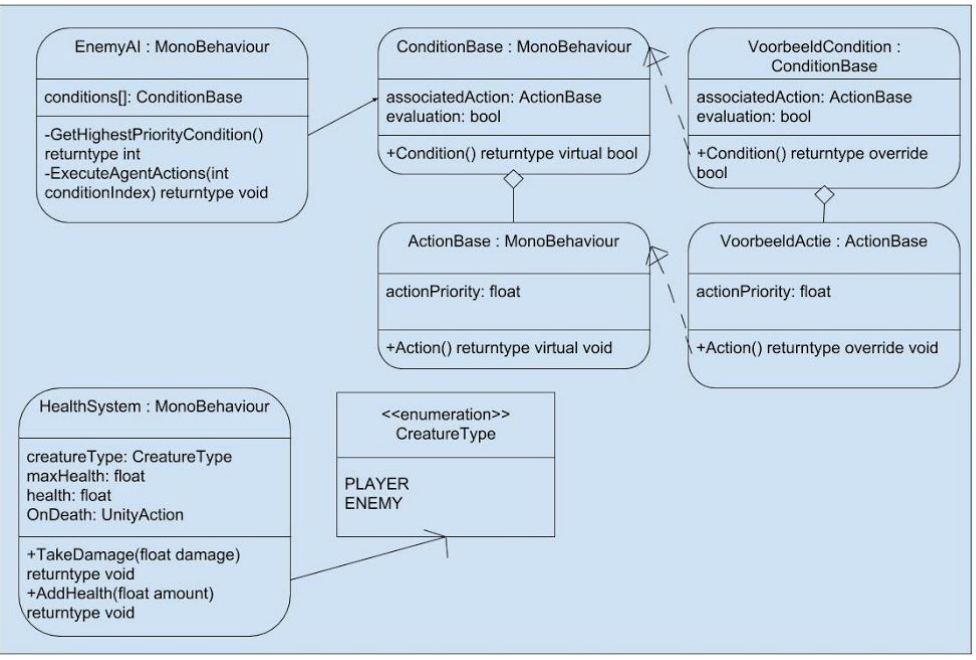
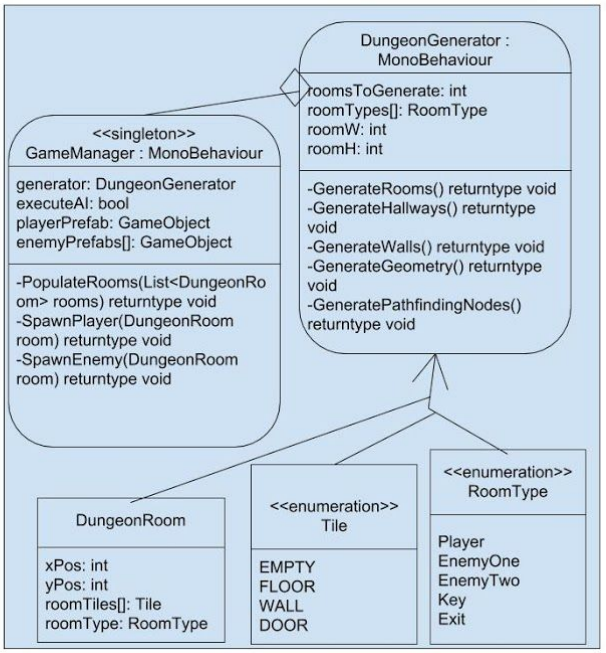
DOCUMENTATIE - ROEL ZWAKMAN
KERNMODULE 3

UML CLASS DIAGRAM





De source image staat ook op github.

ALGEMENE INFORMATIE

HOE ZIT DE GAMEPLAY IN ELKAAR?
De speler wordt in een randomly generated dungeon gespawnt, en moet de sleutel tot de uitgang vinden. Als de sleutel wordt opgepakt, wordt het exit-portaal zichtbaar, en kan de speler deze gebruiken. Onderweg komt de speler waarschijnlijk vijanden tegen die de speler door op ze te schieten kan verslaan.

TOELICHTING OP DE UML?
De AI heeft een MonoBehaviour genaamd EnemyAI. Deze heeft een Array met conditions die erven van ConditionBase ge-assigned in de inspector, en itereert hier doorheen. Elke condition heeft een associatedAction die erft van ActionBase die wordt ge-assigned in de inspector. Na de iteratie door de conditions, wordt de associatedAction met de hoogste prioriteit-waarde, die in de inspector wordt ge-assigned, uitgevoerd. In de actions wordt simpel gedrag geprogrammeerd zoals “loop naar target” of “schiet op target”, en worden vervolgens dus uitgevoerd op basis van prioriteit. Bij een actie als schieten wordt er door de AI een raycast naar de speler gestuurd, en wordt vervolgens op de HealthSystem van de speler een TakeDamage-functie uitgevoerd. Als de AI een andere AI schiet, zal dit op basis van de code in de zelfgeschreven action bepalen of het ook de AI damaged. Dit kan je doen door te checken op de CreatureType in de health-system, en alleen TakeDamage uit te voeren als de geraakte collider de juiste CreatureType heeft. De GameManager subscribed tijdens het spawnen een functie aan een UnityAction genaamd OnDeath waarop de GameManager op basis van wat voor een creature gedood is een actie uit kan voeren, zonder constant een reference naar deze HealthSystems te hoeven hebben.

