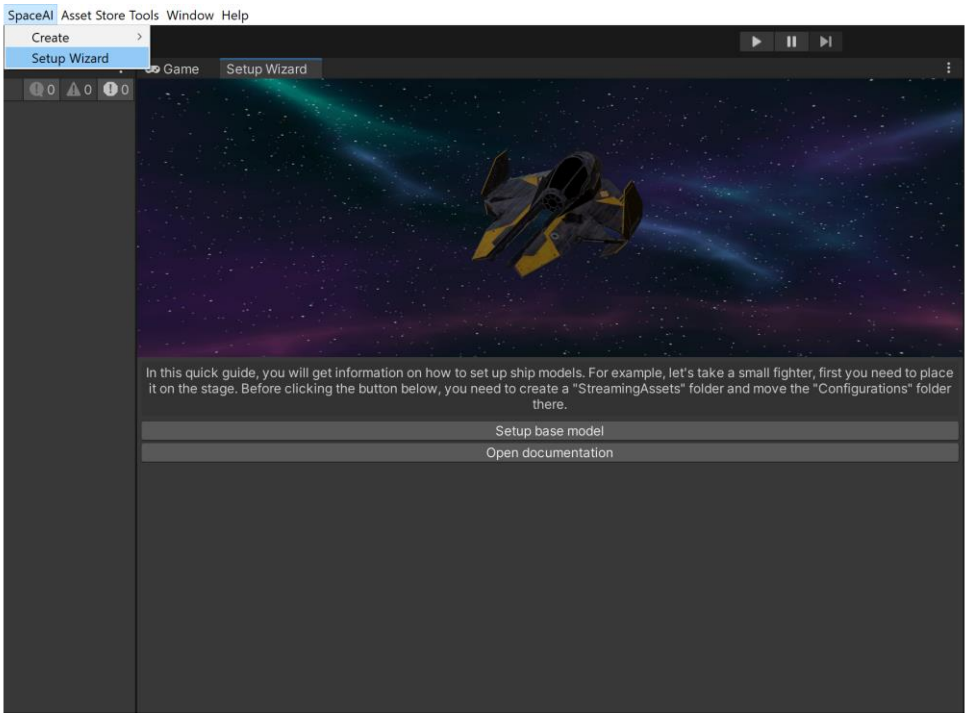
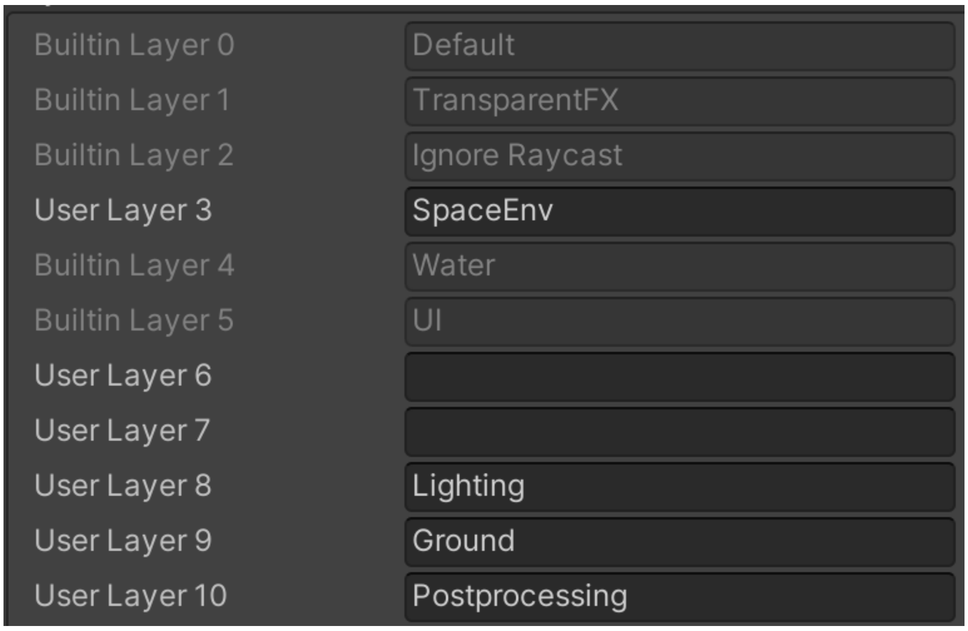
Table of content

