Advanced AI

This page describes how to setup advanced AI to create more interesting enemies (or friends).

Introduction

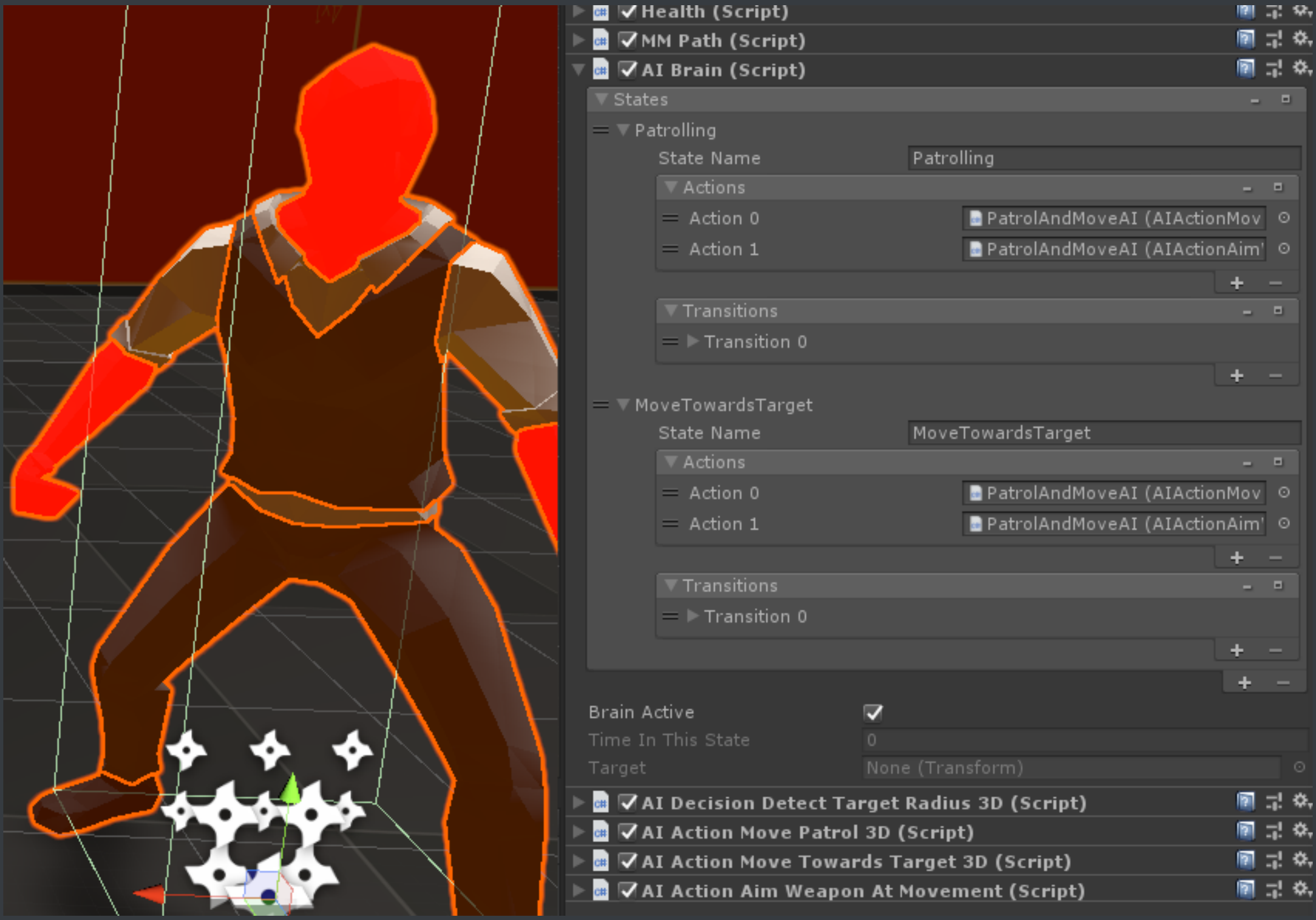
The TopDown Engine provides an advanced AI system that will help you create all sorts of interesting patterns and attitudes. For a brief overview of what you can do with it, you can have a look at the MinimalAI demo scene, which will showcase a tiny portion of the things you can create with this system.

