

Road To Zero

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 ContextMenu Class Reference	17
3.2.1 Detailed Description	19
3.2.2 Constructor & Destructor Documentation	19
3.2.2.1 ContextMenu()	19
3.2.2.2 ~ContextMenu()	20
3.2.3 Member Function Documentation	20

3.2.3.1 __drawConsoleScreenFrame()	20
3.2.3.2 __drawConsoleText()	21
3.2.3.3 __drawVisualScreenFrame()	22
3.2.3.4 __handleKeyPressEvents()	22
3.2.3.5 __handleMouseButtonEvents()	22
3.2.3.6 __sendQuitGameMessage()	23
3.2.3.7 __sendRestartGameMessage()	23
3.2.3.8 __setConsoleState()	23
3.2.3.9 __setConsoleString()	24
3.2.3.10 __setUpConsoleScreen()	25
3.2.3.11 __setUpConsoleScreenFrame()	25
3.2.3.12 __setUpMenuFrame()	27
3.2.3.13 __setUpVisualScreen()	27
3.2.3.14 __setUpVisualScreenFrame()	28
3.2.3.15 draw()	29
3.2.3.16 processEvent()	29
3.2.3.17 processMessage()	30
3.2.4 Member Data Documentation	30
3.2.4.1 assets_manager_ptr	31
3.2.4.2 console_screen	31
3.2.4.3 console_screen_frame_bottom	31
3.2.4.4 console_screen_frame_left	31
3.2.4.5 console_screen_frame_right	31
3.2.4.6 console_screen_frame_top	31
3.2.4.7 console_state	32
3.2.4.8 console_string	32
3.2.4.9 event_ptr	32
3.2.4.10 frame	32
3.2.4.11 game_menu_up	32
3.2.4.12 menu_frame	32
3.2.4.13 message_hub_ptr	33
3.2.4.14 position_x	33
3.2.4.15 position_y	33
3.2.4.16 render_window_ptr	33
3.2.4.17 visual_screen	33
3.2.4.18 visual_screen_frame_bottom	33
3.2.4.19 visual_screen_frame_left	34
3.2.4.20 visual_screen_frame_right	34
3.2.4.21 visual_screen_frame_top	34
3.3 Game Class Reference	34
3.3.1 Detailed Description	36
3.3.2 Constructor & Destructor Documentation	36

3.3.2.1 Game()	36
3.3.2.2 ~Game()	37
3.3.3 Member Function Documentation	37
3.3.3.1 __draw()	37
3.3.3.2 __drawFrameClockOverlay()	37
3.3.3.3 __drawHUD()	38
3.3.3.4 __handleKeyPressEvents()	38
3.3.3.5 __handleMouseButtonEvents()	39
3.3.3.6 __processEvent()	39
3.3.3.7 __processMessage()	40
3.3.3.8 __toggleFrameClockOverlay()	40
3.3.3.9 run()	40
3.3.4 Member Data Documentation	41
3.3.4.1 assets_manager_ptr	41
3.3.4.2 clock	41
3.3.4.3 context_menu_ptr	42
3.3.4.4 event	42
3.3.4.5 frame	42
3.3.4.6 game_loop_broken	42
3.3.4.7 hex_map_ptr	42
3.3.4.8 message_hub	42
3.3.4.9 month	43
3.3.4.10 quit_game	43
3.3.4.11 render_window_ptr	43
3.3.4.12 show_frame_clock_overlay	43
3.3.4.13 time_since_start_s	43
3.3.4.14 year	43
3.4 HexMap Class Reference	44
3.4.1 Detailed Description	46
3.4.2 Constructor & Destructor Documentation	46
3.4.2.1 HexMap()	46
3.4.2.2 ~HexMap()	47
3.4.3 Member Function Documentation	47
3.4.3.1 __assembleHexMap()	47
3.4.3.2 __enforceOceanContinuity()	48
3.4.3.3 __getMajorityTileType()	48
3.4.3.4 __getNeighboursVector()	49
3.4.3.5 __getNoise()	50
3.4.3.6 __getSelectedTile()	51
3.4.3.7 __getValidMapIndexPositions()	52
3.4.3.8 __handleKeyPressEvents()	53
3.4.3.9 __handleMouseButtonEvents()	53

3.4.3.10	__isLakeTouchingOcean()	54
3.4.3.11	__layTiles()	54
3.4.3.12	__procedurallyGenerateTileResources()	56
3.4.3.13	__procedurallyGenerateTileTypes()	57
3.4.3.14	__sendNoTileSelectedMessage()	58
3.4.3.15	__setUpGlassScreen()	58
3.4.3.16	__smoothTileTypes()	58
3.4.3.17	assess()	59
3.4.3.18	clear()	59
3.4.3.19	draw()	59
3.4.3.20	processEvent()	60
3.4.3.21	processMessage()	60
3.4.3.22	reroll()	61
3.4.3.23	toggleResourceOverlay()	61
3.4.4	Member Data Documentation	61
3.4.4.1	assets_manager_ptr	62
3.4.4.2	border_tiles_vec	62
3.4.4.3	event_ptr	62
3.4.4.4	frame	62
3.4.4.5	glass_screen	62
3.4.4.6	hex_map	62
3.4.4.7	message_hub_ptr	63
3.4.4.8	n_layers	63
3.4.4.9	n_tiles	63
3.4.4.10	position_x	63
3.4.4.11	position_y	63
3.4.4.12	render_window_ptr	63
3.4.4.13	tile_position_x_vec	64
3.4.4.14	tile_position_y_vec	64
3.4.4.15	tile_selected	64
3.5	HexTile Class Reference	64
3.5.1	Detailed Description	67
3.5.2	Constructor & Destructor Documentation	67
3.5.2.1	HexTile()	67
3.5.2.2	~HexTile()	68
3.5.3	Member Function Documentation	68
3.5.3.1	__getTileCoordsSubstring()	68
3.5.3.2	__getTileImprovementSubstring()	68
3.5.3.3	__getTileResourceSubstring()	69
3.5.3.4	__getTileTypeSubstring()	69
3.5.3.5	__handleKeyPressEvents()	70
3.5.3.6	__handleMouseButtonEvents()	70

3.5.3.7 __isClicked()	71
3.5.3.8 __sendTileSelectedMessage()	71
3.5.3.9 __sendTileStateMessage()	72
3.5.3.10 __setResourceText()	72
3.5.3.11 __setUpNodeSprite()	73
3.5.3.12 __setUpResourceChipSprite()	73
3.5.3.13 __setUpSelectOutlineSprite()	74
3.5.3.14 __setUpTileSprite()	74
3.5.3.15 assess()	75
3.5.3.16 draw()	75
3.5.3.17 processEvent()	75
3.5.3.18 processMessage()	76
3.5.3.19 setTileResource() [1/2]	76
3.5.3.20 setTileResource() [2/2]	76
3.5.3.21 setTileType() [1/2]	77
3.5.3.22 setTileType() [2/2]	77
3.5.3.23 toggleResourceOverlay()	78
3.5.4 Member Data Documentation	78
3.5.4.1 assets_manager_ptr	79
3.5.4.2 event_ptr	79
3.5.4.3 frame	79
3.5.4.4 has_improvement	79
3.5.4.5 is_selected	79
3.5.4.6 major_radius	79
3.5.4.7 message_hub_ptr	80
3.5.4.8 minor_radius	80
3.5.4.9 node_sprite	80
3.5.4.10 position_x	80
3.5.4.11 position_y	80
3.5.4.12 render_window_ptr	80
3.5.4.13 resource_assessed	81
3.5.4.14 resource_chip_sprite	81
3.5.4.15 resource_text	81
3.5.4.16 select_outline_sprite	81
3.5.4.17 show_node	81
3.5.4.18 show_resource	81
3.5.4.19 tile_resource	82
3.5.4.20 tile_sprite	82
3.5.4.21 tile_type	82
3.6 Message Struct Reference	82
3.6.1 Detailed Description	82
3.6.2 Member Data Documentation	83

3.6.2.1 bool_payload_vec	83
3.6.2.2 channel	83
3.6.2.3 double_payload_vec	83
3.6.2.4 int_payload_vec	83
3.6.2.5 string_payload	83
3.6.2.6 subject	84
3.7 MessageHub Class Reference	84
3.7.1 Detailed Description	85
3.7.2 Constructor & Destructor Documentation	85
3.7.2.1 MessageHub()	85
3.7.2.2 ~MessageHub()	85
3.7.3 Member Function Documentation	85
3.7.3.1 addChannel()	85
3.7.3.2 clear()	86
3.7.3.3 clearMessages()	86
3.7.3.4 hasTraffic()	87
3.7.3.5 isEmpty()	87
3.7.3.6 popMessage()	87
3.7.3.7 receiveMessage()	88
3.7.3.8 removeChannel()	89
3.7.3.9 sendMessage()	89
3.7.4 Member Data Documentation	90
3.7.4.1 message_map	90
4 File Documentation	91
4.1 header/ContextMenu.h File Reference	91
4.1.1 Detailed Description	92
4.1.2 Enumeration Type Documentation	92
4.1.2.1 ConsoleState	92
4.2 header/ESC_core/AssetsManager.h File Reference	92
4.2.1 Detailed Description	93
4.3 header/ESC_core/constants.h File Reference	93
4.3.1 Detailed Description	95
4.3.2 Function Documentation	95
4.3.2.1 FOREST_GREEN()	95
4.3.2.2 LAKE_BLUE()	95
4.3.2.3 MENU_FRAME_GREY()	96
4.3.2.4 MONOCHROME_SCREEN_BACKGROUND()	96
4.3.2.5 MONOCHROME_TEXT_AMBER()	96
4.3.2.6 MONOCHROME_TEXT_GREEN()	96
4.3.2.7 MONOCHROME_TEXT_RED()	96
4.3.2.8 MOUNTAINS_GREY()	97

4.3.2.9 OCEAN_BLUE()	97
4.3.2.10 PLAINS_YELLOW()	97
4.3.2.11 VISUAL_SCREEN_FRAME_GREY()	97
4.3.3 Variable Documentation	97
4.3.3.1 FLOAT_TOLERANCE	98
4.3.3.2 FRAMES_PER_SECOND	98
4.3.3.3 GAME_CHANNEL	98
4.3.3.4 GAME_HEIGHT	98
4.3.3.5 GAME_WIDTH	98
4.3.3.6 NO_TILE_SELECTED_CHANNEL	98
4.3.3.7 SECONDS_PER_FRAME	99
4.3.3.8 SECONDS_PER_MONTH	99
4.3.3.9 SECONDS_PER_YEAR	99
4.3.3.10 TILE_RESOURCE_CUMULATIVE_PROBABILITIES	99
4.3.3.11 TILE_SELECTED_CHANNEL	99
4.3.3.12 TILE_STATE_CHANNEL	99
4.3.3.13 TILE_TYPE_CUMULATIVE_PROBABILITIES	100
4.4 header/ESC_core/doxygen_cite.h File Reference	100
4.4.1 Detailed Description	100
4.5 header/ESC_core/includes.h File Reference	100
4.5.1 Detailed Description	101
4.6 header/ESC_core/MessageHub.h File Reference	101
4.6.1 Detailed Description	102
4.7 header/ESC_core/testing_utils.h File Reference	102
4.7.1 Detailed Description	103
4.7.2 Function Documentation	103
4.7.2.1 expectedErrorNotDetected()	104
4.7.2.2 printGold()	104
4.7.2.3 printGreen()	104
4.7.2.4 printRed()	105
4.7.2.5 testFloatEquals()	105
4.7.2.6 testGreaterThan()	106
4.7.2.7 testGreaterThanOrEqualTo()	106
4.7.2.8 testLessThan()	107
4.7.2.9 testLessThanOrEqualTo()	108
4.7.2.10 testTruth()	108
4.8 header/Game.h File Reference	109
4.9 header/HexMap.h File Reference	110
4.9.1 Detailed Description	110
4.10 header/HexTile.h File Reference	111
4.10.1 Detailed Description	112
4.10.2 Enumeration Type Documentation	112

4.10.2.1 TileResource	112
4.10.2.2 TileType	112
4.11 source/ContextMenu.cpp File Reference	113
4.11.1 Detailed Description	113
4.12 source/ESC_core/AssetsManager.cpp File Reference	113
4.12.1 Detailed Description	113
4.13 source/ESC_core/MessageHub.cpp File Reference	114
4.13.1 Detailed Description	114
4.14 source/ESC_core/testing_utils.cpp File Reference	114
4.14.1 Detailed Description	115
4.14.2 Function Documentation	115
4.14.2.1 expectedErrorNotDetected()	115
4.14.2.2 printGold()	115
4.14.2.3 printGreen()	116
4.14.2.4 printRed()	116
4.14.2.5 testFloatEquals()	116
4.14.2.6 testGreaterThan()	117
4.14.2.7 testGreaterThanOrEqualTo()	117
4.14.2.8 testLessThan()	118
4.14.2.9 testLessThanOrEqualTo()	119
4.14.2.10 testTruth()	119
4.15 source/Game.cpp File Reference	120
4.15.1 Detailed Description	120
4.16 source/HexMap.cpp File Reference	120
4.16.1 Detailed Description	121
4.17 source/HexTile.cpp File Reference	121
4.17.1 Detailed Description	121
4.18 source/main.cpp File Reference	121
4.18.1 Detailed Description	122
4.18.2 Function Documentation	122
4.18.2.1 constructRenderWindow()	122
4.18.2.2 loadAssets()	122
4.18.2.3 main()	122
Bibliography	125
Index	127

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
ContextMenu	A class which defines a context menu for the game	17
Game	A class which acts as the central class for the game, by containing all other classes and implementing the game loop	34
HexMap	A class which defines a hex map of hex tiles	44
HexTile	A class which defines a hex tile of the hex map	64
Message	A structure which defines a standard message format	82
MessageHub	A class which acts as a central hub for inter-object message traffic	84

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

header/ ContextMenu.h	
Header file for the ContextMenu class	91
header/ Game.h	109
header/ HexMap.h	
Header file for the HexMap class	110
header/ HexTile.h	
Header file for the Game class	111
header/ESC_core/ AssetsManager.h	
Header file for the AssetsManager class	92
header/ESC_core/ constants.h	
Header file for various constants	93
header/ESC_core/ doxygen_cite.h	
Header file which simply cites the doxygen tool	100
header/ESC_core/ includes.h	
Header file for various includes	100
header/ESC_core/ MessageHub.h	
Header file for the MessageHub class	101
header/ESC_core/ testing_utils.h	
Header file for various testing utilities	102
source/ ContextMenu.cpp	
Implementation file for the ContextMenu class	113
source/ Game.cpp	
Implementation file for the Game class	120
source/ HexMap.cpp	
Implementation file for the HexMap class	120
source/ HexTile.cpp	
Implementation file for the HexTile class	121
source/ main.cpp	
Implementation file for main() for Road To Zero	121
source/ESC_core/ AssetsManager.cpp	
Implementation file for the AssetsManager class	113
source/ESC_core/ MessageHub.cpp	
Implementation file for the MessageHub class	114
source/ESC_core/ testing_utils.cpp	
Implementation file for various testing utilities	114

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:

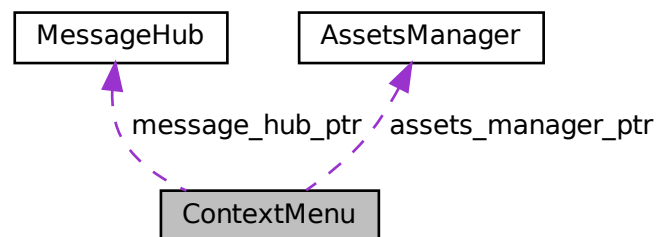
- header/ESC_core/[AssetsManager.h](#)
- source/ESC_core/[AssetsManager.cpp](#)

3.2 ContextMenu Class Reference

A class which defines a context menu for the game.

```
#include <ContextMenu.h>
```

Collaboration diagram for ContextMenu:



Public Member Functions

- [ContextMenu](#) (sf::Event *, sf::RenderWindow *, [AssetsManager](#) *, [MessageHub](#) *)
Constructor for the [ContextMenu](#) class.
- void [processEvent](#) (void)
Method to processEvent [ContextMenu](#). To be called once per event.
- void [processMessage](#) (void)
Method to processMessage [ContextMenu](#). To be called once per message.
- void [draw](#) (void)
Method to draw the hex tile to the render window. To be called once per frame.
- [~ContextMenu](#) (void)
Destructor for the [ContextMenu](#) class.

