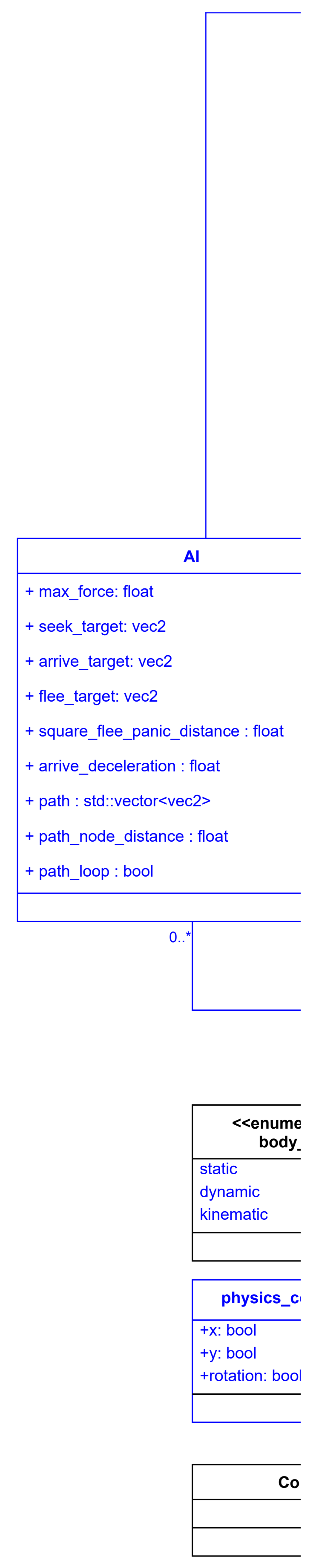

AI
+ max_force: float
+ seek_target: vec2
+ arrive_target: vec2
+ flee_target: vec2
+ square_flee_panic_distance : float
+ arrive_deceleration : float
+ path : std::vector<vec2>
+ path_node_distance : float
+ path_loop : bool
0..*
<<enume
body_
static
dynamic
kinematic
physics_c
+x: bool
+y: bool
+rotation: bool
Co

Component
+active : bool
+gameObjectId : uint32_t
+~Component() : virtual
+get_instances_max() : virtual int
UIObject
+ dimensions : vec2
+ offset : vec2
+~UIObject() : virtual
Button
+ world_space : bool
+get_instances_max() : int
Text
+text : string
+font_family : string
+font: std::optional<Asset>
+world_space : bool
+text_color : Color
+get_instances_max() : int
Camera
+ bg_color : Color
+ zoom : double
+ postion_offset : vec2
+ screen : ivec2
+ viewport_size : vec2
+get_instances_max() : int
AudioSource
- audioClip : Asset
+playOnAwake : bool
+loop : bool
+volume : float
+play(looping) : void
+stop() : void
Collider
+ offset : vec2
CircleCollider
+radius : float
BoxCollider
+ dimensions : vec2
Rigidbody
+mass : float
+gravity_scale : float
+bodyType : body_type
+ linear_Velocity : vec2
+ max_linear_velocity : float
+ linear_velocity_coefficient : vec2
+angular_velocity : float
+angular_velocity_coefficient : float
+max_angular_velocity : float
+constraints : physics_constraints
+elasticity_coefficient : float
+kinematic_collision : bool
+collision_layers : set<int>
+collision_layer : int
+collision_names: set<string>
+collision_tags: set<string>
+get_instances_max() : int
+add_force_linear(const vec2 & force) : void
+add_force_angular(float force) : void
BehaviorScript
+set_script() : BehaviorScript
Sprite
+ source : Asset
+ color : Color
+ flip : FlipSettings
+ sortingLayer : int
+ orderInLayer : int
+ size: vec2
+ angle_offset : float
+ scale_offset : float
+ position_offset: vec2
+ world_space: bool
+ get_instances_max() : int
Animator
+ fps : int
+ col : int
+ row : int
+ looping : bool
+ cycle_start : int
+ cycle_end : int
+ loop() : void
+ play() : void
+ pause() : void
+ stop() : void
+ set_fps(int fps) : void
+ set_cycle_range(int start, int end): void
+ set_anim(int col) : void
+ next_anim() : void
Transform
+position : Vec2
+rotation : float
+scale : float
+get_instances_max() : int
Particle
+position : vec2
+velocity : vec2
+force_over_time: vec2
+lifespan: float
+active : bool
+time_in_life : float
+angle : float
+reset(lifespan, position, velocity, angle) : void
+update(deltaTime) : void
+angle() : void
ParticleEmitter
+offset: Vector2
+max_particles : unsigned int
+emission_rate : float
+min_speed : float
+max_speed : float
+min_angle : float
+max_angle : float
+begin_lifespan : float
+end_lifespan : float
+force_overtime : vec2
+Boundary : boundary
- particles : std::vector<Particle>
Metadata
+name : string
+tag : string
+parent : uint32_t
+childs : vector<uint32_t>
+get_instances_max() : int
<<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&>
Systems
boundary
+width : float
+height : float
+offset : vec2
+reset_on_exit : bool
eration>>
_type
iMouseListener
+OnMouseMoved()
+OnMouseClicked()
+OnMousePressed()
+OnMouseReleased()
iKeyListener
+OnKeyPressed()
+OnKeyReleased()
onstraints
Asset
- src : std::string
Asset(const std::string & src)
<<Struct>>
Vector2
+x : <T>
+y : <T>
lor
Input
Time
Debug
<<singleton>>
SceneManager
+scenes : vector<Scene>
+nextScene : queue<string>
+add_scene(string name) : void
+load_scene(string name) : void
+empty_queue() : void
<<abstract>>
Scene
+ load_scene() : virtual void
+ get_name() : virtual void
+ get_save_manager() : Sav
+ new_object(See gameobje
+ set_persistent(const Asset
ConcreteScene
+load_scene() : void
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
+ set_persistent(bool persistent) : void
+contents
1
*
0..1
0..*

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.

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.

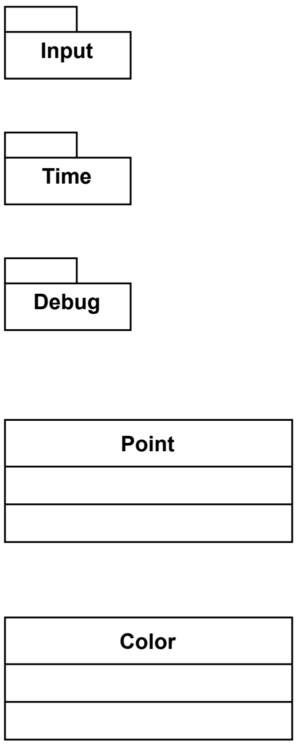
The ParticleEmitter Component stores data for particle system. This components has the min and max values for the particles and some values on how many and how fast particles spawn. Besides the configurations it holds color values so the rendering system can show the particles. the froce over time is 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

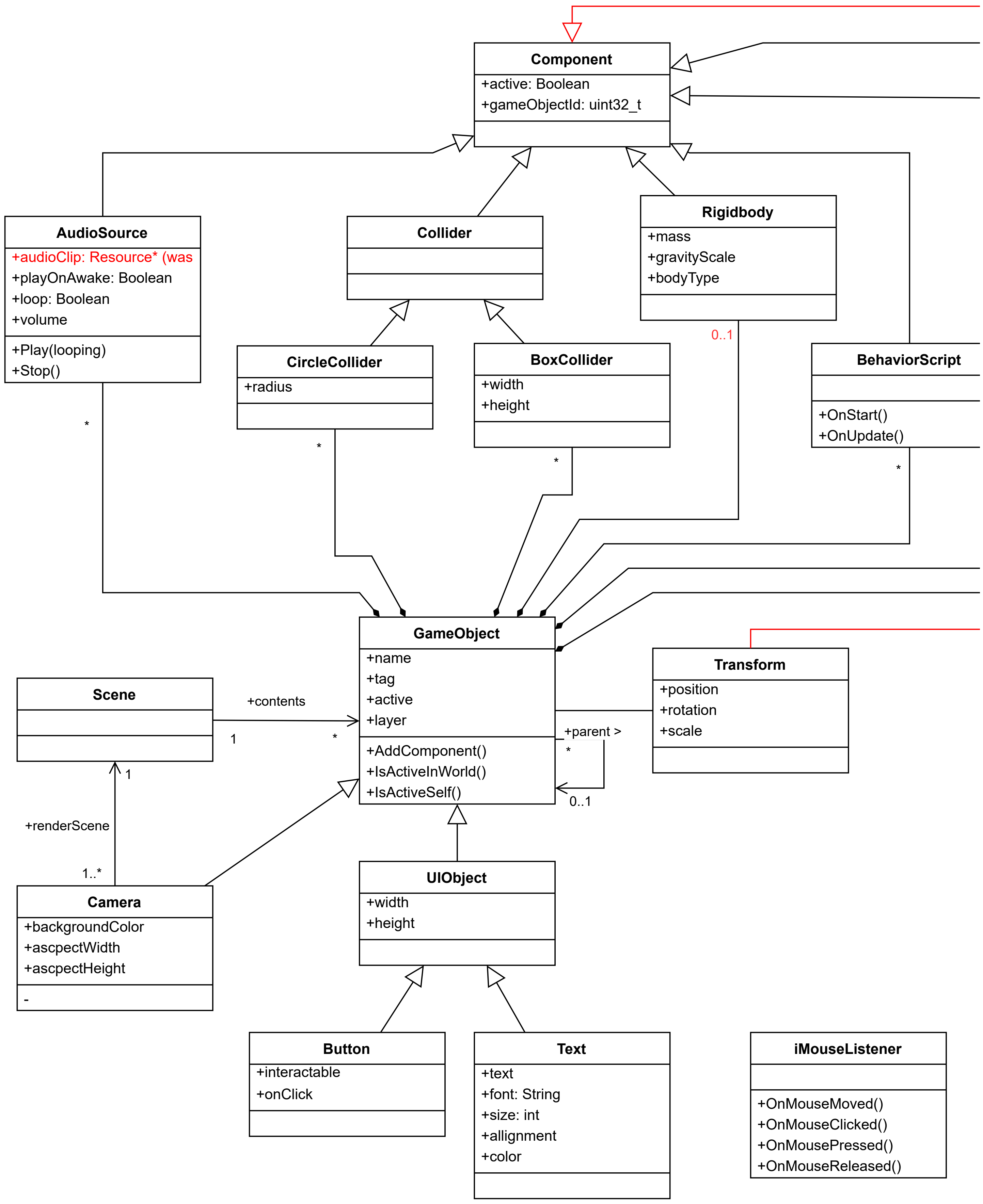
The Metadata Component stores metadata such as name, tag and layer. This data can be used in various systems and scripts.
The Metadata Component also store its parent and child(s). An empty childs vector means that the GameObject has no childs. A parent of UINT32_MAX means that the GameObject has no parent.

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.

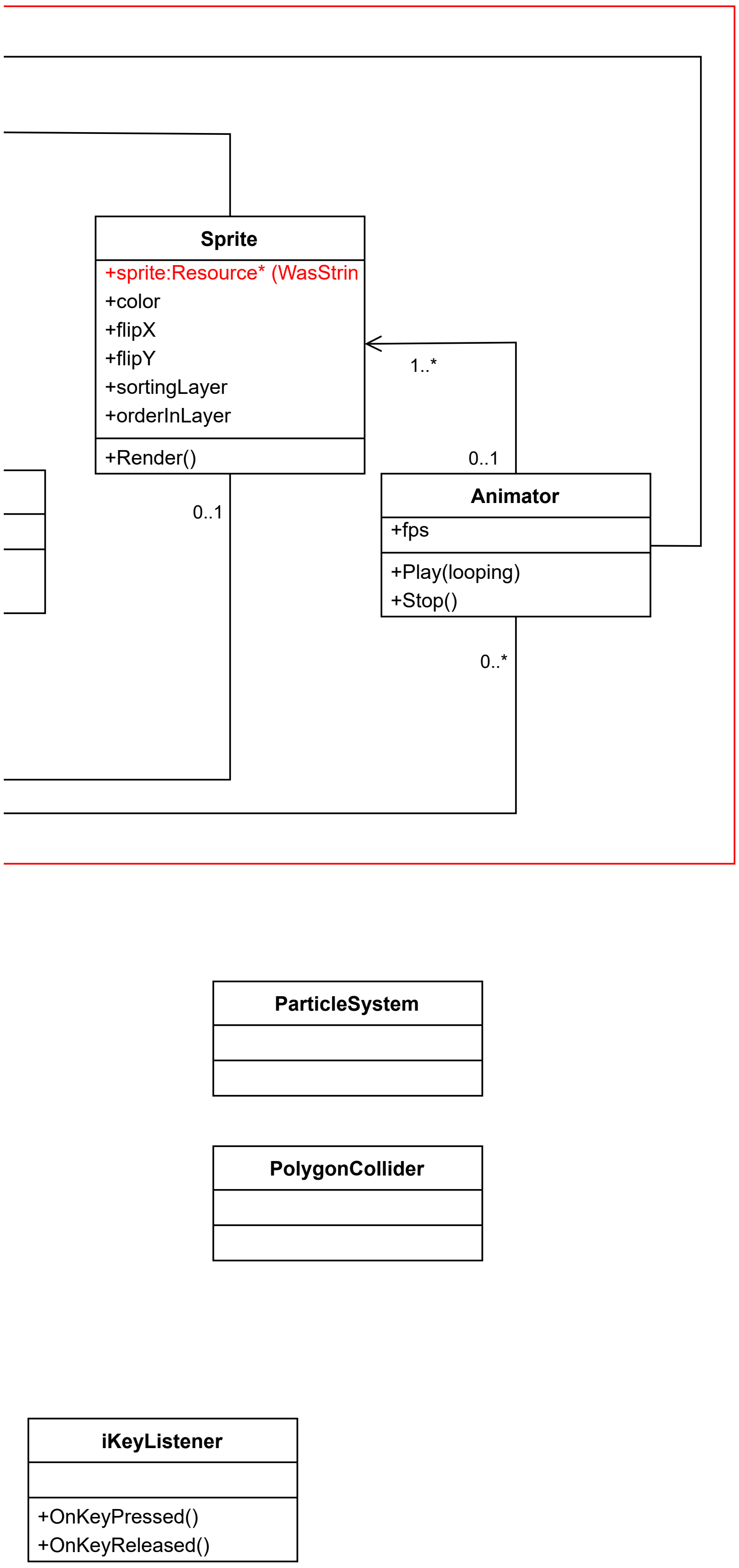
The GameObject is used as a dummy object for the game programmer. The GameObject's only goal is to create an easy/understandable interface for the game programmer. The GameObject

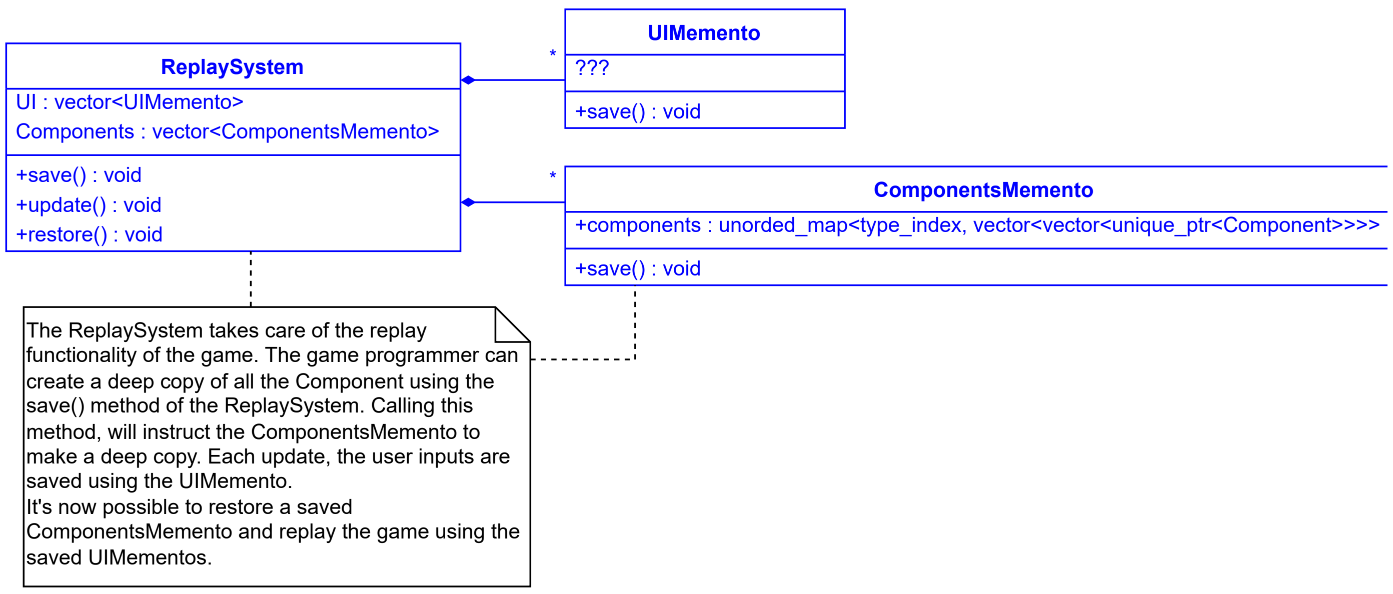
The game programmer creates ConcreteScenes (e.g. each game level might be a seperate ConcreteScene). Each ConcreteScene consists of GameObject(s) with Component(s). The ConcreteScene describes the Scene's state at the start of the Scene. Components like Physics and Scripts allow the game programmer to 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.


Input
Time
Debug
Point
Color

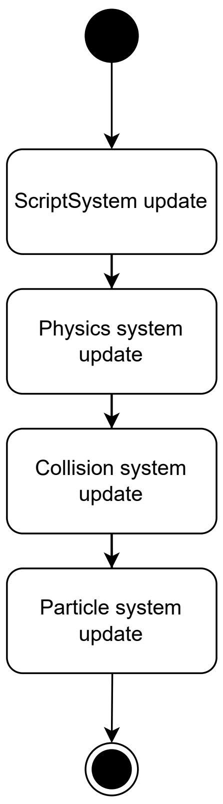


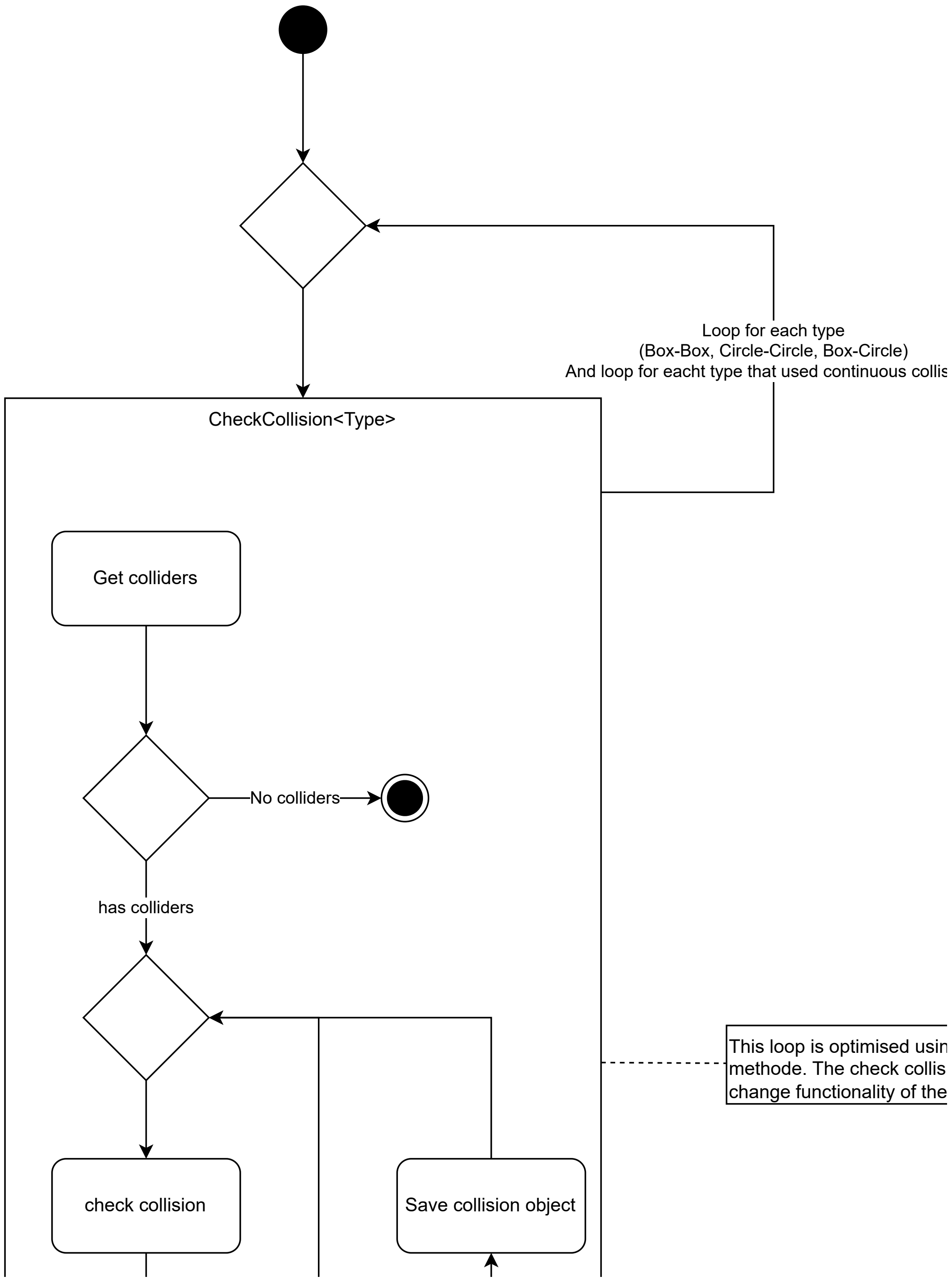
Component
+active: Boolean
+gameObjectId: uint32_t
AudioSource
+audioClip: Resource* (was
+playOnAwake: Boolean
+loop: Boolean
+volume
+Play(looping)
+Stop()
Collider
Rigidbody
+mass
+gravityScale
+bodyType
0..1
CircleCollider
+radius
BoxCollider
+width
+height
BehaviorScript
+OnStart()
+OnUpdate()
*
*
*
*
GameObject
+name
+tag
+active
+layer
+AddComponent()
+IsActiveInWorld()
+IsActiveSelf()
Scene
+contents
1
*
+parent >
*
0..1
Transform
+position
+rotation
+scale
1
+renderScene
1..*
Camera
+backgroundColor
+ascpectWidth
+ascpectHeight
-
UIObject
+width
+height
Button
+interactable
+onClick
Text
+text
+font: String
+size: int
+allignment
+color
iMouseListener
+OnMouseMoved()
+OnMouseClicked()
+OnMousePressed()
+OnMouseReleased()

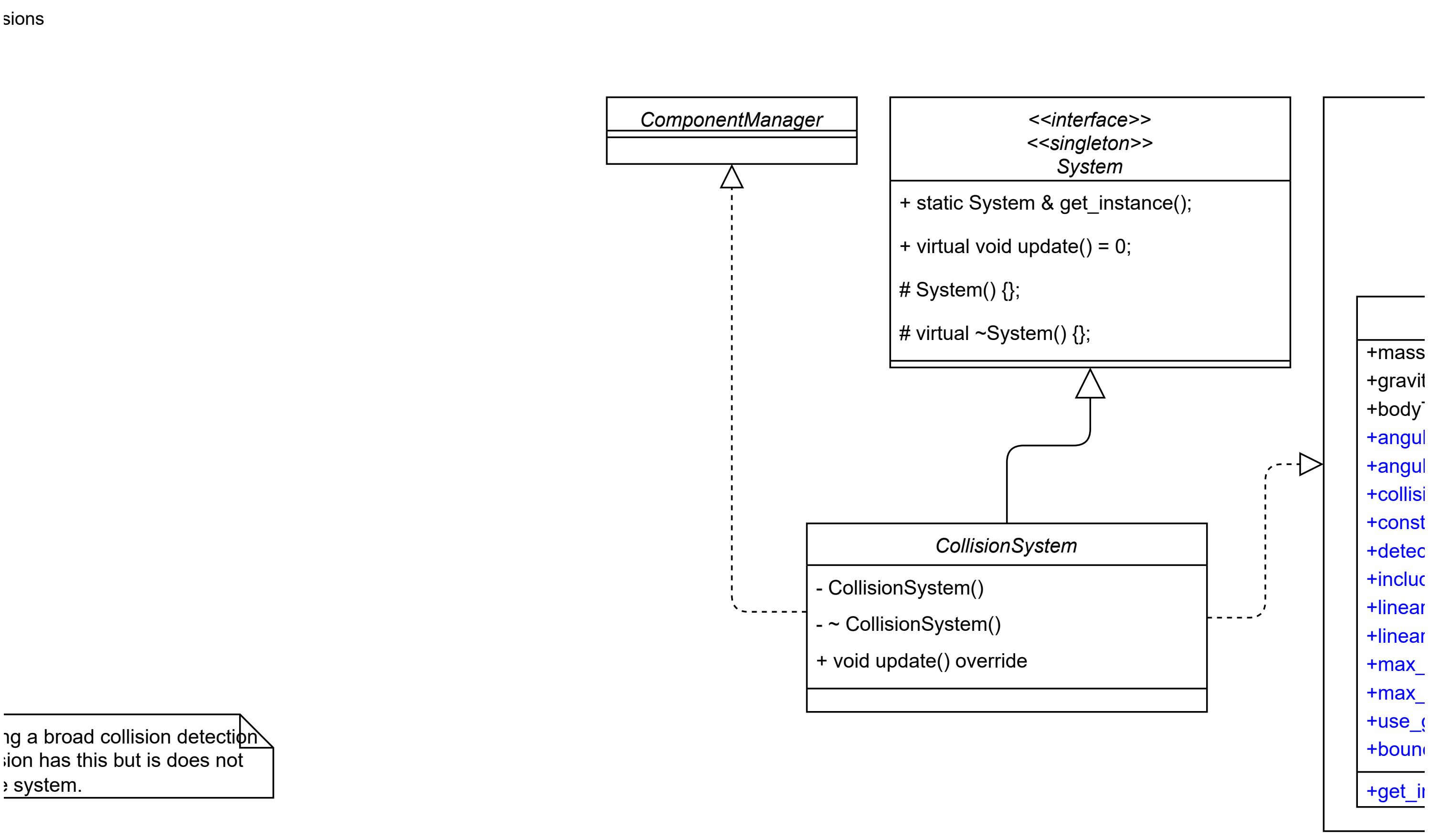

Sprite
+sprite:Resource* (WasStrin
+color
+flipX
+flipY
+sortingLayer
+orderInLayer
+Render()
1..*
0..1
0..1
Animator
+fps
+Play(looping)
+Stop()
0..*
ParticleSystem
PolygonCollider
iKeyListener
+OnKeyPressed()
+OnKeyReleased()

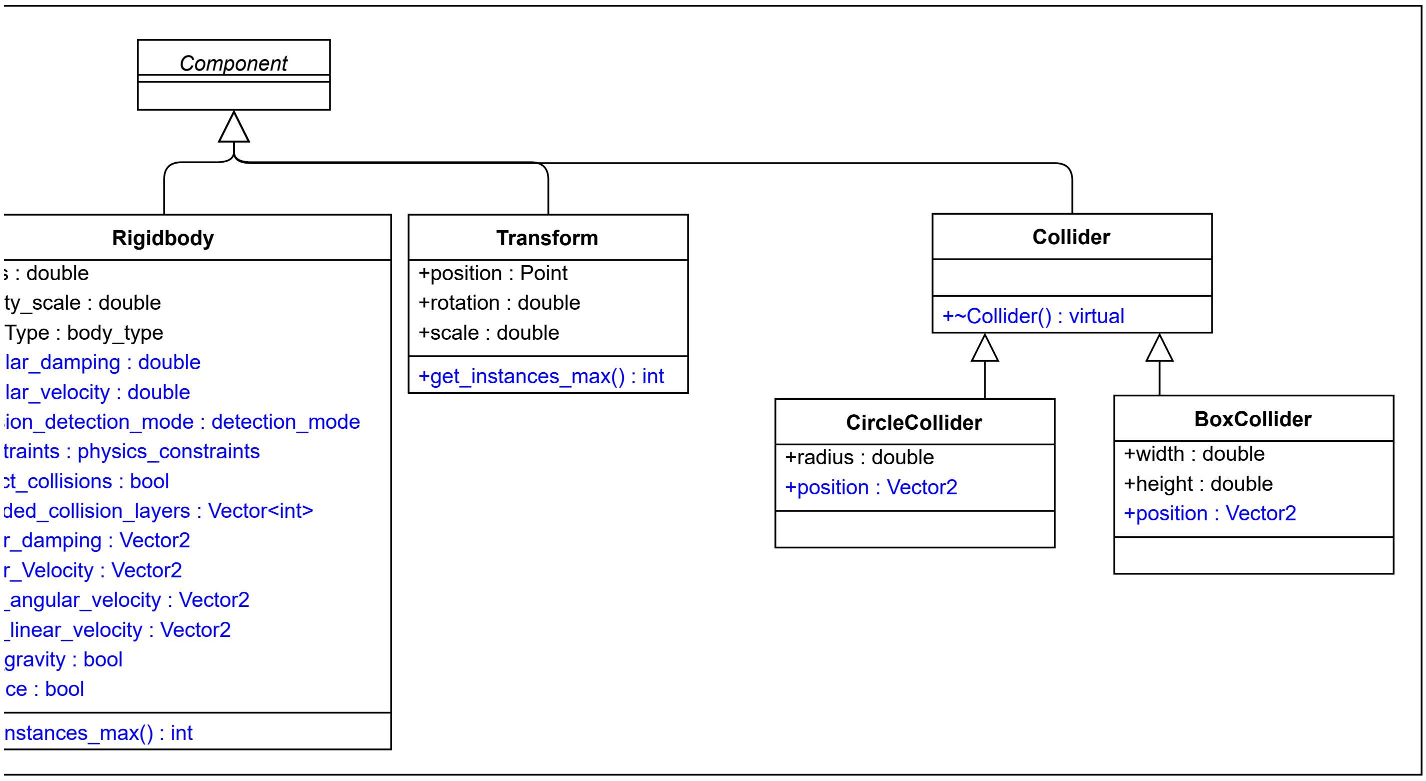


The ReplaySystem takes care of the replay functionality of the game. The game programmer can create a deep copy of all the Component using the save() method of the ReplaySystem. Calling this method, will instruct the ComponentsMemento to make a deep copy. Each update, the user inputs are saved using the UIMemento.
It's now possible to restore a saved ComponentsMemento and replay the game using the saved UIMementos.

