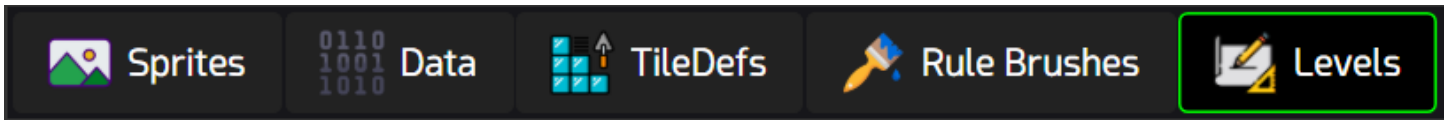


PROJECT WORKFLOW

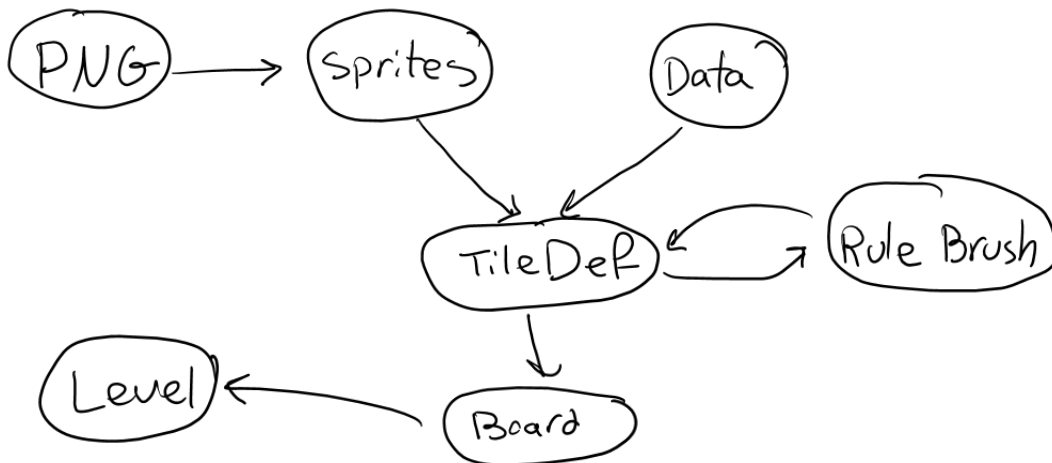
The workflow in Ed tile editor is organized into five sections, available as tabs on the top-left corner of the user interface:



Every step in the workflow serves as a way to create and customize the puzzle pieces that are eventually assembled next to each other in the final tab: The Level Editor.

The usual editing flow goes like this:

1. Import your image files as sprite sheets using a simple drag & drop.
2. If you have multiple sprites in a single file (sprite sheet,) slice them into sprites.
3. Define Tile Definitions (TileDefs) by configuring sprites, animations, sizes and data.
4. Define smart Rule Brushes that automatically choose different TileDefs based on the neighboring tiles.
5. Draw TileDefs on Levels.

