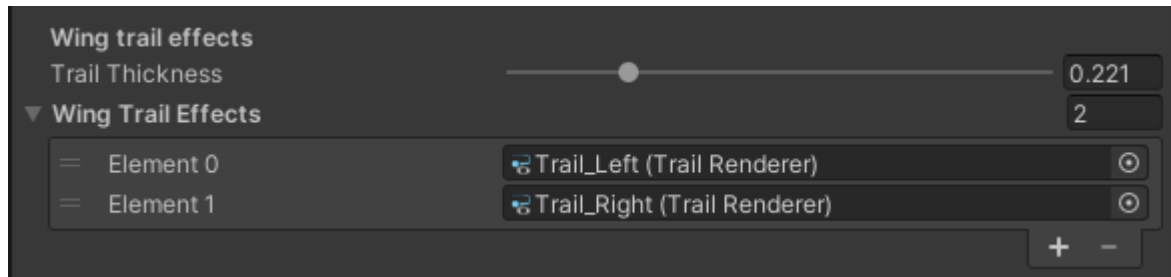

AIRPLANE CONTROLLER DOCUMENTATION

CONTENTS

Controller Editor Values	1
Wing Effects	1
Movement.....	1
Sound.....	2
Propellers and turbine light	2
Colliders	3
Sideways force	4
Turbo heat values.....	4
How To Use.....	5
How To Use In Code	5
Airplane	5
Airplane Inputs.....	6
Runway	7
Video Tutorials	7
• Overview YouTube link	7
• Custom airplane YouTube link.....	7
• How to make hit effect YouTube link	7
• New Flying and runway update YouTube link.....	7

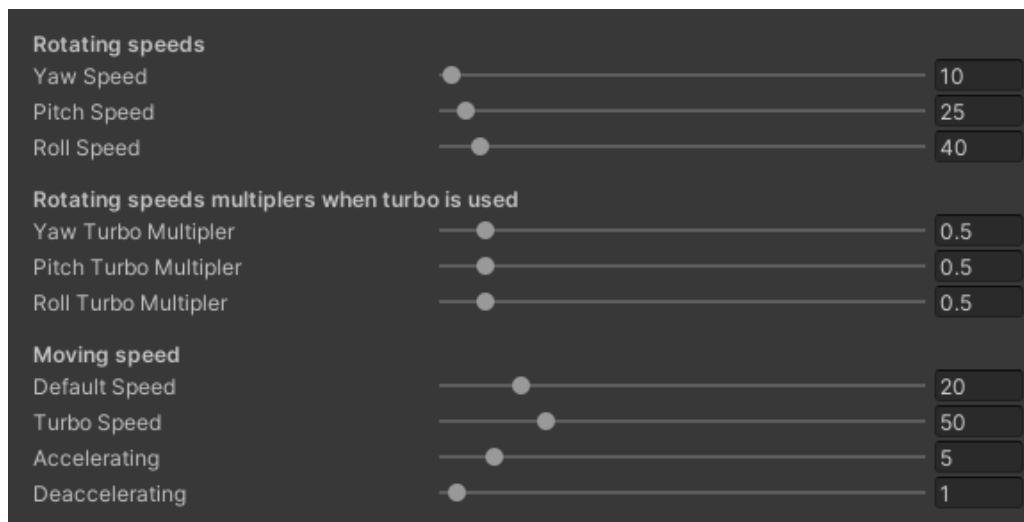
CONTROLLER EDITOR VALUES

WING EFFECTS

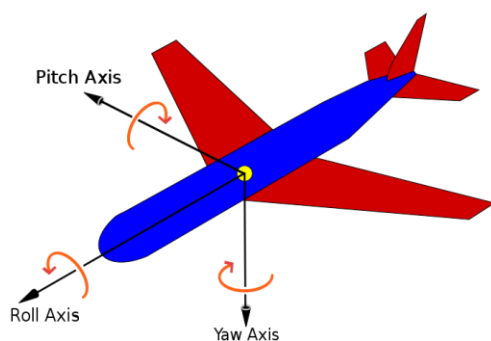


In this list you can put the trail renderers references, when the turbo is used these trails appear as thick as the trail thickness slider determines them.

MOVEMENT

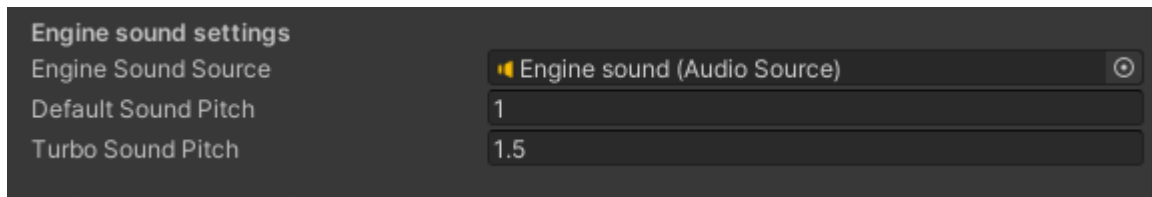


These adjustments define the turning and movement speed of the aircraft. Test different values to find out the best settings for your use.

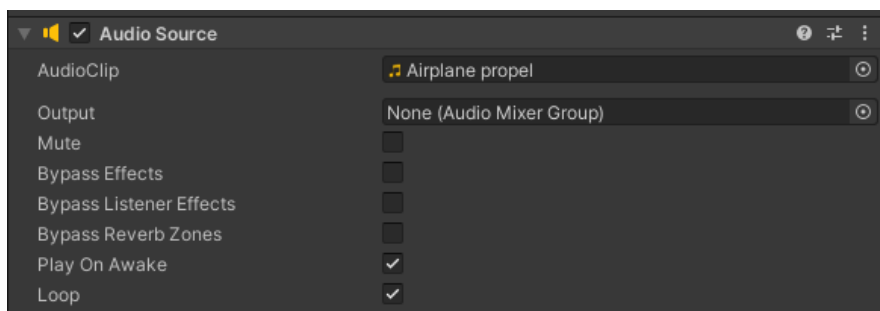


[Image source.](#)

SOUND

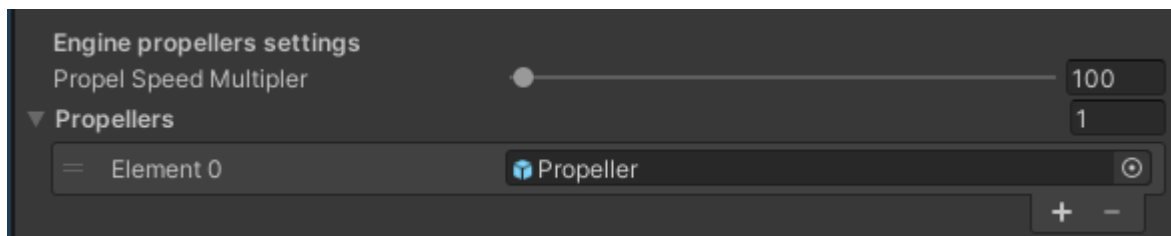


The audio source component reference of the aircraft is placed here. The script automatically adjusts the pitch according to the set values.

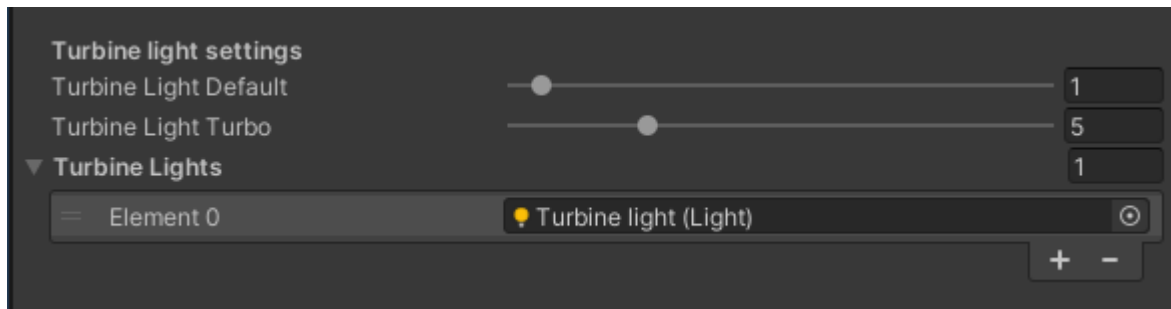


An example of an audio source component. Play is awake and loop Booleans must be on, I recommend using only looping sounds.

PROPELLERS AND TURBINE LIGHT

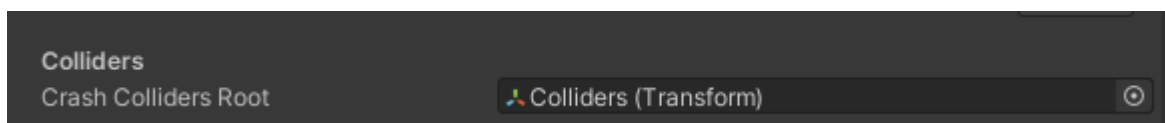


All propeller references are placed here, the code rotates all propellers around the Z axis according to the propel speed multiplier value and current speed. The airplane does not necessarily have to have propellers, but if so, this array must be empty.

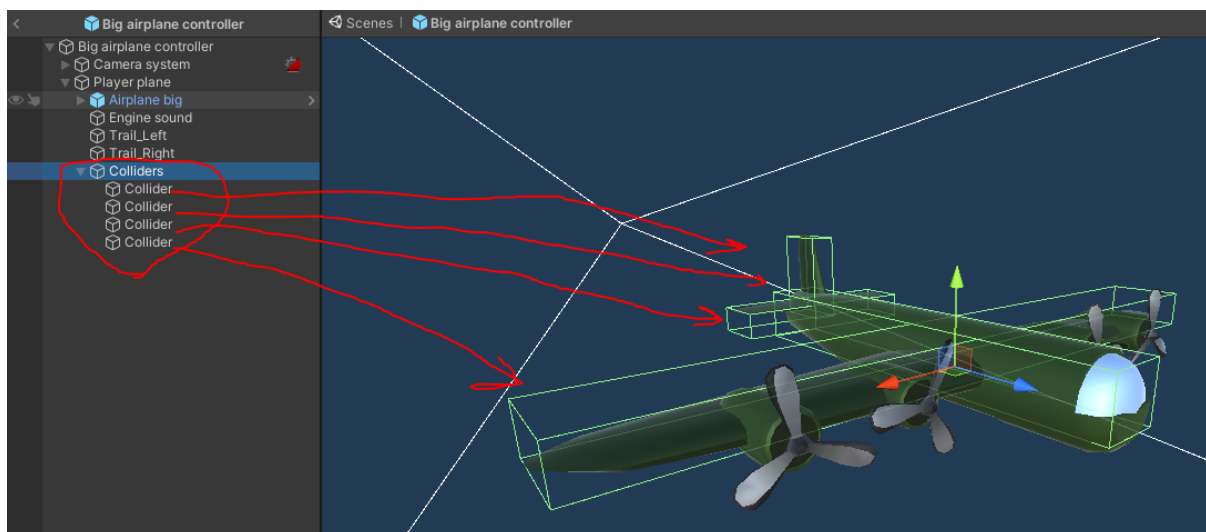


If the aircraft does not use propellers, all the turbine lights references can be put in this list. The code changes the intensity of the lights according to the set values. The airplane does not necessarily have to have turbine lights, but if so, this array must be empty.

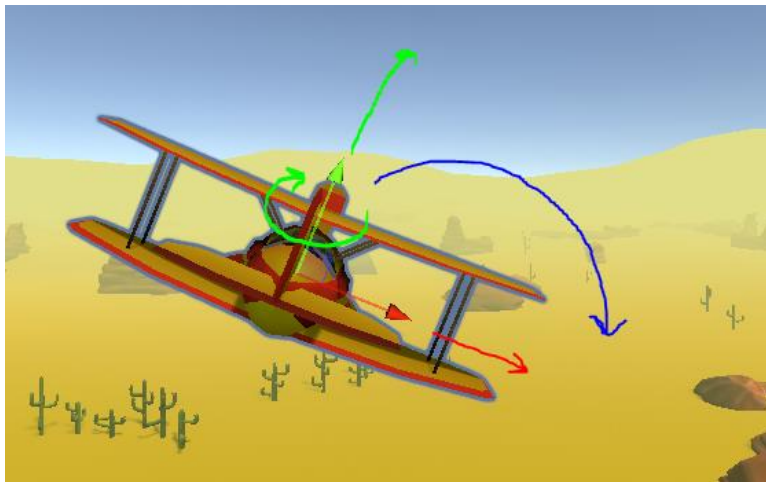
COLLIDERS



A reference to the object that contains all the aircraft's colliders is put here. All colliders must be primitive shape. The code converts all these colliders into triggers, adds a kinematic rigid body and **SimpleAirPlaneCollider** component to them. If one of these triggers even touches something with a collider, the **Crash** function is called.



SIDEWAYS FORCE



Sideway force	
Sideways Movement	15
Sideways Movement X Rot	0.012
Sideways Movement Y Rot	0.93
Sideways Movement Y Pos	0.046

When the aircraft is tilted to the right or left, these values can be used to adjust how much it affects the turning of the aircraft.

The best way to understand these values, is when you play the game and at the same time test different adjustments.

If you do not want these features in your aircraft, set the **Sideways Movement** value to zero.

TURBO HEAT VALUES

Turbo settings	
Turbo Heating Speed	0
Turbo Cooldown Speed	0
Turbo heat values	
Turbo Heat	0
Turbo Overheat Over	0
Turbo Overheat	<input type="checkbox"/>

These adjustments are self-explanatory, but let's explain them a little.

In the **Turbo settings**, you can fine-tune the turbo's heating and cooling rates. The **Turbo Heat Values** section provides real-time information about the turbo's current temperature (do not change in the editor) and **Turbo Overheat** boolean indicates whether it has reached an overheating state.

You can set the **Turbo Overheat Over** value to determine when the turbo should cease overheating and become operational again.

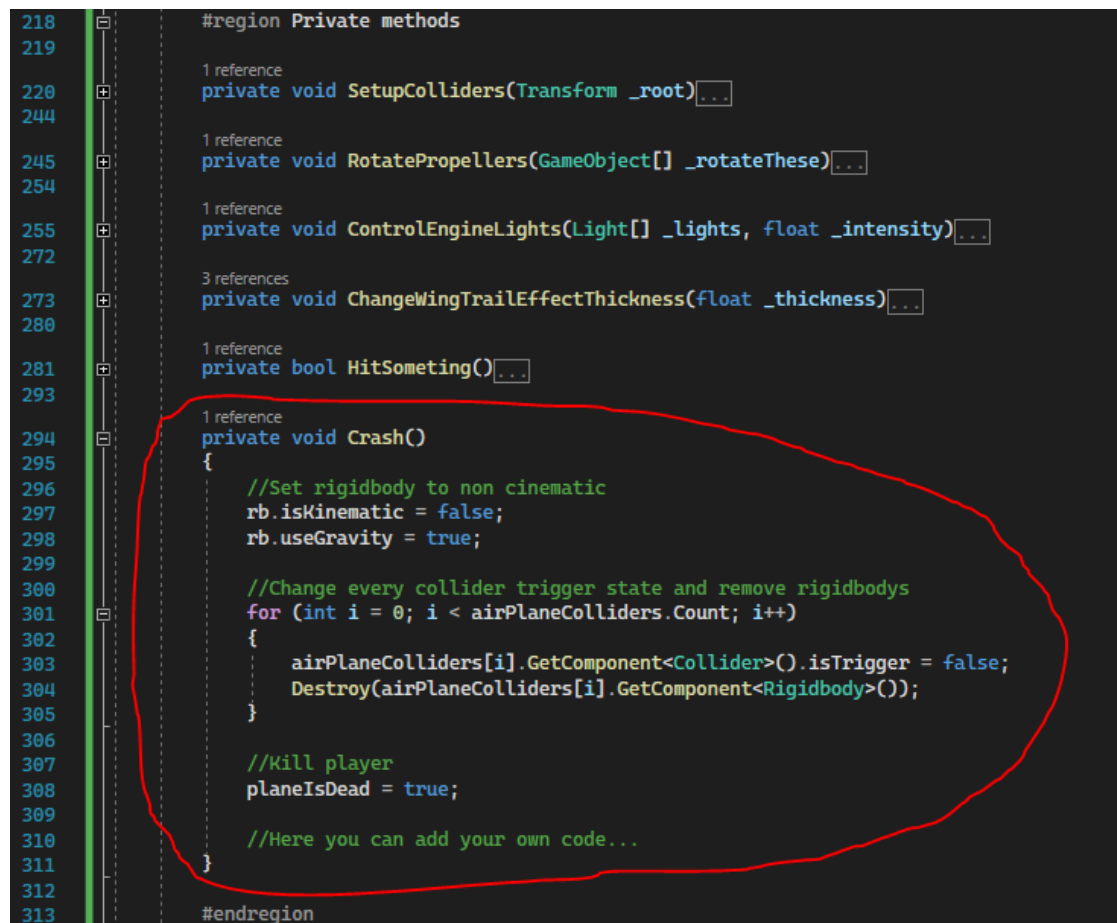
How To Use

From the folder **HeneGames/Simple Airplane Controller/Prefabs/Controllers** You can find all examples of airplane controllers, drag one of these prefabs into one of your own scenes and you're ready to fly (Make sure that there are no cameras or audio listener components in your scene).

The asset also contains example scenes where you can test airplanes.

How To Use IN CODE

AIRPLANE



```
218 #region Private methods
219
220 1 reference
220 private void SetupColliders(Transform _root)[...]
244
245 1 reference
245 private void RotatePropellers(GameObject[] _rotateThese)[...]
254
255 1 reference
255 private void ControlEngineLights(Light[] _lights, float _intensity)[...]
272
273 3 references
273 private void ChangeWingTrailEffectThickness(float _thickness)[...]
280
281 1 reference
281 private bool HitSometing()[...]
293
294 1 reference
294 private void Crash()
295 {
296     //Set rigidbody to non cinematic
297     rb.isKinematic = false;
298     rb.useGravity = true;
299
300     //Change every collider trigger state and remove rigidbodys
301     for (int i = 0; i < airPlaneColliders.Count; i++)
302     {
303         airPlaneColliders[i].GetComponent<Collider>().isTrigger = false;
304         Destroy(airPlaneColliders[i].GetComponent<Rigidbody>());
305     }
306
307     //Kill player
308     planeIsDead = true;
309
310     //Here you can add your own code...
311 }
312
313 #endregion
```

In the Private methods region, there is a function named **Crash**, this function is called if **HitSometing** Boolean is true. The function stops the airplane now, but you can add your own functionalities to this, maybe some kind of explosion effect (see the video tutorials).

```

580 | #region Variables
581 |
582 |     /// <summary>
583 |     /// Returns a percentage of how fast the current speed is from the maximum speed between 0 and 1
584 |     /// </summary>
585 |     /// <returns></returns>
586 |     0 references
587 |     public float PercentToMaxSpeed()...
588 |
589 |     1 reference
590 |     public bool PlaneIsDead()...
591 |
592 |     0 references
593 |     public bool UsingTurbo()...
594 |
595 |     0 references
596 |     public float CurrentSpeed()...
597 |
598 |     /// <summary>
599 |     /// Returns a turbo heat between 0 and 100
600 |     /// </summary>
601 |     /// <returns></returns>
602 |     0 references
603 |     public float TurboHeatValue()...
604 |
605 |     1 reference
606 |     public bool TurboOverheating()...
607 |
608 | #endregion

```

Here you can also find convenient functions.

AIRPLANE INPUTS

```

566 | #region Inputs
567 |
568 |     1 reference
569 |     private void HandleInputs()
570 |     {
571 |         //Rotate inputs
572 |         inputH = Input.GetAxis("Horizontal");
573 |         inputV = Input.GetAxis("Vertical");
574 |
575 |         //Yaw axis inputs
576 |         inputYawLeft = Input.GetKey(KeyCode.Q);
577 |         inputYawRight = Input.GetKey(KeyCode.E);
578 |
579 |         //Turbo
580 |         inputTurbo = Input.GetKey(KeyCode.LeftShift);
581 |     }
582 | #endregion

```

You can find the input handling in the **SimpleAirplaneController** script.

RUNWAY

```
Unity Script (1 asset reference) | 0 references
public class RunwayUIManager : MonoBehaviour
{
    [SerializeField] private Runway runway;
    [SerializeField] private TextMeshProUGUI debugText;
    [SerializeField] private GameObject uiContent;

    Unity Message | 0 references
    private void Update()
    {
        if(runway.AirplaneIsLanding())
        {
            uiContent.SetActive(true);
            debugText.text = "Airplane is landing";
        }
        else if(runway.AirplaneLandingCompleted())
        {
            uiContent.SetActive(true);
            debugText.text = "Press space to launch";
        }
        else if(runway.AriplaneIsTakingOff())
        {
            uiContent.SetActive(true);
            debugText.text = "Airplane is taking off";
        }
        else
        {
            uiContent.SetActive(false);
            debugText.text = "";
        }
    }
}
```

RunwayUIManager code shows an example of what you can ask from the runway. Using these functions, you can conveniently create your own logic.

VIDEO TUTORIALS

- OVERVIEW [YOUTUBE LINK](#)
- CUSTOM AIRPLANE [YOUTUBE LINK](#)
- HOW TO MAKE HIT EFFECT [YOUTUBE LINK](#)
- NEW FLYING AND RUNWAY UPDATE [YOUTUBE LINK](#)