Corgi Pixel Physics

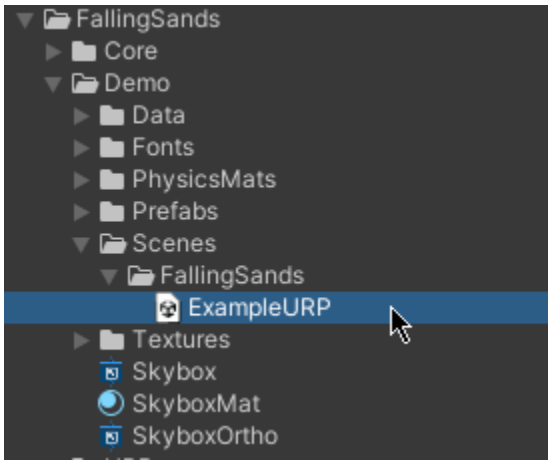
ReadMe v1.0

(latest documentation here)

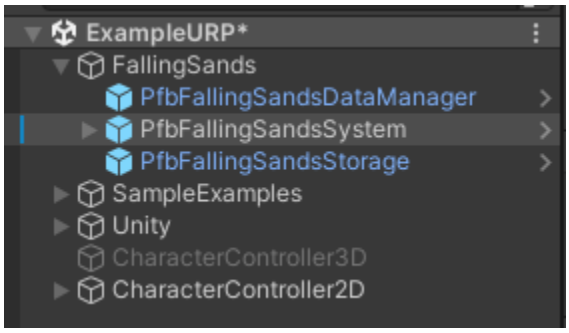
Thanks for purchasing Corgi Pixel Physics! This document will help you get started and explain how everything works. If you have any questions, see the contact me section at the end.

Getting Started

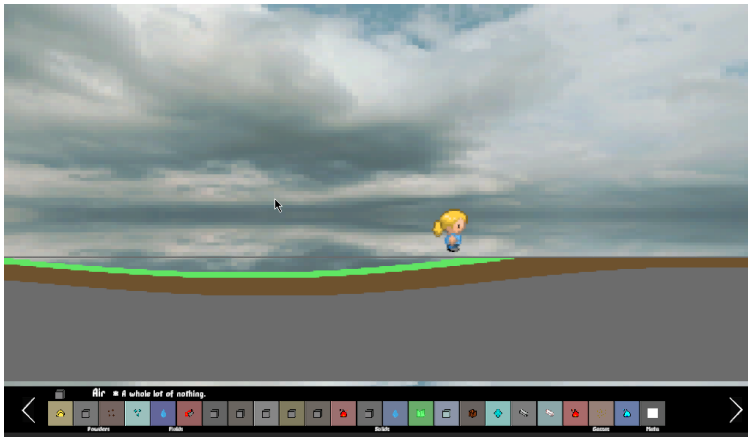
To quickly get started and try out the plugin, navigate to Demo/Scenes/ExampleURP.
Note: the demo scene is currently only set up to work in URP. The asset itself does not necessarily require URP, though.



Once the scene is open, you can find the plugin-specific prefabs here:

