Sten Duindam

KGDEV3 [Herkansing]

Artificial Intelligence

Dungeon Generator

29-6-2018

Inleiding

Voor kernmodule 3 ben ik uitgedaagd om twee verschillende artificial intelligence te maken gepaard met een dungeon die middels procedural generation tot stand komt. Klinkt ingewikkeld he? Is het ook. In feite geef je de computer een aantal regels en vervolgens maakt hij iets naar jouw regels toe.

In dit document beschrijf ik mijn twee AI en dungeon generator.

Inhoud

Proces

Tijdens de lessen kwam ik erachter hoeveel verschillende manieren er wel niet zijn om AI te benaderen of om procedural generated content te maken. Op het bord en in de les leek de stof goed te behappen. Duidelijk, dacht ik. Dat kan ik ook wanneer ik er tijd voor maak.

Helaas is dit niet helemaal goed gegaan.

Na een aantal weken niet focussen op de kernmodule ben ik ervoor gaan zitten. Ik wilde een Behaviour Tree maken. Dit omdat ik graag buiten de unity functionaliteiten wilde gaan én liever tijdens mijn oefeningen geen gebruik maak van assets. Dit resulteerde in drie of vier halve pogingen welke alle stuk voor stuk faalde.

De dungeon generator daarentegen, die lukte wel. Het kwam overeen met Conways Game Of Life die we met de keuzemodule Heavy Object Oriënted Programming gemaakt hebben. Deze lukte mij dan ook wel.

Ik bleef maar stuklopen op de Behaviour Tree maar door alle drukte heb ik niet het besluit genomen hulp te vragen. Dit resulteerde in mijn eerste onvoldoende voor de kernmodule.

Na deze onvoldoende besloot ik toch meer informatie te halen bij Valentijn en mijn bevindingen voor te leggen. Met als gevolg de Behaviour Tree zoals hij nu in volle glorie staat te bloeien.

En naast de boom, een trotse eigenaar van dit project.

AI 1 - HideBot

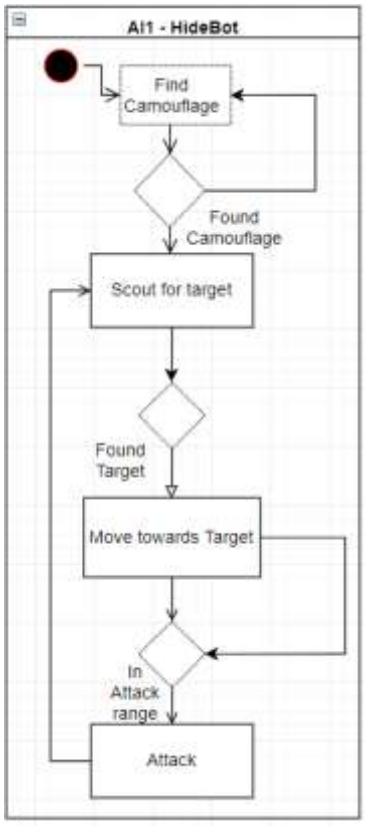
Het idee achter de eerste AI, genaamd HideBot, is een passieve AI die de speler of zijn doelwit aanvalt door zich eerst te verstoppen en vervolgens aan te vallen wanneer het doelwit dichtbij genoeg is. Tijdens het verstoppen veranderd de AI zijn kleur naar die van de omgeving. Daarnaast is de AI niet te detecteren terwijl hij verborgen is.

HideBot is opgezet met een Behaviour Tree en bevat 5 kern acties, namelijk:

- Dekking zoeken
- Een doelwit vinden
- Naar het doelwit toe bewegen
- Het doelwit aanvallen
- In willekeurige richtingen bewegen om dekking te vinden

```
//Setup Behaviour tree
blackboard = new BlackBoard(this.gameObject);

root = new Selector(
    new Sequencer(
        //Go into hiding, to ambush later
        new Camouflage(blackboard),
        new ScoutForTarget(blackboard),
        new MoveToTarget(blackboard),
        new AttackTarget(blackboard, stats, AttackStyle.melee, null)

        ),
    new Selector(
        //walk around and look for a spot to hide)
        new MoveAroundRandom(blackboard)
        )
);
```

