond het een hartstikke leuke opdracht om te doen, al kostte het me wel even om wat motivatie en interesse te vinden. Vanaf het moment dat ik een tijdje in de tutorial was en het opeens snapte, begon het interessant en leuk te worden. Het moeilijkste hiervan is, dat je jezelf er wel aan toe moet zetten om te starten en om de informatie toe te laten.

Ik ga zeker de Behaviour Tree gebruiken in toekomstige projecten, nu ik weet dat deze helemaal niet zo lastig is om te maken en door deze te coderen heb ik veel progressie gemaakt met de manier van coderen en kijken naar classes.