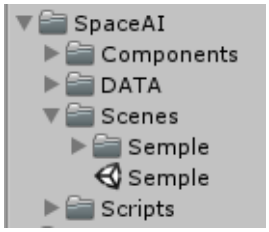
Contents of the documentation

Foreword

Before I explain how everything works here, first I want to tell why. To date, since the first release of this asset, exactly one year has passed. A year ago I started writing a game for virtual reality and could not find anywhere even a simple example of the work of a bot that uses all the dimensions for navigation. After some time (pain and suffering), I managed to build something similar to a primitive form of life in an open space, but I encountered obstacles (cubes cubes). I was interested in how they are bypassed by bots in more advanced games such as for example (x3 terran conflict or elite dangerous). Some time passed (Pain and gray hair) and I finally understood how it works, to some extent. Not ideal but very similar, to the same here there is no path finding algorithms. I did not use them because of the fact that in an open space such algorithms are not effective, they require a lot of resources, and there is no flexibility in use. As a result, when I managed to achieve the desired result, I thought that maybe I'm not the only one who can not find such examples, and decided to release it in the release. Well, perhaps it's time to explain how everything is arranged here.

First start and settings

After importing the Asset, open the scene with an example



when you start, you will see the appearance of a large number of ships, they will begin to attack each other, and simultaneously perform maneuvers deviations from obstacles. Evasion works as follows, an object (ship) with the help of the system raycast finds in itself a component of the collider (any type of collider). Sums its space and generates an indentation to the nearest point near this object. On this there is no difference in the size of the hindrance, the main thing is that it contained a component of the collider.

Configuring the bot

The bot has several components:

Root gameobject

- for movement, maneuvers and avoidance of collision is the class SA_FlightSystem
- behavior, decision-making, the search for goals is carried out by the class SA_AIController
- SA_WeaponController - This controller controls the guns of the ship, also controls the switching of guns if there are several of them on the ship. It does not contain any settings, and it must be on the root gameObject, as it does a search for all embedded GameObjects which contain the SA_WeaponLunchManager component

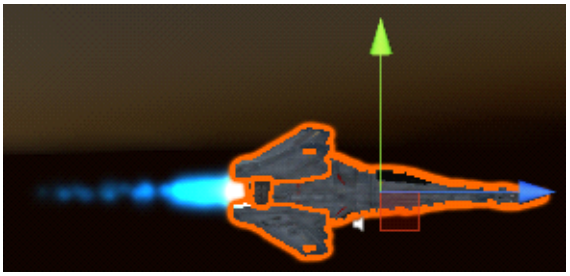
- SA_WeaponLunchManager - responsible for launching shells, and launching missiles
- SA_DamageController it is a universal component that allows you to destroy objects in scene, can be attached to any object, the object must contain a collider of any type (on a large ship turret can be destroyed). To destroy internal objects you need to activate parameter isTrigger = true;
- Rigidbody, useGravity = false; It is also desirable to assign a mass, since shells can strongly affect the flight trajectory
- AudioSource