Public Attributes

- [ConsoleState console_state](#)
The current state of the console screen.
- bool [game_menu_up](#)
Indicates whether or not the game menu is up.
- int [frame](#)
The current frame of this object.
- double [position_x](#)
The position of the object.
- double [position_y](#)
The position of the object.
- std::string [console_string](#)
The string to be printed to the console screen.
- sf::RectangleShape [menu_frame](#)
The frame of the context menu.
- sf::RectangleShape [visual_screen](#)
The context menu screen for visuals.
- sf::ConvexShape [visual_screen_frame_top](#)
The top framing of the visual screen.
- sf::ConvexShape [visual_screen_frame_left](#)
The left framing of the visual screen.
- sf::ConvexShape [visual_screen_frame_bottom](#)
The bottom framing of the visual screen.
- sf::ConvexShape [visual_screen_frame_right](#)
The right framing of the visual screen.
- sf::RectangleShape [console_screen](#)
The context menu console screen (for animated text output).
- sf::ConvexShape [console_screen_frame_top](#)
The top framing of the console screen.
- sf::ConvexShape [console_screen_frame_left](#)
The left framing of the console screen.
- sf::ConvexShape [console_screen_frame_bottom](#)
The bottom framing of the console screen.
- sf::ConvexShape [console_screen_frame_right](#)
The right framing of the console screen.

Private Member Functions

- void [__setUpMenuFrame](#) (void)
Helper method to set up context menu frame (drawable).
- void [__setUpVisualScreen](#) (void)
Helper method to set up context menu visual screen (drawable).
- void [__setUpVisualScreenFrame](#) (void)
Helper method to set up framing for context menu visual screen (drawable).
- void [__drawVisualScreenFrame](#) (void)
Helper method to draw visual screen frame.
- void [__setUpConsoleScreen](#) (void)
Helper method to set up context menu console screen (drawable).
- void [__setUpConsoleScreenFrame](#) (void)

- Helper method to set up framing for context menu console screen (drawable).*
 - void [__drawConsoleScreenFrame](#) (void)
- Helper method to draw console screen frame.*
 - void [__setConsoleState](#) (ConsoleState)
- Helper method to set state of console screen and update string if necessary.*
 - void [__setConsoleString](#) (void)
- Helper method to set console string depending on console state.*
 - void [__drawConsoleText](#) (void)
- Helper method to draw animated text to context menu console screen.*
 - void [__handleKeyPressEvents](#) (void)
- Helper method to handle key press events.*
 - void [__handleMouseButtonEvents](#) (void)
- Helper method to handle mouse button events.*
 - void [__sendQuitGameMessage](#) (void)
- Helper method to format and send a quit game message.*
 - void [__sendRestartGameMessage](#) (void)
- Helper method to format and send a restart game message.*

Private Attributes

- sf::Event * [event_ptr](#)
A pointer to the event class.
- sf::RenderWindow * [render_window_ptr](#)
A pointer to the render window.
- [AssetsManager](#) * [assets_manager_ptr](#)
A pointer to the assets manager.
- [MessageHub](#) * [message_hub_ptr](#)
A pointer to the message hub.

3.2.1 Detailed Description

A class which defines a context menu for the game.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ContextMenu()

```
ContextMenu::ContextMenu (
    sf::Event * event_ptr,
    sf::RenderWindow * render_window_ptr,
    AssetsManager * assets_manager_ptr,
    MessageHub * message_hub_ptr )
```

Constructor for the [ContextMenu](#) class.

Parameters

<i>event_ptr</i>	Pointer to the event class.
<i>render_window_ptr</i>	Pointer to the render window.
<i>assets_manager_ptr</i>	Pointer to the assets manager.
<i>message_hub_ptr</i>	Pointer to the message hub.

```

782 {
783     // 1. set attributes
784
785     // 1.1. private
786     this->event_ptr = event_ptr;
787     this->render_window_ptr = render_window_ptr;
788
789     this->assets_manager_ptr = assets_manager_ptr;
790     this->message_hub_ptr = message_hub_ptr;
791
792     // 1.2. public
793     this->console_state = ConsoleState :: NONE;
794     this->__setConsoleState(ConsoleState :: READY);
795
796     this->game_menu_up = false;
797
798     this->frame = 0;
799
800     this->position_x = GAME_WIDTH;
801     this->position_y = 0;
802
803     // 2. set up and position drawable attributes
804     this->__setUpMenuFrame();
805     this->__setUpVisualScreen();
806     this->__setUpVisualScreenFrame();
807     this->__setUpConsoleScreen();
808     this->__setUpConsoleScreenFrame();
809
810     std::cout << "ContextMenu constructed at " << this << std::endl;
811
812     return;
813 } /* ContextMenu() */

```

3.2.2.2 ~ContextMenu()

```

ContextMenu::~ContextMenu (
    void )

```

Destructor for the [ContextMenu](#) class.

```

952 {
953     std::cout << "ContextMenu at " << this << " destroyed" << std::endl;
954
955     return;
956 } /* ~ContextMenu() */

```

3.2.3 Member Function Documentation

3.2.3.1 __drawConsoleScreenFrame()

```

void ContextMenu::__drawConsoleScreenFrame (
    void ) [private]

```

Helper method to draw console screen frame.

```

433 {

```

```

434     this->render_window_ptr->draw(this->console_screen_frame_top);
435     this->render_window_ptr->draw(this->console_screen_frame_left);
436     this->render_window_ptr->draw(this->console_screen_frame_bottom);
437     this->render_window_ptr->draw(this->console_screen_frame_right);
438
439     return;
440 } /* __drawContextScreenFrame() */

```

3.2.3.2 __drawConsoleText()

```

void ContextMenu::__drawConsoleText (
    void ) [private]

```

Helper method to draw animated text to context menu console screen.

```

550 {
551     // 1. set up console text (drawable)
552     sf::Text console_text(
553         this->console_string,
554         *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
555         16
556     );
557
558     console_text.setFillColor(MONOCROME_TEXT_GREEN);
559
560     console_text.setPosition(
561         this->position_x - 50 - 300 + 16,
562         this->position_y + GAME_HEIGHT - 50 - 340 + 16
563     );
564
565
566     // 2. draw console text
567     this->render_window_ptr->draw(console_text);
568
569
570     // 3. assemble and draw blinking console cursor
571     if ((this->frame % FRAMES_PER_SECOND) > FRAMES_PER_SECOND / 2) {
572         sf::RectangleShape console_cursor(sf::Vector2f(10, 16));
573
574         console_cursor.setFillColor(MONOCROME_TEXT_GREEN);
575
576         console_cursor.setPosition(
577             console_text.getPosition().x,
578             console_text.getPosition().y + console_text.getLocalBounds().height + 10
579         );
580
581         this->render_window_ptr->draw(console_cursor);
582     }
583
584     // 4. updating frame count if console is in menu state
585     if (this->console_state == ConsoleState::MENU) {
586         std::string frame_count_string = "FRAME: ";
587         frame_count_string += std::to_string(this->frame);
588
589         sf::Text frame_count_text(
590             frame_count_string,
591             *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
592             16
593         );
594
595         frame_count_text.setFillColor(MONOCROME_TEXT_GREEN);
596
597         frame_count_text.setPosition(
598             console_text.getPosition().x,
599             console_text.getPosition().y + console_text.getLocalBounds().height - 10
600         );
601
602         this->render_window_ptr->draw(frame_count_text);
603     }
604
605     return;
606 } /* __drawConsoleText() */

```

3.2.3.3 __drawVisualScreenFrame()

```
void ContextMenu::__drawVisualScreenFrame (
    void ) [private]
```

Helper method to draw visual screen frame.

```
208 {
209     this->render_window_ptr->draw(this->visual_screen_frame_top);
210     this->render_window_ptr->draw(this->visual_screen_frame_left);
211     this->render_window_ptr->draw(this->visual_screen_frame_bottom);
212     this->render_window_ptr->draw(this->visual_screen_frame_right);
213
214     return;
215 } /* __drawVisualScreenFrame() */
```

3.2.3.4 __handleKeyPressEvents()

```
void ContextMenu::__handleKeyPressEvents (
    void ) [private]
```

Helper method to handle key press events.

```
621 {
622     switch (this->event_ptr->key.code) {
623         case (sf::Keyboard::Escape): {
624             if (this->console_state == ConsoleState :: MENU) {
625                 this->__setConsoleState(ConsoleState :: READY);
626             }
627
628             else {
629                 this->__setConsoleState(ConsoleState :: MENU);
630             }
631
632             break;
633         }
634
635         case (sf::Keyboard::Q): {
636             if (this->console_state == ConsoleState :: MENU) {
637                 this->__sendQuitGameMessage();
638             }
639
640             }
641
642         case (sf::Keyboard::R): {
643             if (this->console_state == ConsoleState :: MENU) {
644                 this->__sendRestartGameMessage();
645             }
646
647             }
648
649         default: {
650             // do nothing!
651
652             break;
653         }
654     }
655 }
656
657 return;
658 } /* __handleKeyPressEvents() */
```

3.2.3.5 __handleMouseButtonEvents()

```
void ContextMenu::__handleMouseButtonEvents (
    void ) [private]
```

Helper method to handle mouse button events.


```

673 {
674     switch (this->event_ptr->mouseButton.button) {
675         case (sf::Mouse::Left): {
676             //...
677
678             break;
679         }
680
681         case (sf::Mouse::Right): {
682             //...
683
684             break;
685         }
686     }
687
688     default: {
689         // do nothing!
690
691         break;
692     }
693 }
694 }
695
696 return;
697 } /* __handleMouseButtonEvents() */

```

3.2.3.6 __sendQuitGameMessage()

```

void ContextMenu::__sendQuitGameMessage (
    void ) [private]

```

Helper method to format and send a quit game message.

```

712 {
713     Message quit_game_message;
714
715     quit_game_message.channel = GAME_CHANNEL;
716     quit_game_message.subject = "quit game";
717
718     this->message_hub_ptr->sendMessage(quit_game_message);
719
720     return;
721 } /* __sendQuitGameMessage() */

```

3.2.3.7 __sendRestartGameMessage()

```

void ContextMenu::__sendRestartGameMessage (
    void ) [private]

```

Helper method to format and send a restart game message.

```

736 {
737     Message restart_game_message;
738
739     restart_game_message.channel = GAME_CHANNEL;
740     restart_game_message.subject = "restart game";
741
742     this->message_hub_ptr->sendMessage(restart_game_message);
743
744     return;
745 } /* __sendRestartGameMessage() */

```

3.2.3.8 __setConsoleState()

```

void ContextMenu::__setConsoleState (
    ConsoleState console_state ) [private]

```

Helper method to set state of console screen and update string if necessary.

Parameters

<code>console_state</code>	The state (ConsoleState) to set the console to.
----------------------------	---

```

457 {
458     // 1. if no change, do nothing
459     if (this->console_state == console_state) {
460         return;
461     }
462
463     // 2. update console state, set console string accordingly
464     this->console_state = console_state;
465     this->__setConsoleString();
466
467     return;
468 } /* __setConsoleState() */

```

3.2.3.9 __setConsoleString()

```

void ContextMenu::__setConsoleString (
    void ) [private]

```

Helper method to set console string depending on console state.

```

483 {
484     this->console_string.clear();
485
486     switch (this->console_state) {
487         case (ConsoleState :: MENU): {
488             // 32 char x 17 line console "-----\n";
489             this->console_string = "      **** MENU **** \n";
490             this->console_string += " \n";
491             this->console_string += "[R]:  RESTART \n";
492             this->console_string += " \n";
493             this->console_string += "[TAB]: TOGGLE RESOURCE OVERLAY \n";
494             this->console_string += "[T]:  TUTORIAL \n";
495             this->console_string += " \n";
496             this->console_string += " \n";
497             this->console_string += " \n";
498             this->console_string += " \n";
499             this->console_string += " \n";
500             this->console_string += " \n";
501             this->console_string += " \n";
502             this->console_string += "[Q]:  QUIT \n";
503             this->console_string += "[ESC]: CLOSE MENU \n";
504             this->console_string += " \n";
505
506             break;
507         }
508
509         case (ConsoleState :: TILE): {
510             // take console string from tile state message
511
512             break;
513         }
514
515         default: {
516             // 32 char x 17 line console "-----\n";
517             this->console_string = "      **** RTZ 64 CONTEXT V12 **** \n";
518             this->console_string += " \n";
519             this->console_string += "64K RAM SYSTEM  38911 BYTES FREE \n";
520             this->console_string += " \n";
521             this->console_string += "[TAB]: TOGGLE RESOURCE OVERLAY \n";
522             this->console_string += " \n";
523             this->console_string += "[ESC]:          MENU \n";
524             this->console_string += "[LEFT CLICK]: TILE INFO/OPTIONS \n";
525             this->console_string += " \n";
526             this->console_string += " \n";
527             this->console_string += " \n";
528             this->console_string += "READY. \n";
529
530             break;
531         }
532     }
533
534     return;
535 } /* __setConsoleString() */

```

3.2.3.10 __setUpConsoleScreen()

```
void ContextMenu::__setUpConsoleScreen (
    void ) [private]
```

Helper method to set up context menu console screen (drawable).

```
230 {
231     this->console_screen.setSize(sf::Vector2f(300, 340));
232     this->console_screen.setOrigin(300, 340);
233     this->console_screen.setPosition(
234         this->position_x - 50,
235         this->position_y + GAME_HEIGHT - 50
236     );
237     this->console_screen.setFillColor(MONOCHROME_SCREEN_BACKGROUND);
238
239     return;
240 } /* __setUpConsoleScreen() */
```

3.2.3.11 __setUpConsoleScreenFrame()

```
void ContextMenu::__setUpConsoleScreenFrame (
    void ) [private]
```

Helper method to set up framing for context menu console screen (drawable).

```
255 {
256     int n_points = 4;
257
258     // 1. top framing
259     this->console_screen_frame_top.setPointCount(n_points);
260
261     this->console_screen_frame_top.setPoint(
262         0,
263         sf::Vector2f(
264             this->position_x - 50,
265             this->position_y + GAME_HEIGHT - 50 - 340
266         )
267     );
268     this->console_screen_frame_top.setPoint(
269         1,
270         sf::Vector2f(
271             this->position_x - 50 + 16,
272             this->position_y + GAME_HEIGHT - 50 - 340 - 16
273         )
274     );
275     this->console_screen_frame_top.setPoint(
276         2,
277         sf::Vector2f(
278             this->position_x - 350 - 16,
279             this->position_y + GAME_HEIGHT - 50 - 340 - 16
280         )
281     );
282     this->console_screen_frame_top.setPoint(
283         3,
284         sf::Vector2f(
285             this->position_x - 350,
286             this->position_y + GAME_HEIGHT - 50 - 340
287         )
288     );
289
290     this->console_screen_frame_top.setFillColor(VISUAL_SCREEN_FRAME_GREY);
291
292     this->console_screen_frame_top.setOutlineThickness(2);
293     this->console_screen_frame_top.setOutlineColor(sf::Color(0, 0, 0, 255));
294
295     this->console_screen_frame_top.move(0, -2);
296
297
298     // 2. left framing
299     this->console_screen_frame_left.setPointCount(n_points);
300
301     this->console_screen_frame_left.setPoint(
302         0,
303         sf::Vector2f(
304             this->position_x - 350,
305             this->position_y + GAME_HEIGHT - 50 - 340
```

```

306     )
307 );
308 this->console_screen_frame_left.setPoint(
309     1,
310     sf::Vector2f(
311         this->position_x - 350 - 16,
312         this->position_y + GAME_HEIGHT - 50 - 340 - 16
313     )
314 );
315 this->console_screen_frame_left.setPoint(
316     2,
317     sf::Vector2f(
318         this->position_x - 350 - 16,
319         this->position_y + GAME_HEIGHT - 50 + 16
320     )
321 );
322 this->console_screen_frame_left.setPoint(
323     3,
324     sf::Vector2f(
325         this->position_x - 350,
326         this->position_y + GAME_HEIGHT - 50
327     )
328 );
329
330 this->console_screen_frame_left.setFillColors(VISUAL_SCREEN_FRAME_GREY);
331
332 this->console_screen_frame_left.setOutlineThickness(2);
333 this->console_screen_frame_left.setOutlineColor(sf::Color(0, 0, 0, 255));
334
335 this->console_screen_frame_left.move(-2, 0);
336
337
338 // 3. bottom framing
339 this->console_screen_frame_bottom.setPointCount(n_points);
340
341 this->console_screen_frame_bottom.setPoint(
342     0,
343     sf::Vector2f(
344         this->position_x - 350,
345         this->position_y + GAME_HEIGHT - 50
346     )
347 );
348 this->console_screen_frame_bottom.setPoint(
349     1,
350     sf::Vector2f(
351         this->position_x - 350 - 16,
352         this->position_y + GAME_HEIGHT - 50 + 16
353     )
354 );
355 this->console_screen_frame_bottom.setPoint(
356     2,
357     sf::Vector2f(
358         this->position_x - 50 + 16,
359         this->position_y + GAME_HEIGHT - 50 + 16
360     )
361 );
362 this->console_screen_frame_bottom.setPoint(
363     3,
364     sf::Vector2f(
365         this->position_x - 50,
366         this->position_y + GAME_HEIGHT - 50
367     )
368 );
369
370 this->console_screen_frame_bottom.setFillColors(VISUAL_SCREEN_FRAME_GREY);
371
372 this->console_screen_frame_bottom.setOutlineThickness(2);
373 this->console_screen_frame_bottom.setOutlineColor(sf::Color(0, 0, 0, 255));
374
375 this->console_screen_frame_bottom.move(0, 2);
376
377
378 // 4. right framing
379 this->console_screen_frame_right.setPointCount(n_points);
380
381 this->console_screen_frame_right.setPoint(
382     0,
383     sf::Vector2f(
384         this->position_x - 50,
385         this->position_y + GAME_HEIGHT - 50
386     )
387 );
388 this->console_screen_frame_right.setPoint(
389     1,
390     sf::Vector2f(
391         this->position_x - 50 + 16,
392         this->position_y + GAME_HEIGHT - 50 + 16

```

```

393     )
394 );
395 this->console_screen_frame_right.setPoint(
396     2,
397     sf::Vector2f(
398         this->position_x - 50 + 16,
399         this->position_y + GAME_HEIGHT - 50 - 340 - 16
400     )
401 );
402 this->console_screen_frame_right.setPoint(
403     3,
404     sf::Vector2f(
405         this->position_x - 50,
406         this->position_y + GAME_HEIGHT - 50 - 340
407     )
408 );
409
410 this->console_screen_frame_right.setFillColor(VISUAL_SCREEN_FRAME_GREY);
411
412 this->console_screen_frame_right.setOutlineThickness(2);
413 this->console_screen_frame_right.setOutlineColor(sf::Color(0, 0, 0, 255));
414
415 this->console_screen_frame_right.move(2, 0);
416
417 return;
418 } /* __setUpConsoleScreenFrame() */

```

3.2.3.12 __setUpMenuFrame()

```

void ContextMenu::__setUpMenuFrame (
    void ) [private]

```

Helper method to set up context menu frame (drawable).

```

34 {
35     this->menu_frame.setSize(sf::Vector2f(400, GAME_HEIGHT));
36     this->menu_frame.setOrigin(400, 0);
37     this->menu_frame.setPosition(this->position_x, this->position_y);
38     this->menu_frame.setFillColor(MENU_FRAME_GREY);
39
40     return;
41 } /* __setUpMenuFrame() */

```

3.2.3.13 __setUpVisualScreen()

```

void ContextMenu::__setUpVisualScreen (
    void ) [private]

```

Helper method to set up context menu visual screen (drawable).

```

56 {
57     this->visual_screen.setSize(sf::Vector2f(300, 300));
58     this->visual_screen.setOrigin(300, 0);
59     this->visual_screen.setPosition(this->position_x - 50, this->position_y + 50);
60     this->visual_screen.setFillColor(MONOCROME_SCREEN_BACKGROUND);
61
62     return;
63 } /* __setUpVisualScreen() */

```

3.2.3.14 __setUpVisualScreenFrame()

```
void ContextMenu::__setUpVisualScreenFrame (
    void ) [private]
```

Helper method to set up framing for context menu visual screen (drawable).

```
78 {
79     int n_points = 4;
80
81     // 1. top framing
82     this->visual_screen_frame_top.setPointCount(n_points);
83
84     this->visual_screen_frame_top.setPoint(
85         0,
86         sf::Vector2f(this->position_x - 50, this->position_y + 50)
87     );
88     this->visual_screen_frame_top.setPoint(
89         1,
90         sf::Vector2f(this->position_x - 50 + 16, this->position_y + 50 - 16)
91     );
92     this->visual_screen_frame_top.setPoint(
93         2,
94         sf::Vector2f(this->position_x - 350 - 16, this->position_y + 50 - 16)
95     );
96     this->visual_screen_frame_top.setPoint(
97         3,
98         sf::Vector2f(this->position_x - 350, this->position_y + 50)
99     );
100
101     this->visual_screen_frame_top.setFillColor(VISUAL_SCREEN_FRAME_GREY);
102
103     this->visual_screen_frame_top.setOutlineThickness(2);
104     this->visual_screen_frame_top.setOutlineColor(sf::Color(0, 0, 0, 255));
105
106     this->visual_screen_frame_top.move(0, -2);
107
108
109     // 2. left framing
110     this->visual_screen_frame_left.setPointCount(n_points);
111
112     this->visual_screen_frame_left.setPoint(
113         0,
114         sf::Vector2f(this->position_x - 350, this->position_y + 50)
115     );
116     this->visual_screen_frame_left.setPoint(
117         1,
118         sf::Vector2f(this->position_x - 350 - 16, this->position_y + 50 - 16)
119     );
120     this->visual_screen_frame_left.setPoint(
121         2,
122         sf::Vector2f(this->position_x - 350 - 16, this->position_y + 350 + 16)
123     );
124     this->visual_screen_frame_left.setPoint(
125         3,
126         sf::Vector2f(this->position_x - 350, this->position_y + 350)
127     );
128
129     this->visual_screen_frame_left.setFillColor(VISUAL_SCREEN_FRAME_GREY);
130
131     this->visual_screen_frame_left.setOutlineThickness(2);
132     this->visual_screen_frame_left.setOutlineColor(sf::Color(0, 0, 0, 255));
133
134     this->visual_screen_frame_left.move(-2, 0);
135
136
137     // 3. bottom framing
138     this->visual_screen_frame_bottom.setPointCount(n_points);
139
140     this->visual_screen_frame_bottom.setPoint(
141         0,
142         sf::Vector2f(this->position_x - 350, this->position_y + 350)
143     );
144     this->visual_screen_frame_bottom.setPoint(
145         1,
146         sf::Vector2f(this->position_x - 350 - 16, this->position_y + 350 + 16)
147     );
148     this->visual_screen_frame_bottom.setPoint(
149         2,
150         sf::Vector2f(this->position_x - 50 + 16, this->position_y + 350 + 16)
151     );
152     this->visual_screen_frame_bottom.setPoint(
153         3,
154         sf::Vector2f(this->position_x - 50, this->position_y + 350)
155     );
156 }
```

```

156
157     this->visual_screen_frame_bottom.setFillColor(VISUAL_SCREEN_FRAME_GREY);
158
159     this->visual_screen_frame_bottom.setOutlineThickness(2);
160     this->visual_screen_frame_bottom.setOutlineColor(sf::Color(0, 0, 0, 255));
161
162     this->visual_screen_frame_bottom.move(0, 2);
163
164
165     // 4. right framing
166     this->visual_screen_frame_right.setPointCount(n_points);
167
168     this->visual_screen_frame_right.setPoint(
169         0,
170         sf::Vector2f(this->position_x - 50, this->position_y + 350)
171     );
172     this->visual_screen_frame_right.setPoint(
173         1,
174         sf::Vector2f(this->position_x - 50 + 16, this->position_y + 350 + 16)
175     );
176     this->visual_screen_frame_right.setPoint(
177         2,
178         sf::Vector2f(this->position_x - 50 + 16, this->position_y + 50 - 16)
179     );
180     this->visual_screen_frame_right.setPoint(
181         3,
182         sf::Vector2f(this->position_x - 50, this->position_y + 50)
183     );
184
185     this->visual_screen_frame_right.setFillColor(VISUAL_SCREEN_FRAME_GREY);
186
187     this->visual_screen_frame_right.setOutlineThickness(2);
188     this->visual_screen_frame_right.setOutlineColor(sf::Color(0, 0, 0, 255));
189
190     this->visual_screen_frame_right.move(2, 0);
191
192     return;
193 } /* __setUpVisualScreenFrame() */

```

3.2.3.15 draw()

```

void ContextMenu::draw (
    void )

```

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

```

922 {
923     // 1. menu frame
924     this->render_window_ptr->draw(this->menu_frame);
925
926     // 2. visual screen
927     this->render_window_ptr->draw(this->visual_screen);
928     this->__drawVisualScreenFrame();
929
930     // 3. console screen
931     this->render_window_ptr->draw(this->console_screen);
932     this->__drawConsoleScreenFrame();
933     this->__drawConsoleText();
934
935     this->frame++;
936     return;
937 } /* draw() */

```

3.2.3.16 processEvent()

```

void ContextMenu::processEvent (
    void )

```

Method to processEvent [ContextMenu](#). To be called once per event.

```

828 {

```

```

829     if (this->event_ptr->type == sf::Event::KeyPressed) {
830         this->__handleKeyPressEvents();
831     }
832
833     if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
834         this->__handleMouseButtonEvents();
835     }
836
837     return;
838 } /* processEvent() */

```

3.2.3.17 processMessage()

```

void ContextMenu::processMessage (
    void )

```

Method to processMessage [ContextMenu](#). To be called once per message.

```

853 {
854     switch (this->console_state) {
855         case (ConsoleState :: TILE): {
856             // process no tile selected
857             if (not this->message_hub_ptr->isEmpty(NO_TILE_SELECTED_CHANNEL)) {
858                 Message no_tile_selected_message = this->message_hub_ptr->receiveMessage(
859                     NO_TILE_SELECTED_CHANNEL
860                 );
861
862                 if (no_tile_selected_message.subject == "no tile selected") {
863                     this->__setConsoleState(ConsoleState :: READY);
864                     this->message_hub_ptr->popMessage(NO_TILE_SELECTED_CHANNEL);
865                 }
866             }
867
868             // process tile state
869             if (not this->message_hub_ptr->isEmpty(TILE_STATE_CHANNEL)) {
870                 Message tile_state_message = this->message_hub_ptr->receiveMessage(
871                     TILE_STATE_CHANNEL
872                 );
873
874                 if (tile_state_message.subject == "tile state") {
875                     this->console_string = tile_state_message.string_payload;
876                     this->message_hub_ptr->popMessage(TILE_STATE_CHANNEL);
877                 }
878             }
879
880             // process tile selected (subsequent left clicks causing program to hang)
881             if (not this->message_hub_ptr->isEmpty(TILE_SELECTED_CHANNEL)) {
882                 this->message_hub_ptr->popMessage(TILE_SELECTED_CHANNEL);
883             }
884
885             break;
886         }
887
888         default: {
889             // process tile selected
890             if (not this->message_hub_ptr->isEmpty(TILE_SELECTED_CHANNEL)) {
891                 Message tile_selected_message = this->message_hub_ptr->receiveMessage(
892                     TILE_SELECTED_CHANNEL
893                 );
894
895                 if (tile_selected_message.subject == "tile selected") {
896                     this->__setConsoleState(ConsoleState :: TILE);
897                     this->message_hub_ptr->popMessage(TILE_SELECTED_CHANNEL);
898                 }
899             }
900
901             break;
902         }
903     }
904
905     return;
906 } /* processMessage() */

```

3.2.4 Member Data Documentation

3.2.4.1 assets_manager_ptr

`AssetsManager*` ContextMenu::assets_manager_ptr [private]

A pointer to the assets manager.