Core concepts

The Advanced AI system is based on a few core classes you'll find in the MMTools/AI/ folder :

- **AIState** : A State is a combination of one or more actions, and one or more transitions. An example of a state could be "*patrolling until an enemy gets in range*".
- **AIAction** : Actions are behaviours and describe what your character is doing. Examples include patrolling, shooting, jumping, etc. The engine comes with a lot of predefined actions, and it's very easy to create your own.
- **AIDecision** : Decisions are components that will be evaluated by transitions, every frame, and will return true or false. Examples include time spent in a state, distance to a target, or object detection within an area.

- **AITransition** : Transitions are a combination of one or more decisions and destination states whether or not these transitions are true or false. An example of a transition could be "*if an enemy gets in range, transition to the Shooting state*".
- **AIBrain** : the AI brain is responsible from going from one state to the other based on the defined transitions. It's basically just a collection of states, and it's where you'll link all the actions, decisions, states and transitions together.



```
Health (Script)
MM Path (Script)
AI Brain (Script)
  States
    Patrolling
      State Name          Patrolling
      Actions
        Action 0          PatrolAndMoveAI (AIActionMov
        Action 1          PatrolAndMoveAI (AIActionAim'
      Transitions
        Transition 0
    MoveTowardsTarget
      State Name          MoveTowardsTarget
      Actions
        Action 0          PatrolAndMoveAI (AIActionMov
        Action 1          PatrolAndMoveAI (AIActionAim'
      Transitions
        Transition 0
  Brain Active            ✓
  Time In This State      0
  Target                  None (Transform)
AI Decision Detect Target Radius 3D (Script)
AI Action Move Patrol 3D (Script)
AI Action Move Towards Target 3D (Script)
AI Action Aim Weapon At Movement (Script)
```

An example of an Advanced AI powered character's inspector, complete with its brain, three actions and one decision.

All of these will be put on top of an already setup character (check out the dedicated doc for more on this) and can potentially leverage all existing abilities.