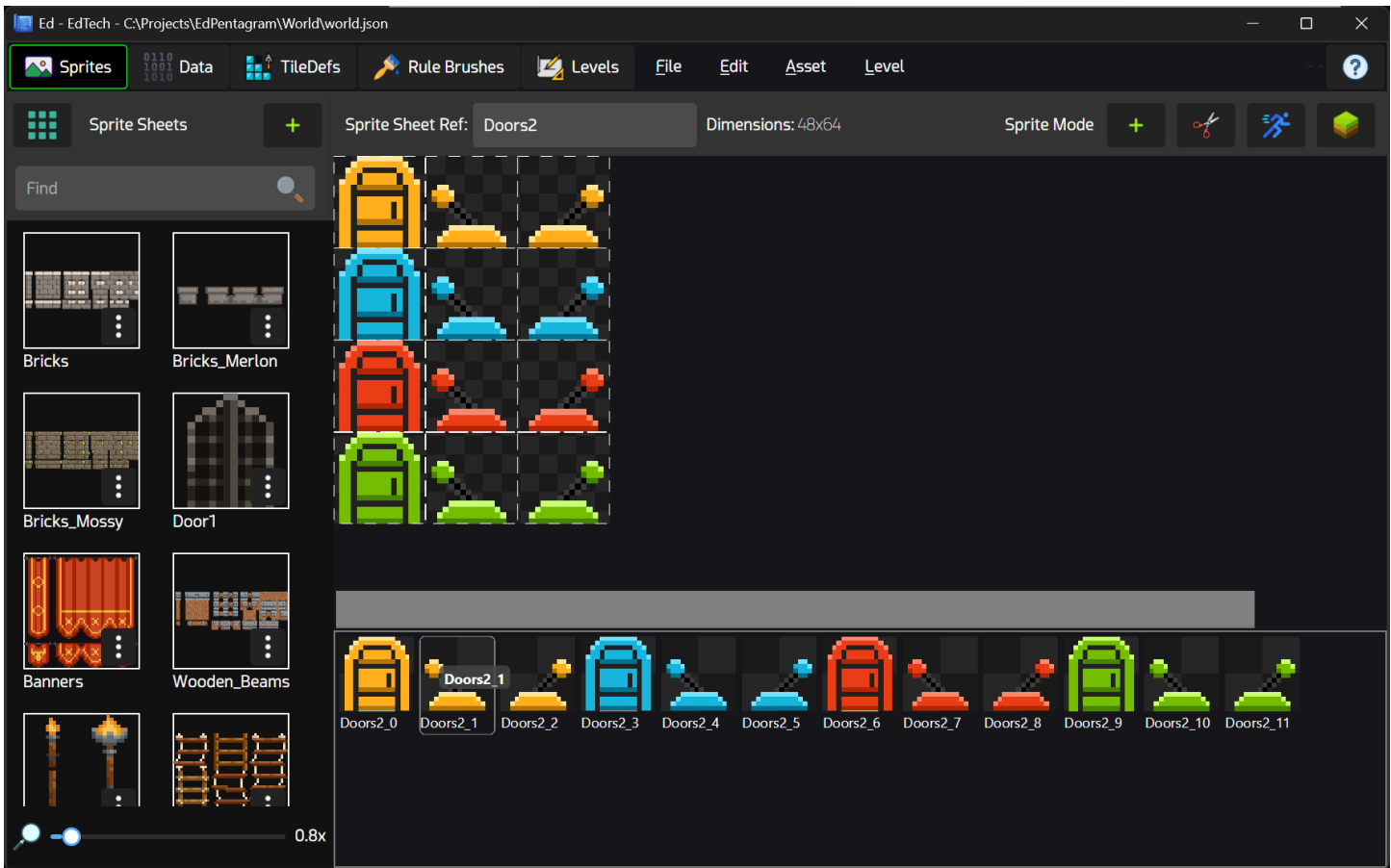


Levels can have multiple boards (tile grids) where TileDefs can be drawn on. The entire project, individual levels, or even individual boards can be exported to various easy-to-parse formats that can be loaded in any game engine easily:

- JSON,
- ASCII Text (Verbose),
- ASCII Text with CSV grids (of characters, integer handles, or hexadecimal handles).


Aside from exporting to text, Ed supports Unity as a first-class target, exporting sliced sprite sheets, levels, grids and scripts into ready-to-run Unity files. You can spawn Ed levels in Unity with one click.


SPRITE EDITOR



Sprite Editor is where you import image files for sprite sheets, and slice them into sprites, that can be assigned to TileDefs and drawn on levels.

You can always switch to Sprite Editor by clicking on the first tab button on the top left corner of the app, or by pressing F1.