3.2.4.2 console_screen

`sf::RectangleShape` ContextMenu::console_screen

The context menu console screen (for animated text output).

3.2.4.3 console_screen_frame_bottom

`sf::ConvexShape` ContextMenu::console_screen_frame_bottom

The bottom framing of the console screen.

3.2.4.4 console_screen_frame_left

`sf::ConvexShape` ContextMenu::console_screen_frame_left

The left framing of the console screen.

3.2.4.5 console_screen_frame_right

`sf::ConvexShape` ContextMenu::console_screen_frame_right

The right framing of the console screen.

3.2.4.6 console_screen_frame_top

`sf::ConvexShape` ContextMenu::console_screen_frame_top

The top framing of the console screen.

3.2.4.7 console_state

`ConsoleState ContextMenu::console_state`

The current state of the console screen.

3.2.4.8 console_string

`std::string ContextMenu::console_string`

The string to be printed to the console screen.

3.2.4.9 event_ptr

`sf::Event* ContextMenu::event_ptr [private]`

A pointer to the event class.

3.2.4.10 frame

`int ContextMenu::frame`

The current frame of this object.

3.2.4.11 game_menu_up

`bool ContextMenu::game_menu_up`

Indicates whether or not the game menu is up.

3.2.4.12 menu_frame

`sf::RectangleShape ContextMenu::menu_frame`

The frame of the context menu.

3.2.4.13 message_hub_ptr

```
MessageHub* ContextMenu::message_hub_ptr [private]
```

A pointer to the message hub.

3.2.4.14 position_x

```
double ContextMenu::position_x
```

The position of the object.

3.2.4.15 position_y

```
double ContextMenu::position_y
```

The position of the object.

3.2.4.16 render_window_ptr

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

A pointer to the render window.

3.2.4.17 visual_screen

```
sf::RectangleShape ContextMenu::visual_screen
```

The context menu screen for visuals.

3.2.4.18 visual_screen_frame_bottom

```
sf::ConvexShape ContextMenu::visual_screen_frame_bottom
```

The bottom framing of the visual screen.

3.2.4.19 visual_screen_frame_left

```
sf::ConvexShape ContextMenu::visual_screen_frame_left
```

The left framing of the visual screen.

3.2.4.20 visual_screen_frame_right

```
sf::ConvexShape ContextMenu::visual_screen_frame_right
```

The right framing of the visual screen.

3.2.4.21 visual_screen_frame_top

```
sf::ConvexShape ContextMenu::visual_screen_frame_top
```

The top framing of the visual screen.

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

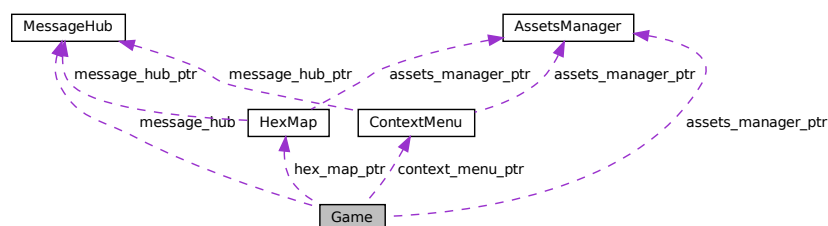
- header/[ContextMenu.h](#)
- source/[ContextMenu.cpp](#)

3.3 Game Class Reference

A class which acts as the central class for the game, by containing all other classes and implementing the game loop.

```
#include <Game.h>
```

Collaboration diagram for Game:



Public Member Functions

- [Game](#) (sf::RenderWindow *, [AssetsManager](#) *)
Constructor for the [Game](#) class.
- bool [run](#) (void)
Method to run game (defines game loop).
- [~Game](#) (void)
Destructor for the [Game](#) class.

Public Attributes

- bool [quit_game](#)
Boolean indicating whether to quit (true) or create a new [Game](#) instance (false).
- bool [game_loop_broken](#)
Boolean indicating whether or not the game loop is broken.
- bool [show_frame_clock_overlay](#)
Boolean indicating whether or not to show frame and clock overlay.
- unsigned long long int [frame](#)
The current frame of the game.
- double [time_since_start_s](#)
The time elapsed [s] since the start of the game.
- unsigned int [year](#)
Current game year.
- unsigned int [month](#)
Current game month.
- sf::Clock [clock](#)
The game clock.
- sf::Event [event](#)
The game events class.
- [MessageHub](#) [message_hub](#)
The message hub (for inter-object message traffic).
- [HexMap](#) * [hex_map_ptr](#)
Pointer to the hex map (defines game world).
- [ContextMenu](#) * [context_menu_ptr](#)
Pointer to the context menu.

Private Member Functions

- void [__toggleFrameClockOverlay](#) (void)
Helper method to toggle frame clock overlay.
- void [__handleKeyPressEvents](#) (void)
Helper method to handle key press events.
- void [__handleMouseButtonEvents](#) (void)
Helper method to handle mouse button events.
- void [__processEvent](#) (void)
Helper method to process [Game](#). To be called once per event.
- void [__processMessage](#) (void)
Helper method to process [Game](#). To be called once per message.
- void [__drawFrameClockOverlay](#) (void)
Helper method to draw frame clock overlay.
- void [__drawHUD](#) (void)
Helper method to heads-up display (HUD).
- void [__draw](#) (void)
Helper method to draw game to the render window. To be called once per frame.

Private Attributes

- `sf::RenderWindow * render_window_ptr`
A pointer to the render window.
- `AssetsManager * assets_manager_ptr`
A pointer to the assets manager.

3.3.1 Detailed Description

A class which acts as the central class for the game, by containing all other classes and implementing the game loop.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Game()

```
Game::Game (
    sf::RenderWindow * render_window_ptr,
    AssetsManager * assets_manager_ptr )
```

Constructor for the `Game` class.

```
337 {
338     // 1. set attributes
339
340     // 1.1. private
341     this->render_window_ptr = render_window_ptr;
342
343     this->assets_manager_ptr = assets_manager_ptr;
344
345     // 1.2. public
346     this->quit_game = false;
347     this->game_loop_broken = false;
348     this->show_frame_clock_overlay = false;
349
350     this->frame = 0;
351     this->time_since_start_s = 0;
352
353     double seconds_since_epoch = time(NULL);
354     double years_since_epoch = seconds_since_epoch / SECONDS_PER_YEAR;
355
356     this->year = 1970 + (int)years_since_epoch;
357     this->month = (years_since_epoch - (int)years_since_epoch) * 12 + 1;
358
359     this->hex_map_ptr = new HexMap(
360         6,
361         &(this->event),
362         this->render_window_ptr,
363         this->assets_manager_ptr,
364         &(this->message_hub)
365     );
366
367     this->context_menu_ptr = new ContextMenu(
368         &(this->event),
369         this->render_window_ptr,
370         this->assets_manager_ptr,
371         &(this->message_hub)
372     );
373
374     // 2. add message channel(s)
375     this->message_hub.addChannel(GAME_CHANNEL);
376
377     std::cout << "Game constructed at " << this << std::endl;
378
379     return;
380 } /* Game() */
```

3.3.2.2 ~Game()

```
Game::~~Game (
    void )
```

Destructor for the [Game](#) class.

```
457 {
458     // 1. clean up attributes
459     delete this->hex_map_ptr;
460     delete this->context_menu_ptr;
461
462     std::cout << "Game at " << this << " destroyed" << std::endl;
463
464     return;
465 } /* ~Game() */
```

3.3.3 Member Function Documentation

3.3.3.1 __draw()

```
void Game::__draw (
    void ) [private]
```

Helper method to draw game to the render window. To be called once per frame.

```
304 {
305     this->__drawHUD();
306
307     if (this->show_frame_clock_overlay) {
308         this->__drawFrameClockOverlay();
309     }
310
311     return;
312 } /* draw() */
```

3.3.3.2 __drawFrameClockOverlay()

```
void Game::__drawFrameClockOverlay (
    void ) [private]
```

Helper method to draw frame clock overlay.

```
200 {
201     std::string frame_clock_string = "FRAME: ";
202     frame_clock_string += std::to_string(this->frame);
203     frame_clock_string += "\nTIME SINCE START [s]: ";
204     frame_clock_string += std::to_string(this->time_since_start_s);
205
206     sf::Text frame_clock_text(
207         frame_clock_string,
208         *(this->assets_manager_ptr->getFont("DroidSansMono")),
209         16
210     );
211
212     sf::RectangleShape frame_clock_backing(
213         sf::Vector2f(
214             1.02 * frame_clock_text.getLocalBounds().width,
215             1.20 * frame_clock_text.getLocalBounds().height
216         )
217     );
218     frame_clock_backing.setFillColor(sf::Color(0, 0, 0, 255));
219
220     this->render_window_ptr->draw(frame_clock_backing);
221     this->render_window_ptr->draw(frame_clock_text);
222
223     return;
224 } /* __drawFrameClockOverlay() */
```

3.3.3.3 __drawHUD()

```
void Game::__drawHUD (
    void ) [private]
```

Helper method to heads-up display (HUD).

```
239 {
240     // 1. first line
241     std::string HUD_string = "YEAR: ";
242     HUD_string += std::to_string(this->year);
243
244     HUD_string += "    MONTH: ";
245     HUD_string += std::to_string(this->month);
246
247     HUD_string += "    POPULATION: ";
248     HUD_string += std::to_string(0);    //<--- CHANGE THIS!
249
250     HUD_string += "    CREDITS: ";
251     HUD_string += std::to_string(0);    //<--- CHANGE THIS!
252     HUD_string += " K";
253
254     sf::Text HUD_text(
255         HUD_string,
256         *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
257         16
258     );
259
260     HUD_text.setPosition(
261         (800 - HUD_text.getLocalBounds().width) / 2,
262         8
263     );
264
265     HUD_text.setFillColor(MONOCROME_TEXT_GREEN);
266
267     this->render_window_ptr->draw(HUD_text);
268
269
270     // 2. second line
271     HUD_string = "DEMAND: ";
272     HUD_string += std::to_string(0);    //<--- CHANGE THIS!
273     HUD_string += " kWh";
274
275     HUD_string += "    EMISSIONS: ";
276     HUD_string += std::to_string(0);    //<--- CHANGE THIS!
277     HUD_string += " tonnes (CO2e)";
278
279     HUD_text.setString(HUD_string);
280
281     HUD_text.setPosition(
282         (800 - HUD_text.getLocalBounds().width) / 2,
283         35
284     );
285
286     this->render_window_ptr->draw(HUD_text);
287
288     return;
289 }    /* __drawHUD() */
```

3.3.3.4 __handleKeyPressEvents()

```
void Game::__handleKeyPressEvents (
    void ) [private]
```

Helper method to handle key press events.

```
59 {
60     switch (this->event.key.code) {
61         case (sf::Keyboard::Tilde): {
62             this->__toggleFrameClockOverlay();
63
64             break;
65         }
66
67
68         case (sf::Keyboard::Tab): {
69             this->hex_map_ptr->togleResourceOverlay();
```



```

70
71         break;
72     }
73
74     default: {
75         // do nothing!
76
77         break;
78     }
79 }
80
81
82 return;
83 } /* __handleKeyPressEvents() */

```

3.3.3.5 __handleMouseButtonEvents()

```

void Game::__handleMouseButtonEvents (
    void ) [private]

```

Helper method to handle mouse button events.

```

98 {
99     switch (this->event.mouseButton.button) {
100         case (sf::Mouse::Left): {
101             //...
102
103             break;
104         }
105
106         case (sf::Mouse::Right): {
107             //...
108
109             break;
110         }
111
112         default: {
113             // do nothing!
114
115             break;
116         }
117     }
118 }
119
120
121 return;
122 } /* __handleMouseButtonEvents() */

```

3.3.3.6 __processEvent()

```

void Game::__processEvent (
    void ) [private]

```

Helper method to process [Game](#). To be called once per event.

```

138 {
139     if (this->event.type == sf::Event::Closed) {
140         this->quit_game = true;
141         this->game_loop_broken = true;
142     }
143
144     if (this->event.type == sf::Event::KeyPressed) {
145         this->__handleKeyPressEvents();
146     }
147
148     if (this->event.type == sf::Event::MouseButtonPressed) {
149         this->__handleMouseButtonEvents();
150     }
151
152     return;
153 } /* __processEvent() */

```

3.3.3.7 __processMessage()

```
void Game::__processMessage (
    void ) [private]
```

Helper method to process `Game`. To be called once per message.

```
168 {
169     if (not this->message_hub.isEmpty(GAME_CHANNEL)) {
170         Message game_channel_message = this->message_hub.receiveMessage(GAME_CHANNEL);
171
172         if (game_channel_message.subject == "quit game") {
173             this->quit_game = true;
174             this->game_loop_broken = true;
175             this->message_hub.popMessage(GAME_CHANNEL);
176         }
177
178         if (game_channel_message.subject == "restart game") {
179             this->game_loop_broken = true;
180             this->message_hub.popMessage(GAME_CHANNEL);
181         }
182     }
183
184     return;
185 } /* __processMessage() */
```

3.3.3.8 __toggleFrameClockOverlay()

```
void Game::__toggleFrameClockOverlay (
    void ) [private]
```

Helper method to toggle frame clock overlay.

```
34 {
35     if (this->show_frame_clock_overlay) {
36         this->show_frame_clock_overlay = false;
37     }
38
39     else {
40         this->show_frame_clock_overlay = true;
41     }
42
43     return;
44 } /* __toggleFrameClockOverlay() */
```

3.3.3.9 run()

```
bool Game::run (
    void )
```

Method to run game (defines game loop).

Returns

Boolean indicating whether to quit (true) or create a new [Game](#) instance (false).

```

398 {
399     // 1. play brand animation
400     //...
401
402     // 2. show splash screen
403     //...
404
405     // 3. start game loop
406     while (not this->game_loop_broken) {
407         this->time_since_start_s = this->clock.getElapsedTime().asSeconds();
408
409         if (this->time_since_start_s >= (this->frame + 1) * SECONDS_PER_FRAME) {
410             // 6.1. process events
411             while (this->render_window_ptr->pollEvent(this->event)) {
412                 this->hex_map_ptr->processEvent();
413                 this->context_menu_ptr->processEvent();
414                 this->__processEvent();
415             }
416
417             // 6.2. process messages
418             while (this->message_hub.hasTraffic()) {
419                 this->hex_map_ptr->processMessage();
420                 this->context_menu_ptr->processMessage();
421                 this->__processMessage();
422             }
423
424             // 6.3. draw frame
425             this->render_window_ptr->clear();
426
427             this->hex_map_ptr->draw();
428             this->context_menu_ptr->draw();
429             this->__draw();
430
431             this->render_window_ptr->display();
432
433             // 6.4. increment frame
434             this->frame++;
435         }
436     }
437     return this->quit_game;
438 }
439 /* run() */

```

3.3.4 Member Data Documentation

3.3.4.1 assets_manager_ptr

`AssetsManager* Game::assets_manager_ptr` [private]

A pointer to the assets manager.

