Simple2DEngine

Release latest

Apr 14, 2019

Contents

Simple2DEngine is a simple game engine based on SFML and written in C++.

CHAPTER 1

Build Status

Linux GCC	Linux Clang	MacOS	Windows

CHAPTER 2

Documentation

2.1 Documentation status

You can read online documentation here.

2.2 How to build documentation

First of all you need `exhale` python package that can be install by

```
pip install exhale
```

In `CMake` configuration step set `BUILD_DOCS` to `YES`. Documentation will be build by default and will be in `docs` folder.

```
mkdir build
cd build
cmake -DBUILD_DOCS=YES ..
```

If you want to install documentation you can use this after doc building.

```
cmake --build . --target install
```

Directory for installation can be set by `-DCMAKE_INSTALL_PREFIX` variable.

CHAPTER 3

Building instructions

3.1 Prerequisites

- `CMake 3.2` or newer
- `SFML 2.5.1` or newer
- Compiler with `C++14` support

3.2 Targets

- Build
  - *all*
  - *clean*
  - *install* - install binaries and docs (if builded) into *CMAKE_INSTALL_PREFIX*
  - *simple2dengine* - build Simple2DEngine libraries
- Testing (if `BUILD_UNITTESTS` set to `YES`)
  - *unit* - build and run unit tests only
  - *unitVerbose* - build and run unit tests only with verbose
- Miscellaneous
  - *doc* - build documentation
- External
  - *external-Catch-update* - update Catch (Unit Testing library)

3.3 CMake Variables

- `-DCMAKE_INSTALL_PREFIX` - location for installation
- `-DCMAKE_BUILD_TYPE` - for build type
- `-DBUILD_UNITTESTS` - boolean for Unit Tests building
- `-DBUILD_DOCS` - boolean for documentation building
- `-DBUILD_DEMO` - boolean for demo building

3.4 Build example

Debug build which will install itself into default path (`build/dist`).

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE="Debug"
cmake --build .
cmake --build . --target install
```

Release build which will install itself into `install` directory in project root dir.

```
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX="../install" -DCMAKE_BUILD_TYPE="Release"
cmake --build .
cmake --build . --target install
```

CHAPTER 4

License

Simple2DEngine uses a BSD 3-clause license.

CHAPTER 5

Content

5.1 Library API

5.1.1 Class Hierarchy

5.1.2 File Hierarchy

5.1.3 Full API

Namespaces

Namespace simple2dengine

main namespace of *Namespace simple2dengine*.

Page Contents

Classes

- *Template Struct Asset*
- *Struct BaseAsset*
- *Struct Configuration*

Classes and Structs

Template Struct Asset

- Defined in *File loader.h*

Page Contents

Inheritance Relationships

Base Type

- `public simple2dengine::BaseAsset` (*Struct BaseAsset*)

Template Parameter Order

1. `typename T`

Struct Documentation

template<typename **T**>
struct Asset : **public** simple2dengine::*BaseAsset*

Asset struct for loader.

Template Parameters

- `T`: - loaded resource.

Public Members

const T ***asset**

Struct BaseAsset

- Defined in *File loader.h*

Inheritance Relationships

Derived Type

- `public simple2dengine::Asset< T >` (*Template Struct Asset*)

Struct Documentation

struct BaseAsset

Base *Asset* for loader.

Subclassed by *simple2dengine::Asset< T >*

Struct Configuration

- Defined in *File configuration.h*

Page Contents

Struct Documentation

`struct Configuration`

Configuration Struct.

Public Members

int **`fps`** = 0

fps in engine

Window **`window`**

See *Window*.

Struct Window

- Defined in *File configuration.h*

Page Contents

Struct Documentation

`struct Window`

Window configuration.

Public Members

int **`width`** = 0

width of window

int **`height`** = 0

height of window

std::string **`name`**

name of window

Class AssetManager

- Defined in *File asset_manager.h*

Page Contents

Class Documentation

class AssetManager

Asset Manager. You can get it from *Engine*.

See *Engine*.

Public Functions

void **registerLoader**(std::shared_ptr<*Loader*> *loader*, **const** std::vector<std::string> *&extensions*)

Register loader for specific file extension.

See *Loader*.

Parameters

- `loader`: loader
- `extensions`: vector of extensions

void **load**(**const** std::string *&filename*)

Load an *Asset*.

Parameters

- `filename`: path to asset

void **unload**(**const** std::string *&filename*)

Unload an *Asset*.

Parameters

- `filename`: path to asset

template<class **T**>
const *T* ***getAsset**(**const** std::string *&filename*) **const**

Get loaded *Asset*.

Return const T* loaded asset

Template Parameters

- `T`: *Asset* class name

Parameters

- `filename`: path to asset

std::shared_ptr<*Loader*> **getLoader**(**const** std::string &*filename*) **const**

Get the *Loader* object.

Return loader

See *Loader*

Parameters

- `filename`: loader for a specific file extensions

Class CanvasNode

- Defined in *File canvas_node.h*

Page Contents

Inheritance Relationships

Base Type

- `public simple2dengine::Node` (*Class Node*)

Derived Types

- `public simple2dengine::SpriteNode` (*Class SpriteNode*)
- `public simple2dengine::TextNode` (*Class TextNode*)

Class Documentation

class CanvasNode : **public** simple2dengine::*Node*

Canvas node. Used for drawing, transforming, positioning.

Subclassed by *simple2dengine::SpriteNode*, *simple2dengine::TextNode*

Public Functions

bool **addChild**(std::shared_ptr<*CanvasNode*> *child*)

Add child to canvas node tree.

Return true if successfully add a node, otherwise return false.

Parameters

- `child`: will be added to node tree.

void **`setPosition`**(**`const`** sf::Vector2f &*position*)
Set position of *Node*.

Parameters

- `position`: x and y coordinates relative to its parent.

void **`move`**(**`const`** sf::Vector2f &*position*)
Move *Node* on specified coordinates.

Parameters

- `position`: x and y coordinate relative to its parent.

`const` sf::Vector2f &**`getPosition`**() **`const`**
Get position of *Node* relative to parent.

Return const Vector2f& x and y coordinate relative to parent.

`const` sf::Vector2f &**`getGlobalPosition`**() **`const`**
Get global position of *Node*.

Return const Vector2f& x and y coordinate relative to window (global position).

void **`setVisible`**(bool *isVisible*)
Set visibility of *Node*. If parent is invisible, curent *Node* will be invisible too.

Parameters

- `isVisible`: visibility of *Node*.

bool **`isVisible`**() **`const`**
Check if *Node* is visible or not.

Return bool visibility of *Node*.

bool **`isVisibleInTree`**() **`const`**
Check if *Node* or it parents are visible or not. If someone of *Node* or its parents is invisible - return value will be false.

Return bool visibility of *Node* or it parents.

void **`setAnchor`**(**`const`** *Anchor* *anchor*)
Set *Node* anchor.

Parameters

- `anchor`: anchor.

Anchor **getAnchor**() **const**

Get anchor of *Node*.

Return Anchor.

virtual void **updateTransform**()

Update Canvas node transform to correctly draw it.

Class Engine

- Defined in *File engine.h*

Page Contents

Class Documentation

class Engine

Engine initialization. It is a starting point for Simple2DEngine. You can init engine with *Engine* engine(config) and start with engine.run().

Public Functions

Engine(**const** *Configuration* &*config*)

Engine initialization.

See *Configuration*

Parameters

- `config`: *Configuration* for *Engine*

void **run**()

Start *Engine*. Call it after initialization and activation of scene.

void **stop**()

Stop *Engine*. You can call it in any scene. It will stop engine loop.

SceneManager &**getSceneManager**()

Get scene manager.

See *SceneManager*.

AssetManager &**getAssetManager**()

Get asset manager.

See *AssetManager*.

InputManager &**getInputManager**()
Get input manager.

See *InputManager*.

const *Configuration* &**getConfiguration**()
Get current configuration.

See *Configuration*.

sf::RenderWindow &**getRenderWindow**()
Get render window. It is used by SFML and needed for drawing objects.

Class FontLoader

- Defined in *File font_loader.h*

Page Contents

Inheritance Relationships

Base Type

- `public simple2dengine::Loader` (*Class Loader*)

Class Documentation

class FontLoader : **public** simple2dengine::*Loader*
Font *Loader*.

Public Functions

virtual void **load**(**const** std::string &*filename*)
Load asset with name.

Parameters

- `filename`: name of asset.

virtual void **unload**(**const** std::string &*filename*)
Unload asset with name.

Parameters

- `filename`: name of asset.

virtual *BaseAsset* ***getAsset**(**const** std::string &*filename*) **const**

Get loaded asset with name.

Return laoded asset

Parameters

- `filename`: name of asset.

Class InputManager

- Defined in *File input_manager.h*

Page Contents

Class Documentation

class InputManager

Scene Manager. You can get it from *Engine*.

See *Engine*.

Public Functions

void **registerAction**(**const** std::string &*action*, **const** sf::Keyboard::Key *keyboardKey*)

Attach Action to keyboard button.

Parameters

- `actionName`: - name of action.
- `keyboardKey`: - keyboard key from SFML.

void **registerAction**(**const** std::string &*action*, **const** sf::Mouse::Button *mouseButton*)

Attach Action to mouse button.

Parameters

- `actionName`: - name of action.
- `keyboardKey`: - mouse button from SFML.

void **unregisterAction**(**const** std::string &*action*)

Remove Action.

Parameters

- `actionName`: - name of action.

bool **isActionPressed**(**const** std::string &*action*) **const**

Check if action is pressed or not.

Return true - if action is pressed.

Return false - if action is not pressed.

Parameters

- `action`: - action to check.

sf::Vector2i **getMousePosition**(**const** sf::Window &*relativeTo*) **const**

Get the current position of the mouse in window coordinates. This function returns the current position of the mouse cursor, relative to the given window.

Return Current position of the mouse.

Parameters

- `relativeTo`: - Coordinates from window.

Class Loader

- Defined in *File loader.h*

Page Contents

Inheritance Relationships

Derived Types

- `public simple2dengine::FontLoader` (*Class FontLoader*)
- `public simple2dengine::SoundLoader` (*Class SoundLoader*)
- `public simple2dengine::TextureLoader` (*Class TextureLoader*)

Class Documentation

class Loader

Base *Loader*.

Subclassed by *simple2dengine::FontLoader*, *simple2dengine::SoundLoader*, *simple2dengine::TextureLoader*

Public Functions

virtual ~Loader() = 0
Destroy the *Loader* object.

virtual void **load**(**const** std::string &*filename*) = 0
Load asset with name.

Parameters

- `filename`: name of asset.

virtual void **unload**(**const** std::string &*filename*) = 0
Unload asset with name.

Parameters

- `filename`: name of asset.

virtual *BaseAsset* ***getAsset**(**const** std::string &*filename*) **const** = 0
Get loaded asset.

Return loaded asset.

Parameters

- `filename`: name of asset.

Class MusicNode

- Defined in *File music_node.h*

Page Contents

Inheritance Relationships

Base Types

- `public simple2dengine::Node` (*Class Node*)
- `public Music`

Class Documentation

class MusicNode : **public** simple2dengine::*Node*, **public** Music
Streamed music node played from an audio file.

Public Functions

void **`setMusic`**(**`const`** std::string &*filename*)
Construct a new Music *Node*.

Parameters

- `filename`: Name of music file with relative or full path.

Class Node

- Defined in *File node.h*

Page Contents

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< Node >`

Derived Types

- `public simple2dengine::CanvasNode` (*Class CanvasNode*)
- `public simple2dengine::MusicNode` (*Class MusicNode*)
- `public simple2dengine::SoundNode` (*Class SoundNode*)
- `public simple2dengine::TimerNode` (*Class TimerNode*)

Class Documentation

`class Node` : **`public`** std::enable_shared_from_this<*Node*>
Base node class. Used everywhere in *Engine*. Before run engine loop you should create any node and activate it in engine.

Subclassed by *simple2dengine::CanvasNode*, *simple2dengine::MusicNode*, *simple2dengine::SoundNode*, *simple2dengine::TimerNode*

Public Functions

Node(**const** std::string &*nodeName*)
Construct a new *Node* with ref to engine and with name.

See *Engine*.

Parameters

- `nodeName`: name of the node.

virtual ~Node()
Destroy the *Node* object.

virtual void **onCreate**()
Notifier. Will be called when node or it parent added to scene manager.

See *SceneManager*

virtual void **onEnter**()
Notifier. Will be called when node or in parent activated (become current) in scene manager.

See *SceneManager*

virtual void **onUpdate**(int)
Notifier. Will be called on every tick when node or it parent is active in scene manager.

See *SceneManager*

virtual void **onInput**(sf::Event)
Process input events like mouse movement, key press and release, etc.

See *SceneManager*

Parameters

- `event`: input event.

virtual void **onExit**()
Notifier. Will be called on every tick when node or it parent became inactive in scene manager.

See *SceneManager*

virtual void **onDestroy**()
Notifier. Will be called on every tick when node or it parent was removed from scene manager.

See *SceneManager*

bool **addChild**(std::shared_ptr<*Node*> *child*)
Add child to node tree.

Return true if successfully add a node, otherwise return false.

Parameters

- `child`: will be added to node tree.

bool **removeChild**(**const** std::string &*childName*)

Remove child from node tree.

Return true if successfully add a node, otherwise return false.

Parameters

- `childName`: name of child to remove from node tree.

void **clear**()

Remove all children from node tree.

int **getIndex**() **const**

Return Index of *Node* in its parent. If node has no parent, return 0.

Return int Index of *Node* in its parent.

const std::string &**getName**() **const**

Get the Name of *Node*.

Return const std::string& name of *Node*.

std::shared_ptr<*Node*> **getParent**() **const**

Get parent node of *Node*.

Return std::shared_ptr<Node> parent of *Node*.

std::shared_ptr<*Node*> **getRoot**()

Get root node of *Node*.

Return std::shared_ptr<Node> root of *Node*.

const std::vector<std::shared_ptr<*Node*>> &**getChildren**() **const**

Get all children of *Node*.

Return std::vector<std::shared_ptr<Node>> children.

std::shared_ptr<*Node*> **getNode**(**const** std::string &*path*)

Get node from scene tree by the provided path. Path example: "../player", "../..", "..", "player/sprite", "." ".." - it is a parent of node. "." - current node. "player/sprite" - get child with name "player" in current node, in "player" try to find child with name "sprite".

Return std::shared_ptr<Node> node in path if it exist, otherwise return nullptr.

Parameters

- `path`: path to *Node*.

Protected Functions

virtual void **update**(int *deltaInMs*)
Update logic of engine.

Parameters

- `deltaInMs`: delta time from previous update in milliseconds.

virtual void **render**()
Render and Display scene.

Protected Attributes

Engine ***engine** = nullptr

Class SceneManager

- Defined in *File scene_manager.h*

Page Contents

Class Documentation

class SceneManager
Scene Manager. You can get it from *Engine*.

See *Engine*.

Public Functions

SceneManager()
Construct a new Scene Manager object.

void **addScene**(std::shared_ptr<*Node*> *scene*)
Add node to scene manager.

Parameters

- `node`: *Node* to add.

void **removeSceneImmediately**(**const** std::string &*name*)
Remove node with a name immediately. This method is not a safe for deleting.

Parameters

- `name`: Name of scene to delete.

void **removeScene**(**const** std::string &*name*)
Add node to erasing queue. Queues a node for deletion at the next frame. When deleted, all of its child nodes will be deleted as well. This method ensures it's safe to delete the node.

Parameters

- `name`: Name of scene to delete.

void **activateScene**(**const** std::string &*name*)
Activate scene with a name. Activated scene will be displaying on a next tick.

Parameters

- `name`: Name of scene.

int **getSceneCount**() **const**
Get the Scenes Count.

Return int Scenes Count

void **clear**()
Safely remove and notify all scenes.

void **update**(int *deltaInMs*)
Update logic of engine.

Parameters

- `deltaInMs`: delta time from previous update in milliseconds

void **render**()
Render and Display scene.

void **input**(sf::Event *event*)
Process input events from SFML.

Parameters

- `event`: input event.

Class SoundLoader

- Defined in *File sound_loader.h*

Page Contents

Inheritance Relationships

Base Type

- `public simple2dengine::Loader` (*Class Loader*)

Class Documentation

`class SoundLoader` : **`public`** simple2dengine::*Loader*

Sound *Loader*.

Public Functions

`virtual` void **`load`**(**`const`** std::string &*filename*)

Load asset with name.

Parameters

- `filename`: name of asset.

`virtual` void **`unload`**(**`const`** std::string &*filename*)

Unload asset with name.

Parameters

- `filename`: name of asset.

`virtual` *BaseAsset* ***`getAsset`**(**`const`** std::string &*filename*) **`const`**

Get loaded asset with name.

Return laoded asset

Parameters

- `filename`: name of asset.

Class SoundNode

- Defined in *File sound_node.h*

Page Contents

Inheritance Relationships

Base Types

- `public simple2dengine::Node` (*Class Node*)
- `public Sound`

Class Documentation

`class SoundNode` : **`public`** simple2dengine::*Node*, **`public`** Sound
Regular sound node that can be played in the audio environment.

Public Functions

void **`setSound`**(**`const`** *AssetManager* &*assetManager*, **`const`** std::string &*filename*)
Set the source file containing the audio data to play.

See *AssetManager*.

Parameters

- `assetManager`: *Asset* Manager where asset should be stored.
- `filename`: Name of file with relative or full path.

Class SpriteNode

- Defined in *File sprite_node.h*

Page Contents

Inheritance Relationships

Base Types

- `public simple2dengine::CanvasNode` (*Class CanvasNode*)
- `public Sprite`

Class Documentation

`class SpriteNode` : **`public`** simple2dengine::*CanvasNode*, **`public`** Sprite
Sprite node. Used to draw different images.

Public Functions

void **setImage**(**const** *AssetManager* *&assetManager*, **const** std::string *&filename*)
Set or load image/texture.

See *AssetManager*.

Parameters

- `assetManager`: *Asset* Manager where asset should be stored.
- `filename`: Name of file with relative or full path.

virtual void **updateTransform**()
Update transform of the sprite to correctly display it.

Protected Functions

virtual void **render**()
Override base *render()*. We need to draw an image.

See *Node*.

Class TextNode

- Defined in *File text_node.h*

Page Contents

Inheritance Relationships

Base Types

- `public simple2dengine::CanvasNode` (*Class CanvasNode*)
- `public Text`

Class Documentation

class TextNode : **public** simple2dengine::*CanvasNode*, **public** Text
Text node. Used to draw text strings.

Public Functions

void **`setFont`**`(`**`const`** *AssetManager &assetManager*, **`const`** std::string *&filename*`)`
Set or load font.

See *AssetManager*.

Parameters

- `assetManager`: *Asset* Manager where asset should be stored.
- `filename`: Name of file with relative or full path.

void **`setString`**`(`**`const`** std::string *&textString*`)`
Set text string.

Parameters

- `textString`: - text to display.

void **`setCharacterSize`**`(`unsigned int *size*`)`
Set size.

Parameters

- `size`: of text.

`virtual` void **`updateTransform`**`()`
Update transform of the text to correctly display it.

Protected Functions

`virtual` void **`render`**`()`
Override base *render()*. We need to draw an image.

See *Node*.

Class TextureLoader

- Defined in *File texture_loader.h*

Page Contents

Inheritance Relationships

Base Type

- `public simple2dengine::Loader` (*Class Loader*)

Class Documentation

class TextureLoader : **public** simple2dengine::*Loader*

Texture *Loader*.

Public Functions

virtual void **load**(**const** std::string &*filename*)

Load asset with name.

Parameters

- `filename`: name of asset.

virtual void **unload**(**const** std::string &*filename*)

Unload asset with name.

Parameters

- `filename`: name of asset.

virtual *BaseAsset* ***getAsset**(**const** std::string &*filename*) **const**

Get loaded asset with name.

Return laoded asset

Parameters

- `filename`: name of asset.

Class TimerNode

- Defined in *File timer_node.h*

Page Contents

Inheritance Relationships

Base Type

- `public simple2dengine::Node` (*Class Node*)

Class Documentation

`class TimerNode` : **`public`** simple2dengine::*Node*

Timer *Node*. Used to set timer and execute command on timer finish.

Public Functions

`TimerNode``(`**`const`** std::string &*nodeName*, unsigned int *time* = 0, bool *isOneShot* = true`)`

Construct a new Time *Node*.

See *Engine*.

See *Node*.

Parameters

- `nodeName`: name of the node.

void **`setTime`**`(`unsigned int *time*`)`

Set finish time.

Parameters

- `time`: - time until onTimeout will be called.

void **`start`**`()`

Start timer.

void **`pause`**`()`

Pause timer.

void **`reset`**`()`

Reset timer to 0.

bool **`isPaused`**`()` **`const`**

Check if timer is paused.

Return bool pause or not.

void **`setOneShot`**`(`bool *oneShot*`)`

Set timer to one shot.

Parameters

- `oneShot`: if true - one shot timer.

bool **`isOneShot`**`()` **`const`**

Check if timer is one shot.

Return bool one shot or not.

void **onTimeout** (std::function<void)
> *function*Store a function that will be called when timeout happens. For example you can use it like this:

timer->onTimeout([this] () { sound->stop(); text->setText("Lose!"); });

Parameters

- `function`: lambda function

Protected Functions

virtual void **update** (int *deltaInMs*)
Override base *update()*. We need to calculate.

See *Node*.

Enums

Enum Anchor

- Defined in *File anchor.h*

Enum Documentation

enum simple2dengine::**Anchor**
Anchor bit flags for setAnchor.

See *CanvasNode*.

Values:

None = 0

Top = 1 << 0

Left = 1 << 1

Bottom = 1 << 2

Right = 1 << 3

Center = 1 << 4

Enum NodeState

- Defined in *File node.h*

Enum Documentation

enum simple2dengine::**NodeState**

Current state of *Node* in Scene Manager.

See *SceneManager*

See *Node*

Values:

None

Creating

Entering

Updating

Exiting

Destroying

Functions

Function simple2dengine::operator &

- Defined in *File anchor.h*

Function Documentation

constexpr enum *Anchor* simple2dengine::**operator&**(**const enum** *Anchor a*, **const enum** *Anchor b*)

Function simple2dengine::operator|

- Defined in *File anchor.h*

Function Documentation

constexpr enum *Anchor* simple2dengine::**operator|**(**const enum** *Anchor a*, **const enum** *Anchor b*)

Directories

Directory simple2dengine

Directory path: `simple2dengine`

Subdirectories

- *Directory core*
- *Directory managers*
- *Directory nodes*

Files

- *File configuration.h*
- *File engine.h*

Directory core

Parent directory (`simple2dengine`)

Directory path: `simple2dengine/core`

Files

- *File anchor.h*

Directory managers

Parent directory (`simple2dengine`)

Directory path: `simple2dengine/managers`

Subdirectories

- *Directory loaders*

Files

- *File asset_manager.h*
- *File input_manager.h*
- *File scene_manager.h*

Directory loaders

Parent directory (`simple2dengine/managers`)

Directory path: `simple2dengine/managers/loaders`

Files

- *File font_loader.h*
- *File loader.h*
- *File sound_loader.h*
- *File texture_loader.h*

Directory nodes

Parent directory (`simple2dengine`)

Directory path: `simple2dengine/nodes`

Subdirectories

- *Directory canvas*

Files

- *File music_node.h*
- *File node.h*
- *File sound_node.h*
- *File timer_node.h*

Directory canvas

Parent directory (`simple2dengine/nodes`)

Directory path: `simple2dengine/nodes/canvas`

Files

- *File canvas_node.h*
- *File sprite_node.h*
- *File text_node.h*

Files

File anchor.h

Parent directory (`simple2dengine/core`)

Page Contents

Definition (`simple2dengine/core/anchor.h`)

Program Listing for File anchor.h

Return to documentation for file (`simple2dengine/core/anchor.h`)

```
#ifndef _SIMPLE2DENGINE_CORE_ANCHOR_H_
#define _SIMPLE2DENGINE_CORE_ANCHOR_H_

namespace simple2dengine
{
    enum class Anchor : unsigned int
    {
        None = 0,
        Top = 1 << 0,
        Left = 1 << 1,
        Bottom = 1 << 2,
        Right = 1 << 3,
        Center = 1 << 4
    };

    constexpr enum Anchor operator|(const enum Anchor a, const enum Anchor b)
    {
        return static_cast<enum Anchor>(static_cast<unsigned int>(a) | static_cast
↪<unsigned int>(b));
    }

    constexpr enum Anchor operator&(const enum Anchor a, const enum Anchor b)
    {
        return static_cast<enum Anchor>(static_cast<unsigned int>(a) & static_cast
↪<unsigned int>(b));
    }
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_CORE_ANCHOR_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-03-06 Copyright (c) 2019

Included By

- *File canvas_node.h*

Namespaces

- *Namespace simple2dengine*

Enums

- *Enum Anchor*

Functions

- *Function simple2dengine::operator &*
- *Function simple2dengine::operator|*

File asset_manager.h

Parent directory (`simple2dengine/managers`)

Asset Manager.

Page Contents

Definition (`simple2dengine/managers/asset_manager.h`)

Program Listing for File asset_manager.h

Return to documentation for file (`simple2dengine/managers/asset_manager.h`)

```
#ifndef _SIMPLE2DENGINE_MANAGERS_ASSET_MANAGER_H_
#define _SIMPLE2DENGINE_MANAGERS_ASSET_MANAGER_H_

#include <iostream>
#include <memory>
#include <string>
```

(continues on next page)

(continued from previous page)

```
#include <unordered_map>
#include <vector>

#include "simple2dengine/managers/loaders/loader.h"

namespace simple2dengine
{
    class AssetManager
    {
      public:
        void registerLoader(std::shared_ptr<Loader> loader,
                            const std::vector<std::string>& extensions);
        void load(const std::string& filename);
        void unload(const std::string& filename);
        template<class T> const T* getAsset(const std::string& filename) const
        {
            std::shared_ptr<Loader> loader = getLoader(filename);
            if(!loader)
            {
                std::cout << "Error when getting asset '" << filename
                          << "': no loaders found for extension!" << std::endl;
                return nullptr;
            }

            BaseAsset* asset = loader->getAsset(filename);
            if(asset)
            {
                Asset<T>* loadedAsset = static_cast<Asset<T>*>(asset);
                if(loadedAsset != nullptr)
                {
                    const T* ret_asset = loadedAsset->asset;
                    delete asset;
                    return ret_asset;
                }
            }

            return nullptr;
        }
        std::shared_ptr<Loader> getLoader(const std::string& filename) const;

      private:
        std::unordered_map<std::string, std::shared_ptr<Loader>> loaders; // all␣
↪loaders
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_MANAGERS_ASSET_MANAGER_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-20 Copyright (c) 2019

Includes

- `iostream`
- `memory`
- `simple2dengine/managers/loaders/loader.h` (*File loader.h*)
- `string`
- `unordered_map`
- `vector`

Included By

- *File engine.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class AssetManager*

File canvas_node.h

Parent directory (`simple2dengine/nodes/canvas`)

Canvas Node.

Page Contents

Definition (`simple2dengine/nodes/canvas/canvas_node.h`)

Program Listing for File canvas_node.h

Return to documentation for file (`simple2dengine/nodes/canvas/canvas_node.h`)

```
#ifndef _SIMPLE2DENGINE_NODES_CANVAS_CANVAS_NODE_H_
#define _SIMPLE2DENGINE_NODES_CANVAS_CANVAS_NODE_H_

#include <string>

#include "simple2dengine/core/anchor.h"
#include "simple2dengine/engine.h"
#include "simple2dengine/nodes/node.h"

namespace simple2dengine
{
    class CanvasNode : public Node
    {
      public:
        using Node::Node;
        bool addChild(std::shared_ptr<CanvasNode> child);
        void setPosition(const sf::Vector2f& position);
        void move(const sf::Vector2f& position);
        const sf::Vector2f& getPosition() const;
        const sf::Vector2f& getGlobalPosition() const;
        void setVisible(bool isVisible);
        bool isVisible() const;
        bool isVisibleInTree() const;
        void setAnchor(const Anchor anchor);
        Anchor getAnchor() const;
        virtual void updateTransform();

      private:
        bool visible = true;                                      // visibility of node
        Anchor anchor = Anchor::Top | Anchor::Left;               // anchor of node
        sf::Vector2f position = sf::Vector2f(0.0f, 0.0f);        // position that␣
↪relative to parents
        sf::Vector2f globalPosition = sf::Vector2f(0.0f, 0.0f); // global position
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_NODES_CANVAS_CANVAS_NODE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-03-05 Copyright (c) 2019

Includes

- `simple2dengine/core/anchor.h` (*File anchor.h*)
- `simple2dengine/engine.h` (*File engine.h*)
- `simple2dengine/nodes/node.h` (*File node.h*)
- `string`

Included By

- *File sprite_node.h*

- *File text_node.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class CanvasNode*

File configuration.h

Parent directory (`simple2dengine`)

Configuration class.

Page Contents

Definition (`simple2dengine/configuration.h`)

Program Listing for File configuration.h

Return to documentation for file (`simple2dengine/configuration.h`)

```
#ifndef _SIMPLE2DENGINE_CONFIGURATION_H_
#define _SIMPLE2DENGINE_CONFIGURATION_H_

#include <string>

namespace simple2dengine
{
    struct Window
    {
        int width = 0;
        int height = 0;
        std::string name;
    };

    struct Configuration
    {
```

(continues on next page)

(continued from previous page)

```
        int fps = 0;
        Window window;
    };

} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_CONFIGURATION_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-20 Copyright (c) 2019

Includes

- `string`

Included By

- *File engine.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Struct Configuration*
- *Struct Window*

File engine.h

Parent directory (`simple2dengine`)

Engine class.

Page Contents

Definition (`simple2dengine/engine.h`)

Program Listing for File engine.h

Return to documentation for file (`simple2dengine/engine.h`)

```
#ifndef _SIMPLE2DENGINE_ENGINE_H_
#define _SIMPLE2DENGINE_ENGINE_H_

#include <memory>
#include <string>

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

#include "simple2dengine/configuration.h"
#include "simple2dengine/managers/asset_manager.h"
#include "simple2dengine/managers/input_manager.h"
#include "simple2dengine/managers/loaders/font_loader.h"
#include "simple2dengine/managers/loaders/sound_loader.h"
#include "simple2dengine/managers/loaders/texture_loader.h"
#include "simple2dengine/managers/scene_manager.h"

namespace simple2dengine
{
    class Engine
    {
      public:
        Engine(const Configuration& config);
        void run();
        void stop();
        SceneManager& getSceneManager();
        AssetManager& getAssetManager();
        InputManager& getInputManager();
        const Configuration& getConfiguration();
        sf::RenderWindow& getRenderWindow();

      private:
        void update(int deltaInMs);
        void render();

        sf::RenderWindow window;     // SFML window
        sf::Clock deltaClock;        // Help to calculate delta for update method
        bool isRunning = false;      // Running state
        Configuration configuration; // Configuration object (window size, fps, etc)
        // managers
        SceneManager sceneManager; // operates with scenes
        AssetManager assetManager; // operates with assets
        InputManager inputManager; // operates with input
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_ENGINE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-17 Copyright (c) 2019

Includes

- `SFML/Graphics.hpp`
- `SFML/System.hpp`
- `SFML/Window.hpp`
- `memory`
- `simple2dengine/configuration.h` (*File configuration.h*)
- `simple2dengine/managers/asset_manager.h` (*File asset_manager.h*)
- `simple2dengine/managers/input_manager.h` (*File input_manager.h*)
- `simple2dengine/managers/loaders/font_loader.h` (*File font_loader.h*)
- `simple2dengine/managers/loaders/sound_loader.h` (*File sound_loader.h*)
- `simple2dengine/managers/loaders/texture_loader.h` (*File texture_loader.h*)
- `simple2dengine/managers/scene_manager.h` (*File scene_manager.h*)
- `string`

Included By

- *File canvas_node.h*
- *File sprite_node.h*
- *File text_node.h*
- *File music_node.h*
- *File sound_node.h*
- *File timer_node.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class Engine*

File font_loader.h

Parent directory (simple2dengine/managers/loaders)

Font Loader.

Page Contents

Definition (simple2dengine/managers/loaders/font_loader.h)

Program Listing for File font_loader.h

Return to documentation for file (simple2dengine/managers/loaders/font_loader.h)

```
#ifndef _SIMPLE2DENGINE_MANAGERS_LOADERS_FONT_LOADER_H_
#define _SIMPLE2DENGINE_MANAGERS_LOADERS_FONT_LOADER_H_

#include <memory>
#include <string>
#include <unordered_map>

#include "simple2dengine/managers/loaders/loader.h"

#include "SFML/Graphics/Font.hpp"

namespace simple2dengine
{
    class FontLoader : public Loader
    {
      public:
        virtual void load(const std::string& filename) final;
        virtual void unload(const std::string& filename) final;
        virtual BaseAsset* getAsset(const std::string& filename) const final;

      private:
        std::unordered_map<std::string, sf::Font> fonts; // Loaded fonts
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_MANAGERS_LOADERS_FONT_LOADER_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-23 Copyright (c) 2019

Includes

- `SFML/Graphics/Font.hpp`
- `memory`
- `simple2dengine/managers/loaders/loader.h` (*File loader.h*)
- `string`
- `unordered_map`

Included By

- *File engine.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class FontLoader*

File input_manager.h

Parent directory (`simple2dengine/managers`)

Input Manager.

Page Contents

Definition (`simple2dengine/managers/input_manager.h`)

Program Listing for File input_manager.h

Return to documentation for file (`simple2dengine/managers/input_manager.h`)

```
#ifndef _SIMPLE2DENGINE_MANAGERS_INPUT_MANAGER_H_
#define _SIMPLE2DENGINE_MANAGERS_INPUT_MANAGER_H_

#include <string>
#include <unordered_map>
#include <vector>

#include "SFML/Window/Keyboard.hpp"
#include "SFML/Window/Mouse.hpp"

#include "simple2dengine/nodes/node.h"

namespace simple2dengine
{
    class InputManager
    {
      public:
        void registerAction(const std::string& action, const sf::Keyboard::Key␣
↪keyboardKey);
        void registerAction(const std::string& action, const sf::Mouse::Button␣
↪mouseButton);
        void unregisterAction(const std::string& action);
        bool isActionPressed(const std::string& action) const;
        sf::Vector2i getMousePosition(const sf::Window& relativeTo) const;

      private:
        std::unordered_map<std::string, std::vector<sf::Keyboard::Key>>
            keyboardActions; // actions for keyboard buttons
        std::unordered_map<std::string, std::vector<sf::Mouse::Button>>
            mouseActions; // actions for mouse buttons
    };

} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_MANAGERS_INPUT_MANAGER_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-25 Copyright (c) 2019

Includes

- `SFML/Window/Keyboard.hpp`
- `SFML/Window/Mouse.hpp`
- `simple2dengine/nodes/node.h` (*File node.h*)
- `string`

- `unordered_map`
- `vector`

Included By

- *File engine.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class InputManager*

File loader.h

Parent directory (`simple2dengine/managers/loaders`)

Base Loader.

Page Contents

Definition (`simple2dengine/managers/loaders/loader.h`)

Program Listing for File loader.h

Return to documentation for file (`simple2dengine/managers/loaders/loader.h`)

```
#ifndef _SIMPLE2DENGINE_MANAGERS_LOADERS_LOADER_H_
#define _SIMPLE2DENGINE_MANAGERS_LOADERS_LOADER_H_

#include <memory>
#include <string>

namespace simple2dengine
{
    struct BaseAsset
```

(continues on next page)

(continued from previous page)

```
    {
    };
    template<typename T> struct Asset : public BaseAsset
    {
        const T* asset;
    };
    class Loader
    {
      public:
        virtual ~Loader() = 0;
        virtual void load(const std::string& filename) = 0;
        virtual void unload(const std::string& filename) = 0;
        virtual BaseAsset* getAsset(const std::string& filename) const = 0;
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_MANAGERS_LOADERS_LOADER_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-21 Copyright (c) 2019

Includes

- `memory`
- `string`

Included By

- *File asset_manager.h*
- *File font_loader.h*
- *File sound_loader.h*
- *File texture_loader.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Template Struct Asset*
- *Struct BaseAsset*
- *Class Loader*

File music_node.h

Parent directory (simple2dengine/nodes)

Music Node.

Page Contents

Definition (simple2dengine/nodes/music_node.h)

Program Listing for File music_node.h

Return to documentation for file (simple2dengine/nodes/music_node.h)

```
#ifndef _SIMPLE2DENGINE_NODES_MUSIC_NODE_H_
#define _SIMPLE2DENGINE_NODES_MUSIC_NODE_H_

#include <string>

#include "SFML/Audio/Music.hpp"

#include "simple2dengine/engine.h"
#include "simple2dengine/nodes/node.h"

namespace simple2dengine
{
    class MusicNode : public Node, public sf::Music
    {
      public:
        using Node::Node;
        void setMusic(const std::string& filename);
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_NODES_MUSIC_NODE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-27 Copyright (c) 2019

Includes

- SFML/Audio/Music.hpp

- simple2dengine/engine.h (*File engine.h*)
- simple2dengine/nodes/node.h (*File node.h*)
- string

Namespaces

- *Namespace simple2dengine*

Classes

- *Class MusicNode*

File node.h

Parent directory (simple2dengine/nodes)

Base node.

Page Contents

Definition (`simple2dengine/nodes/node.h`)

Program Listing for File node.h

Return to documentation for file (simple2dengine/nodes/node.h)

```
#ifndef _SIMPLE2DENGINE_NODES_NODE_H_
#define _SIMPLE2DENGINE_NODES_NODE_H_

#include <memory>
#include <string>
#include <vector>

#include "SFML/Graphics/Rect.hpp"
#include "SFML/System/Vector2.hpp"
#include "SFML/Window/Event.hpp"

```

(continues on next page)

(continued from previous page)

```
namespace simple2dengine
{
    enum class NodeState : unsigned int
    {
        None,
        Creating,
        Entering,
        Updating,
        Exiting,
        Destroying
    };

    class Engine;
    class Node : public std::enable_shared_from_this<Node>
    {
      public:
        Node(const std::string& nodeName) : name(nodeName){};

        virtual ~Node(){};

        virtual void onCreate(){};
        virtual void onEnter(){};
        virtual void onUpdate(int /*deltaInMs*/){};
        virtual void onInput(sf::Event /*event*/){};
        virtual void onExit(){};
        virtual void onDestroy(){};
        bool addChild(std::shared_ptr<Node> child);
        bool removeChild(const std::string& childName);
        void clear();
        int getIndex() const;
        const std::string& getName() const;

        std::shared_ptr<Node> getParent() const;
        std::shared_ptr<Node> getRoot();
        const std::vector<std::shared_ptr<Node>>& getChildren() const;
        std::shared_ptr<Node> getNode(const std::string& path);

      protected:
        virtual void update(int deltaInMs);
        virtual void render();

        Engine* engine = nullptr; // engine pointer

      private:
        void notifyCreate();
        void notifyEnter();
        void notifyInput(sf::Event event);
        void notifyExit();
        void notifyDestroy();

      private:
        std::vector<std::shared_ptr<Node>> children; // all child nodes
        std::weak_ptr<Node> parent;                  // parent node

        std::string name; // name of node
        int index = 0;    // index of node in its parent
```

(continues on next page)

(continued from previous page)

```
        NodeState state = NodeState::None; // current state of node

        friend class SceneManager;
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_NODES_NODE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-17 Copyright (c) 2019

Includes

- `SFML/Graphics/Rect.hpp`
- `SFML/System/Vector2.hpp`
- `SFML/Window/Event.hpp`
- `memory`
- `string`
- `vector`

Included By

- *File input_manager.h*
- *File scene_manager.h*
- *File canvas_node.h*
- *File music_node.h*
- *File sound_node.h*
- *File timer_node.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class Node*

Enums

- *Enum NodeState*

File scene_manager.h

Parent directory (simple2dengine/managers)

Scene Manager.

Page Contents

Definition (simple2dengine/managers/scene_manager.h)

Program Listing for File scene_manager.h

Return to documentation for file (simple2dengine/managers/scene_manager.h)

```
#ifndef _SIMPLE2DENGINE_MANAGERS_SCENE_MANAGER_H_
#define _SIMPLE2DENGINE_MANAGERS_SCENE_MANAGER_H_

#include <memory>
#include <string>
#include <unordered_map>
#include <vector>

#include "simple2dengine/nodes/node.h"

namespace simple2dengine
{
    class SceneManager
    {
      public:
        SceneManager() : currentScene(nullptr){};
        void addScene(std::shared_ptr<Node> scene);
        void removeSceneImmediately(const std::string& name);
        void removeScene(const std::string& name);
        void activateScene(const std::string& name);
        int getSceneCount() const;
        void clear();
        void update(int deltaInMs);
        void render();
        void input(sf::Event event);

      private:
        std::shared_ptr<Node> currentScene = nullptr;                 // current␣
↪scene
        std::unordered_map<std::string, std::shared_ptr<Node>> scenes; // all scenes␣
↪(nodes)
```

(continues on next page)

(continued from previous page)

```
        std::vector<std::shared_ptr<Node>> deletionQueue; // nodes that will be␣
↪destroyed on next tick

        Engine* engine = nullptr; // engine pointer

        friend class Engine;
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_MANAGERS_SCENE_MANAGER_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-17 Copyright (c) 2019

Includes

- `memory`
- `simple2dengine/nodes/node.h` (*File node.h*)
- `string`
- `unordered_map`
- `vector`

Included By

- *File engine.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class SceneManager*

File sound_loader.h

Parent directory (`simple2dengine/managers/loaders`)

Sound Loader.

Page Contents

Definition (`simple2dengine/managers/loaders/sound_loader.h`)

Program Listing for File sound_loader.h

Return to documentation for file (`simple2dengine/managers/loaders/sound_loader.h`)

```
#ifndef _SIMPLE2DENGINE_MANAGERS_LOADERS_SOUND_LOADER_H_
#define _SIMPLE2DENGINE_MANAGERS_LOADERS_SOUND_LOADER_H_

#include <memory>
#include <string>
#include <unordered_map>

#include "simple2dengine/managers/loaders/loader.h"

#include "SFML/Audio/SoundBuffer.hpp"

namespace simple2dengine
{
    class SoundLoader : public Loader
    {
      public:
        virtual void load(const std::string& filename) final;
        virtual void unload(const std::string& filename) final;
        virtual BaseAsset* getAsset(const std::string& filename) const final;

      private:
        std::unordered_map<std::string, sf::SoundBuffer> buffers; // loaded sound␣
↪buffers
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_MANAGERS_LOADERS_SOUND_LOADER_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-22 Copyright (c) 2019

Includes

- `SFML/Audio/SoundBuffer.hpp`
- `memory`
- `simple2dengine/managers/loaders/loader.h` (*File loader.h*)

- string
- unordered_map

Included By

- *File engine.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class SoundLoader*

File sound_node.h

Parent directory (simple2dengine/nodes)

Sound Node.

Page Contents

Definition (simple2dengine/nodes/sound_node.h)

Program Listing for File sound_node.h

Return to documentation for file (simple2dengine/nodes/sound_node.h)

```
#ifndef _SIMPLE2DENGINE_NODES_SOUND_NODE_H_
#define _SIMPLE2DENGINE_NODES_SOUND_NODE_H_

#include <string>

#include "SFML/Audio/Sound.hpp"

#include "simple2dengine/engine.h"
#include "simple2dengine/nodes/node.h"

namespace simple2dengine
```

(continues on next page)

(continued from previous page)

```
{
    class SoundNode : public Node, public sf::Sound
    {
      public:
        using Node::Node;
        void setSound(const AssetManager& assetManager, const std::string& filename);
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_NODES_SOUND_NODE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-22 Copyright (c) 2019

Includes

- `SFML/Audio/Sound.hpp`
- `simple2dengine/engine.h` (*File engine.h*)
- `simple2dengine/nodes/node.h` (*File node.h*)
- `string`

Namespaces

- *Namespace simple2dengine*

Classes

- *Class SoundNode*

File sprite_node.h

Parent directory (`simple2dengine/nodes/canvas`)

Sprite Node.