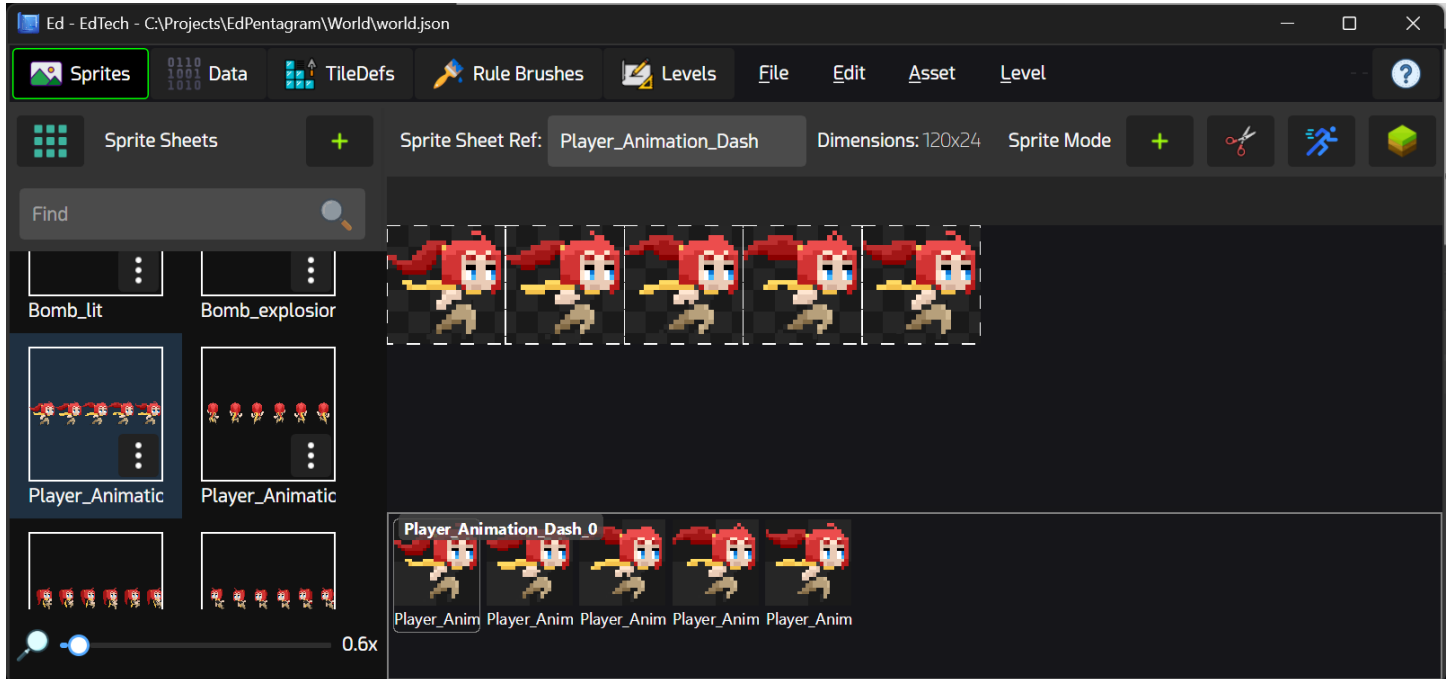
You can import image files by drag & dropping them onto the app or clicking the  button in front of *Sprite Sheets*. By default, when you import an image file, a sprite covering the entire image is created.


If your file is a sprite sheet, you should slice it into separate sprites. The auto slice tool  allows you to do that quickly by setting a default sprite size, or sprite count within the sprite sheet.

