

HelloWorld

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
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	19
3.2.2 Constructor & Destructor Documentation	19
3.2.2.1 HexMap()	19
3.2.2.2 ~HexMap()	20
3.2.3 Member Function Documentation	20

3.2.3.1 __assembleHexMap()	20
3.2.3.2 __enforceOceanContinuity()	21
3.2.3.3 __getMajorityTileType()	21
3.2.3.4 __getNeighboursVector()	22
3.2.3.5 __getNoise()	23
3.2.3.6 __getValidMapIndexPositions()	24
3.2.3.7 __isLakeTouchingOcean()	25
3.2.3.8 __layTiles()	25
3.2.3.9 __procedurallyGenerateTileResources()	28
3.2.3.10 __procedurallyGenerateTileTypes()	28
3.2.3.11 __smoothTileTypes()	29
3.2.3.12 clear()	29
3.2.3.13 draw()	30
3.2.3.14 process()	30
3.2.3.15 reroll()	30
3.2.3.16 toggleResourceOverlay()	31
3.2.4 Member Data Documentation	31
3.2.4.1 assets_manager_ptr	31
3.2.4.2 border_tiles_vec	31
3.2.4.3 frame	31
3.2.4.4 hex_map	32
3.2.4.5 inputs_handler_ptr	32
3.2.4.6 messages_handler_ptr	32
3.2.4.7 n_layers	32
3.2.4.8 n_tiles	32
3.2.4.9 position_x	32
3.2.4.10 position_y	33
3.2.4.11 render_window_ptr	33
3.2.4.12 tile_position_x_vec	33
3.2.4.13 tile_position_y_vec	33
3.3 HexTile Class Reference	33
3.3.1 Detailed Description	35
3.3.2 Constructor & Destructor Documentation	35
3.3.2.1 HexTile()	35
3.3.2.2 ~HexTile()	36
3.3.3 Member Function Documentation	36
3.3.3.1 __setResourceText()	36
3.3.3.2 __setUpNodeSprite()	37
3.3.3.3 __setUpResourceChip()	38
3.3.3.4 __setUpTileSprite()	38
3.3.3.5 assess()	38
3.3.3.6 draw()	39

3.3.3.7 process()	39
3.3.3.8 setTileResource() [1/2]	39
3.3.3.9 setTileResource() [2/2]	40
3.3.3.10 setTileType() [1/2]	40
3.3.3.11 setTileType() [2/2]	41
3.3.3.12 toggleResourceOverlay()	42
3.3.4 Member Data Documentation	42
3.3.4.1 assets_manager_ptr	42
3.3.4.2 frame	42
3.3.4.3 inputs_handler_ptr	42
3.3.4.4 major_radius	43
3.3.4.5 messages_handler_ptr	43
3.3.4.6 minor_radius	43
3.3.4.7 node_sprite	43
3.3.4.8 position_x	43
3.3.4.9 position_y	43
3.3.4.10 render_window_ptr	44
3.3.4.11 resource_assessed	44
3.3.4.12 resource_chip_sprite	44
3.3.4.13 resource_text	44
3.3.4.14 show_node	44
3.3.4.15 show_resource	44
3.3.4.16 tile_resource	45
3.3.4.17 tile_sprite	45
3.3.4.18 tile_type	45
3.4 InputsHandler Class Reference	45
3.4.1 Detailed Description	46
3.4.2 Constructor & Destructor Documentation	46
3.4.2.1 InputsHandler()	46
3.4.2.2 ~InputsHandler()	46
3.4.3 Member Function Documentation	46
3.4.3.1 __constructKeyCodeMap()	47
3.4.3.2 printKeysPressed()	50
3.4.3.3 process()	51
3.4.3.4 reset()	51
3.4.4 Member Data Documentation	52
3.4.4.1 key_code_map	52
3.4.4.2 key_press_vec	52
3.4.4.3 key_pressed_once_vec	52
3.4.4.4 mouse_left_click	52
3.5 MessagesHandler Class Reference	53
3.5.1 Detailed Description	53

3.5.2 Constructor & Destructor Documentation	53
3.5.2.1 MessagesHandler()	53
3.5.2.2 ~MessagesHandler()	53
3.5.3 Member Function Documentation	54
3.5.3.1 process()	54
4 File Documentation	55
4.1 header/ESC_core/AssetsManager.h File Reference	55
4.1.1 Detailed Description	55
4.2 header/ESC_core/constants.h File Reference	56
4.2.1 Detailed Description	56
4.2.2 Variable Documentation	56
4.2.2.1 FRAMES_PER_SECOND	56
4.2.2.2 SECONDS_PER_FRAME	56
4.3 header/ESC_core/doxygen_cite.h File Reference	57
4.3.1 Detailed Description	57
4.4 header/ESC_core/includes.h File Reference	57
4.4.1 Detailed Description	58
4.5 header/ESC_core/InputsHandler.h File Reference	58
4.5.1 Detailed Description	58
4.6 header/ESC_core/MessagesHandler.h File Reference	59
4.6.1 Detailed Description	59
4.7 header/ESC_core/testing_utils.h File Reference	59
4.7.1 Detailed Description	60
4.7.2 Function Documentation	60
4.7.2.1 expectedErrorNotDetected()	60
4.7.2.2 printGold()	61
4.7.2.3 printGreen()	61
4.7.2.4 printRed()	61
4.7.2.5 testFloatEquals()	62
4.7.2.6 testGreaterThan()	62
4.7.2.7 testGreaterThanOrEqualTo()	63
4.7.2.8 testLessThan()	64
4.7.2.9 testLessThanOrEqualTo()	64
4.7.2.10 testTruth()	65
4.7.3 Variable Documentation	66
4.7.3.1 FLOAT_TOLERANCE	66
4.8 header/HexMap/HexMap.h File Reference	66
4.8.1 Detailed Description	66
4.9 header/HexMap/HexTile.h File Reference	67
4.9.1 Detailed Description	68
4.9.2 Enumeration Type Documentation	68

4.9.2.1 TileResource	68
4.9.2.2 TileType	69
4.9.3 Function Documentation	69
4.9.3.1 FOREST_GREEN()	69
4.9.3.2 LAKE_BLUE()	69
4.9.3.3 MOUNTAINS_GREY()	70
4.9.3.4 OCEAN_BLUE()	70
4.9.3.5 PLAINS_YELLOW()	70
4.9.4 Variable Documentation	70
4.9.4.1 tile_resource_cumulative_probabilities	70
4.9.4.2 tile_type_cumulative_probabilities	71
4.10 source/ESC_core/AssetsManager.cpp File Reference	71
4.10.1 Detailed Description	71
4.11 source/ESC_core/InputsHandler.cpp File Reference	71
4.11.1 Detailed Description	71
4.12 source/ESC_core/MessagesHandler.cpp File Reference	72
4.12.1 Detailed Description	72
4.13 source/ESC_core/testing_utils.cpp File Reference	72
4.13.1 Detailed Description	73
4.13.2 Function Documentation	73
4.13.2.1 expectedErrorNotDetected()	73
4.13.2.2 printGold()	73
4.13.2.3 printGreen()	74
4.13.2.4 printRed()	74
4.13.2.5 testFloatEquals()	74
4.13.2.6 testGreaterThan()	75
4.13.2.7 testGreaterThanOrEqualTo()	75
4.13.2.8 testLessThan()	76
4.13.2.9 testLessThanOrEqualTo()	77
4.13.2.10 testTruth()	77
4.14 source/HexMap/HexMap.cpp File Reference	78
4.14.1 Detailed Description	78
4.15 source/HexMap/HexTile.cpp File Reference	78
4.15.1 Detailed Description	79
4.16 test/ESC_core/test_AssetsManager.cpp File Reference	79
4.16.1 Detailed Description	79
4.16.2 Function Documentation	79
4.16.2.1 main()	80
4.17 test/ESC_core/test_InputsHandler.cpp File Reference	82
4.17.1 Detailed Description	82
4.17.2 Function Documentation	82
4.17.2.1 main()	83

4.18 test/ESC_core/test_MessagesHandler.cpp File Reference	84
4.18.1 Detailed Description	84
4.18.2 Function Documentation	85
4.18.2.1 main()	85
4.19 test/HexMap/test_HexMap.cpp File Reference	86
4.19.1 Detailed Description	86
4.19.2 Function Documentation	86
4.19.2.1 main()	87
Bibliography	89
Index	91

Chapter 1

Class Index

1.1 Class List

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

AssetsManager	A class which manages visual and sound assets	5
HexMap	A class which defines a hex map of hex tiles	17
HexTile	A class which defines a hex tile of the hex map	33
InputsHandler	A class which handles inputs from peripherals (i.e., keyboard and mouse)	45
MessagesHandler	A class which handles message traffic between game objects	53

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

header/ESC_core/ AssetsManager.h	
Header file for the AssetsManager class	55
header/ESC_core/ constants.h	
Header file for various constants	56
header/ESC_core/ doxygen_cite.h	
Header file which simply cites the doxygen tool	57
header/ESC_core/ includes.h	
Header file for various includes	57
header/ESC_core/ InputsHandler.h	
Header file for the InputsHandler class	58
header/ESC_core/ MessagesHandler.h	
Header file for the MessagesHandler class	59
header/ESC_core/ testing_utils.h	
Header file for various testing utilities	59
header/HexMap/ HexMap.h	
Header file for the HexMap class	66
header/HexMap/ HexTile.h	
Header file for the HexTile class	67
source/ESC_core/ AssetsManager.cpp	
Implementation file for the AssetsManager class	71
source/ESC_core/ InputsHandler.cpp	
Implementation file for the InputsHandler class	71
source/ESC_core/ MessagesHandler.cpp	
Implementation file for the MessagesHandler class	72
source/ESC_core/ testing_utils.cpp	
Implementation file for various testing utilities	72
source/HexMap/ HexMap.cpp	
Implementation file for the HexMap class	78
source/HexMap/ HexTile.cpp	
Implementation file for the HexTile class	78
test/ESC_core/ test_AssetsManager.cpp	
Suite of tests for the AssetsManager class	79
test/ESC_core/ test_InputsHandler.cpp	
Suite of tests for the InputsHandler class	82
test/ESC_core/ test_MessagesHandler.cpp	
Suite of tests for the MessagesHandler class	84
test/HexMap/ test_HexMap.cpp	
Suite of tests for the HexMap class	86

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:

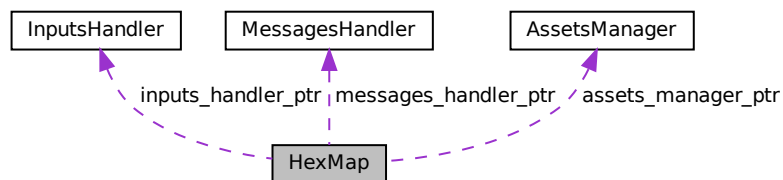
- header/ESC_core/[AssetsManager.h](#)
- source/ESC_core/[AssetsManager.cpp](#)

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](#) *, sf::RenderWindow *)
Constructor for the [HexMap](#) class.
- void [process](#) (void)
Method to process [HexMap](#). To be called once per frame;.
- void [reroll](#) (void)
Method to re-roll the hex map.
- void [toggleResourceOverlay](#) (void)
Method to toggle the hex map resource overlay.
- void [draw](#) (void)
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 [n_tiles](#)
The number of tiles 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::vector< double > [tile_position_x_vec](#)
A vector of tile x positions.
- std::vector< double > [tile_position_y_vec](#)
A vector of tile y position.
- std::vector< [HexTile](#) * > [border_tiles_vec](#)
A vector of pointers to the border tiles.
- std::map< double, std::map< double, [HexTile](#) * > > [hex_map](#)
A position-indexed, nested map of hex tiles.

