Discussion of the game asset pipeline including file structure of all game data files.

Asset file formats

The game requires certain file formats for types of assets that are present in the game. The game uses three asset types to create gameplay and the game world. These assets are textures, audio and levels.

Textures are stored in the direct draw surface (DDS) format. This is an optimised format that compresses the texture data so that it has a smaller memory footprint and can store a pre-generated mipmap chain, removing the need to generate this at runtime. Using this file format will consume less memory per texture and will decrease bus times when uploading texture data to the GPU as the texture data is smaller, improving game performance when loading and using textures. However, the compression method used is lossy and this will reduce the image quality that is used in the game.

Audio is stored as a wav file. This file format stores uncompressed audio to be loaded and played by the game's audio system.

Levels are stored in a custom file format called LEVEL. A LEVEL file contains the data that make up a level in the game. This is the entities and the components that those entities contain along with the component properties and their values. The LEVEL file stores data in yaml. Yaml is a human readable format that stores scalar values such as strings and integers inside vectors such as maps and arrays. The data contained can be edited by hand which makes it easier to make small changes to a level without needing to load it into a tool.

The LEVEL file stores data in maps with key value pairs to label data. The begins with the level key, stored with its name value. This can be used to internally give each level a specific name.

```
Level: Untitled
```

The file then starts an array that contains all of the entities in the level. Each entity is stored alongside its id and an array containing all of the components that the entity has. Each component stores each of its properties as a key value pair, where the key is used to access the value of the property during deserialization.

```
Entities:
  - Entity: 51
    TransformComponent:
      Position: [488, -5061]
      Scale: [1, 1]
      RotationRadians: 0
    SpriteComponent:
      RelativePosition: [0, 0]
      RelativeRotationRadians: 0
      RelativeScale: [1, 1]
      Colour: [1, 1, 1, 1]
      ZOrder: 0
      Texture: Assets\scifi_tilesheet@2.dds
      SourceRectTop: 0
      SourceRectRight: 128
      SourceRectBottom: 128
      SourceRectLeft: 0
      Origin: [64, 64]
  - Entity: 189
    TransformComponent:
      Position: [1512, -69]
      Scale: [1, 1]
      RotationRadians: 0
    SpriteComponent:
      RelativePosition: [0, 0]
      RelativeRotationRadians: 0
      RelativeScale: [1, 1]
      Colour: [1, 1, 1, 1]
      ZOrder: 0
      Texture: Assets\scifi_tilesheet@2.dds
      SourceRectTop: 0
      SourceRectRight: 128
      SourceRectBottom: 128
      SourceRectLeft: 0
      Origin: [64, 64]
```

Description of pipeline

The DirectXToolkit library provides a command line tool to convert texture images from a format that would be created by an artist, such as a jpeg, png or targa, into a compressed dds file.

A new asset pipeline that would handle this process in a more visual way was created. The new pipeline is implemented through a level editor tool that was created to allow designers to build level content for the game and export optimised and game ready files to be used in the game.

The tool provides functionality for entities to be added to a game scene, displayed in the scene outliner panel. Entities can be selected from this panel to be worked on. These entities can have components added to them. Components are a collection of data properties that can be edited inside the details panel and can be used to customise entities. For example, all entities have a transform component which gives them a position, rotation and scale property. These properties can be edited to transform the entity in the game scene. Other components that can be added

to entities in the editor are sprite render to add sprite properties, sprite animation render to add animated sprite properties, enemy spawner to add enemy spawn properties and an environment tag to mark the entity as an entity that should have environment collision built around it.

The tool also provides functionality to create and edit tilemaps. Properties of the tilemap such as its width, height, size of an individual tile and the source sprite sheet texture that contains the tiles can be set up during creation of the tilemap. A grid of entities is then created where each entity represents a tile in the tile map by a sprite component. The tilemap can be positioned in the scene and have the index of each of its tiles in the source sprite sheet edited in the details panel.

Once the scene has been completed it can be saved. Saving the scene writes a .LEVEL file storing the information about the entities in the scene, what components they have and the properties of those components. Any textures that have been imported to create the scene are written into direct draw surface format and stored alongside the .LEVEL file. These files can then be included in the game assets folder and loaded by the game.

.LEVEL file specification
Read and written using the yaml-cpp library.

Documentation to write a file using the library can be read here (Bedeer.J, 2021, How to Emit YAML).

Documentation to parse a file using the library can be read here (Bedeer.J, 2021, How to Parse a Document (Old API)).