3.3.4.2 clock

`sf::Clock Game::clock`

The game clock.

3.3.4.3 context_menu_ptr

`ContextMenu*` `Game::context_menu_ptr`

Pointer to the context menu.

3.3.4.4 event

`sf::Event` `Game::event`

The game events class.

3.3.4.5 frame

`unsigned long long int` `Game::frame`

The current frame of the game.

3.3.4.6 game_loop_broken

`bool` `Game::game_loop_broken`

Boolean indicating whether or not the game loop is broken.

3.3.4.7 hex_map_ptr

`HexMap*` `Game::hex_map_ptr`

Pointer to the hex map (defines game world).

3.3.4.8 message_hub

`MessageHub` `Game::message_hub`

The message hub (for inter-object message traffic).

3.3.4.9 month

```
unsigned int Game::month
```

Current game month.

3.3.4.10 quit_game

```
bool Game::quit_game
```

Boolean indicating whether to quit (true) or create a new [Game](#) instance (false).

3.3.4.11 render_window_ptr

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

A pointer to the render window.

3.3.4.12 show_frame_clock_overlay

```
bool Game::show_frame_clock_overlay
```

Boolean indicating whether or not to show frame and clock overlay.

3.3.4.13 time_since_start_s

```
double Game::time_since_start_s
```

The time elapsed [s] since the start of the game.

3.3.4.14 year

```
unsigned int Game::year
```

Current game year.

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

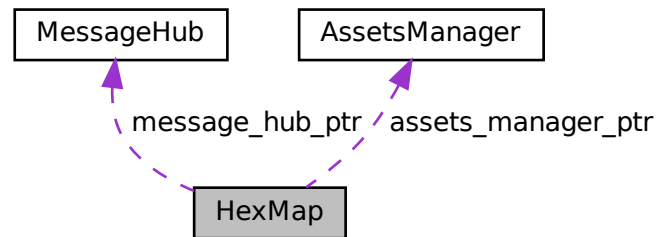
- header/[Game.h](#)
- source/[Game.cpp](#)

3.4 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, sf::Event *, sf::RenderWindow *, [AssetsManager](#) *, [MessageHub](#) *)
Constructor (intended) for the [HexMap](#) class.
- void [assess](#) (void)
Method to assess the resource of the selected tile.
- void [reroll](#) (void)
Method to re-roll the hex map.
- void [toggleResourceOverlay](#) (void)
Method to toggle the hex map resource overlay.
- void [processEvent](#) (void)
Method to process [HexMap](#). To be called once per event.
- void [processMessage](#) (void)
Method to process [HexMap](#). To be called once per message.
- void [draw](#) (void)
Method to draw the hex map to the render window. To be called once per frame.
- void [clear](#) (void)
Method to clear the hex map.
- [~HexMap](#) (void)
Destructor for the [HexMap](#) class.

Public Attributes

- bool [tile_selected](#)
A boolean which indicates if a tile is currently selected.
- 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.
- sf::RectangleShape [glass_screen](#)
To give the effect of an old glass screen over the hex map.
- 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 [__setUpGlassScreen](#) (void)
Helper method to set up glass screen effect (drawable).
- 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.
- [HexTile](#) * [__getSelectedTile](#) (void)
Helper method to get pointer to selected tile.
- void [__handleKeyPressEvents](#) (void)
Helper method to handle key press events.
- void [__handleMouseButtonEvents](#) (void)
Helper method to handle mouse button events.
- void [__sendNoTileSelectedMessage](#) (void)
Helper method to format and send message on no tile selected.

Private Attributes

- sf::Event * [event_ptr](#)
A pointer to the event class.
- sf::RenderWindow * [render_window_ptr](#)
A pointer to the render window.
- [AssetsManager](#) * [assets_manager_ptr](#)
A pointer to the assets manager.
- [MessageHub](#) * [message_hub_ptr](#)
A pointer to the message hub.

3.4.1 Detailed Description

A class which defines a hex map of hex tiles.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 HexMap()

```
HexMap::HexMap (
    int n_layers,
    sf::Event * event_ptr,
    sf::RenderWindow * render_window_ptr,
    AssetsManager * assets_manager_ptr,
    MessageHub * message_hub_ptr )
```

Constructor (intended) for the [HexMap](#) class.

Parameters

<i>n_layers</i>	The number of layers in the HexMap .
<i>event_ptr</i>	Pointer to the event class.
<i>render_window_ptr</i>	Pointer to the render window.
<i>assets_manager_ptr</i>	Pointer to the assets manager.
<i>message_hub_ptr</i>	Pointer to the message hub.


```

972 {
973     // 1. set attributes
974
975     // 1.1. private
976     this->event_ptr = event_ptr;
977     this->render_window_ptr = render_window_ptr;
978
979     this->assets_manager_ptr = assets_manager_ptr;
980     this->message_hub_ptr = message_hub_ptr;
981
982     // 1.2. public
983     this->tile_selected = false;
984
985     this->frame = 0;
986
987     this->n_layers = n_layers;
988     if (this->n_layers < 0) {
989         this->n_layers = 0;
990     }
991
992     this->position_x = 400;
993     this->position_y = 400;
994
995     // 2. assemble n layer hex map
996     this->__assembleHexMap();
997
998     // 3. set up and position drawable attributes
999     this->__setUpGlassScreen();
1000
1001     // 4. add message channel(s)
1002     this->message_hub_ptr->addChannel(TILE_SELECTED_CHANNEL);
1003     this->message_hub_ptr->addChannel(NO_TILE_SELECTED_CHANNEL);
1004     this->message_hub_ptr->addChannel(TILE_STATE_CHANNEL);
1005
1006     std::cout << "HexMap constructed at " << this << std::endl;
1007
1008     return;
1009 } /* HexMap(), intended */

```

3.4.2.2 ~HexMap()

```

HexMap::~HexMap (
    void )

```

Destructor for the [HexMap](#) class.

```

1273 {
1274     this->clear();
1275
1276     std::cout << "HexMap at " << this << " destroyed" << std::endl;
1277
1278     return;
1279 } /* ~HexMap() */

```

3.4.3 Member Function Documentation

3.4.3.1 __assembleHexMap()

```

void HexMap::__assembleHexMap (
    void ) [private]

```

Helper method to assemble the hex map.

```

758 {
759     // 1. seed RNG (using milliseconds since 1 Jan 1970)
760     unsigned long long int milliseconds_since_epoch =
761         std::chrono::duration_cast<std::chrono::milliseconds>(

```

```

762         std::chrono::system_clock::now().time_since_epoch()
763         ).count();
764     srand(millisecons_since_epoch);
765
766     // 2. lay tiles
767     this->__layTiles();
768
769     // 3. procedurally generate types
770     this->__procedurallyGenerateTileTypes();
771
772     // 4. procedurally generate resources
773     this->__procedurallyGenerateTileResources();
774
775     return;
776 } /* __assembleHexMap() */

```

3.4.3.2 __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.

```

669 {
670     std::cout << "enforcing ocean continuity ..." << std::endl;
671
672     bool tile_changed = false;
673
674     // 1. scan tiles and enforce (where appropriate)
675     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
676     std::map<double, HexTile*>::iterator hex_map_iter_y;
677     HexTile* hex_ptr;
678     for (
679         hex_map_iter_x = this->hex_map.begin();
680         hex_map_iter_x != this->hex_map.end();
681         hex_map_iter_x++
682     ) {
683         for (
684             hex_map_iter_y = hex_map_iter_x->second.begin();
685             hex_map_iter_y != hex_map_iter_x->second.end();
686             hex_map_iter_y++
687         ) {
688             hex_ptr = hex_map_iter_y->second;
689
690             if (this->__isLakeTouchingOcean(hex_ptr)) {
691                 hex_ptr->setTileType(TileType::OCEAN);
692                 tile_changed = true;
693             }
694         }
695     }
696
697     if (tile_changed) {
698         this->__enforceOceanContinuity();
699     }
700     else {
701         return;
702     }
703 } /* __enforceOceanContinuity() */

```

3.4.3.3 __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

<code>hex_ptr</code>	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.

```

525 {
526     // 1. init type count map
527     std::map<TileType, int> type_count_map;
528     type_count_map[hex_ptr->tile_type] = 1;
529
530     // 2. survey neighbours, count type instances
531     std::vector<HexTile*> neighbours_vec = this->__getNeighboursVector(hex_ptr);
532
533     for (size_t i = 0; i < neighbours_vec.size(); i++) {
534         if (type_count_map.count(neighbours_vec[i]->tile_type) <= 0) {
535             type_count_map[neighbours_vec[i]->tile_type] = 1;
536         }
537         else {
538             type_count_map[neighbours_vec[i]->tile_type] += 1;
539         }
540     }
541
542     // 3. find majority tile type
543     int max_count = -1 * std::numeric_limits<int>::infinity();
544     TileType majority_tile_type = hex_ptr->tile_type;
545
546     std::map<TileType, int>::iterator map_iter;
547     for (
548         map_iter = type_count_map.begin();
549         map_iter != type_count_map.end();
550         map_iter++
551     ){
552         if (map_iter->second > max_count) {
553             max_count = map_iter->second;
554             majority_tile_type = map_iter->first;
555         }
556     }
557
558     // 4. detect ties
559     for (
560         map_iter = type_count_map.begin();
561         map_iter != type_count_map.end();
562         map_iter++
563     ){
564         if (
565             map_iter->second == max_count and
566             map_iter->first != majority_tile_type
567         ) {
568             majority_tile_type = hex_ptr->tile_type;
569             break;
570         }
571     }
572
573     return majority_tile_type;
574 } /* __getMajorityTileType() */

```

3.4.3.4 __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

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

Returns

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

```

467 {
468     std::vector<HexTile*> neighbours_vec;
469
470     // 1. build potential neighbour positions
471     std::vector<double> potential_neighbour_x_vec(6, 0);
472     std::vector<double> potential_neighbour_y_vec(6, 0);
473
474     for (int i = 0; i < 6; i++) {
475         potential_neighbour_x_vec[i] = hex_ptr->position_x +
476             2 * hex_ptr->minor_radius * cos((60 * i) * (M_PI / 180));
477
478         potential_neighbour_y_vec[i] = hex_ptr->position_y +
479             2 * hex_ptr->minor_radius * sin((60 * i) * (M_PI / 180));
480     }
481
482     // 2. populate neighbours vector
483     std::vector<double> map_index_positions;
484     double potential_x = 0;
485     double potential_y = 0;
486
487     for (int i = 0; i < 6; i++) {
488         potential_x = potential_neighbour_x_vec[i];
489         potential_y = potential_neighbour_y_vec[i];
490
491         map_index_positions = this->__getValidMapIndexPositions(
492             potential_x,
493             potential_y
494         );
495
496         if (not (map_index_positions[0] == -1)) {
497             neighbours_vec.push_back(
498                 this->hex_map[map_index_positions[0]][map_index_positions[1]]
499             );
500         }
501     }
502
503     return neighbours_vec;
504 } /* __getNeighbourVector() */

```

3.4.3.5 __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].

```

247 {
248     // 1. generate random amplitude, wave number, direction, and phase vectors
249     std::vector<double> random_amplitude_vec(n_components, 0);
250     std::vector<double> random_wave_number_vec(n_components, 0);
251     std::vector<double> random_frequency_vec(n_components, 0);
252     std::vector<double> random_direction_vec(n_components, 0);
253     std::vector<double> random_phase_vec(n_components, 0);
254
255     for (int i = 0; i < n_components; i++) {

```

```

256         random_amplitude_vec[i] = 10 * ((double)rand() / RAND_MAX);
257
258         random_wave_number_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
259
260         random_frequency_vec[i] = ((double)rand() / RAND_MAX);
261
262         random_direction_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
263
264         random_phase_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
265     }
266
267     // 2. generate noise vec
268     double amp = 0;
269     double wave_no = 0;
270     double freq = 0;
271     double dir = 0;
272     double phase = 0;
273
274     double x = 0;
275     double y = 0;
276     double t = time(NULL);
277
278     double max_noise = -1 * std::numeric_limits<double>::infinity();
279     double min_noise = std::numeric_limits<double>::infinity();
280
281     double noise = 0;
282     std::vector<double> noise_vec(n_elements, 0);
283
284     for (int i = 0; i < n_elements; i++) {
285         x = this->tile_position_x_vec[i] - this->position_x;
286         y = this->tile_position_y_vec[i] - this->position_y;
287
288         for (int j = 0; j < n_components; j++) {
289             amp = random_amplitude_vec[j];
290             wave_no = random_wave_number_vec[j];
291             freq = random_frequency_vec[j];
292             dir = random_direction_vec[j];
293             phase = random_phase_vec[j];
294
295             noise += (amp / (j + 1)) * cos(
296                 wave_no * (j + 1) * (x * sin(dir) + y * cos(dir)) +
297                 2 * M_PI * (j + 1) * freq * t +
298                 phase
299             );
300         }
301
302         noise_vec[i] = noise;
303
304         if (noise > max_noise) {
305             max_noise = noise;
306         }
307
308         else if (noise < min_noise) {
309             min_noise = noise;
310         }
311
312         noise = 0;
313     }
314
315     // 3. normalize noise vec
316     for (int i = 0; i < n_elements; i++) {
317         noise_vec[i] = (noise_vec[i] - min_noise) / (max_noise - min_noise);
318
319         if (noise_vec[i] < 0) {
320             noise_vec[i] = 0;
321         }
322         else if (noise_vec[i] > 1) {
323             noise_vec[i] = 1;
324         }
325     }
326
327     return noise_vec;
328 } /* __getNoise() */

```

3.4.3.6 __getSelectedTile()

```

HexTile * HexMap::__getSelectedTile (
    void ) [private]

```

Helper method to get pointer to selected tile.

