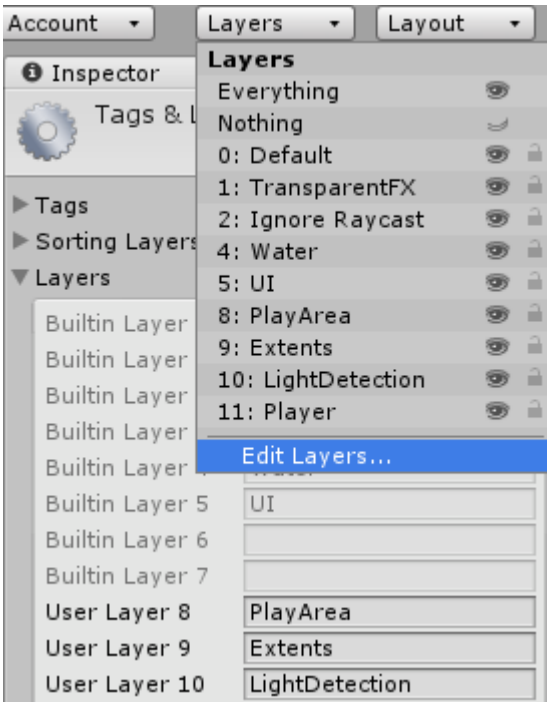
Green Mamba Stealth for Unity:

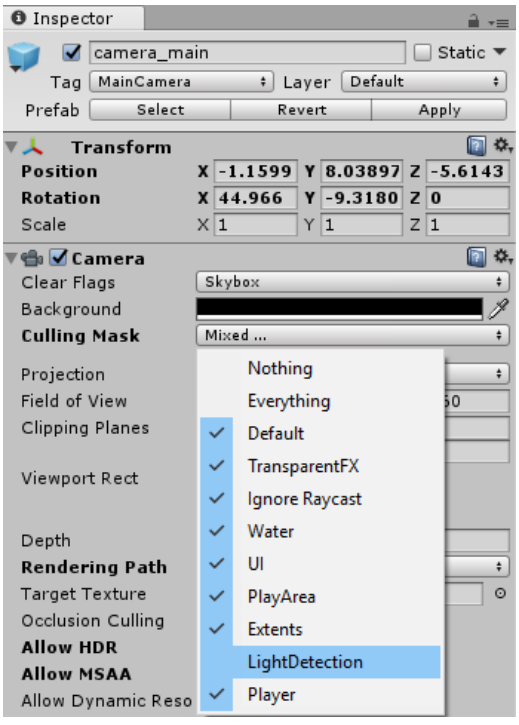
The main feature of this package is the ability to detect dynamic shadow interactions with the player. This means we can detect how much light is hitting the player and use this to drive stealth mechanics that are more realistic than the traditional method of simple raycasting. Rather than being limited to hiding behind objects, the player can literally hide in the shadows.

Setup and Integration:

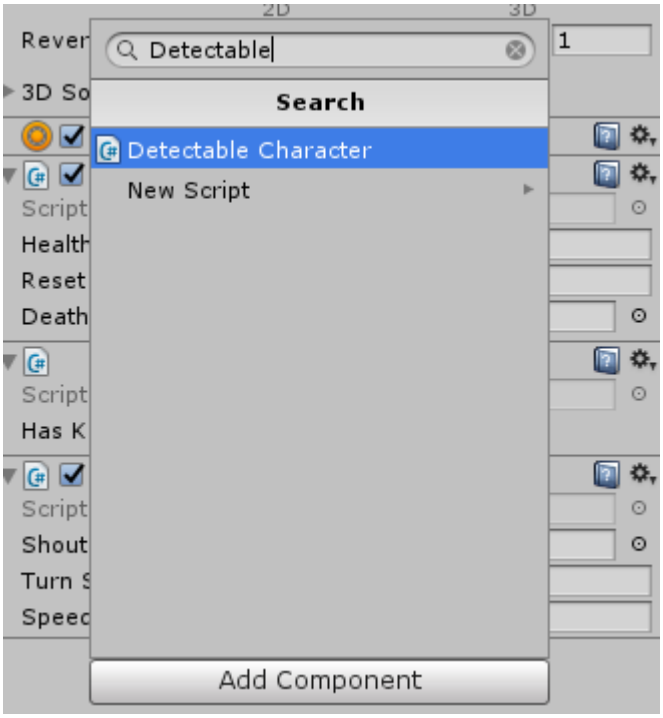
1) Add a layer to your project named LightDetection. This layer will be used to prevent cameras from rendering objects that are used for light detection.



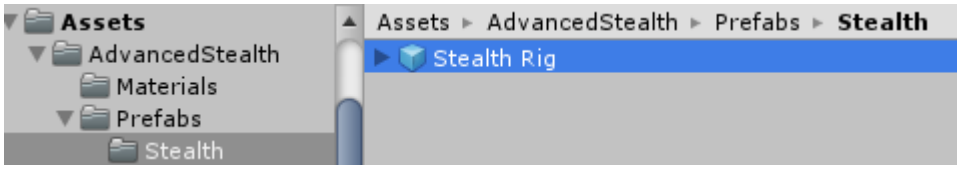
2) Select each camera that is used for gameplay rendering. Select the dropdown next to the "Culling Mask" field in the inspector for the Camera component, and uncheck the LightDetection Layer.