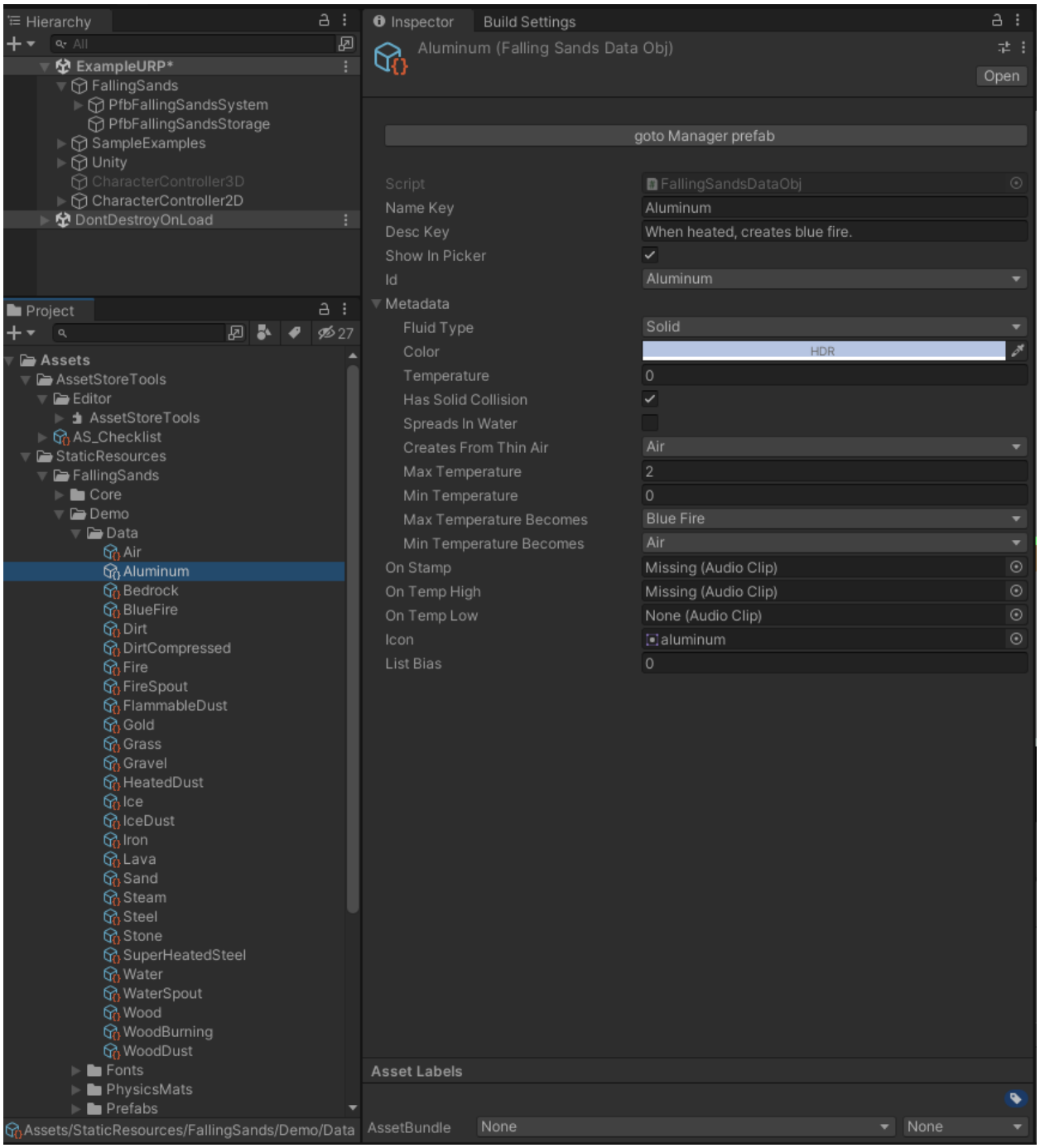


Hit play and try it out! The controls are WASD + Space to move. Arrow keys move the camera. Mouse left-click will select materials and place them in the world.



The Material System

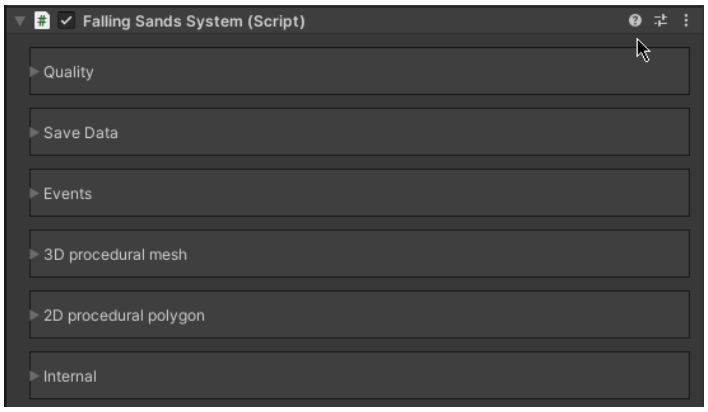
All pixels have physics. Each pixel has it's own material id and temperature. Pixel materials are defined by `FallingSandsDataObj` (scriptable objects). You can find a bunch of them under `Demo/Data`.



