Twin Stick Shooter Kit

Get Started

Setup

Twicks has been created for Unity 2021.3 LTS and the Built-In Pipeline.
The easiest way to get started with the package is to create a new project with Unity Hub using the 3D Core project template.

Once the project has been loaded you can import **Twicks** via the Package Manager

Troubleshooting

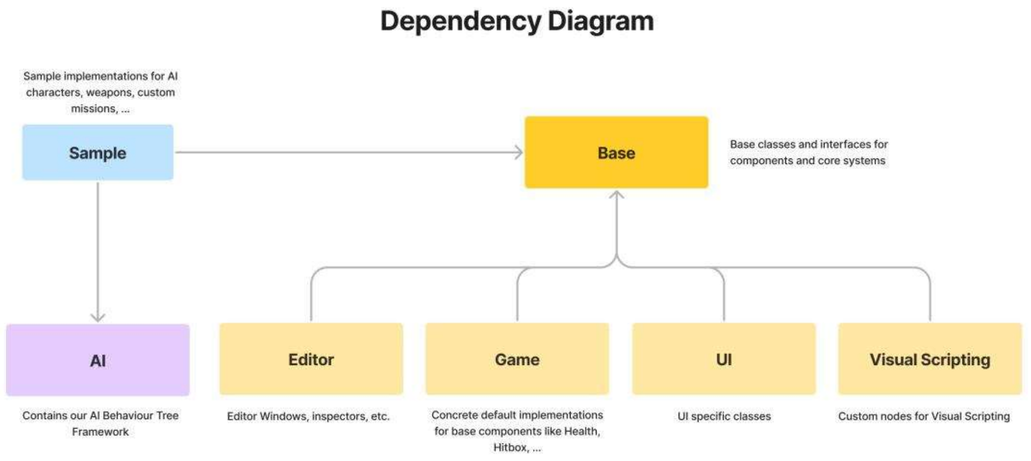
Depending on the version of Unity installed on your system, it might happen that problems occur when using the package, which are not related to the asset itself. Please check the following questions:

- **Can you build an empty project?**
  The simplest way to find out if your system is not configured properly to build a project, is to create a new empty project containing only a sample scene and trying to build it. If this is not possible, the problem is not related to the asset.
- **Can you build the project with Unity 2021.3?**
  The project has been developed for Unity 2021.3; therefore, we highly recommend using this version to build it. If you are using a newer version and are encountering problems with building the project, the easiest way to get everything working might be to go back to 2021.3.
- **Did you modify any of the included assets?**
  If you modified the project, it might be that the problems occurred due to these modifications. To check out if this is the case, create a new empty project, import the asset and check if the problems still occur. If these steps did not help and you are still having problems with the project or getting shown any error messages after the import, please contact us via email (support@ilumisoft.de)

Project Architecture

For **Twicks** we made heavy use of abstract classes and interfaces to define reusable and interchangeable components. Additionally, we split the project into multiple Assemblies to improve compile time and reduce coupling. When opening the scripts folder (*Assets/Ilumisoft/Twin Stick Shooter Kit/Scripts*) you will find the script files well organized into different folders, each one representing a different assembly.

The following dependency diagram illustrates how the assemblies depend on each other.


Dependency Diagram
Sample implementations for AI characters, weapons, custom missions, ...
Sample
Base
Base classes and interfaces for components and core systems
AI
Editor
Game
UI
Visual Scripting
Contains our AI Behaviour Tree Framework
Editor Windows, inspectors, etc.
Concrete default implementations for base components like Health, Hitbox, ...
UI specific classes
Custom nodes for Visual Scripting

As you will notice, the **Sample**, **Editor**, **Game**, **UI** and **Visual Scripting** assembly depend on the **Base** assembly, but not on each other. Vice versa the **Base** assembly does not depend on any of these and stands on its own. It is the core of the framework and mainly contains abstract classes and interfaces; the other assemblies can rely on.
The **Game** assembly contains default implementations for most of these base components. For example, the **Health**, **Hitbox** and **Loot** component, which inherit from the abstract **HealthComponent**, **HitboxComponent** and **LootComponent** classes of the Base assembly.