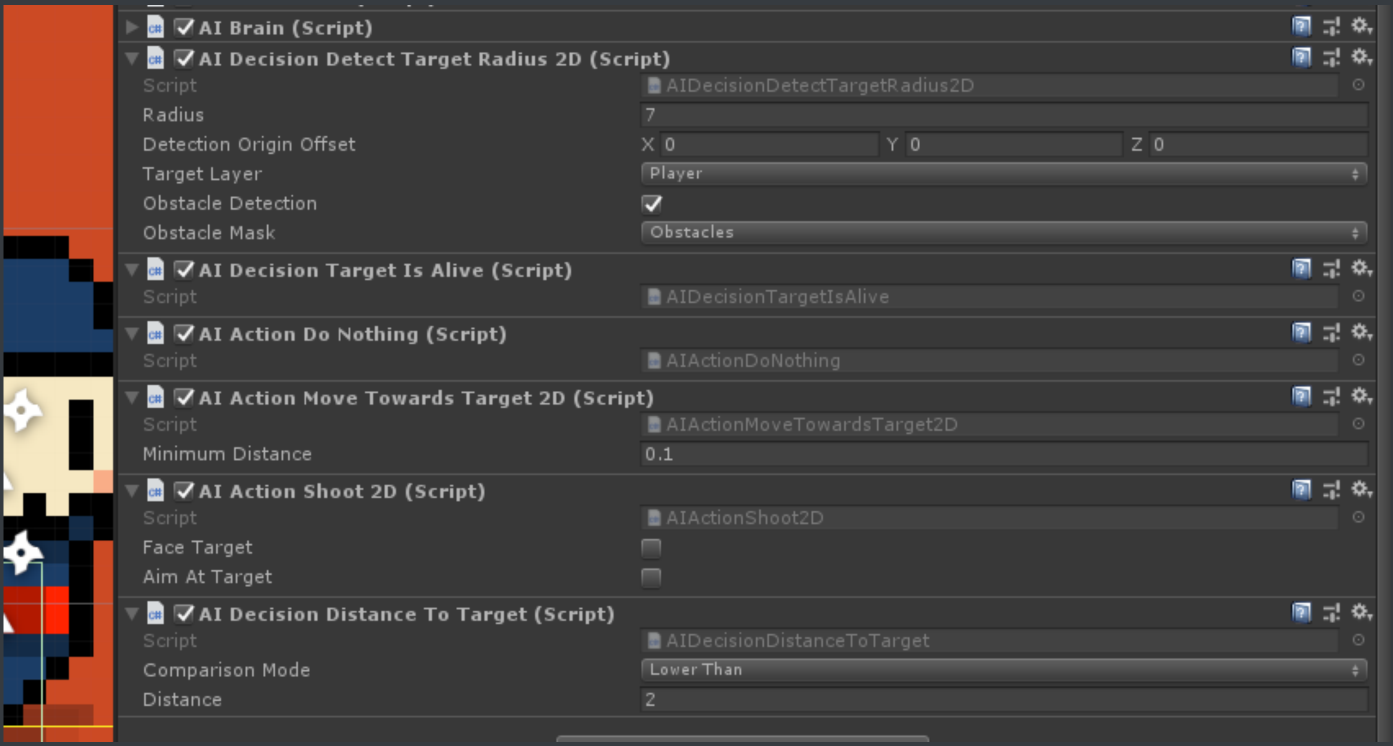
Creating an advanced AI character

We'll recreate one of the example characters available in the KoalaDungeon demo scene, the NinjaSwordMaster. Creating the base character is no different from creating any character, so we won't go over that again here. Once that base character is ready, you'll want to add an AIBrain to it.

What we want from this character is the following behaviour : do nothing until an enemy gets close enough, and if that's the case move towards it, and attack if close enough. Then if the target gets out of range, get back to doing nothing and waiting for a target.

To get there, we'll use some of the ready made actions and decisions available in the engine. We'll add the following components : AIDecisionDetectTargetRadius2D, AIDecisionTargetIsAlive, AIActionDoNothing, AIActionMoveTowardsTarget2D, AIActionShoot2D, AIDecisionDistanceToTarget. Each of these comes with an inspector where we'll need to set a few things :

- **AIDecisionDetectTargetRadius2D** : we define our radius (7 in the example), our target layer should be Player, and we want to use the Obstacles layer as our Obstacle mask.
- **AIDecisionTargetIsAlive** : we'll use that to make sure our target is still alive. No setup is required
- **AIActionDoNothing** : no setup required here
- **AIActionMoveTowardsTarget2D** : we'll set the minimum distance to 0.1, that's the distance to the target at which the character will consider its journey complete
- **AIActionShoot2D** : everything should be unchecked
- **AIDecisionDistanceToTarget** : we want to have this decision return true if the distance is lower than 2, so we set our comparison mode to "lower than", and our distance to 2.



AI Brain (Script)

AI Decision Detect Target Radius 2D (Script)

Script	AIDecisionDetectTargetRadius2D
Radius	7
Detection Origin Offset	X 0 Y 0 Z 0
Target Layer	Player
Obstacle Detection	✓
Obstacle Mask	Obstacles

AI Decision Target Is Alive (Script)

Script	AIDecisionTargetIsAlive

AI Action Do Nothing (Script)

Script	AIActionDoNothing

AI Action Move Towards Target 2D (Script)

Script	AIActionMoveTowardsTarget2D
Minimum Distance	0.1

AI Action Shoot 2D (Script)