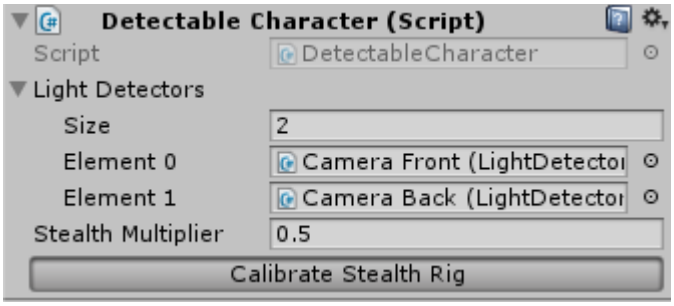
3) Select the root transform of each character you would like to use the stealth system for in the hierarchy. Click “Add Component” in the Inspector, and select the “Detectable Character” component. Do not press the “Calibrate Stealth Rig” button yet (but it’s okay if you already did).



4) Locate the “Stealth Rig” prefab, found in the AdvancedStealth/Prefabs/Stealth folder. Place this prefab as a child of each character you’ve added a Detectable Character component to. If your character has a skeleton, it’s best placed as a child of the Hip bone.



5) In the root of the character, where you added the DetectableCharacter component, press the “Calibrate Stealth Rig” button. This will automatically set up the rig for your character.



AI Integration:

For AIs that you want to be able to see DetectableCharacters, add the Sensors prefab as a child of their head, if they have a skeleton. If not, add it as a child of the character, and place it where their head is. Be sure that the Sensors object is facing forward, relative to the character. To be sure of this, you can select the object, be sure that the Tool Handle Rotation (in the upper left, to the right of the tool selection icons) is set to Local, not global, and ensure that the blue axis of the Tool Handle points in your character's forward direction.

To integrate the sensors into your own AI, UnityEvents are used. The Sensors component has two UnityEvents: ObjectVisibleHandler and ObjectInvisibleHandler. Your AI can subscribe to these events either through code or through the inspector. Your AI must have public methods that accept a single IDetectable argument for you to subscribe. These events are triggered after the AI has tested for layer, lighting, distance, and physical obstruction, so no further checks are necessary.

The Visibility Curve in the sensors may need adjustment to be suitable for your game. You can also use different curves to create characters with different perceptions; a character wearing sunglasses might not be able to spot a hidden character as easily as others. The X axis in this curve represents the distance between the two characters, and the Y axis represents a multiplier to the hidden player's stealth. The highest number on the X axis represents the furthest the character can see. In the default Visibility curve on the prefab, you can see that when the player is less than half the maximum distance away, their visibility increases greatly.

You can use the "Detection Mask" and "Invisible Tags" fields to limit AI detection based on layer and tag.

Usage Tips:

Tooltips:

Curious about what a field of a component does? Check the tooltip! Nearly everything has one.

Demo Project:

Included in this asset is a modified version of a project from an old Unity stealth tutorial. This project has been updated to incorporate this stealth system. The char_robotGuard objects have had the sensors prefab set up as a child of their Head in their skeleton, with the ObjectVisibleHandler and ObjectInvisibleHandler mapped to the AI. The player (char_ethan) has been set up as a detectable character, with the Stealth Rig as a child of their hips. When the scene runs, you can observe the dial in the GUI responding to the light affecting your character, and use the shadows to hide from the guards. You may safely delete the entire DemoProject folder after you no longer need it.

Stealth Display:

An on-screen display of a character's stealth rating helps the player maintain awareness of the tension of hiding. An example of a display is included in the AdvancedStealth/Prefabs/UI folder. Add the "Stealth Display Dial" prefab to your scene's GUI canvas, and set the Detectable field of the "Visibility Meter" to the character whose visibility you'd like to track.

Crouching/Sneaking Bonuses:

By default, this script doesn't take into account any bonuses for crouching and sneaking, since implementations of that behavior can vary greatly. Instead, the DetectableCharacter script has a property named StealthMultiplier. Visibility is multiplied by this number, and you can change it to represent bonuses. For example, adding 0.1 while crouching will give the player a small bonus to their stealth.

Lighting:

If you play your scene and the light is not finished baking, lights set to "Baked only" may not impact stealth. Be aware of problems like light bleeding; some difficult to spot lighting issues will affect stealth detection.

Debugging:
You can find a prefab in the AdvancedStealth/Prefabs/UI folder that will help you see what the scripts are seeing. Place this prefab in the scene as a child of your GUI canvas. Find the Stealth Rig of the character you would like to debug, and for each Light Detector in the Stealth Rig's children, assign the corresponding child of the Debug Display to the "Debug Render" field of the Light Detector. For example, assign "Camera Front Debug" to the Light Detector on "Camera Front".

You can also select the sensors of an AI and check the "Verbose" checkbox. This will display extensive information in the Console about the interactions of the sensors with detectables, including how much distance is affecting their vision.