AI 1 - Flowchart



AI 2 - ShootBot

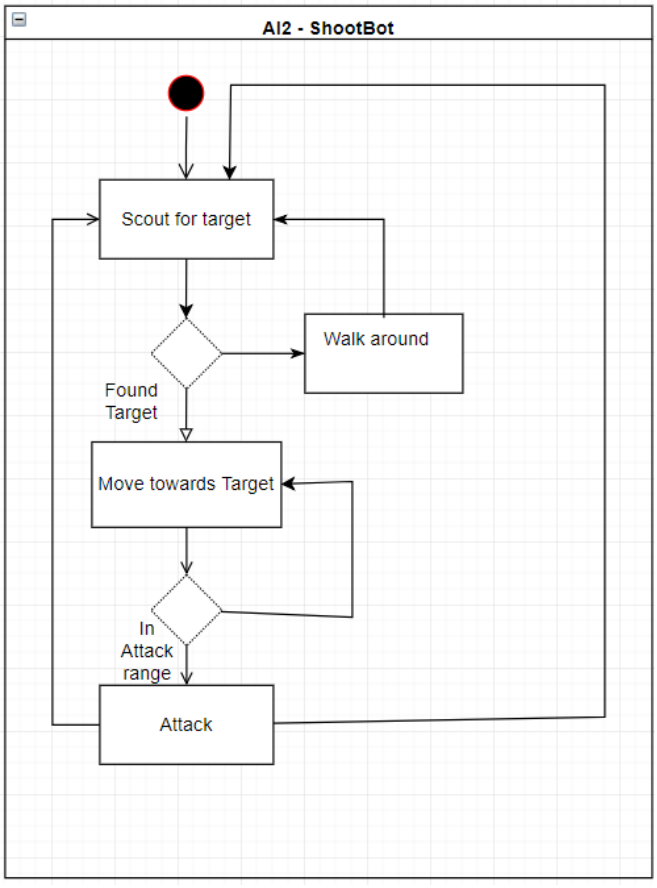
In tegenstelling tot HideBot loopt de ShootBot rond op een navmesh. Hij patrouilleert door de verschillende kamers opzoek naar een doelwit. Wanneer hij een doelwit heeft gevonden gaat hij schieten.

ShootBot bevat 4 kern acties:

- Een doelwit vinden
- Naar het doelwit toe bewegen
- Het doelwit aanvallen
- Van punt naar punt bewegen via navmesh

```
//Setup tree
blackboard = new BlackBoard(this.gameObject);

//Edit blackboard variables
blackboard.isNavmeshAgent = true;
blackboard.myRange = 2;
blackboard.lookRange = 1;

root = new Selector(
    new Sequencer(
        new ScoutForTarget(blackboard),
        new MoveToTarget(blackboard),
        new AttackTarget(blackboard,stats, AttackStyle.range, projectile)
        ),
    //Patrol
    new Patrol(blackboard, waypoints)
    );
```

AI 2 - Flowchart



Behaviour Tree

Beide AI zijn gebouwd met de Behaviour Tree. De Behaviour Tree wordt opgezet in de start functie of intialisatie van het script. Een Behaviour Tree heeft drie hoofdonderdelen:

- **Enum ReturnType**
  Iedere functie die uitgevoerd kan worden door de Behaviour Tree heeft drie mogelijke return waarden. **Success** wanneer een actie correct is uitgevoerd, **Running** wanneer de actie momenteel bezig is met zijn doel bereiken (bijvoorbeeld ergens heen bewegen) en **Failure** wanneer de actie niet mogelijk is om uit te voeren.

- Selector & Sequencer
Afhankelijk van bovenstaande returntypes voeren deze scripts acties uit.
Een **Sequencer** doet alles in volgorde van aangeleverde parameters, zo voert hij eerst functie 1 uit en pas wannneer deze success terugkoppelt gaat hij door naar functie 2 enz. Dit is handig om bijvoorbeeld hierärchie te creëren.
Een **Selector** stopt met door zijn acties heen werken wanneer er eentje success resulteert. Wanneer een actie Failure terugkoppelt gaat hij door naar zijn volgende actie om te kijken of die wel uitgevoerd kan worden.

```
public class Sequencer : BTNode {

    private BTNode[] nodes;

    public Sequencer(params BTNode[] nodes) {
        this.nodes = nodes;
    }

    public override ReturnType Run() {

        foreach(BTNode node in nodes) {
            ReturnType result = node.Run();
            switch (result) {
                case ReturnType.Success:
                    break;
                case ReturnType.Failure:
                    return ReturnType.Failure;
                case ReturnType.Running:
                    return ReturnType.Running;
            }
        }

        return ReturnType.Success;

    }
}
```

Selector

```
using UnityEngine;

public class Selector : BTNode {

    private BTNode[] nodes;

    public Selector(params BTNode[] nodes) {
        this.nodes = nodes;
        }

    public override ReturnType Run() {

        foreach (BTNode node in nodes) {
            ReturnType result = node.Run();
            switch (result) {
                case ReturnType.Success:
                    Debug.Log("Success!");
                    return ReturnType.Success;
                case ReturnType.Failure:
                    break;
                case ReturnType.Running:
                    break;
                }
            }


        return ReturnType.Failure;
        }
    }
```

Sequencer

- BlackBoard

Iedere instantie van een Behaviour Tree heeft ook variabele nodig om mee te werken, denk aan posities, het betreffende GameObject of beweeg snelheid.

Dit wordt allemaal opgeslagen in een BlackBoard. Dit BlackBoard wordt meegegeven als parameter naar de geïnstantieerde acties zodat alle acties toegang hebben tot de variabelen. En dus geen referentie nodig hebben naar andere acties of scripts.

```
public class BlackBoard {

    public BlackBoard(GameObject myObject) {
        myAIObject = myObject;
        }

    //Target
    public GameObject target;
    public Vector3 targetPosition() {
        if (target != null) {
            return target.transform.position;
            }
        else {
            return new Vector3(0, 0, 0);
            }
        }

    //Self
    public BTNode currentAction;
    public GameObject myAIObject;
    public Vector3 currentPosition() { return myAIObject.transform.position;}
    public bool hasTarget() ...

    public bool canMove = true;
    public float moveSpeed = 0.25f;
    public bool isNavmeshAgent = false;
    public bool canAttack = false;

    public int myRange = 2;
    public float lookRange = 1.5f;
    public Animator anim;
    }
```

BlackBoard

De acties van een Behaviour Tree worden leafs of nodes genoemd, de eigenschap van zo een script is dat het een Run() of Action() functie inherit van een abstract class. Op deze manier kan iedere actie uitgevoerd worden met dezelfde call functie en kan iedere actie in een Nodes[] toegevoegd worden voor eenvoudige referenties.

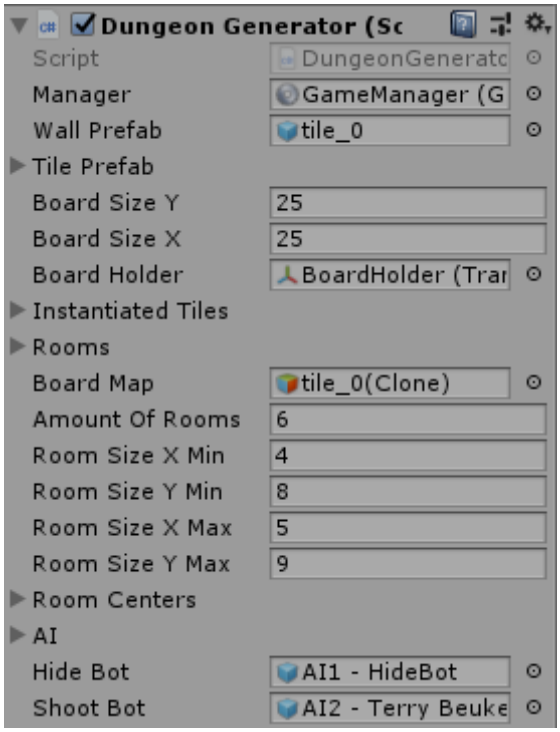
```
public enum ReturnType { Success, Running, Failure };

public abstract class BTNode: MonoBehaviour {

    public abstract ReturnType Run();

}
```