Private Member Functions

- void [__layTiles](#) (void)
Helper method to lay the hex tiles down to generate the game world.
- std::vector< double > [__getNoise](#) (int, int=128)
Helper method to generate a vector of noise, with values mapped to the closed interval [0, 1]. Applies a random cosine series approach.
- void [__procedurallyGenerateTileTypes](#) (void)
Helper method to procedurally generate tile types and set tiles accordingly.
- std::vector< double > [__getValidMapIndexPositions](#) (double, double)
Helper method to translate given position into valid index position for a.
- std::vector< [HexTile](#) * > [__getNeighboursVector](#) ([HexTile](#) *)
Helper method to assemble a vector pointers to all neighbours of the given tile.
- [TileType](#) [__getMajorityTileType](#) ([HexTile](#) *)
Function to return majority tile type of a tile and its neighbours. If no clear majority, simply returns the type of the given tile.
- void [__smoothTileTypes](#) (void)
Helper method to smooth tile types using a majority rules approach.
- bool [__isLakeTouchingOcean](#) ([HexTile](#) *)
- void [__enforceOceanContinuity](#) (void)
Helper method to scan tiles and enforce ocean continuity. That is to say, if a lake tile is found to be in contact with an ocean tile, then it becomes ocean.
- void [__procedurallyGenerateTileResources](#) (void)
Helper method to procedurally generate tile resources and set tiles accordingly.
- 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.
- [sf::RenderWindow](#) * [render_window_ptr](#)
A pointer to the render window.

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,
    sf::RenderWindow * render_window_ptr )
```

Constructor for the [HexMap](#) class.

Parameters

<i>n_layers</i>	The number of layers in the HexMap .
<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.
<i>render_window_ptr</i>	Pointer to the render window.

```
797 {
798     // 1. set attributes
799     this->assets_manager_ptr = assets_manager_ptr;
800     this->inputs_handler_ptr = inputs_handler_ptr;
801     this->messages_handler_ptr = messages_handler_ptr;
802     this->render_window_ptr = render_window_ptr;
803
804     this->frame = 0;
805
806     this->n_layers = n_layers;
807     if (this->n_layers < 0) {
808         this->n_layers = 0;
809     }
810
811     this->position_x = 400;
812     this->position_y = 400;
813
814     // 2. assemble n layer hex map
```

```

815     this->__assembleHexMap();
816
817     std::cout << "HexMap constructed at " << this << std::endl;
818
819     return;
820 } /* HexMap() */

```

3.2.2.2 ~HexMap()

```

HexMap::~~HexMap (
    void )

```

Destructor for the [HexMap](#) class.

```

997 {
998     this->clear();
999
1000     std::cout << "HexMap at " << this << " destroyed" << std::endl;
1001
1002     return;
1003 } /* ~HexMap() */

```

3.2.3 Member Function Documentation

3.2.3.1 __assembleHexMap()

```

void HexMap::__assembleHexMap (
    void ) [private]

```

Helper method to assemble the hex map.

```

738 {
739     // 1. seed RNG (using milliseconds since 1 Jan 1970)
740     unsigned long long int milliseconds_since_epoch =
741         std::chrono::duration_cast<std::chrono::milliseconds>(
742             std::chrono::system_clock::now().time_since_epoch()
743         ).count();
744     srand(milliseconds_since_epoch);
745
746     // 2. lay tiles
747     this->__layTiles();
748
749     // 3. procedurally generate types
750     this->__procedurallyGenerateTileTypes();
751
752     // 4. procedurally generate resources
753     this->__procedurallyGenerateTileResources();
754
755     return;
756 } /* __assembleHexMap() */

```

3.2.3.2 __enforceOceanContinuity()

```
void HexMap::__enforceOceanContinuity (
    void ) [private]
```

Helper method to scan tiles and enforce ocean continuity. That is to say, if a lake tile is found to be in contact with an ocean tile, then it becomes ocean.

```
649 {
650     std::cout << "enforcing ocean continuity ..." << std::endl;
651
652     bool tile_changed = false;
653
654     // 1. scan tiles and enforce (where appropriate)
655     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
656     std::map<double, HexTile*>::iterator hex_map_iter_y;
657     HexTile* hex_ptr;
658     for (
659         hex_map_iter_x = this->hex_map.begin();
660         hex_map_iter_x != this->hex_map.end();
661         hex_map_iter_x++
662     ) {
663         for (
664             hex_map_iter_y = hex_map_iter_x->second.begin();
665             hex_map_iter_y != hex_map_iter_x->second.end();
666             hex_map_iter_y++
667         ) {
668             hex_ptr = hex_map_iter_y->second;
669
670             if (this->__isLakeTouchingOcean(hex_ptr)) {
671                 hex_ptr->setTileType(TileType :: OCEAN);
672                 tile_changed = true;
673             }
674         }
675     }
676
677     if (tile_changed) {
678         this->__enforceOceanContinuity();
679     }
680     else {
681         return;
682     }
683 } /* __enforceOceanContinuity() */
```

3.2.3.3 __getMajorityTileType()

```
TileType HexMap::__getMajorityTileType (
    HexTile * hex_ptr ) [private]
```

Function to return majority tile type of a tile and its neighbours. If no clear majority, simply returns the type of the given tile.

Parameters

<i>hex_ptr</i>	Pointer to the given tile.
----------------	----------------------------

Returns

The majority tile type of the tile and its neighbours. If no clear majority type, then the type of the given tile is simply returned.

```
505 {
506     // 1. init type count map
507     std::map<TileType, int> type_count_map;
508     type_count_map[hex_ptr->tile_type] = 1;
509
510     // 2. survey neighbours, count type instances
```

```

511     std::vector<HexTile*> neighbours_vec = this->__getNeighboursVector(hex_ptr);
512
513     for (size_t i = 0; i < neighbours_vec.size(); i++) {
514         if (type_count_map.count(neighbours_vec[i]->tile_type) <= 0) {
515             type_count_map[neighbours_vec[i]->tile_type] = 1;
516         }
517         else {
518             type_count_map[neighbours_vec[i]->tile_type] += 1;
519         }
520     }
521
522     // 3. find majority tile type
523     int max_count = -1 * std::numeric_limits<int>::infinity();
524     TileType majority_tile_type = hex_ptr->tile_type;
525
526     std::map<TileType, int>::iterator map_iter;
527     for (
528         map_iter = type_count_map.begin();
529         map_iter != type_count_map.end();
530         map_iter++
531     ){
532         if (map_iter->second > max_count) {
533             max_count = map_iter->second;
534             majority_tile_type = map_iter->first;
535         }
536     }
537
538     // 4. detect ties
539     for (
540         map_iter = type_count_map.begin();
541         map_iter != type_count_map.end();
542         map_iter++
543     ){
544         if (
545             map_iter->second == max_count and
546             map_iter->first != majority_tile_type
547         ) {
548             majority_tile_type = hex_ptr->tile_type;
549             break;
550         }
551     }
552
553     return majority_tile_type;
554 } /* __getMajorityTileType() */

```

3.2.3.4 __getNeighboursVector()

```

std::vector< HexTile * > HexMap::__getNeighboursVector (
    HexTile * hex_ptr ) [private]

```

Helper method to assemble a vector pointers to all neighbours of the given tile.

Parameters

<i>hex_ptr</i>	A pointer to the given tile.
----------------	------------------------------

Returns

A vector of pointers to all neighbours of the given tile.

```

447 {
448     std::vector<HexTile*> neighbours_vec;
449
450     // 1. build potential neighbour positions
451     std::vector<double> potential_neighbour_x_vec(6, 0);
452     std::vector<double> potential_neighbour_y_vec(6, 0);
453
454     for (int i = 0; i < 6; i++) {
455         potential_neighbour_x_vec[i] = hex_ptr->position_x +
456             2 * hex_ptr->minor_radius * cos((60 * i) * (M_PI / 180));
457
458         potential_neighbour_y_vec[i] = hex_ptr->position_y +

```

```

459         2 * hex_ptr->minor_radius * sin((60 * i) * (M_PI / 180));
460     }
461
462     // 2. populate neighbours vector
463     std::vector<double> map_index_positions;
464     double potential_x = 0;
465     double potential_y = 0;
466
467     for (int i = 0; i < 6; i++) {
468         potential_x = potential_neighbour_x_vec[i];
469         potential_y = potential_neighbour_y_vec[i];
470
471         map_index_positions = this->__getValidMapIndexPositions(
472             potential_x,
473             potential_y
474         );
475
476         if (not (map_index_positions[0] == -1)) {
477             neighbours_vec.push_back(
478                 this->hex_map[map_index_positions[0]][map_index_positions[1]]
479             );
480         }
481     }
482
483     return neighbours_vec;
484 } /* __getNeighbourVector() */

```

3.2.3.5 __getNoise()

```

std::vector< double > HexMap::__getNoise (
    int n_elements,
    int n_components = 128 ) [private]

```

Helper method to generate a vector of noise, with values mapped to the closed interval [0, 1]. Applies a random cosine series approach.

Parameters

<i>n_elements</i>	The number of elements in the generated noise vector.
<i>n_components</i>	The number of components to use in the random cosine series. Defaults to 64.

Returns

A vector of noise, with values mapped to the closed interval [0, 1].

```

227 {
228     // 1. generate random amplitude, wave number, direction, and phase vectors
229     std::vector<double> random_amplitude_vec(n_components, 0);
230     std::vector<double> random_wave_number_vec(n_components, 0);
231     std::vector<double> random_frequency_vec(n_components, 0);
232     std::vector<double> random_direction_vec(n_components, 0);
233     std::vector<double> random_phase_vec(n_components, 0);
234
235     for (int i = 0; i < n_components; i++) {
236         random_amplitude_vec[i] = 10 * ((double)rand() / RAND_MAX);
237
238         random_wave_number_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
239
240         random_frequency_vec[i] = ((double)rand() / RAND_MAX);
241
242         random_direction_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
243
244         random_phase_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
245     }
246
247     // 2. generate noise vec
248     double amp = 0;
249     double wave_no = 0;
250     double freq = 0;
251     double dir = 0;

```

```

252     double phase = 0;
253
254     double x = 0;
255     double y = 0;
256     double t = time(NULL);
257
258     double max_noise = -1 * std::numeric_limits<double>::infinity();
259     double min_noise = std::numeric_limits<double>::infinity();
260
261     double noise = 0;
262     std::vector<double> noise_vec(n_elements, 0);
263
264     for (int i = 0; i < n_elements; i++) {
265         x = this->tile_position_x_vec[i] - this->position_x;
266         y = this->tile_position_y_vec[i] - this->position_y;
267
268         for (int j = 0; j < n_components; j++) {
269             amp = random_amplitude_vec[j];
270             wave_no = random_wave_number_vec[j];
271             freq = random_frequency_vec[j];
272             dir = random_direction_vec[j];
273             phase = random_phase_vec[j];
274
275             noise += (amp / (j + 1)) * cos(
276                 wave_no * (j + 1) * (x * sin(dir) + y * cos(dir)) +
277                 2 * M_PI * (j + 1) * freq * t +
278                 phase
279             );
280         }
281
282         noise_vec[i] = noise;
283
284         if (noise > max_noise) {
285             max_noise = noise;
286         }
287
288         else if (noise < min_noise) {
289             min_noise = noise;
290         }
291
292         noise = 0;
293     }
294
295     // 3. normalize noise vec
296     for (int i = 0; i < n_elements; i++) {
297         noise_vec[i] = (noise_vec[i] - min_noise) / (max_noise - min_noise);
298
299         if (noise_vec[i] < 0) {
300             noise_vec[i] = 0;
301         }
302         else if (noise_vec[i] > 1) {
303             noise_vec[i] = 1;
304         }
305     }
306
307     return noise_vec;
308 } /* __getNoise() */

```

3.2.3.6 __getValidMapIndexPositions()

```

std::vector< double > HexMap::__getValidMapIndexPositions (
    double potential_x,
    double potential_y ) [private]

```

Helper method to translate given position into valid index position for a.

Parameters

<i>potential_x</i>	The potential x position of the tile.
<i>potential_y</i>	The potential y position of the tile.

Returns

A vector of positions, either valid for indexing into the hex map, or sentinel values (-1) if invalid.

```

393 {
394     std::vector<double> map_index_positions = {-1, -1};
395
396     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
397     std::map<double, HexTile*>::iterator hex_map_iter_y;
398     HexTile* hex_ptr;
399
400     double distance = 0;
401
402     for (
403         hex_map_iter_x = this->hex_map.begin();
404         hex_map_iter_x != this->hex_map.end();
405         hex_map_iter_x++
406     ) {
407         for (
408             hex_map_iter_y = hex_map_iter_x->second.begin();
409             hex_map_iter_y != hex_map_iter_x->second.end();
410             hex_map_iter_y++
411         ) {
412             hex_ptr = hex_map_iter_y->second;
413
414             distance = sqrt(
415                 pow(hex_ptr->position_x - potential_x, 2) +
416                 pow(hex_ptr->position_y - potential_y, 2)
417             );
418
419             if (distance <= hex_ptr->minor_radius / 4) {
420                 map_index_positions = {hex_ptr->position_x, hex_ptr->position_y};
421                 return map_index_positions;
422             }
423         }
424     }
425
426     return map_index_positions;
427 } /* __isInHexMap() */

```

3.2.3.7 __isLakeTouchingOcean()

```

bool HexMap::__isLakeTouchingOcean (
    HexTile * hex_ptr ) [private]

616 {
617     // 1. if not lake tile, return
618     if (not (hex_ptr->tile_type == TileType :: LAKE)) {
619         return false;
620     }
621
622     // 2. scan neighbours for ocean tiles
623     std::vector<HexTile*> neighbours_vec = this->__getNeighboursVector(hex_ptr);
624
625     for (size_t i = 0; i < neighbours_vec.size(); i++) {
626         if (neighbours_vec[i]->tile_type == TileType :: OCEAN) {
627             return true;
628         }
629     }
630
631     return false;
632 } /* __isLakeTouchingOcean() */

```

3.2.3.8 __layTiles()

```

void HexMap::__layTiles (
    void ) [private]

```

Helper method to lay the hex tiles down to generate the game world.

```

34 {
35     this->n_tiles = 0;

```

```

36
37 // 1. add origin tile
38 HexTile* hex_ptr = new HexTile(
39     this->position_x,
40     this->position_y,
41     this->assets_manager_ptr,
42     this->inputs_handler_ptr,
43     this->messages_handler_ptr,
44     this->render_window_ptr
45 );
46
47 this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
48 this->tile_position_x_vec.push_back(hex_ptr->position_x);
49 this->tile_position_y_vec.push_back(hex_ptr->position_y);
50 this->n_tiles++;
51
52
53 // 2. fill out first row (reflect across origin tile)
54 for (int i = 0; i < this->n_layers; i++) {
55     hex_ptr = new HexTile(
56         this->position_x + 2 * (i + 1) * hex_ptr->minor_radius,
57         this->position_y,
58         this->assets_manager_ptr,
59         this->inputs_handler_ptr,
60         this->messages_handler_ptr,
61         this->render_window_ptr
62     );
63
64     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
65     this->tile_position_x_vec.push_back(hex_ptr->position_x);
66     this->tile_position_y_vec.push_back(hex_ptr->position_y);
67     this->n_tiles++;
68
69     if (i == this->n_layers - 1) {
70         this->border_tiles_vec.push_back(hex_ptr);
71     }
72
73     hex_ptr = new HexTile(
74         this->position_x - 2 * (i + 1) * hex_ptr->minor_radius,
75         this->position_y,
76         this->assets_manager_ptr,
77         this->inputs_handler_ptr,
78         this->messages_handler_ptr,
79         this->render_window_ptr
80     );
81
82     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
83     this->tile_position_x_vec.push_back(hex_ptr->position_x);
84     this->tile_position_y_vec.push_back(hex_ptr->position_y);
85     this->n_tiles++;
86
87     if (i == this->n_layers - 1) {
88         this->border_tiles_vec.push_back(hex_ptr);
89     }
90 }
91
92
93 // 3. fill out subsequent rows (reflect across first row)
94 HexTile* first_row_left_tile = hex_ptr;
95
96 int offset_count = 1;
97
98 double x_offset = 0;
99 double y_offset = 0;
100
101 for (
102     int row_width = 2 * this->n_layers;
103     row_width > this->n_layers;
104     row_width--
105 ) {
106     // 3.1. upper row
107     x_offset = first_row_left_tile->position_x +
108         2 * offset_count * first_row_left_tile->minor_radius *
109         cos(60 * (M_PI / 180));
110
111     y_offset = first_row_left_tile->position_y -
112         2 * offset_count * first_row_left_tile->minor_radius *
113         sin(60 * (M_PI / 180));
114
115     hex_ptr = new HexTile(
116         x_offset,
117         y_offset,
118         this->assets_manager_ptr,
119         this->inputs_handler_ptr,
120         this->messages_handler_ptr,
121         this->render_window_ptr
122     );

```

```

123
124     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
125     this->tile_position_x_vec.push_back(hex_ptr->position_x);
126     this->tile_position_y_vec.push_back(hex_ptr->position_y);
127     this->n_tiles++;
128
129     this->border_tiles_vec.push_back(hex_ptr);
130
131     for (int i = 1; i < row_width; i++) {
132         x_offset += 2 * first_row_left_tile->minor_radius;
133
134         hex_ptr = new HexTile(
135             x_offset,
136             y_offset,
137             this->assets_manager_ptr,
138             this->inputs_handler_ptr,
139             this->messages_handler_ptr,
140             this->render_window_ptr
141         );
142
143         this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
144         this->tile_position_x_vec.push_back(hex_ptr->position_x);
145         this->tile_position_y_vec.push_back(hex_ptr->position_y);
146         this->n_tiles++;
147
148         if (row_width == this->n_layers + 1 or i == row_width - 1) {
149             this->border_tiles_vec.push_back(hex_ptr);
150         }
151     }
152
153     // 3.2. lower row
154     x_offset = first_row_left_tile->position_x +
155         2 * offset_count * first_row_left_tile->minor_radius *
156         cos(60 * (M_PI / 180));
157
158     y_offset = first_row_left_tile->position_y +
159         2 * offset_count * first_row_left_tile->minor_radius *
160         sin(60 * (M_PI / 180));
161
162     hex_ptr = new HexTile(
163         x_offset,
164         y_offset,
165         this->assets_manager_ptr,
166         this->inputs_handler_ptr,
167         this->messages_handler_ptr,
168         this->render_window_ptr
169     );
170
171     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
172     this->tile_position_x_vec.push_back(hex_ptr->position_x);
173     this->tile_position_y_vec.push_back(hex_ptr->position_y);
174     this->n_tiles++;
175
176     this->border_tiles_vec.push_back(hex_ptr);
177
178     for (int i = 1; i < row_width; i++) {
179         x_offset += 2 * first_row_left_tile->minor_radius;
180
181         hex_ptr = new HexTile(
182             x_offset,
183             y_offset,
184             this->assets_manager_ptr,
185             this->inputs_handler_ptr,
186             this->messages_handler_ptr,
187             this->render_window_ptr
188         );
189
190         this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
191         this->tile_position_x_vec.push_back(hex_ptr->position_x);
192         this->tile_position_y_vec.push_back(hex_ptr->position_y);
193         this->n_tiles++;
194
195         if (row_width == this->n_layers + 1 or i == row_width - 1) {
196             this->border_tiles_vec.push_back(hex_ptr);
197         }
198     }
199
200     offset_count++;
201 }
202
203 return;
204 } /* __layTiles() */

```

3.2.3.9 __procedurallyGenerateTileResources()

```
void HexMap::__procedurallyGenerateTileResources (
    void ) [private]
```

Helper method to procedurally generate tile resources and set tiles accordingly.