Open Setup Wizard, it will guide you



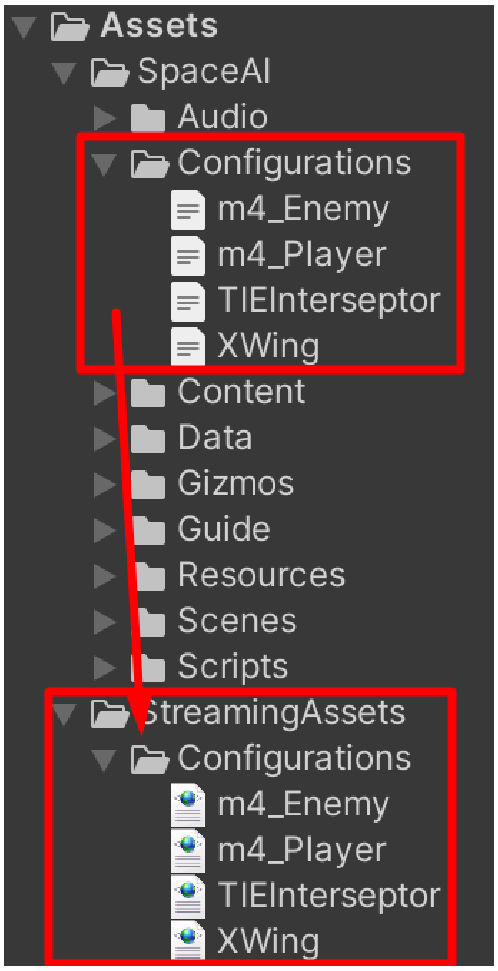
In this quick guide, you will get information on how to set up ship models. For example, let's take a small fighter, first you need to place it on the stage. Before clicking the button below, you need to create a "StreamingAssets" folder and move the "Configurations" folder there.

Configurate layers



Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
User Layer 3	SpaceEnv
Builtin Layer 4	Water
Builtin Layer 5	UI
User Layer 6	
User Layer 7	
User Layer 8	Lighting
User Layer 9	Ground
User Layer 10	Postprocessing

Create StreamingAssets folder and put Configurations folder there



AI

The finite state machine (FSM) implemented in the received code is used to control the behavior of AI-controlled ships in a game. The FSM consists of several states, each representing a specific behavior or action the ship can perform. Let's provide a detailed description of the FSM based on the provided code:

1. **FSMState** (Abstract Base Class):

- This class serves as the abstract base class for all the states in the FSM.

- It defines the common properties and methods required for a state, such as the owner ship, a transition-state map, and the state ID.

- Key methods include AddTransition, DeleteTransition, GetOutputState, DoBeforeEntering, DoBeforeLeaving, Reason, and Act.

2. **AttackState**:

- This state represents the behavior of the ship when it is in attack mode.

- It inherits from FSMState and implements the Reason and Act methods.

- The DoBeforeEntering method is called before entering the state and initializes relevant variables.

- The Act method controls the ship's behavior during the attack state, including setting a target position, checking if the way is clear, and launching weapons or controlling turrets.

- The Reason method decides whether to transition to another state based on conditions such as the proximity of the enemy, the ship's location, or the need to patrol.

3. **IdleState**:

- This state represents the ship's idle behavior when it is not engaged in combat.

- It inherits from FSMState and implements the Reason and Act methods.

- The DoBeforeEntering method initializes variables related to the idle state, such as timing and target scanning parameters.

- The Act method controls the ship's behavior during the idle state, such as following a target or transitioning to attack mode.

- The Reason method determines whether to transition to another state, such as attack or patrol, based on the ship's current situation and enemy presence.

4. **TurnState**:

- This state represents a turning maneuver behavior for the ship.

- It inherits from FSMState and implements the Reason and Act methods.

- The DoBeforeEntering method sets up variables related to the turning maneuver, such as duration, roll frequency, and roll direction.

- The Act method controls the ship's behavior during the turn state, performing a rolling maneuver.

- The Reason method decides whether to transition to another state, such as attack or patrol, based on a time threshold and the presence of an enemy.

5. **FSMSystem**:

- This class manages the overall FSM for a ship.

- It keeps track of the current state and provides methods to perform transitions between states.

- The Update method is typically called in the ship's update loop to continuously update the FSM's current state.

These are the main components of the FSM implemented in the received code. The FSM allows the AI-controlled ships to exhibit different behaviors based on their current state and transition between states based on specific conditions and events, providing dynamic and varied gameplay.