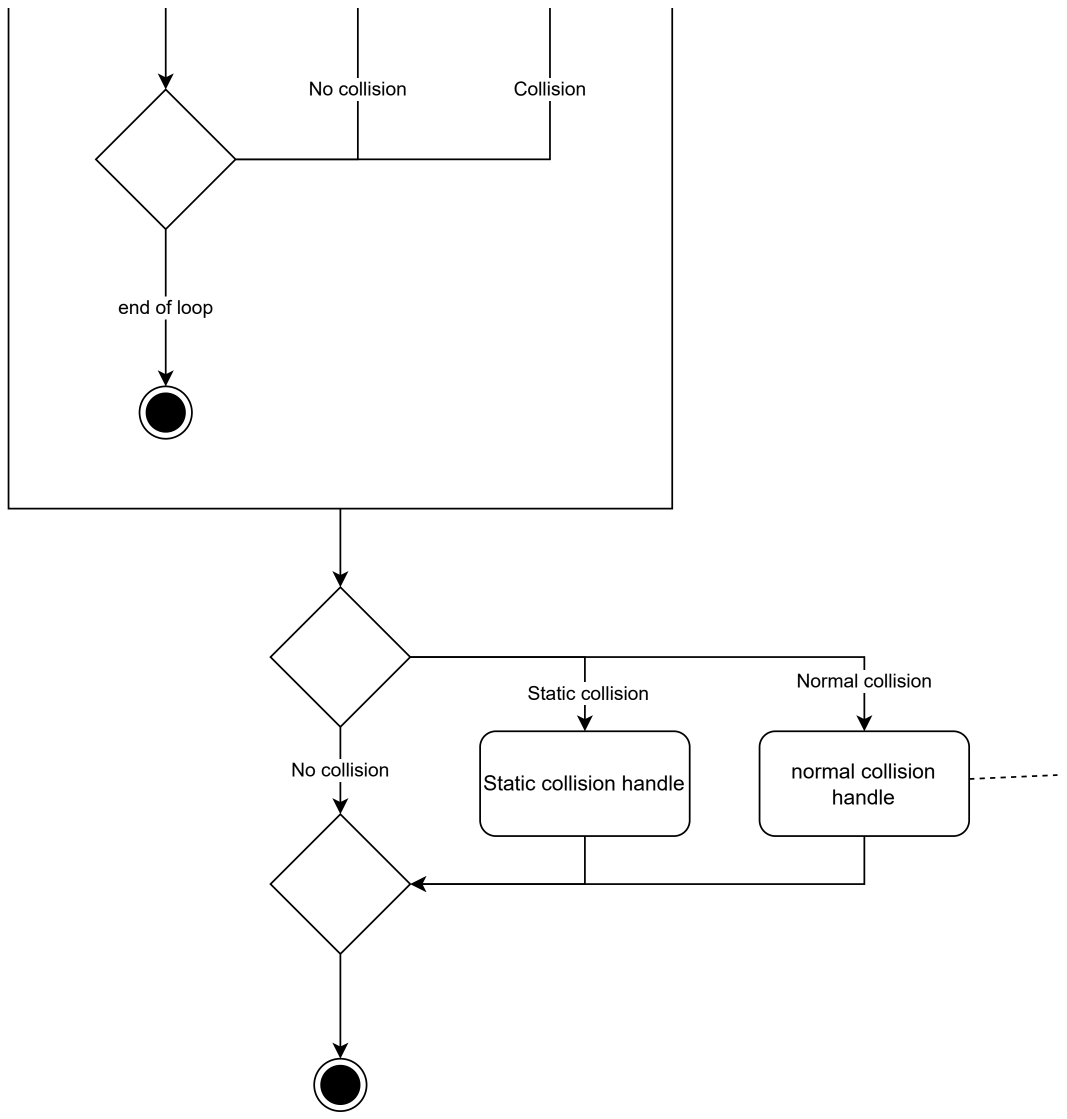

ScriptSystem update
Physics system update
Collision system update
Particle system update


Loop for each type
(Box-Box, Circle-Circle, Box-Circle)
And loop for eacht type that used continuous collis
CheckCollision<Type>
Get colliders
No colliders
has colliders
check collision
Save collision object
This loop is optimised usin
methode. The check collis
change functionality of the

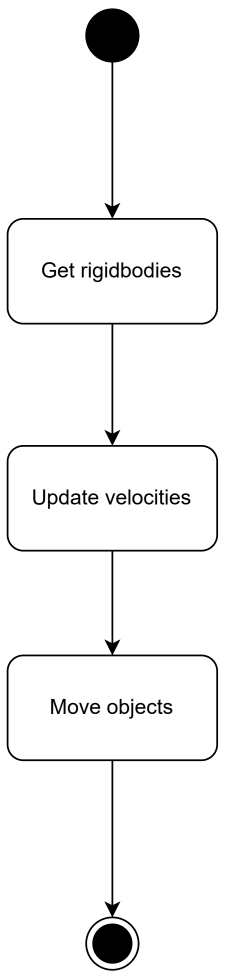

sions
ComponentManager
<<interface>>
<<singleton>>
System
+ static System & get_instance();
+ virtual void update() = 0;
# System() {};
# virtual ~System() {};
CollisionSystem
- CollisionSystem()
- ~ CollisionSystem()
+ void update() override
ıg a broad collision detection
ion has this but is does not
system.
+mass
+gravit
+body
+angul
+angul
+collisi
+const
+detec
+incluc
+linear
+linear
+max_
+max_
+use_
+boun
+get_i

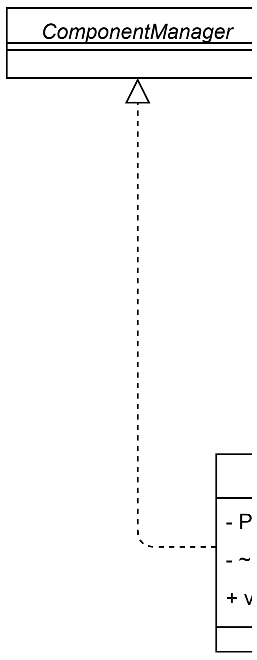


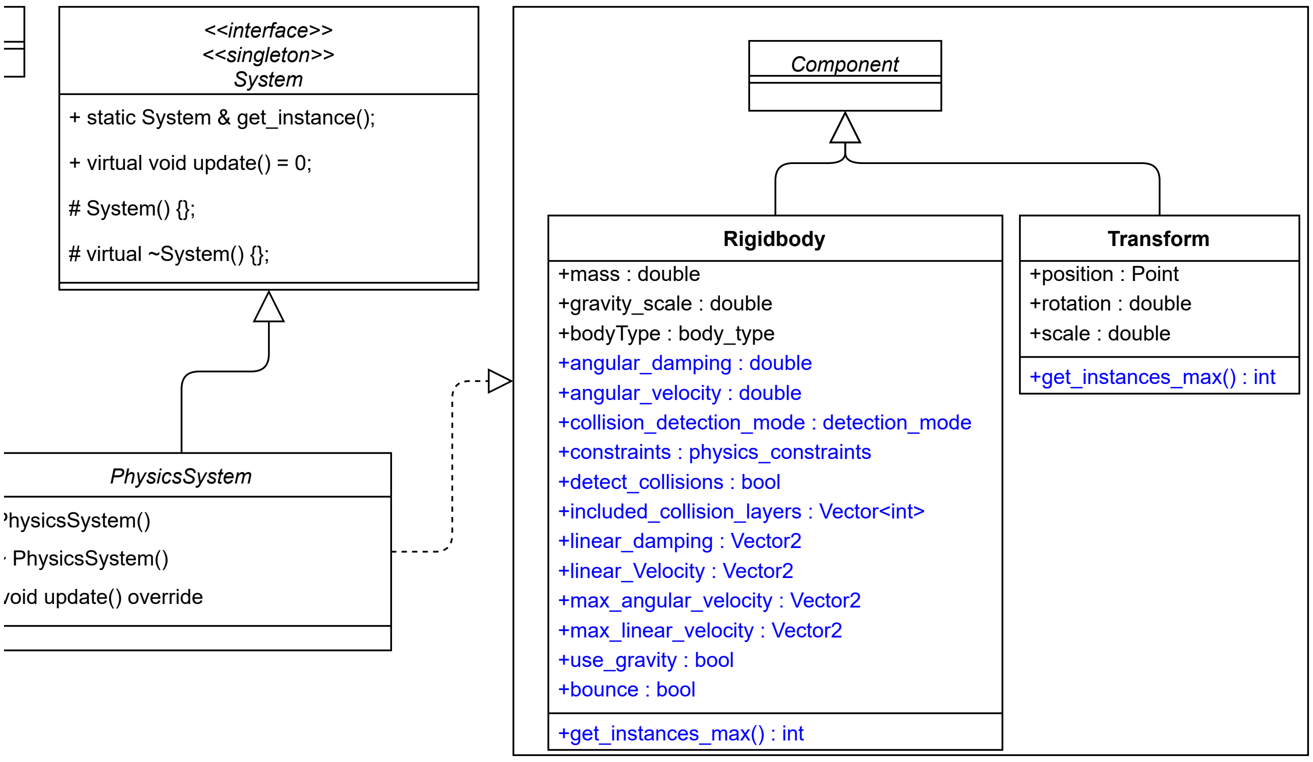
Component
Rigidbody
s : double
ty_scale : double
Type : body_type
lar_damping : double
lar_velocity : double
ion_detection_mode : detection_mode
traints : physics_constraints
ct_collisions : bool
ded_collision_layers : Vector<int>
r_damping : Vector2
r_Velocity : Vector2
_angular_velocity : Vector2
_linear_velocity : Vector2
gravity : bool
ce : bool
nstances_max() : int
Transform
+position : Point
+rotation : double
+scale : double
+get_instances_max() : int
Collider
+~Collider() : virtual
CircleCollider
+radius : double
+position : Vector2
BoxCollider
+width : double
+height : double
+position : Vector2


No collision
Collision
end of loop
Static collision
Normal collision
No collision
Static collision handle
normal collision handle

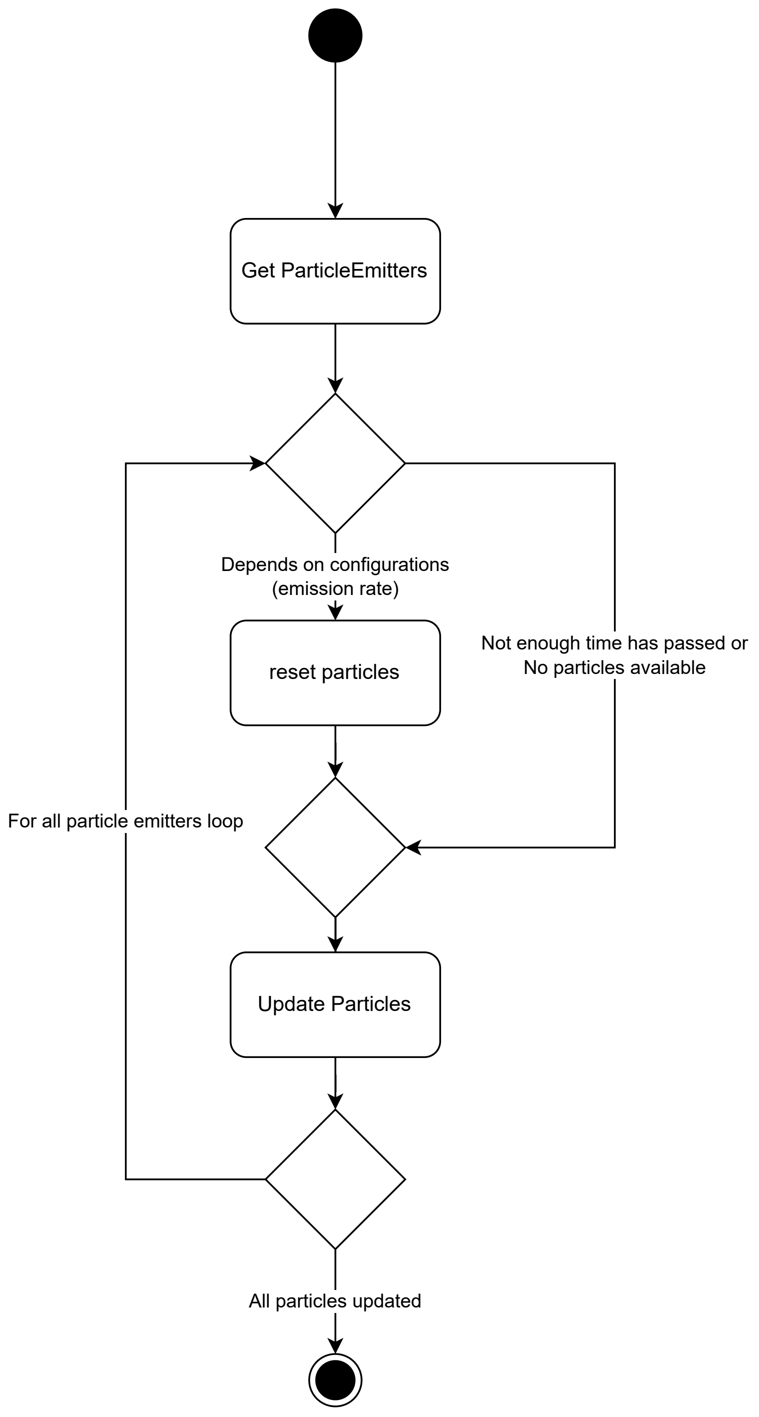
The static collision handle and normal collision handle
are event handled by the system or user scripts


Get rigidbodies
Update velocities
Move objects

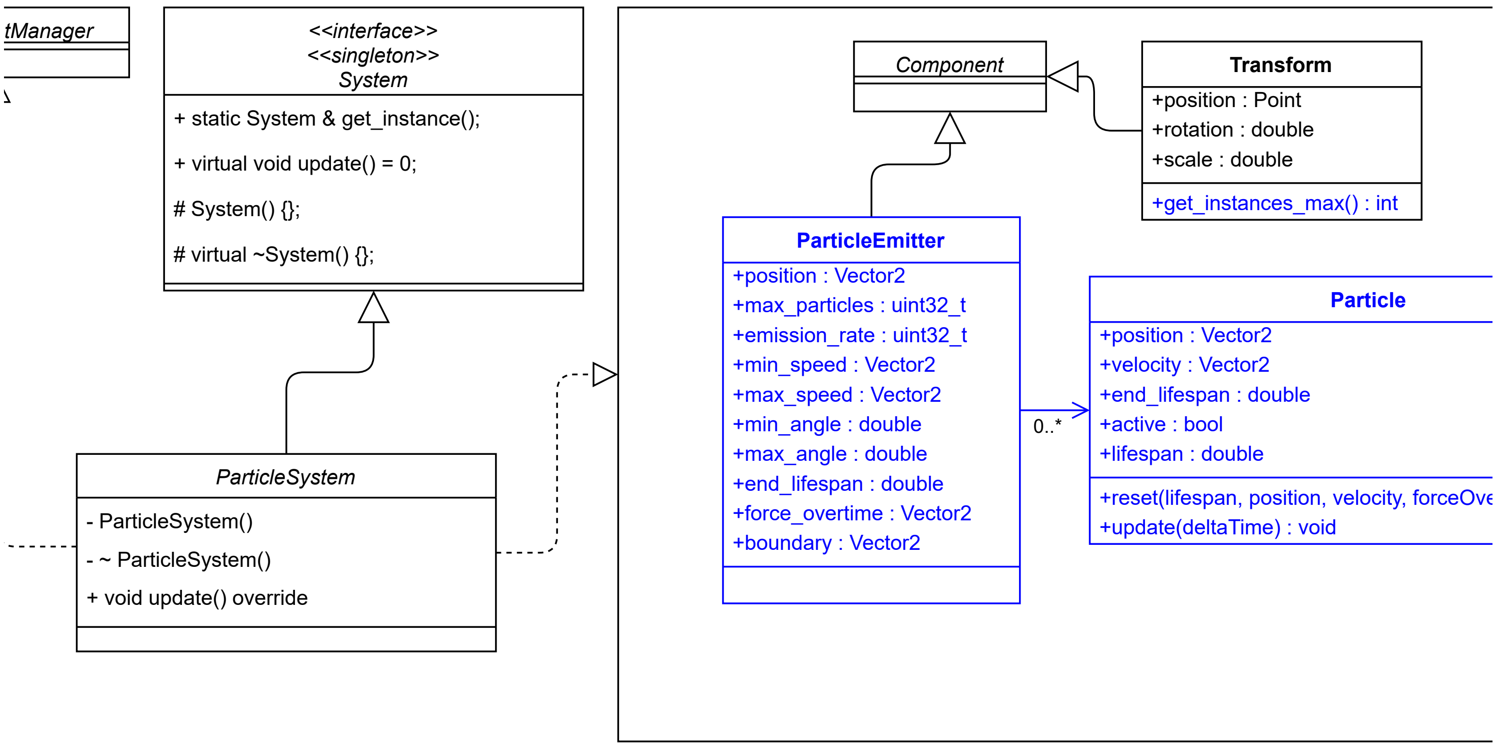

ComponentManager
- P
- ~
+ v

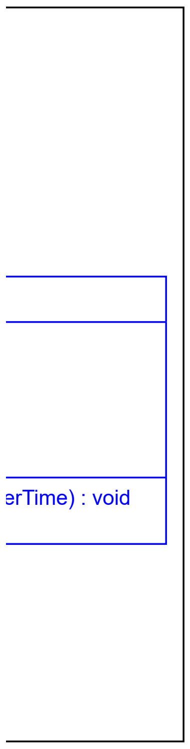


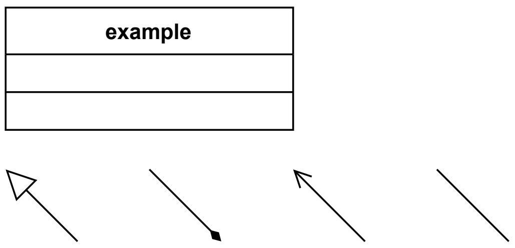
<<interface>>
<<singleton>>
System
+ static System & get_instance();
+ virtual void update() = 0;
# System() {};
# virtual ~System() {};
PhysicsSystem
'hysicsSystem()
· PhysicsSystem()
/oid update() override
Component
Rigidbody
+mass : double
+gravity_scale : double
+bodyType : body_type
+angular_damping : 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
Transform
+position : Point
+rotation : double
+scale : double
+get_instances_max() : int

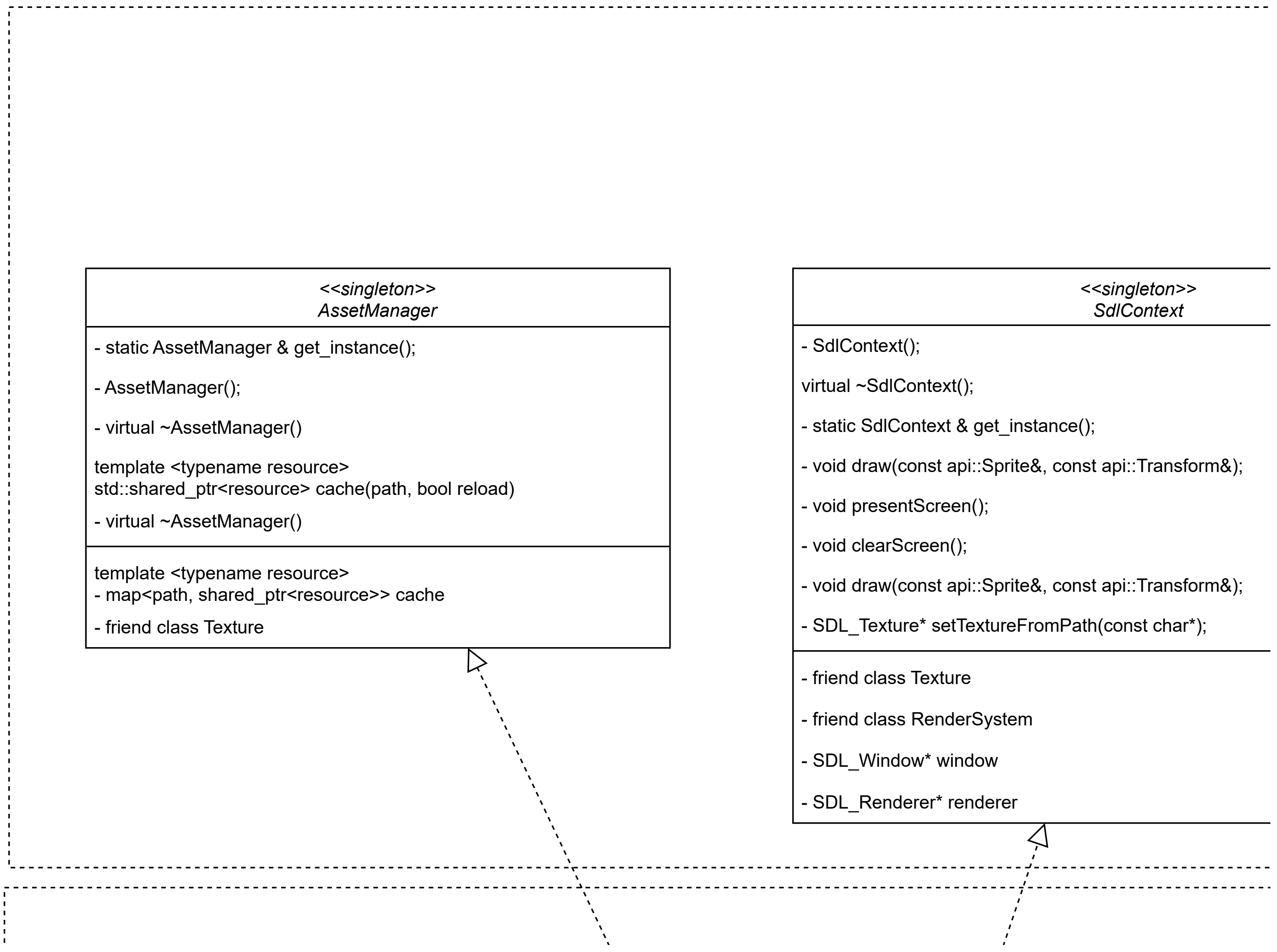

Get ParticleEmitters
Depends on configurations (emission rate)
reset particles
Not enough time has passed or No particles available
For all particle emitters loop
Update Particles
All particles updated

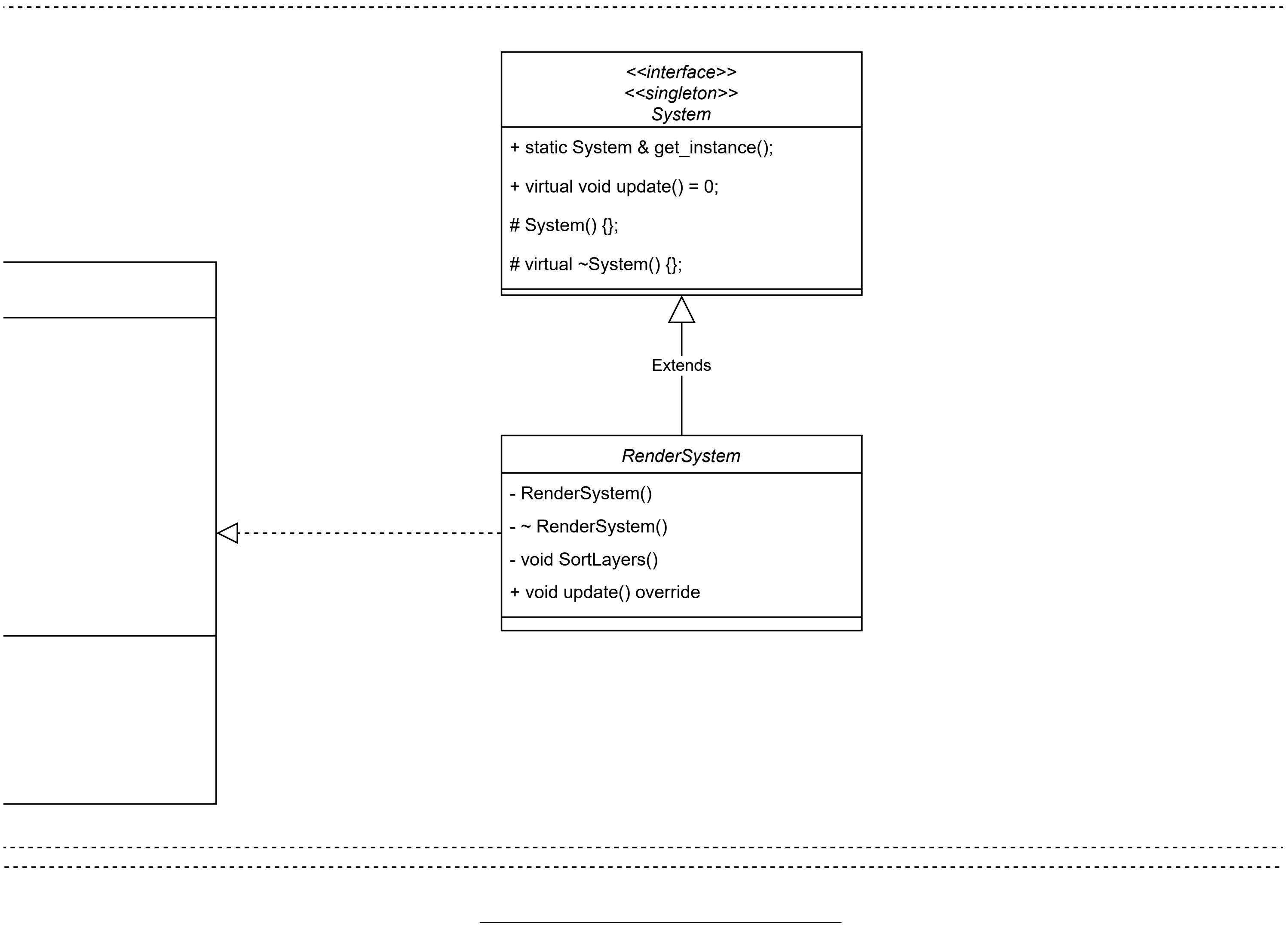
Component


tManager
<<interface>>
<<singleton>>
System
+ static System & get_instance();
+ virtual void update() = 0;
# System() {};
# virtual ~System() {};
ParticleSystem
- ParticleSystem()
- ~ ParticleSystem()
+ void update() override
Component
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
0..*
Particle
+position : Vector2
+velocity : Vector2
+end_lifespan : double
+active : bool
+lifespan : double
+reset(lifespan, position, velocity, forceOve
+update(deltaTime) : void