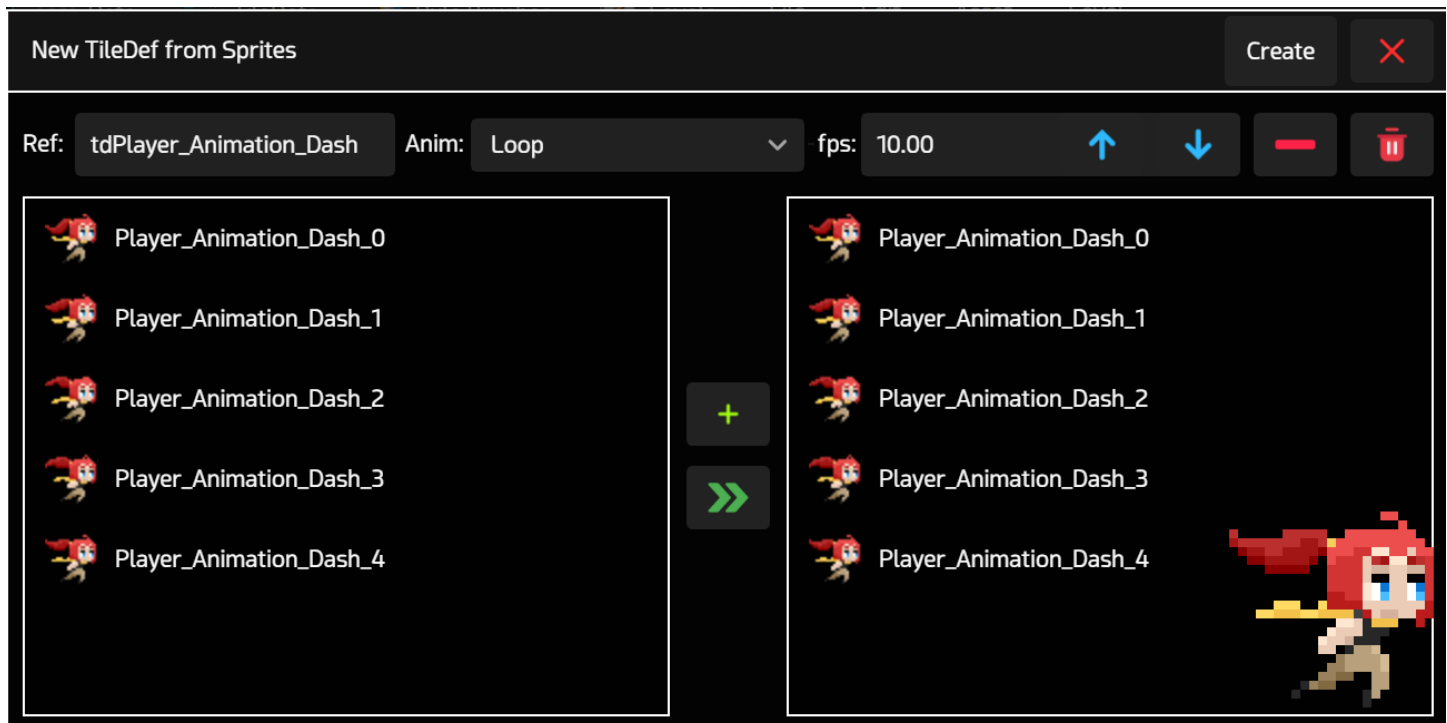
Once you have your sprites set up, you can immediately start drawing by switching to the Level Editor tab.

ANIMATIONS

Animations are defined as a set of keyframes of sprites in a TileDef.




To quickly create an animation from a sprite sheet, select your sprite sheet in the Sprites Editor (F1) and click the  button on the top right corner of the app to open the New TileDef dialog box.

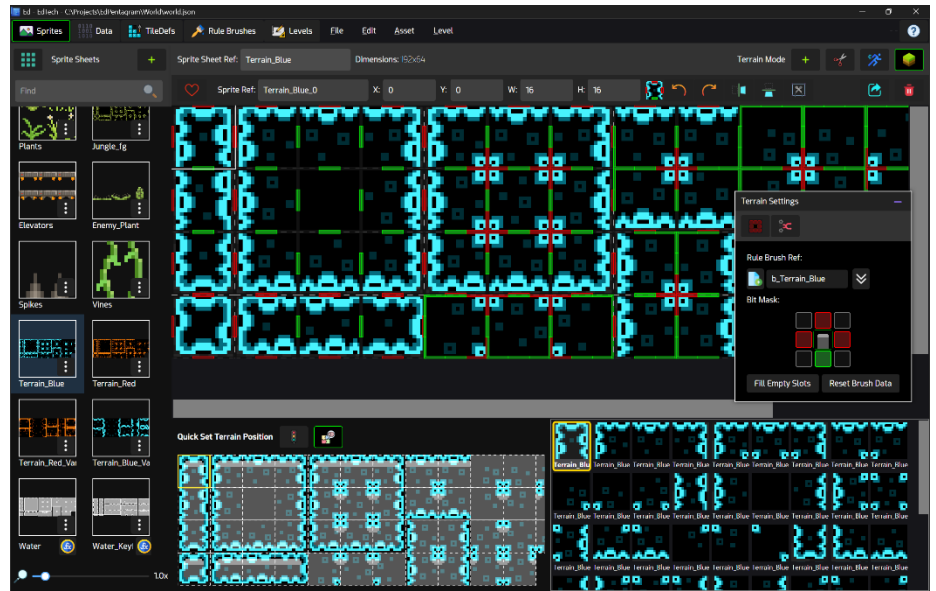




Select the sprites from the right box and add them to the list of key frames (left box,) give it a name (Ref) and press Create. Now you can draw this animated TileDef on the level.

