Dogtag Documentation of Stuff implemented by Jan

AI-System:

How it basically Works:

- For the AI a Finit State Machine (FSM, ‚EnemyStateMachine.cs‘) is used that basically switches between different States on certain conditions
    - The basic States are:
        - Idle ('IdleState.cs')
        - Alert ('AlertState.cs')
        - Chase ('ChaseState.cs')
        - Attack ('AttackState.cs')
        - All those States derive from a Base State ('BaseState.cs') that provides the shared logic

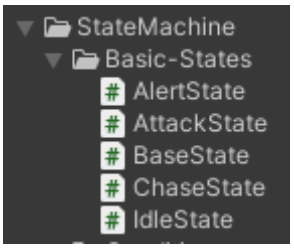
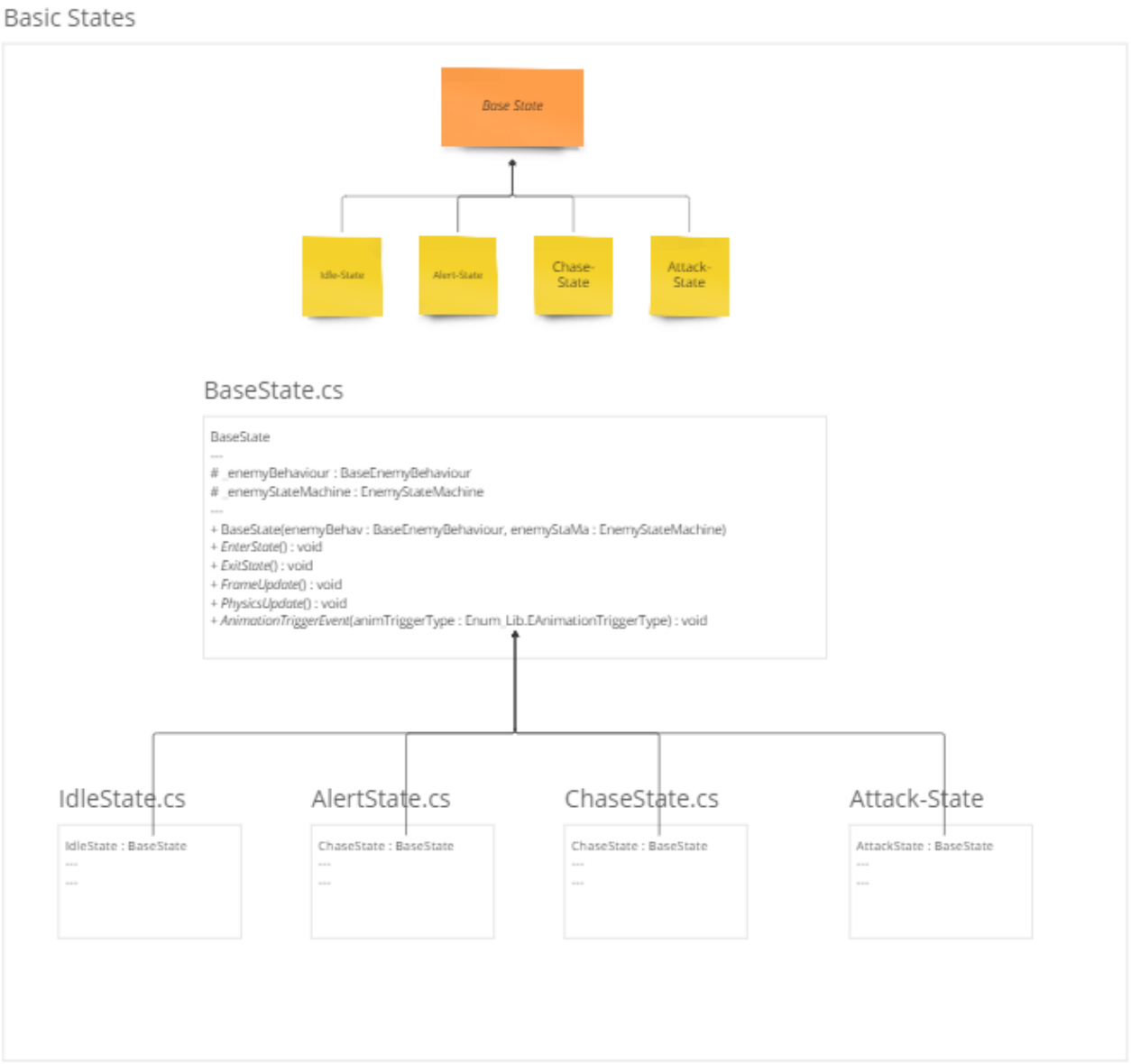


Fig. 1: *Base States*

Fig 2: *FSM that handels the Basic swapping between different States*

Fig. 3.