erTime) : void

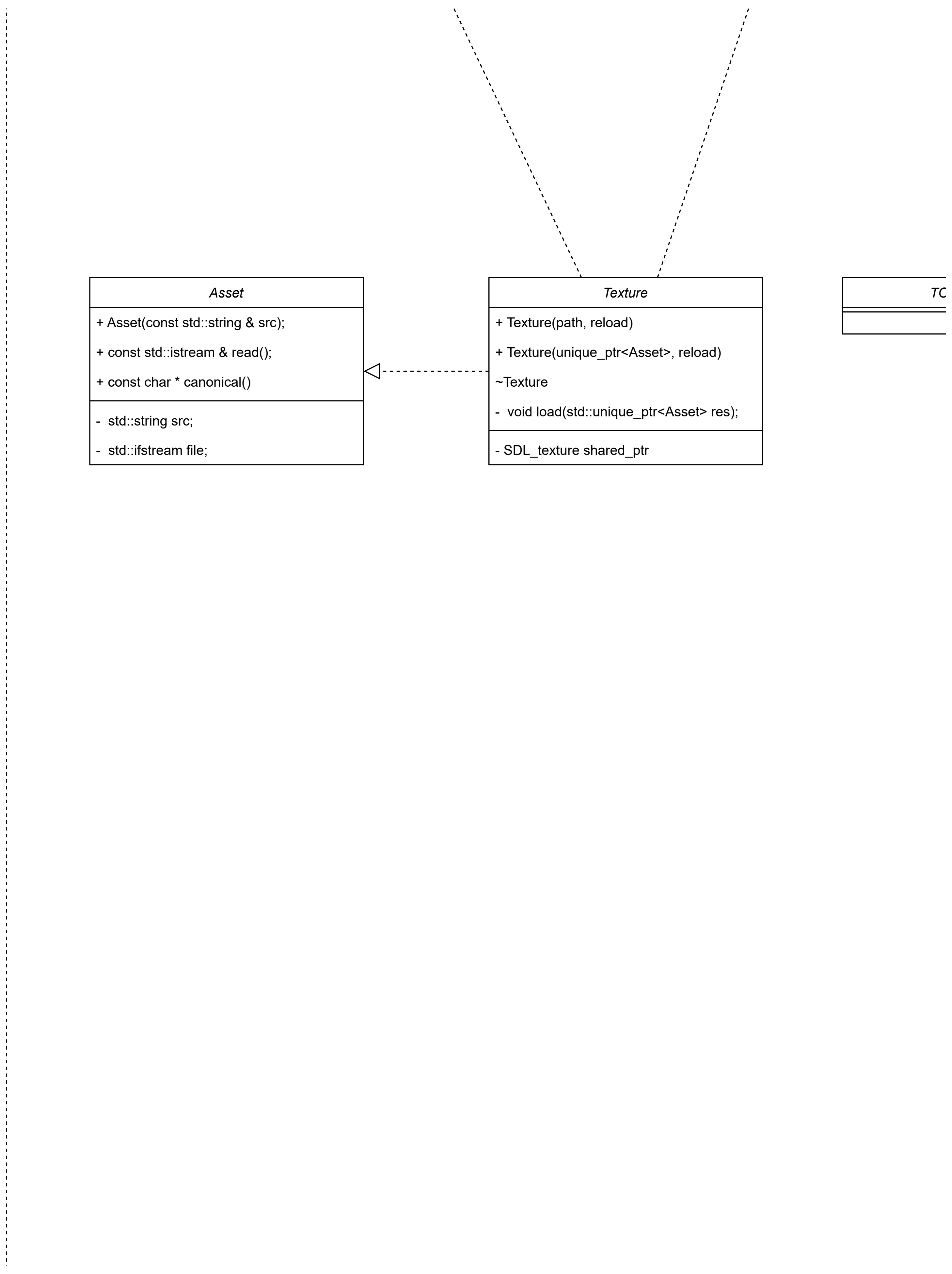

example

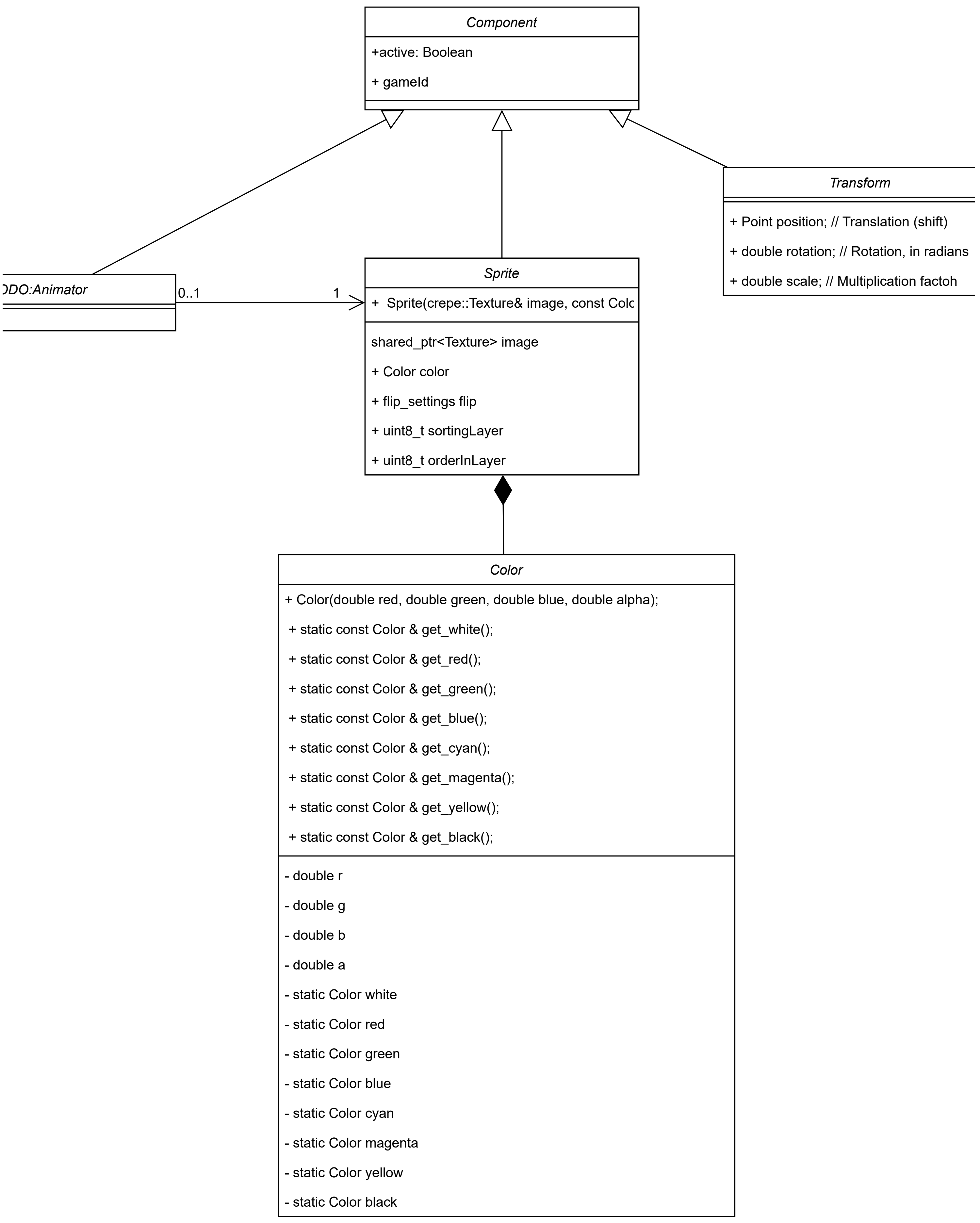

<<singleton>>
AssetManager
- static AssetManager & get_instance();
- AssetManager();
- virtual ~AssetManager()
template <typename resource>
std::shared_ptr<resource> cache(path, bool reload)
- virtual ~AssetManager()
template <typename resource>
- map<path, shared_ptr<resource>> cache
- friend class Texture
<<singleton>>
SdlContext
- SdlContext();
virtual ~SdlContext();
- static SdlContext & get_instance();
- void draw(const api::Sprite&, const api::Transform&);
- void presentScreen();
- void clearScreen();
- void draw(const api::Sprite&, const api::Transform&);
- SDL_Texture* setTextureFromPath(const char*);
- friend class Texture
- friend class RenderSystem
- SDL_Window* window
- SDL_Renderer* renderer



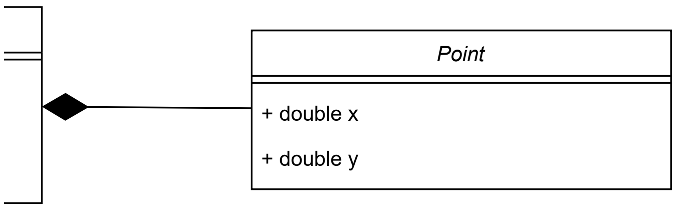
<<interface>>
<<singleton>>
System
+ static System & get_instance();
+ virtual void update() = 0;
# System() {};
# virtual ~System() {};
Extends
RenderSystem
- RenderSystem()
- ~ RenderSystem()
- void SortLayers()
+ void update() override

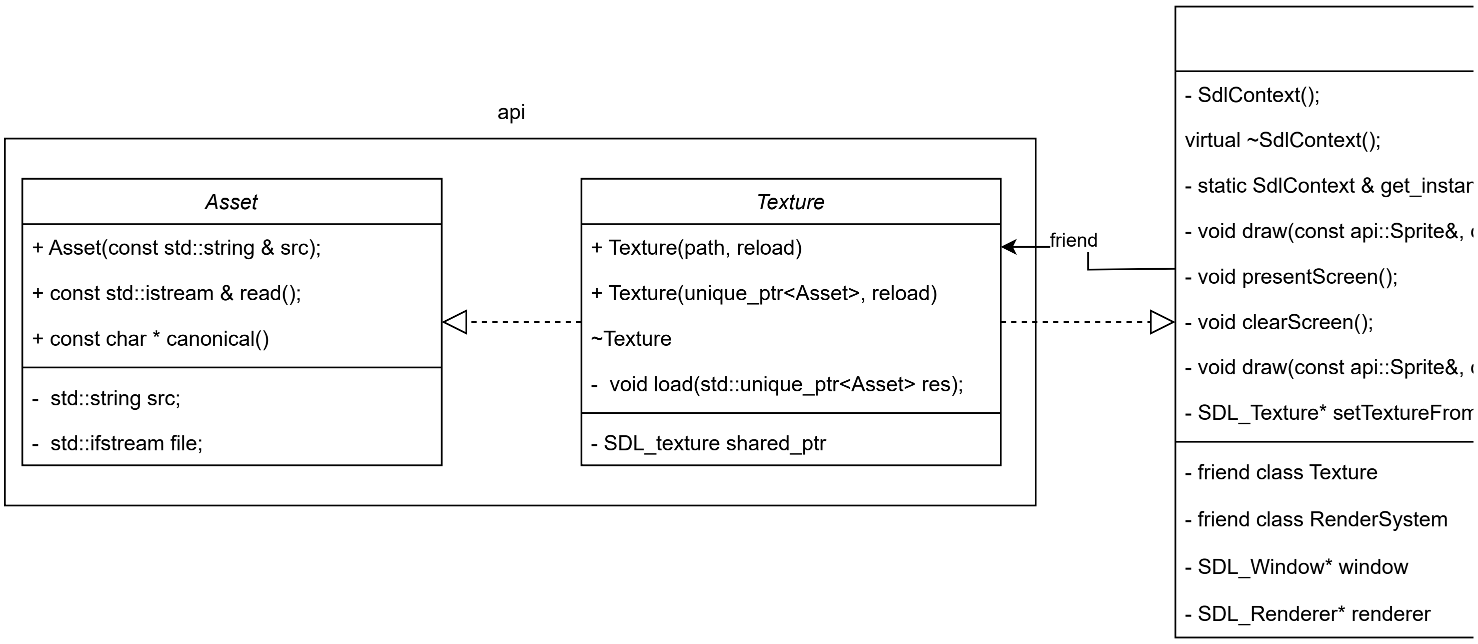
Engine

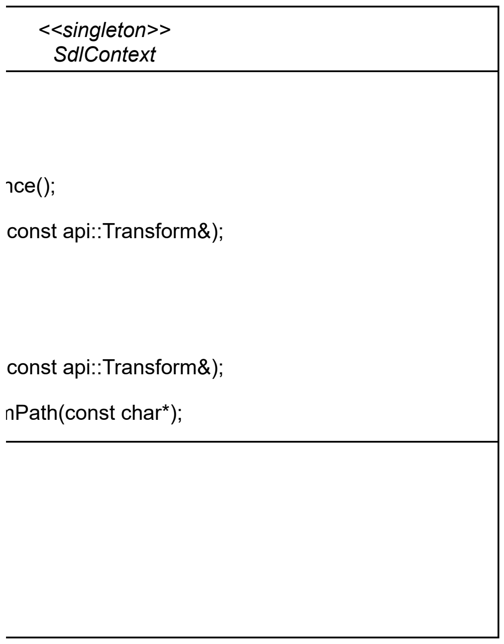

Asset
+ Asset(const std::string & src);
+ const std::istream & read();
+ const char * canonical()
- std::string src;
- std::ifstream file;
Texture
+ Texture(path, reload)
+ Texture(unique_ptr<Asset>, reload)
~Texture
- void load(std::unique_ptr<Asset> res);
- SDL_texture shared_ptr
TC

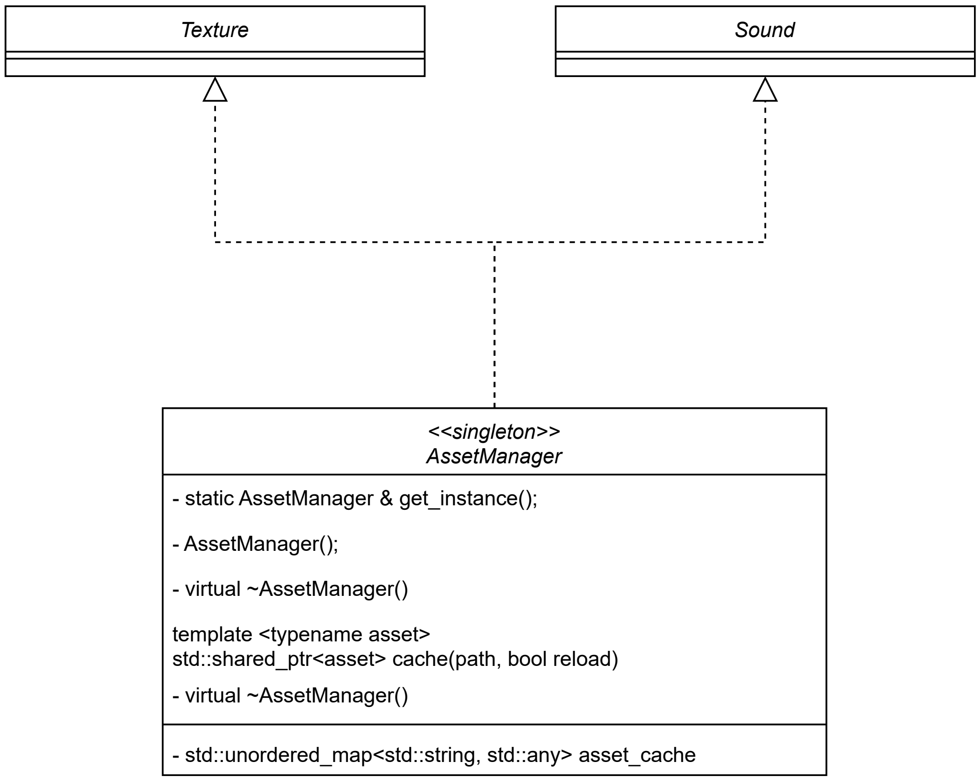

Component
+active: Boolean
+ gameId
Transform
+ Point position; // Translation (shift)
+ double rotation; // Rotation, in radians
+ double scale; // Multiplication factoh
DO:Animator
0..1
1
Sprite
+ Sprite(crepe::Texture& image, const Colc
shared_ptr<Texture> image
+ Color color
+ flip_settings flip
+ uint8_t sortingLayer
+ uint8_t orderInLayer
Color
+ Color(double red, double green, double blue, double alpha);
+ static const Color & get_white();
+ static const Color & get_red();
+ static const Color & get_green();
+ static const Color & get_blue();
+ static const Color & get_cyan();
+ static const Color & get_magenta();
+ static const Color & get_yellow();
+ static const Color & get_black();
- double r
- double g
- double b
- double a
- static Color white
- static Color red
- static Color green
- static Color blue
- static Color cyan
- static Color magenta
- static Color yellow
- static Color black

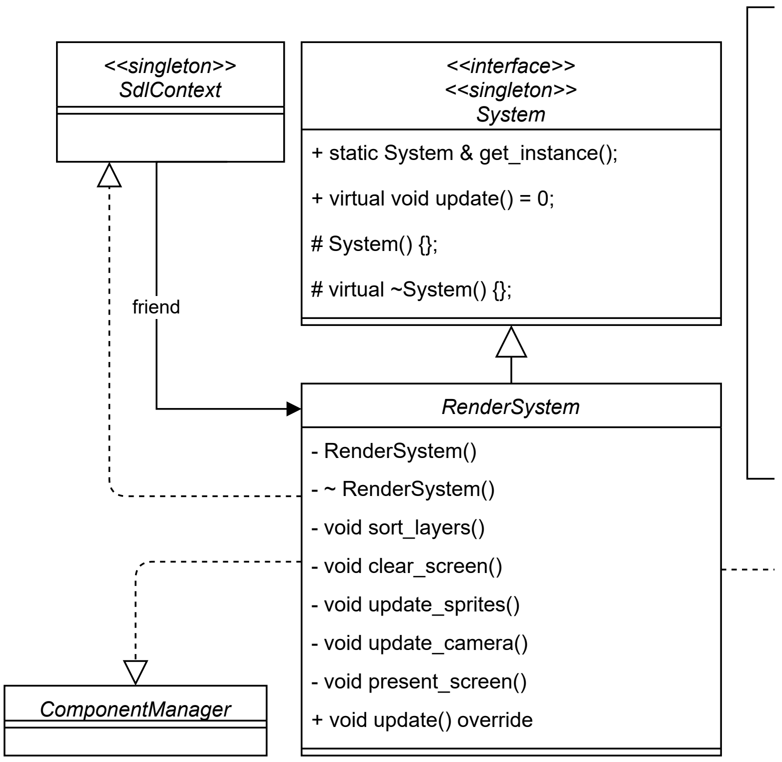
API



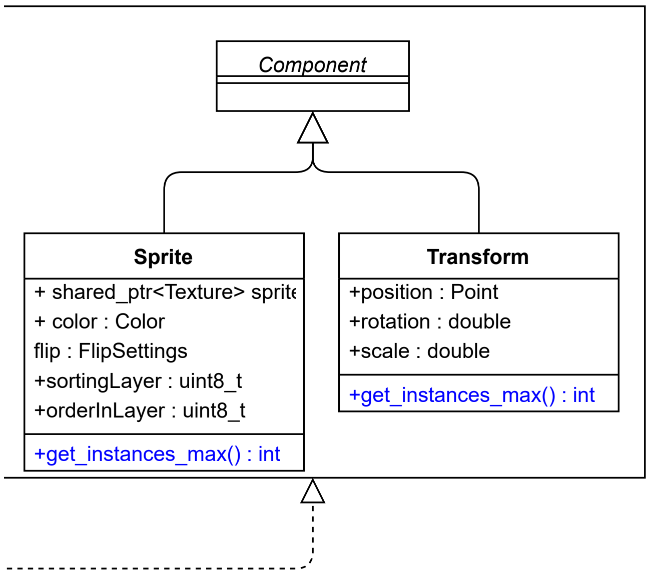

api
Asset
+ Asset(const std::string & src);
+ const std::istream & read();
+ const char * canonical()
- std::string src;
- std::ifstream file;
Texture
+ Texture(path, reload)
+ Texture(unique_ptr<Asset>, reload)
~Texture
- void load(std::unique_ptr<Asset> res);
- SDL_texture shared_ptr
friend
- SdlContext();
virtual ~SdlContext();
- static SdlContext & get_instar
- void draw(const api::Sprite&, c
- void presentScreen();
- void clearScreen();
- void draw(const api::Sprite&, c
- SDL_Texture* setTextureFrom
- friend class Texture
- friend class RenderSystem
- SDL_Window* window
- SDL_Renderer* renderer


