System Explanation

Foreword:

Voor deze opdracht heb ik een Utility System gemaakt waarmee ik 2 agents met verschillende states tegen elkaar kan laten vechten.

De assets die ik hiervoor gebruikt heb zijn allemaal afkomstig uit een ander project waar ik ze eerder zelf voor gemaakt heb.

About the system:

Het Utility System is een systeem waarbij alle keuzes gemaakt worden door middel van "conditionFactors". Deze conditionFactors bepalen samen de "utilityValue" voor een state en de state met de hoogste utilityValue wordt de nieuwe state.

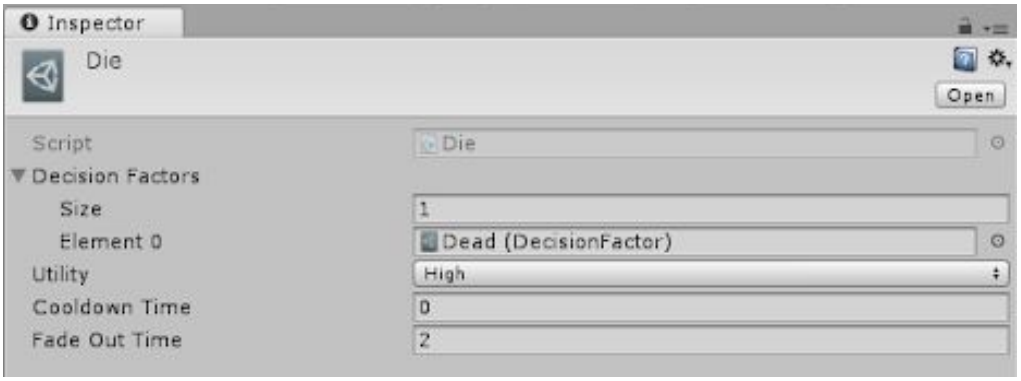
Ik heb er verder voor gekozen om alle conditionFactors met elkaar te vermenigvuldigen, en dit dan weer met de utility zelf te vermenigvuldigen. Dit ziet er in code dan als volgt uit:

```
public void CalculateUtility(){
	utilityValue = 0f;
	if (decisionFactors.Length > 0) {
		utilityValue = decisionFactors [0].Value (statsModel);

		for (int i = 1; i < decisionFactors.Length; i++) {
			utilityValue *= decisionFactors [i].Value (statsModel);
		}
	}

	utilityValue *= OveralUtilityFactor();
}

public float OveralUtilityFactor(){
	return (float)utility / 3f;
}
```

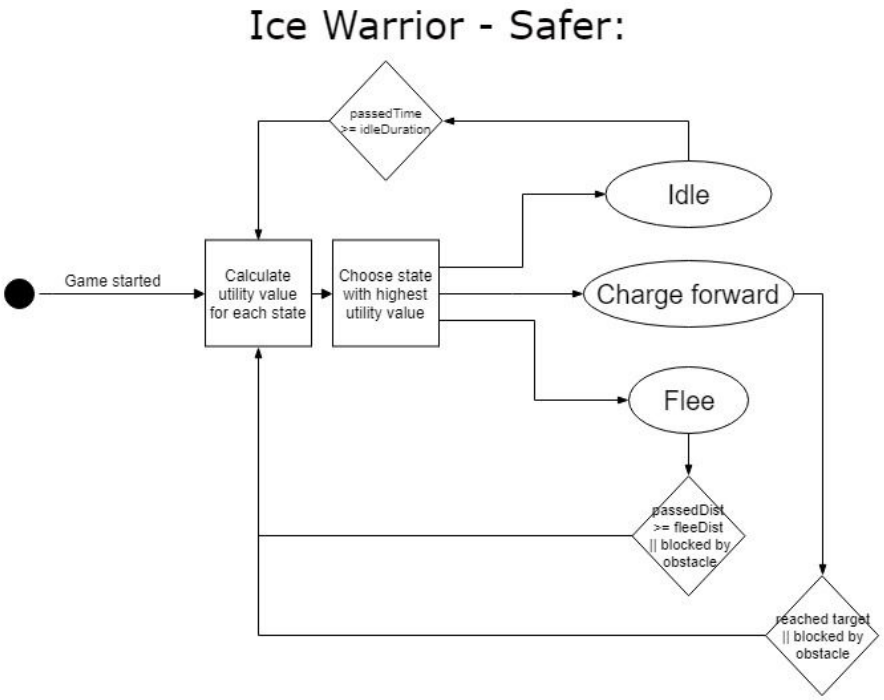
Mijn systeem werkt als volgt voor de gebruiker: de gebruiker kan zowel een conditionFactor of een state aanmaken als ScriptableObject. Deze state kan hij dan weer een aantal conditionFactors laten hebben, die nodig zijn om de transition naar deze state te maken.