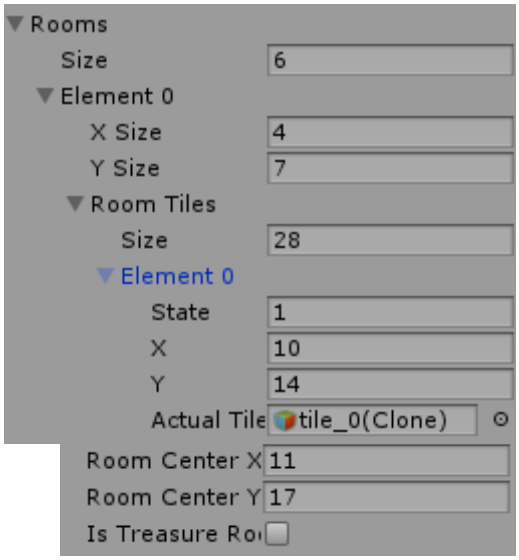
Base Node

Ik heb gekozen om een Behaviour Tree te maken omdat het een functionaliteit is van C# base, ik wilde testen of ik game gedrag kon maken zonder Unity functionaliteiten. Deze structuur is te gebruiken in welke taal ik ook ga programmeren. Daarnaast worden alle functies heel compact opgezet waardoor ze voor meerdere doeleinden gebruikt kunnen worden. Zo heb ik in AI1 en AI2 drie functies hergebruikt bij elkaar. Wanneer het systeem staat is het eenvoudig meerdere verschillende AI of uitwerkingen te maken. Vooral dit laatste vind ik erg interessant want het ontleend zich goed aan bijvoorbeeld een Tool.

Dungeon generator

Naast de twee AI is er ook een procedural dungeon generator waarin ze leven. In de editor ziet dat er zo uit, aan de hand van een aantal parameters wordt deze geïnstantiate:





De dungeon wordt opgezet met een 6-tal functies. Allereerst wordt het bord gemaakt, mocht er al een bord bestaan dan wordt de oude verwijderd. Het maken van het bord gebeurd door middel van een dubbele array met Tiles[,]. Door de Tiles[,] te linken aan twee getallen kunnen wij ze refereren naar X en Y-coördinaten, dit creëert een bord afhankelijk van de ingegeven X en Y-afmetingen.

```
35      void Start() ...
42
43      public Vector3 GetStartPos() ...
46
47      //To make a new board we clean the last one so nothing overlays or gets left behind
48      public void CleanBoard() ...
64
65      public void CreateBoard(int xSize, int ySize) ...
136
137     public void CreateRoom(int _startPosX, int _startPosY) ...
175
176     public void CreatePaths(Room _currentRoom) ...
220
221     //A function that checks in a one wider square than the actual to be placed room so there are no openings in walls or overlays in rooms.
222     public bool CheckIfRoomFits(int _startPosX, int _startPosY, int _roomSizeX, int _roomSizeY) ...
269
270     //Create tiles and objects based on the settings
271     public void InstantiateBoard() ...
301 }
302
303 [System.Serializable]
304 public class Tile ...
323
324 [System.Serializable]
325 public class Room ...
```

Vervolgens krijgt iedere Tile een enum waarde, namelijk: death, alive, wall of path. Dit wordt in de Tile opgeslagen om te zorgen dat we deze later weer kunnen bereiken om het bord te instantiaten.

