# Entity-component system

Each entity is composed of one or more components, usually assigned at entity's construction. Each component adds data neccessary for one system to use the entity.

Implemented components:

* **ColliderRectComponent** - adds a rectangular collider shape
* **Node2DComponent** - adds scene node data. Dependency for all components that make use of 2D transformations (position, rotation, scaling)
* **RigidBody2DComponent** - adds physics properties to the entity. Unfinished and unused since we don't have a physics engine yet.
* **Visible2DComponent** - adds a visible sprite-based animation
* **AbstractBehaviorComponent** - a base for adding event-based behavior
* **EventQueueComponent** - adds the ability to receive event notifications and react to them

# Scene graph (2D)

The scene graph is made of entities with Node2DComponents. Each node can have child nodes and each child's transformation is relative to their parent's absolute transformation. Each scene has a root node entity (named "RootNode") with no parent, whose transformation is absolute.

# Asset management

The asset manager creates assets on-demand. When an asset based on a file is requested, the manager creates it if it doesn't exist yet, but doesn't load it. The assets can be manually loaded or unloaded at any time, but they exist as long as at least one AssetPtr reference to them exists.

Currently supported assets are: \* Textures \* Animations \* Scenes

## TextureAsset

An image based asset. Can be loaded from many image file types.

## Animation2DAsset

A sprite-based animation. Can be loaded from a JSON file with the following structure:

{  
 // Texture asset filename  
 "texture": "filename",  
 // Relative area of the texture used as the sprite sheet. Values 0.0-1.0  
 "texture": [<Left>, <Top>, <Right>, <Bottom>],  
 // number of rows in the sprite sheet  
 "rows": <number>,  
 // number of columns in the sprite sheet  
 "columns": <number>,  
 // The 0-based index of the sheet frame used as the first frame of the anim  
 "frame-first": <index>,  
 // The index of the sheet frame used as the last frame of the anim  
 "frame-last": <index>  
}

## Scene2D

A 2d node-based scene. Can be loaded from a JSON file with the following structure:

[// List of entities  
 {// entity  
 "name": "Name of the entity",  
 "components": [// list of components  
 {// component  
 "type": "Type of the component",  
 "property1": "value1",  
 "property2": "value2"  
 // properties depend on the component type  
 }  
 ]  
 }  
  
]

# Examples

## Loading a scene and adding a node entity

AssetPtr<Scene2D> myScene = AssetManager::instance().GetScene2D("scenes/MyScene.scene");  
AssetManager::instance().Load("scenes/MyScene.scene");  
myScene->createEntity(  
 { new Node2DComponent(myScene->GetRootNode()) },  
 "RootChildNode"  
);

## Adding an entity with behavior to the scene

Entity \*playerEntity = myScene->createEntity(  
 {   
 new Node2DComponent(  
 myScene->GetRootNode(),  
 glm::vec2(5.0, 6.0)// position at (x=5, y=6)  
 ),  
 new MyPlayerBehaviorComponent()  
 },  
 "Player"  
);

## Defining a behavior component

class MyPlayerBehaviorComponent : public AbstractBehaviorComponent  
{  
public:  
 void onCreate()  
 {  
 speed = 5.0;  
 }  
  
 void onUpdate(const ComponentUpdateEvent& ev)  
 {  
 // requires a node component  
 Node2DComponent \*transform = &CompReq<Node2DComponent>();  
  
 transform->Move(glm::vec2(  
 speed \* ev.sElapsed,  
 0.0  
 ));  
 }  
   
 void onCollision(const ComponentCollidedEvent& ev)  
 {  
 // maybe explode or something  
 }  
  
protected:  
 float speed;  
}

# References

The scene graph system is inspired by my experience with Ogre3D graphics engine. The ECS is loosely based on my experience with Unity3D engine and various internet sources.

https://www.ogre3d.org/

https://unity.com/

http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/

https://en.wikipedia.org/wiki/Entity\_component\_system