Basic States
Base State
Idle-State
Alert-State
Chase-State
Attack-State
BaseState.cs
BaseState

_enemyBehaviour : BaseEnemyBehaviour
_enemyStateMachine : EnemyStateMachine

+ BaseState(enemyBehav : BaseEnemyBehaviour, enemyStaMa : EnemyStateMachine)
+ EnterState() : void
+ ExitState() : void
+ FrameUpdate() : void
+ PhysicsUpdate() : void
+ AnimationTriggerEvent(animTriggerType : Enum_Lib.EAnimationTriggerType) : void
IdleState.cs
IdleState : BaseState
AlertState.cs
ChaseState : BaseState
ChaseState.cs
ChaseState : BaseState
Attack-State
AttackState : BaseState

Fig. 4:

- Every Enemy Type derives from the ‚BaseEnemy.cs' and implements the ‚EnemyStateMachine.cs' and therefore the specific AI-behaviour

Fig. 5:

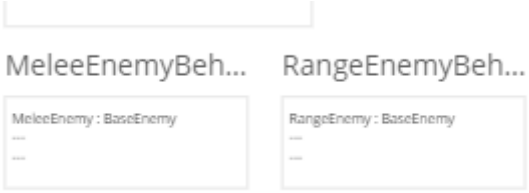


Fig 6:

- Also The System is additionally build up modular by using Scriptable Objects that actually implements the specific behaviour for the Basic States and need to be set to the specific Enemy Types.

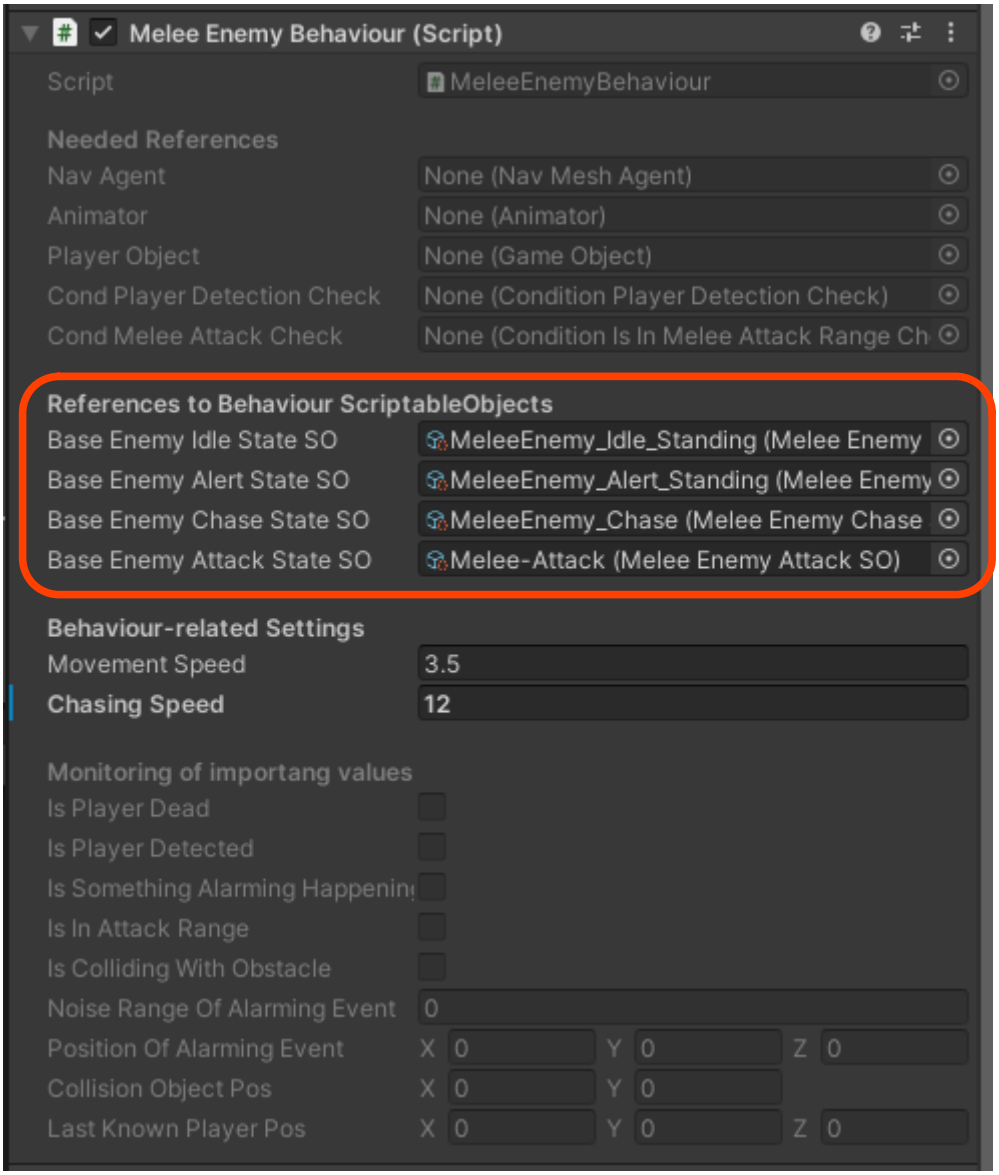


Fig. 7: *Setup of a Melee Enemy Type with its specific Behaviour States (Scriptable Objects) that differ from the ones for the Range Enemy Type. Will be explained in more detail a bit later.*