Controllers

Ship Controllers

Namespace: SpaceAI.Ship

The Ship Controllers namespace contains classes that represent various ship controllers in the SpaceAI. These controllers extend the base ship functionality provided by the SA_BaseShip class and implement specific behavior and logic for different ship types.

SA_BaseShip Class

The SA_BaseShip class is an abstract class that serves as the base for all ship controllers. It provides common functionality and properties for ships, such as handling ship systems, registering collision events, and applying damage. It also defines abstract methods that derived ship controllers must implement, such as Move() and OnSystemsReady().

SA_ShipController Class

The SA_ShipController class is a derived class of SA_BaseShip and represents a specific type of ship controller in the SpaceAI game. It extends the base ship functionality to provide movement control and behavior specific to this ship type. It overrides the Move() method to calculate ship rotation and velocity based on the target position and ship configuration. It also provides additional methods for handling ship movement calculations, checking if the ship is too far from its target position, and setting the current enemy target.

The `SA_BaseShip` class is an abstract class that serves as the base for all ship controllers in the SpaceAI game. It implements the `SA_IShip` and `SA_IDamage` interfaces and provides common functionality and properties for ships.

Public Properties:

- ShipConfiguration: Returns the ship configuration manager instance that holds the ship's configuration data.

- WeaponControll: Returns the weapon controller instance associated with the ship.

- CurrentAIProvider: Returns the current AI provider instance responsible for providing AI functionality to the ship.

- CurrentEnemy: Gets or sets the current enemy target of the ship.

- GetCurrentTargetPosition: Gets the current target position of the ship.

- CurrentShipSize: Gets the current size of the ship.

- CurrentShipTransform: Gets the transform component of the ship.

- CurrentMesh: Gets the mesh component of the ship.

Events:

- CollisionEvent: An event that is triggered when a collision occurs with the ship. Subscribers can listen to this event to handle collision-related logic.

Protected Methods:

- Move(): An abstract method that must be implemented by derived ship controllers to define the ship's movement behavior.

- OnSystemsReady(SA_ShipSystemsInitedEvent e): A method that is called when the ship systems are ready. Derived ship controllers can override this method to perform additional logic when the ship systems are initialized.

Private Methods:

- Start(): A coroutine method that is automatically called when the ship is enabled. It initializes ship components, runs built-in ship systems, and enters a continuous loop for ship operations.

- OnEnable(): A method that is called when the ship is enabled. It adds an event listener for the ship systems initialized event.

- OnDisable(): A method that is called when the ship is disabled. It removes the event listener for the ship systems initialized event and cleans up ship system event subscriptions.

- OnCollisionEnter(Collision collision): A method that is called when a collision occurs with the ship. It invokes the CollisionEvent, triggering any registered collision event handlers.

- RunBuildInSystems(): A coroutine method that runs built-in ship systems. It handles obstacle avoidance and updates the ship's shield state.

- InitShipComponents(): A coroutine method that initializes ship components, loads ship data, and initializes built-in ship systems.

- GetComponents(): A private method that retrieves and assigns references to ship components such as the mesh, rigidbody, and on-fire particle system.

- LoadShipData(): A coroutine method for loading ship data. It copies the ship template file and loads the ship configuration from XML data.

- LoadShipDataUnityOnly(): A method for loading ship data in the Unity Editor. It loads the ship configuration from XML data.

- InitBuildInSystems(): A private method that initializes the built-in ship systems such as the obstacle system, weapon controller, and health provider. It also adds the shield system if available.

- Validate(): A method available only in the Unity Editor for validating ship components and settings.

Interface Methods:

The SA_BaseShip class implements the following interface methods:

- SubscribeEvent(Action<Collision> collisionEvent): Subscribes to the collision event by adding the specified collision event handler.

- SetTarget(Vector3 target): Sets the target position for the ship to move towards.

- SetTarget(Transform target): Sets the target position for the ship to move towards, based on the target's transform component.

- SetTarget(GameObject target): Sets the target position for the ship to move towards, based on the target's position.

- Ship(): Returns the group type of the ship.

- WayIsFree(): Checks if the way for the ship is clear of obstacles. Returns true if the way is clear, and false otherwise.

- CanFollowTarget(bool followTarget): Sets whether the ship should follow the target or not. Returns the updated value

of followTarget.

- ToFar(): Checks if the ship has gone too far from its target position. Returns true if the ship is too far, and false otherwise.

- SetCurrentEnemy(GameObject newTarget): Sets the current enemy target of the ship.