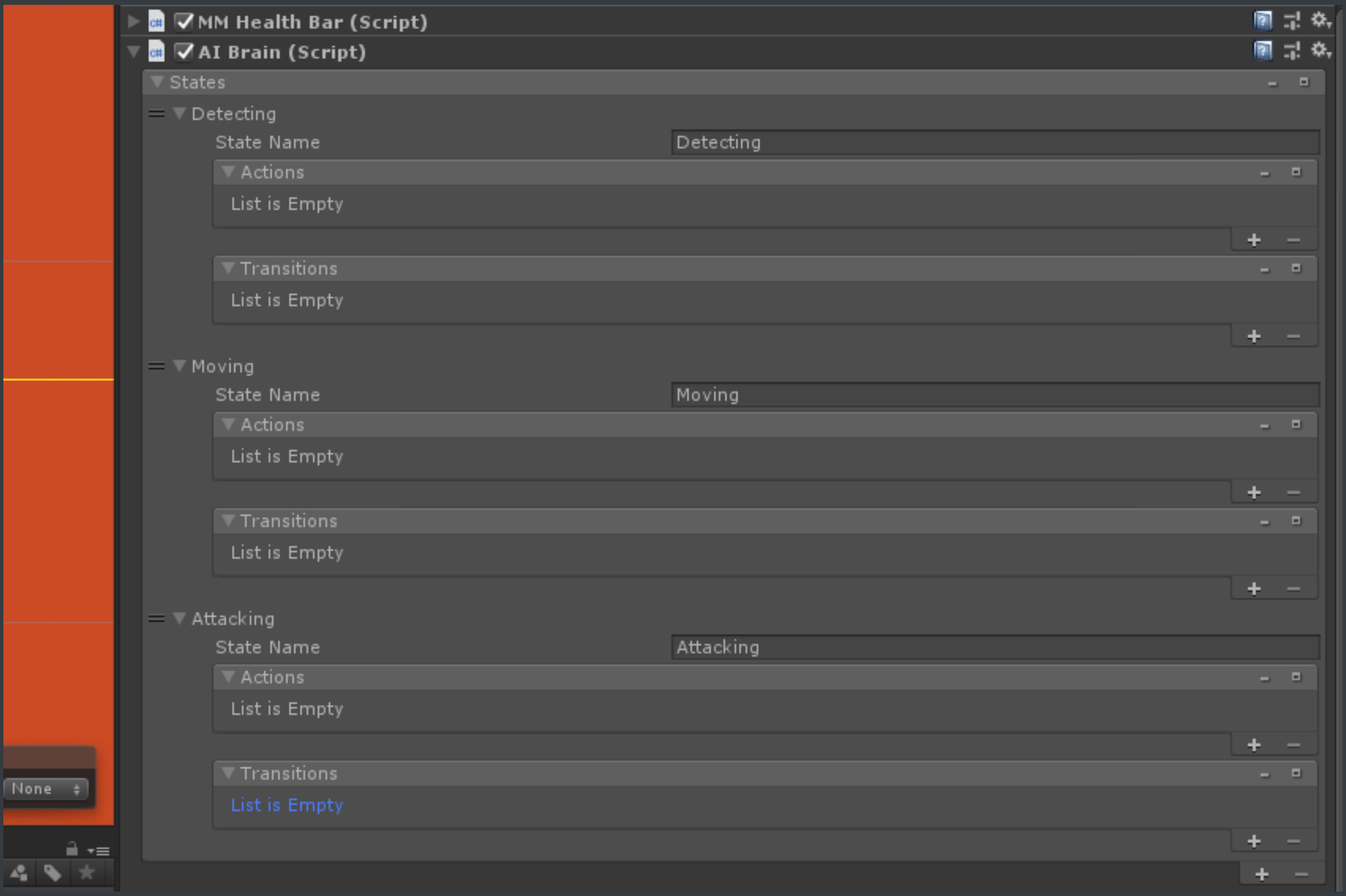
Script	AIActionShoot2D
Face Target	☐
Aim At Target	☐

AI Decision Distance To Target (Script)

Script	AIDecisionDistanceToTarget
Comparison Mode	Lower Than
Distance	2

Our character's inspector.