TERRAIN RULE BRUSHES

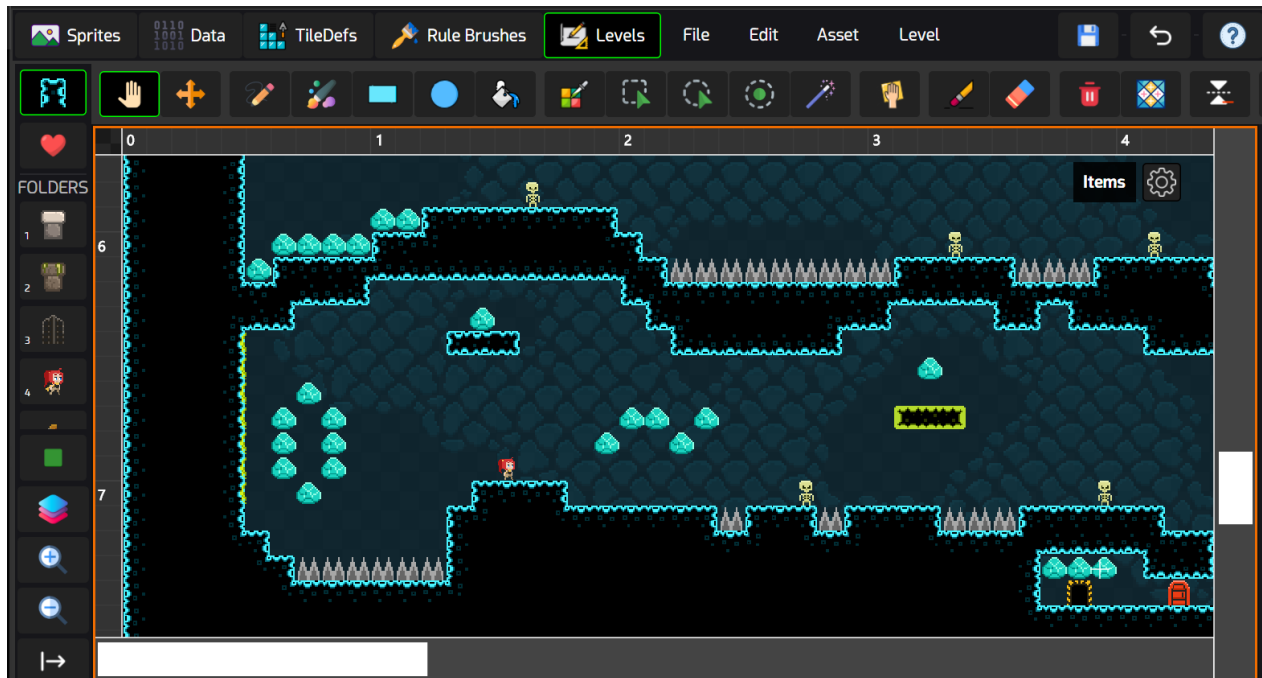
The Quick Terrain toolset allows you to quickly mark specific corner sprites of a terrain tile set and generate rule brushes from them. To activate Quick Terrain mode in the Sprite Sheet Editor (F1), click on the  icon on the top right corner of the interface.

It opens two panels: *Terrain Settings* on the right side, and *Quick Set Terrain Position* on the bottom.



To start a new rule brush, click on the  button in the *Terrain Settings* dialog box, or manually enter a new brush name. Now simply click on a sprite on the top part, and then click on the associated corners on the *Quick Set Terrain Position* panel. Once you have assembled all pieces of the terrain rule brush, click  to generate rule brushes from the marked sprites.

Once the rule brushes are generated, you can go back to the Level Editor (F5), select the new rule brush, and start drawing on a board. TileDefs are automatically selected based on their neighbors.



SAVING THE PROJECT – DIRECTORY STRUCTURE

There are three ways to save an Ed project: to a directory, as a package, or in the internal library.