Level : Name
Entities:

- Entity: id
    - TransformComponent:
        - Position: [x, y]
        - Scale: [x, y]
        - RotationRadians: x
    - SpriteAnimComponent:
        - RelativePosition: [x, y]
        - RelativeRotationRadians: x
        - RelativeScale: [x, y]
        - Colour: [r, g, b, a]
        - ZOrder: x
        - Texture: filepath\filename.ext
        - Frames:
            - FramexSourceRectTop: x
            - FramexSourceRecRight: x
            - FramexSourceRectBottom: x
            - FramexSourceRectLeft: x
        - PlaybackRate: x
        - Looping: x
        - Origin: [x, y]
    - EnvironmentTag:
    - SpriteComponent:
        - RelativePosition: [x, y]
        - RelativeRotationRadians: x
        - RelativeScale: [x, y]
        - Colour: [r, g, b, a]
        - ZOrder: x
        - Texture: filepath\filename.ext
        - SourceRectTop: x
        - SourceRectRight: x
        - SourceRectBottom: x
        - SourceRectLeft: x
        - Origin: [x, y]
    - EnemySpawnerComponent:
    - RelativePosition: [x, y]
    - EnemyType: x
    - SpawnCount: x
    - SpawnInterval: x

.DDS file specification
The format of the direct draw surface can be read about here (Microsoft, 2021, Programming Guide for DDS).

.WAV file specification
The format of a wave audio file can be read about here (Fileformat, 2021, What is a WAV File?).

Effectiveness of the asset pipeline
The tools developed to facilitate the pipeline were successful at providing a visual way to build levels and handle data optimisations and file formats. Multiple levels were created for the game using the tools that were each unique and featured different patterns of gameplay. Each of these levels could be loaded by the game and played offering a variety of gameplay scenarios to the player. The pipeline developed enables non-technical developers to create this content for the game allowing for it to be expanded upon in the future without the need to change the underlying codebase.

Furthermore, technical details about required file formats for the game were abstracted away from the developer by the tool. A variety of popular texture file formats were accepted by the tool which were converted into the optimised file format, ready for the game. This removes the need to manually manage and produce data in the required file format and improves the useability of the tool as it can fit into and be used in a wide variety of scenarios and pipelines incorporating other third party tools.

The .LEVEL file outputted by the editor tool is stored in a human readable format. This allows for small edits to be made to the level by editing the .LEVEL file by hand in a text editor tool. This can make small fixes, tweaks or additions quick to implement as the level editing tool is not needed. This could allow for changes to be made on machines that may not have the level editing tool installed. A drawback to the file being in a human readable format is that the file is slower to load than if the level data was stored in a binary format. This could be improved by creating two versions of the level file where one is in binary format and the other is in the human readable format. This would be similar to the Maya binary and Maya ascii files used to save scenes by Autodesk Maya digital content creation tool. Having the two file formats would allow them to be used interchangeably when the scenario demands certain advantages.

A disadvantage of using an in-house developed pipeline tool is that there is not a community producing tutorials around how to use it. This resulted in time needing to be spent training members of the development team how to use the tool which took time away from development. Furthermore, user experience feedback was generated as different people used the tool. Different people found areas of the tool unintuitive or awkward to use. This feedback could be implemented in the tool and used to improve its user facing design and usage.

However, using an in-house developed tool also provided advantages. Bugs that were encountered were able to be identified and fixed rapidly. New features or user experience changes were also able to be implemented to make the tool more usable by the development

team. This would allow the tool to be worked with easier by more of the team, improving development time and the quality of the level built.

References

GitHub (2021) *yaml-cpp*. Available from: https://github.com/jbeder/yaml-cpp [Accessed 17 April 2021].

Bedeer, J (2021) *How to Emit YAML*. Available from: https://github.com/jbeder/yaml-cpp/wiki/How-To-Emit-YAML [Accessed 26 April 2021].

Bedeer, J (2021) *How to Parse a Document (Old API)*. Available from: https://github.com/jbeder/yaml-cpp/wiki/How-To-Parse-A-Document-(Old-API)L [Accessed 26 April 2021].

Microsoft (2018) *Programming Guide for DDS*. Available from: https://docs.microsoft.com/en-us/windows/win32/direct3ddds/dx-graphics-dds-pguide [Accessed 26 April 2021].

Fileformat (2021) *What is a WAV file?*. Available from: https://docs.fileformat.com/audio/wav/ [Accessed 26 April 2021].