Children gameObjects

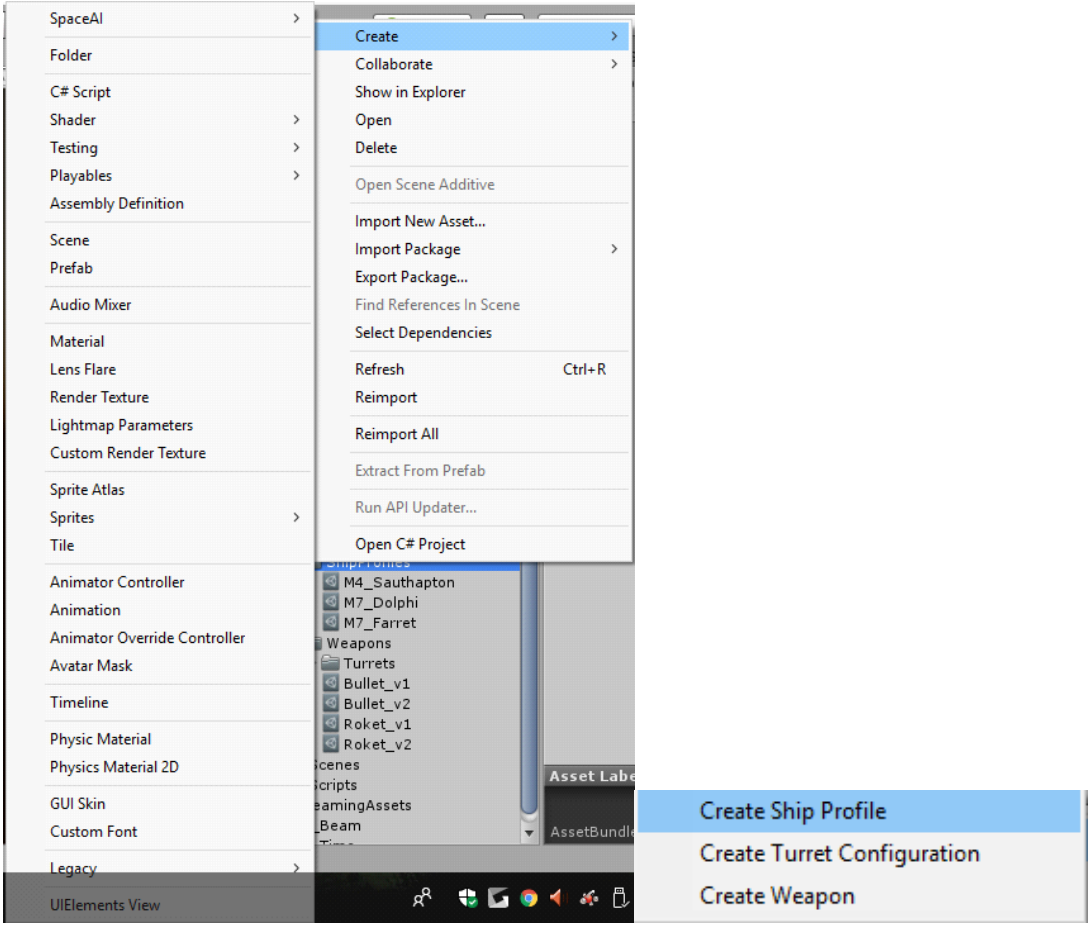
- SA_WeaponLunchManager this component must be inside the main game object, responsible for launching rockets and projectiles.

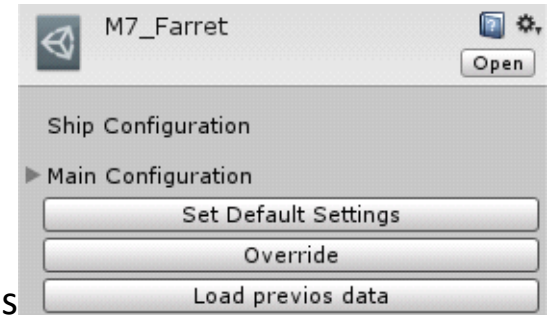
Base configuration:

- Be sure that your 3D models are placed along the z-vector, this is the main vector of the movement.

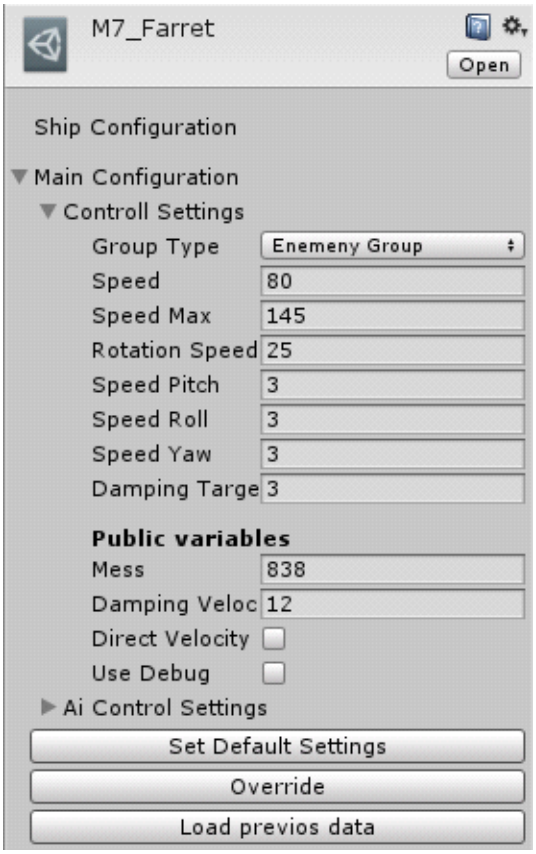


- Create a configuration file for the bot, Create->SpaceAI





the created file will look like this



The configuration file is divided into two parts, (Control Settings) is responsible for motion control, and (AI Control Settings) this group of settings is responsible for the behavior of the bot.

Control Settings:

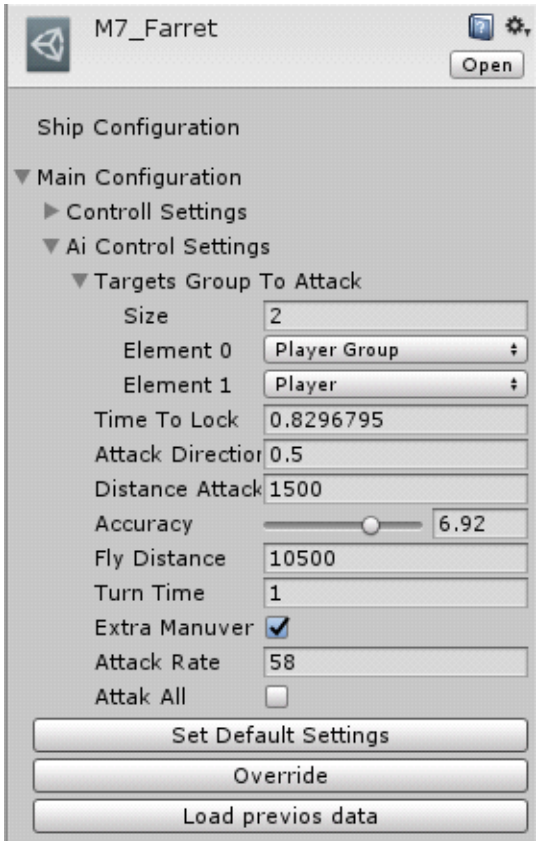
- Group Type - this is an enum variable, here you are pointing to what group of objects to treat, only four of them now, you can

add more. To do this, open the GroupType. And add the values

```
namespace SpaceAI.Core
{
   public enum GroupType
   {
      EnemenyGroup,
      NeutralGroup,
      PlayerGroup,
      Player
      // Place more groups here
   }
}
```

- Speed – average speed
- Speed Max - maximum speed, if the target is far bot changes the average speed to the maximum
- Rotation Speed - speed and sensitivity to maneuver management
- Speed Pich - speed and sensitivity to maneuver management
- Speed Roll - speed and sensitivity to maneuver management
- Speed Yaw - speed and sensitivity to maneuver management
- Damping Target - sensitivity damping to the target
- Mess - riggidbody mass
- Damping Velocity - sensitivity to velocity vector
- Direct Velocity - if true this riggidbody will not receive effect by other force.
- Debug – if true enable raycast lines, gizmos

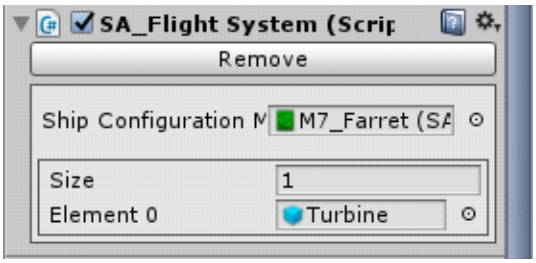
AI Control Settings:



- Targets Group To Attack – enum math, the number (fractions) that the bot can attack
- Time To Lock – deleyed lock targets
- Attack Direction - aiming vector
- Flight Distance - the bot maximum flight distance, from the center of coordinates (vector zero)
- Turn Time - evasion time
- Extra Manuver - if enabled, the bot will perform large maneuvers during the attack (affects the aiming)
- Attack Rate - priority level of the target, (from 0 to 100) the higher the higher the priority, if 100 bots will attack the target as a brutal Hulk

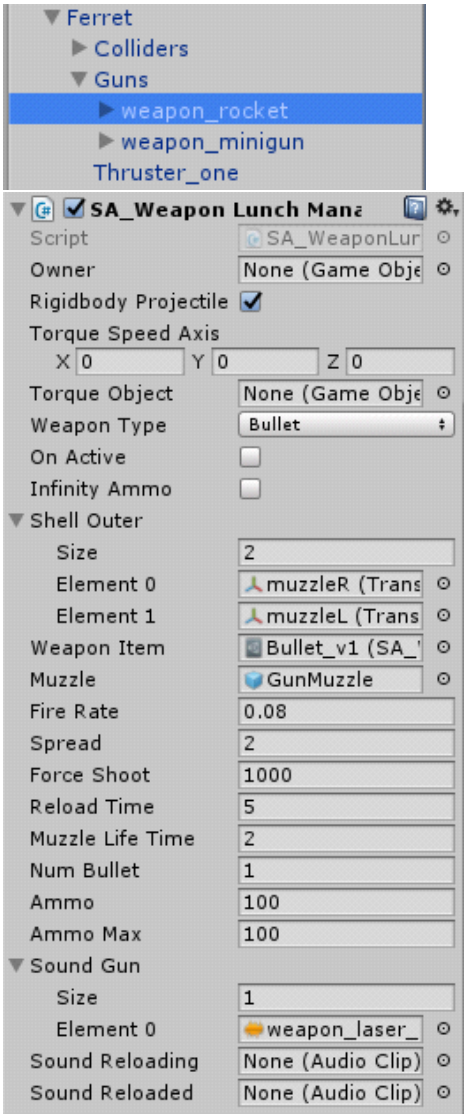
Settings after creating a file can be set to random values, when you click (Set Default Settings), you can speed up the setting.

Also, the created data can be saved to an xml file, and in case of what to load the saved data. After you configure the file, the it must be placed in the SA_FlightSystem component.



When you place this component, the necessary components are pulled automatically. The delete button, completely removes all dependent components together with SA_FlighSystem.
Array below, these are additional settings, they are optional. Control the rotation of the turbine along the z-coordinate.

- Setting up weapons, create a new empty object inside bot shipadd the SA_WeaponLunchManager component