This architecture has several benefits:

- When making a change for example in the **Game** assembly, none of the other assemblies need to be recompiled, because they do not depend on it
- By depending on abstractions, components are more reusable and interchangeable. You can easily swap out the components defined in the **Game** assembly with your own, since the framework only depends on their abstract base classes and not on the concrete implementations.

How to use and extend the framework

Our **Sample** folder represents an example of how you can create logic and content on top of the framework. It contains enemies with a simple sample AI based on **Behaviour Trees** and contains examples of how you can create custom weapons, pickups and missions via scripting.
By creating a custom Assembly Definition, we made sure that it only depends on the **Base** and **AI** assembly.

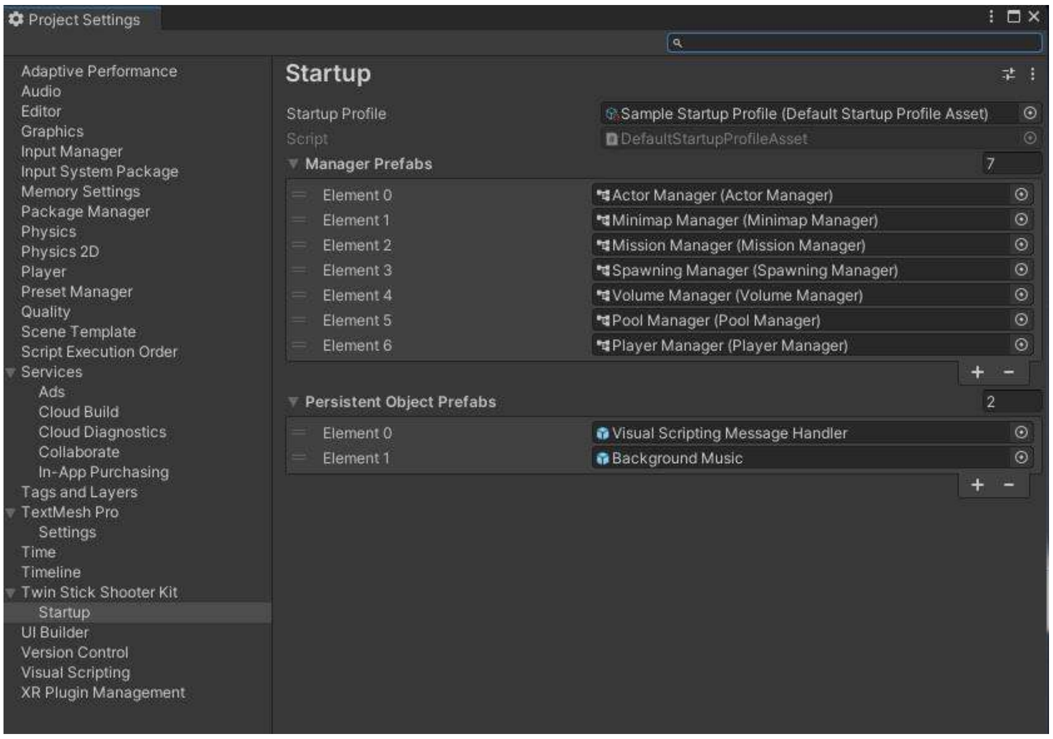
We recommend that you start by exploring the content and playing around with it. You can use and extend the included scripts and prefabs or use them as a reference to build your own.

Manager System

Twicks comes with a powerful and customizable manager system, allowing you to easily add and configure persistent systems and managers for your game. These systems are automatically created at startup and can be accessed from script using a service locator.