- Every Enemy Object additionally implements Specific ConditionChecks for the FSM to actually work
    - At this point for example for the Melee Enemy Type there is a concitionCheck for if the PlayerObj. Is detected and if the Enemy is in Attack Range.
        - If the Player is detected the Enemy will change from any state to the ‚ChaseState.cs‘
        - If the Enemy is in Attack Rage it will be switched to the ‚AttackState.cs’

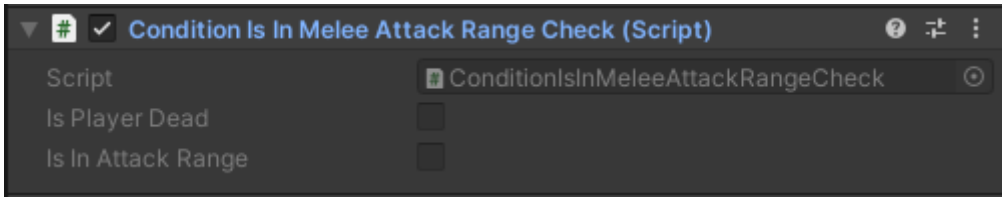


Fig. 8: *The 'ConditionIsInMeleeAttackRangeCheck'. This one checks if the Enemy is in Attack Range to the PlayerObj. And fires an according event if so with a according boolian value that determines if it shall be switched to the ‚AttackState.cs’.*

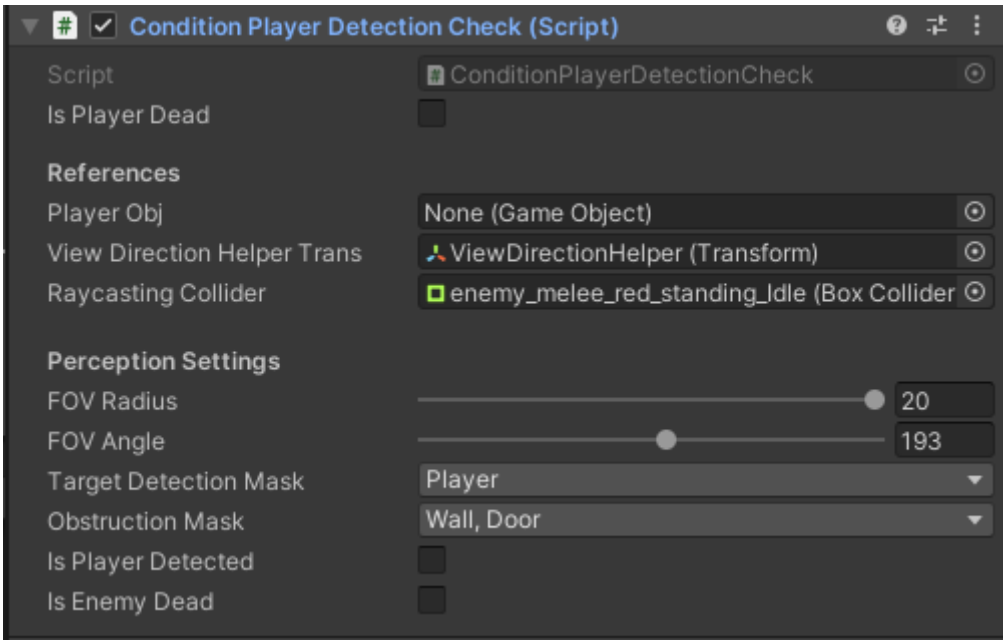


Fig. 9: *The ConditionPlayerDetectionCheck returns a boolian value on if the Player is detected or not. It works as aa kind of Field of View by wich Obstacles can avoid Player detection if they are between the enemy and player.*

- As shown in the Figure 7, The Specific AI-Behaviour is defined in Scriptable Objects (Behaviour ScriptableObjects).
    - These ‚BehaviourScriptableObjects‘ provide all the specific Logic for a certain Behaviour. And will be called when the actual State (e.g. Idle) will be called in the ‚BaseEnemyBehaviour.cs’ (see Fig. 5)
    - The specific Behaviour Scriptable Objects derive from the according BaseStatesSO (e.g. ‚BaseEnemyIdleStateSO.cs’)
        - therefore the specific Enemybehaviour can be designed pretty versatile and modular