```
698 {
699     // 1. get random cosine series noise vec
700     std::vector<double> noise_vec = this->__getNoise(this->n_tiles);
701
702     // 2. set tile resources based on random cosine series noise
703     int noise_idx = 0;
704
705     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
706     std::map<double, HexTile*>::iterator hex_map_iter_y;
707     for (
708         hex_map_iter_x = this->hex_map.begin();
709         hex_map_iter_x != this->hex_map.end();
710         hex_map_iter_x++
711     ) {
712         for (
713             hex_map_iter_y = hex_map_iter_x->second.begin();
714             hex_map_iter_y != hex_map_iter_x->second.end();
715             hex_map_iter_y++
716         ) {
717             hex_map_iter_y->second->setTileResource(noise_vec[noise_idx]);
718             noise_idx++;
719         }
720     }
721
722     return;
723 } /* __procedurallyGenerateTileResources() */
```

3.2.3.10 __procedurallyGenerateTileTypes()

```
void HexMap::__procedurallyGenerateTileTypes (
    void ) [private]
```

Helper method to procedurally generate tile types and set tiles accordingly.

```
323 {
324     // 1. get random cosine series noise vec
325     std::vector<double> noise_vec = this->__getNoise(this->n_tiles);
326
327     // 2. set initial tile types based on either random cosine series noise or white
328     //     noise (decided by coin toss)
329     int noise_idx = 0;
330
331     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
332     std::map<double, HexTile*>::iterator hex_map_iter_y;
333     for (
334         hex_map_iter_x = this->hex_map.begin();
335         hex_map_iter_x != this->hex_map.end();
336         hex_map_iter_x++
337     ) {
338         for (
339             hex_map_iter_y = hex_map_iter_x->second.begin();
340             hex_map_iter_y != hex_map_iter_x->second.end();
341             hex_map_iter_y++
342         ) {
343             if ((double)rand() / RAND_MAX > 0.5) {
344                 hex_map_iter_y->second->setTileType(noise_vec[noise_idx]);
345             }
346             else {
347                 hex_map_iter_y->second->setTileType((double)rand() / RAND_MAX);
348             }
349             noise_idx++;
350         }
351     }
352
353     // 3. smooth tile types (majority rules)
354     this->__smoothTileTypes();
355
356     // 4. set border tile type to ocean
357     for (size_t i = 0; i < this->border_tiles_vec.size(); i++) {
358         this->border_tiles_vec[i]->setTileType(TileType :: OCEAN);
359     }
360 }
```

```

359     }
360
361     // 5. enforce ocean continuity (i.e. all lake tiles touching ocean become ocean)
362     this->__enforceOceanContinuity();
363
364     return;
365 } /* __procedurallyGenerateTileTypes() */

```

3.2.3.11 __smoothTileTypes()

```

void HexMap::__smoothTileTypes (
    void ) [private]

```

Helper method to smooth tile types using a majority rules approach.

```

569 {
570     std::cout << "smoothing ..." << std::endl;
571
572     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
573     std::map<double, HexTile*>::iterator hex_map_iter_y;
574     HexTile* hex_ptr;
575     TileType majority_tile_type;
576
577     for (
578         hex_map_iter_x = this->hex_map.begin();
579         hex_map_iter_x != this->hex_map.end();
580         hex_map_iter_x++
581     ) {
582         for (
583             hex_map_iter_y = hex_map_iter_x->second.begin();
584             hex_map_iter_y != hex_map_iter_x->second.end();
585             hex_map_iter_y++
586         ) {
587             hex_ptr = hex_map_iter_y->second;
588             majority_tile_type = this->__getMajorityTileType(hex_ptr);
589
590             if (majority_tile_type != hex_ptr->tile_type) {
591                 hex_ptr->setTileType(majority_tile_type);
592             }
593         }
594     }
595
596     return;
597 } /* __smoothTileTypes() */

```

3.2.3.12 clear()

```

void HexMap::clear (
    void )

```

Method to clear the hex map.

```

959 {
960     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
961     std::map<double, HexTile*>::iterator hex_map_iter_y;
962     for (
963         hex_map_iter_x = this->hex_map.begin();
964         hex_map_iter_x != this->hex_map.end();
965         hex_map_iter_x++
966     ) {
967         for (
968             hex_map_iter_y = hex_map_iter_x->second.begin();
969             hex_map_iter_y != hex_map_iter_x->second.end();
970             hex_map_iter_y++
971         ) {
972             delete hex_map_iter_y->second;
973         }
974     }
975     this->hex_map.clear();
976
977     this->tile_position_x_vec.clear();
978     this->tile_position_y_vec.clear();
979     this->border_tiles_vec.clear();
980
981     return;
982 } /* clear() */

```

3.2.3.13 draw()

```
void HexMap::draw (
    void )
```

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

```
925 {
926     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
927     std::map<double, HexTile*>::iterator hex_map_iter_y;
928     for (
929         hex_map_iter_x = this->hex_map.begin();
930         hex_map_iter_x != this->hex_map.end();
931         hex_map_iter_x++
932     ) {
933         for (
934             hex_map_iter_y = hex_map_iter_x->second.begin();
935             hex_map_iter_y != hex_map_iter_x->second.end();
936             hex_map_iter_y++
937         ) {
938             hex_map_iter_y->second->draw();
939         }
940     }
941
942     this->frame++;
943     return;
944 } /* draw() */
```

3.2.3.14 process()

```
void HexMap::process (
    void )
```

Method to process [HexMap](#). To be called once per frame;.

```
835 {
836     // 1. handle inputs
837     //...
838
839     // 2. process tiles
840     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
841     std::map<double, HexTile*>::iterator hex_map_iter_y;
842     for (
843         hex_map_iter_x = this->hex_map.begin();
844         hex_map_iter_x != this->hex_map.end();
845         hex_map_iter_x++
846     ) {
847         for (
848             hex_map_iter_y = hex_map_iter_x->second.begin();
849             hex_map_iter_y != hex_map_iter_x->second.end();
850             hex_map_iter_y++
851         ) {
852             hex_map_iter_y->second->process();
853         }
854     }
855
856     return;
857 } /* process() */
```

3.2.3.15 reroll()

```
void HexMap::reroll (
    void )
```

Method to re-roll the hex map.

```
872 {
873     this->clear();
874     this->__assembleHexMap();
875
876     return;
877 } /* reroll() */
```

3.2.3.16 toggleResourceOverlay()

```
void HexMap::toggleResourceOverlay (
    void )
```

Method to toggle the hex map resource overlay.

```
892 {
893     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
894     std::map<double, HexTile*>::iterator hex_map_iter_y;
895     for (
896         hex_map_iter_x = this->hex_map.begin();
897         hex_map_iter_x != this->hex_map.end();
898         hex_map_iter_x++
899     ) {
900         for (
901             hex_map_iter_y = hex_map_iter_x->second.begin();
902             hex_map_iter_y != hex_map_iter_x->second.end();
903             hex_map_iter_y++
904         ) {
905             hex_map_iter_y->second->toggleResourceOverlay();
906         }
907     }
908     return;
909 }
910 } /* toggleResourceOverlay() */
```

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 border_tiles_vec

```
std::vector<HexTile*> HexMap::border_tiles_vec
```

A vector of pointers to the border tiles.

3.2.4.3 frame

```
int HexMap::frame
```

The current frame of this object.

3.2.4.4 hex_map

```
std::map<double, std::map<double, HexTile*> > HexMap::hex_map
```

A position-indexed, nested map of hex tiles.

3.2.4.5 inputs_handler_ptr

```
InputsHandler* HexMap::inputs_handler_ptr [private]
```

A pointer to the inputs handler.

3.2.4.6 messages_handler_ptr

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

A pointer to the messages handler.

3.2.4.7 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

3.2.4.8 n_tiles

```
int HexMap::n_tiles
```

The number of tiles in the hex map.

3.2.4.9 position_x

```
double HexMap::position_x
```

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

3.2.4.10 position_y

```
double HexMap::position_y
```

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

3.2.4.11 render_window_ptr

```
sf::RenderWindow* HexMap::render_window_ptr [private]
```

A pointer to the render window.

3.2.4.12 tile_position_x_vec

```
std::vector<double> HexMap::tile_position_x_vec
```

A vector of tile x positions.

3.2.4.13 tile_position_y_vec

```
std::vector<double> HexMap::tile_position_y_vec
```

A vector of tile y position.

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

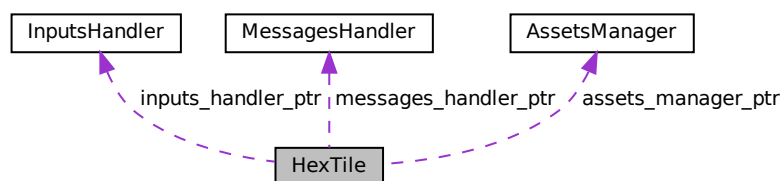
- header/HexMap/[HexMap.h](#)
- source/HexMap/[HexMap.cpp](#)

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](#) *, [sf::RenderWindow](#) *)
Constructor for the [HexTile](#) class.
- void [setTileType](#) ([TileType](#))
Method to set the tile type (by enum value).
- void [setTileType](#) (double)
Method to set the tile type (by numeric input).
- void [setTileResource](#) ([TileResource](#))
Method to set the tile resource (by enum value).
- void [setTileResource](#) (double)
Method to set the tile resource (by numeric input).
- void [toggleResourceOverlay](#) (void)
Method to toggle the tile resource overlay.
- void [assess](#) (void)
Method to assess the tile's resource.
- void [process](#) (void)
Method to process [HexTile](#). To be called once per frame.
- void [draw](#) (void)
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](#)
- [TileResource](#) [tile_resource](#)
- bool [show_node](#)
A boolean which indicates whether or not to show the tile node.
- bool [show_resource](#)
A boolean which indicates whether or not to show resource value.
- bool [resource_assessed](#)
A boolean which indicates whether or not the resource has been assessed.
- 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.
- [sf::CircleShape](#) [resource_chip_sprite](#)
A circle shape which represents a resource chip.
- [sf::Text](#) [resource_text](#)
A text representation of the resource.

Private Member Functions

- void [__setUpNodeSprite](#) (void)
Helper method to set up node sprite.
- void [__setUpTileSprite](#) (void)
Helper method to set up tile sprite.
- void [__setUpResourceChip](#) (void)
Helper method to set up resource chip sprite.
- void [__setResourceText](#) (void)
Helper method to set up resource text.

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.
- sf::RenderWindow * [render_window_ptr](#)
A pointer to the render window.

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,
    sf::RenderWindow * render_window_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.
<i>render_window_ptr</i>	Pointer to the render window.

```

230 {
231     // 1. set attributes
232     this->assets_manager_ptr = assets_manager_ptr;
233     this->inputs_handler_ptr = inputs_handler_ptr;
234     this->messages_handler_ptr = messages_handler_ptr;
235     this->render_window_ptr = render_window_ptr;
236
237     this->show_node = false;
238     this->show_resource = false;
239     this->resource_assessed = true;
240
241     this->frame = 0;
242
243     this->position_x = position_x;
244     this->position_y = position_y;
245
246     this->major_radius = 32;
247     this->minor_radius = (sqrt(3) / 2) * this->major_radius;
248
249     // 2. set up and position drawable attributes
250     this->__setUpNodeSprite();
251     this->__setUpTileSprite();
252     this->__setUpResourceChip();
253     this->__setUpResourceText();
254
255     // 3. set tile type and resource (default to forest and average)
256     this->setTileType(TileType :: FOREST);
257     this->setTileResource(TileResource :: AVERAGE);
258
259     std::cout << "HexTile constructed at " << this << std::endl;
260
261     return;
262 } /* HexTile() */

```

3.3.2.2 ~HexTile()

```

HexTile::~HexTile (
    void )

```

Destructor for the [HexTile](#) class.

```

554 {
555     std::cout << "HexTile at " << this << " destroyed" << std::endl;
556
557     return;
558 } /* ~HexTile() */

```

3.3.3 Member Function Documentation

3.3.3.1 __setResourceText()

```

void HexTile::__setResourceText (
    void ) [private]

```

Helper method to set up resource text.

```

123 {
124     this->resource_text.setFont(*(assets_manager_ptr->getFont("DroidSansMono")));
125
126     switch (this->tile_resource) {
127         case (TileResource :: POOR): {
128             this->resource_text.setString("-2");
129
130             break;
131         }
132
133         case (TileResource :: BELOW_AVERAGE): {

```

```

134         this->resource_text.setString("-1");
135
136         break;
137     }
138
139     case (TileResource :: AVERAGE): {
140         this->resource_text.setString("0");
141
142         break;
143     }
144
145     case (TileResource :: ABOVE_AVERAGE): {
146         this->resource_text.setString("+1");
147
148         break;
149     }
150
151     case (TileResource :: GOOD): {
152         this->resource_text.setString("+2");
153
154         break;
155     }
156
157     default: {
158         this->resource_text.setString("?");
159
160         break;
161     }
162 }
163
164 if (not this->resource_assessed) {
165     this->resource_text.setString("?");
166 }
167
168 this->resource_text.setCharacterSize(16);
169
170 this->resource_text.setOrigin(
171     this->resource_text.getLocalBounds().width / 2,
172     this->resource_text.getLocalBounds().height / 2
173 );
174
175 this->resource_text.setFillColor(sf::Color(0, 0, 0, 255));
176
177 this->resource_text.setPosition(
178     this->position_x,
179     this->position_y - 4
180 );
181
182 return;
183 } /* __setResourceText() */

```

3.3.3.2 __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() */

```

3.3.3.3 __setUpResourceChip()

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

Helper method to set up resource chip sprite.

```
96 {
97     this->resource_chip_sprite.setRadius(2 * this->minor_radius / 3);
98
99     this->resource_chip_sprite.setOrigin(
100         this->resource_chip_sprite.getLocalBounds().width / 2,
101         this->resource_chip_sprite.getLocalBounds().height / 2
102     );
103
104     this->resource_chip_sprite.setPosition(this->position_x, this->position_y);
105
106     this->resource_chip_sprite.setFillColor(sf::Color(175, 175, 175, 175));
107
108     return;
109 } /* __setUpResourceChip() */
```

3.3.3.4 __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(1);
78     this->tile_sprite.setOutlineColor(sf::Color(175, 175, 175, 255));
79
80     return;
81 } /* __setUpTileSprite() */
```

3.3.3.5 assess()

```
void HexTile::assess (
    void )
```

Method to assess the tile's resource.

```
483 {
484     this->resource_assessed = true;
485
486     return;
487 } /* assess() */
```

3.3.3.6 draw()

```
void HexTile::draw (
    void )
```

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

```
522 {
523     // 1. draw hex
524     this->render_window_ptr->draw(this->tile_sprite);
525
526     // 2. draw node
527     if (this->show_node) {
528         this->render_window_ptr->draw(this->node_sprite);
529     }
530
531     // 3. draw resource
532     if (this->show_resource) {
533         this->render_window_ptr->draw(this->resource_chip_sprite);
534         this->render_window_ptr->draw(this->resource_text);
535     }
536
537     this->frame++;
538     return;
539 } /* draw() */
```

3.3.3.7 process()

```
void HexTile::process (
    void )
```

Method to process [HexTile](#). To be called once per frame.

```
502 {
503     //...
504
505     return;
506 } /* process() */
```

3.3.3.8 setTileResource() [1/2]

```
void HexTile::setTileResource (
    double input_value )
```

Method to set the tile resource (by numeric input).

Parameters

<i>input_value</i>	A numerical input in the closed interval [0, 1].
--------------------	--

```
408 {
409     // 1. check input
410     if (input_value < 0 or input_value > 1) {
411         std::string error_str = "ERROR HexTile::setTileResource() given input value is ";
412         error_str += "not in the closed interval [0, 1]";
413
414         #ifdef _WIN32
415             std::cout << error_str << std::endl;
416         #endif /* _WIN32 */
417
418         throw std::runtime_error(error_str);
419     }
420 }
```

```

421 // 2. convert input value to tile resource
422 TileResource tile_resource;
423
424 if (input_value <= tile_resource_cumulative_probabilities[0]) {
425     tile_resource = TileResource :: POOR;
426 }
427 else if (input_value <= tile_resource_cumulative_probabilities[1]) {
428     tile_resource = TileResource :: BELOW_AVERAGE;
429 }
430 else if (input_value <= tile_resource_cumulative_probabilities[2]) {
431     tile_resource = TileResource :: AVERAGE;
432 }
433 else if (input_value <= tile_resource_cumulative_probabilities[3]) {
434     tile_resource = TileResource :: ABOVE_AVERAGE;
435 }
436 else {
437     tile_resource = TileResource :: GOOD;
438 }
439
440 // 3. call alternate method
441 this->setTileResource(tile_resource);
442
443 return;
444 } /* setTileResource(double) */

```

3.3.3.9 setTileResource() [2/2]

```

void HexTile::setTileResource (
    TileResource tile_resource )

```

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

Parameters

<i>tile_resource</i>	The resource (TileResource) value to attribute to the tile.
----------------------	---

```

386 {
387     this->tile_resource = tile_resource;
388     this->__setResourceText();
389
390     return;
391 } /* setTileResource(TileResource) */

```

3.3.3.10 setTileType() [1/2]

```

void HexTile::setTileType (
    double input_value )

```

Method to set the tile type (by numeric input).

Parameters

<i>input_value</i>	A numerical input in the closed interval [0, 1].
--------------------	--

```

336 {
337     // 1. check input
338     if (input_value < 0 or input_value > 1) {
339         std::string error_str = "ERROR HexTile::setTileType() given input value is ";
340         error_str += "not in the closed interval [0, 1]";
341
342         #ifdef _WIN32
343             std::cout << error_str << std::endl;

```



```

344         #endif /* _WIN32 */
345
346         throw std::runtime_error(error_str);
347     }
348
349     // 2. convert input value to tile type
350     TileType tile_type;
351
352     if (input_value <= tile_type_cumulative_probabilities[0]) {
353         tile_type = TileType :: LAKE;
354     }
355     else if (input_value <= tile_type_cumulative_probabilities[1]) {
356         tile_type = TileType :: PLAINS;
357     }
358     else if (input_value <= tile_type_cumulative_probabilities[2]) {
359         tile_type = TileType :: FOREST;
360     }
361     else {
362         tile_type = TileType :: MOUNTAINS;
363     }
364
365     // 3. call alternate method
366     this->setTileType(tile_type);
367
368     return;
369 } /* setTileType(double) */

```

3.3.3.11 setTileType() [2/2]

```

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.
------------------	---

```

277 {
278     this->tile_type = tile_type;
279
280     switch (this->tile_type) {
281         case (TileType :: FOREST): {
282             this->tile_sprite.setFillColor(FOREST_GREEN);
283
284             break;
285         }
286
287         case (TileType :: LAKE): {
288             this->tile_sprite.setFillColor(LAKE_BLUE);
289
290             break;
291         }
292
293         case (TileType :: MOUNTAINS): {
294             this->tile_sprite.setFillColor(MOUNTAINS_GREY);
295
296             break;
297         }
298
299         case (TileType :: OCEAN): {
300             this->tile_sprite.setFillColor(OCEAN_BLUE);
301
302             break;
303         }
304
305         case (TileType :: PLAINS): {
306             this->tile_sprite.setFillColor(PLAINS_YELLOW);
307
308             break;
309         }
310
311         default: {
312             // do nothing!
313

```

```
314         break;
315     }
316 }
317
318 return;
319 } /* setTileType(TileType) */
```

3.3.3.12 toggleResourceOverlay()

```
void HexTile::toggleResourceOverlay (
    void )
```

Method to toggle the tile resource overlay.

```
459 {
460     if (this->show_resource) {
461         this->show_resource = false;
462     }
463     else {
464         this->show_resource = true;
465     }
466
467     return;
468 } /* toggleResourceOverlay() */
```

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 render_window_ptr

```
sf::RenderWindow* HexTile::render_window_ptr [private]
```

A pointer to the render window.

3.3.4.11 resource_assessed

```
bool HexTile::resource_assessed
```

A boolean which indicates whether or not the resource has been assessed.

3.3.4.12 resource_chip_sprite

```
sf::CircleShape HexTile::resource_chip_sprite
```

A circle shape which represents a resource chip.

3.3.4.13 resource_text

```
sf::Text HexTile::resource_text
```

A text representation of the resource.

3.3.4.14 show_node

```
bool HexTile::show_node
```

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

3.3.4.15 show_resource

```
bool HexTile::show_resource
```

A boolean which indicates whether or not to show resource value.

3.3.4.16 tile_resource

`TileResource` HexTile::tile_resource

3.3.4.17 tile_sprite

`sf::ConvexShape` HexTile::tile_sprite

A convex shape which represents the tile.

3.3.4.18 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

- bool [mouse_left_click](#)
A boolean which indicates if the mouse left button has been clicked.
- 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.

```
518 {
519     std::cout << "InputsHandler at " << this << " destroyed" << std::endl;
520
521     return;
522 } /* ~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.

