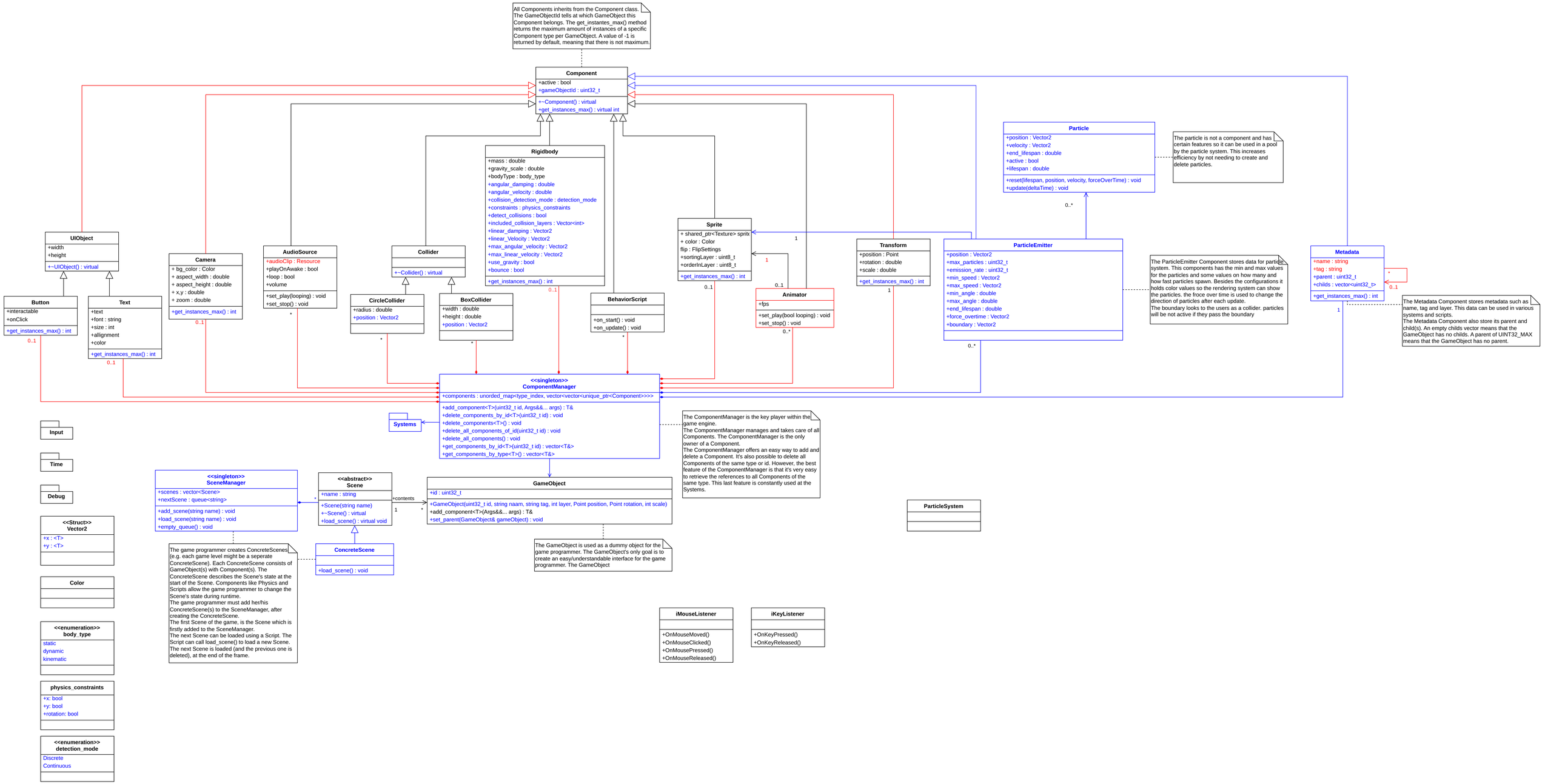
All Components inherits from the Component class. The GameObjectId tells at which GameObject this Component belongs. The get_instantes_max() method returns the maximum amount of instances of a specific Component type per GameObject. A value of -1 is returned by default, meaning that there is not maximum.
Component
+active : bool
+gameObjectId : uint32_t
+~Component() : virtual
+get_instances_max() : virtual int
Rigidbody
+mass : double
+gravity_scale : double
+bodyType : body_type
+angular_damping : double
+angular_velocity : double
+collision_detection_mode : detection_mode
+constraints : physics_constraints
+detect_collisions : bool
+included_collision_layers : Vector<int>
+linear_damping : Vector2
+linear_Velocity : Vector2
+max_angular_velocity : Vector2
+max_linear_velocity : Vector2
+use_gravity : bool
+bounce : bool
+get_instances_max() : int
Particle
+position : Vector2
+velocity : Vector2
+end_lifespan : double
+active : bool
+lifespan : double
+reset(lifespan, position, velocity, forceOverTime) : void
+update(deltaTime) : void
The particle is not a component and has certain features so it can be used in a pool by the particle system. This increases efficiency by not needing to create and delete particles.