Material Properties

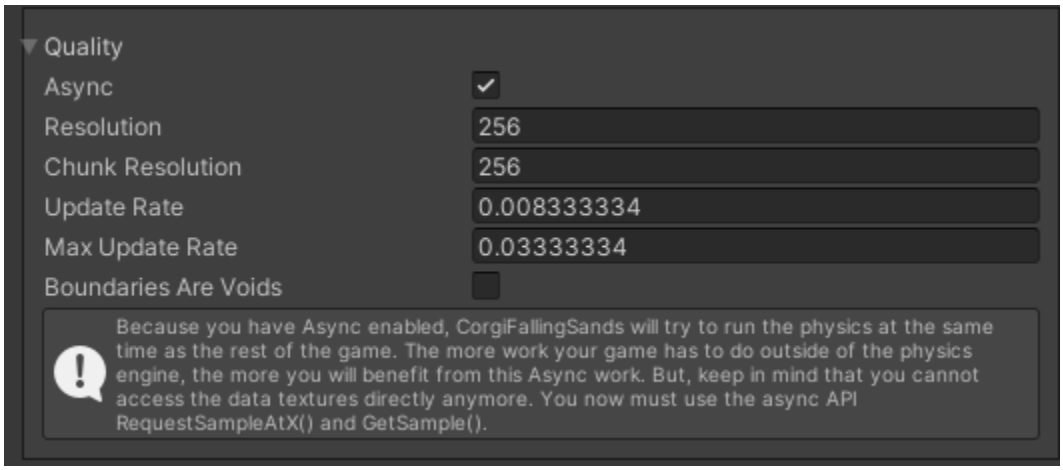
- **NameKey** and **DescKey** are just the names and descriptions. You can use these however you feel like.
- **ShowInPicker** is just for the demo, having it enabled will allow the material to be selected in the demo.
- **ID** is automatically set by the plugin. It's a helpful enum that is generated for use in scripting. Do not use this enum in the inspector.
- **Metadata** is used by the physics simulation. You can set physics properties for the material here.
    - **FluidType** determines how the material behaves.
        - Air does nothing.
        - Solid does nothing, but blocks others from moving through it.
        - Sand is like a solid, but falls.
        - Fluid tries to go down in any direction.
        - Gas tries to go up in any direction.
    - **Color** is used by the demo for drawing the material.
    - **Temperature** will be written to the temperature data in the physics simulation every physics step.
    - **HasSolidCollision** is for the demo, it's used for the physics mesh generation so the character controller has something to walk on.
    - **SpreadsInWater** is used by grass, which replaces water with itself.
    - **CreatesFromThinAir** is used for spouts, such as a water creating material.
    - Min/**MaxTemperature** and Min/**MaxTemperatureBecomes** are used for transforming the material when it's local temperature reaches theses values. For example, water becomes Ice when cooled, but it can also become Steam when heated.
- **OnStamp**, **OnTempHigh**, **OnTempLow** can be used for playing audio if GenerateEvents is enabled on the physics system.
- **ListBias** is used by the demo's UI.

FallingSandsSystem



The prefab PfbFallingSandsSystem is an example on how to use the core `FallingSandsWorld`. The script FallingSandsSystem handles everything from creating, simulating, saving, loading, generating chunks, and creating physics meshes.

Quality

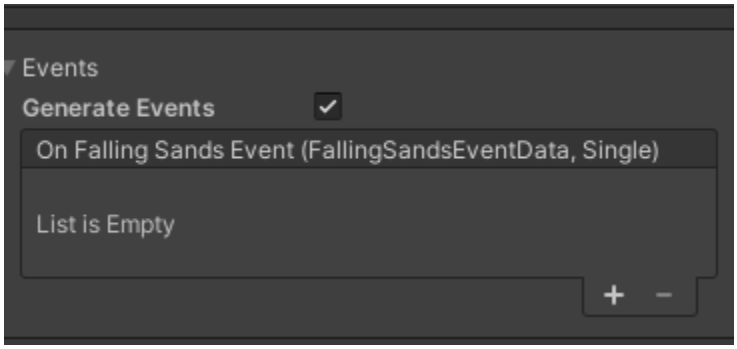


- **Async** will tell the system to run the rest of the game alongside the physics system. This is the recommended and optimal way to use this plugin.
    - Because of it's async nature, you will be unable to access data directly on the main thread, usually. Unity will prevent you from doing this in the Editor. Do not try to work around this. If you want main thread access at all times, disable Async! To access and edit from the main thread, please use the Bridge components `FallingSandsSampleX` and `FallingSandsMouseWriter` or the scripting API .
- **resolution** is for the pixel physics simulation resolution.
- **chunkResolution** is for the infinite world's chunk sizes.
- updateRate is how frequently the physics system will update. Think of these in terms of "ticks per second" or tps. Use values such as "1/60", "1/120" etc.
- **maxUpdateRate** is the maximum time that will be considered as passed since the previous frame. Use multiples of **updateRate** for the best results.
- **BoundariesAreVoids** will change boundaries of the physics world from solids to airs. This means fluids will travel off-screen and be lost forever.