Startup Profiles

To make it easy for you to configure which objects should be created at startup, we created Startup Profiles. A Startup Profile allows you to define and configure all persistent managers, the system should handle. You can review and edit the current startup profile in the Project Settings (Edit->Project Settings->Twin Stick Shooter Kit->Startup)



Startup

Startup Profile	Sample Startup Profile (Default Startup Profile Asset)
Script	DefaultStartupProfileAsset
Manager Prefabs	7
Element 0	Actor Manager (Actor Manager)
Element 1	Minimap Manager (Minimap Manager)
Element 2	Mission Manager (Mission Manager)
Element 3	Spawning Manager (Spawning Manager)
Element 4	Volume Manager (Volume Manager)
Element 5	Pool Manager (Pool Manager)
Element 6	Player Manager (Player Manager)
Persistent Object Prefabs	2
Element 0	Visual Scripting Message Handler
Element 1	Background Music

Aside from manager prefabs, you can also use the startup configuration to create other persistent objects, which will also be created at startup and added to **DontDestroyOnLoad**. For example, we created a custom startup profile for the sample (*Assets->Create->Configuration->Startup Profile*) and added a persistent game object for the Background Music. Another useful example would be to add Debug Tools to your startup profile, while in development (or create a special startup profile for debugging).

Access managers via script

To get a refence to a manager component at runtime from your script, you can call the **Get<T>()** method of our **Managers** locator class. For example, to get a reference to the Volume Manager, you would call:

IVolumeManager = Managers.Get<IVolumeManager>();

Custom Managers

To add a new Manager to the game, you will need to write a component inheriting of the **ManagerComponent** class and create a prefab for it. You can then add this prefab to the active startup profile.

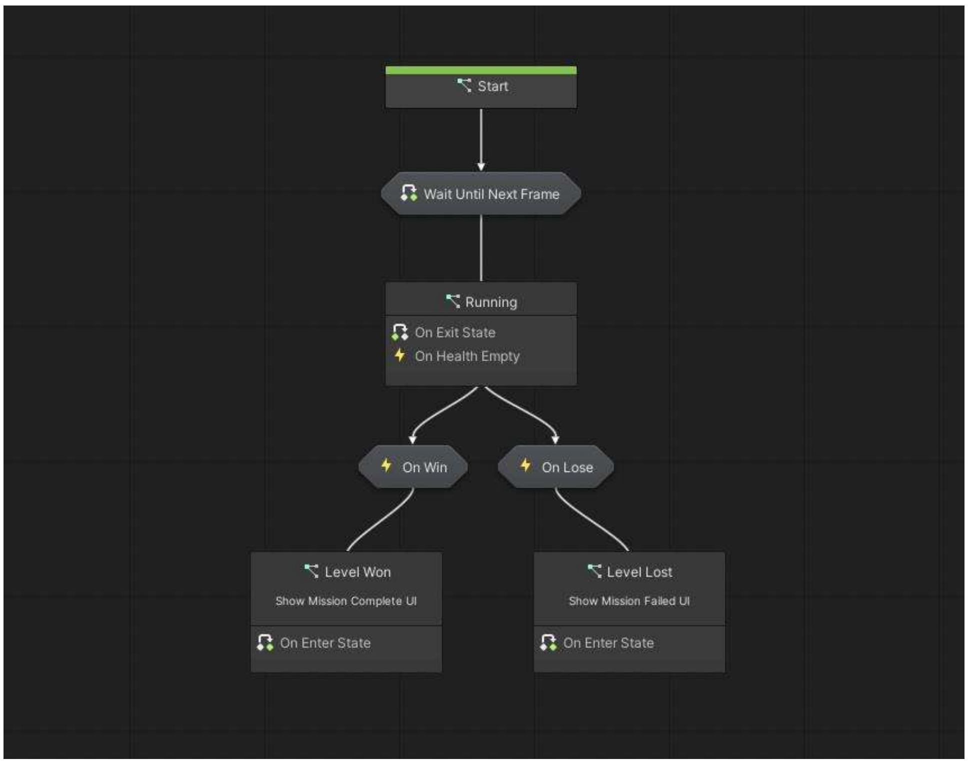
You can also easily switch out our default implementations of existing managers. You could for example replace the default Volume Manager with your own, by creating a new Manager class, implement the **IVolumeManager** interface and assign the new prefab to the startup profile.