```

465 {
466     std::string print_str = "";
467
468     for (size_t i = 0; i < this->key_press_vec.size(); i++) {
469         if (this->key_press_vec[i]) {
470             print_str += this->key_code_map[sf::Keyboard::Key(i)];
471             print_str += ", ";
472         }
473     }
474
475     if (not print_str.empty()) {
476         std::cout << "Keys pressed: " << print_str << std::endl;
477     }
478
479     return;
480 } /* 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             break;
414         }
415         case (sf::Event::KeyReleased): {
416             this->key_pressed_once_vec[event_ptr->key.code] = false;
417             this->key_press_vec[event_ptr->key.code] = false;
418             break;
419         }
420         case (sf::Event::MouseButtonPressed): {
421             if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
422                 this->mouse_left_click = true;
423                 std::cout << "left click" << std::endl;
424             }
425             break;
426         }
427         case (sf::Event::MouseButtonReleased): {
428             this->mouse_left_click = false;
429             break;
430         }
431         default: {
432             // do nothing!
433             break;
434         }
435     }
436
437     return;
438 } /* process() */

```

3.4.3.4 reset()

```

void InputsHandler::reset (
    void )

```

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

```

495 {
496     this->mouse_left_click = false;
497
498     for (size_t i = 0; i < this->key_press_vec.size(); i++) {
499         this->key_pressed_once_vec[i] = false;
500     }
501
502     return;
503 } /* 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.

3.4.4.4 mouse_left_click

```
bool InputsHandler::mouse_left_click
```

A boolean which indicates if the mouse left button has been clicked.

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.
- void [process](#) (void)
Method to process messages. To be called once per frame.
- [~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.

```
86 {  
87     std::cout << "MessagesHandler at " << this << " destroyed" << std::endl;  
88  
89     return;  
90 } /* ~MessagesHandler() */
```

3.5.3 Member Function Documentation

3.5.3.1 process()

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

Method to process messages. To be called once per frame.

```
67 {  
68     //...  
69  
70     return;  
71 } /* process() */
```

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
    AssetsManagerH[header/ESC_core/AssetsManager.h]
    HexTileH[header/HexMap/HexTile.h]
    HexMapH[header/HexMap/HexMap.h]
    AssetsManagerCpp[source/ESC_core/AssetsManager.cpp]
    TestAssetsManagerCpp[test/ESC_core/test_AssetsManager.cpp]
    HexMapCpp[source/HexMap/HexMap.cpp]
    TestHexMapCpp[test/HexMap/test_HexMap.cpp]

    AssetsManagerH --> HexTileH
    AssetsManagerH --> AssetsManagerCpp
    AssetsManagerH --> TestAssetsManagerCpp
    AssetsManagerH --> HexMapH
    AssetsManagerH --> TestHexMapCpp
    HexTileH --> HexMapH
    HexMapCpp --> HexMapH
    TestHexMapCpp --> HexMapH
  
```

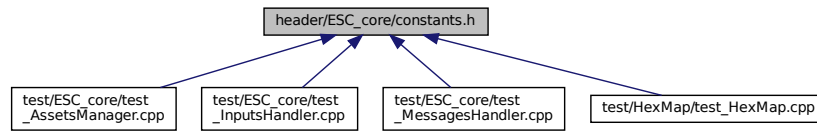
- class `AssetsManager`
A class which manages visual and sound assets.

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
```

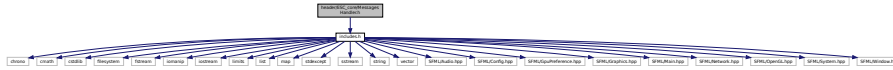
Target seconds per frame (just reciprocal of target frames per second).

4.6 header/ESC_core/MessagesHandler.h File Reference

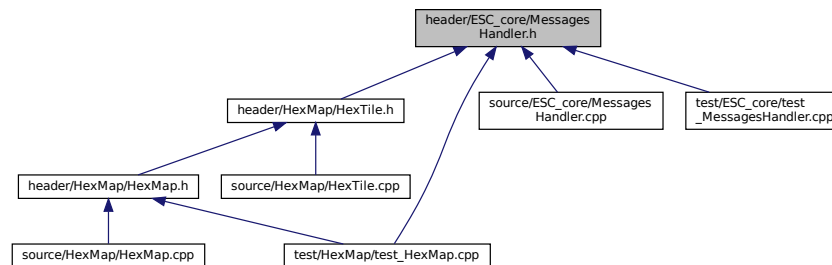
Header file for the [MessagesHandler](#) class.

```
#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

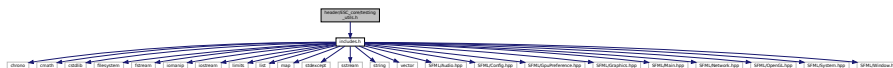
Header file for the [MessagesHandler](#) class.

4.7 header/ESC_core/testing_utils.h File Reference

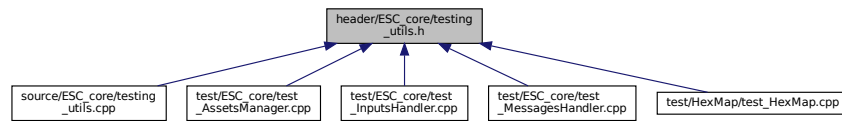
Header file for various testing utilities.

```
#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() */

```

4.7.3 Variable Documentation

4.7.3.1 FLOAT_TOLERANCE

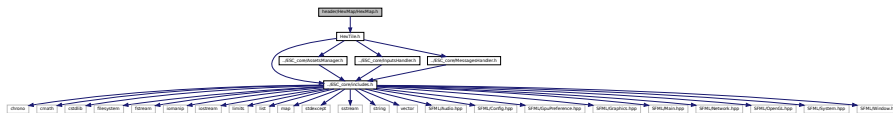
```
const double FLOAT_TOLERANCE = 1e-6
```

Tolerance for floating point equality tests.

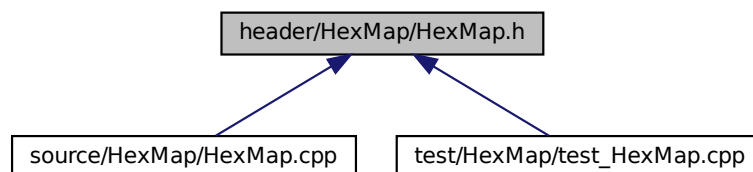
4.8 header/HexMap/HexMap.h File Reference

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

```
#include "HexTile.h"
Include dependency graph for HexMap.h:
```



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



Classes

- class [HexMap](#)
A class which defines a hex map of hex tiles.

4.8.1 Detailed Description

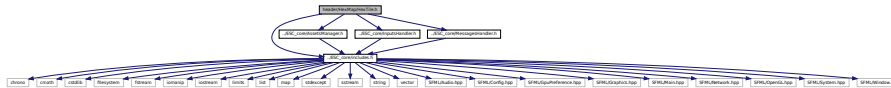
Header file for the [HexMap](#) class.

4.9 header/HexMap/HexTile.h File Reference

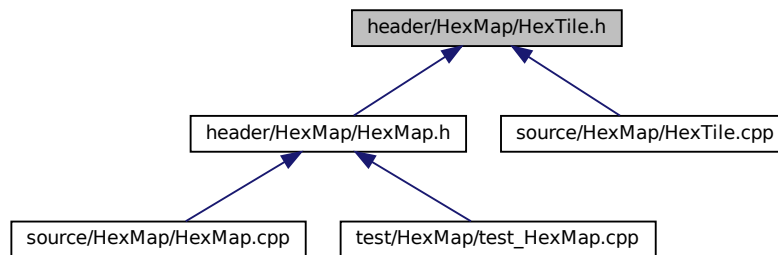
Header file for the [HexTile](#) class.

```
#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.
- enum [TileResource](#) {
POOR , BELOW_AVERAGE , AVERAGE , ABOVE_AVERAGE ,
GOOD , N_TILE_RESOURCES }
An enumeration of the different tile resource values.

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.

Variables

- const std::vector< double > [tile_type_cumulative_probabilities](#)
- const std::vector< double > [tile_resource_cumulative_probabilities](#)

4.9.1 Detailed Description

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

4.9.2 Enumeration Type Documentation

4.9.2.1 TileResource

enum [TileResource](#)

An enumeration of the different tile resource values.

Enumerator

POOR	A poor resource value.
BELOW_AVERAGE	A below average resource value.
AVERAGE	An average resource value.
ABOVE_AVERAGE	An above average resource value.
GOOD	A good resource value.
N_TILE_RESOURCES	A simple hack to get the number of elements in TileResource.

```

63         {
64     POOR,
65     BELOW\_AVERAGE,
66     AVERAGE,
67     ABOVE\_AVERAGE,
68     GOOD,
69     N\_TILE\_RESOURCES
70 };

```

4.9.2.2 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 <code>TileType</code> .

```
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.

4.9.4 Variable Documentation

4.9.4.1 tile_resource_cumulative_probabilities

```
const std::vector<double> tile_resource_cumulative_probabilities
```

Initial value:

```
= {
    0.10,
    0.30,
    0.70,
    0.90,
    1.00
}
```


4.13.1 Detailed Description

Implementation file for various testing utilities.

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

4.13.2 Function Documentation

4.13.2.1 `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() */
```

4.13.2.2 `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 <code>std::cout</code> .
------------------	---

```
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 std::cout.
------------------	---

```
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.

4.15.1 Detailed Description

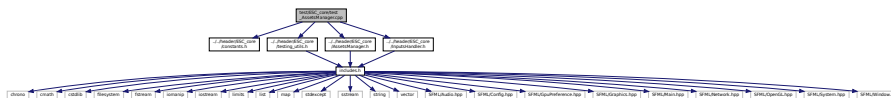
Implementation file for the [HexTile](#) class.

A class which defines a tile of a hex map.