Save Data

- **StorageManager** is a reference to your storage manager. Corgi Pixel Physics comes with one for Windows. If you want to support other platforms, you'll need to override FallingSandsStorage.
- **LoadOnStart** and **SaveOnDestroy** are for saving and loading the whole world when the game starts or closes.
- **SaveAndUnloadFarAwayChunks** will keep memory usage low by unloading chunks and storing them to disk if they get too far away from the player.
- **LoadChunksAfterGenerate** will load those unloaded chunks back into memory when the player gets close again.

Events



- **GenerateEvents** being enabled will tell the physics system to output a list of events.

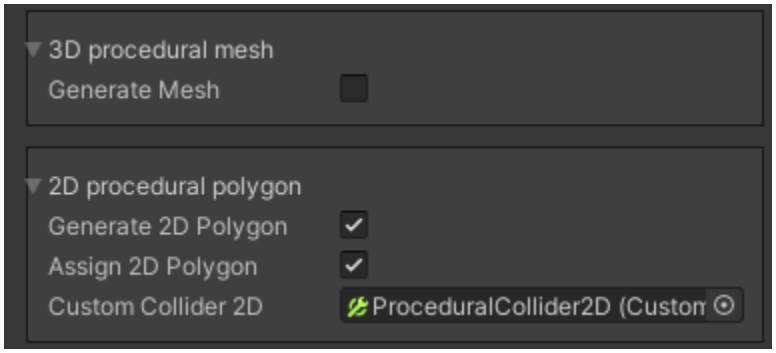
```
[System.Serializable]
public struct FallingSandsEventData
{
    public int index;
    public FallingDataType id_a;
    public FallingDataType id_b;
    public float temperature;
}
```

  These events are reported as the FallingSandsEventData struct.
  - **index** is the data index where the event happened.
  - **id_a** and **id_b** are the material ids of the affected pixels.
  - **temperature** was the temperature at the location of the event.

- **OnFallingSandsEvent(FallingSandsEventData data)** is the callback you can hook into for reacting to these events.

Note: Event generation can be variably slow. Don’t use them if you do not really need them.

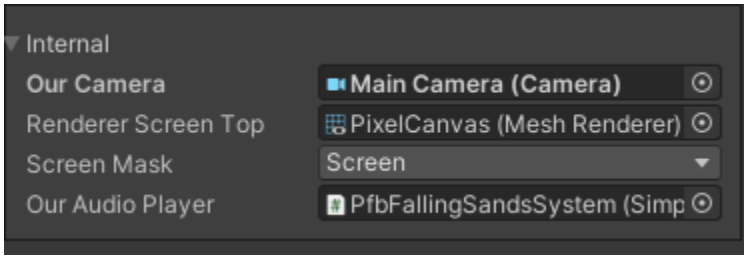
Physics Mesh Generation



- **GenerateMesh** and **Generate2DPolygon** are used for generating physics meshes for both the 3D and 2D physics systems within Unity. Use only the one you need.
- **MeshCollider** and **CustomCollider2D** will be used for auto assignment when its ready.

Note: Unity 2021.2 and onward have a much faster API for setting 2D mesh colliders. If you need 2D, it's recommended to use this version or later.

Internal Settings



These settings are just random references necessary for the demo to run.

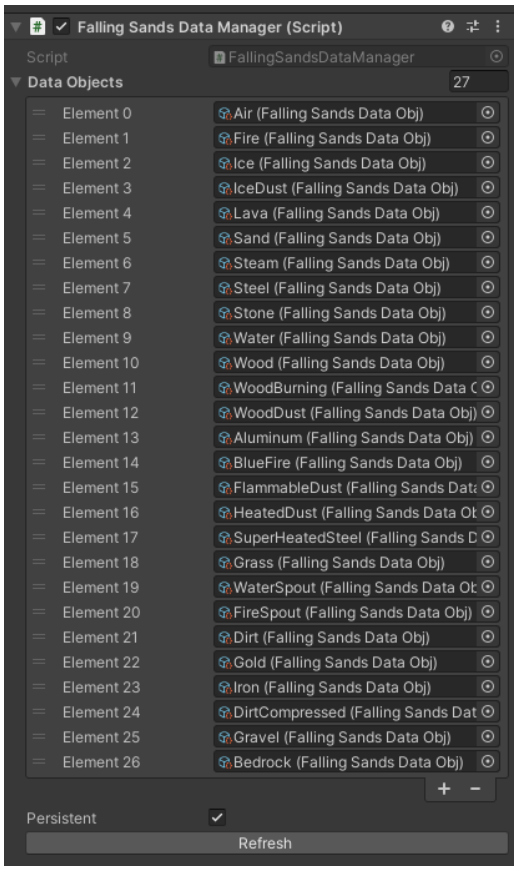
FallingSandsStorageManager

- **SaveDataSubFolder** is used for storing the chunks of the infinite world.