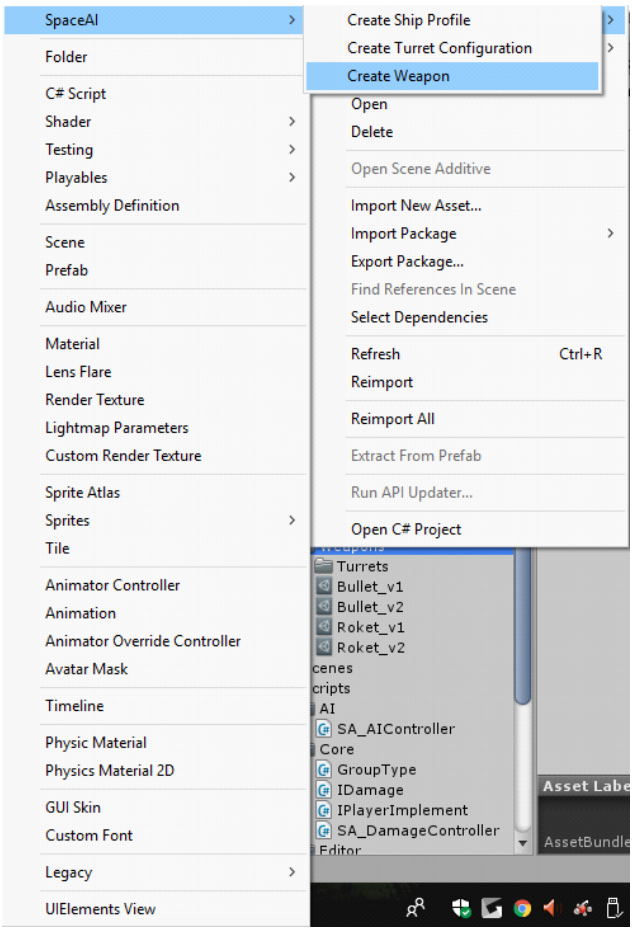


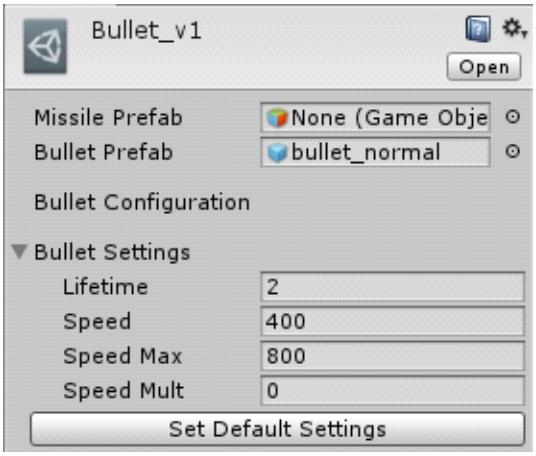
SA_WeaponLunchManager:

- Owner – varible from base class SA_DamageBase is required to simply get the data into the SA_DamageController component
- On Active -Tells the SA_WeaponController controller that it is ready. (will be hidden in future versions)
- Infinity Ammo – if true this weapon will not be recharged
- Shell Outer - points for the shot.
- Weapon Type - choice of weapon type
- Weapon Item - an object that is necessary for shooting, can be (bullets, missiles) the creation of this object will be described below.

- Muzzle - shot effect
- Fire rate - shots per second
- Spread - scattering of projectiles
- Force Shoot - speed and strength of the shot
- Reaload Time
- Num Bullet - number of projectiles from the shot point
- Ammo – current ammo capacity
- Max Ammo - maximum ammunition quantity
- Sound Gun - set of gunshot sounds
- Sound Realoding - if there is sound, the Reaload Time will be the length of the audio track
- Sound Realoded – realoded sound fx

To create a type of weapon, for example, shells you need to create a file, (scriptable object)





In Bullet Prefab put a prefab that contains the SA_Bullet or SA_Missile or SA_MissileTypes component you will see a class with a small number of settings (at the moment this system is in the base version). After place it in the variable of the SA_WeaponLunchManager

component **Weapon Item** do not forget to specify the type of object (missile, bullets)

This completes the bot setup.

Player introduction

If the player's controller already exists or not, you need to add an interface and implement its function.

```
using UnityEngine;
using SpaceAI.Core; // It contains the main components
using System;

public class PlayerImplementationExample: MonoBehaviour, IPlayerImplement // interface, for the introduction of the player
    {
        public GroupType playerType; // This variable is necessary to implement the interface
             /// <summary>
            /// This is an interface function, it returns a value (enum GroupType)
            /// </summary>
            /// <returns></returns>

        public GroupType ImplementPlayer()    {
            return playerType;
        }
}
```

Also it is necessary to specify a tag for the player, by default "Player" the basic implementation is complete, the player is ready. Also on the player's object, you can add the following components:

- SA_WeaponController
- SA_WeaponLunchManager
- SA_DamageController

To use the weapon system by the player, configure it as well as in the settings for the bot, after you need to add the following code:

using SpaceAI.WeaponSystem;

```
public  SA_WeaponController  weaponController;

void  Start()
{
    weaponController=GetComponent<SA_WeaponController();
GameObject.tag=SpaceAI.FlightSystemModule.SA_AIController.SHIPS_S
ERCH_TAG1;
}

public void Fire()
{
    bool fireAGun = Input.GetKey(KeyCode.Mouse0);
    if (fireAGun)
    {
        weaponController.LaunchWeapon();
    }
```

```
}

public void Update()
{
    Fire();
}
```

Also, if you want, you can use the autopilot system of bots by the player (not tested) the description will be below. In this Asset there is no implementation of the player, there is only an example that is also used for testing.

Structure of components/classes

API documentation in progress.