SAVE TO DIRECTORY

This is the recommended approach for working with Ed on desktop, and especially when collaborating in teams and committing to git. When you save to a directory, Ed creates the following file structure:

- **world.json:** This file contains the base data of your project and is placed in the destination directory's root.
- **TileDefs directory:** This directory contains a single JSON file per TileDef created.
- **Assets directory:** The image files are placed here. If you encounter missing bitmaps (shown in clay-pink color), you can manually move your image files to this folder and restart the app.
- **world.jsons directory:** Other data related to rule brushes, folders, etc. are placed as JSON files in this directory. Levels and boards are also stored here, in JSON.
- **Thumbnails directory:** This directory contains thumbnails associated with various assets. You don't need to push this to git.

If you collaborate with others, you can add **world.state.json** to `.gitignore`, so your user preferences are kept local. The **Thumbnails** directory can also be ignored.

SAVE AS PACKAGE (.MOOSEED FILE)

When you save as a package, you get a single file containing all the data mentioned above, in an ordinary zip format, with a `.mooseed` extension. This is handy when you move files around on iPads and iPhones or don't work in a collaborative environment. You can rename this to `.zip` and extract it at any time.

SAVE TO INTERNAL LIBRARY

Ed creates a folder in your user's Documents folder (on desktop) and in a sandbox on iOS containing auto-saves and internal library files. Every few minutes, Ed creates an auto-save for the open project that is stored in the internal library, and also available in the Documents/MooseEd folder.

You can also manually quick-save projects to the internal library and load them easily without having to deal with file system. This is another ideal way to work with temporary projects on iOS.

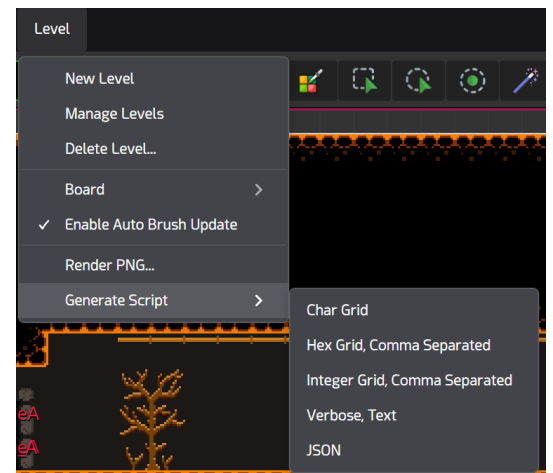
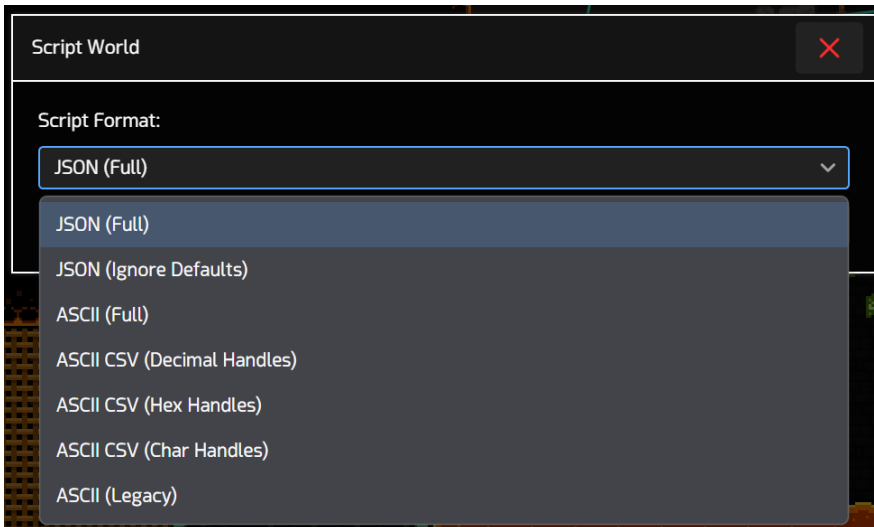
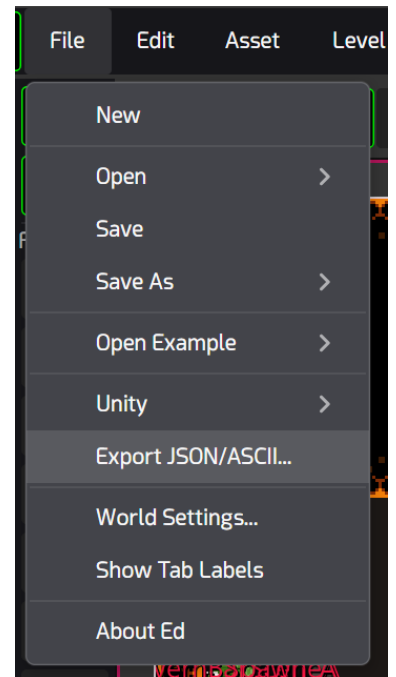
EXPORT TO JSON/ASCII