Our character now has all the actions and decisions we need to define its behaviour. All that's left to do is plug all that into its brain. As stated before, our character will have 2 states : patrolling and attacking. So in its AIBrain component, we'll simply add three states by pressing the bottom right "+" sign three times. We'll name our first state "Detecting", our second state "Moving", and our third state "Attacking".



MM Health Bar (Script)

AI Brain (Script)

States

Detecting

State Name: Detecting

Actions

List is Empty

Transitions

List is Empty

Moving

State Name: Moving

Actions

List is Empty

Transitions

List is Empty

Attacking

State Name: Attacking

Actions

List is Empty

Transitions

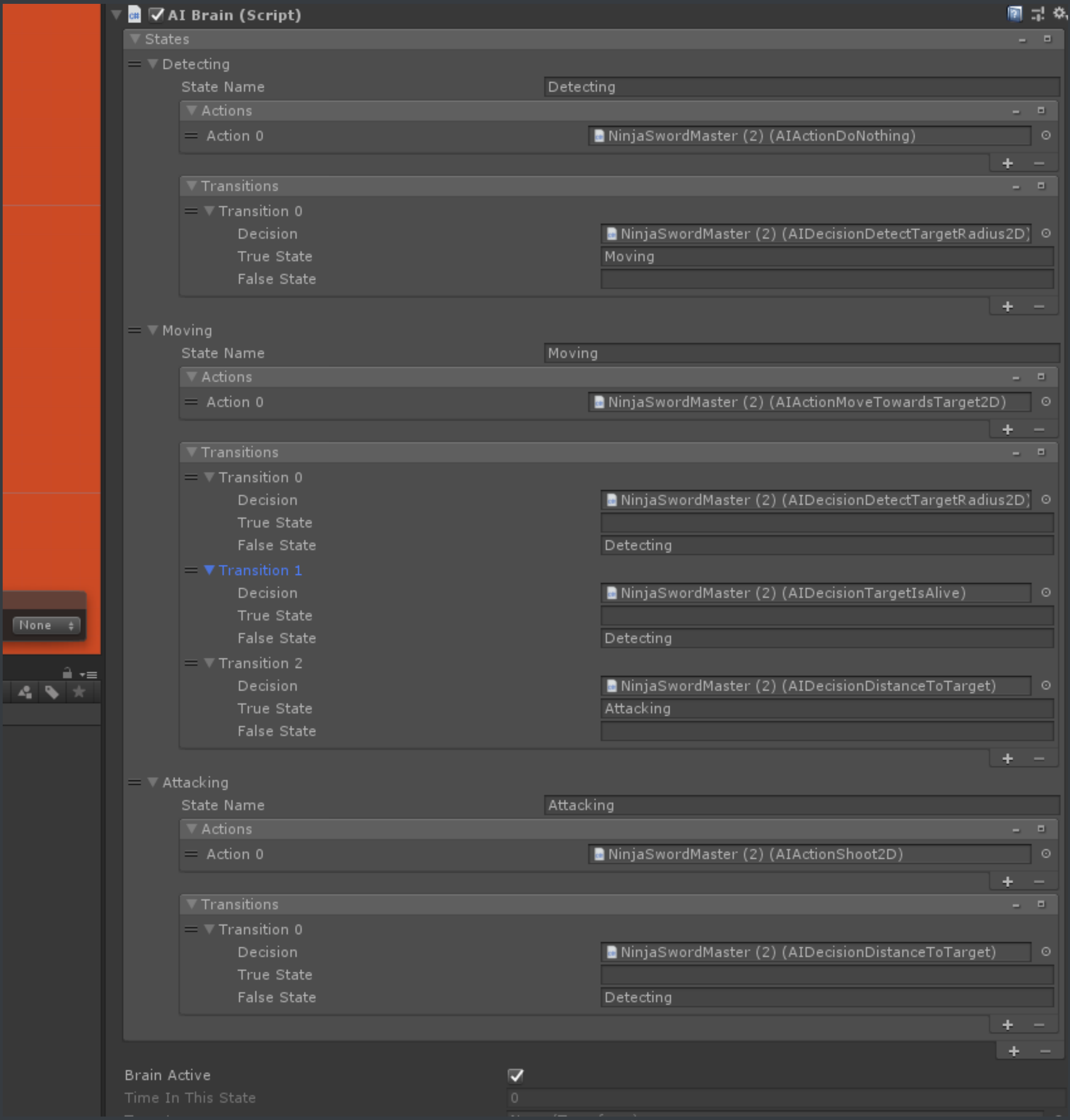
List is Empty

Naming our states.

Now in the Detecting state we'll only have one action, and one transition, so we'll add an Action and a Decision by pressing their respective "+" signs once each. We'll now drag our character's AIActionDoNothing component into the Actions' array slot, and we'll drag the AIDecisionDetectTargetRadius2D into the Transitions' first Decision slot. Note that from v1.10 onwards, you can also select that action from a dropdown. We want to switch to moving when the player is in range, so we'll put "Moving" in our transition's *True State* field.

We'll proceed in the same way for our second state : we'll add an action and, this time, three transitions, then drag our AIActionMoveTowardsTarget2D component into our Moving state's Action slot, and then move our AIDecisionDetectTargetRadius2D into the first transition's detection slot, AIDecisionTargetIsAlive into the second transition's slot, and finally AIDecisionDistanceToTarget in the 3rd transition's slot. When the target is not in range anymore we want to go back to Detecting, so we'll put Detecting in our first transition's false state, and we also want to go back to detecting when the target is dead, so we'll put Detecting in our second transition's false state slot. And if the distance to target is low enough, we want to attack, so we'll put Attacking in our 3rd transition's true state slot.