4.16 test/ESC_core/test_AssetsManager.cpp File Reference

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

```
#include "../..//header/ESC_core/constants.h"
#include "../..//header/ESC_core/testing_utils.h"
#include "../..//header/ESC_core/AssetsManager.h"
#include "../..//header/ESC_core/InputsHandler.h"
Include dependency graph for test_AssetsManager.cpp:
```



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 )
37 {
38     #ifdef _WIN32
39         activateVirtualTerminal();
40     #endif /* _WIN32 */
41
42     printGold("\tTesting AssetsManager");
43     std::cout << std::endl;
44
45     srand(time(NULL));
46     int n_dots = 8;
47
48
49     try {
50         // 1. construct
51         InputsHandler inputs_handler;
52         AssetsManager assets_manager;
53
54
55         // 2. load/open some test assets
56         assets_manager.loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
57         assets_manager.loadTexture(
58             "assets/ESC_brand/ESC_key_98x81.png",
59             "ESC_key_98x81"
60         );
61         assets_manager.loadSound("assets/ESC_brand/key_press.ogg", "key_press");
62         assets_manager.loadTrack(
63             "assets/audio/tracks/AlexanderBlu_BackgroundElectronicModernMusic.ogg",
64             "AlexanderBlu_BackgroundElectronicModernMusic"
65         );
66
67
68         // 3. test game loop
69         sf::Clock clock;
70         sf::Event event;
71         sf::RenderWindow window(sf::VideoMode(800, 600), "Testing AssetsManager");
72
73         double screen_width = window.getSize().x;
74         double screen_height = window.getSize().y;
75
76         testFloatEquals(
77             screen_width,
78             800,
79             __FILE__,
80             __LINE__
81         );
82
83         testFloatEquals(
84             screen_height,
85             600,
86             __FILE__,
87             __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.getTexture("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();
175
176             window.draw(ESC_key);
177             window.draw(click_text);
178
179             window.display();
180
181             alpha -= 8;
182             if (alpha < 0) {
183                 alpha = 0;
184             }
185
186             click_text.setFillColor(sf::Color(255, 255, 255, alpha));
187
188             std::cout << frame << " : " << time_since_run_s << "\r" << std::flush;
189             frame++;
190         }
191     }
192 }
193
194
195 catch (...) {
196     //...
197
198     printGold(" ");
199     for (int i = 0; i < n_dots; i++) {
200         printGold(".");
201     }
202     printGold(" ");
203     printRed("FAIL");

```

```

204         std::cout << std::endl;
205         throw;
206     }
207
208
209     //...
210
211     printGold(" ");
212     for (int i = 0; i < n_dots; i++) {
213         printGold(".");
214     }
215     printGold(" ");
216     printGreen("PASS");
217     std::cout << std::endl;
218
219     return 0;
220 } /* main() */

```

4.17 test/ESC_core/test_InputsHandler.cpp File Reference

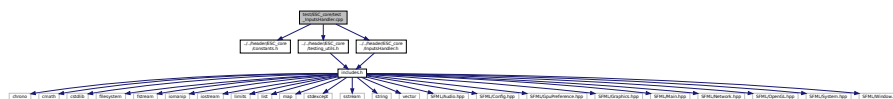
Suite of tests for the [InputsHandler](#) class.

```

#include "../..header/ESC_core/constants.h"
#include "../..header/ESC_core/testing_utils.h"
#include "../..header/ESC_core/InputsHandler.h"

```

Include dependency graph for test_InputsHandler.cpp:



Functions

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

4.17.1 Detailed Description

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

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

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                 {
107                     inputs_handler.process(&event);
108
109                     if (event.type == sf::Event::Closed) {
110                         window.close();
111                     }
112                 }
113
114                 window.clear();
115                 window.display();

```

```

116
117         inputs_handler.printKeysPressed();
118
119         if (inputs_handler.key_pressed_once_vec[sf::Keyboard::Enter]) {
120             std::cout << "Enter" << std::endl;
121         }
122
123
124
125         inputs_handler.reset();
126
127         std::cout << frame << " : " << time_since_run_s << "\r" << std::flush;
128         frame++;
129     }
130 }
131 }
132
133
134 catch (...) {
135     //...
136
137     printGold(" ");
138     for (int i = 0; i < n_dots; i++) {
139         printGold(".");
140     }
141     printGold(" ");
142     printRed("FAIL");
143     std::cout << std::endl;
144     throw;
145 }
146
147
148 //...
149
150 printGold(" ");
151 for (int i = 0; i < n_dots; i++) {
152     printGold(".");
153 }
154 printGold(" ");
155 printGreen("PASS");
156 std::cout << std::endl;
157
158 return 0;
159 } /* main() */

```

4.18 test/ESC_core/test_MessagesHandler.cpp File Reference

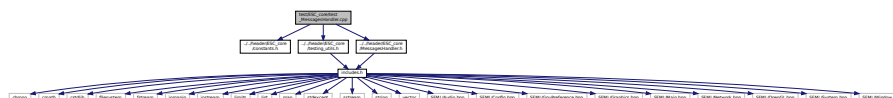
Suite of tests for the [MessagesHandler](#) class.

```

#include "../..header/ESC_core/constants.h"
#include "../..header/ESC_core/testing_utils.h"
#include "../..header/ESC_core/MessagesHandler.h"

```

Include dependency graph for test_MessagesHandler.cpp:



Functions

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

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

```

```

107         for (int i = 0; i < n_dots; i++) {
108             printGold(".");
109         }
110         printGold(" ");
111         printRed("FAIL");
112         std::cout << std::endl;
113         throw;
114     }
115
116
117     //...
118
119     printGold(" ");
120     for (int i = 0; i < n_dots; i++) {
121         printGold(".");
122     }
123     printGold(" ");
124     printGreen("PASS");
125     std::cout << std::endl;
126
127     return 0;
128 } /* main() */

```

4.19 test/HexMap/test_HexMap.cpp File Reference

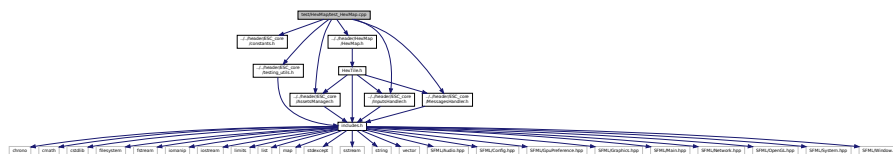
Suite of tests for the [HexMap](#) class.

```

#include "../..header/ESC_core/constants.h"
#include "../..header/ESC_core/testing_utils.h"
#include "../..header/ESC_core/AssetsManager.h"
#include "../..header/ESC_core/InputsHandler.h"
#include "../..header/ESC_core/MessagesHandler.h"
#include "../..header/HexMap/HexMap.h"

```

Include dependency graph for test_HexMap.cpp:



Functions

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

4.19.1 Detailed Description

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

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

4.19.2 Function Documentation

4.19.2.1 main()

```

int main (
    int argc,
    char ** argv )
40 {
41     #ifdef _WIN32
42         activateVirtualTerminal();
43     #endif /* _WIN32 */
44
45     printGold("\tTesting HexMap");
46     std::cout << std::endl;
47
48     srand(time(NULL));
49     int n_dots = 8;
50
51
52     try {
53         // 1. construct, load/open some test assets
54         AssetsManager assets_manager;
55         InputsHandler inputs_handler;
56         MessagesHandler messages_handler;
57
58         assets_manager.loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
59
60
61         // 2. test game loop
62         sf::Clock clock;
63         sf::Event event;
64         sf::RenderWindow window(sf::VideoMode(1200, 800), "Testing AssetsManager");
65
66         double screen_width = window.getSize().x;
67         double screen_height = window.getSize().y;
68
69         testFloatEquals(
70             screen_width,
71             1200,
72             __FILE__,
73             __LINE__
74         );
75
76         testFloatEquals(
77             screen_height,
78             800,
79             __FILE__,
80             __LINE__
81         );
82
83         unsigned long long int frame = 0;
84         double time_since_run_s = 0;
85
86         HexMap hex_map(
87             6,
88             &assets_manager,
89             &inputs_handler,
90             &messages_handler,
91             &window
92         );
93
94         while (window.isOpen()) {
95             time_since_run_s = clock.getElapsedTime().asSeconds();
96
97             if (
98                 time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
99             ) {
100                 while (window.pollEvent(event))
101                 {
102                     inputs_handler.process(&event);
103
104                     if (event.type == sf::Event::Closed) {
105                         window.close();
106                     }
107                 }
108
109                 hex_map.process();
110
111                 if (inputs_handler.key_pressed_once_vec[sf::Keyboard::Q]) {
112                     hex_map.reroll();
113                 }
114
115                 if (inputs_handler.key_pressed_once_vec[sf::Keyboard::R]) {
116                     hex_map.toggleResourceOverlay();
117                 }
118
119                 window.clear();

```