- ApplyDamage(float damage, GameObject killer): Applies damage to the ship and handles the consequences, such as triggering the ship's destruction when its health reaches zero.

The SA_ShipController class is a derived class of SA_BaseShip in the SpaceAI.Ship namespace. It represents a specific type of ship controller in the game and extends the base ship functionality to provide movement control and behavior specific to this ship type.

Fields:

- isReady: Indicates whether the ship is ready to operate.

- worldSpeed: The speed multiplier for world movement.

Methods:

- OnSystemsReady(SA_ShipSystemsInitedEvent e): Overrides the base method to handle ship systems initialization. It creates an AI provider and sets up the base model finite state machine (FSM) for the ship.

- FixedUpdate(): Called every fixed frame update. It updates the ship's AI states and performs ship movement.

- Move(): Overrides the base method to define ship movement logic. It calculates ship rotation and velocity based on the target position and ship configuration.

- MovementCalculation(): Calculates the ship's movement based on its configuration and target position.

- ToFar(): Overrides the base method to determine if the ship is too far from its target position based on the configured fly distance.

- SetCurrentEnemy(GameObject newTarget): Overrides the base method to set the current enemy target for the ship.

Remarks:

SA_ShipController extends SA_BaseShip to provide specific ship controller functionality for a particular ship type in the SpaceAI game. It customizes the ship's movement behavior based on its configuration and target position by overriding necessary methods.

DataManagment

SpaceAI Data Management

This documentation provides an overview of the classes and functions within the "SpaceAI.DataManagment" namespace. The codebase handles data management and configuration settings for spaceships and AI in the SpaceAI.

SA_AIConfiguration Class

- Namespace: SpaceAI.DataManagment

- Description: This class represents the AI configuration for a spaceship. It includes properties such as the group type, group types to take action, ship target scan range, and target request frequency.

SA_FileManager Class

- Namespace: SpaceAI.DataManagment
- Description: This class provides functions for saving and loading XML files related to ship configurations and other data.

- `SaveXml<T>(T obj, string fileName)`: Saves an object of type T as an XML file with the given filename.
- `LoadXml<T>(string fileName)`: Loads an object of type T from the XML file with the given filename.
- `SaveXmlConfigUnityOnly<T>(T obj, string fileName)`: Saves an object of type T as an XML file in the Unity streaming assets folder.
- `LoadXmlConfigUnityOnlyAsync<T>(string fileName)`: Asynchronously loads an object of type T from the XML file in the Unity streaming assets folder.
- `LoadXmlConfigUnityOnly<T>(string fileName)`: Loads an object of type T from the XML file in the Unity streaming assets folder.
- `CopyShipTemplateFile(string fileName)`: Copies a ship template file from the Unity streaming assets folder to the persistent data path.

SA_ItemsStaf Class

- Namespace: SpaceAI.DataManagment

- Description: This class represents the items (such as sounds, effects, and prefabs) associated with a spaceship.

SA_MainConfigs Class

- Namespace: SpaceAI.DataManagment

- Description: This class contains the main configuration settings for a spaceship. It includes properties for movement, damage, and flying settings.

SA_Options Class

- Namespace: SpaceAI.DataManagment

- Description: This class represents options for a spaceship, including settings for turrets.

SA_ShieldsConfiguration Class

- Namespace: SpaceAI.DataManagment

- Description: This class represents the configuration settings for a spaceship's shields. It includes properties to enable shields, handle collision events, and control shield power.

SA_ShipConfigurationManager Class

- Namespace: SpaceAI.DataManagment

- Description: This class manages the configuration settings for a spaceship. It contains instances of other configuration classes (e.g., AI, shields, options), ship systems, and items. It provides functions for saving and loading ship configurations.

- `Save(string fileName)`: Saves the ship configuration as an XML file in the Unity streaming assets folder.
- `Load(string fileName)`: Loads the ship configuration from the XML file in the Unity streaming assets folder.
- `SetAsDefault()`: Sets default values for the main configuration settings.

SA_ShipSystems Class

- Namespace: SpaceAI.DataManagment
- Description: This class represents the ship systems of a spaceship. It holds references to scriptable objects for different ship systems and allows creating instances of these systems.

SA_ShipSettingsConfiguration Class

- Namespace: SpaceAI.DataManagment
- Description: This class is a custom editor for the SA_ShipConfigurationManager. It provides an inspector GUI to edit and manage spaceship configurations. It allows setting default settings, saving, loading, and displaying properties of the ship configuration.