In our last state, Attacking, we'll add one action and one transition. Then we want to drag our AIActionShoot2D in the Action slot, and AIDecisionDistanceToTarget into the transition's decision slot. If that distance is too high, we want to go back to Detecting, so we'll put Detecting in the false state slot.



```
AI Brain (Script)
States
  Detecting
    State Name      Detecting
    Actions
      Action 0      NinjaSwordMaster (2) (AIActionDoNothing)
    Transitions
      Transition 0
        Decision    NinjaSwordMaster (2) (AIDecisionDetectTargetRadius2D)
        True State  Moving
        False State
  Moving
    State Name      Moving
    Actions
      Action 0      NinjaSwordMaster (2) (AIActionMoveTowardsTarget2D)
    Transitions
      Transition 0
        Decision    NinjaSwordMaster (2) (AIDecisionDetectTargetRadius2D)
        True State
        False State Detecting
      Transition 1
        Decision    NinjaSwordMaster (2) (AIDecisionTargetIsAlive)
        True State
        False State Detecting
      Transition 2
        Decision    NinjaSwordMaster (2) (AIDecisionDistanceToTarget)
        True State  Attacking
        False State
  Attacking
    State Name      Attacking
    Actions
      Action 0      NinjaSwordMaster (2) (AIActionShoot2D)
    Transitions
      Transition 0
        Decision    NinjaSwordMaster (2) (AIDecisionDistanceToTarget)
        True State
        False State Detecting
Brain Active        ✓
Time In This State  0
```

Our brain all set up!

To make sure our brain is properly setup, we can just "read" it. From top to bottom, we have a character that is doing nothing, while waiting until a target gets within its radius. Then it starts to move, unless the target is dead or too far / out of sight. If, however, the target is close enough, it will attack, and go back to detecting if the target gets far enough.