<<singleton>>
SdlContext
nce();
const api::Transform&);
const api::Transform&);
nPath(const char*);

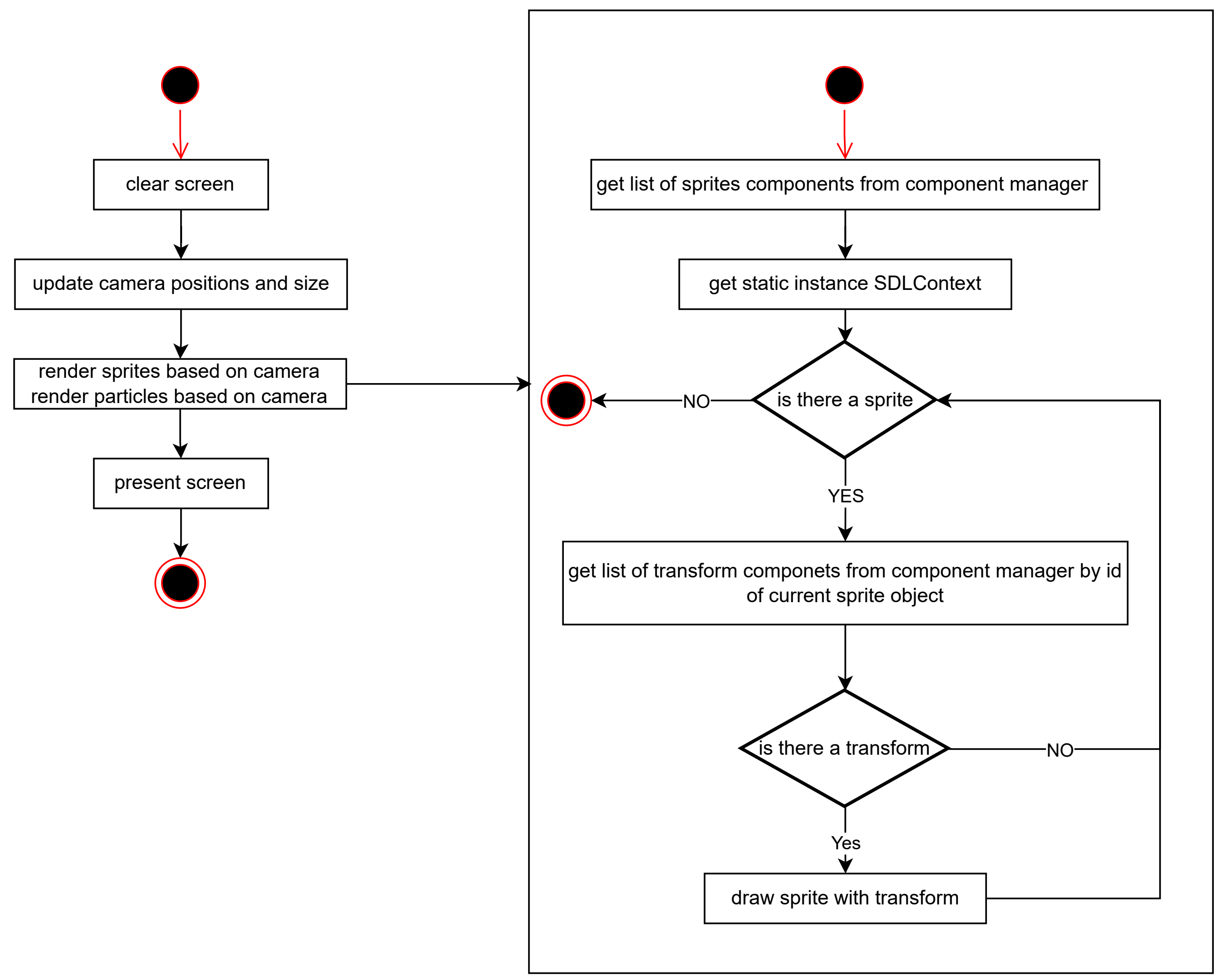

Texture
Sound
<<singleton>>
AssetManager
- static AssetManager & get_instance();
- AssetManager();
- virtual ~AssetManager()
template <typename asset>
std::shared_ptr<asset> cache(path, bool reload)
- virtual ~AssetManager()
- std::unordered_map<std::string, std::any> asset_cache

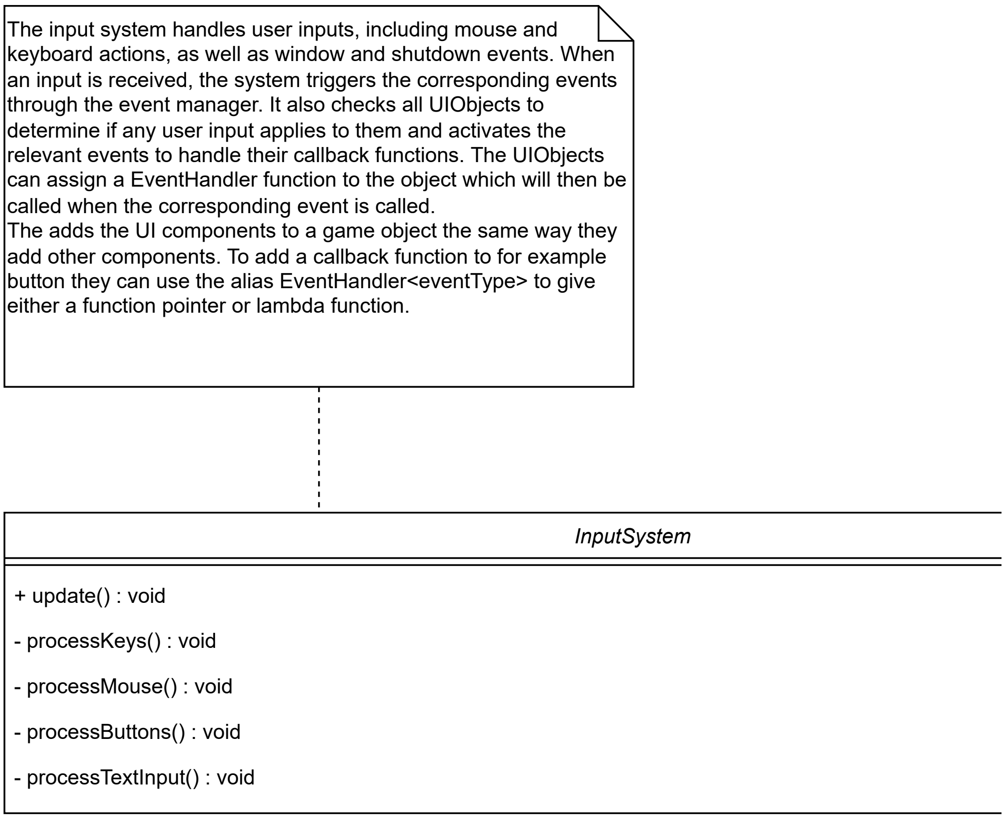


<<singleton>>
SdlContext
<<interface>>
<<singleton>>
System
+ static System & get_instance();
+ virtual void update() = 0;
# System() {};
# virtual ~System() {};
friend
RenderSystem
- RenderSystem()
- ~ RenderSystem()
- void sort_layers()
- void clear_screen()
- void update_sprites()
- void update_camera()
- void present_screen()
+ void update() override
ComponentManager

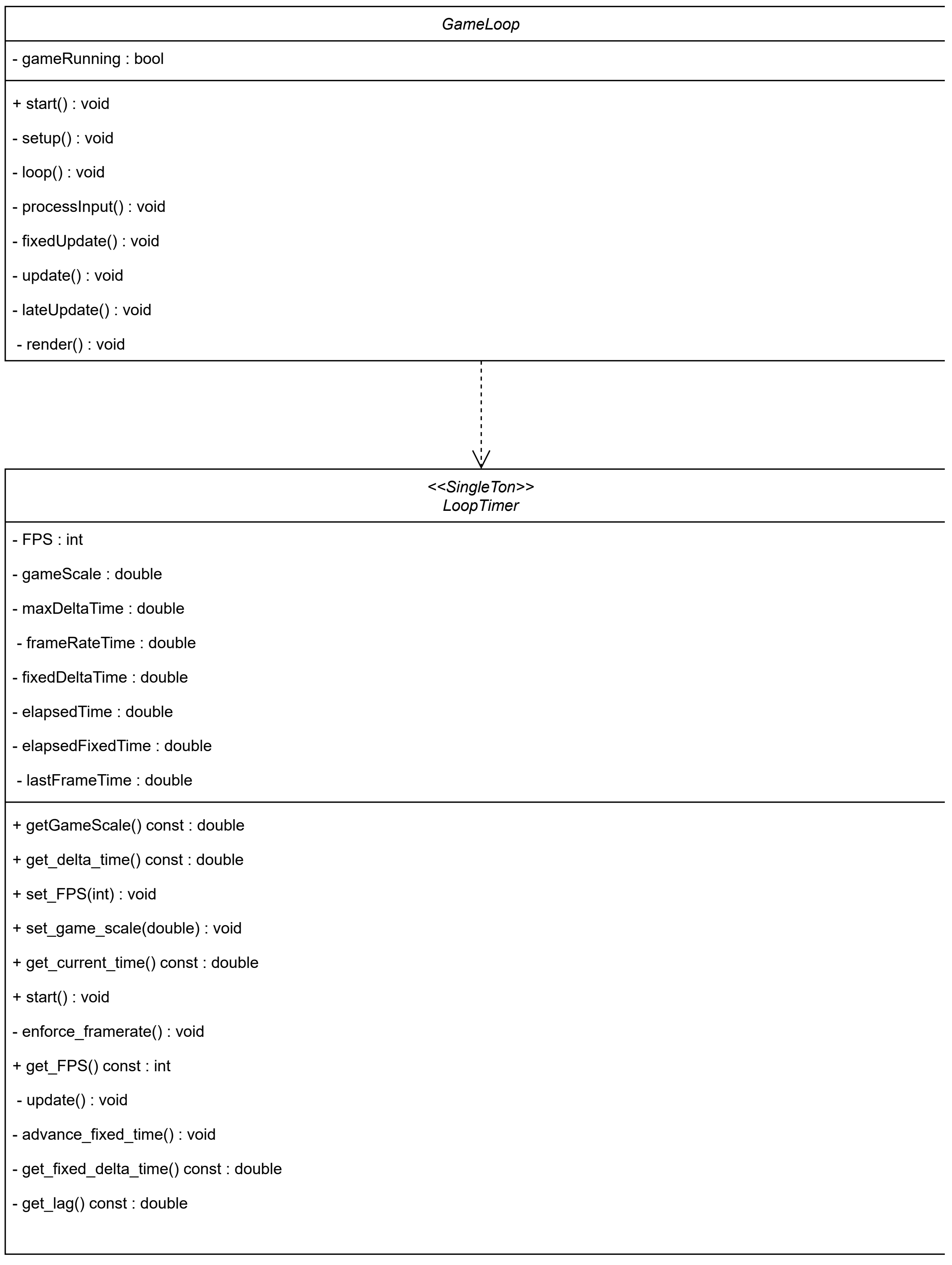


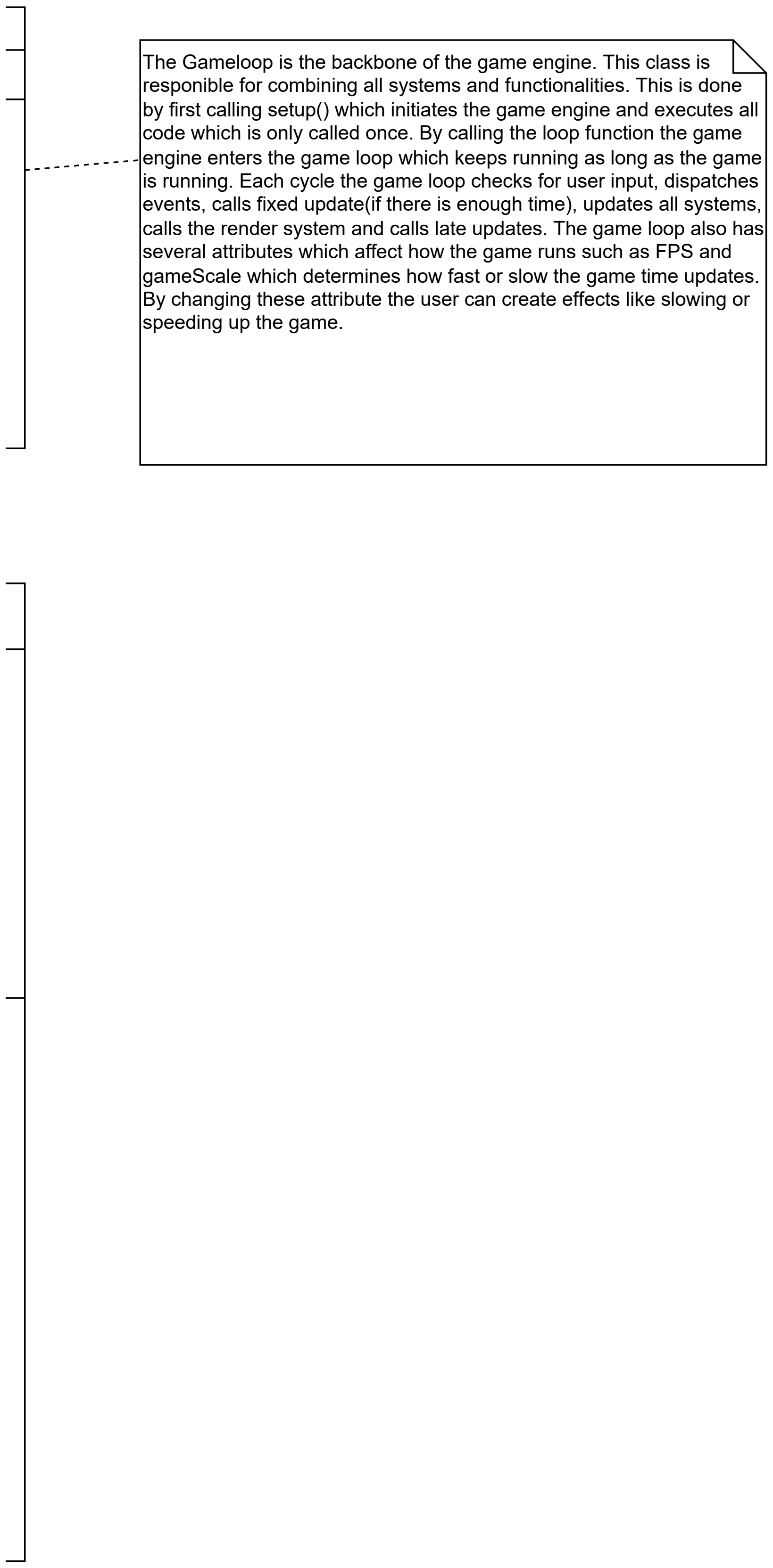
Component
Sprite
+ shared_ptr<Texture> sprite
+ color : Color
flip : FlipSettings
+sortingLayer : uint8_t
+orderInLayer : uint8_t
+get_instances_max() : int
Transform
+position : Point
+rotation : double
+scale : double
+get_instances_max() : int