BaseShipLoader Class

- Namespace: SpaceAI.DataManagment

- Description: This class is a custom editor for the SA_BaseShip script. It allows editing and managing base spaceship properties. The exact implementation details are not provided in the given code snippet.

This documentation provides an overview of the data

EventBus

SA_EventsBus class:

- This class serves as the main event bus and contains methods for adding and removing event listeners, as well as publishing events.

- It uses a dictionary called `Listeners` to store event types as keys and a list of event handlers as values.

- The `CHandler` class is a helper class that wraps an event handler and provides access to it.

- The event bus supports deferred actions, allowing the addition and removal of event handlers even when the bus is locked.

Handler delegate:

- This delegate defines the signature of event handlers. It is a generic delegate that takes an event argument derived from the `SA_IEvent` interface.

SA_IDeferredAction interface and DeferredActions enum:

- These interfaces and enum define the types of deferred actions that can be performed on the event bus.

- `SA_IDeferredAction` is implemented by classes representing deferred actions, such as adding or removing event handlers.

SA_AddDeferredAction class:

- This class represents a deferred action to add an event handler to the event bus.

SA_PublishDeferredAction class:

- This class represents a deferred action to publish an event to the event bus.

SA_RemoveDeferredAction class:

- This class represents a deferred action to remove an event handler from the event bus.

SA_IEvent interface:

- This interface serves as a marker interface for event structs. Events that need to be published through the event bus should implement this interface.

SA_ShipRegistryEvent struct:

- This struct represents an event indicating the registration of a ship.

SA_ShipSystemsInitedEvent struct:

- This struct represents an event indicating that the ship systems have been initialized.

SA_ShipTargetRequesEvent struct:

- This struct represents an event requesting a ship's target within a specified scan range.

SA_ShipUnRegisterEvent struct:

- This struct represents an event indicating the unregistration of a ship.

Overall, this event bus implementation provides a mechanism for loosely coupling event producers and consumers within the SpaceAI system. It allows different parts of the system to communicate by publishing and subscribing to events.

Interfaces

SA_IDamage interface (namespace: SpaceAI.Core):

- This interface defines a method `ApplyDamage` that can be implemented by classes to apply damage to an object. It takes a `float` value for the amount of damage and a `GameObject` parameter representing the entity causing the damage.

SA_IDamageSender interface (namespace: SpaceAI.Core):

- This interface defines properties `Owner` and `Target` that can be implemented by classes representing a damage sender. The `Owner` property returns the GameObject that owns the damage sender, while the `Target` property returns the GameObject that is the target of the damage sender.

GroupType enum (namespace: SpaceAI.Ship):

- This enum represents the type of a ship's group, such as "Enemy" or "Player." It can be extended to include more group types as needed.

SA_IShip interface (namespace: SpaceAI.Ship):

- This interface defines the properties and methods that a ship class should implement.

- It includes properties like `ShipConfiguration` (a ship configuration manager), `WeaponController` (a weapon controller for the ship), `CurrentAIProvider` (the current AI provider for the ship), `CurrentEnemy` (the current enemy target), `CurrentMesh` (the current mesh of the ship), `CurrentShipTransform` (the transform of the ship), `GetCurrentTargetPosition` (the current target position), `CurrentShipSize` (the size of the ship), etc.

- It also includes methods like `WayIsFree` (checks if the way for the ship is free), `SubscribeEvent` (subscribes to a collision event), `SetTarget` (sets the target position, transform, or GameObject for the ship), `CanFollowTarget` (checks if the ship can follow the target), `ToFar` (checks if the ship is too far from the target), and `SetCurrentEnemy` (sets the current enemy target).

SA_IShipSystem interface (namespace: SpaceAI.ShipSystems):

- This interface defines the methods that a ship system class should implement.

- It includes the `Init` method, which initializes the ship system with a ship and a GameObject, and the `ShipSystemEvent` method, which handles ship system events, such as collisions.

ShellType enum (namespace: SpaceAI.WeaponSystem):

- This enum represents the type of a weapon shell, such as "Laser" or "Missiles." It can be extended to include more shell types as needed.

SA_IWeapon interface (namespace: SpaceAI.WeaponSystem):

- This interface defines the methods and properties that a weapon class should implement.

- It includes the `Shoot` method, which initiates the shooting action of the weapon, and the `SetOwner` method, which sets the owner ship of the weapon.