Congratulations, you've completed what is probably the "hardest" thing to do with the TopDown Engine, and you now know how to create all sorts of character behaviours. You can of course have more than one action in a state (running and jumping for example) and more than one transition in a state, so you can really come up with complex stuff.

Actions

The engine comes with all of these predefined actions, ready to use in your own characters :

- **AIActionAimObject** : will let you aim an object's axis (usually right or forward) at either the character's movement direction or its aim direction.
- **AIActionAimWeaponAtMovement** : will force the weapon aim to the movement's direction.
- **AIActionChangeWeapon** : used to force your character to switch to another weapon. Just drag a weapon prefab into its NewWeapon slot and you're good to go.
- **AIActionCrouchStart** : makes your character crouch
- **AIActionCrouchStop** : makes your character stop crouching
- **AIActionDoNothing** : as the name implies, does nothing. Just waits there.
- **AIActionMMFeedbacks** : lets you play any MMFeedbacks you bind into its inspector
- **AIActionFaceTowardsTarget2D** : this AI action that lets you change the CharacterOrientation2D's facing direction of an AI, and an example of it in action in the KoalaDungeon demo scene
- **AIActionMoveAwayFromTarget2D/3D** : makes the character move in the direction opposite to the target
- **AIActionMovePatrol2D/3D** : will have your character patrol through a set of nodes defined in an MMPath component. Make sure you set your MMPath's cycle option to Loop or BackAndForth (you can't patrol along a path in OnlyOnce mode, it has to be continuous).
- **AIActionInvertPatrolDirection** : lets you invert the direction of a target 2D or 3D patrol
- **AIActionMoveRandomly2D/3D** : makes the character move randomly, until it finds an

obstacle in its path, in which case it'll pick a new direction at random

- **AIActionMoveRandomlyGrid** : moves randomly on a grid, whether a 2D or a 3D one
- **AIActionMoveTowardsTarget2D/3D** : directs the CharacterHorizontalMovement ability to move in the direction of the target.
- **AIActionPathfinderToPatrol3D** : will use the Pathfinding3D ability to move back to its last patrol point
- **AIActionPathfinderToTarget3D** : will use the Pathfinding3D ability to move to the target if a path can be found
- **AIActionReload** : causes the agent to reload its current weapon
- **AIActionRotateConeOfVision2D** : will rotate this AI's ConeOfVision2D either towards the AI's movement or its weapon aim direction
- **AIActionRotateTowardsTarget2D/3D** : will make the CharacterOrientation3D ability rotate the character towards the Brain's target
- **AIActionRunStart** : will cause the agent to start running (requires a CharacterRun ability)
- **AIActionRunStop** : will cause the agent to stop running (requires a CharacterRun ability)
- **AIActionSetLastKnownPositionAsTarget** : will set the last known position of the target as the new Target
- **AIActionShoot2D/3D** : shoots using the currently equipped weapon. If your weapon is in auto mode, will shoot until you exit the state, and will only shoot once in SemiAuto mode. You can optionnally have the character face (left/right) the target, and aim at it (if the weapon has a WeaponAim component), or pick various points of origin to define how aim direction will be computed.
- **AIActionSwapBrain** : lets you brain a Character's current Brain for another one. Useful for things like boss phases, for example
- **AIActionUnityEvents** : lets you trigger any Unity event you bind into its inspector

Just like everything else in the engine, you're encouraged to create your own. Let's look at an Action's code to see how it works :

```
public class AIActionJump : AIAction
{
    public int NumberOfJumps = 1;
    protected CharacterJump _characterJump;
```

```
        protected int _numberOfJumps = 0;

        protected override void Initialization()
        {
            _characterJump = this.gameObject.GetComponent<CharacterJump>();
        }

        public override void PerformAction()
        {
            Jump();
        }

        protected virtual void Jump()
        {
            if (_numberOfJumps < NumberOfJumps)
            {
                _characterJump.JumpStart();
                _numberOfJumps++;
            }
        }

        public override void OnEnterState()
        {
            base.OnEnterState();
            _numberOfJumps = 0;
        }
    }
```