You can export the entire project, single levels, or even boards individually, as JSON or plain-text ASCII files. Grid data can be stored as verbose line-by-line tile definitions, or CSVs with integers, hex, or characters, representing TileDefs.

To export the entire project to text:

Use the **File > Export JSON/ASCII...** menu option. You'll get a list of various text formats you can choose to export your project in.

If you choose any of the ASCII formats, you'll get to configure the indentation and whether you want the full project metadata (including sprite sheets, rule brushes and so on) to also be exported. Exporting all the data is usually not necessary for running the actual game.



To export an individual level or board to text:

Use the **Level > Generate Script** menu and select your desired format to export an entire level. To export just the selected board, use the **Level > Board > Generate Script** instead.

The most complete representation of your project can be obtained when you export the entire project as JSON (Full) or JSON (Ignore Defaults). Combining this file with what's inside the Assets folder, you have everything you need to load all the data in your favorite game engine.

EXPORT TO UNITY

Ed supports Unity as a first-class target game engine. If you use Unity, you can simply rely on the one-click Unity export functionality in Ed to quickly generate Unity asset and code files and iteratively build your game.

To export to Unity, use the *File > Unity > Settings...* menu option. Once you have your Unity export preferences configured, you can use the *Quick Export to Directory* to generate Unity files with a single click.

The *Unity Export Options* dialog provides various checkboxes and settings to control what is exported and how, but you can rely on the default settings for an optimal experience.

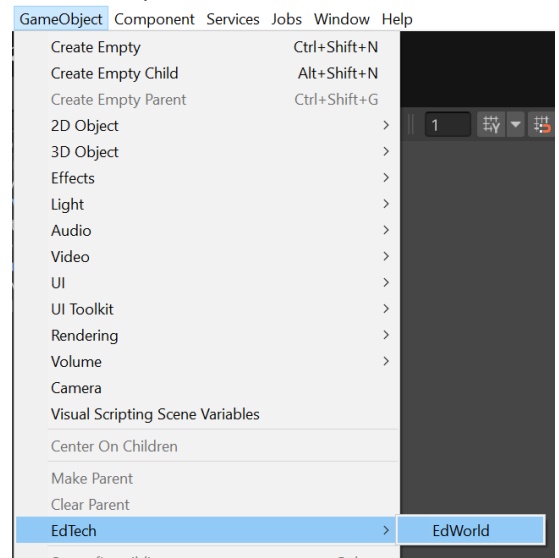
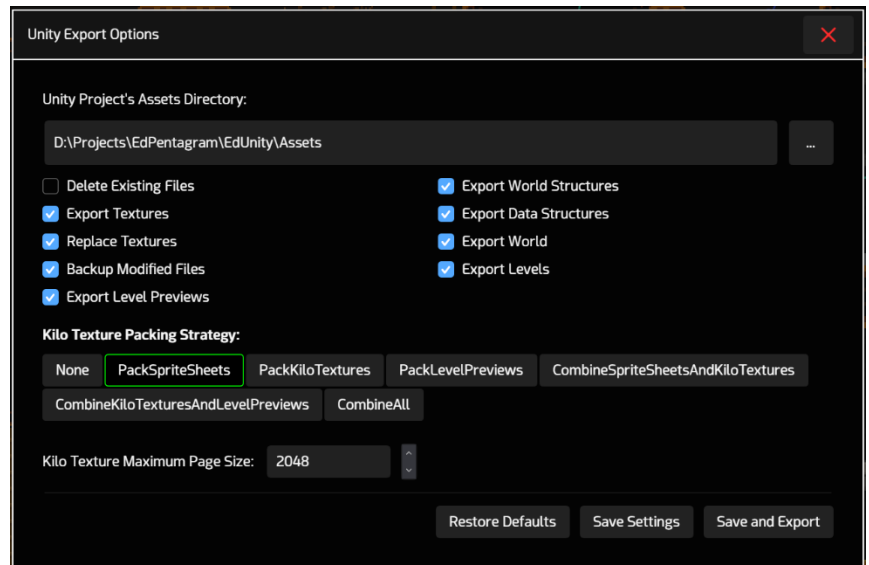
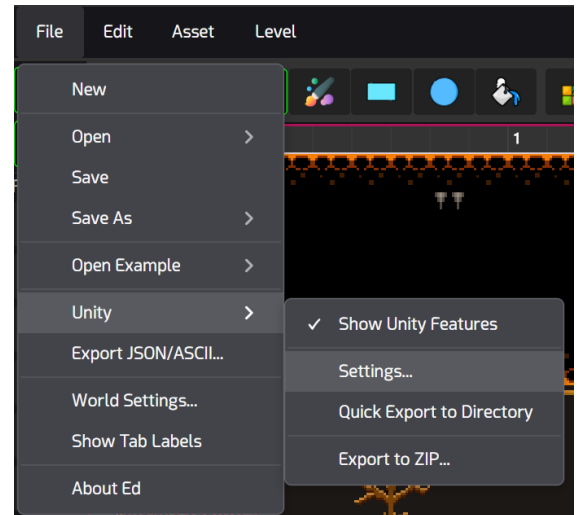
Simply point to the *Assets* directory of your Unity project and click *Save and Export*.