Alle states zijn van tevoren in code gedefinieerd, net als alle variabelen die de speler in de conditionFactors kan gebruiken.

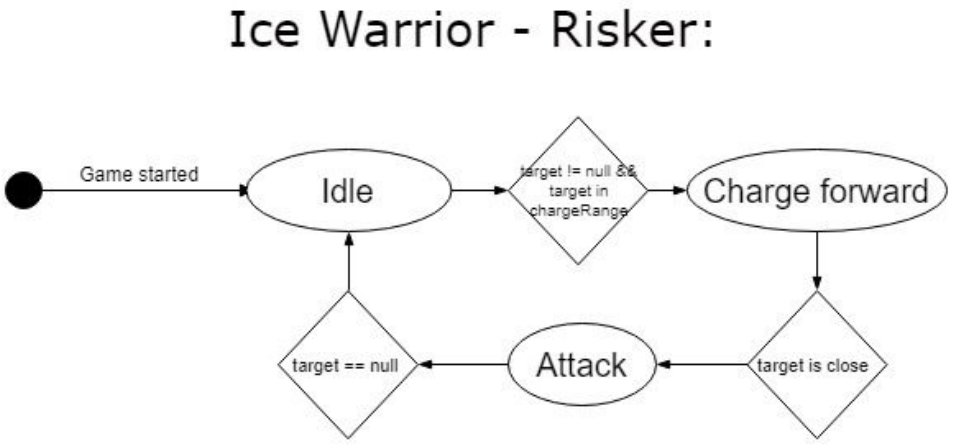
De Die-State ziet er dan bijvoorbeeld als volgt uit in de inspector:



Het gedrag van de eerste AI werkt grofweg als volgt *(ik heb een aantal states weggelaten, omdat dit het diagram minder overzichtelijk maakt terwijl het principe voor alle states vergelijkbaar is)*:

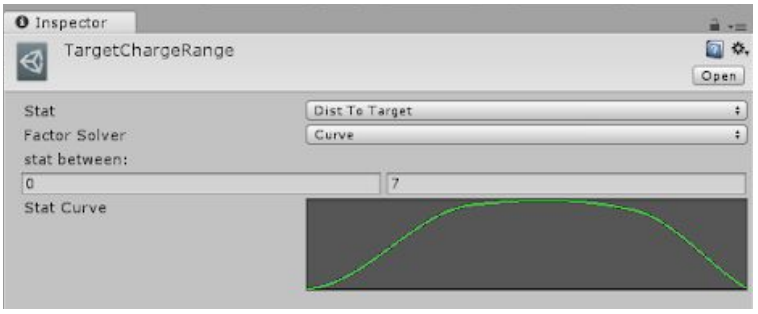


Voor de speler wordt het gedrag van zo'n agent dan als volgt waargenomen:

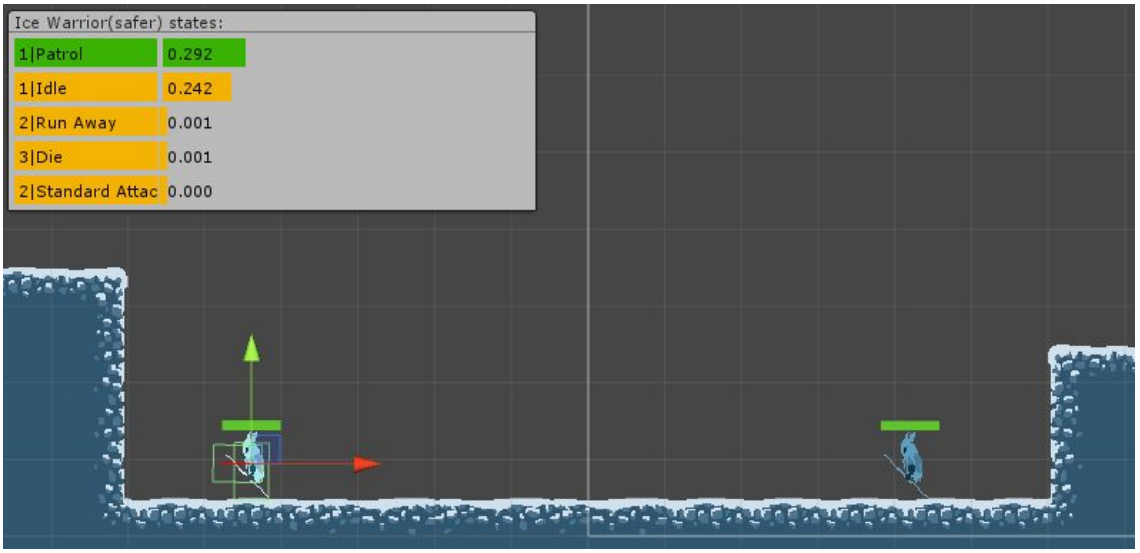


De conditionFactors zijn wat mij betreft het belangrijkste onderdeel van een Utility System. Daarom zal ik deze iets gedetailleerder uitleggen.
Elke conditionFactor maakt gebruik van een variabele. Deze variabele is een enum, die in code weer gelinkt wordt met een bepaald getal.
Ik heb er bewust voor gekozen om deze variabelen niet ook als ScriptableObject te maken omdat de werking van deze variabelen zo verschillend is. Zo heb ik bijvoorbeeld een variabele voor het aantal levens, maar ook een variabele voor de afstand naar het verste object die wordt berekend door een raycast naar links en naar rechts te doen en de verste afstand van de twee te pakken.

Wat verder een belangrijk element is aan de conditionFactors is dat ik de gebruiker meerdere manieren gegeven heb om een verband te verzinnen met een variabele. De gebruiker kan kiezen uit een boolean(de variabele is kleiner, of juist groter dan een bepaald getal), een random-vermenigvuldiger, of een curve. Ook heb ik met behulp van een custom inspector ervoor gezorgd dat de speler alleen relevante informatie te zien krijgt bij de geselecteerde opties voor zijn decisionFactor.
Dit ziet er dan als volgt uit in de inspector:



Ten slotte heb ik nog een tool gemaakt waarmee je in de inspector kunt zien wat de utilityValue van de verschillende states is. Hierdoor kun je vrij eenvoudig zien wat er goed werkt, of juist niet aan het beslissend vermogen van de agent. Dit ziet er dan als volgt uit:



Evade State:

Uiteindelijk heb ik besloten dat ik ook een Evade-state wou maken. Hierdoor zou het gedrag van de agents namelijk veel leuker kunnen worden.

Bij het maken van deze state kwam ik op een aantal interessante bevindingen. Ten eerste kwam ik erachter dat ik bij het maken van deze state wel gedrag moest definiëren wat ook al ongeveer voorkwam in andere states. (het laten bewegen van de agent) Hierdoor ben ik eraan gaan twijfelen of een meer modulaire aanpak niet toch beter zou kunnen werken.

Ten tweede was het aardig lastig om erachter te komen wat de snelheid van de andere AI was, en zou ik achteraf gezien liever een conditionFactor willen hebben die checked op welke state de andere agent heeft. Hiermee zou het een stuk makkelijker zijn om een Agent op het juiste moment een andere Agent te laten ontwijken!

Verder kwam ik erachter dat of de agent evade heel erg afhangt van of hij op het juiste moment klaar is met de vorige state. Dat is dus een beperking van mijn versie van een Utility System: je kunt niet de state-transitions precies timen. Uiteraard zou je hier wel iets voor kunnen maken; bijvoorbeeld een systeem waarbij de transitie naar de Evade state ook geëvalueerd wordt tijdens het runnen van states.

Conclusion:

Het is mij gelukt om een modulair utility system te maken. Ik ben tevreden met het resultaat omdat je als gebruiker veel invloed hebt over de manier waarop beslissingen genomen worden en omdat je als gebruiker dankzij mijn visualization-tool een goed beeld hebt van de utilityValues die berekend worden.

Ook ben ik veel te weten gekomen over hoe een Utility System werkt, wat de voordelen en de beperkingen van dit systeem zijn en hoe je het zelf maakt. Dit project was voor mij dus erg leerzaam!

Evade behaviour, final version: