Spaceship AI

--- SINCRESS ---

Welcome to the documentation of the **Spaceship AI** for Unity3D. This document describes the example scene, the provided assets and the scripts of the framework. To find a description of a script, just search its name in this document. Thank you for selecting the *Spaceship AI* for your project, I hope you enjoy using this package and expanding on it.

1. Features

This asset provides a physically simulated model of a spaceship (which has colliders and rigidbodies and exists within Unity's physics engine) which is being driven by PID (Proportional Integral Derivative) controllers. This enables the user to easily give orders to the ships which can autonomously navigate and perform the orders (such as Move, Follow, Attack, Dock, etc.). Furthermore, this ship model can easily be expanded on by adding other modules such as player input, ship settings, weapon systems and other.

Here's what this asset provides:

- Rigidbody physics driven ship model
- Modular ship scripting enables easy expansion and adding of features
- AI command framework for easy order issuing and management
- Modular system for adding new orders
- Several example orders implemented (Move, Idle, Patrol)
- UI markers for ships with ship selection feature
- Game proven mechanics and a well-polished framework

Let's follow up with a short overview of the main classes which correspond to the principal components mentioned above:

The **Ship** class is the main part which ties all the ship components together. Currently it connects the **ShipPhysics** and the **ShipAI** classes. This is where you would want to tie in your main ship functionality and add other ship features such as ShipWeapons, ShipPowerups, ShipPlayerInput or others. **ShipPhysics** deals with applying forces to the ship as a result of throttle and steering input. It also defines how the ship handles: the speed and maneuovrability of the vessel. **ShipAI** is the class responsible for processing the current command given to the ship and serves as an interface between the orders and the ship. It also provides methods which allow you to issue orders to the ship, making it very simple to place an order: for example,

shipGameObject.GetComponent<ShipAI>().MoveTo(targetposition)

Order class defines the base for all orders which are implemented in their respective classes. Each order must have a name and an UpdateState method which receives the ship's context in each call and applies the necessary throttle input depending on the environment. All the orders have one thing in common: they all need to steer the ship toward the target. For this purpose, the **SteerAction** class is used. It applies the steering input to face the destination (or target) by using PID controllers.

These are the main components of the project, however, there are some other interesting features. The **HUDMarkers** class draws clickable ship markers on the user interface. Once a marker is clicked the ship is marked as selected and orders can be given to it. An order can be issued to the selected ship through the **OrderMenu** which is displayed in the upper left corner of the user interface. The **ConsoleOutput** allows printing notifications and warnings to the user interface and occupies the upper right of the UI.

2. Example Scene

The example scene is in the root of the project folder. It showcases the features of this framework in a basic example. The main camera is fixed and does not move, but has an ambiental starfield and views the entire field. The **Canvas** displays the user interface: the HUDMarkers, ConsoleOutput and OrderMenu. Five **AI ships** with placeholder models are

present, marked *Alpha* through *Echo.* The Patrol order is issued to every ship initially by the **AIController** object. To provide the ambient an asteroid spawner is included, courtesy of https://github.com/brihernandez/UnityCommon/blob/f288ee4bd2593cf8ebbdd9046197a475141cad17/Assets/RandomAreaSpawner/Code/RandomAreaSpawner.cs. The rest of the objects are waypoints used for navigation. Note that waypoints are tagged with *Waypoint* and ships with *Ship* tags.

3. Scripts

This section provides a script reference for the components of the framework. The code provided is self documenting and has a lot of comments to speed up your development and help you understand the components. However, if you need further explanation how a function works or what it does, or perhaps a high level overview, read it here.

3.1. Ship

Ship.cs

One of the classes present on every spaceship gameObject. Carries basic info about the ship and references to all the other components, such as *ShipPhysics, ShipAI* or any other similar classes you might wish to add. This is convenient because if any other part of your game needs to access a certain component it's all possible just by having a reference to the *Ship* script.

ShipPhysics.cs

This class is tasked with adding forces to the ship's rigidbody depending on the **linear** and **angular** input given. There is no implementation of player input given, but it's pretty similar to what the AI uses. **Throttle** goes from 0.0 to 1.0, and so does **strafe** (Y axis of the linear input). The steering inputs (angular forces) do the same but they are directly supplied by the *ShipSteering* script. The values are read in Update from the *ShipAI* script (there they are being set by the current order) and are then applied to the ship using *AddRelativeForce* and *AddRelativeTorque* in FixedUpdate.

ShipAI.cs

Responsible for assigning and performing orders of a ship. Orders are issued to a ship through this script and it carries all the variables manipulated by the orders themselves. In each Update execution, the current Order's UpdateState method is invoked and the reference to *this* is passed. That's how the Order receives the ship's context and gets the info needed to make decisions. The most important variable used by the orders is ***tempDest*** which is the current destination for each order. If it has the value (0, 0, 0) that means it isn't used and the destination is a Transform from the **wayPointList**. If the order requires only one destination only the first element of the wayPointList is used.

PIDController.cs

An implementation of the proportional integral derivative controller. Wikipedia and Youtube can offer a far better explanation than I can, but in short, it computes output based on the current input, the error value (difference between expected and actual output) and the gradient of the error (the speed at which the error increases or decreases). To do that, it uses a feedback loop and three parameters: the coefficients P, I and D. By modifying these, you get different ship behavior. This class is only used by the **SteeringAction** because that's the only thing that needs a PID controller to operate.

3.2. Order

SteeringAction.cs

This is used by every order. Every order needs the ship to turn towards a certain destination (or at least every order I've come across: Attack, MoveTo, Follow, Protect, Patrol, etc.). It uses two PID controllers: one for **angular velocity** and the other one for the **angle** itself.

Order.cs

Derive every order from the **Order** class. Every order must have a *Name* and an *UpdateState* function which receives the ShipAI context. This function is not a MonoBehavior. This asset gives several implementations of orders:

OrderMove – commands the ship to move to a given position or target

OrderIdle – makes the ship fly randomly by giving random destinations close to the ship once it has reached the current destination

OrderPatrol – commands the ship to navigate a set of waypoints in a circular manner

OrderFollow – orders the ship to follow a Transform

Study these orders to understand how they work and how to implement your own. Remember that all these orders do is modulate the throttle depending on the distance to waypoints and onother parameters.

3.3. Miscellaneous scripts

HUDMarkers.cs

This script displays rectangular target markers on the Canvas so that you could see the ships better and select them by clicking on them. It also holds a reference to the currently selected ship which is used by the *OrderMenu* which issues orders to it. The markers use a marker pool to avoid costly instantiation and destruction. Feel free to improve on this if you wish to use it in your project.

OrderMenu.cs

Pretty simple script gives a really nice example of how easy it is to give an order to a selected ship, once you have its gameObject reference. Just obtain the reference to the **ShipAI** script and call the method corresponding to the order you want.

ConsoleOutput.cs

Allows you to print stuff directly to the user interface, with custom formatting and colors. Static reference to the class is available using a Singleton pattern, so to print something all you need to do is

ConsoleOutput.Instance.PostMessage("mymessage", Color.red);

I hope you find this package helpful and easy to use in your project. If you have questions or feedback please contact me at sincress@gmail.com. Please rate this Asset on the Unity Asset Store, it means a lot.

Sincress