```
120
121         hex_map.draw();
122
123         window.display();
124
125         inputs_handler.reset();
126
127         std::cout << frame << " : " << time_since_run_s << "\r" << std::flush;
128         frame++;
129     }
130 }
131 }
132
133
134 catch (...) {
135     //...
136
137     printGold(" ");
138     for (int i = 0; i < n_dots; i++) {
139         printGold(".");
140     }
141     printGold(" ");
142     printRed("FAIL");
143     std::cout << std::endl;
144     throw;
145 }
146
147
148 //...
149
150 printGold(" ");
151 for (int i = 0; i < n_dots; i++) {
152     printGold(".");
153 }
154 printGold(" ");
155 printGreen("PASS");
156 std::cout << std::endl;
157
158 return 0;
159 } /* main() */
```


Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL <https://www.sfml-dev.org/>. 58

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL <https://www.doxygen.nl>. 57

Wikipedia. Hexagon, 2023. URL <https://en.wikipedia.org/wiki/Hexagon>. 35

Index

- __assembleHexMap
 - HexMap, [20](#)
 - __constructKeyCodeMap
 - InputsHandler, [46](#)
 - __enforceOceanContinuity
 - HexMap, [20](#)
 - __getMajorityTileType
 - HexMap, [21](#)
 - __getNeighboursVector
 - HexMap, [22](#)
 - __getNoise
 - HexMap, [23](#)
 - __getValidMapIndexPositions
 - HexMap, [24](#)
 - __isLakeTouchingOcean
 - HexMap, [25](#)
 - __layTiles
 - HexMap, [25](#)
 - __loadSoundBuffer
 - AssetsManager, [7](#)
 - __procedurallyGenerateTileResources
 - HexMap, [27](#)
 - __procedurallyGenerateTileTypes
 - HexMap, [28](#)
 - __setResourceText
 - HexTile, [36](#)
 - __setUpNodeSprite
 - HexTile, [37](#)
 - __setUpResourceChip
 - HexTile, [37](#)
 - __setUpTileSprite
 - HexTile, [38](#)
 - __smoothTileTypes
 - HexMap, [29](#)
 - ~AssetsManager
 - AssetsManager, [6](#)
 - ~HexMap
 - HexMap, [20](#)
 - ~HexTile
 - HexTile, [36](#)
 - ~InputsHandler
 - InputsHandler, [46](#)
 - ~MessagesHandler
 - MessagesHandler, [53](#)
- ABOVE_AVERAGE
 - HexTile.h, [68](#)
- assess
 - HexTile, [38](#)
- assets_manager_ptr
 - HexMap, [31](#)
 - HexTile, [42](#)
- 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](#)
- AVERAGE
 - HexTile.h, [68](#)
- BELOW_AVERAGE
 - HexTile.h, [68](#)
- border_tiles_vec
 - HexMap, [31](#)
- clear
 - AssetsManager, [8](#)
 - HexMap, [29](#)
- constants.h
 - FRAMES_PER_SECOND, [56](#)
 - SECONDS_PER_FRAME, [56](#)
- current_track
 - AssetsManager, [16](#)
- draw
 - HexMap, [29](#)
 - HexTile, [38](#)
- expectedErrorNotDetected

- testing_utils.cpp, 73
- testing_utils.h, 60
- FLOAT_TOLERANCE
 - testing_utils.h, 66
- font_map
 - AssetsManager, 16
- FOREST
 - HexTile.h, 69
- FOREST_GREEN
 - HexTile.h, 69
- frame
 - HexMap, 31
 - HexTile, 42
- FRAMES_PER_SECOND
 - constants.h, 56
- getCurrentTrackKey
 - AssetsManager, 9
- getFont
 - AssetsManager, 9
- getSound
 - AssetsManager, 10
- getSoundBuffer
 - AssetsManager, 10
- getTexture
 - AssetsManager, 11
- getTrackStatus
 - AssetsManager, 11
- GOOD
 - HexTile.h, 68
- header/ESC_core/AssetsManager.h, 55
- header/ESC_core/constants.h, 56
- header/ESC_core/doxygen_cite.h, 57
- header/ESC_core/includes.h, 57
- header/ESC_core/InputsHandler.h, 58
- header/ESC_core/MessagesHandler.h, 59
- header/ESC_core/testing_utils.h, 59
- header/HexMap/HexMap.h, 66
- header/HexMap/HexTile.h, 67
- hex_map
 - HexMap, 31
- HexMap, 17
 - __assembleHexMap, 20
 - __enforceOceanContinuity, 20
 - __getMajorityTileType, 21
 - __getNeighboursVector, 22
 - __getNoise, 23
 - __getValidMapIndexPositions, 24
 - __isLakeTouchingOcean, 25
 - __layTiles, 25
 - __procedurallyGenerateTileResources, 27
 - __procedurallyGenerateTileTypes, 28
 - __smoothTileTypes, 29
 - ~HexMap, 20
 - assets_manager_ptr, 31
 - border_tiles_vec, 31
 - clear, 29
 - draw, 29
 - frame, 31
 - hex_map, 31
 - HexMap, 19
 - inputs_handler_ptr, 32
 - messages_handler_ptr, 32
 - n_layers, 32
 - n_tiles, 32
 - position_x, 32
 - position_y, 32
 - process, 30
 - render_window_ptr, 33
 - reroll, 30
 - tile_position_x_vec, 33
 - tile_position_y_vec, 33
 - toggleResourceOverlay, 30
- HexTile, 33
 - __setResourceText, 36
 - __setUpNodeSprite, 37
 - __setUpResourceChip, 37
 - __setUpTileSprite, 38
 - ~HexTile, 36
 - assess, 38
 - assets_manager_ptr, 42
 - draw, 38
 - frame, 42
 - HexTile, 35
 - inputs_handler_ptr, 42
 - major_radius, 42
 - messages_handler_ptr, 43
 - minor_radius, 43
 - node_sprite, 43
 - position_x, 43
 - position_y, 43
 - process, 39
 - render_window_ptr, 43
 - resource_assessed, 44
 - resource_chip_sprite, 44
 - resource_text, 44
 - setTileResource, 39, 40
 - setTileType, 40, 41
 - show_node, 44
 - show_resource, 44
 - tile_resource, 44
 - tile_sprite, 45
 - tile_type, 45
 - toggleResourceOverlay, 42
- HexTile.h
 - ABOVE_AVERAGE, 68
 - AVERAGE, 68
 - BELOW_AVERAGE, 68
 - FOREST, 69
 - FOREST_GREEN, 69
 - GOOD, 68
 - LAKE, 69
 - LAKE_BLUE, 69
 - MOUNTAINS, 69
 - MOUNTAINS_GREY, 69

- N_TILE_RESOURCES, 68
- N_TILE_TYPES, 69
- OCEAN, 69
- OCEAN_BLUE, 70
- PLAINS, 69
- PLAINS_YELLOW, 70
- POOR, 68
- tile_resource_cumulative_probabilities, 70
- tile_type_cumulative_probabilities, 70
- TileResource, 68
- TileType, 68
- inputs_handler_ptr
 - HexMap, 32
 - HexTile, 42
- InputsHandler, 45
 - __constructKeyCodeMap, 46
 - ~InputsHandler, 46
 - InputsHandler, 46
 - key_code_map, 52
 - key_press_vec, 52
 - key_pressed_once_vec, 52
 - mouse_left_click, 52
 - printKeysPressed, 50
 - process, 51
 - reset, 51
- key_code_map
 - InputsHandler, 52
- key_press_vec
 - InputsHandler, 52
- key_pressed_once_vec
 - InputsHandler, 52
- LAKE
 - HexTile.h, 69
- LAKE_BLUE
 - HexTile.h, 69
- loadFont
 - AssetsManager, 12
- loadSound
 - AssetsManager, 12
- loadTexture
 - AssetsManager, 13
- loadTrack
 - AssetsManager, 14
- main
 - test_AssetsManager.cpp, 79
 - test_HexMap.cpp, 86
 - test_InputsHandler.cpp, 82
 - test_MessagesHandler.cpp, 85
- major_radius
 - HexTile, 42
- messages_handler_ptr
 - HexMap, 32
 - HexTile, 43
- MessagesHandler, 53
 - ~MessagesHandler, 53
- MessagesHandler, 53
 - process, 54
- minor_radius
 - HexTile, 43
- MOUNTAINS
 - HexTile.h, 69
- MOUNTAINS_GREY
 - HexTile.h, 69
- mouse_left_click
 - InputsHandler, 52
- n_layers
 - HexMap, 32
- N_TILE_RESOURCES
 - HexTile.h, 68
- N_TILE_TYPES
 - HexTile.h, 69
- n_tiles
 - HexMap, 32
- nextTrack
 - AssetsManager, 14
- node_sprite
 - HexTile, 43
- OCEAN
 - HexTile.h, 69
- OCEAN_BLUE
 - HexTile.h, 70
- pauseTrack
 - AssetsManager, 15
- PLAINS
 - HexTile.h, 69
- PLAINS_YELLOW
 - HexTile.h, 70
- playTrack
 - AssetsManager, 15
- POOR
 - HexTile.h, 68
- position_x
 - HexMap, 32
 - HexTile, 43
- position_y
 - HexMap, 32
 - HexTile, 43
- previousTrack
 - AssetsManager, 15
- printGold
 - testing_utils.cpp, 73
 - testing_utils.h, 61
- printGreen
 - testing_utils.cpp, 73
 - testing_utils.h, 61
- printKeysPressed
 - InputsHandler, 50
- printRed
 - testing_utils.cpp, 74
 - testing_utils.h, 61
- process

- HexMap, [30](#)
- HexTile, [39](#)
- InputsHandler, [51](#)
- MessagesHandler, [54](#)
- render_window_ptr
 - HexMap, [33](#)
 - HexTile, [43](#)
- reroll
 - HexMap, [30](#)
- reset
 - InputsHandler, [51](#)
- resource_assessed
 - HexTile, [44](#)
- resource_chip_sprite
 - HexTile, [44](#)
- resource_text
 - HexTile, [44](#)
- SECONDS_PER_FRAME
 - constants.h, [56](#)
- setTileResource
 - HexTile, [39](#), [40](#)
- setTileType
 - HexTile, [40](#), [41](#)
- show_node
 - HexTile, [44](#)
- show_resource
 - HexTile, [44](#)
- sound_map
 - AssetsManager, [16](#)
- soundbuffer_map
 - AssetsManager, [16](#)
- source/ESC_core/AssetsManager.cpp, [71](#)
- source/ESC_core/InputsHandler.cpp, [71](#)
- source/ESC_core/MessagesHandler.cpp, [72](#)
- source/ESC_core/testing_utils.cpp, [72](#)
- source/HexMap/HexMap.cpp, [78](#)
- source/HexMap/HexTile.cpp, [78](#)
- stopTrack
 - AssetsManager, [15](#)
- test/ESC_core/test_AssetsManager.cpp, [79](#)
- test/ESC_core/test_InputsHandler.cpp, [82](#)
- test/ESC_core/test_MessagesHandler.cpp, [84](#)
- test/HexMap/test_HexMap.cpp, [86](#)
- test_AssetsManager.cpp
 - main, [79](#)
- test_HexMap.cpp
 - main, [86](#)
- test_InputsHandler.cpp
 - main, [82](#)
- test_MessagesHandler.cpp
 - main, [85](#)
- testFloatEquals
 - testing_utils.cpp, [74](#)
 - testing_utils.h, [62](#)
- testGreaterThan
 - testing_utils.cpp, [75](#)
 - testing_utils.h, [62](#)
- testGreaterThanOrEqualTo
 - testing_utils.cpp, [75](#)
 - testing_utils.h, [63](#)
- testing_utils.cpp
 - expectedErrorNotDetected, [73](#)
 - printGold, [73](#)
 - printGreen, [73](#)
 - printRed, [74](#)
 - testFloatEquals, [74](#)
 - testGreaterThan, [75](#)
 - testGreaterThanOrEqualTo, [75](#)
 - testLessThan, [76](#)
 - testLessThanOrEqualTo, [77](#)
 - testTruth, [77](#)
- testing_utils.h
 - expectedErrorNotDetected, [60](#)
 - FLOAT_TOLERANCE, [66](#)
 - printGold, [61](#)
 - printGreen, [61](#)
 - printRed, [61](#)
 - testFloatEquals, [62](#)
 - testGreaterThan, [62](#)
 - testGreaterThanOrEqualTo, [63](#)
 - testLessThan, [64](#)
 - testLessThanOrEqualTo, [64](#)
 - testTruth, [65](#)
- testLessThan
 - testing_utils.cpp, [76](#)
 - testing_utils.h, [64](#)
- testLessThanOrEqualTo
 - testing_utils.cpp, [77](#)
 - testing_utils.h, [64](#)
- testTruth
 - testing_utils.cpp, [77](#)
 - testing_utils.h, [65](#)
- texture_map
 - AssetsManager, [16](#)
- tile_position_x_vec
 - HexMap, [33](#)
- tile_position_y_vec
 - HexMap, [33](#)
- tile_resource
 - HexTile, [44](#)
- tile_resource_cumulative_probabilities
 - HexTile.h, [70](#)
- tile_sprite
 - HexTile, [45](#)
- tile_type
 - HexTile, [45](#)
- tile_type_cumulative_probabilities
 - HexTile.h, [70](#)
- TileResource
 - HexTile.h, [68](#)
- TileType
 - HexTile.h, [68](#)
- toggleResourceOverlay
 - HexMap, [30](#)

HexTile, [42](#)
track_map
AssetsManager, [17](#)