As you can see this class overrides a few methods :

- **Initialization**, in which we'll do whatever we need to do to initialize our Action, in this case we store the CharacterJump ability for future use.
- **PerformAction** : this will be called everytime our character is in the state this Action is in. In this case we'll just call our Jump method, that in turns calls the CharacterJump ability's JumpStart method if our conditions are met.

- **OnEnterState** : everytime we'll go back to the state that contains this Action, we'll want to reset our current number of jumps.

Decisions

- **AIDecisionDetectTargetConeOfVision2D/3D** : uses a field of view to determine whether or not a target is in view. Returns true if that's the case.
- **AIDecisionDetectTargetLine2D** : will return true if any object on its TargetLayer layermask enters its line of sight. It will also set the Brain's Target to that object. You can choose to have it in ray mode, in which case its line of sight will be an actual line (a raycast), or have it be wider (in which case it'll use a spherecast). You can also specify an offset for the ray's origin, and an obstacle layer mask that will block it.
- **AIDecisionDetectTargetLine3D** : lets you detect targets using raycasts or boxcasts in line in a 3D context
- **AIDecisionTargetRadius2D/3D** : will return true if an object on its TargetLayer layermask is within its specified radius, false otherwise. It will also set the Brain's Target to that object.
- **AIDecisionDistanceToTarget** : will return true if the current Brain's Target is within the specified range, false otherwise.
- **AIDecisionGrounded** : will return true if the character is grounded, false otherwise.
- **AIDecisionHealth** : will return true if the specified Health conditions are met. You can have it be lower, strictly lower, equal, higher or strictly higher than the specified value.
- **AIDecisionHit** : returns true if the Character got hit this frame, or after the specified number of hits has been reached.
- **AIDecisionLineOfSightToTarget2D/3D** : will return true if a line can be traced to the target from the specified offset and not hitting the specified layers
- **AIDecisionNextFrame** : will return true when entering the state this Decision is on.
- **AIDecisionRandom** : will roll a dice and return true if the result is below or equal the odds value specified in its inspector
- **AIDecisionReloadNeeded** : returns true if a reload is needed at the frame it's evaluated at
- **AIDecisionTargetFacingAI2D** : will return true if the Brain's current target is facing this character. Yes, it's quite specific to Ghosts. But hey now you can use it too!
- **AIDecisionTargetIsAlive** : will return true if the target is alive
- **AIDecisionTargetIsNull** : will return true if the target is null
- **AIDecisionTimeInState** : will return true after the specified duration (in seconds) has

passed since the Brain has been in the state this decision is in. Use it to do something X seconds after having done something else.
- **AIDecisionTimeSinceStart** : will return true after the specified duration (in seconds) has passed since the level was loaded.

And just like Actions, it's very easy to create your own Decisions.

Organizing AI components

When building complex characters, your Character's inspector list of components can get pretty **massive**, between the agent's core classes (Character, TopDownController, etc), the abilities, and the brain, actions and decisions. Fortunately, v1.10 introduced the ability to **split** these abilities across multiple nodes, and that's also true for the brain, actions and decisions.

You can see an example of a Character setup that way in the Loft3D demo scene, with the **PatrolSeekAndDestroyAI** prefab. At its top level, you'll find its core classes. Then, on empty game objects nested under it, you'll find a node with its abilities (Abilities), and one with its brain (AIBrain). To push things further, if needed, the actions and decisions can also be split across more nodes. The only important thing about this setup is to make sure you've bound your Brain to the Character's inspector, in its **CharacterBrain** field. If you don't, it'll automatically look for one at its own level. And for abilities, you can learn more about it in the dedicated section of the documentation.