- It also includes the `Settings` property, which returns the settings of the weapon, and the `SetFireShells` method, which sets the fire shells (damage handlers) for the weapon.

These interfaces and enumerations provide a basis for defining and implementing damage-related functionality, ship systems, and weapon systems in the SpaceAI namespace.

Systems

SpaceAI.ShipSystems.SA_Shield

Description

The `SA_Shield` class represents a shield system for a spaceship in the SpaceAI. It provides functionality for creating and managing a shield that can absorb damage. The shield's appearance and behavior can be customized through various properties.

Constructors

- `public SA_Shield()`: Initializes a new instance of the `SA_Shield` class with default values.

- `public SA_Shield(SA_IShip ship, GameObject shieldField)`: Initializes a new instance of the `SA_Shield` class with the specified ship and shield field parameters.

Properties

- `public float ShieldPower`: Gets or sets the power of the shield.

Methods

- `public override SA_IShipSystem Init(SA_IShip ship, GameObject gameObject)`: Initializes the shield system with the specified ship and game object.

- `public override void ShipSystemEvent(Collision obj)`: Handles ship system events, such as collisions. Activates the shield and processes collision points if collisionEnter is enabled.

- `public void OnHit(Vector3 hitPoint, float hitPower = 0.0f, float hitAlpha = 1.0f)`: Sends impact coordinates and hit properties to the shield. This method can be used by other scripts to simulate hits on the shield.

- `public void UpdateFade()`: Updates the fading effect of the shield by gradually reducing the alpha values of the impact points.

Internal Fields

- `private GameObject field`: Reference to the shield field GameObject.

- `private bool collisionEnter`: Determines whether the shield reacts to collision events.

- `private float decaySpeed`: The speed at which the shield impact points fade over time.

- `private float reactSpeed`: The minimum interval between registering new impact points.

- `private bool fixNonUniformScale`: Determines whether the shield should fix non-uniform scale issues.

- `private float shieldPower`: The current power level of the shield.

- `private Transform rootTransform`: The transform of the ship to which the shield is attached.

- `private Material mat`: Reference to the shield field's material.

- `private MeshFilter mesh`: Reference to the shield field's mesh.

- `private Vector4[] shaderPos`: An array of impact points in shader format.

- `private int curProp`: The index of the current impact point being processed.

- `private int interpolators`: The maximum number of impact points to support.

- `private int[] shaderPosID, shaderPowID`: An array of shader property IDs for optimizing performance.

- `private float curTime`: The current time since the last impact point was registered.

- `private SA_ShipConfigurationManager Configuration`: Reference to the ship's configuration manager.

SpaceAI.Weapons.SA_Missile

Description
The `SA_Missile` class represents a missile weapon in the SpaceAI game. It provides functionality for controlling the movement and behavior of a missile projectile. The missile can be guided or unguided, and it can be affected by Perlin noise to simulate realistic motion.

Fields

- `public MissileSettings missileSettings`: The settings for the missile, including speed, damping, and noise parameters.

Methods

- `private void FixedUpdate()`: Performs the physics-based movement of the missile. It applies speed, damping, and Perlin noise to calculate the missile's velocity and updates its position.

Internal Fields

- `private float perlinTime`: The current time used to calculate Perlin noise values.

SpaceAI.Weapons.SA_MissileTypes

Description

The `SA_MissileTypes` class represents different types of missiles in the SpaceAI game. It provides functionality for controlling the movement and behavior of missiles based on their type,

such as unguided, guided, or predictive missiles.

Fields

- `public MissileSettings missaleSettings`: The settings for the missile, including type, velocity, and alignment speed.

Methods

- `void Awake()`: Called when the script instance is being loaded. Initializes the transform reference.

- `void FixedUpdate()`: Called every fixed frame update. Controls the movement and alignment of the missile based on its type.

- `public static Vector3 Predict(Vector3 sPos, Vector3 tPos, Vector3 tLastPos, float pSpeed)`: Predicts the position of a target based on its current position, last position, and velocity. Used for predictive missiles.

- `static float GetProjFlightTime(Vector3 dist, Vector3 tVel, float pSpeed)`: Calculates the flight time of a projectile based on the distance to the target, target velocity, and projectile speed. Used for predictive missiles.

1. **SA_DamageSandler.cs**: This script handles damage and explosion effects for a projectile. It is responsible for dealing damage to targets in its vicinity, creating explosion effects, and managing the lifetime of the projectile.