WELKE DESIGN PATTERNS OF ANDERE PROGRAMMEERSTUFF HEB IK GEBRUIKT?
Ik heb in deze game een singleton gebruikt voor de GameManager, en enumerators om op basis van de waardes van de enums andere acties uit te voeren. Ook heb ik voor het eerst UnityActions / een event systeem gebruikt, en heb ik de AI op een modulaire een simpel uitbreidbare manier opgezet. De AI is hierdoor niet een harde state-machine, maar meer condition en priority-based, waardoor de AI met weinig moeite veel verschillende gedrag kan vertonen. Ook heb ik mijn eigen pathfindingsysteem geschreven, hoewel die wat moeilijker zal zijn om aan te passen aan andere types games, maar daar schrijf ik later nog wat over.

WAT HAD IK LIEVER ANDERS GEDAAN?
Zoals ik al zei, is het pathfindingsysteem niet goed toe te passen op een aantal andere genres van games. Dit komt omdat hij moeite heeft met concave areas als de speler in een ander gebied zit. Dat maakt in een room-based game als deze niet uit omdat de AI pas activeert als de speler dichtbij genoeg is, maar in grotere gebieden zou dit voor problemen kunnen zorgen. Daarnaast zou ik in een volgende game een paar aanpassingen hebben gemaakt aan de manier waarop ik UnityActions gebruik, omdat ik qua structuur hierin volgensmij een beetje de kracht van events heb vermeden. Ook had ik graag wat meer tijd in de aankleding van alles willen stoppen, want het is nu technisch gezien een spel wat je kan spelen, maar zelfs een healthbar zit er niet in. Dit kwam echt door tijdsdruk vanwege andere herkansingen, waardoor ik meer heb gefocust op de structuur.

ARTIFICIAL INTELLIGENCE

HOE WERKT MIJN AI-SYSTEEM?

Mijn AI-systeem werkt modulair. Je hebt losse MonoBehaviours die erven en overriden van ConditionBase en ActionBase. In de workflow kan je in losse MonoBehaviours *conditions* en *actions* programmeren. Elke *condition* heeft een veld met *associatedAction*, waar je de gewenste *action* na de *condition* insleept. Er wordt door het script EnemyAI door elke condition geloopt, gekeken of hij *true* returnt, en voert vervolgens de actie uit met de hoogste priority. In het kort:

Elke ***condition*** returnt een boolean en heeft een reference aan een ***action***

en

Elke ***action*** voert het gedrag uit.

WAT ZIJN MIJN TWEE AI-TYPES?

De twee enemy-types in mijn game zijn de *boomer* en de *ranger*. Beide enemy-types hebben dezelfde *conditions* maar hebben andere *actions*. Het zijn hierdoor wel compleet verschillende enemies geworden, waar ik best blij mee ben. De actions/behaviours zijn als volgt:

Boomer

- Rent dichter naar de speler toe.
- Explodeert wanneer hij dicht genoeg bij de speler is, en doodt de speler.
- Explodeert na een aantal seconden als de levenspunten onder een bepaald niveau zitten. Dit kan de speler ook doodden als deze dichtbij genoeg is.

Ranger

- Rent dichter naar de speler toe en schiet ondertussen.
- Als de speler dichtbij genoeg is, stopt de enemy met dichterbij komen, maar blijft schieten.
- Als de levenspunten onder een bepaalt niveau zijn, zal de enemy zichzelf na een aantal seconden healen.

PATHFINDING

Ik heb in mijn game een custom pathfinding-systeem geïnspireerd door A* maar het werkt met concave kamers in een aantal gevallen niet, maar die gevallen komen niet voor in mijn game. Een agent kijkt welke aanliggende node/room het dichtst bij de speler is en het dichtst bij de agent zelf.

DUNGEON GENERATION

HOE WERKT MIJN DUNGEON GENERATOR?

In mijn dungeon generator worden eerst de rooms op willekeurige posities geïnitialiseerd en krijgen een RoomType mee die aangeeft wat de GameManager tijdens de initialisatie in deze kamers moet spawnen. Vervolgens wordt door alle rooms ge-ïterate en vervolgens een hallway naar de volgende kamer gemaakt. Dan wordt de pathfinding gespawnd, en als al die hiervoor staande informatie klaarstaat, wordt er na het spawnen van de geometry van de rooms en hallways gecheckt op overgangen tussen rooms en hallways en vervolgens daarop een deur gespawnt.

HOE WORDT DIE DUNGEON SPEELBAAR GEMAAKT?

De GameManager heeft een reference aan de DungeonGenerator, en gebruikt de RoomTypes van de rooms om te kijken wat er moet worden gespawnt. RoomType is een enumerator, wat het makkelijk maakt om de volgorde waarin gameplay-elementen zoals de player en enemy spawn en de key en de exit worden gegenereerd te veranderen. In elke kamer wordt de bijpassende prefab gespawnd, en subscribed de GameManager zich aan bepaalde UnityActions van deze gameplay-elementen. Zo kan de GameManager makkelijk wanneer de speler de sleutel oppakt de uitgang zichtbaar maken. Daarnaast kan de GameManager zo als de speler dood gaat de AI van de enemies uitzetten, en de scene opnieuw laden.