When you open your Unity project, you will see a new menu called *EdTech* appearing under the *GameObject* creation menu. Use it to add an *EdWorld* to your scene.

The *EdWorld* object manages Ed data and can spawn levels and tiles on the scene, at edit or run time.

Another feature of EdTech for Unity is supporting level previews: To keep things simple, but have an idea of how the Ed level looks when spawned, previews of the Ed level are spawned only at edit-time, making it easy to align Unity objects with the Ed world.

Previews automatically get destroyed in play-time and the tiles themselves are spawned instead.



EDTECH FOR UNITY

For every level in the Ed project, the EdWorld script provides two buttons to either set that level as the starting level that's spawned on play-time, or spawn preview sprites of that level for edit-mode.

If your scene is going to be specific to that level, it will be convenient to press both buttons for the level you want.

Ed sprites are also exported and already sliced in Unity, stored in the *DefaultSpriteMap* scriptable object.

Animations are also available in Unity, and animated TileDefs play instantly and with zero additional code.

Most of the assets, including TileDefs are stored in the *EdWorldData* static class as objects, and can be directly accessed in code.

Level names are also stored as *const string* definitions in the *EdWorldData* class.

All spawned tiles will have a *Tile* script attached to them. Additionally, any data structure that's marked as *MonoBehaviour* in Ed will have a corresponding script attached to it in Unity.

For example, if you create a data structure called *Player* in Ed and assign it to your player TileDef, in Unity it will have the *Player : MonoBehaviour* script attached to it automatically. This way you can create your logic in Unity, declare the class names in Ed and assign them to TileDefs, and have a ready-to-play game every time you export from Ed to Unity.

ACCESSING SPRITES

To access a sprite by name, you can use the **EdWorldExtensions.GetUnitySpriteById(EdWorld world, string id)** static function, which returns a Unity sprite. This can be used in any place in Unity as a regular sprite, including on GUI image elements.

CALLBACK AFTER WORLD SPAWN

To have a callback after EdWorld spawns the tiles, you can set the *edWorld.PostStart* variable to a method of your choice. *PostStart* is called after everything is spawned. There should only be one EdWorld at any time in the scene, and it can be accessed via *EdWorld.Instance* static variable.

MANUALLY SPAWN A LEVEL

To manually spawn a level at any time, you can call **EdWorldExtensions.SpawnLevelById(EdWorld world, string id)**, passing the name of the level as the id.