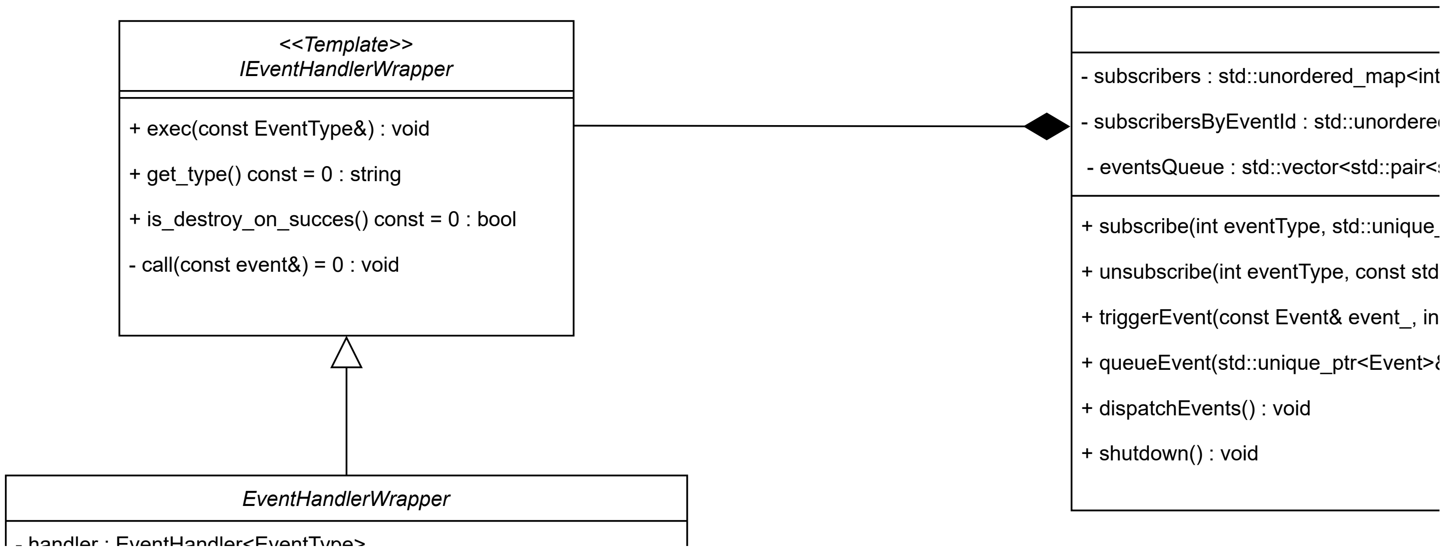

clear screen
update camera positions and size
render sprites based on camera
render particles based on camera
present screen
get list of sprites components from component manager
get static instance SDLContext
is there a sprite
NO
YES
get list of transform componets from component manager by id of current sprite object
is there a transform
NO
Yes
draw sprite with transform

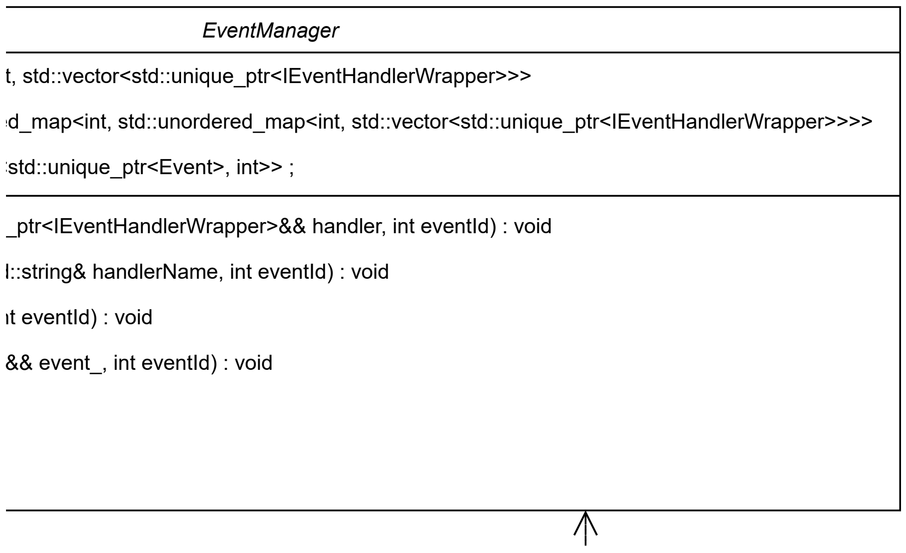


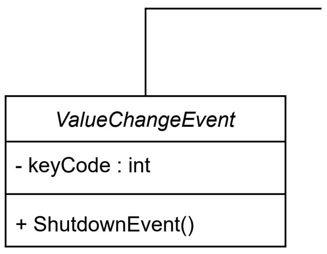
The input system handles user inputs, including mouse and keyboard actions, as well as window and shutdown events. When an input is received, the system triggers the corresponding events through the event manager. It also checks all UIObjects to determine if any user input applies to them and activates the relevant events to handle their callback functions. The UIObjects can assign a EventHandler function to the object which will then be called when the corresponding event is called.
The adds the UI components to a game object the same way they add other components. To add a callback function to for example button they can use the alias EventHandler<eventType> to give either a function pointer or lambda function.

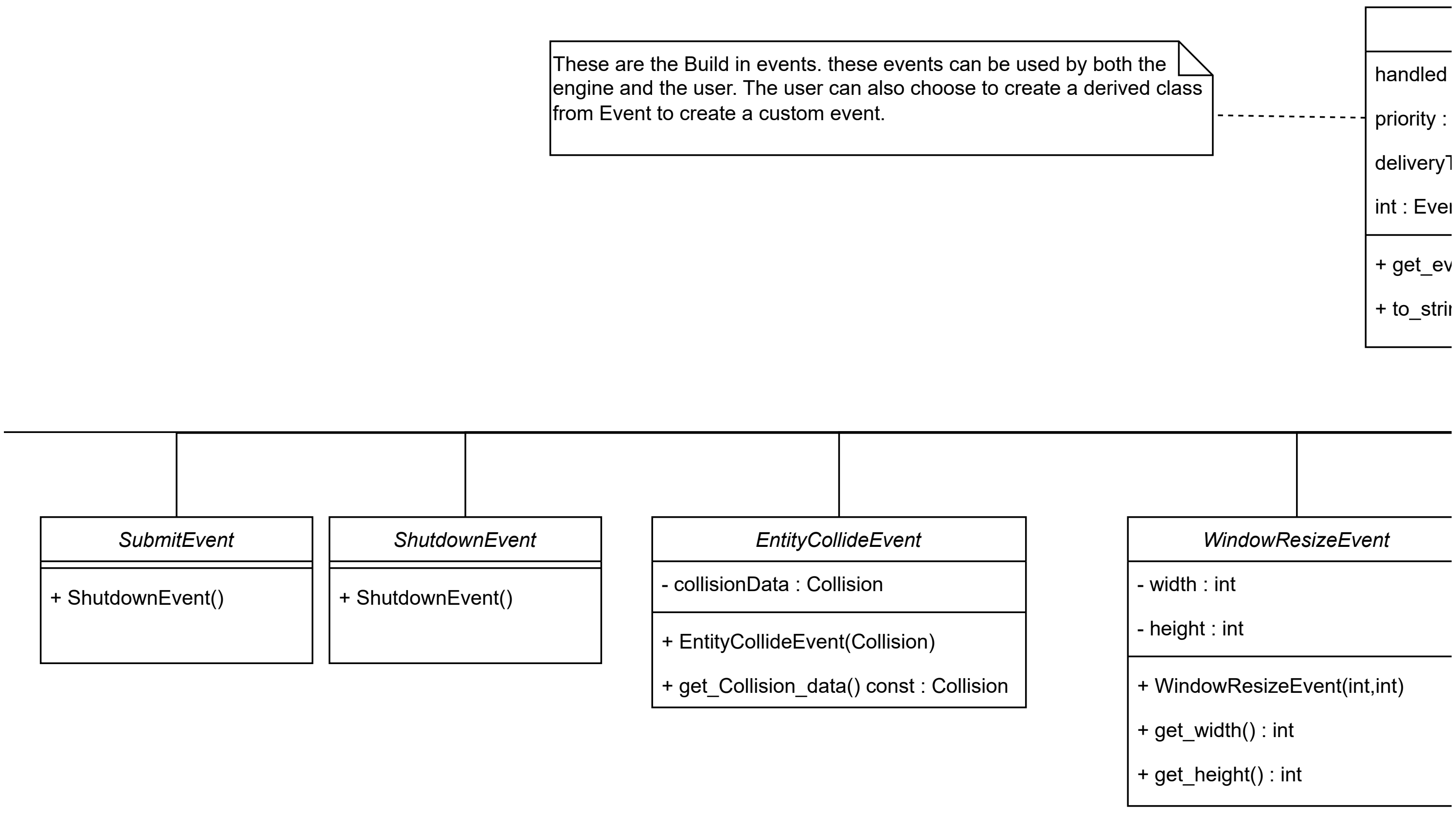

GameLoop
- gameRunning : bool
+ start() : void
- setup() : void
- loop() : void
- processInput() : void
- fixedUpdate() : void
- update() : void
- lateUpdate() : void
- render() : void
<<SingleTon>>
LoopTimer
- FPS : int
- gameScale : double
- maxDeltaTime : double
- frameRateTime : double
- fixedDeltaTime : double
- elapsedTime : double
- elapsedFixedTime : double
- lastFrameTime : double
+ getGameScale() const : double
+ get_delta_time() const : double
+ set_FPS(int) : void
+ set_game_scale(double) : void
+ get_current_time() const : double
+ start() : void
- enforce_framerate() : void
+ get_FPS() const : int
- update() : void
- advance_fixed_time() : void
- get_fixed_delta_time() const : double
- get_lag() const : double

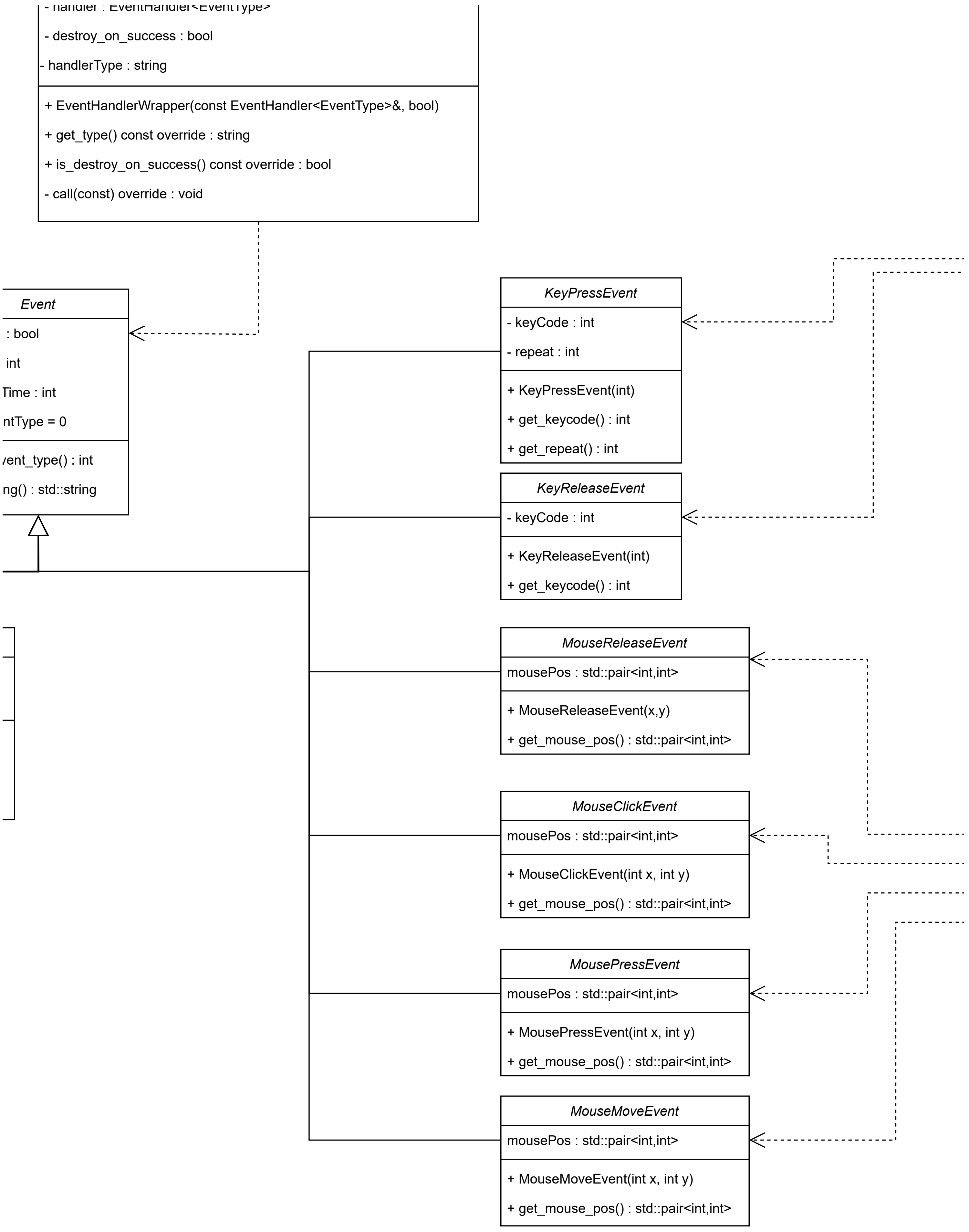

The Gameloop is the backbone of the game engine. This class is responible for combining all systems and functionalities. This is done by first calling setup() which initiates the game engine and executes all code which is only called once. By calling the loop function the game engine enters the game loop which keeps running as long as the game is running. Each cycle the game loop checks for user input, dispatches events, calls fixed update(if there is enough time), updates all systems, calls the render system and calls late updates. The game loop also has several attributes which affect how the game runs such as FPS and gameScale which determines how fast or slow the game time updates. By changing these attribute the user can create effects like slowing or speeding up the game.