2. **SA_Turret.cs**: This script represents a turret that can rotate and aim at targets. It can be controlled independently or automatically aim at targets based on certain conditions. The turret has a base for horizontal rotation and barrels for vertical rotation.

SA_DamageSandler.cs:

Properties and Fields:

- `ShellType ShellType`: Type of the projectile shell.
- `bool Explosive`: Whether the projectile is explosive.
- `float ExplosionRadius`: The radius of the explosion if the projectile is explosive.
- `float ExplosionForce`: The force of the explosion if the projectile is explosive.
- `float DestryAfterDuration`: The duration after which the projectile should be deactivated.

Methods:

- `void OnEnable()`: Called when the object is enabled. Sets up the projectile, ignores collisions with owner's colliders, and manages its lifetime.
- `void Active()`: Called when the projectile should be activated (e.g., on impact). Handles explosion effects and damage application.
- `void Deactivate(GameObject gameObject)`: Deactivates the specified GameObject.
- `async void Deactivate(GameObject gameObject, float time)`: Asynchronously deactivates the specified GameObject after a given time duration.

- `void ExplosionDamage()`: Applies damage and explosion force to nearby colliders when the projectile is explosive.

- `void OnCollisionEnter(Collision collision)`: Called when the projectile collides with something. Handles damage application and activation.

1. **Namespace and Using Directives**: The script is defined inside the "SpaceAI.WeaponSystem" namespace and imports the "SpaceAI.Ship" and "UnityEngine" namespaces.

2. **Inheritance**: The script inherits from "SA_WeaponLunchManager," suggesting that it extends functionality from another class named "SA_WeaponLunchManager."

3. **Public Fields and Properties**:

   - `runRotationsInFixed`: A boolean flag that determines if turret rotations should be performed in `FixedUpdate` rather than `Update`.

   - `turretBase`: An array of transforms representing the turret's base for horizontal rotation.

   - `turretBarrels`: An array of transforms representing the turret's barrels for vertical rotation. Note that it must be a child of the turret's base.

   - `turnRate`: Turn rate of the turret's base and barrels in degrees per second.

   - `limitTraverse`: A boolean flag that, when true, indicates that the turret's rotation is limited by traverse angles.

   - `leftTraverse`: The left traverse angle limit in degrees.

   - `rightTraverse`: The right traverse angle limit in degrees.

   - `elevation`: The upward elevation angle limit in degrees.

   - `depression`: The downward depression angle limit in degrees.

   - `showArcs`: A boolean flag to show the arcs that the turret can aim through for debugging.

   - `showDebugRay`: A boolean flag to draw a debug ray showing where the turret is aiming when the game is running in the editor.

   - `independent`: A boolean flag that determines if the turret aims at targets automatically.

   - `shellOut`: An array of transforms representing the positions from where projectiles are launched.

   - `aimPoint`: The point where the turret should aim.

4. **Private Fields**:

   - `aiming`: A boolean flag indicating if the turret is currently aiming at a target.

   - `weaponController`: A reference to the "SA_WeaponController" component on the parent "SA_IShip" game object.

5. **Properties**:

   - `Idle`: A boolean property that returns true if the turret is idle (not aiming).

   - `AtRest`: A boolean property that represents if the turret is currently at rest (not rotating).

6. **Methods**:

   - `Update()`: Handles turret rotation and aiming when `runRotationsInFixed` is false.

   - `IndependentShipTurret()`: Controls the turret's aiming behavior when it's independent of the ship.

   - `FixedUpdate()`: Handles turret rotation and aiming when `runRotationsInFixed` is true.

   - `SetAimpoint(Vector3 position)`: Sets the aim point for the turret to aim at.

   - `SetAimpointFromShip()`: Sets the aim point based on the ship's position and direction.

   - `SetIdle(bool idle)`: Sets the turret to idle mode (not aiming).

   - `RotateTurret()`: Calls `RotateBase()` and `RotateBarrels()` to rotate the turret's base and barrels.

   - `RotateBase()`: Rotates the turret's base to aim at the target.

   - `RotateBarrels()`: Rotates the turret's barrels to aim at the target.

   - `RotateToIdle()`: Rotates the turret to its idle position when not aiming.

   - `DrawDebugRays()`: Draws debug

rays to visualize where the turret is aiming.

   - `ClearTransforms()`: A method intended for the Unity Editor, which clears the turret's transform references.