Returns

Pointer to selected tile (or NULL if no tile selected).

```

793 {
794     HexTile* selected_tile_ptr = NULL;
795
796     bool break_flag = false;
797     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
798     std::map<double, HexTile*>::iterator hex_map_iter_y;
799
800     for (
801         hex_map_iter_x = this->hex_map.begin();
802         hex_map_iter_x != this->hex_map.end();
803         hex_map_iter_x++
804     ) {
805         for (
806             hex_map_iter_y = hex_map_iter_x->second.begin();
807             hex_map_iter_y != hex_map_iter_x->second.end();
808             hex_map_iter_y++
809         ) {
810             if (hex_map_iter_y->second->is_selected) {
811                 selected_tile_ptr = hex_map_iter_y->second;
812                 break_flag = true;
813             }
814
815             if (break_flag) {
816                 break;
817             }
818         }
819
820         if (break_flag) {
821             break;
822         }
823     }
824
825     return selected_tile_ptr;
826 } /* __getSelectedTile() */

```

3.4.3.7 __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.

```

413 {
414     std::vector<double> map_index_positions = {-1, -1};
415
416     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
417     std::map<double, HexTile*>::iterator hex_map_iter_y;
418     HexTile* hex_ptr;
419
420     double distance = 0;
421
422     for (
423         hex_map_iter_x = this->hex_map.begin();

```

```

424         hex_map_iter_x != this->hex_map.end();
425         hex_map_iter_x++
426     ) {
427         for (
428             hex_map_iter_y = hex_map_iter_x->second.begin();
429             hex_map_iter_y != hex_map_iter_x->second.end();
430             hex_map_iter_y++
431         ) {
432             hex_ptr = hex_map_iter_y->second;
433
434             distance = sqrt (
435                 pow(hex_ptr->position_x - potential_x, 2) +
436                 pow(hex_ptr->position_y - potential_y, 2)
437             );
438
439             if (distance <= hex_ptr->minor_radius / 4) {
440                 map_index_positions = {hex_ptr->position_x, hex_ptr->position_y};
441                 return map_index_positions;
442             }
443         }
444     }
445
446     return map_index_positions;
447 } /* __isInHexMap() */

```

3.4.3.8 __handleKeyPressEvents()

```

void HexMap::__handleKeyPressEvents (
    void ) [private]

```

Helper method to handle key press events.

```

841 {
842     switch (this->event_ptr->key.code) {
843         case (sf::Keyboard::Escape): {
844             this->tile_selected = false;
845         }
846
847         default: {
848             // do nothing!
849
850             break;
851         }
852     }
853 }
854
855 return;
856 } /* __handleKeyPressEvents() */

```

3.4.3.9 __handleMouseButtonEvents()

```

void HexMap::__handleMouseButtonEvents (
    void ) [private]

```

Helper method to handle mouse button events.

```

871 {
872     switch (this->event_ptr->mouseButton.button) {
873         case (sf::Mouse::Left): {
874             HexTile* hex_ptr = this->__getSelectedTile();
875
876             if (hex_ptr != NULL) {
877                 this->tile_selected = true;
878             }
879
880             else if (this->tile_selected) {
881                 this->tile_selected = false;
882                 this->__sendNoTileSelectedMessage();
883             }
884
885             break;

```

```

886     }
887
888
889     case (sf::Mouse::Right): {
890         if (this->tile_selected) {
891             this->tile_selected = false;
892             this->__sendNoTileSelectedMessage();
893         }
894
895         break;
896     }
897
898
899     default: {
900         // do nothing!
901
902         break;
903     }
904 }
905
906 return;
907 } /* __handleMouseButtonEvents() */

```

3.4.3.10 __isLakeTouchingOcean()

```

bool HexMap::__isLakeTouchingOcean (
    HexTile * hex_ptr ) [private]
636 {
637     // 1. if not lake tile, return
638     if (not (hex_ptr->tile_type == TileType :: LAKE)) {
639         return false;
640     }
641
642     // 2. scan neighbours for ocean tiles
643     std::vector<HexTile*> neighbours_vec = this->__getNeighboursVector(hex_ptr);
644
645     for (size_t i = 0; i < neighbours_vec.size(); i++) {
646         if (neighbours_vec[i]->tile_type == TileType :: OCEAN) {
647             return true;
648         }
649     }
650
651     return false;
652 } /* __isLakeTouchingOcean() */

```

3.4.3.11 __layTiles()

```

void HexMap::__layTiles (
    void ) [private]

```

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

```

54 {
55     this->n_tiles = 0;
56
57     // 1. add origin tile
58     HexTile* hex_ptr = new HexTile(
59         this->position_x,
60         this->position_y,
61         this->event_ptr,
62         this->render_window_ptr,
63         this->assets_manager_ptr,
64         this->message_hub_ptr
65     );
66
67     this->hex_map[this->position_x][this->position_y] = hex_ptr;
68     this->tile_position_x_vec.push_back(this->position_x);
69     this->tile_position_y_vec.push_back(this->position_y);
70     this->n_tiles++;
71
72

```



```

73 // 2. fill out first row (reflect across origin tile)
74 for (int i = 0; i < this->n_layers; i++) {
75     hex_ptr = new HexTile(
76         this->position_x + 2 * (i + 1) * hex_ptr->minor_radius,
77         this->position_y,
78         this->event_ptr,
79         this->render_window_ptr,
80         this->assets_manager_ptr,
81         this->message_hub_ptr
82     );
83
84     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
85     this->tile_position_x_vec.push_back(hex_ptr->position_x);
86     this->tile_position_y_vec.push_back(hex_ptr->position_y);
87     this->n_tiles++;
88
89     if (i == this->n_layers - 1) {
90         this->border_tiles_vec.push_back(hex_ptr);
91     }
92
93     hex_ptr = new HexTile(
94         this->position_x - 2 * (i + 1) * hex_ptr->minor_radius,
95         this->position_y,
96         this->event_ptr,
97         this->render_window_ptr,
98         this->assets_manager_ptr,
99         this->message_hub_ptr
100    );
101
102    this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
103    this->tile_position_x_vec.push_back(hex_ptr->position_x);
104    this->tile_position_y_vec.push_back(hex_ptr->position_y);
105    this->n_tiles++;
106
107    if (i == this->n_layers - 1) {
108        this->border_tiles_vec.push_back(hex_ptr);
109    }
110 }
111
112
113 // 3. fill out subsequent rows (reflect across first row)
114 HexTile* first_row_left_tile = hex_ptr;
115
116 int offset_count = 1;
117
118 double x_offset = 0;
119 double y_offset = 0;
120
121 for (
122     int row_width = 2 * this->n_layers;
123     row_width > this->n_layers;
124     row_width--
125 ) {
126     // 3.1. upper row
127     x_offset = first_row_left_tile->position_x +
128         2 * offset_count * first_row_left_tile->minor_radius *
129         cos(60 * (M_PI / 180));
130
131     y_offset = first_row_left_tile->position_y -
132         2 * offset_count * first_row_left_tile->minor_radius *
133         sin(60 * (M_PI / 180));
134
135     hex_ptr = new HexTile(
136         x_offset,
137         y_offset,
138         this->event_ptr,
139         this->render_window_ptr,
140         this->assets_manager_ptr,
141         this->message_hub_ptr
142     );
143
144     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
145     this->tile_position_x_vec.push_back(hex_ptr->position_x);
146     this->tile_position_y_vec.push_back(hex_ptr->position_y);
147     this->n_tiles++;
148
149     this->border_tiles_vec.push_back(hex_ptr);
150
151     for (int i = 1; i < row_width; i++) {
152         x_offset += 2 * first_row_left_tile->minor_radius;
153
154         hex_ptr = new HexTile(
155             x_offset,
156             y_offset,
157             this->event_ptr,
158             this->render_window_ptr,
159             this->assets_manager_ptr,

```

```

160         this->message_hub_ptr
161     );
162
163     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
164     this->tile_position_x_vec.push_back(hex_ptr->position_x);
165     this->tile_position_y_vec.push_back(hex_ptr->position_y);
166     this->n_tiles++;
167
168     if (row_width == this->n_layers + 1 or i == row_width - 1) {
169         this->border_tiles_vec.push_back(hex_ptr);
170     }
171 }
172
173 // 3.2. lower row
174 x_offset = first_row_left_tile->position_x +
175     2 * offset_count * first_row_left_tile->minor_radius *
176     cos(60 * (M_PI / 180));
177
178 y_offset = first_row_left_tile->position_y +
179     2 * offset_count * first_row_left_tile->minor_radius *
180     sin(60 * (M_PI / 180));
181
182 hex_ptr = new HexTile(
183     x_offset,
184     y_offset,
185     this->event_ptr,
186     this->render_window_ptr,
187     this->assets_manager_ptr,
188     this->message_hub_ptr
189 );
190
191 this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
192 this->tile_position_x_vec.push_back(hex_ptr->position_x);
193 this->tile_position_y_vec.push_back(hex_ptr->position_y);
194 this->n_tiles++;
195
196 this->border_tiles_vec.push_back(hex_ptr);
197
198 for (int i = 1; i < row_width; i++) {
199     x_offset += 2 * first_row_left_tile->minor_radius;
200
201     hex_ptr = new HexTile(
202         x_offset,
203         y_offset,
204         this->event_ptr,
205         this->render_window_ptr,
206         this->assets_manager_ptr,
207         this->message_hub_ptr
208     );
209
210     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
211     this->tile_position_x_vec.push_back(hex_ptr->position_x);
212     this->tile_position_y_vec.push_back(hex_ptr->position_y);
213     this->n_tiles++;
214
215     if (row_width == this->n_layers + 1 or i == row_width - 1) {
216         this->border_tiles_vec.push_back(hex_ptr);
217     }
218 }
219
220 offset_count++;
221 }
222
223 return;
224 } /* __layTiles() */

```

3.4.3.12 __procedurallyGenerateTileResources()

```

void HexMap::__procedurallyGenerateTileResources (
    void ) [private]

```

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

```

718 {
719     // 1. get random cosine series noise vec
720     std::vector<double> noise_vec = this->__getNoise(this->n_tiles);
721
722     // 2. set tile resources based on random cosine series noise
723     int noise_idx = 0;

```

```

724
725     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
726     std::map<double, HexTile*>::iterator hex_map_iter_y;
727     for (
728         hex_map_iter_x = this->hex_map.begin();
729         hex_map_iter_x != this->hex_map.end();
730         hex_map_iter_x++
731     ) {
732         for (
733             hex_map_iter_y = hex_map_iter_x->second.begin();
734             hex_map_iter_y != hex_map_iter_x->second.end();
735             hex_map_iter_y++
736         ) {
737             hex_map_iter_y->second->setTileResource(noise_vec[noise_idx]);
738             noise_idx++;
739         }
740     }
741
742     return;
743 } /* __procedurallyGenerateTileResources() */

```

3.4.3.13 __procedurallyGenerateTileTypes()

```

void HexMap::__procedurallyGenerateTileTypes (
    void ) [private]

```

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

```

343 {
344     // 1. get random cosine series noise vec
345     std::vector<double> noise_vec = this->__getNoise(this->n_tiles);
346
347     // 2. set initial tile types based on either random cosine series noise or white
348     //     noise (decided by coin toss)
349     int noise_idx = 0;
350
351     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
352     std::map<double, HexTile*>::iterator hex_map_iter_y;
353     for (
354         hex_map_iter_x = this->hex_map.begin();
355         hex_map_iter_x != this->hex_map.end();
356         hex_map_iter_x++
357     ) {
358         for (
359             hex_map_iter_y = hex_map_iter_x->second.begin();
360             hex_map_iter_y != hex_map_iter_x->second.end();
361             hex_map_iter_y++
362         ) {
363             if ((double)rand() / RAND_MAX > 0.5) {
364                 hex_map_iter_y->second->setTileType(noise_vec[noise_idx]);
365             }
366             else {
367                 hex_map_iter_y->second->setTileType((double)rand() / RAND_MAX);
368             }
369             noise_idx++;
370         }
371     }
372
373     // 3. smooth tile types (majority rules)
374     this->__smoothTileTypes();
375
376     // 4. set border tile type to ocean
377     for (size_t i = 0; i < this->border_tiles_vec.size(); i++) {
378         this->border_tiles_vec[i]->setTileType(TileType :: OCEAN);
379     }
380
381     // 5. enforce ocean continuity (i.e. all lake tiles touching ocean become ocean)
382     this->__enforceOceanContinuity();
383
384     return;
385 } /* __procedurallyGenerateTileTypes() */

```

3.4.3.14 __sendNoTileSelectedMessage()

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

Helper method to format and send message on no tile selected.

```
922 {
923     Message no_tile_selected_message;
924
925     no_tile_selected_message.channel = NO_TILE_SELECTED_CHANNEL;
926     no_tile_selected_message.subject = "no tile selected";
927
928     this->message_hub_ptr->sendMessage(no_tile_selected_message);
929
930     return;
931 } /* __sendNoTileSelectedMessage() */
```

3.4.3.15 __setUpGlassScreen()

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

Helper method to set up glass screen effect (drawable).

```
34 {
35     this->glass_screen.setSize(sf::Vector2f(GAME_WIDTH, GAME_HEIGHT));
36     this->glass_screen.setFillColor(sf::Color(MONOCROME_SCREEN_BACKGROUND));
37
38     return;
39 } /* __setUpGlassScreen() */
```

3.4.3.16 __smoothTileTypes()

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

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

```
589 {
590     std::cout << "smoothing ..." << std::endl;
591
592     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
593     std::map<double, HexTile*>::iterator hex_map_iter_y;
594     HexTile* hex_ptr;
595     TileType majority_tile_type;
596
597     for (
598         hex_map_iter_x = this->hex_map.begin();
599         hex_map_iter_x != this->hex_map.end();
600         hex_map_iter_x++
601     ) {
602         for (
603             hex_map_iter_y = hex_map_iter_x->second.begin();
604             hex_map_iter_y != hex_map_iter_x->second.end();
605             hex_map_iter_y++
606         ) {
607             hex_ptr = hex_map_iter_y->second;
608             majority_tile_type = this->__getMajorityTileType(hex_ptr);
609
610             if (majority_tile_type != hex_ptr->tile_type) {
611                 hex_ptr->setTileType(majority_tile_type);
612             }
613         }
614     }
615
616     return;
617 } /* __smoothTileTypes() */
```

3.4.3.17 assess()

```
void HexMap::assess (
    void )
```

Method to assess the resource of the selected tile.

```
1024 {
1025     HexTile* selected_tile_ptr = this->__getSelectedTile();
1026     if (selected_tile_ptr != NULL) {
1027         selected_tile_ptr->assess();
1028     }
1029
1030     return;
1031 } /* assess() */
```

3.4.3.18 clear()

```
void HexMap::clear (
    void )
```

Method to clear the hex map.

```
1235 {
1236     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
1237     std::map<double, HexTile*>::iterator hex_map_iter_y;
1238     for (
1239         hex_map_iter_x = this->hex_map.begin();
1240         hex_map_iter_x != this->hex_map.end();
1241         hex_map_iter_x++
1242     ) {
1243         for (
1244             hex_map_iter_y = hex_map_iter_x->second.begin();
1245             hex_map_iter_y != hex_map_iter_x->second.end();
1246             hex_map_iter_y++
1247         ) {
1248             delete hex_map_iter_y->second;
1249         }
1250     }
1251     this->hex_map.clear();
1252
1253     this->tile_position_x_vec.clear();
1254     this->tile_position_y_vec.clear();
1255     this->border_tiles_vec.clear();
1256
1257     return;
1258 } /* clear() */
```

3.4.3.19 draw()