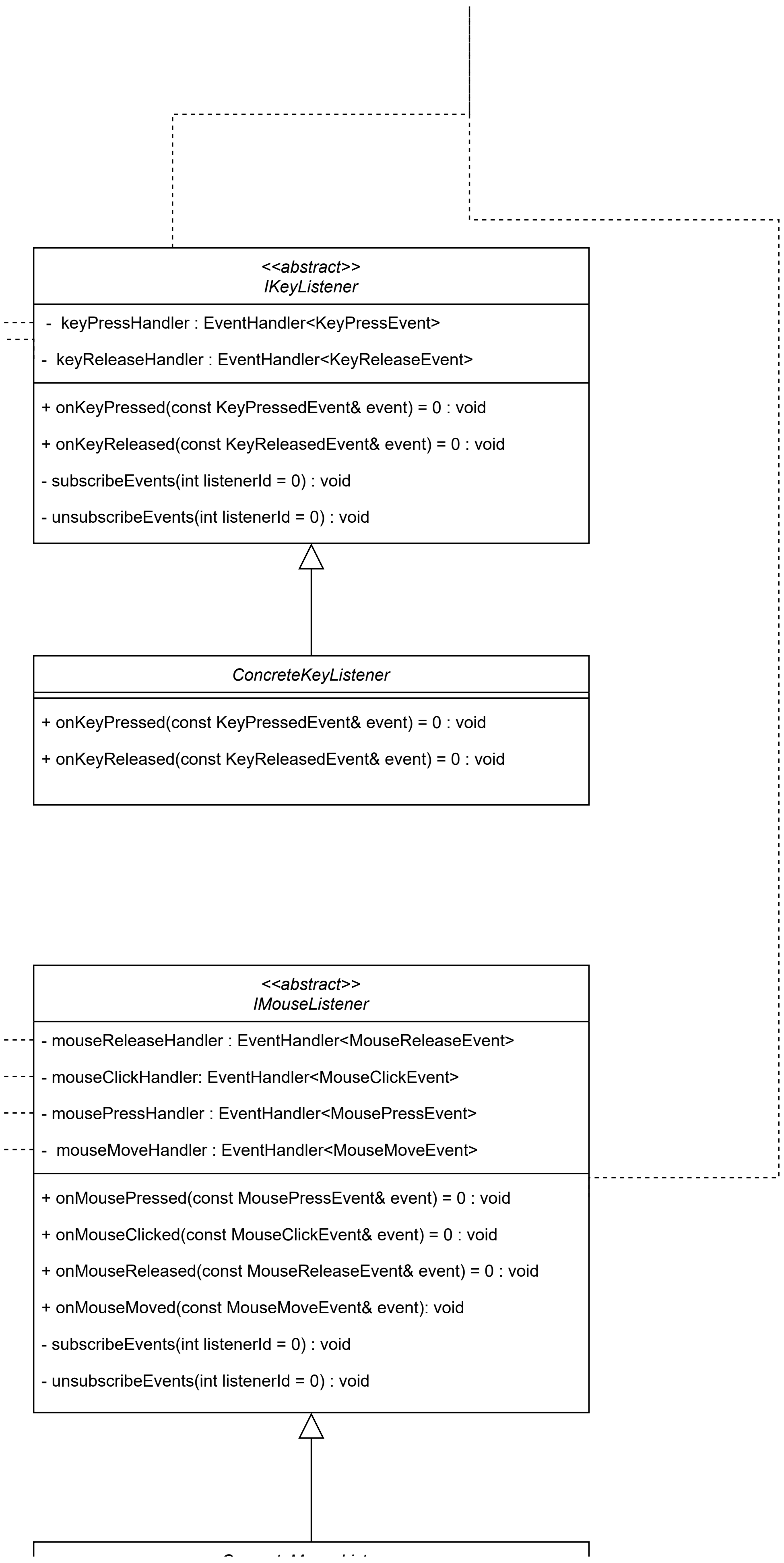

<<Template>>
IEventHandlerWrapper
+ exec(const EventType&) : void
+ get_type() const = 0 : string
+ is_destroy_on_succes() const = 0 : bool
- call(const event&) = 0 : void
EventHandlerWrapper
- handler : EventHandler<EventType>
- subscribers : std::unordered_map<int
- subscribersByEventId : std::unordere
- eventsQueue : std::vector<std::pair<
+ subscribe(int eventType, std::unique_
+ unsubscribe(int eventType, const std
+ triggerEvent(const Event& event_, in
+ queueEvent(std::unique_ptr<Event>&
+ dispatchEvents() : void
+ shutdown() : void


EventManager
t, std::vector<std::unique_ptr<IEventHandlerWrapper>>>
:d_map<int, std::unordered_map<int, std::vector<std::unique_ptr<IEventHandlerWrapper>>>>
:std::unique_ptr<Event>, int>> ;
_ptr<IEventHandlerWrapper>&& handler, int eventId) : void
l::string& handlerName, int eventId) : void
ıt eventId) : void
&& event_, int eventId) : void

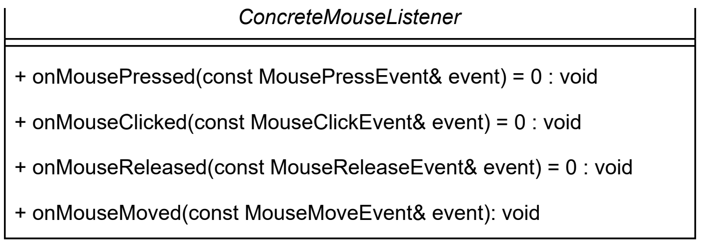

ValueChangeEvent
- keyCode : int
+ ShutdownEvent()


These are the Build in events. these events can be used by both the engine and the user. The user can also choose to create a derived class from Event to create a custom event.
handled
priority :
deliveryT
int : Even
+ get_ev
+ to_strin
SubmitEvent
+ ShutdownEvent()
ShutdownEvent
+ ShutdownEvent()
EntityCollideEvent
- collisionData : Collision
+ EntityCollideEvent(Collision)
+ get_Collision_data() const : Collision
WindowResizeEvent
- width : int
- height : int
+ WindowResizeEvent(int,int)
+ get_width() : int
+ get_height() : int


- handler : EventHandler<EventType>
- destroy_on_success : bool
- handlerType : string
+ EventHandlerWrapper(const EventHandler<EventType>&, bool)
+ get_type() const override : string
+ is_destroy_on_success() const override : bool
- call(const) override : void
Event
: bool
int
Time : int
ntType = 0
vent_type() : int
ng() : std::string
KeyPressEvent
- keyCode : int
- repeat : int
+ KeyPressEvent(int)
+ get_keycode() : int
+ get_repeat() : int
KeyReleaseEvent
- keyCode : int
+ KeyReleaseEvent(int)
+ get_keycode() : int
MouseReleaseEvent
mousePos : std::pair<int,int>
+ MouseReleaseEvent(x,y)
+ get_mouse_pos() : std::pair<int,int>
MouseClickEvent
mousePos : std::pair<int,int>
+ MouseClickEvent(int x, int y)
+ get_mouse_pos() : std::pair<int,int>
MousePressEvent
mousePos : std::pair<int,int>
+ MousePressEvent(int x, int y)
+ get_mouse_pos() : std::pair<int,int>
MouseMoveEvent
mousePos : std::pair<int,int>
+ MouseMoveEvent(int x, int y)
+ get_mouse_pos() : std::pair<int,int>

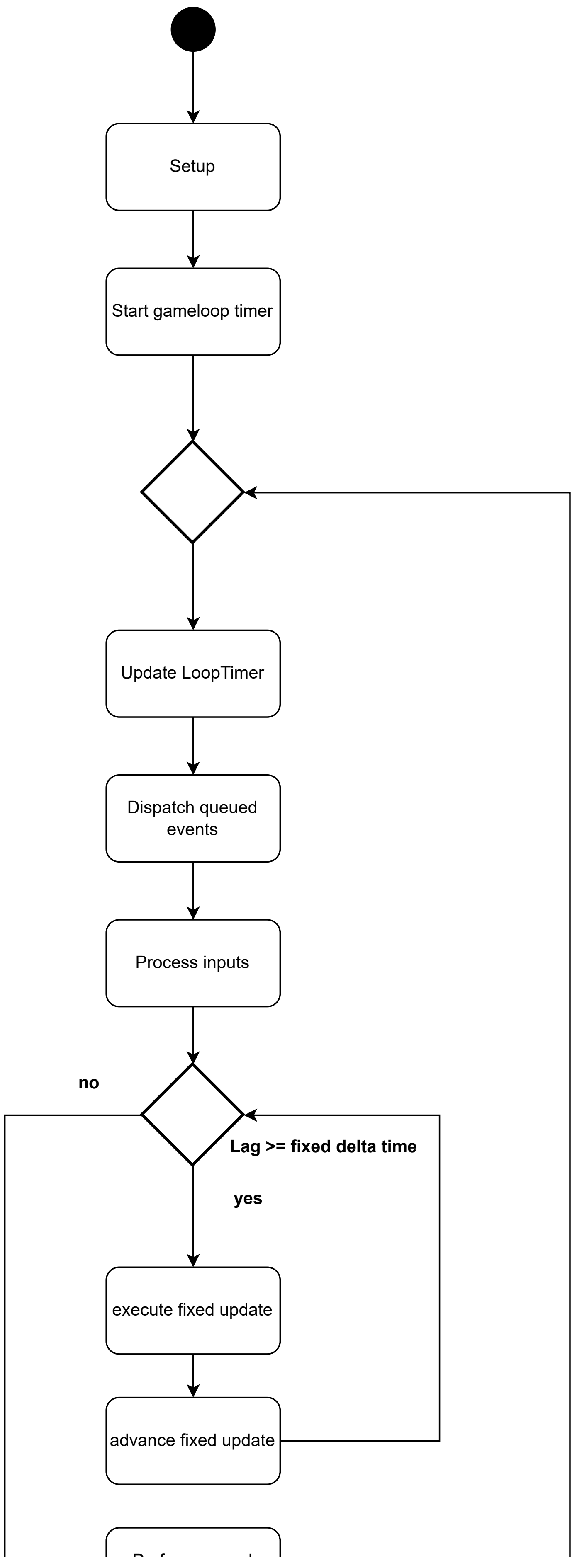


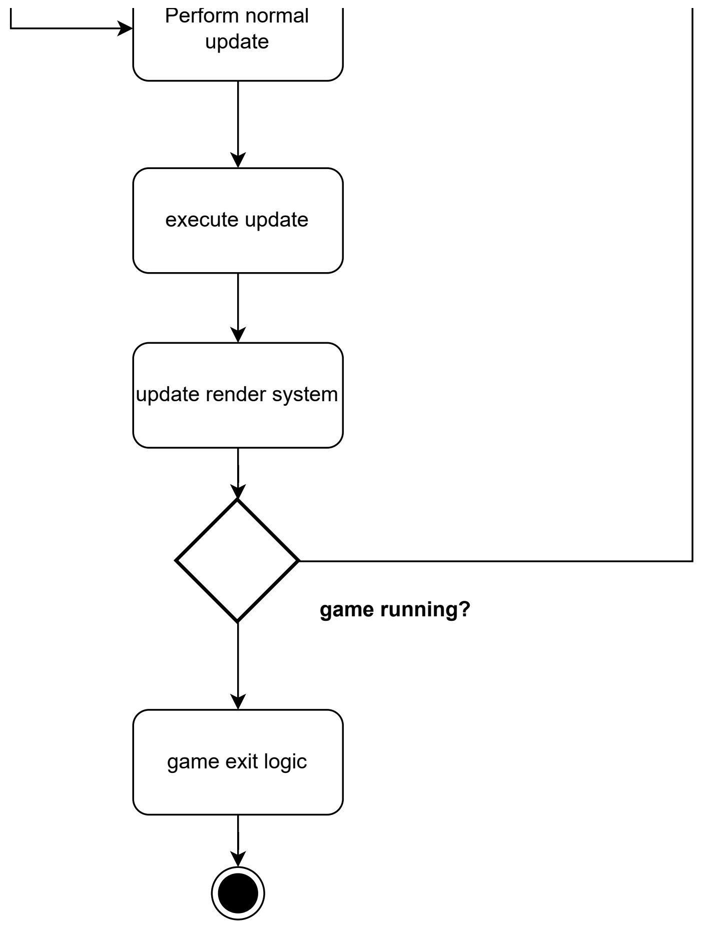
<<abstract>>
IKeyListener
- keyPressHandler : EventHandler<KeyPressEvent>
- keyReleaseHandler : EventHandler<KeyReleaseEvent>
+ onKeyPressed(const KeyPressedEvent& event) = 0 : void
+ onKeyReleased(const KeyReleasedEvent& event) = 0 : void
- subscribeEvents(int listenerId = 0) : void
- unsubscribeEvents(int listenerId = 0) : void
ConcreteKeyListener
+ onKeyPressed(const KeyPressedEvent& event) = 0 : void
+ onKeyReleased(const KeyReleasedEvent& event) = 0 : void
<<abstract>>
IMouseListener
- mouseReleaseHandler : EventHandler<MouseReleaseEvent>
- mouseClickHandler: EventHandler<MouseClickEvent>
- mousePressHandler : EventHandler<MousePressEvent>
- mouseMoveHandler : EventHandler<MouseMoveEvent>
+ onMousePressed(const MousePressEvent& event) = 0 : void
+ onMouseClicked(const MouseClickEvent& event) = 0 : void
+ onMouseReleased(const MouseReleaseEvent& event) = 0 : void
+ onMouseMoved(const MouseMoveEvent& event): void
- subscribeEvents(int listenerId = 0) : void
- unsubscribeEvents(int listenerId = 0) : void



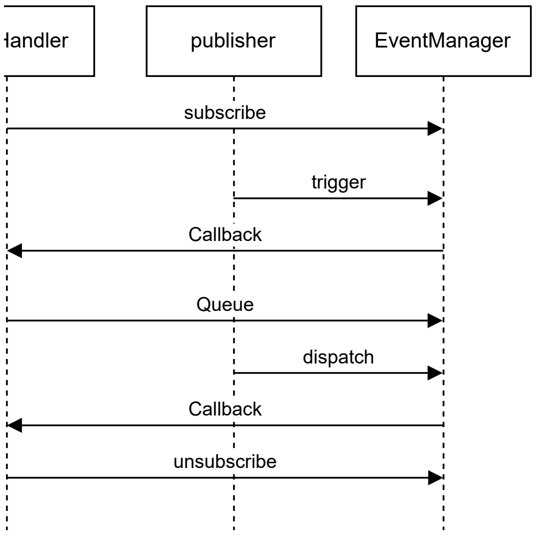
ConcreteMouseListener

```
+ onMousePressed(const MousePressEvent& event) = 0 : void
+ onMouseClicked(const MouseClickEvent& event) = 0 : void
+ onMouseReleased(const MouseReleaseEvent& event) = 0 : void
+ onMouseMoved(const MouseMoveEvent& event): void
```


Setup
Start gameloop timer
Update LoopTimer
Dispatch queued events
Process inputs
no
Lag >= fixed delta time
yes
execute fixed update
advance fixed update


Perform normal update
execute update
update render system
game running?
game exit logic

eventH


Handler
publisher
EventManager
subscribe
trigger
Callback
Queue
dispatch
Callback
unsubscribe