1. **SA_DamageBase**:

   - This class implements the `SA_IDamageSendler` interface.

   - It defines fields for a rigidbody, explosion effect prefab, explosion sound, life time of the effect, and damage amount.

   - It has methods to set the owner and target of the damage and provides properties for accessing the rigidbody, owner, and target.

2. **SA_WeaponBase**:

   - This class is a simple MonoBehaviour that provides public fields for the owner and target of a weapon.

3. **SA_WeaponController**:

   - This class implements the `SA_IShipSystem` interface and represents a weapon controller for a ship.

   - It contains a list of weapons (`WeaponLists`) and provides methods to initialize the controller, get the current weapon, launch weapons, switch weapons, reset turrets, and control turrets.

   - The constructor initializes the `WeaponLists` by retrieving all `SA_IWeapon` components from the owner ship's children.

   - The `LaunchWeapon` method launches a specific weapon at the given index.

   - The `LaunchWeapons` method launches all weapons in the `WeaponLists`.

   - The `ResetTurrets` method sets all turret weapons to idle mode.

   - The `TurretsControl` methods set the target and aim point for turret weapons.

4. **SA_WeaponLunchManager**:

   - This class extends `SA_WeaponBase` and implements the `SA_IWeapon` interface.

   - It represents a weapon launcher and contains various settings for the weapon, such as shell type, muzzle, fire rate, spread, force, reload time, ammo, sound effects, etc.

   - It has methods to initialize the weapon, shoot projectiles, and set the owner and fire shells.

   - The `Shoot` method is responsible for firing the weapon. It calculates the direction of the shot based on the spread, checks for ammo availability, and spawns projectiles from a pool.

1. `SA_TurretEditor`: This class is a custom editor for the `SA_Turret` component. It provides a custom inspector GUI for the turret and handles the drawing of turret arcs in the scene view.

2. `SA_HealthProvider`: This class represents the health system for a ship. It handles applying damage, tracking health points, managing shields, and triggering effects when the ship is destroyed.

3. `SA_ObstacleSystem`: This class is a ship system that deals with avoiding obstacles. It includes functionality to detect obstacles, find escape directions, and navigate the ship away from the obstacles.

4. `SA_ShipSystem`: This abstract class serves as the base class for ship systems. It provides a common interface and basic functionality for ship systems.

5. `ShipSystemFactory`: This class is a factory for creating ship systems. It maintains a dictionary of available system types and can create instances of specific ship systems based on their type.

Overall, these classes contribute to the ship and weapon systems in the game, including turret management, health tracking, obstacle avoidance, and system creation.

Other

Namespace: SpaceAI.SceneTools

This namespace contains the `SA_Manager` class, which is responsible for managing ship scanning and targeting functionality in the SpaceAI framework.

1. SA_Manager
   - Description: Manages ship scanning and targeting functionality.
   - Inherits: MonoBehaviour
   - Properties:
     - shipPrefabs: An array of ship prefabs to spawn.
     - storedObjs: A list of stored ship objects.
     - shipCount: The total number of ships to spawn.
     - initTime: The initial time before spawning the next ship.
     - m_count: The current count of spawned ships.
     - t: The time value used for spawning ships.

    - i: The index used for selecting ship prefabs.
    - SharedTargets: A list of shared ship targets.
    - BulletPool: A list of bullet pools.
  - Methods:
    - Awake(): Called when the script instance is being loaded.
    - Start(): Called before the first frame update.
    - OnEnable(): Called when the script instance is being enabled.
    - OnDisable(): Called when the script instance is being disabled.
    - OnShipTargetUpdate(SA_ShipTargetRequesEvent e): Event handler for ship target update events.
    - OnShipUnRegisterUpdate(SA_ShipUnRegisterEvent e): Event handler for ship unregistration events.
    - OnShipRegistryUpdate(SA_ShipRegistryEvent e): Event handler for ship registration events.
    - Update(): Called once per frame.
    - UnSubscribe(SA_IShip shipObj, bool deactivate): Removes a ship object from the list and scene.
    - Subscribe(SA_IShip shipObj): Adds a ship object to the list.
    - GetTarget(SA_IShip shipController, float scanRange): Performs a basic target search filter.

The `SA_Manager` class provides functionality for dynamically spawning ships, managing their registration and unregistration, and handling ship targeting based on a basic search filter. It utilizes event-based communication through the `SA_EventsBus` class and works in conjunction with ship and weapon system classes in the SpaceAI.