Visual Scripting Support

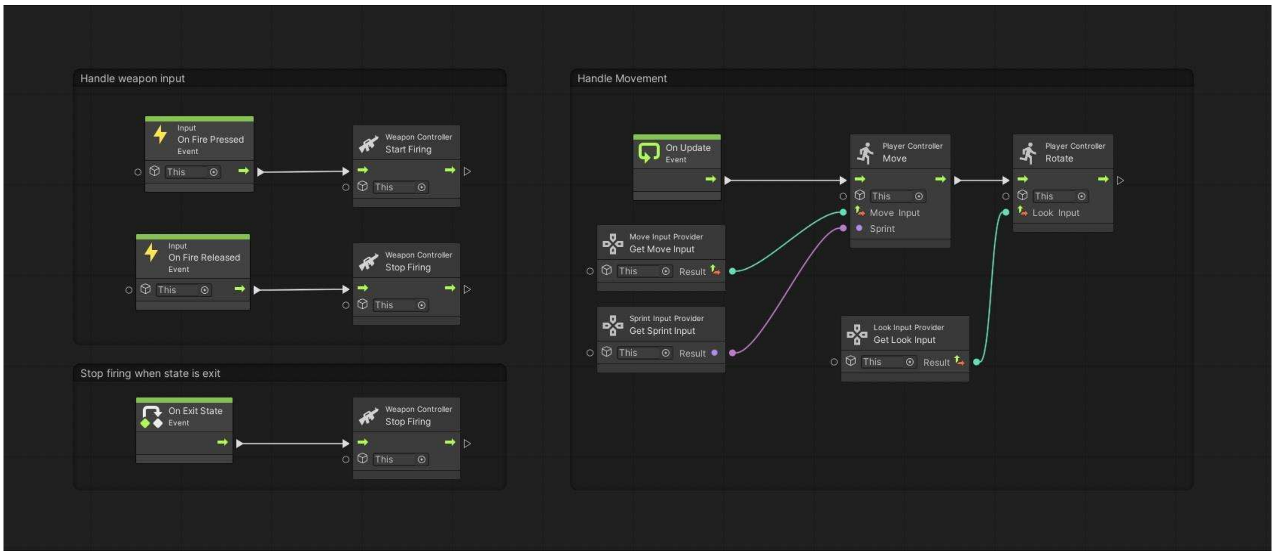
While we think that **Visual Scripting** is not a good choice when implementing complex logic or behaviour, it can come very handy when prototyping, to control the game flow or to control interactions on a high-level using State Machines.

Therefore, **Twicks** comes with basic support for **Visual Scripting**. We implemented custom nodes to interact with some of the core components and custom event nodes, allowing you to receive events in Visual Scripting Graphs.

We also implemented some of the **Sample** content using **Visual Scripting**, like the game flow and the player state machine, to give you an example of how you can use **Twicks** with **Visual Scripting**.



Game Flow State Machine Sample


Handle weapon input
Input On Fire Pressed Event
This
Weapon Controller Start Firing
This
Input On Fire Released Event
This
Weapon Controller Stop Firing
This
Stop firing when state is exit
On Exit State Event
Weapon Controller Stop Firing
This
Handle Movement
On Update Event
Move Input Provider Get Move Input
This
Result
Sprint Input Provider Get Sprint Input
This
Result
Player Controller Move
This
Move Input
Sprint
Look Input Provider Get Look Input
This
Result
Player Controller Rotate
This
Look Input

Player Alive State Script Graph (Sample)

Support

If you like the project, please take a minute and give us a rating in the Asset Store. This really helps us to create and improve our Unity Assets. If you encounter any problems, errors, or have a question do not hesitate to contact us via email: support@ilumisoft.de