Een Tile ziet er zo uit:

```
[System.Serializable]
public class Tile{
    [SerializeField]
    public int state;
    public int x;
    public int y;
    public enum tileState {death,alive,wall,path};
    public GameObject actualTile;

    //Functions to change variables in the class
    public void SetState(tileState _state) {
        state = (int)_state;
        }

    //Set positions
    public void SetPos(int _x, int _y) {
        x = _x;
        y = _y;
        }
}
```

Aan de hand van het aantal gewenste kamers worden er in de grid die gecreëerd is kamers gemaakt. Deze worden gecheckt op of ze passen, niet op een andere kamer worden gemaakt en binnen het grid zitten. Iedere kamer wordt opgeslagen in een list om ook weer eenvoudig toegang te bieden vanuit het script. De kamers passen de tiles aan binnen hun X en Y-afmeting, om aan te geven dat ze een kamer zijn. Zij krijgen de 'alive' state.

Na de kamers worden er gangen gemaakt. Dit gaat willekeurig van kamer[0] naar een andere kamer. De paden worden berekend vanuit het midden van elke kamer, deze kunnen we makkelijk bereiken omdat we iedere kamer + tiles hebben opgeslagen. Alle gangpad tiles krijgen de 'path' state. Wanneer een kamer geen pad heeft gekregen wordt het automatisch een Treasure Room. Hier kunnen bijvoorbeeld schatten staan.

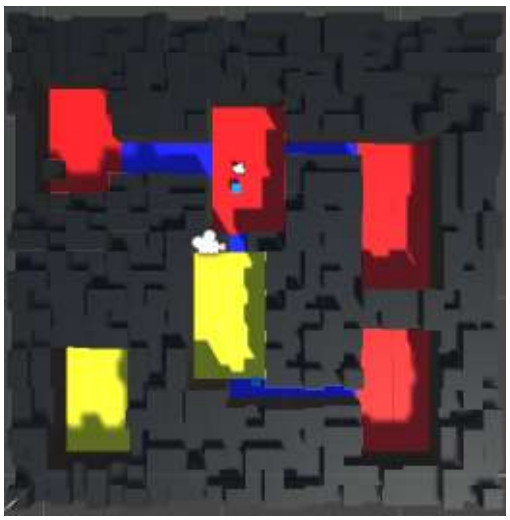
De speler krijgt het midden van kamer[0] als start positie.

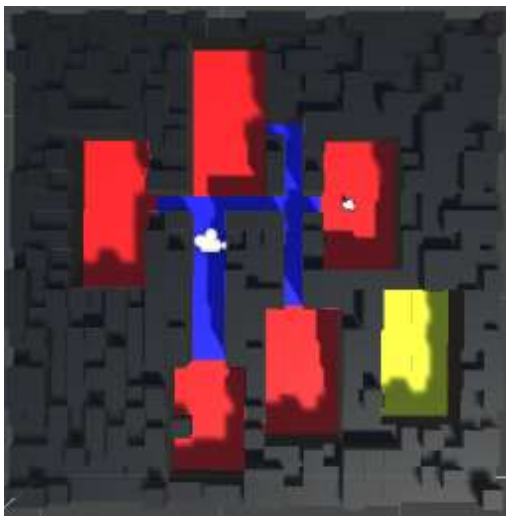
Nadat alle tiles een waarde hebben gekregen worden er voor iedere Tile Class GameObjects geïnstantiate en gelinkt aan de Tile. Zodat we bij het GameObject kunnen. Afhankelijk van de state krijgen ze een kleur en een scale.

Nadat de kamers gemaakt en geïnstantiate zijn worden er in een aantal willekeurige kamer AI geïnstantiate voor wat leven in de dungeon.

Met als resultaat:

Twee iteraties





Slotwoord

Na de eerste keer niet uit de stof te kunnen komen ben ik blij dat ik hulp gevraagd heb. De gedachte zat goed maar in de uitwerking ontbrak het hem. Zonder die middag zitten was ik er waarschijnlijk niet uitgekomen.

De procedural dungeon was erg leuk om te maken, misschien lag dit ook aan de weinig bugs die ik tegen kwam tijdens het maken...

Al met al is het een onwijs interessant blok geweest met hele goede stof. Wel was het wat hoog gegrepen voor mij vond ik. Maar dit lag ook zeker aan de inzet en uren die in het vak gestoken zijn van mij, met een goede focus had dit in één keer gelukt.