2D Space Shooter Kit Documentation

About this Starter Kit

This starter kit was designed to provide completely custom, and customizable examples for learning or getting a head start on building 2D Space Shooters. Read through the scripts, add on to them, and write new ones to build your own custom 2D Space Shooter.

All of the assets included in this file are free to use, but we recommend creating your own media to make your game unique. All of the scripts can be used as-is but we also recommend using them as a reference or build onto to create unique and great games.

---

1) Preparing Scenes for Building

To be able to compile the game you must add all of the scenes to the build settings. When adding scenes to the Build Settings, make sure the scene "Loader" is at the top of the list in Build Settings.

To do this, double click on the scene "Loader", then in Unity go to File>Build Settings...

Then click "Add Current". Once Loader is added, then do this same process with the rest of the scenes (Menu, Game, Game2). Order doesn't matter at this point as long as the scene "Loader" is first. You can create as many levels as you want, but you have to add them to the build settings as well.

2) Building for Mobile

2d Space Shooter was designed to run in landscape. To to make sure that the game runs in landscape, go to File>Build Settings...

Click on which mobile platform you want to build to (iOS or Android), then click Switch Platform.

After this, click Player Settings... On the right, options for player settings will pop up.

Click on the Android or iOS tab (depending on which one you chose) below "Per-Platform Settings".

Click on the drop-down menu to the right of Default Orientation* and choose Auto Rotation. Then options below will appear to choose which orientations to allow. Make sure only Landscape Right, and Landscape Left are chosen.

3) Building for Web/Standalone

You do not need to tamper with orientation like mobile for web/standalone builds. Make sure to follow the steps from "Preparing Scenes for Building". Otherwise you are will be ready to build as many versions for web/standalone as you'd like!

4) Inputs and Controls

- Web and Desktop Controls -
The ship's controls to move up, down, left, and right reside in playercontrols.js. The default controls for Web and Desktop builds are both WASD, Up Down Left Right. These can be changed in the playercontrols.js script. Notice there are 2 iterations of these controls for #if UNITY_WEB and UNITY_STANDALONE. What this does is it will only compile those pieces of the script, depending on what platform you have your build set to.

The ship fires bullets through the script weaponsystem.js. The default for firing bullets is Spacebar for both Web and Desktop builds. These can easily be changed in weaponsystem.js. Notice there are also 2 iterations of shooting controls in #if UNITY_WEB and UNITY_STANDALONE. The reasons are the same as described above.

- Mobile Controls -
Mobile Controls are also in playercontrols.js with 2 iterations within #if UNITY_ANDROID and UNITY_IOS. These are almost identical, aside from 1 numerical difference in Mathf.SmoothStep. These controls can be edited in any way you'd like, but by default it translates the touch to the cameras world view.

The ship on mobile versions fires bullets through the script weaponsystem.js. To change how the bullets fire for mobile, they can be changed there in both iterations of #if UNITY_IOS and UNITY_ANDROID.

- 1axis and 2axis ship controls. We set this up as a public variable that can be easily changed in the playercontrols.js script in the inspector. It is a true/false statement that can be checked or unchecked, depending on which style of ship controls you'd like your game to use.

5) Simple level system

We built a simple example of how a leveling system can work in a space shooter. The GUI assets that are affected by leveling up are GUI/leveltext (a GUIText), and GUI/expbar (a GUITexture), found in the game scene's hierarchy. These are attached to the experiencemanager.js script and are changed once the experiencemanager.js script receives a certain amount of experience orbs.

Once an orb hits the player, experiencemanager.js recognizes this in the OnTriggerEnter function and proceeds to add 1 experience point to the variable expAmount, then destroys the orb so it can't any more to expAmount.

Once enough experience is collected, experiencemanager.js will update the blue bar at the top based on current exp, and increase the leveltext by 1 (it says Lvl 1 at the top). After this it will send a message that weaponsystem.js receives to accomodate for the change in level, and have the ability to update how it shoots bullets accordingly.

This system is not required to be used, but is a good example of how a simple leveling system for a space shooter can keep the player engaged and working towards making their ship more powerful.

Contact Us

If you have any issues, questions, or suggestions, please do not hesitate to contact us! Email us at any time at support@cinoptstudios.com and we will get back to you as soon as we can.