UIObject
+width
+height
+~UIObject() : virtual
Camera
+ bg_color : Color
+ aspect_width : double
+ aspect_height : double
+ x,y : double
+ zoom : double
+get_instances_max() : int
AudioSource
+audioClip : Resource
+playOnAwake : bool
+loop : bool
+volume
+set_play(looping) : void
+set_stop() : void
Collider
+~Collider() : virtual
Sprite
+ shared_ptr<Texture> sprite
+ color : Color
flip : FlipSettings
+sortingLayer : uint8_t
+orderInLayer : uint8_t
+get_instances_max() : int
Transform
+position : Point
+rotation : double
+scale : double
+get_instances_max() : int
ParticleEmitter
+position : Vector2
+max_particles : uint32_t
+emission_rate : uint32_t
+min_speed : Vector2
+max_speed : Vector2
+min_angle : double
+max_angle : double
+end_lifespan : double
+force_overtime : Vector2
+boundary : Vector2
The ParticleEmitter Component stores data for particle system. This components has the min and max values for the particles and some values on how many and how fast particles spawn. Besides the configurations it holds color values so the rendering system can show the particles. the froce over time is used to change the direction of particles after each update.
The boundary looks to the users as a collider. particles will be not active if they pass the boundary
Metadata
+name : string
+tag : string
+parent : uint32_t
+childs : vector<uint32_t>
+get_instances_max() : int
The Metadata Component stores metadata such as name, tag and layer. This data can be used in various systems and scripts.
The Metadata Component also store its parent and child(s). An empty childs vector means that the GameObject has no childs. A parent of UINT32_MAX means that the GameObject has no parent.
Button
+interactable
+onClick
+get_instances_max() : int
Text
+text
+font : string
+size : int
+allignment
+color
+get_instances_max() : int
CircleCollider
+radius : double
+position : Vector2
BoxCollider
+width : double
+height : double
+position : Vector2
BehaviorScript
+on_start() : void
+on_update() : void
Animator
+fps
+set_play(bool looping) : void
+set_stop() : void
<<singleton>>
ComponentManager
+components : unorded_map<type_index, vector<vector<unique_ptr<Component>>>
+add_component<T>(uint32_t id, Args&&... args) : T&
+delete_components_by_id<T>(uint32_t id) : void
+delete_components<T>() : void
+delete_all_components_of_id(uint32_t id) : void
+delete_all_components() : void
+get_components_by_id<T>(uint32_t id) : vector<T&>
+get_components_by_type<T>() : vector<T&>
The ComponentManager is the key player within the game engine.
The ComponentManager manages and takes care of all Components. The ComponentManager is the only owner of a Component.
The ComponentManager offers an easy way to add and delete a Component. It's also possible to delete all Components of the same type or id. However, the best feature of the ComponentManager is that it's very easy to retrieve the references to all Components of the same type. This last feature is constantly used at the Systems.
Systems
Input
Time
Debug
<<Struct>>
Vector2
+x : <T>
+y : <T>
Color
<<enumeration>>
body_type
static
dynamic
kinematic
physics_constraints
+x: bool
+y: bool
+rotation: bool
<<enumeration>>
detection_mode
Discrete
Continuous
<<singleton>>
SceneManager
+scenes : vector<Scene>
+nextScene : queue<string>
+add_scene(string name) : void
+load_scene(string name) : void
+empty_queue() : void
<<abstract>>
Scene
+name : string
+Scene(string name)
+~Scene() : virtual
+load_scene() : virtual void
+contents
GameObject
+id : uint32_t
+GameObject(uint32_t id, string naam, string tag, int layer, Point position, Point rotation, int scale)
+add_component<T>(Args&&... args) : T&
+set_parent(GameObject& gameObject) : void
The GameObject is used as a dummy object for the game programmer. The GameObject's only goal is to create an easy/understandable interface for the game programmer. The GameObject
ConcreteScene
+load_scene() : void
The game programmer creates ConcreteScenes (e.g. each game level might be a seperate ConcreteScene). Each ConcreteScene consists of GameObject(s) with Component(s). The ConcreteScene describes the Scene's state at the start of the Scene. Components like Physics and Scripts allow the game programmer to change the Scene's state during runtime.
The game programmer must add her/his ConcreteScene(s) to the SceneManager, after creating the ConcreteScene.
The first Scene of the game, is the Scene which is firstly added to the SceneManager.
The next Scene can be loaded using a Script. The Script can call load_scene() to load a new Scene.
The next Scene is loaded (and the previous one is deleted), at the end of the frame.
ParticleSystem
iMouseListener
+OnMouseMoved()
+OnMouseClicked()
+OnMousePressed()
+OnMouseReleased()
iKeyListener
+OnKeyPressed()
+OnKeyReleased()