

HelloWorld

Generated by Doxygen 1.9.1



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 AssetsManager Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 AssetsManager()	6
3.1.2.2 ~AssetsManager()	7
3.1.3 Member Function Documentation	7
3.1.3.1 __loadSoundBuffer()	7
3.1.3.2 clear()	8
3.1.3.3 getCurrentTrackKey()	9
3.1.3.4 getFont()	9
3.1.3.5 getSound()	10
3.1.3.6 getSoundBuffer()	10
3.1.3.7 getTexture()	11
3.1.3.8 getTrackStatus()	11
3.1.3.9 loadFont()	12
3.1.3.10 loadSound()	12
3.1.3.11 loadTexture()	13
3.1.3.12 loadTrack()	14
3.1.3.13 nextTrack()	15
3.1.3.14 pauseTrack()	15
3.1.3.15 playTrack()	15
3.1.3.16 previousTrack()	15
3.1.3.17 stopTrack()	16
3.1.4 Member Data Documentation	16
3.1.4.1 current_track	16
3.1.4.2 font_map	16
3.1.4.3 sound_map	16
3.1.4.4 soundbuffer_map	16
3.1.4.5 texture_map	17
3.1.4.6 track_map	17
3.2 HexMap Class Reference	17
3.2.1 Detailed Description	18
3.2.2 Constructor & Destructor Documentation	18
3.2.2.1 HexMap()	18
3.2.2.2 ~HexMap()	19
3.2.3 Member Function Documentation	19

3.2.3.1 __assembleHexMap()	19
3.2.3.2 clear()	21
3.2.3.3 draw()	21
3.2.4 Member Data Documentation	22
3.2.4.1 assets_manager_ptr	22
3.2.4.2 frame	22
3.2.4.3 hex_map	22
3.2.4.4 inputs_handler_ptr	22
3.2.4.5 messages_handler_ptr	23
3.2.4.6 n_layers	23
3.2.4.7 position_x	23
3.2.4.8 position_y	23
3.3 HexTile Class Reference	23
3.3.1 Detailed Description	25
3.3.2 Constructor & Destructor Documentation	25
3.3.2.1 HexTile()	25
3.3.2.2 ~HexTile()	26
3.3.3 Member Function Documentation	26
3.3.3.1 __setUpNodeSprite()	26
3.3.3.2 __setUpTileSprite()	26
3.3.3.3 draw()	27
3.3.3.4 setTileType()	27
3.3.4 Member Data Documentation	28
3.3.4.1 assets_manager_ptr	28
3.3.4.2 frame	28
3.3.4.3 inputs_handler_ptr	28
3.3.4.4 major_radius	28
3.3.4.5 messages_handler_ptr	28
3.3.4.6 minor_radius	29
3.3.4.7 node_sprite	29
3.3.4.8 position_x	29
3.3.4.9 position_y	29
3.3.4.10 show_node	29
3.3.4.11 tile_sprite	29
3.3.4.12 tile_type	30
3.4 InputsHandler Class Reference	30
3.4.1 Detailed Description	30
3.4.2 Constructor & Destructor Documentation	31
3.4.2.1 InputsHandler()	31
3.4.2.2 ~InputsHandler()	31
3.4.3 Member Function Documentation	31
3.4.3.1 __constructKeyCodeMap()	31

3.4.3.2 printKeysPressed()	35
3.4.3.3 process()	36
3.4.3.4 reset()	36
3.4.4 Member Data Documentation	36
3.4.4.1 key_code_map	36
3.4.4.2 key_press_vec	37
3.4.4.3 key_pressed_once_vec	37
3.5 MessagesHandler Class Reference	37
3.5.1 Detailed Description	37
3.5.2 Constructor & Destructor Documentation	37
3.5.2.1 MessagesHandler()	38
3.5.2.2 ~MessagesHandler()	38
<b>4 File Documentation</b>	<b>39</b>
4.1 header/ESC_core/AssetsManager.h File Reference	39
4.1.1 Detailed Description	39
4.2 header/ESC_core/constants.h File Reference	40
4.2.1 Detailed Description	40
4.2.2 Variable Documentation	40
4.2.2.1 FRAMES_PER_SECOND	40
4.2.2.2 SECONDS_PER_FRAME	40
4.3 header/ESC_core/doxygen_cite.h File Reference	41
4.3.1 Detailed Description	41
4.4 header/ESC_core/includes.h File Reference	41
4.4.1 Detailed Description	42
4.5 header/ESC_core/InputsHandler.h File Reference	42
4.5.1 Detailed Description	42
4.6 header/ESC_core/MessagesHandler.h File Reference	43
4.6.1 Detailed Description	43
4.7 header/ESC_core/testing_utils.h File Reference	43
4.7.1 Detailed Description	44
4.7.2 Function Documentation	44
4.7.2.1 expectedErrorNotDetected()	44
4.7.2.2 printGold()	45
4.7.2.3 printGreen()	45
4.7.2.4 printRed()	45
4.7.2.5 testFloatEquals()	46
4.7.2.6 testGreaterThan()	46
4.7.2.7 testGreaterThanOrEqualTo()	47
4.7.2.8 testLessThan()	48
4.7.2.9 testLessThanOrEqualTo()	48
4.7.2.10 testTruth()	49

4.7.3 Variable Documentation . . . . .	50
4.7.3.1 FLOAT_TOLERANCE . . . . .	50
4.8 header/HexMap/HexMap.h File Reference . . . . .	50
4.8.1 Detailed Description . . . . .	50
4.9 header/HexMap/HexTile.h File Reference . . . . .	51
4.9.1 Detailed Description . . . . .	52
4.9.2 Enumeration Type Documentation . . . . .	52
4.9.2.1 TileType . . . . .	52
4.9.3 Function Documentation . . . . .	52
4.9.3.1 FOREST_GREEN() . . . . .	52
4.9.3.2 LAKE_BLUE() . . . . .	53
4.9.3.3 MOUNTAINS_GREY() . . . . .	53
4.9.3.4 OCEAN_BLUE() . . . . .	53
4.9.3.5 PLAINS_YELLOW() . . . . .	53
4.10 source/ESC_core/AssetsManager.cpp File Reference . . . . .	53
4.10.1 Detailed Description . . . . .	54
4.11 source/ESC_core/InputsHandler.cpp File Reference . . . . .	54
4.11.1 Detailed Description . . . . .	54
4.12 source/ESC_core/MessagesHandler.cpp File Reference . . . . .	54
4.12.1 Detailed Description . . . . .	54
4.13 source/ESC_core/testing_utils.cpp File Reference . . . . .	55
4.13.1 Detailed Description . . . . .	55
4.13.2 Function Documentation . . . . .	55
4.13.2.1 expectedErrorNotDetected() . . . . .	55
4.13.2.2 printGold() . . . . .	56
4.13.2.3 printGreen() . . . . .	56
4.13.2.4 printRed() . . . . .	56
4.13.2.5 testFloatEquals() . . . . .	57
4.13.2.6 testGreaterThan() . . . . .	57
4.13.2.7 testGreaterThanOrEqualTo() . . . . .	58
4.13.2.8 testLessThan() . . . . .	59
4.13.2.9 testLessThanOrEqualTo() . . . . .	59
4.13.2.10 testTruth() . . . . .	60
4.14 source/HexMap/HexMap.cpp File Reference . . . . .	61
4.14.1 Detailed Description . . . . .	61
4.15 source/HexMap/HexTile.cpp File Reference . . . . .	61
4.15.1 Detailed Description . . . . .	61
4.16 test/ESC_core/test_AssetsManager.cpp File Reference . . . . .	61
4.16.1 Detailed Description . . . . .	62
4.16.2 Function Documentation . . . . .	62
4.16.2.1 main() . . . . .	62
4.17 test/ESC_core/test_InputsHandler.cpp File Reference . . . . .	64

---

4.17.1 Detailed Description . . . . .	64
4.17.2 Function Documentation . . . . .	65
4.17.2.1 main() . . . . .	65
4.18 test/ESC_core/test_MessagesHandler.cpp File Reference . . . . .	66
4.18.1 Detailed Description . . . . .	67
4.18.2 Function Documentation . . . . .	67
4.18.2.1 main() . . . . .	67
4.19 test/HexMap/test_HexMap.cpp File Reference . . . . .	68
4.19.1 Detailed Description . . . . .	68
4.19.2 Function Documentation . . . . .	69
4.19.2.1 main() . . . . .	69
<b>Bibliography</b>	<b>71</b>
<b>Index</b>	<b>73</b>





# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">AssetsManager</a>	A class which manages visual and sound assets . . . . .	5
<a href="#">HexMap</a>	A class which defines a hex map of hex tiles . . . . .	17
<a href="#">HexTile</a>	A class which defines a hex tile of the hex map . . . . .	23
<a href="#">InputsHandler</a>	A class which handles inputs from peripherals (i.e., keyboard and mouse) . . . . .	30
<a href="#">MessagesHandler</a>	A class which handles message traffic between game objects . . . . .	37



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

header/ESC_core/ <a href="#">AssetsManager.h</a>	
Header file for the <a href="#">AssetsManager</a> class . . . . .	39
header/ESC_core/ <a href="#">constants.h</a>	
Header file for various constants . . . . .	40
header/ESC_core/ <a href="#">doxygen_cite.h</a>	
Header file which simply cites the doxygen tool . . . . .	41
header/ESC_core/ <a href="#">includes.h</a>	
Header file for various includes . . . . .	41
header/ESC_core/ <a href="#">InputsHandler.h</a>	
Header file for the <a href="#">InputsHandler</a> class . . . . .	42
header/ESC_core/ <a href="#">MessagesHandler.h</a>	
Header file for the <a href="#">MessagesHandler</a> class . . . . .	43
header/ESC_core/ <a href="#">testing_utils.h</a>	
Header file for various testing utilities . . . . .	43
header/HexMap/ <a href="#">HexMap.h</a>	
Header file for the <a href="#">HexMap</a> class . . . . .	50
header/HexMap/ <a href="#">HexTile.h</a>	
Header file for the <a href="#">HexTile</a> class . . . . .	51
source/ESC_core/ <a href="#">AssetsManager.cpp</a>	
Implementation file for the <a href="#">AssetsManager</a> class . . . . .	53
source/ESC_core/ <a href="#">InputsHandler.cpp</a>	
Implementation file for the <a href="#">InputsHandler</a> class . . . . .	54
source/ESC_core/ <a href="#">MessagesHandler.cpp</a>	
Implementation file for the <a href="#">MessagesHandler</a> class . . . . .	54
source/ESC_core/ <a href="#">testing_utils.cpp</a>	
Implementation file for various testing utilities . . . . .	55
source/HexMap/ <a href="#">HexMap.cpp</a>	
Implementation file for the <a href="#">HexMap</a> class . . . . .	61
source/HexMap/ <a href="#">HexTile.cpp</a>	
Implementation file for the <a href="#">HexTile</a> class . . . . .	61
test/ESC_core/ <a href="#">test_AssetsManager.cpp</a>	
Suite of tests for the <a href="#">AssetsManager</a> class . . . . .	61
test/ESC_core/ <a href="#">test_InputsHandler.cpp</a>	
Suite of tests for the <a href="#">InputsHandler</a> class . . . . .	64
test/ESC_core/ <a href="#">test_MessagesHandler.cpp</a>	
Suite of tests for the <a href="#">MessagesHandler</a> class . . . . .	66
test/HexMap/ <a href="#">test_HexMap.cpp</a>	
Suite of tests for the <a href="#">HexMap</a> class . . . . .	68



## Chapter 3

# Class Documentation

### 3.1 AssetsManager Class Reference

A class which manages visual and sound assets.

```
#include <AssetsManager.h>
```

#### Public Member Functions

- [AssetsManager](#) (void)  
*Constructor for the [AssetsManager](#) class.*
- void [loadFont](#) (std::string, std::string)  
*Method to load a font and insert it into the font map.*
- void [loadTexture](#) (std::string, std::string)  
*Method to load a texture and insert it into the texture map.*
- void [loadSound](#) (std::string, std::string)  
*Method to load a sound and insert it into the sound map. Automatically creates a corresponding sf::SoundBuffer.*
- void [loadTrack](#) (std::string, std::string)  
*Method to load a track (sf::Music) and insert it into the track map.*
- sf::Font \* [getFont](#) (std::string)  
*Method to get font associated with given font key.*
- sf::Texture \* [getTexture](#) (std::string)  
*Method to get texture associated with given texture key.*
- sf::SoundBuffer \* [getSoundBuffer](#) (std::string)  
*Method to get soundbuffer associated with given sound key.*
- sf::Sound \* [getSound](#) (std::string)  
*Method to get sound associated with given sound key.*
- void [playTrack](#) (void)  
*Method to play the current track.*
- void [pauseTrack](#) (void)  
*Method to pause the current track.*
- void [stopTrack](#) (void)  
*Method to stop the current track.*
- void [nextTrack](#) (void)  
*Method to advance to the next track. Wraps around if the end of the track map is reached.*

- void [previousTrack](#) (void)  
*Method to return to the previous track. Wraps around if the beginning of the track map is reached.*
- std::string [getCurrentTrackKey](#) (void)  
*Method to get track key for current track.*
- sf::SoundSource::Status [getTrackStatus](#) (void)  
*Method to get the status of the current track.*
- void [clear](#) (void)  
*Method to clear all loaded assets.*
- [~AssetsManager](#) (void)  
*Destructor for the [AssetsManager](#) class.*

## Public Attributes

- std::map< std::string, sf::Font \* > [font\\_map](#)  
*A map of pointers to loaded fonts.*
- std::map< std::string, sf::Texture \* > [texture\\_map](#)  
*A map of pointers to loaded textures.*
- std::map< std::string, sf::SoundBuffer \* > [soundbuffer\\_map](#)  
*A map of pointers to sound buffers.*
- std::map< std::string, sf::Sound \* > [sound\\_map](#)  
*A map of pointers to loaded sounds.*
- std::map< std::string, sf::Music \* >::iterator [current\\_track](#)  
*A map iterator which corresponds to the current track (i.e., the track currently being played).*
- std::map< std::string, sf::Music \* > [track\\_map](#)  
*A map of pointers to opened tracks (i.e. sf::Music).*

## Private Member Functions

- void [\\_\\_loadSoundBuffer](#) (std::string, std::string)  
*Helper method to load a soundbuffer and insert it into the soundbuffer map. Should only be called by [loadSound\(\)](#), to create an sf::SoundBuffer corresponding to the loaded sf::Sound.*

### 3.1.1 Detailed Description

A class which manages visual and sound assets.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 AssetsManager()

```
AssetsManager::AssetsManager (
    void )
```

Constructor for the [AssetsManager](#) class.

```
110 {
111     //...
112
113     std::cout << "AssetsManager constructed at " << this << std::endl;
114
115     return;
116 } /* AssetsManager() */
```

### 3.1.2.2 ~AssetsManager()

```
AssetsManager::~AssetsManager (
    void )
```

Destructor for the [AssetsManager](#) class.

```
739 {
740     this->clear();
741
742     std::cout << "AssetsManager at " << this << " destroyed" << std::endl;
743
744     return;
745 } /* ~AssetsManager() */
```

## 3.1.3 Member Function Documentation

### 3.1.3.1 \_\_loadSoundBuffer()

```
void AssetsManager::__loadSoundBuffer (
    std::string path_2_sound,
    std::string sound_key ) [private]
```

Helper method to load a soundbuffer and insert it into the soundbuffer map. Should only be called by [loadSound\(\)](#), to create an `sf::SoundBuffer` corresponding to the loaded `sf::Sound`.

#### Parameters

<i>path_2_sound</i>	A path (either relative or absolute) to the sound file.
<i>sound_key</i>	A key associated with the sound (for indexing into the soundbuffer map).

```
47 {
48     // 1. check key, throw error if already in use
49     if (this->soundbuffer_map.count(sound_key) > 0) {
50         std::string error_str = "ERROR AssetsManager::__loadSoundBuffer() sound key ";
51         error_str += sound_key;
52         error_str += " is already in use";
53
54         this->clear();
55
56         #ifdef _WIN32
57             std::cout << error_str << std::endl;
58         #endif /* _WIN32 */
59
60         throw std::runtime_error(error_str);
61     }
62
63
64     // 2. load from file, throw error on fail
65     sf::SoundBuffer* soundbuffer_ptr = new sf::SoundBuffer();
66
67     if (not soundbuffer_ptr->loadFromFile(path_2_sound)) {
68         std::string error_str = "ERROR AssetsManager::__loadSoundBuffer() could not load ";
69         error_str += "soundbuffer at ";
70         error_str += path_2_sound;
71
72         this->clear();
73
74         #ifdef _WIN32
75             std::cout << error_str << std::endl;
76         #endif /* _WIN32 */
77
78         throw std::runtime_error(error_str);
79     }
80
81 }
```

```

82 // 3. insert into soundbuffer map
83 this->soundbuffer_map.insert(
84     std::pair<std::string, sf::SoundBuffer*>(sound_key, soundbuffer_ptr)
85 );
86
87 std::cout << "SoundBuffer " << sound_key << " inserted into soundbuffer map" <<
88     std::endl;
89
90 return;
91 } /* __loadSoundBuffer() */

```

### 3.1.3.2 clear()

```

void AssetsManager::clear (
    void )

```

Method to clear all loaded assets.

```

646 {
647     // 1. clear fonts
648     std::map<std::string, sf::Font*>::iterator font_iter;
649     for (
650         font_iter = this->font_map.begin();
651         font_iter != this->font_map.end();
652         font_iter++
653     ) {
654         delete font_iter->second;
655
656         std::cout << "Font " << font_iter->first << " deleted from font map" <<
657             std::endl;
658     }
659     this->font_map.clear();
660
661     // 2. clear textures
662     std::map<std::string, sf::Texture*>::iterator texture_iter;
663     for (
664         texture_iter = this->texture_map.begin();
665         texture_iter != this->texture_map.end();
666         texture_iter++
667     ) {
668         delete texture_iter->second;
669
670         std::cout << "Texture " << texture_iter->first << " deleted from texture map" <<
671             std::endl;
672     }
673     this->texture_map.clear();
674
675     // 3. clear sound buffers
676     std::map<std::string, sf::SoundBuffer*>::iterator soundbuffer_iter;
677     for (
678         soundbuffer_iter = this->soundbuffer_map.begin();
679         soundbuffer_iter != this->soundbuffer_map.end();
680         soundbuffer_iter++
681     ) {
682         delete soundbuffer_iter->second;
683
684         std::cout << "SoundBuffer " << soundbuffer_iter->first <<
685             " deleted from soundbuffer map" << std::endl;
686     }
687     this->soundbuffer_map.clear();
688
689     // 4. clear sounds
690     std::map<std::string, sf::Sound*>::iterator sound_iter;
691     for (
692         sound_iter = this->sound_map.begin();
693         sound_iter != this->sound_map.end();
694         sound_iter++
695     ) {
696         sound_iter->second->stop();
697         delete sound_iter->second;
698
699         std::cout << "Sound " << sound_iter->first << " deleted from sound map" <<
700             std::endl;
701     }
702     this->sound_map.clear();
703
704 }

```



```

707
708 // 5. clear tracks
709 std::map<std::string, sf::Music*>::iterator track_iter;
710 for (
711     track_iter = this->track_map.begin();
712     track_iter != this->track_map.end();
713     track_iter++)
714 {
715     track_iter->second->stop();
716     delete track_iter->second;
717
718     std::cout << "Track " << track_iter->first << " deleted from track map" <<
719         std::endl;
720 }
721 this->track_map.clear();
722
723 return;
724 } /* clear() */

```

### 3.1.3.3 getCurrentTrackKey()

```

std::string AssetsManager::getCurrentTrackKey (
    void )

```

Method to get track key for current track.

#### Returns

The track key for the current track.

```

610 {
611     return this->current_track->first;
612 } /* getCurrentTrackKey() */

```

### 3.1.3.4 getFont()

```

sf::Font * AssetsManager::getFont (
    std::string font_key )

```

Method to get font associated with given font key.

#### Parameters

<i>font_key</i>	A key associated with the font (for indexing into the font map).
-----------------	--

#### Returns

A pointer to the corresponding font.

```

351 {
352     // 1. check key, throw error if not found
353     if (this->font_map.count(font_key) <= 0) {
354         std::string error_str = "ERROR AssetsManager::getFont() font key ";
355         error_str += font_key;
356         error_str += " is not contained in font map";
357
358         this->clear();
359
360         #ifdef _WIN32

```

```

361         std::cout << error_str << std::endl;
362     #endif /* _WIN32 */
363
364     throw std::runtime_error(error_str);
365 }
366
367 return this->font_map[font_key];
368 } /* getFont() */

```

### 3.1.3.5 getSound()

```

sf::Sound * AssetsManager::getSound (
    std::string sound_key )

```

Method to get sound associated with given sound key.

#### Parameters

<i>sound_key</i>	A key associated with the sound (for indexing into the sound map).
------------------	--

#### Returns

A pointer to the corresponding sound.

```

461 {
462     // 1. check key, throw error if not found
463     if (this->sound_map.count(sound_key) <= 0) {
464         std::string error_str = "ERROR AssetsManager::getSound() sound key ";
465         error_str += sound_key;
466         error_str += " is not contained in sound map";
467
468         this->clear();
469
470         #ifdef _WIN32
471             std::cout << error_str << std::endl;
472         #endif /* _WIN32 */
473
474         throw std::runtime_error(error_str);
475     }
476
477     return this->sound_map[sound_key];
478 } /* getSound() */

```

### 3.1.3.6 getSoundBuffer()

```

sf::SoundBuffer * AssetsManager::getSoundBuffer (
    std::string sound_key )

```

Method to get soundbuffer associated with given sound key.

#### Parameters

<i>sound_key</i>	A key associated with the soundbuffer (for indexing into the soundbuffer map).
------------------	--

**Returns**

A pointer to the corresponding soundbuffer.

```

425 {
426     // 1. check key, throw error if not found
427     if (this->soundbuffer_map.count(sound_key) <= 0) {
428         std::string error_str = "ERROR AssetsManager::getSoundBuffer() sound key ";
429         error_str += sound_key;
430         error_str += " is not contained in soundbuffer map";
431
432         this->clear();
433
434         #ifdef _WIN32
435             std::cout << error_str << std::endl;
436         #endif /* _WIN32 */
437
438         throw std::runtime_error(error_str);
439     }
440
441     return this->soundbuffer_map[sound_key];
442 } /* getSoundBuffer() */

```

**3.1.3.7 getTexture()**

```

sf::Texture * AssetsManager::getTexture (
    std::string texture_key )

```

Method to get texture associated with given texture key.

**Parameters**

<i>texture_key</i>	A key associated with the texture (for indexing into the texture map).
--------------------	--

**Returns**

A pointer to the corresponding texture.

```

388 {
389     // 1. check key, throw error if not found
390     if (this->texture_map.count(texture_key) <= 0) {
391         std::string error_str = "ERROR AssetsManager::getTexture() texture key ";
392         error_str += texture_key;
393         error_str += " is not contained in texture map";
394
395         this->clear();
396
397         #ifdef _WIN32
398             std::cout << error_str << std::endl;
399         #endif /* _WIN32 */
400
401         throw std::runtime_error(error_str);
402     }
403
404     return this->texture_map[texture_key];
405 } /* getTexture() */

```

**3.1.3.8 getTrackStatus()**

```

sf::SoundSource::Status AssetsManager::getTrackStatus (
    void )

```

Method to get the status of the current track.

**Returns**

The status of the current track.

```
629 {
630     return this->current_track->second->getStatus();
631 } /* getTrackStatus */
```

**3.1.3.9 loadFont()**

```
void AssetsManager::loadFont (
    std::string path_2_font,
    std::string font_key )
```

Method to load a font and insert it into the font map.

**Parameters**

<i>path_2_font</i>	A path (either relative or absolute) to the font file.
<i>font_key</i>	A key associated with the font (for indexing into the font map).

```
135 {
136     // 1. check key, throw error if already in use
137     if (this->font_map.count(font_key) > 0) {
138         std::string error_str = "ERROR AssetsManager::loadFont() font key ";
139         error_str += font_key;
140         error_str += " is already in use";
141
142         this->clear();
143
144         #ifdef _WIN32
145             std::cout << error_str << std::endl;
146         #endif /* _WIN32 */
147
148         throw std::runtime_error(error_str);
149     }
150
151     // 2. load from file, throw error on fail
152     sf::Font* font_ptr = new sf::Font();
153
154     if (not font_ptr->loadFromFile(path_2_font)) {
155         std::string error_str = "ERROR AssetsManager::loadFont() could not load ";
156         error_str += "font at ";
157         error_str += path_2_font;
158
159         this->clear();
160
161         #ifdef _WIN32
162             std::cout << error_str << std::endl;
163         #endif /* _WIN32 */
164
165         throw std::runtime_error(error_str);
166     }
167
168     // 3. insert into font map
169     this->font_map.insert(std::pair<std::string, sf::Font*>(font_key, font_ptr));
170
171     std::cout << "Font " << font_key << " inserted into font map" << std::endl;
172
173     return;
174 } /* loadFont() */
```

**3.1.3.10 loadSound()**

```
void AssetsManager::loadSound (
```

```
std::string path_2_sound,
std::string sound_key )
```

Method to load a sound and insert it into the sound map. Automatically creates a corresponding sf::SoundBuffer.

#### Parameters

<i>path_2_sound</i>	A path (either relative or absolute) to the sound file.
<i>sound_key</i>	A key associated with the sound (for indexing into the sound map).

```
259 {
260     // 1. create an associated sf::SoundBuffer
261     this->__loadSoundBuffer(path_2_sound, sound_key);
262
263     // 2. associate sf::Sound with sf::SoundBuffer
264     sf::Sound* sound_ptr = new sf::Sound();
265     sound_ptr->setBuffer(*(this->soundbuffer_map[sound_key]));
266
267     // 3. insert into sound map
268     this->sound_map.insert(std::pair<std::string, sf::Sound*>(sound_key, sound_ptr));
269
270     std::cout << "Sound " << sound_key << " inserted into sound map" << std::endl;
271
272     return;
273 } /* loadSound() */
```

#### 3.1.3.11 loadTexture()

```
void AssetsManager::loadTexture (
    std::string path_2_texture,
    std::string texture_key )
```

Method to load a texture and insert it into the texture map.

#### Parameters

<i>path_2_texture</i>	A path (either relative or absolute) to the texture file.
<i>texture_key</i>	A key associated with the texture (for indexing into the texture map).

```
196 {
197     // 1. check key, throw error if already in use
198     if (this->texture_map.count(texture_key) > 0) {
199         std::string error_str = "ERROR AssetsManager::loadTexture() texture key ";
200         error_str += texture_key;
201         error_str += " is already in use";
202
203         this->clear();
204
205         #ifdef _WIN32
206             std::cout << error_str << std::endl;
207         #endif /* _WIN32 */
208
209         throw std::runtime_error(error_str);
210     }
211
212     // 2. load from file, throw error on fail
213     sf::Texture* texture_ptr = new sf::Texture();
214
215     if (not texture_ptr->loadFromFile(path_2_texture)) {
216         std::string error_str = "ERROR AssetsManager::loadTexture() could not load ";
217         error_str += "texture at ";
218         error_str += path_2_texture;
219
220         this->clear();
221
222         #ifdef _WIN32
223             std::cout << error_str << std::endl;
224         #endif
```

```

225         #endif /* _WIN32 */
226
227         throw std::runtime_error(error_str);
228     }
229
230
231     // 3. insert into texture map
232     this->texture_map.insert(
233         std::pair<std::string, sf::Texture*>(texture_key, texture_ptr)
234     );
235
236     std::cout << "Texture " << texture_key << " inserted into texture map" << std::endl;
237
238     return;
239 } /* loadTexture() */

```

### 3.1.3.12 loadTrack()

```

void AssetsManager::loadTrack (
    std::string path_2_track,
    std::string track_key )

```

Method to load a track (sf::Music) and insert it into the track map.

#### Parameters

<i>path_2_track</i>	A path (either relative or absolute) to the track file.
<i>track_key</i>	A key associated with the track (for indexing into the track map).

```

292 {
293     // 1. check key, throw error if already in use
294     if (this->track_map.count(track_key) > 0) {
295         std::string error_str = "ERROR AssetsManager::loadTrack() track key ";
296         error_str += track_key;
297         error_str += " is already in use";
298
299         this->clear();
300
301         #ifdef _WIN32
302             std::cout << error_str << std::endl;
303         #endif /* _WIN32 */
304
305         throw std::runtime_error(error_str);
306     }
307
308     // 2. open from file, throw error on fail
309     sf::Music* track_ptr = new sf::Music();
310
311     if (not track_ptr->openFromFile(path_2_track)) {
312         std::string error_str = "ERROR AssetsManager::loadTrack() could not open ";
313         error_str += "track at ";
314         error_str += path_2_track;
315
316         this->clear();
317
318         #ifdef _WIN32
319             std::cout << error_str << std::endl;
320         #endif /* _WIN32 */
321
322         throw std::runtime_error(error_str);
323     }
324
325     // 3. insert into track map
326     this->track_map.insert(std::pair<std::string, sf::Music*>(track_key, track_ptr));
327     this->current_track = this->track_map.begin();
328
329     std::cout << "Track " << track_key << " inserted into track map" << std::endl;
330
331     return;
332 } /* loadTrack() */

```

### 3.1.3.13 nextTrack()

```
void AssetsManager::nextTrack (
    void )
```

Method to advance to the next track. Wraps around if the end of the track map is reached.

```
551 {
552     // 1. stop current track
553     this->stopTrack();
554
555     // 2. increment current track
556     this->current_track++;
557
558     // 3. handle wrap around
559     if (this->current_track == this->track_map.end()) {
560         this->current_track = this->track_map.begin();
561     }
562
563     return;
564 } /* nextTrack() */
```

### 3.1.3.14 pauseTrack()

```
void AssetsManager::pauseTrack (
    void )
```

Method to pause the current track.

```
512 {
513     this->current_track->second->pause();
514
515     return;
516 } /* pauseTrack() */
```

### 3.1.3.15 playTrack()

```
void AssetsManager::playTrack (
    void )
```

Method to play the current track.

```
493 {
494     this->current_track->second->play();
495
496     return;
497 } /* playTrack() */
```

### 3.1.3.16 previousTrack()

```
void AssetsManager::previousTrack (
    void )
```

Method to return to the previous track. Wraps around if the beginning of the track map is reached.

```
580 {
581     // 1. stop current track
582     this->stopTrack();
583
584     // 2. handle wrap around
585     if (this->current_track == this->track_map.begin()) {
586         this->current_track = this->track_map.end();
587     }
588
589     // 3. decrement current track
590     this->current_track--;
591
592     return;
593 } /* previousTrack() */
```

### 3.1.3.17 stopTrack()

```
void AssetsManager::stopTrack (
    void )
```

Method to stop the current track.

```
531 {
532     this->current_track->second->stop();
533
534     return;
535 } /* stopTrack() */
```

## 3.1.4 Member Data Documentation

### 3.1.4.1 current\_track

```
std::map<std::string, sf::Music*>::iterator AssetsManager::current_track
```

A map iterator which corresponds to the current track (i.e., the track currently being played).

### 3.1.4.2 font\_map

```
std::map<std::string, sf::Font*> AssetsManager::font_map
```

A map of pointers to loaded fonts.

### 3.1.4.3 sound\_map

```
std::map<std::string, sf::Sound*> AssetsManager::sound_map
```

A map of pointers to loaded sounds.

### 3.1.4.4 soundbuffer\_map

```
std::map<std::string, sf::SoundBuffer*> AssetsManager::soundbuffer_map
```

A map of pointers to sound buffers.



### 3.1.4.5 texture\_map

```
std::map<std::string, sf::Texture*> AssetsManager::texture_map
```

A map of pointers to loaded textures.

### 3.1.4.6 track\_map

```
std::map<std::string, sf::Music*> AssetsManager::track_map
```

A map of pointers to opened tracks (i.e. sf::Music).

The documentation for this class was generated from the following files:

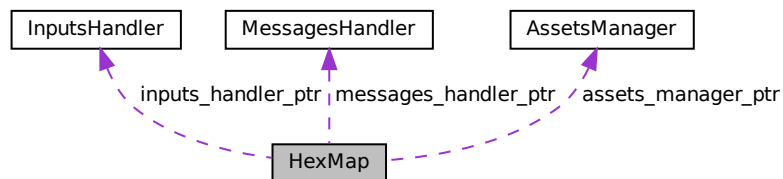
- header/ESC\_core/[AssetsManager.h](#)
- source/ESC\_core/[AssetsManager.cpp](#)

## 3.2 HexMap Class Reference

A class which defines a hex map of hex tiles.

```
#include <HexMap.h>
```

Collaboration diagram for HexMap:



### Public Member Functions

- [HexMap](#) (int, [AssetsManager](#) \*, [InputsHandler](#) \*, [MessagesHandler](#) \*)  
*Constructor for the [HexMap](#) class.*
- void [draw](#) (sf::RenderWindow \*)  
*Method to draw the hex map to the render window. To be called only once per frame!*
- void [clear](#) (void)  
*Method to clear the hex map.*
- [~HexMap](#) (void)  
*Destructor for the [HexMap](#) class.*

## Public Attributes

- int [n\\_layers](#)  
*The number of layers in the hex map.*
- int [frame](#)  
*The current frame of this object.*
- double [position\\_x](#)  
*The x position of the hex map's origin (i.e. central) tile.*
- double [position\\_y](#)  
*The y position of the hex map's origin (i.e. central) tile.*
- `std::map< double, std::map< double, HexTile * > >` [hex\\_map](#)  
*A position-indexed, nested map of hex tiles.*

## Private Member Functions

- void [\\_\\_assembleHexMap](#) (void)  
*Helper method to assemble the hex map.*

## Private Attributes

- [AssetsManager](#) \* [assets\\_manager\\_ptr](#)  
*A pointer to the assets manager.*
- [InputsHandler](#) \* [inputs\\_handler\\_ptr](#)  
*A pointer to the inputs handler.*
- [MessagesHandler](#) \* [messages\\_handler\\_ptr](#)  
*A pointer to the messages handler.*

### 3.2.1 Detailed Description

A class which defines a hex map of hex tiles.

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 HexMap()

```
HexMap::HexMap (
    int n_layers,
    AssetsManager * assets_manager_ptr,
    InputsHandler * inputs_handler_ptr,
    MessagesHandler * messages_handler_ptr )
```

Constructor for the [HexMap](#) class.

## Parameters

<i>n_layers</i>	The number of layers in the <a href="#">HexMap</a> .
<i>assets_manager_ptr</i>	Pointer to the assets manager.
<i>inputs_handler_ptr</i>	Pointer to the inputs handler.
<i>messages_handler_ptr</i>	Pointer to the messages handler.

```

191 {
192     // 1. set attributes
193     this->assets_manager_ptr = assets_manager_ptr;
194     this->inputs_handler_ptr = inputs_handler_ptr;
195     this->messages_handler_ptr = messages_handler_ptr;
196
197     this->frame = 0;
198
199     this->n_layers = n_layers;
200     if (this->n_layers < 0) {
201         this->n_layers = 0;
202     }
203
204     this->position_x = 400;
205     this->position_y = 400;
206
207     // 2. assemble n layer hex map
208     this->__assembleHexMap();
209
210     std::cout << "HexMap constructed at " << this << std::endl;
211
212     return;
213 } /* HexMap() */

```

## 3.2.2.2 ~HexMap()

```

HexMap::~HexMap (
    void )

```

Destructor for the [HexMap](#) class.

```

299 {
300     this->clear();
301
302     std::cout << "HexMap at " << this << " destroyed" << std::endl;
303
304     return;
305 } /* ~HexMap() */

```

## 3.2.3 Member Function Documentation

## 3.2.3.1 \_\_assembleHexMap()

```

void HexMap::__assembleHexMap (
    void ) [private]

```

Helper method to assemble the hex map.

```

34 {
35     // 1. add origin tile
36     HexTile* hex_ptr = new HexTile(
37         this->position_x,
38         this->position_y,
39         this->assets_manager_ptr,
40         this->inputs_handler_ptr,
41         this->messages_handler_ptr

```

```

42     );
43
44     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
45
46
47     // 2. fill out first row (reflect across origin tile)
48     for (int i = 0; i < this->n_layers; i++) {
49         hex_ptr = new HexTile(
50             this->position_x + 2 * (i + 1) * hex_ptr->minor_radius,
51             this->position_y,
52             this->assets_manager_ptr,
53             this->inputs_handler_ptr,
54             this->messages_handler_ptr
55         );
56
57         this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
58
59         hex_ptr = new HexTile(
60             this->position_x - 2 * (i + 1) * hex_ptr->minor_radius,
61             this->position_y,
62             this->assets_manager_ptr,
63             this->inputs_handler_ptr,
64             this->messages_handler_ptr
65         );
66
67         this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
68     }
69
70
71     // 3. fill out subsequent rows (reflect across first row)
72     HexTile* first_row_left_tile = hex_ptr;
73
74     int offset_count = 1;
75
76     double x_offset = 0;
77     double y_offset = 0;
78
79     for (
80         int row_width = 2 * this->n_layers;
81         row_width > this->n_layers;
82         row_width--
83     ) {
84         // 3.1. upper row
85         x_offset = first_row_left_tile->position_x +
86             2 * offset_count * first_row_left_tile->minor_radius *
87             cos(60 * (M_PI / 180));
88
89         y_offset = first_row_left_tile->position_y -
90             2 * offset_count * first_row_left_tile->minor_radius *
91             sin(60 * (M_PI / 180));
92
93         hex_ptr = new HexTile(
94             x_offset,
95             y_offset,
96             this->assets_manager_ptr,
97             this->inputs_handler_ptr,
98             this->messages_handler_ptr
99         );
100
101         this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
102
103         for (int i = 1; i < row_width; i++) {
104             x_offset += 2 * first_row_left_tile->minor_radius;
105
106             hex_ptr = new HexTile(
107                 x_offset,
108                 y_offset,
109                 this->assets_manager_ptr,
110                 this->inputs_handler_ptr,
111                 this->messages_handler_ptr
112             );
113
114             this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
115         }
116
117         // 3.2. lower row
118         x_offset = first_row_left_tile->position_x +
119             2 * offset_count * first_row_left_tile->minor_radius *
120             cos(60 * (M_PI / 180));
121
122         y_offset = first_row_left_tile->position_y +
123             2 * offset_count * first_row_left_tile->minor_radius *
124             sin(60 * (M_PI / 180));
125
126         hex_ptr = new HexTile(
127             x_offset,
128             y_offset,

```

```

129         this->assets_manager_ptr,
130         this->inputs_handler_ptr,
131         this->messages_handler_ptr
132     );
133
134     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
135
136     for (int i = 1; i < row_width; i++) {
137         x_offset += 2 * first_row_left_tile->minor_radius;
138
139         hex_ptr = new HexTile(
140             x_offset,
141             y_offset,
142             this->assets_manager_ptr,
143             this->inputs_handler_ptr,
144             this->messages_handler_ptr
145         );
146
147         this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
148     }
149
150     offset_count++;
151 }
152
153 return;
154 } /* __assembleHexMap() */

```

### 3.2.3.2 clear()

```

void HexMap::clear (
    void )

```

Method to clear the hex map.

```

265 {
266     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
267     std::map<double, HexTile*>::iterator hex_map_iter_y;
268     for (
269         hex_map_iter_x = this->hex_map.begin();
270         hex_map_iter_x != this->hex_map.end();
271         hex_map_iter_x++
272     ) {
273         for (
274             hex_map_iter_y = hex_map_iter_x->second.begin();
275             hex_map_iter_y != hex_map_iter_x->second.end();
276             hex_map_iter_y++
277         ) {
278             delete hex_map_iter_y->second;
279         }
280     }
281     this->hex_map.clear();
282
283     return;
284 } /* clear() */

```

### 3.2.3.3 draw()

```

void HexMap::draw (
    sf::RenderWindow * window_ptr )

```

Method to draw the hex map to the render window. To be called only once per frame!

#### Parameters

<i>window_ptr</i>	A pointer to the render window.
-------------------	---------------------------------

```

231 {
232     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
233     std::map<double, HexTile*>::iterator hex_map_iter_y;
234     for (
235         hex_map_iter_x = this->hex_map.begin();
236         hex_map_iter_x != this->hex_map.end();
237         hex_map_iter_x++
238     ) {
239         for (
240             hex_map_iter_y = hex_map_iter_x->second.begin();
241             hex_map_iter_y != hex_map_iter_x->second.end();
242             hex_map_iter_y++
243         ) {
244             hex_map_iter_y->second->draw(window_ptr);
245         }
246     }
247
248     this->frame++;
249     return;
250 } /* draw() */

```

### 3.2.4 Member Data Documentation

#### 3.2.4.1 assets\_manager\_ptr

`AssetsManager*` HexMap::assets\_manager\_ptr [private]

A pointer to the assets manager.

#### 3.2.4.2 frame

`int` HexMap::frame

The current frame of this object.

#### 3.2.4.3 hex\_map

`std::map<double, std::map<double, HexTile*> >` HexMap::hex\_map

A position-indexed, nested map of hex tiles.

#### 3.2.4.4 inputs\_handler\_ptr

`InputsHandler*` HexMap::inputs\_handler\_ptr [private]

A pointer to the inputs handler.

#### 3.2.4.5 messages\_handler\_ptr

```
MessagesHandler* HexMap::messages_handler_ptr [private]
```

A pointer to the messages handler.

#### 3.2.4.6 n\_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

#### 3.2.4.7 position\_x

```
double HexMap::position_x
```

The x position of the hex map's origin (i.e. central) tile.

#### 3.2.4.8 position\_y

```
double HexMap::position_y
```

The y position of the hex map's origin (i.e. central) tile.

The documentation for this class was generated from the following files:

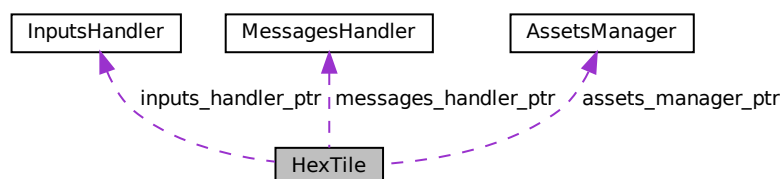
- header/HexMap/[HexMap.h](#)
- source/HexMap/[HexMap.cpp](#)

## 3.3 HexTile Class Reference

A class which defines a hex tile of the hex map.

```
#include <HexTile.h>
```

Collaboration diagram for HexTile:



## Public Member Functions

- [HexTile](#) (double, double, [AssetsManager](#) \*, [InputsHandler](#) \*, [MessagesHandler](#) \*)  
*Constructor for the [HexTile](#) class.*
- void [setTileType](#) ([TileType](#))  
*Method to set the tile type (by enum value).*
- void [draw](#) (sf::RenderWindow \*)  
*Method to draw the hex tile to the render window. To be called only once per frame!*
- [~HexTile](#) (void)  
*Destructor for the [HexTile](#) class.*

## Public Attributes

- [TileType](#) [tile\\_type](#)
- bool [show\\_node](#)  
*A boolean which indicates whether or not to show the tile node.*
- int [frame](#)  
*The current frame of this object.*
- double [position\\_x](#)  
*The x position of the tile.*
- double [position\\_y](#)  
*The y position of the tile.*
- double [major\\_radius](#)  
*The radius of the smallest bounding circle.*
- double [minor\\_radius](#)  
*The radius of the largest inscribed circle.*
- sf::CircleShape [node\\_sprite](#)  
*A circle shape to mark the tile node.*
- sf::ConvexShape [tile\\_sprite](#)  
*A convex shape which represents the tile.*

## Private Member Functions

- void [\\_\\_setUpNodeSprite](#) (void)  
*Helper method to set up node sprite.*
- void [\\_\\_setUpTileSprite](#) (void)  
*Helper method to set up tile sprite.*

## Private Attributes

- [AssetsManager](#) \* [assets\\_manager\\_ptr](#)  
*A pointer to the assets manager.*
- [InputsHandler](#) \* [inputs\\_handler\\_ptr](#)  
*A pointer to the inputs handler.*
- [MessagesHandler](#) \* [messages\\_handler\\_ptr](#)  
*A pointer to the messages handler.*



### 3.3.1 Detailed Description

A class which defines a hex tile of the hex map.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 HexTile()

```
HexTile::HexTile (
    double position_x,
    double position_y,
    AssetsManager * assets_manager_ptr,
    InputsHandler * inputs_handler_ptr,
    MessagesHandler * messages_handler_ptr )
```

Constructor for the [HexTile](#) class.

Ref: [Wikipedia \[2023\]](#)

#### Parameters

<i>position_x</i>	The x position of the tile.
<i>position_y</i>	The y position of the tile.
<i>assets_manager_ptr</i>	Pointer to the assets manager.
<i>inputs_handler_ptr</i>	Pointer to the inputs handler.
<i>messages_handler_ptr</i>	Pointer to the messages handler.

```
124 {
125     // 1. set attributes
126     this->assets_manager_ptr = assets_manager_ptr;
127     this->inputs_handler_ptr = inputs_handler_ptr;
128     this->messages_handler_ptr = messages_handler_ptr;
129
130     this->show_node = false;
131
132     this->frame = 0;
133
134     this->position_x = position_x;
135     this->position_y = position_y;
136
137     this->major_radius = 32;
138     this->minor_radius = (sqrt(3) / 2) * this->major_radius;
139
140     // 2. set up and position the node sprite
141     this->__setUpNodeSprite();
142
143     // 3. set up and position the tile sprite
144     this->__setUpTileSprite();
145
146     // 4. set tile type (default to forest)
147     this->setTileType(TileType :: FOREST);
148
149     std::cout << "HexTile constructed at " << this << std::endl;
150
151     return;
152 } /* HexTile() */
```

### 3.3.2.2 ~HexTile()

```
HexTile::~HexTile (
    void )
```

Destructor for the [HexTile](#) class.

```
253 {
254     std::cout << "HexTile at " << this << " destroyed" << std::endl;
255
256     return;
257 } /* ~HexTile() */
```

## 3.3.3 Member Function Documentation

### 3.3.3.1 \_\_setUpNodeSprite()

```
void HexTile::__setUpNodeSprite (
    void ) [private]
```

Helper method to set up node sprite.

```
34 {
35     this->node_sprite.setRadius(4);
36
37     this->node_sprite.setOrigin(
38         this->node_sprite.getLocalBounds().width / 2,
39         this->node_sprite.getLocalBounds().height / 2
40     );
41
42     this->node_sprite.setPosition(this->position_x, this->position_y);
43
44     this->node_sprite.setFillColor(sf::Color(255, 0, 0, 255));
45
46     return;
47 } /* __setUpNodeSprite() */
```

### 3.3.3.2 \_\_setUpTileSprite()

```
void HexTile::__setUpTileSprite (
    void ) [private]
```

Helper method to set up tile sprite.

```
62 {
63     int n_points = 6;
64
65     this->tile_sprite.setPointCount(n_points);
66
67     for (int i = 0; i < n_points; i++) {
68         this->tile_sprite.setPoint(
69             i,
70             sf::Vector2f(
71                 this->position_x + this->major_radius * cos((30 + 60 * i) * (M_PI / 180)),
72                 this->position_y + this->major_radius * sin((30 + 60 * i) * (M_PI / 180))
73             )
74         );
75     }
76
77     this->tile_sprite.setOutlineThickness(2);
78     this->tile_sprite.setOutlineColor(sf::Color(0, 0, 0, 255));
79
80     return;
81 } /* __setUpTileSprite() */
```

### 3.3.3.3 draw()

```
void HexTile::draw (
    sf::RenderWindow * window_ptr )
```

Method to draw the hex tile to the render window. To be called only once per frame!

#### Parameters

<i>window_ptr</i>	A pointer to the render window.
-------------------	---------------------------------

```
227 {
228     // 1. draw hex
229     window_ptr->draw(this->tile_sprite);
230
231     // 2. draw node
232     if (this->show_node) {
233         window_ptr->draw(this->node_sprite);
234     }
235
236     this->frame++;
237     return;
238 } /* draw() */
```

### 3.3.3.4 setTileType()

```
void HexTile::setTileType (
    TileType tile_type )
```

Method to set the tile type (by enum value).

#### Parameters

<i>tile_type</i>	The type (TileType) to set the tile to.
------------------	---

```
167 {
168     this->tile_type = tile_type;
169
170     switch (this->tile_type) {
171         case (TileType :: FOREST): {
172             this->tile_sprite.setFillColor(FOREST_GREEN);
173
174             break;
175         }
176
177         case (TileType :: LAKE): {
178             this->tile_sprite.setFillColor(LAKE_BLUE);
179
180             break;
181         }
182
183         case (TileType :: MOUNTAINS): {
184             this->tile_sprite.setFillColor(MOUNTAINS_GREY);
185
186             break;
187         }
188
189         case (TileType :: OCEAN): {
190             this->tile_sprite.setFillColor(OCEAN_BLUE);
191
192             break;
193         }
194
195         case (TileType :: PLAINS): {
196             this->tile_sprite.setFillColor(PLAINS_YELLOW);
197
198             break;
199         }
200     }
```

```
200
201     default: {
202         // do nothing!
203
204         break;
205     }
206 }
207
208 return;
209 } /* setTileType() */
```

### 3.3.4 Member Data Documentation

#### 3.3.4.1 assets\_manager\_ptr

`AssetsManager*` HexTile::assets\_manager\_ptr [private]

A pointer to the assets manager.

#### 3.3.4.2 frame

`int` HexTile::frame

The current frame of this object.

#### 3.3.4.3 inputs\_handler\_ptr

`InputsHandler*` HexTile::inputs\_handler\_ptr [private]

A pointer to the inputs handler.

#### 3.3.4.4 major\_radius

`double` HexTile::major\_radius

The radius of the smallest bounding circle.

#### 3.3.4.5 messages\_handler\_ptr

`MessagesHandler*` HexTile::messages\_handler\_ptr [private]

A pointer to the messages handler.

#### 3.3.4.6 minor\_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

#### 3.3.4.7 node\_sprite

```
sf::CircleShape HexTile::node_sprite
```

A circle shape to mark the tile node.

#### 3.3.4.8 position\_x

```
double HexTile::position_x
```

The x position of the tile.

#### 3.3.4.9 position\_y

```
double HexTile::position_y
```

The y position of the tile.

#### 3.3.4.10 show\_node

```
bool HexTile::show_node
```

A boolean which indicates whether or not to show the tile node.

#### 3.3.4.11 tile\_sprite

```
sf::ConvexShape HexTile::tile_sprite
```

A convex shape which represents the tile.

### 3.3.4.12 tile\_type

`TileType HexTile::tile_type`

The documentation for this class was generated from the following files:

- [header/HexMap/HexTile.h](#)
- [source/HexMap/HexTile.cpp](#)

## 3.4 InputsHandler Class Reference

A class which handles inputs from peripherals (i.e., keyboard and mouse).

```
#include <InputsHandler.h>
```

### Public Member Functions

- [InputsHandler](#) (void)  
*Constructor for the [InputsHandler](#) class.*
- void [process](#) (sf::Event \*)
- void [printKeysPressed](#) (void)  
*Method to print out which keys are currently pressed.*
- void [reset](#) (void)  
*Method to reset [InputsHandler](#). To be called once per frame (at end of frame!).*
- [~InputsHandler](#) (void)  
*Destructor for the [InputsHandler](#) class.*

### Public Attributes

- std::vector< bool > [key\\_pressed\\_once\\_vec](#)  
*A vector (bool) which indicates which keys have been pressed once. Useful for discrete inputs.*
- std::vector< bool > [key\\_press\\_vec](#)  
*A vector <bool> which indicates which keys are currently pressed. Useful for smooth movement.*
- std::map< sf::Keyboard::Key, std::string > [key\\_code\\_map](#)  
*A map from key codes to corresponding string representations.*

### Private Member Functions

- void [\\_\\_constructKeyCodeMap](#) (void)  
*Helper method to construct a map from sf::Keyboard::Key to a string representation of the corresponding key.*

### 3.4.1 Detailed Description

A class which handles inputs from peripherals (i.e., keyboard and mouse).

## 3.4.2 Constructor & Destructor Documentation

### 3.4.2.1 InputsHandler()

```
InputsHandler::InputsHandler (
    void )
```

Constructor for the [InputsHandler](#) class.

```
379 {
380     this->key_pressed_once_vec.resize(sf::Keyboard::KeyCount, false);
381     this->key_press_vec.resize(sf::Keyboard::KeyCount, false);
382
383     this->__constructKeyCodeMap();
384
385     std::cout << "InputsHandler constructed at " << this << std::endl;
386
387     return;
388 } /* InputsHandler() */
```

### 3.4.2.2 ~InputsHandler()

```
InputsHandler::~~InputsHandler (
    void )
```

Destructor for the [InputsHandler](#) class.

```
499 {
500     std::cout << "InputsHandler at " << this << " destroyed" << std::endl;
501
502     return;
503 } /* ~InputsHandler() */
```

## 3.4.3 Member Function Documentation

### 3.4.3.1 \_\_constructKeyCodeMap()

```
void InputsHandler::__constructKeyCodeMap (
    void ) [private]
```

Helper method to construct a map from sf::Keyboard::Key to a string representation of the corresponding key.

```
35 {
36     // 1. unknown keys
37     this->key_code_map.insert(
38         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Unknown, "Unknown")
39     );
40
41
42     // 2. alpha keys
43     this->key_code_map.insert(
44         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::A, "A")
45     );
46     this->key_code_map.insert(
47         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::B, "B")
48     );
49     this->key_code_map.insert(
50         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::C, "C")
```

```

51     );
52     this->key_code_map.insert(
53         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::D, "D")
54     );
55     this->key_code_map.insert(
56         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::E, "E")
57     );
58     this->key_code_map.insert(
59         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F, "F")
60     );
61     this->key_code_map.insert(
62         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::G, "G")
63     );
64     this->key_code_map.insert(
65         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::H, "H")
66     );
67     this->key_code_map.insert(
68         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::I, "I")
69     );
70     this->key_code_map.insert(
71         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::J, "J")
72     );
73     this->key_code_map.insert(
74         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::K, "K")
75     );
76     this->key_code_map.insert(
77         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::L, "L")
78     );
79     this->key_code_map.insert(
80         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::M, "M")
81     );
82     this->key_code_map.insert(
83         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::N, "N")
84     );
85     this->key_code_map.insert(
86         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::O, "O")
87     );
88     this->key_code_map.insert(
89         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::P, "P")
90     );
91     this->key_code_map.insert(
92         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Q, "Q")
93     );
94     this->key_code_map.insert(
95         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::R, "R")
96     );
97     this->key_code_map.insert(
98         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::S, "S")
99     );
100    this->key_code_map.insert(
101        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::T, "T")
102    );
103    this->key_code_map.insert(
104        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::U, "U")
105    );
106    this->key_code_map.insert(
107        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::V, "V")
108    );
109    this->key_code_map.insert(
110        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::W, "W")
111    );
112    this->key_code_map.insert(
113        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::X, "X")
114    );
115    this->key_code_map.insert(
116        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Y, "Y")
117    );
118    this->key_code_map.insert(
119        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Z, "Z")
120    );
121
122
123    // 3. numeric keys
124    this->key_code_map.insert(
125        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num0, "0")
126    );
127    this->key_code_map.insert(
128        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num1, "1")
129    );
130    this->key_code_map.insert(
131        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num2, "2")
132    );
133    this->key_code_map.insert(
134        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num3, "3")
135    );
136    this->key_code_map.insert(
137        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num4, "4")

```



```

138     );
139     this->key_code_map.insert (
140         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num5, "5")
141     );
142     this->key_code_map.insert (
143         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num6, "6")
144     );
145     this->key_code_map.insert (
146         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num7, "7")
147     );
148     this->key_code_map.insert (
149         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num8, "8")
150     );
151     this->key_code_map.insert (
152         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num9, "9")
153     );
154     this->key_code_map.insert (
155         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad0, "0")
156     );
157     this->key_code_map.insert (
158         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad1, "1")
159     );
160     this->key_code_map.insert (
161         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad2, "2")
162     );
163     this->key_code_map.insert (
164         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad3, "3")
165     );
166     this->key_code_map.insert (
167         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad4, "4")
168     );
169     this->key_code_map.insert (
170         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad5, "5")
171     );
172     this->key_code_map.insert (
173         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad6, "6")
174     );
175     this->key_code_map.insert (
176         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad7, "7")
177     );
178     this->key_code_map.insert (
179         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad8, "8")
180     );
181     this->key_code_map.insert (
182         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad9, "9")
183     );
184
185
186     // 4. direction keys
187     this->key_code_map.insert (
188         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Left, "Left")
189     );
190     this->key_code_map.insert (
191         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Right, "Right")
192     );
193     this->key_code_map.insert (
194         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Up, "Up")
195     );
196     this->key_code_map.insert (
197         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Down, "Down")
198     );
199
200
201     // 5. function keys
202     this->key_code_map.insert (
203         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F1, "F1")
204     );
205     this->key_code_map.insert (
206         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F2, "F2")
207     );
208     this->key_code_map.insert (
209         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F3, "F3")
210     );
211     this->key_code_map.insert (
212         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F4, "F4")
213     );
214     this->key_code_map.insert (
215         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F5, "F5")
216     );
217     this->key_code_map.insert (
218         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F6, "F6")
219     );
220     this->key_code_map.insert (
221         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F7, "F7")
222     );
223     this->key_code_map.insert (
224         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F8, "F8")

```

```

225     );
226     this->key_code_map.insert (
227         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F9, "F9")
228     );
229     this->key_code_map.insert (
230         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F10, "F10")
231     );
232     this->key_code_map.insert (
233         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F11, "F11")
234     );
235     this->key_code_map.insert (
236         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F12, "F12")
237     );
238     this->key_code_map.insert (
239         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F13, "F13")
240     );
241     this->key_code_map.insert (
242         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F14, "F14")
243     );
244     this->key_code_map.insert (
245         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F15, "F15")
246     );
247
248
249     // 6. other keys
250     this->key_code_map.insert (
251         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Escape, "Escape")
252     );
253     this->key_code_map.insert (
254         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LControl, "LCtrl")
255     );
256     this->key_code_map.insert (
257         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LShift, "LShift")
258     );
259     this->key_code_map.insert (
260         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LAlt, "LAlt")
261     );
262     this->key_code_map.insert (
263         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LSystem, "LSystem")
264     );
265     this->key_code_map.insert (
266         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RControl, "RCtrl")
267     );
268     this->key_code_map.insert (
269         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RShift, "RShift")
270     );
271     this->key_code_map.insert (
272         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RAlt, "RAlt")
273     );
274     this->key_code_map.insert (
275         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RSystem, "RSystem")
276     );
277     this->key_code_map.insert (
278         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Menu, "Menu")
279     );
280     this->key_code_map.insert (
281         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LBracket, "LBracket")
282     );
283     this->key_code_map.insert (
284         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RBracket, "RBracket")
285     );
286     this->key_code_map.insert (
287         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Semicolon, "Semicolon")
288     );
289     this->key_code_map.insert (
290         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Comma, "Comma")
291     );
292     this->key_code_map.insert (
293         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Period, "Period")
294     );
295     this->key_code_map.insert (
296         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Quote, "Quote")
297     );
298     this->key_code_map.insert (
299         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Slash, "Slash")
300     );
301     this->key_code_map.insert (
302         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backslash, "Backslash")
303     );
304     this->key_code_map.insert (
305         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tilde, "Tilde")
306     );
307     this->key_code_map.insert (
308         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Equal, "Equal")
309     );
310     this->key_code_map.insert (
311         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Hyphen, "Hyphen")

```

```

312     );
313     this->key_code_map.insert (
314         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Space, "Space")
315     );
316     this->key_code_map.insert (
317         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Enter, "Enter")
318     );
319     this->key_code_map.insert (
320         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backspace, "Backspace")
321     );
322     this->key_code_map.insert (
323         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tab, "Tab")
324     );
325     this->key_code_map.insert (
326         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageUp, "PageUp")
327     );
328     this->key_code_map.insert (
329         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageDown, "PageDown")
330     );
331     this->key_code_map.insert (
332         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::End, "End")
333     );
334     this->key_code_map.insert (
335         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Home, "Home")
336     );
337     this->key_code_map.insert (
338         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Insert, "Insert")
339     );
340     this->key_code_map.insert (
341         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Delete, "Delete")
342     );
343     this->key_code_map.insert (
344         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Add, "Add")
345     );
346     this->key_code_map.insert (
347         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Subtract, "Subtract")
348     );
349     this->key_code_map.insert (
350         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Multiply, "Multiply")
351     );
352     this->key_code_map.insert (
353         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Divide, "Divide")
354     );
355     this->key_code_map.insert (
356         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Pause, "Pause")
357     );
358
359     return;
360 } /* __constructKeyCodeMap() */

```

### 3.4.3.2 printKeysPressed()

```

void InputsHandler::printKeysPressed (
    void )

```

Method to print out which keys are currently pressed.

```

448 {
449     std::string print_str = "";
450
451     for (size_t i = 0; i < this->key_press_vec.size(); i++) {
452         if (this->key_press_vec[i]) {
453             print_str += this->key_code_map[sf::Keyboard::Key(i)];
454             print_str += ", ";
455         }
456     }
457
458     if (not print_str.empty()) {
459         std::cout << "Keys pressed: " << print_str << std::endl;
460     }
461
462     return;
463 } /* printKeysPressed() */

```

### 3.4.3.3 process()

```

void InputsHandler::process (
    sf::Event * event_ptr )
405 {
406     // 1. update state of key press vectors
407     switch (event_ptr->type) {
408         case (sf::Event::KeyPressed): {
409             if (not this->key_press_vec[event_ptr->key.code]) {
410                 this->key_pressed_once_vec[event_ptr->key.code] = true;
411             }
412             this->key_press_vec[event_ptr->key.code] = true;
413         }
414         break;
415     }
416     case (sf::Event::KeyReleased): {
417         this->key_pressed_once_vec[event_ptr->key.code] = false;
418         this->key_press_vec[event_ptr->key.code] = false;
419     }
420     break;
421     default: {
422         // do nothing!
423     }
424     break;
425 }
426 return;
427 }
428 }
429 }
430 }
431 }
432 }
433 } /* process() */

```

### 3.4.3.4 reset()

```

void InputsHandler::reset (
    void )

```

Method to reset [InputsHandler](#). To be called once per frame (at end of frame!).

```

478 {
479     for (size_t i = 0; i < this->key_press_vec.size(); i++) {
480         this->key_pressed_once_vec[i] = false;
481     }
482     return;
483 }
484 } /* reset() */

```

## 3.4.4 Member Data Documentation

### 3.4.4.1 key\_code\_map

```
std::map<sf::Keyboard::Key, std::string> InputsHandler::key_code_map
```

A map from key codes to corresponding string representations.

#### 3.4.4.2 key\_press\_vec

```
std::vector<bool> InputsHandler::key_press_vec
```

A vector <bool> which indicates which keys are currently pressed. Useful for smooth movement.

#### 3.4.4.3 key\_pressed\_once\_vec

```
std::vector<bool> InputsHandler::key_pressed_once_vec
```

A vector (bool) which indicates which keys have been pressed once. Useful for discrete inputs.

The documentation for this class was generated from the following files:

- [header/ESC\\_core/InputsHandler.h](#)
- [source/ESC\\_core/InputsHandler.cpp](#)

## 3.5 MessagesHandler Class Reference

A class which handles message traffic between game objects.

```
#include <MessagesHandler.h>
```

### Public Member Functions

- [MessagesHandler](#) (void)  
*Constructor for the [MessagesHandler](#) class.*
- [~MessagesHandler](#) (void)  
*Destructor for the [MessagesHandler](#) class.*

### 3.5.1 Detailed Description

A class which handles message traffic between game objects.

### 3.5.2 Constructor & Destructor Documentation

### 3.5.2.1 MessagesHandler()

```
MessagesHandler::MessagesHandler (
    void )
```

Constructor for the [MessagesHandler](#) class.

```
46 {
47     //...
48
49     std::cout << "MessagesHandler constructed at " << this << std::endl;
50
51     return;
52 } /* MessagesHandler() */
```

### 3.5.2.2 ~MessagesHandler()

```
MessagesHandler::~~MessagesHandler (
    void )
```

Destructor for the [MessagesHandler](#) class.

```
75 {
76     std::cout << "MessagesHandler at " << this << " destroyed" << std::endl;
77
78     return;
79 } /* ~MessagesHandler() */
```

The documentation for this class was generated from the following files:

- [header/ESC\\_core/MessagesHandler.h](#)
- [source/ESC\\_core/MessagesHandler.cpp](#)

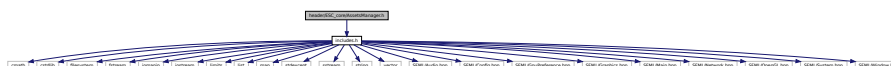
## File Documentation

Header file for the `AssetsManager` class.

```

// ...
// Include dependency graph for AssetsManager.h:

```



```

graph TD
    A[header/ESC_core/AssetsManager.h] --> B[header/HexMap/HexTile.h]
    A --> C[source/ESC_core/AssetsManager.cpp]
    A --> D[test/ESC_core/test_AssetsManager.cpp]
    B --> E[header/HexMap/HexMap.h]
    B --> F[source/HexMap/HexTile.cpp]
    E --> G[source/HexMap/HexMap.cpp]
    E --> H[test/HexMap/test_HexMap.cpp]
    G --> H
  
```

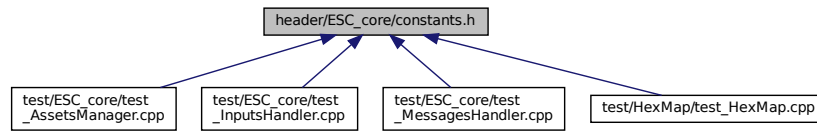
- class `AssetsManager`  
*A class which manages visual and sound assets.*

Header file for the `AssetsManager` class.

## 4.2 header/ESC\_core/constants.h File Reference

Header file for various constants.

This graph shows which files directly or indirectly include this file:



### Variables

- const int `FRAMES_PER_SECOND` = 60  
*Target frames per second.*
- const double `SECONDS_PER_FRAME` = 1.0 / 60  
*Target seconds per frame (just reciprocal of target frames per second).*

### 4.2.1 Detailed Description

Header file for various constants.

### 4.2.2 Variable Documentation

#### 4.2.2.1 FRAMES\_PER\_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

#### 4.2.2.2 SECONDS\_PER\_FRAME

```
const double SECONDS_PER_FRAME = 1.0 / 60
```

Target seconds per frame (just reciprocal of target frames per second).





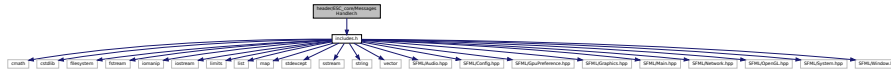


## 4.6 header/ESC\_core/MessagesHandler.h File Reference

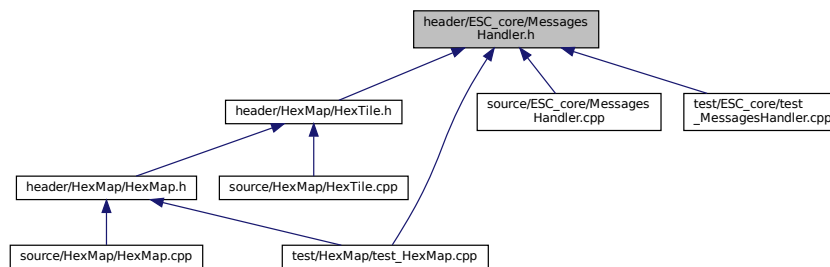
Header file for the [MessagesHandler](#) class.

```
#include "includes.h"
```

Include dependency graph for MessagesHandler.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [MessagesHandler](#)  
A class which handles message traffic between game objects.

### 4.6.1 Detailed Description

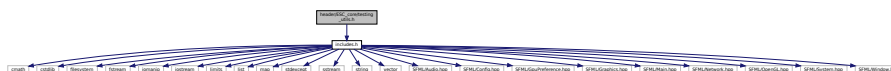
Header file for the [MessagesHandler](#) class.

## 4.7 header/ESC\_core/testing\_utils.h File Reference

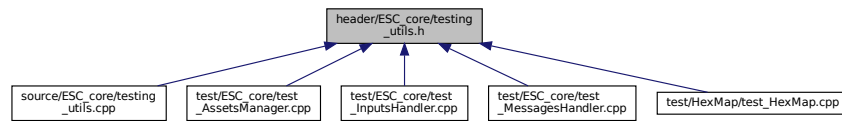
Header file for various testing utilities.

```
#include "includes.h"
```

Include dependency graph for testing\_utils.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void `printGreen` (std::string)  
A function that sends green text to std::cout.
- void `printGold` (std::string)  
A function that sends gold text to std::cout.
- void `printRed` (std::string)  
A function that sends red text to std::cout.
- void `testFloatEquals` (double, double, std::string, int)  
Tests for the equality of two floating point numbers  $x$  and  $y$  (to within `FLOAT_TOLERANCE`).
- void `testGreaterThan` (double, double, std::string, int)  
Tests if  $x > y$ .
- void `testGreaterThanOrEqualTo` (double, double, std::string, int)  
Tests if  $x \geq y$ .
- void `testLessThan` (double, double, std::string, int)  
Tests if  $x < y$ .
- void `testLessThanOrEqualTo` (double, double, std::string, int)  
Tests if  $x \leq y$ .
- void `testTruth` (bool, std::string, int)  
Tests if the given statement is true.
- void `expectedErrorNotDetected` (std::string, int)  
A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

## Variables

- const double `FLOAT_TOLERANCE` = 1e-6  
Tolerance for floating point equality tests.

### 4.7.1 Detailed Description

Header file for various testing utilities.

This is a library of utility functions used throughout the various test suites.

### 4.7.2 Function Documentation

#### 4.7.2.1 `expectedErrorNotDetected()`

```
void expectedErrorNotDetected (
    std::string file,
    int line )
```

A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

## Parameters

<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

430 {
431     std::string error_str = "\n ERROR   failed to throw expected error prior to line ";
432     error_str += std::to_string(line);
433     error_str += " of ";
434     error_str += file;
435
436     #ifdef _WIN32
437         std::cout << error_str << std::endl;
438     #endif
439
440     throw std::runtime_error(error_str);
441     return;
442 } /* expectedErrorNotDetected() */

```

## 4.7.2.2 printGold()

```

void printGold (
    std::string input_str )

```

A function that sends gold text to std::cout.

## Parameters

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```

82 {
83     std::cout << "\x1B[33m" << input_str << "\033[0m";
84     return;
85 } /* printGold() */

```

## 4.7.2.3 printGreen()

```

void printGreen (
    std::string input_str )

```

A function that sends green text to std::cout.

## Parameters

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```

62 {
63     std::cout << "\x1B[32m" << input_str << "\033[0m";
64     return;
65 } /* printGreen() */

```

## 4.7.2.4 printRed()

```

void printRed (

```

```
std::string input_str )
```

A function that sends red text to `std::cout`.

#### Parameters

<i>input_str</i>	The text of the string to be sent to <code>std::cout</code> .
------------------	---

```
102 {
103     std::cout << "\x1B[31m" << input_str << "\033[0m";
104     return;
105 } /* printRed() */
```

#### 4.7.2.5 testFloatEquals()

```
void testFloatEquals (
    double x,
    double y,
    std::string file,
    int line )
```

Tests for the equality of two floating point numbers *x* and *y* (to within `FLOAT_TOLERANCE`).

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in " <code>__FILE__</code> ").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in " <code>__LINE__</code> ").

```
136 {
137     if (fabs(x - y) <= FLOAT_TOLERANCE) {
138         return;
139     }
140
141     std::string error_str = "ERROR: testFloatEquals():\t in ";
142     error_str += file;
143     error_str += "\tline ";
144     error_str += std::to_string(line);
145     error_str += ":\t\n";
146     error_str += std::to_string(x);
147     error_str += " and ";
148     error_str += std::to_string(y);
149     error_str += " are not equal to within +/- ";
150     error_str += std::to_string(FLOAT_TOLERANCE);
151     error_str += "\n";
152
153     #ifdef _WIN32
154         std::cout << error_str << std::endl;
155     #endif
156
157     throw std::runtime_error(error_str);
158     return;
159 } /* testFloatEquals() */
```

#### 4.7.2.6 testGreaterThan()

```
void testGreaterThan (
    double x,
```

```
double y,
std::string file,
int line )
```

Tests if  $x > y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
189 {
190     if (x > y) {
191         return;
192     }
193
194     std::string error_str = "ERROR: testGreaterThan():\t in ";
195     error_str += file;
196     error_str += "\tline ";
197     error_str += std::to_string(line);
198     error_str += ":\t\n";
199     error_str += std::to_string(x);
200     error_str += " is not greater than ";
201     error_str += std::to_string(y);
202     error_str += "\n";
203
204     #ifdef _WIN32
205         std::cout << error_str << std::endl;
206     #endif
207
208     throw std::runtime_error(error_str);
209     return;
210 } /* testGreaterThan() */
```

#### 4.7.2.7 testGreaterThanOrEqualTo()

```
void testGreaterThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )
```

Tests if  $x \geq y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
240 {
241     if (x >= y) {
242         return;
243     }
244
245     std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
246     error_str += file;
247     error_str += "\tline ";
248     error_str += std::to_string(line);
249     error_str += ":\t\n";
```

```

250     error_str += std::to_string(x);
251     error_str += " is not greater than or equal to ";
252     error_str += std::to_string(y);
253     error_str += "\n";
254
255     #ifdef _WIN32
256         std::cout << error_str << std::endl;
257     #endif
258
259     throw std::runtime_error(error_str);
260     return;
261 } /* testGreaterThanOrEqualTo() */

```

#### 4.7.2.8 testLessThan()

```

void testLessThan (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x < y$ .

##### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

291 {
292     if (x < y) {
293         return;
294     }
295
296     std::string error_str = "ERROR: testLessThan():\t in ";
297     error_str += file;
298     error_str += "\tline ";
299     error_str += std::to_string(line);
300     error_str += ":\t\n";
301     error_str += std::to_string(x);
302     error_str += " is not less than ";
303     error_str += std::to_string(y);
304     error_str += "\n";
305
306     #ifdef _WIN32
307         std::cout << error_str << std::endl;
308     #endif
309
310     throw std::runtime_error(error_str);
311     return;
312 } /* testLessThan() */

```

#### 4.7.2.9 testLessThanOrEqualTo()

```

void testLessThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x \leq y$ .



## Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

342 {
343     if (x <= y) {
344         return;
345     }
346
347     std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
348     error_str += file;
349     error_str += "\tline ";
350     error_str += std::to_string(line);
351     error_str += ":\t\n";
352     error_str += std::to_string(x);
353     error_str += " is not less than or equal to ";
354     error_str += std::to_string(y);
355     error_str += "\n";
356
357     #ifdef _WIN32
358         std::cout << error_str << std::endl;
359     #endif
360
361     throw std::runtime_error(error_str);
362     return;
363 } /* testLessThanOrEqualTo() */

```

## 4.7.2.10 testTruth()

```

void testTruth (
    bool statement,
    std::string file,
    int line )

```

Tests if the given statement is true.

## Parameters

<i>statement</i>	The statement whose truth is to be tested ("1 == 0", for example).
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

390 {
391     if (statement) {
392         return;
393     }
394
395     std::string error_str = "ERROR: testTruth():\t in ";
396     error_str += file;
397     error_str += "\tline ";
398     error_str += std::to_string(line);
399     error_str += ":\t\n";
400     error_str += "Given statement is not true";
401
402     #ifdef _WIN32
403         std::cout << error_str << std::endl;
404     #endif
405
406     throw std::runtime_error(error_str);
407     return;
408 } /* testTruth() */

```

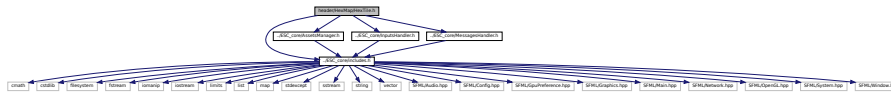


## 4.9 header/HexMap/HexTile.h File Reference

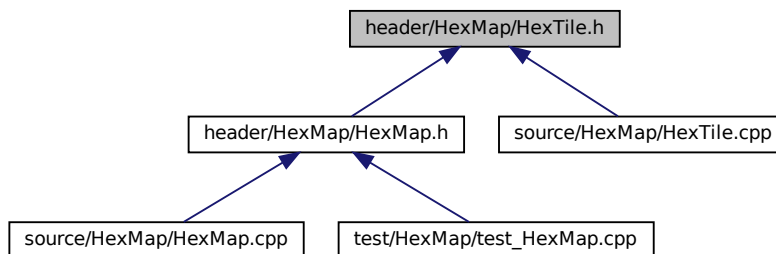
Header file for the [HexTile](#) class.

```
#include "../ESC_core/includes.h"
#include "../ESC_core/AssetsManager.h"
#include "../ESC_core/InputsHandler.h"
#include "../ESC_core/MessagesHandler.h"
```

Include dependency graph for HexTile.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [HexTile](#)

*A class which defines a hex tile of the hex map.*

### Enumerations

- enum [TileType](#) {  
[FOREST](#) , [LAKE](#) , [MOUNTAINS](#) , [OCEAN](#) ,  
[PLAINS](#) , [N\\_TILE\\_TYPES](#) }

*An enumeration of the different tile types.*

### Functions

- const sf::Color [FOREST\\_GREEN](#) (34, 139, 34)  
*The base colour of a forest tile.*
- const sf::Color [LAKE\\_BLUE](#) (0, 102, 204)  
*The base colour of a lake (water) tile.*
- const sf::Color [MOUNTAINS\\_GREY](#) (97, 110, 113)  
*The base colour of a mountains tile.*
- const sf::Color [OCEAN\\_BLUE](#) (0, 51, 102)  
*The base colour of an ocean (water) tile.*
- const sf::Color [PLAINS\\_YELLOW](#) (245, 222, 133)  
*The base colour of a plains tile.*

### 4.9.1 Detailed Description

Header file for the [HexTile](#) class.

### 4.9.2 Enumeration Type Documentation

#### 4.9.2.1 TileType

```
enum TileType
```

An enumeration of the different tile types.

Enumerator

FOREST	A forest tile.
LAKE	A lake tile.
MOUNTAINS	A mountains tile.
OCEAN	An ocean tile.
PLAINS	A plains tile.
N_TILE_TYPES	A simple hack to get the number of elements in TileType.

```
34     {  
35     FOREST,  
36     LAKE,  
37     MOUNTAINS,  
38     OCEAN,  
39     PLAINS,  
40     N\_TILE\_TYPES  
41 };
```

### 4.9.3 Function Documentation

#### 4.9.3.1 FOREST\_GREEN()

```
const sf::Color FOREST_GREEN (  
    34 ,  
    139 ,  
    34 )
```

The base colour of a forest tile.

#### 4.9.3.2 LAKE\_BLUE()

```
const sf::Color LAKE_BLUE (
    0 ,
    102 ,
    204 )
```

The base colour of a lake (water) tile.

#### 4.9.3.3 MOUNTAINS\_GREY()

```
const sf::Color MOUNTAINS_GREY (
    97 ,
    110 ,
    113 )
```

The base colour of a mountains tile.

#### 4.9.3.4 OCEAN\_BLUE()

```
const sf::Color OCEAN_BLUE (
    0 ,
    51 ,
    102 )
```

The base colour of an ocean (water) tile.

#### 4.9.3.5 PLAINS\_YELLOW()

```
const sf::Color PLAINS_YELLOW (
    245 ,
    222 ,
    133 )
```

The base colour of a plains tile.

#### 4.10 source/ESC\_core/AssetsManager.cpp File Reference

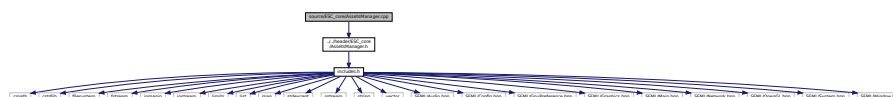
Implementation file for the `AssetsManager` class.

```
#include "../..header/ESC_core/AssetsManager.h"
```

```

// Include dependency graph for AssetsManager.cpp:

```







**Parameters**

<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

430 {
431     std::string error_str = "\n ERROR   failed to throw expected error prior to line ";
432     error_str += std::to_string(line);
433     error_str += " of ";
434     error_str += file;
435
436     #ifdef _WIN32
437         std::cout << error_str << std::endl;
438     #endif
439
440     throw std::runtime_error(error_str);
441     return;
442 } /* expectedErrorNotDetected() */

```

**4.13.2.2 printGold()**

```

void printGold (
    std::string input_str )

```

A function that sends gold text to std::cout.

**Parameters**

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```

82 {
83     std::cout << "\x1B[33m" << input_str << "\033[0m";
84     return;
85 } /* printGold() */

```

**4.13.2.3 printGreen()**

```

void printGreen (
    std::string input_str )

```

A function that sends green text to std::cout.

**Parameters**

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```

62 {
63     std::cout << "\x1B[32m" << input_str << "\033[0m";
64     return;
65 } /* printGreen() */

```

**4.13.2.4 printRed()**

```

void printRed (

```



```
std::string input_str )
```

A function that sends red text to `std::cout`.

#### Parameters

<i>input_str</i>	The text of the string to be sent to <code>std::cout</code> .
------------------	---

```
102 {
103     std::cout << "\x1B[31m" << input_str << "\033[0m";
104     return;
105 } /* printRed() */
```

#### 4.13.2.5 testFloatEquals()

```
void testFloatEquals (
    double x,
    double y,
    std::string file,
    int line )
```

Tests for the equality of two floating point numbers *x* and *y* (to within `FLOAT_TOLERANCE`).

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in " <code>__FILE__</code> ").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in " <code>__LINE__</code> ").

```
136 {
137     if (fabs(x - y) <= FLOAT_TOLERANCE) {
138         return;
139     }
140
141     std::string error_str = "ERROR: testFloatEquals():\t in ";
142     error_str += file;
143     error_str += "\tline ";
144     error_str += std::to_string(line);
145     error_str += ":\t\n";
146     error_str += std::to_string(x);
147     error_str += " and ";
148     error_str += std::to_string(y);
149     error_str += " are not equal to within +/- ";
150     error_str += std::to_string(FLOAT_TOLERANCE);
151     error_str += "\n";
152
153     #ifdef _WIN32
154         std::cout << error_str << std::endl;
155     #endif
156
157     throw std::runtime_error(error_str);
158     return;
159 } /* testFloatEquals() */
```

#### 4.13.2.6 testGreaterThan()

```
void testGreaterThan (
    double x,
```

```
double y,
std::string file,
int line )
```

Tests if  $x > y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
189 {
190     if (x > y) {
191         return;
192     }
193
194     std::string error_str = "ERROR: testGreaterThan():\t in ";
195     error_str += file;
196     error_str += "\tline ";
197     error_str += std::to_string(line);
198     error_str += ":\t\n";
199     error_str += std::to_string(x);
200     error_str += " is not greater than ";
201     error_str += std::to_string(y);
202     error_str += "\n";
203
204     #ifdef _WIN32
205         std::cout << error_str << std::endl;
206     #endif
207
208     throw std::runtime_error(error_str);
209     return;
210 } /* testGreaterThan() */
```

#### 4.13.2.7 testGreaterThanOrEqualTo()

```
void testGreaterThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )
```

Tests if  $x \geq y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
240 {
241     if (x >= y) {
242         return;
243     }
244
245     std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
246     error_str += file;
247     error_str += "\tline ";
248     error_str += std::to_string(line);
249     error_str += ":\t\n";
```

```

250     error_str += std::to_string(x);
251     error_str += " is not greater than or equal to ";
252     error_str += std::to_string(y);
253     error_str += "\n";
254
255     #ifdef _WIN32
256         std::cout << error_str << std::endl;
257     #endif
258
259     throw std::runtime_error(error_str);
260     return;
261 } /* testGreaterThanOrEqualTo() */

```

#### 4.13.2.8 testLessThan()

```

void testLessThan (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x < y$ .

##### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

291 {
292     if (x < y) {
293         return;
294     }
295
296     std::string error_str = "ERROR: testLessThan():\t in ";
297     error_str += file;
298     error_str += "\tline ";
299     error_str += std::to_string(line);
300     error_str += ":\t\n";
301     error_str += std::to_string(x);
302     error_str += " is not less than ";
303     error_str += std::to_string(y);
304     error_str += "\n";
305
306     #ifdef _WIN32
307         std::cout << error_str << std::endl;
308     #endif
309
310     throw std::runtime_error(error_str);
311     return;
312 } /* testLessThan() */

```

#### 4.13.2.9 testLessThanOrEqualTo()

```

void testLessThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x \leq y$ .

## Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

342 {
343     if (x <= y) {
344         return;
345     }
346
347     std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
348     error_str += file;
349     error_str += "\tline ";
350     error_str += std::to_string(line);
351     error_str += ":\t\n";
352     error_str += std::to_string(x);
353     error_str += " is not less than or equal to ";
354     error_str += std::to_string(y);
355     error_str += "\n";
356
357     #ifdef _WIN32
358         std::cout << error_str << std::endl;
359     #endif
360
361     throw std::runtime_error(error_str);
362     return;
363 } /* testLessThanOrEqualTo() */

```

## 4.13.2.10 testTruth()

```

void testTruth (
    bool statement,
    std::string file,
    int line )

```

Tests if the given statement is true.

## Parameters

<i>statement</i>	The statement whose truth is to be tested ("1 == 0", for example).
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

390 {
391     if (statement) {
392         return;
393     }
394
395     std::string error_str = "ERROR: testTruth():\t in ";
396     error_str += file;
397     error_str += "\tline ";
398     error_str += std::to_string(line);
399     error_str += ":\t\n";
400     error_str += "Given statement is not true";
401
402     #ifdef _WIN32
403         std::cout << error_str << std::endl;
404     #endif
405
406     throw std::runtime_error(error_str);
407     return;
408 } /* testTruth() */

```



## Functions

- int [main](#) (int argc, char \*\*argv)

### 4.16.1 Detailed Description

Suite of tests for the [AssetsManager](#) class.

A suite of tests for the [AssetsManager](#) class.

### 4.16.2 Function Documentation

#### 4.16.2.1 main()

```
int main (
    int argc,
    char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif /* _WIN32 */

    printGold("\tTesting AssetsManager");
    std::cout << std::endl;

    srand(time(NULL));
    int n_dots = 8;

    try {
        // 1. construct
        InputsHandler inputs_handler;
        AssetsManager assets_manager;

        // 2. load/open some test assets
        assets_manager.loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
        assets_manager.loadTexture(
            "assets/ESC_brand/ESC_key_98x81.png",
            "ESC_key_98x81"
        );
        assets_manager.loadSound("assets/ESC_brand/key_press.ogg", "key_press");
        assets_manager.loadTrack(
            "assets/audio/tracks/AlexanderBlu_BackgroundElectronicModernMusic.ogg",
            "AlexanderBlu_BackgroundElectronicModernMusic"
        );

        // 3. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(800, 600), "Testing AssetsManager");

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            800,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            600,
            __FILE__,
            __LINE__
        );
    }
}
```

```

88     );
89
90     unsigned long long int frame = 0;
91     double time_since_run_s = 0;
92
93     assets_manager.playTrack();
94
95     sf::Sprite ESC_key(*(assets_manager.getTextTexture("ESC_key_98x81")));
96
97     double sprite_width = ESC_key.getLocalBounds().width;
98     double sprite_height = ESC_key.getLocalBounds().height;
99
100    double sprite_velocity_x = 256 * (2 * ((double)rand() / RAND_MAX) - 1);
101    double sprite_velocity_y = 256 * (2 * ((double)rand() / RAND_MAX) - 1);
102
103    ESC_key.setOrigin(sprite_width / 2, sprite_height / 2);
104    ESC_key.setPosition(
105        (screen_width - sprite_width) * ((double)rand() / RAND_MAX) + sprite_width / 2,
106        (screen_height - sprite_height) * ((double)rand() / RAND_MAX) + sprite_height / 2
107    );
108
109    sf::Text click_text(
110        "CLICK!",
111        *(assets_manager.getFont("DroidSansMono")),
112        16
113    );
114
115    double text_width = click_text.getLocalBounds().width;
116    double text_height = click_text.getLocalBounds().height;
117
118    click_text.setOrigin(text_width / 2, text_height / 2);
119
120    int alpha = 255;
121
122    click_text.setFillColor(sf::Color(255, 255, 255, alpha));
123
124    while (window.isOpen()) {
125        time_since_run_s = clock.getElapsedTime().asSeconds();
126
127        if (
128            time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
129        ) {
130            while (window.pollEvent(event))
131            {
132                //...
133
134                if (event.type == sf::Event::Closed) {
135                    window.close();
136                }
137            }
138
139            ESC_key.move(
140                sprite_velocity_x * SECONDS_PER_FRAME,
141                sprite_velocity_y * SECONDS_PER_FRAME
142            );
143
144            if (
145                ESC_key.getPosition().x <= sprite_width / 2 or
146                ESC_key.getPosition().x >= screen_width - sprite_width / 2
147            ) {
148                sprite_velocity_x *= -1;
149
150                assets_manager.getSound("key_press")->play();
151
152                alpha = 255;
153                click_text.setPosition(
154                    ESC_key.getPosition().x,
155                    ESC_key.getPosition().y
156                );
157            }
158
159            if (
160                ESC_key.getPosition().y <= sprite_height / 2 or
161                ESC_key.getPosition().y >= screen_height - sprite_height / 2
162            ) {
163                sprite_velocity_y *= -1;
164
165                assets_manager.getSound("key_press")->play();
166
167                alpha = 255;
168                click_text.setPosition(
169                    ESC_key.getPosition().x,
170                    ESC_key.getPosition().y
171                );
172            }
173
174            window.clear();

```





## 4.17.2 Function Documentation

### 4.17.2.1 main()

```

int main (
    int argc,
    char ** argv )
{
    36 {
    37     #ifdef _WIN32
    38         activateVirtualTerminal();
    39     #endif /* _WIN32 */
    40
    41     printGold("\tTesting InputsHandler");
    42     std::cout << std::endl;
    43
    44     srand(time(NULL));
    45     int n_dots = 8;
    46
    47
    48     try {
    49         // 1. construct and spot check attributes
    50         InputsHandler inputs_handler;
    51
    52         testFloatEquals(
    53             int(sf::Keyboard::KeyCount),
    54             101,
    55             __FILE__,
    56             __LINE__
    57         );
    58
    59         testFloatEquals(
    60             inputs_handler.key_press_vec.size(),
    61             int(sf::Keyboard::KeyCount),
    62             __FILE__,
    63             __LINE__
    64         );
    65
    66         testFloatEquals(
    67             inputs_handler.key_pressed_once_vec.size(),
    68             int(sf::Keyboard::KeyCount),
    69             __FILE__,
    70             __LINE__
    71         );
    72
    73
    74         // 2. test game loop
    75         sf::Clock clock;
    76         sf::Event event;
    77         sf::RenderWindow window(sf::VideoMode(800, 600), "Testing InputsHandler");
    78
    79         double screen_width = window.getSize().x;
    80         double screen_height = window.getSize().y;
    81
    82         testFloatEquals(
    83             screen_width,
    84             800,
    85             __FILE__,
    86             __LINE__
    87         );
    88
    89         testFloatEquals(
    90             screen_height,
    91             600,
    92             __FILE__,
    93             __LINE__
    94         );
    95
    96         unsigned long long int frame = 0;
    97         double time_since_run_s = 0;
    98
    99         while (window.isOpen()) {
    100             time_since_run_s = clock.getElapsedTime().asSeconds();
    101
    102             if (
    103                 time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
    104             ) {
    105                 while (window.pollEvent(event))
    106                     {

```



### 4.18.1 Detailed Description

Suite of tests for the [MessagesHandler](#) class.

A suite of tests for the [MessagesHandler](#) class.

### 4.18.2 Function Documentation

#### 4.18.2.1 main()

```
int main (
    int argc,
    char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif /* _WIN32 */

    printGold("\tTesting MessagesHandler");
    std::cout << std::endl;

    srand(time(NULL));
    int n_dots = 8;

    try {
        // 1. construct
        MessagesHandler messages_handler;

        // 2. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(800, 600), "Testing MessagesHandler");

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            800,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            600,
            __FILE__,
            __LINE__
        );

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

        while (window.isOpen()) {
            time_since_run_s = clock.getElapsedTime().asSeconds();

            if (
                time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
            ) {
                while (window.pollEvent(event))
                {
                    //...

                    if (event.type == sf::Event::Closed) {
                        window.close();
                    }
                }

                window.clear();
                window.display();
            }
        }
    }
}
```



## 4.19.2 Function Documentation

### 4.19.2.1 main()

```

int main (
    int argc,
    char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif /* _WIN32 */

    printGold("\tTesting HexMap");
    std::cout << std::endl;

    srand(time(NULL));
    int n_dots = 8;

    try {
        // 1. construct
        AssetsManager assets_manager;
        InputsHandler inputs_handler;
        MessagesHandler messages_handler;

        HexMap hex_map(6, &assets_manager, &inputs_handler, &messages_handler);

        // 2. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(1200, 800), "Testing AssetsManager");

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            1200,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            800,
            __FILE__,
            __LINE__
        );

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

        //...

        while (window.isOpen()) {
            time_since_run_s = clock.getElapsedTime().asSeconds();

            if (
                time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
            ) {
                while (window.pollEvent(event))
                {
                    //...

                    if (event.type == sf::Event::Closed) {
                        window.close();
                    }
                }

                //...

                window.clear();

                hex_map.draw(&window);

                window.display();

                std::cout << frame << " : " << time_since_run_s << "\r" << std::flush;
            }
        }
    }
}

```

```
111         frame++;
112     }
113 }
114 }
115
116
117 catch (...) {
118     //...
119
120     printGold(" ");
121     for (int i = 0; i < n_dots; i++) {
122         printGold(".");
123     }
124     printGold(" ");
125     printRed("FAIL");
126     std::cout << std::endl;
127     throw;
128 }
129
130
131 //...
132
133 printGold(" ");
134 for (int i = 0; i < n_dots; i++) {
135     printGold(".");
136 }
137 printGold(" ");
138 printGreen("PASS");
139 std::cout << std::endl;
140
141 return 0;
142 } /* main() */
```

# Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL <https://www.sfml-dev.org/>. 42

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL <https://www.doxygen.nl>. 41

Wikipedia. Hexagon, 2023. URL <https://en.wikipedia.org/wiki/Hexagon>. 25





# Index

- [\\_\\_assembleHexMap](#)
    - [HexMap, 19](#)
  - [\\_\\_constructKeyCodeMap](#)
    - [InputsHandler, 31](#)
  - [\\_\\_loadSoundBuffer](#)
    - [AssetsManager, 7](#)
  - [\\_\\_setUpNodeSprite](#)
    - [HexTile, 26](#)
  - [\\_\\_setUpTileSprite](#)
    - [HexTile, 26](#)
  - [~AssetsManager](#)
    - [AssetsManager, 6](#)
  - [~HexMap](#)
    - [HexMap, 19](#)
  - [~HexTile](#)
    - [HexTile, 25](#)
  - [~InputsHandler](#)
    - [InputsHandler, 31](#)
  - [~MessagesHandler](#)
    - [MessagesHandler, 38](#)
- [assets\\_manager\\_ptr](#)
  - [HexMap, 22](#)
  - [HexTile, 28](#)
- [AssetsManager, 5](#)
  - [\\_\\_loadSoundBuffer, 7](#)
  - [~AssetsManager, 6](#)
  - [AssetsManager, 6](#)
  - [clear, 8](#)
  - [current\\_track, 16](#)
  - [font\\_map, 16](#)
  - [getCurrentTrackKey, 9](#)
  - [getFont, 9](#)
  - [getSound, 10](#)
  - [getSoundBuffer, 10](#)
  - [getTexture, 11](#)
  - [getTrackStatus, 11](#)
  - [loadFont, 12](#)
  - [loadSound, 12](#)
  - [loadTexture, 13](#)
  - [loadTrack, 14](#)
  - [nextTrack, 14](#)
  - [pauseTrack, 15](#)
  - [playTrack, 15](#)
  - [previousTrack, 15](#)
  - [sound\\_map, 16](#)
  - [soundbuffer\\_map, 16](#)
  - [stopTrack, 15](#)
  - [texture\\_map, 16](#)
  - [track\\_map, 17](#)

- [clear](#)
  - [AssetsManager, 8](#)
  - [HexMap, 21](#)
- [constants.h](#)
  - [FRAMES\\_PER\\_SECOND, 40](#)
  - [SECONDS\\_PER\\_FRAME, 40](#)
- [current\\_track](#)
  - [AssetsManager, 16](#)
- [draw](#)
  - [HexMap, 21](#)
  - [HexTile, 26](#)
- [expectedErrorNotDetected](#)
  - [testing\\_utils.cpp, 55](#)
  - [testing\\_utils.h, 44](#)
- [FLOAT\\_TOLERANCE](#)
  - [testing\\_utils.h, 50](#)
- [font\\_map](#)
  - [AssetsManager, 16](#)
- [FOREST](#)
  - [HexTile.h, 52](#)
- [FOREST\\_GREEN](#)
  - [HexTile.h, 52](#)
- [frame](#)
  - [HexMap, 22](#)
  - [HexTile, 28](#)
- [FRAMES\\_PER\\_SECOND](#)
  - [constants.h, 40](#)
- [getCurrentTrackKey](#)
  - [AssetsManager, 9](#)
- [getFont](#)
  - [AssetsManager, 9](#)
- [getSound](#)
  - [AssetsManager, 10](#)
- [getSoundBuffer](#)
  - [AssetsManager, 10](#)
- [getTexture](#)
  - [AssetsManager, 11](#)
- [getTrackStatus](#)
  - [AssetsManager, 11](#)
- [header/ESC\\_core/AssetsManager.h, 39](#)
- [header/ESC\\_core/constants.h, 40](#)
- [header/ESC\\_core/doxygen\\_cite.h, 41](#)
- [header/ESC\\_core/includes.h, 41](#)
- [header/ESC\\_core/InputsHandler.h, 42](#)
- [header/ESC\\_core/MessagesHandler.h, 43](#)
- [header/ESC\\_core/testing\\_utils.h, 43](#)

- header/HexMap/HexMap.h, 50
- header/HexMap/HexTile.h, 51
- hex\_map
  - HexMap, 22
- HexMap, 17
  - \_\_assembleHexMap, 19
  - ~HexMap, 19
  - assets\_manager\_ptr, 22
  - clear, 21
  - draw, 21
  - frame, 22
  - hex\_map, 22
  - HexMap, 18
  - inputs\_handler\_ptr, 22
  - messages\_handler\_ptr, 22
  - n\_layers, 23
  - position\_x, 23
  - position\_y, 23
- HexTile, 23
  - \_\_setUpNodeSprite, 26
  - \_\_setUpTileSprite, 26
  - ~HexTile, 25
  - assets\_manager\_ptr, 28
  - draw, 26
  - frame, 28
  - HexTile, 25
  - inputs\_handler\_ptr, 28
  - major\_radius, 28
  - messages\_handler\_ptr, 28
  - minor\_radius, 28
  - node\_sprite, 29
  - position\_x, 29
  - position\_y, 29
  - setTileType, 27
  - show\_node, 29
  - tile\_sprite, 29
  - tile\_type, 29
- HexTile.h
  - FOREST, 52
  - FOREST\_GREEN, 52
  - LAKE, 52
  - LAKE\_BLUE, 52
  - MOUNTAINS, 52
  - MOUNTAINS\_GREY, 53
  - N\_TILE\_TYPES, 52
  - OCEAN, 52
  - OCEAN\_BLUE, 53
  - PLAINS, 52
  - PLAINS\_YELLOW, 53
  - TileType, 52
- inputs\_handler\_ptr
  - HexMap, 22
  - HexTile, 28
- InputsHandler, 30
  - \_\_constructKeyCodeMap, 31
  - ~InputsHandler, 31
  - InputsHandler, 31
  - key\_code\_map, 36
  - key\_press\_vec, 36
  - key\_pressed\_once\_vec, 37
  - printKeysPressed, 35
  - process, 35
  - reset, 36
- key\_code\_map
  - InputsHandler, 36
- key\_press\_vec
  - InputsHandler, 36
- key\_pressed\_once\_vec
  - InputsHandler, 37
- LAKE
  - HexTile.h, 52
- LAKE\_BLUE
  - HexTile.h, 52
- loadFont
  - AssetsManager, 12
- loadSound
  - AssetsManager, 12
- loadTexture
  - AssetsManager, 13
- loadTrack
  - AssetsManager, 14
- main
  - test\_AssetsManager.cpp, 62
  - test\_HexMap.cpp, 69
  - test\_InputsHandler.cpp, 65
  - test\_MessagesHandler.cpp, 67
- major\_radius
  - HexTile, 28
- messages\_handler\_ptr
  - HexMap, 22
  - HexTile, 28
- MessagesHandler, 37
  - ~MessagesHandler, 38
  - MessagesHandler, 37
- minor\_radius
  - HexTile, 28
- MOUNTAINS
  - HexTile.h, 52
- MOUNTAINS\_GREY
  - HexTile.h, 53
- n\_layers
  - HexMap, 23
- N\_TILE\_TYPES
  - HexTile.h, 52
- nextTrack
  - AssetsManager, 14
- node\_sprite
  - HexTile, 29
- OCEAN
  - HexTile.h, 52
- OCEAN\_BLUE
  - HexTile.h, 53

- pauseTrack
  - AssetsManager, 15
- PLAINS
  - HexTile.h, 52
- PLAINS\_YELLOW
  - HexTile.h, 53
- playTrack
  - AssetsManager, 15
- position\_x
  - HexMap, 23
  - HexTile, 29
- position\_y
  - HexMap, 23
  - HexTile, 29
- previousTrack
  - AssetsManager, 15
- printGold
  - testing\_utils.cpp, 56
  - testing\_utils.h, 45
- printGreen
  - testing\_utils.cpp, 56
  - testing\_utils.h, 45
- printKeysPressed
  - InputsHandler, 35
- printRed
  - testing\_utils.cpp, 56
  - testing\_utils.h, 45
- process
  - InputsHandler, 35
- reset
  - InputsHandler, 36
- SECONDS\_PER\_FRAME
  - constants.h, 40
- setTileType
  - HexTile, 27
- show\_node
  - HexTile, 29
- sound\_map
  - AssetsManager, 16
- soundbuffer\_map
  - AssetsManager, 16
- source/ESC\_core/AssetsManager.cpp, 53
- source/ESC\_core/InputsHandler.cpp, 54
- source/ESC\_core/MessagesHandler.cpp, 54
- source/ESC\_core/testing\_utils.cpp, 55
- source/HexMap/HexMap.cpp, 61
- source/HexMap/HexTile.cpp, 61
- stopTrack
  - AssetsManager, 15
- test/ESC\_core/test\_AssetsManager.cpp, 61
- test/ESC\_core/test\_InputsHandler.cpp, 64
- test/ESC\_core/test\_MessagesHandler.cpp, 66
- test/HexMap/test\_HexMap.cpp, 68
- test\_AssetsManager.cpp
  - main, 62
- test\_HexMap.cpp
  - main, 69
- test\_InputsHandler.cpp
  - main, 65
- test\_MessagesHandler.cpp
  - main, 67
- testFloatEquals
  - testing\_utils.cpp, 57
  - testing\_utils.h, 46
- testGreaterThan
  - testing\_utils.cpp, 57
  - testing\_utils.h, 46
- testGreaterThanOrEqualTo
  - testing\_utils.cpp, 58
  - testing\_utils.h, 47
- testing\_utils.cpp
  - expectedErrorNotDetected, 55
  - printGold, 56
  - printGreen, 56
  - printRed, 56
  - testFloatEquals, 57
  - testGreaterThan, 57
  - testGreaterThanOrEqualTo, 58
  - testLessThan, 59
  - testLessThanOrEqualTo, 59
  - testTruth, 60
- testing\_utils.h
  - expectedErrorNotDetected, 44
  - FLOAT\_TOLERANCE, 50
  - printGold, 45
  - printGreen, 45
  - printRed, 45
  - testFloatEquals, 46
  - testGreaterThan, 46
  - testGreaterThanOrEqualTo, 47
  - testLessThan, 48
  - testLessThanOrEqualTo, 48
  - testTruth, 49
- testLessThan
  - testing\_utils.cpp, 59
  - testing\_utils.h, 48
- testLessThanOrEqualTo
  - testing\_utils.cpp, 59
  - testing\_utils.h, 48
- testTruth
  - testing\_utils.cpp, 60
  - testing\_utils.h, 49
- texture\_map
  - AssetsManager, 16
- tile\_sprite
  - HexTile, 29
- tile\_type
  - HexTile, 29
- TileType
  - HexTile.h, 52
- track\_map
  - AssetsManager, 17