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

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

```
1180 {
1181     // 1. draw background
1182     sf::Color glass_screen_colour = this->glass_screen.getFillColor();
1183     glass_screen_colour.a = 255;
1184     this->glass_screen.setFillColor(glass_screen_colour);
1185
1186     this->render_window_ptr->draw(this->glass_screen);
1187
1188     // 2. draw all tiles in order
1189     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
1190     std::map<double, HexTile*>::iterator hex_map_iter_y;
1191     for (
1192         hex_map_iter_x = this->hex_map.begin();
1193         hex_map_iter_x != this->hex_map.end();
```

```

1194         hex_map_iter_x++
1195     ) {
1196         for (
1197             hex_map_iter_y = hex_map_iter_x->second.begin();
1198             hex_map_iter_y != hex_map_iter_x->second.end();
1199             hex_map_iter_y++
1200         ) {
1201             hex_map_iter_y->second->draw();
1202         }
1203     }
1204
1205     // 3. redraw selected tile
1206     HexTile* selected_tile_ptr = this->__getSelectedTile();
1207     if (selected_tile_ptr != NULL) {
1208         selected_tile_ptr->draw();
1209     }
1210
1211     // 4. draw glass screen
1212     glass_screen_colour = this->glass_screen.getFillColor();
1213     glass_screen_colour.a = 40;
1214     this->glass_screen.setFillColor(glass_screen_colour);
1215
1216     this->render_window_ptr->draw(this->glass_screen);
1217
1218     this->frame++;
1219     return;
1220 } /* draw() */

```

3.4.3.20 processEvent()

```

void HexMap::processEvent (
    void )

```

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

```

1099 {
1100     // 1. process HexTile events
1101     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
1102     std::map<double, HexTile*>::iterator hex_map_iter_y;
1103     for (
1104         hex_map_iter_x = this->hex_map.begin();
1105         hex_map_iter_x != this->hex_map.end();
1106         hex_map_iter_x++
1107     ) {
1108         for (
1109             hex_map_iter_y = hex_map_iter_x->second.begin();
1110             hex_map_iter_y != hex_map_iter_x->second.end();
1111             hex_map_iter_y++
1112         ) {
1113             hex_map_iter_y->second->processEvent();
1114         }
1115     }
1116
1117     // 2. process HexMap events
1118     if (this->event_ptr->type == sf::Event::KeyPressed) {
1119         this->__handleKeyPressEvents();
1120     }
1121
1122     if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
1123         this->__handleMouseButtonEvents();
1124     }
1125
1126     return;
1127 } /* processEvent() */

```

3.4.3.21 processMessage()

```

void HexMap::processMessage (
    void )

```

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

```

1142 {
1143     // 1. process HexTile messages
1144     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
1145     std::map<double, HexTile*>::iterator hex_map_iter_y;
1146     for (
1147         hex_map_iter_x = this->hex_map.begin();
1148         hex_map_iter_x != this->hex_map.end();
1149         hex_map_iter_x++
1150     ) {
1151         for (
1152             hex_map_iter_y = hex_map_iter_x->second.begin();
1153             hex_map_iter_y != hex_map_iter_x->second.end();
1154             hex_map_iter_y++
1155         ) {
1156             hex_map_iter_y->second->processMessage();
1157         }
1158     }
1159
1160     // 2. process HexMap messages
1161     //...
1162
1163     return;
1164 } /* processMessage() */

```

3.4.3.22 reroll()

```

void HexMap::reroll (
    void )

```

Method to re-roll the hex map.

```

1046 {
1047     this->clear();
1048     this->__assembleHexMap();
1049
1050     return;
1051 } /* reroll() */

```

3.4.3.23 toggleResourceOverlay()

```

void HexMap::toggleResourceOverlay (
    void )

```

Method to toggle the hex map resource overlay.

```

1066 {
1067     std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
1068     std::map<double, HexTile*>::iterator hex_map_iter_y;
1069     for (
1070         hex_map_iter_x = this->hex_map.begin();
1071         hex_map_iter_x != this->hex_map.end();
1072         hex_map_iter_x++
1073     ) {
1074         for (
1075             hex_map_iter_y = hex_map_iter_x->second.begin();
1076             hex_map_iter_y != hex_map_iter_x->second.end();
1077             hex_map_iter_y++
1078         ) {
1079             hex_map_iter_y->second->toggleResourceOverlay();
1080         }
1081     }
1082
1083     return;
1084 } /* toggleResourceOverlay() */

```

3.4.4 Member Data Documentation

3.4.4.1 assets_manager_ptr

```
AssetsManager* HexMap::assets_manager_ptr [private]
```

A pointer to the assets manager.

3.4.4.2 border_tiles_vec

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

A vector of pointers to the border tiles.

3.4.4.3 event_ptr

```
sf::Event* HexMap::event_ptr [private]
```

A pointer to the event class.

3.4.4.4 frame

```
int HexMap::frame
```

The current frame of this object.

3.4.4.5 glass_screen

```
sf::RectangleShape HexMap::glass_screen
```

To give the effect of an old glass screen over the hex map.

3.4.4.6 hex_map

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

A position-indexed, nested map of hex tiles.

3.4.4.7 message_hub_ptr

```
MessageHub* HexMap::message_hub_ptr [private]
```

A pointer to the message hub.

3.4.4.8 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

3.4.4.9 n_tiles

```
int HexMap::n_tiles
```

The number of tiles in the hex map.

3.4.4.10 position_x

```
double HexMap::position_x
```

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

3.4.4.11 position_y

```
double HexMap::position_y
```

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

3.4.4.12 render_window_ptr

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

A pointer to the render window.

3.4.4.13 tile_position_x_vec

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

A vector of tile x positions.

3.4.4.14 tile_position_y_vec

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

A vector of tile y position.

3.4.4.15 tile_selected

```
bool HexMap::tile_selected
```

A boolean which indicates if a tile is currently selected.

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

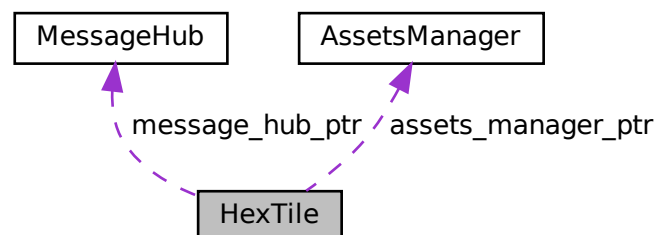
- header/[HexMap.h](#)
- source/[HexMap.cpp](#)

3.5 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, sf::Event *, sf::RenderWindow *, [AssetsManager](#) *, [MessageHub](#) *)
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 [processEvent](#) (void)
Method to process [HexTile](#). To be called once per event.
- void [processMessage](#) (void)
Method to process [HexTile](#). To be called once per message.
- void [draw](#) (void)
Method to draw the hex tile to the render window. To be called 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.
- bool [is_selected](#)
A boolean which indicates whether or not the tile is selected.
- bool [has_improvement](#)
A boolean which indicates if tile has improvement or not.
- 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::ConvexShape [select_outline_sprite](#)

A convex shape which outlines the tile when selected.
- 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 [__setUpSelectOutlineSprite](#) (void)
- Helper method to set up select outline sprite.*
- void [__setUpResourceChipSprite](#) (void)
- Helper method to set up resource chip sprite.*
- void [__setResourceText](#) (void)
- Helper method to set up resource text.*
- bool [__isClicked](#) (void)
- Helper method to determine if tile was clicked on.*
- void [__handleKeyPressEvents](#) (void)
- Helper method to handle key press events.*
- void [__handleMouseButtonEvents](#) (void)
- Helper method to handle mouse button events.*
- void [__sendTileSelectedMessage](#) (void)
- Helper method to format and send message on tile selection.*
- std::string [__getTileCoordsSubstring](#) (void)
- Helper method to assemble and return tile coordinates substring.*
- std::string [__getTileTypeSubstring](#) (void)
- Helper method to assemble and return tile type substring.*
- std::string [__getTileResourceSubstring](#) (void)
- Helper method to assemble and return tile resource substring.*
- std::string [__getTileImprovementSubstring](#) (void)
- void [__sendTileStateMessage](#) (void)
- Helper method to format and send tile state message.*

Private Attributes

- sf::Event * [event_ptr](#)
- A pointer to the event class.*
- sf::RenderWindow * [render_window_ptr](#)
- A pointer to the render window.*
- [AssetsManager](#) * [assets_manager_ptr](#)
- A pointer to the assets manager.*
- [MessageHub](#) * [message_hub_ptr](#)
- A pointer to the message hub.*

3.5.1 Detailed Description

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

3.5.2 Constructor & Destructor Documentation

3.5.2.1 HexTile()

```
HexTile::HexTile (
    double position_x,
    double position_y,
    sf::Event * event_ptr,
    sf::RenderWindow * render_window_ptr,
    AssetsManager * assets_manager_ptr,
    MessageHub * message_hub_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>event_ptr</i>	Pointer to the event class.
<i>render_window_ptr</i>	Pointer to the render window.
<i>assets_manager_ptr</i>	Pointer to the assets manager.
<i>message_hub_ptr</i>	Pointer to the message hub.

```
606 {
607     // 1. set attributes
608
609     // 1.1. private
610     this->event_ptr = event_ptr;
611     this->render_window_ptr = render_window_ptr;
612
613     this->assets_manager_ptr = assets_manager_ptr;
614     this->message_hub_ptr = message_hub_ptr;
615
616     // 1.2. public
617     this->show_node = false;
618     this->show_resource = false;
619     this->resource_assessed = false;
620     this->is_selected = false;
621     this->has_improvement = false;
622
623     this->frame = 0;
624
625     this->position_x = position_x;
626     this->position_y = position_y;
627
628     this->major_radius = 32;
629     this->minor_radius = (sqrt(3) / 2) * this->major_radius;
630
631     // 2. set up and position drawable attributes
632     this->__setUpNodeSprite();
633     this->__setUpTileSprite();
634     this->__setUpSelectOutlineSprite();
635     this->__setUpResourceChipSprite();
636     this->__setUpResourceText();
```

```

637
638 // 3. set tile type and resource (default to forest and average)
639 this->setTileType(TileType :: FOREST);
640 this->setTileResource(TileResource :: AVERAGE);
641
642 std::cout << "HexTile constructed at " << this << std::endl;
643
644 return;
645 } /* HexTile() */

```

3.5.2.2 ~HexTile()

```

HexTile::~HexTile (
    void )

```

Destructor for the [HexTile](#) class.

```

975 {
976     std::cout << "HexTile at " << this << " destroyed" << std::endl;
977
978     return;
979 } /* ~HexTile() */

```

3.5.3 Member Function Documentation

3.5.3.1 __getTileCoordsSubstring()

```

std::string HexTile::__getTileCoordsSubstring (
    void ) [private]

```

Helper method to assemble and return tile coordinates substring.

Returns

Tile coordinates substring.

```

375 {
376     std::string coords_substring = "TILE COORDS: (";
377     coords_substring += std::to_string(int(this->position_x - 400));
378     coords_substring += ", ";
379     coords_substring += std::to_string(int(this->position_y - 400));
380     coords_substring += ")\n";
381
382     return coords_substring;
383 } /* __getTileCoordsSubstring() */

```

3.5.3.2 __getTileImprovementSubstring()

```

std::string HexTile::__getTileImprovementSubstring (
    void ) [private]

```

```

497 {
498     std::string improvement_substring = "TILE IMPROVEMENT: ";
499
500     if (this->has_improvement) {
501         //...
502     }
503
504     else {
505         improvement_substring += "NONE\n";
506     }
507
508     return improvement_substring;
509 } /* __getTileImprovementSubstring() */

```

3.5.3.3 __getTileResourceSubstring()

```
std::string HexTile::__getTileResourceSubstring (
    void ) [private]
```

Helper method to assemble and return tile resource substring.

Returns

Tile resource substring.

```
464 {
465     std::string resource_substring = "TILE RESOURCE:      ";
466
467     if (this->resource_assessed) {
468         switch (this->tile_resource) {
469             //...
470
471             default: {
472                 resource_substring += "???\n";
473                 break;
474             }
475         }
476     }
477 }
478
479 else {
480     resource_substring += "[A]: ASSESS\n";
481 }
482
483 return resource_substring;
484 }
485 /* __getTileResourceSubstring() */
```

3.5.3.4 __getTileTypeSubstring()

```
std::string HexTile::__getTileTypeSubstring (
    void ) [private]
```

Helper method to assemble and return tile type substring.

Returns

Tile type substring.

```
400 {
401     std::string type_substring = "TILE TYPE:          ";
402
403     switch (this->tile_type) {
404         case (TileType :: FOREST): {
405             type_substring += "FOREST\n";
406             break;
407         }
408
409         case (TileType :: LAKE): {
410             type_substring += "LAKE\n";
411             break;
412         }
413
414         case (TileType :: MOUNTAINS): {
415             type_substring += "MOUNTAINS\n";
416             break;
417         }
418
419         case (TileType :: OCEAN): {
420
421
422
423
424
425
```

```

426         type_substring += "OCEAN\n";
427
428         break;
429     }
430
431
432     case (TileType :: PLAINS): {
433         type_substring += "PLAINS\n";
434
435         break;
436     }
437
438
439     default: {
440         type_substring += "???\n";
441
442         break;
443     }
444 }
445
446 return type_substring;
447 } /* __getTileTypeSubstring() */

```

3.5.3.5 __handleKeyPressEvents()

```

void HexTile::__handleKeyPressEvents (
    void ) [private]

```

Helper method to handle key press events.

```

268 {
269     switch (this->event_ptr->key.code) {
270         case (sf::Keyboard::Escape): {
271             this->is_selected = false;
272         }
273
274
275         default: {
276             // do nothing!
277
278             break;
279         }
280     }
281
282     return;
283 } /* __handleKeyPressEvents() */

```

3.5.3.6 __handleMouseButtonEvents()

```

void HexTile::__handleMouseButtonEvents (
    void ) [private]

```

Helper method to handle mouse button events.

```

298 {
299     switch (this->event_ptr->mouseButton.button) {
300         case (sf::Mouse::Left): {
301             if (this->__isClicked()) {
302                 std::cout << "Tile (" << this->position_x << ", " <<
303                     this->position_y << ") was selected" << std::endl;
304
305                 this->is_selected = true;
306
307                 this->__sendTileSelectedMessage();
308                 this->__sendTileStateMessage();
309             }
310
311             else {
312                 this->is_selected = false;
313             }
314

```



```

315         break;
316     }
317
318
319     case (sf::Mouse::Right): {
320         this->is_selected = false;
321         break;
322     }
323
324
325
326     default: {
327         // do nothing!
328         break;
329     }
330 }
331 }
332
333 return;
334 } /* __handleMouseButtonEvents() */

```

3.5.3.7 __isClicked()

```

bool HexTile::__isClicked (
    void ) [private]

```

Helper method to determine if tile was clicked on.

Returns

Boolean indicating whether or not tile was clicked on.