Fig. 10: *Setting up of the Behaviour of an Melee Enemy Type by assigning the specific Behaviour Scriptable Objetcs to the according states.*

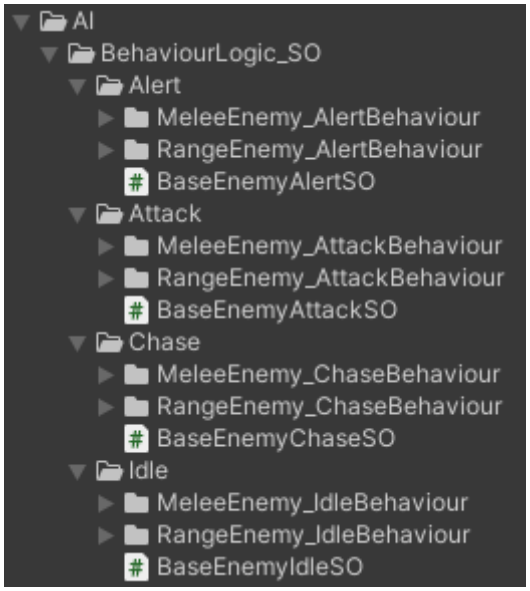


Fig. 11: *The BehaviourScriptable Objects that so far that implement the Logic for the specific behaviour.*

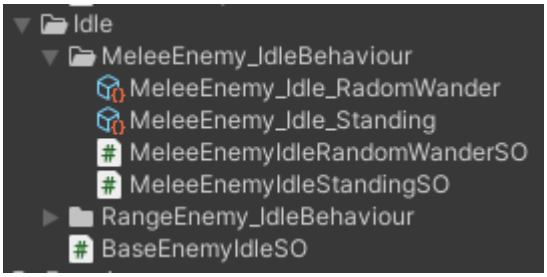


Fig 12: *The Idle Behaviour for the MeleeEnemy as example. Here we have two different Idle-Behaviours in seperate Scriptable Objects defined, a RandomWander Behaviour and a Standing behaviour. That appropriately can be assigned to the specific State on the MeleeEnemy Type.*

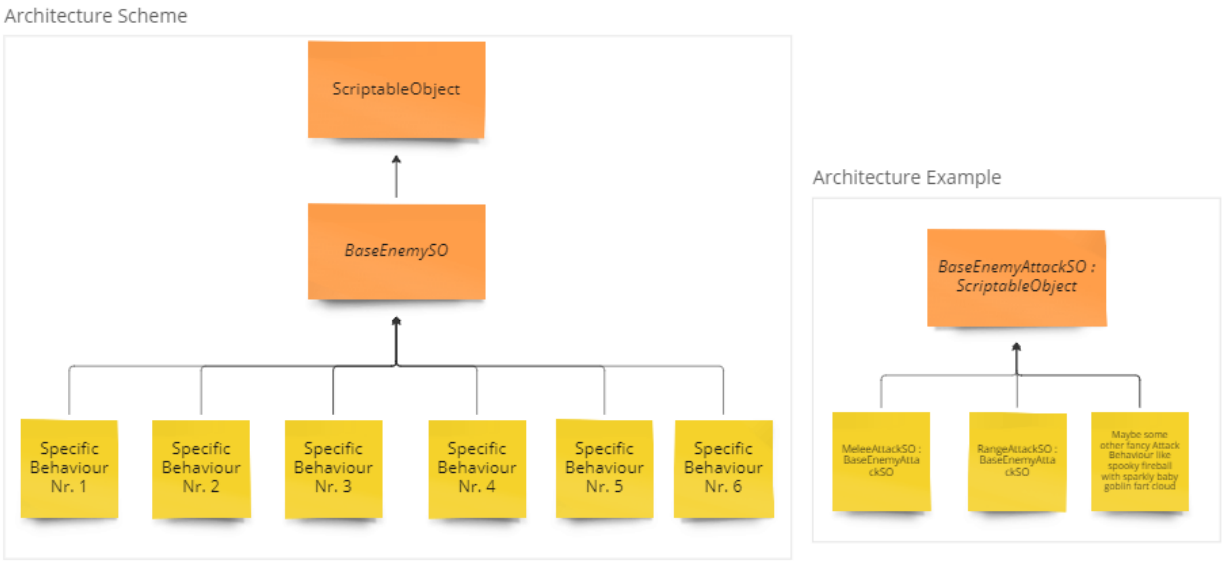


Fig. 13: Scheme of the Behaviour Scriptable Objects

Fig 14: The BehaviourScriptableObject that implements the actual Idle.Behaviour