This script is meant to be an example on how to store world data. You should override this when building a game for anything other than the Standalone player. It uses **Application.persistentDataPath** as the root save location, and **System.IO** calls which may not be supported on non Standalone platforms.

The functions are simple virtual functions which can be overriden via a new class which inherits from `FallingSandsStorageManager`.

FallingSandsDataManager



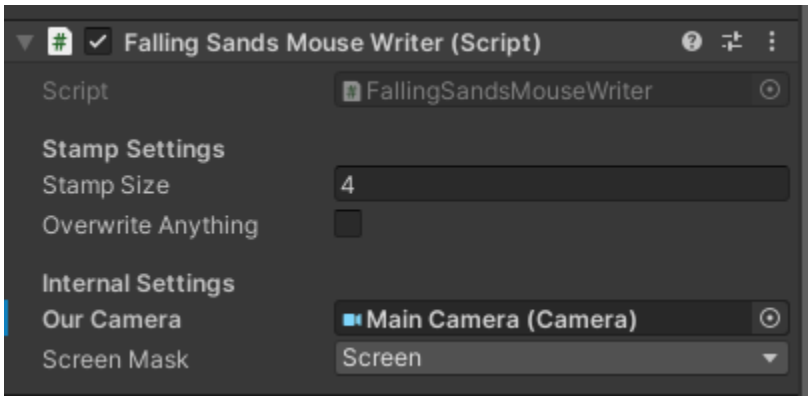
This script is placed on the very important prefab **PfbFallingSandsDataManager**. It keeps a list of all **FallingSandsDataObjs** in the project. You can hit Refresh to update this list at any time. It will automatically create an enum of the IDs and automatically assign them for you.

This prefab should only be used in the prefab view.

Component Bridges

Because of the async nature of this plugin, you normally can not access any of the data on the main thread. To work around this, I have provided a set of “bridge” components to bridge the gap between the monobehavior world and the physics world. There are also some scripting APIs you can access to do the same thing and more.

FallingSandsMouseWriter



This script uses the internal API:

```
 FallingSandsSystem.RequestStampAtScreenPosition(FallingData data, float temperature, Vector3 screenPosition, int radius);
```

This inserts an async stamp request. Will place pixels of type id of size radius into the requested position during the next physics pass.

FallingSandsSampleAt[Mouse|Point|Transform]

These scripts use the internal API:

```
public int RequestSampleAtScreenPosition(Vector3 screenPosition)
public int RequestSampleAtWorldPosition(Vector3 worldPosition)
```

Use this to sample the pixel physics world. This returns a ticket. Use it one frame later to get the result via GetSample(ticket);

```
public SampleData GetSample(int ticket)
```

One frame after requesting a ticket via RequestSampleAt(), you can use GetSample(ticket) to view the sample.

Example usage:

```
private void Update()
{
      if(_sampleTicket >= 0)
      {
          Sample = FallingSandsSystem.Instance.GetSample(_sampleTicket);
      }

      _sampleTicket =
FallingSandsSystem.Instance.RequestSampleAtWorldPosition(transform.position);
}
```

SampleData is a struct which contains the **id** and **temperature** of the pixel at the requested location.

FallingSandsCollisionRigidbody[2D]

FallingSandsCollisionRigidbody and **FallingSandsCollisionRigidbody2D** can be placed on **Rigidbody** or **Rigidbody2D**s so that the physics system can keep track of them when moving around the world. When the demo's character controller moves off-screen, these tracked rigidbodies will be shifted along with the world data to keep things on-screen.

FallingSandsCharacterController[2D]

FallingSandsCharacterController and **FallingSandsCharacterController2D** are included with the demo as simple examples of moving around a player in the pixel physics world. I recommend sticking to 2D, since the physics world is 2D. You can use 3D if you choose to, it's simply a bit wasteful in the context of the demo.

Troubleshooting

- If performance is slow, try lowering the resolution of the physics system.
- This physics plugin requires the Job System, Mathematics, Burst Compiler, and Collections packages to be installed. If any are missing, you can add them via the Unity package manager (window -> package manager) by hitting the + on the top right and adding by the following names:
    - com.unity.collections
    - com.unity.burst
    - com.unity.jobs
    - com.unity.mathematics

Contact Me!

If anything is unclear or even if you just want to give feedback or suggestions: please contact me!

- You can email me at coty@wanderingcorgi.com
- Or you can hit me up on Discord (Coty#2845).
- There's also a support discord here: https://discord.gg/n23MtuE