Page Contents

Definition (`simple2dengine/nodes/canvas/sprite_node.h`)

Program Listing for File sprite_node.h

Return to documentation for file (`simple2dengine/nodes/canvas/sprite_node.h`)

```
#ifndef _SIMPLE2DENGINE_NODES_CANVAS_SPRITE_NODE_H_
#define _SIMPLE2DENGINE_NODES_CANVAS_SPRITE_NODE_H_

#include <string>

#include "SFML/Graphics/Sprite.hpp"

#include "simple2dengine/engine.h"
#include "simple2dengine/nodes/canvas/canvas_node.h"

namespace simple2dengine
{
    class SpriteNode : public CanvasNode, public sf::Sprite
    {
      public:
        using CanvasNode::CanvasNode;
        void setImage(const AssetManager& assetManager, const std::string& filename);
        virtual void updateTransform() override;

        using CanvasNode::getPosition;
        using CanvasNode::move;
        using CanvasNode::setPosition;

      protected:
        virtual void render() override;
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_NODES_CANVAS_SPRITE_NODE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-19 Copyright (c) 2019

Includes

- `SFML/Graphics/Sprite.hpp`
- `simple2dengine/engine.h` (*File engine.h*)
- `simple2dengine/nodes/canvas/canvas_node.h` (*File canvas_node.h*)
- `string`

Namespaces

- *Namespace simple2dengine*

Classes

- *Class SpriteNode*

File text_node.h

Parent directory (`simple2dengine/nodes/canvas`)

Text Node.

Page Contents

Definition (`simple2dengine/nodes/canvas/text_node.h`)

Program Listing for File text_node.h

Return to documentation for file (`simple2dengine/nodes/canvas/text_node.h`)

```
#ifndef _SIMPLE2DENGINE_NODES_CANVAS_TEXT_NODE_H_
#define _SIMPLE2DENGINE_NODES_CANVAS_TEXT_NODE_H_

#include <string>

#include "SFML/Graphics/Text.hpp"

#include "simple2dengine/engine.h"
#include "simple2dengine/nodes/canvas/canvas_node.h"

namespace simple2dengine
{
    class TextNode : public CanvasNode, public sf::Text
    {
      public:
        using CanvasNode::CanvasNode;
        void setFont(const AssetManager& assetManager, const std::string& filename);
        void setString(const std::string& textString);
        void setCharacterSize(unsigned int size);
        virtual void updateTransform() override;

        using CanvasNode::getPosition;
        using CanvasNode::move;
        using CanvasNode::setPosition;

      protected:
```

(continues on next page)

(continued from previous page)

```
        virtual void render() override;
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_NODES_CANVAS_TEXT_NODE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-24 Copyright (c) 2019

Includes

- `SFML/Graphics/Text.hpp`
- `simple2dengine/engine.h` (*File engine.h*)
- `simple2dengine/nodes/canvas/canvas_node.h` (*File canvas_node.h*)
- `string`

Namespaces

- *Namespace simple2dengine*

Classes

- *Class TextNode*

File texture_loader.h

Parent directory (`simple2dengine/managers/loaders`)

Texture Loader.

Page Contents

Definition (`simple2dengine/managers/loaders/texture_loader.h`)

Program Listing for File texture_loader.h

Return to documentation for file (`simple2dengine/managers/loaders/texture_loader.h`)

```
#ifndef _SIMPLE2DENGINE_MANAGERS_LOADERS_TEXTURE_LOADER_H_
#define _SIMPLE2DENGINE_MANAGERS_LOADERS_TEXTURE_LOADER_H_

#include <memory>
#include <string>
#include <unordered_map>

#include "simple2dengine/managers/loaders/loader.h"

#include "SFML/Graphics/Texture.hpp"

namespace simple2dengine
{
    class TextureLoader : public Loader
    {
      public:
        virtual void load(const std::string& filename) final;
        virtual void unload(const std::string& filename) final;
        virtual BaseAsset* getAsset(const std::string& filename) const final;

      private:
        std::unordered_map<std::string, sf::Texture> textures; // loaded textures
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_MANAGERS_LOADERS_TEXTURE_LOADER_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-21 Copyright (c) 2019

Includes

- `SFML/Graphics/Texture.hpp`
- `memory`
- `simple2dengine/managers/loaders/loader.h` (*File loader.h*)
- `string`
- `unordered_map`

Included By

- *File engine.h*

Namespaces

- *Namespace simple2dengine*

Classes

- *Class TextureLoader*

File timer_node.h

Parent directory (`simple2dengine/nodes`)

Timer Node.

Page Contents

Definition (`simple2dengine/nodes/timer_node.h`)

Program Listing for File timer_node.h

Return to documentation for file (`simple2dengine/nodes/timer_node.h`)

```
#ifndef _SIMPLE2DENGINE_NODES_TIMER_NODE_H_
#define _SIMPLE2DENGINE_NODES_TIMER_NODE_H_

#include <functional>
#include <string>

#include "simple2dengine/engine.h"
#include "simple2dengine/nodes/node.h"

namespace simple2dengine
{
    class TimerNode : public Node
    {
      public:
        TimerNode(const std::string& nodeName, unsigned int time = 0, bool isOneShot␣
↪= true)
            : Node(nodeName), finishTime(time), oneShot(isOneShot){};
        void setTime(unsigned int time);
        void start();
        void pause();
        void reset();
```

(continues on next page)

(continued from previous page)

```
        bool isPaused() const;
        void setOneShot(bool oneShot);
        bool isOneShot() const;
        void onTimeout(std::function<void()> function);

      protected:
        virtual void update(int deltaInMs) override;

      private:
        unsigned int finishTime = 0;  // amount of time need to send finish signal
        unsigned int elapsedTime = 0; // current elapsed time
        bool oneShot = true;          // if true - timer will not restart
        bool paused = false;          // if pause if true - time will not elapse

        std::function<void()> timeoutFunc; // signal on timeout
    };
} // namespace simple2dengine

#endif // _SIMPLE2DENGINE_NODES_TIMER_NODE_H_
```

Detailed Description

Ilya Bardinov (ilya.bardinov@gmail.com) 2019-02-26 Copyright (c) 2019

Includes

- `functional`
- `simple2dengine/engine.h` (*File engine.h*)
- `simple2dengine/nodes/node.h` (*File node.h*)
- `string`

Namespaces

- *Namespace simple2dengine*

Classes

- *Class TimerNode*

Index