```

236 {
237     sf::Vector2i mouse_position = sf::Mouse::getPosition(*render_window_ptr);
238
239     double mouse_x = mouse_position.x;
240     double mouse_y = mouse_position.y;
241
242     double distance = sqrt(
243         pow(this->position_x - mouse_x, 2) +
244         pow(this->position_y - mouse_y, 2)
245     );
246
247     if (distance < this->minor_radius) {
248         return true;
249     }
250     else {
251         return false;
252     }
253 } /* __isClicked() */

```

3.5.3.8 __sendTileSelectedMessage()

```

void HexTile::__sendTileSelectedMessage (
    void ) [private]

```

Helper method to format and send message on tile selection.

```

349 {
350     Message tile_selected_message;
351
352     tile_selected_message.channel = TILE_SELECTED_CHANNEL;
353     tile_selected_message.subject = "tile selected";
354
355     this->message_hub_ptr->sendMessage(tile_selected_message);
356
357     return;
358 } /* __sendTileSelectedMessage() */

```

3.5.3.9 __sendTileStateMessage()

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

Helper method to format and send tile state message.

```
524 {
525     Message tile_state_message;
526
527     tile_state_message.channel = TILE_STATE_CHANNEL;
528     tile_state_message.subject = "tile state";
529
530
531     //          32 char x 17 line console "-----\n";
532     std::string string_payload      = "    **** TILE INFO/OPTIONS **** \n";
533     string_payload                  += " \n";
534
535     string_payload                  += this->__getTileCoordsSubstring();
536     string_payload                  += " \n";
537
538     string_payload                  += this->__getTileTypeSubstring();
539     string_payload                  += this->__getTileResourceSubstring();
540     string_payload                  += this->__getTileImprovementSubstring();
541
542     string_payload                  += " \n";
543     string_payload                  += " \n";
544     string_payload                  += " \n";
545     string_payload                  += " \n";
546     string_payload                  += " \n";
547     string_payload                  += " \n";
548     string_payload                  += " \n";
549     string_payload                  += " \n";
550     string_payload                  += " \n";
551     string_payload                  += " ";
552
553
554     tile_state_message.string_payload = string_payload;
555
556     this->message_hub_ptr->sendMessage(tile_state_message);
557
558     return;
559 } /* __sendTileStateMessage() */
```

3.5.3.10 __setResourceText()

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

Helper method to set up resource text.

```
159 {
160     this->resource_text.setFont(*(assets_manager_ptr->getFont("DroidSansMono")));
161
162     switch (this->tile_resource) {
163     case (TileResource :: POOR): {
164         this->resource_text.setString("-2");
165
166         break;
167     }
168
169     case (TileResource :: BELOW_AVERAGE): {
170         this->resource_text.setString("-1");
171
172         break;
173     }
174
175     case (TileResource :: AVERAGE): {
176         this->resource_text.setString("0");
177
178         break;
179     }
180
181     case (TileResource :: ABOVE_AVERAGE): {
182         this->resource_text.setString("+1");
183
184         break;
185     }
186 }
```

```

185         }
186
187         case (TileResource :: GOOD): {
188             this->resource_text.setString("+2");
189
190             break;
191         }
192
193         default: {
194             this->resource_text.setString("?");
195
196             break;
197         }
198     }
199
200     if (not this->resource_assessed) {
201         this->resource_text.setString("?");
202     }
203
204     this->resource_text.setCharacterSize(16);
205
206     this->resource_text.setOrigin(
207         this->resource_text.getLocalBounds().width / 2,
208         this->resource_text.getLocalBounds().height / 2
209     );
210
211     this->resource_text.setFillColor(sf::Color(0, 0, 0, 255));
212
213     this->resource_text.setPosition(
214         this->position_x,
215         this->position_y - 4
216     );
217
218     return;
219 } /* __setResourceText() */

```

3.5.3.11 __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() */

```

3.5.3.12 __setUpResourceChipSprite()

```

void HexTile::__setUpResourceChipSprite (
    void ) [private]

```

Helper method to set up resource chip sprite.

```

132 {
133     this->resource_chip_sprite.setRadius(2 * this->minor_radius / 3);
134
135     this->resource_chip_sprite.setOrigin(
136         this->resource_chip_sprite.getLocalBounds().width / 2,
137         this->resource_chip_sprite.getLocalBounds().height / 2

```

```

138     );
139
140     this->resource_chip_sprite.setPosition(this->position_x, this->position_y);
141
142     this->resource_chip_sprite.setFillColor(sf::Color(175, 175, 175, 175));
143
144     return;
145 } /* __setUpResourceChip() */

```

3.5.3.13 __setUpSelectOutlineSprite()

```

void HexTile::__setUpSelectOutlineSprite (
    void ) [private]

```

Helper method to set up select outline sprite.

```

96 {
97     int n_points = 6;
98
99     this->select_outline_sprite.setPointCount(n_points);
100
101     for (int i = 0; i < n_points; i++) {
102         this->select_outline_sprite.setPoint(
103             i,
104             sf::Vector2f(
105                 this->position_x + this->major_radius * cos((30 + 60 * i) * (M_PI / 180)),
106                 this->position_y + this->major_radius * sin((30 + 60 * i) * (M_PI / 180))
107             )
108         );
109     }
110
111     this->select_outline_sprite.setOutlineThickness(4);
112     this->select_outline_sprite.setOutlineColor(MONOCROME_TEXT_RED);
113
114     this->select_outline_sprite.setFillColor(sf::Color(0, 0, 0, 0));
115
116     return;
117 } /* __setUpSelectOutline() */

```

3.5.3.14 __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() */

```

3.5.3.15 assess()

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

Method to assess the tile's resource.

```
866 {
867     this->resource_assessed = true;
868     this->__setResourceText ();
869
870     return;
871 } /* assess() */
```

3.5.3.16 draw()

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

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

```
931 {
932     // 1. draw hex
933     this->render_window_ptr->draw(this->tile_sprite);
934
935     // 2. draw node
936     if (this->show_node) {
937         this->render_window_ptr->draw(this->node_sprite);
938     }
939
940     // 3. draw resource
941     if (this->show_resource) {
942         this->render_window_ptr->draw(this->resource_chip_sprite);
943         this->render_window_ptr->draw(this->resource_text);
944     }
945
946     // 4. draw selection outline
947     if (this->is_selected) {
948         sf::Color outline_colour = this->select_outline_sprite.getOutlineColor();
949
950         outline_colour.a =
951             255 * pow(cos((M_PI * this->frame) / (1.5 * FRAMES_PER_SECOND)), 2);
952
953         this->select_outline_sprite.setOutlineColor(outline_colour);
954
955         this->render_window_ptr->draw(this->select_outline_sprite);
956     }
957
958     this->frame++;
959     return;
960 } /* draw() */
```

3.5.3.17 processEvent()

```
void HexTile::processEvent (
    void )
```

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

```
886 {
887     if (this->event_ptr->type == sf::Event::KeyPressed) {
888         this->__handleKeyPressEvents();
889     }
890
891     if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
892         this->__handleMouseButtonEvents();
893     }
894
895     return;
896 } /* processEvent() */
```

3.5.3.18 processMessage()

```
void HexTile::processMessage (
    void )
```

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

```
911 {
912     //...
913
914     return;
915 } /* processMessage() */
```

3.5.3.19 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].
--------------------	--

```
791 {
792     // 1. check input
793     if (input_value < 0 or input_value > 1) {
794         std::string error_str = "ERROR HexTile::setTileResource() given input value is ";
795         error_str += "not in the closed interval [0, 1]";
796
797         #ifdef _WIN32
798             std::cout << error_str << std::endl;
799         #endif /* _WIN32 */
800
801         throw std::runtime_error(error_str);
802     }
803
804     // 2. convert input value to tile resource
805     TileResource tile_resource;
806
807     if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[0]) {
808         tile_resource = TileResource :: POOR;
809     }
810     else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[1]) {
811         tile_resource = TileResource :: BELOW_AVERAGE;
812     }
813     else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[2]) {
814         tile_resource = TileResource :: AVERAGE;
815     }
816     else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[3]) {
817         tile_resource = TileResource :: ABOVE_AVERAGE;
818     }
819     else {
820         tile_resource = TileResource :: GOOD;
821     }
822
823     // 3. call alternate method
824     this->setTileResource(tile_resource);
825
826     return;
827 } /* setTileResource(double) */
```

3.5.3.20 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.
----------------------	---

```

769 {
770     this->tile_resource = tile_resource;
771     this->__setResourceText();
772
773     return;
774 } /* setTileResource(TileResource) */

```

3.5.3.21 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].
--------------------	--

```

719 {
720     // 1. check input
721     if (input_value < 0 or input_value > 1) {
722         std::string error_str = "ERROR HexTile::setTileType() given input value is ";
723         error_str += "not in the closed interval [0, 1]";
724
725         #ifdef _WIN32
726             std::cout << error_str << std::endl;
727         #endif /* _WIN32 */
728
729         throw std::runtime_error(error_str);
730     }
731
732     // 2. convert input value to tile type
733     TileType tile_type;
734
735     if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[0]) {
736         tile_type = TileType :: LAKE;
737     }
738     else if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[1]) {
739         tile_type = TileType :: PLAINS;
740     }
741     else if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[2]) {
742         tile_type = TileType :: FOREST;
743     }
744     else {
745         tile_type = TileType :: MOUNTAINS;
746     }
747
748     // 3. call alternate method
749     this->setTileType(tile_type);
750
751     return;
752 } /* setTileType(double) */

```

3.5.3.22 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.
------------------	---

```

660 {
661     this->tile_type = tile_type;
662
663     switch (this->tile_type) {
664         case (TileType :: FOREST): {
665             this->tile_sprite.setFillColor(FOREST_GREEN);
666
667             break;
668         }
669
670         case (TileType :: LAKE): {
671             this->tile_sprite.setFillColor(LAKE_BLUE);
672
673             break;
674         }
675
676         case (TileType :: MOUNTAINS): {
677             this->tile_sprite.setFillColor(MOUNTAINS_GREY);
678
679             break;
680         }
681
682         case (TileType :: OCEAN): {
683             this->tile_sprite.setFillColor(OCEAN_BLUE);
684
685             break;
686         }
687
688         case (TileType :: PLAINS): {
689             this->tile_sprite.setFillColor(PLAINS_YELLOW);
690
691             break;
692         }
693
694         default: {
695             // do nothing!
696
697             break;
698         }
699     }
700
701     return;
702 } /* setTileType(TileType) */

```

3.5.3.23 toggleResourceOverlay()

```

void HexTile::toggleResourceOverlay (
    void )

```

Method to toggle the tile resource overlay.

```

842 {
843     if (this->show_resource) {
844         this->show_resource = false;
845     }
846     else {
847         this->show_resource = true;
848     }
849
850     return;
851 } /* toggleResourceOverlay() */

```

3.5.4 Member Data Documentation

3.5.4.1 assets_manager_ptr

```
AssetsManager* HexTile::assets_manager_ptr [private]
```

A pointer to the assets manager.

3.5.4.2 event_ptr

```
sf::Event* HexTile::event_ptr [private]
```

A pointer to the event class.

3.5.4.3 frame

```
int HexTile::frame
```

The current frame of this object.

3.5.4.4 has_improvement

```
bool HexTile::has_improvement
```

A boolean which indicates if tile has improvement or not.

3.5.4.5 is_selected

```
bool HexTile::is_selected
```

A boolean which indicates whether or not the tile is selected.

3.5.4.6 major_radius

```
double HexTile::major_radius
```

The radius of the smallest bounding circle.

3.5.4.7 message_hub_ptr

```
MessageHub* HexTile::message_hub_ptr [private]
```

A pointer to the message hub.

3.5.4.8 minor_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

3.5.4.9 node_sprite

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

A circle shape to mark the tile node.

3.5.4.10 position_x

```
double HexTile::position_x
```

The x position of the tile.

3.5.4.11 position_y

```
double HexTile::position_y
```

The y position of the tile.

3.5.4.12 render_window_ptr

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

A pointer to the render window.

3.5.4.13 resource_assessed

```
bool HexTile::resource_assessed
```

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

3.5.4.14 resource_chip_sprite

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

A circle shape which represents a resource chip.

3.5.4.15 resource_text

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

A text representation of the resource.

3.5.4.16 select_outline_sprite

```
sf::ConvexShape HexTile::select_outline_sprite
```

A convex shape which outlines the tile when selected.

3.5.4.17 show_node

```
bool HexTile::show_node
```

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

3.5.4.18 show_resource

```
bool HexTile::show_resource
```

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

3.5.4.19 tile_resource

`TileResource` HexTile::tile_resource

3.5.4.20 tile_sprite

`sf::ConvexShape` HexTile::tile_sprite

A convex shape which represents the tile.

3.5.4.21 tile_type

`TileType` HexTile::tile_type

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

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

3.6 Message Struct Reference

A structure which defines a standard message format.

```
#include <MessageHub.h>
```

Public Attributes

- `std::string` `channel` = ""
A string identifying the appropriate channel for this message.
- `std::string` `subject` = ""
A string describing the message subject.
- `std::vector< bool >` `bool_payload_vec` = {}
A vector <bool> payload.
- `std::vector< int >` `int_payload_vec` = {}
A vector <int> payload.
- `std::vector< double >` `double_payload_vec` = {}
A vector <double> payload.
- `std::string` `string_payload` = ""
A string payload.

3.6.1 Detailed Description

A structure which defines a standard message format.

3.6.2 Member Data Documentation

3.6.2.1 bool_payload_vec

```
std::vector<bool> Message::bool_payload_vec = {}
```

A vector <bool> payload.

3.6.2.2 channel

```
std::string Message::channel = ""
```

A string identifying the appropriate channel for this message.

3.6.2.3 double_payload_vec

```
std::vector<double> Message::double_payload_vec = {}
```

A vector <double> payload.

3.6.2.4 int_payload_vec

```
std::vector<int> Message::int_payload_vec = {}
```

A vector <int> payload.

3.6.2.5 string_payload

```
std::string Message::string_payload = ""
```

A string payload.

3.6.2.6 subject

```
std::string Message::subject = ""
```

A string describing the message subject.

The documentation for this struct was generated from the following file:

- header/ESC_core/[MessageHub.h](#)

3.7 MessageHub Class Reference

A class which acts as a central hub for inter-object message traffic.

```
#include <MessageHub.h>
```

Public Member Functions

- [MessageHub](#) (void)
Constructor for the [MessageHub](#) class.
- bool [hasTraffic](#) (void)
Method to determine if there remains any message traffic.
- void [addChannel](#) (std::string)
Method to add channel to message map.
- void [removeChannel](#) (std::string)
Method to remove channel from message map.
- void [sendMessage](#) ([Message](#))
Method to send a message to the message map.
- bool [isEmpty](#) (std::string)
Method to check if channel is empty.
- [Message](#) [receiveMessage](#) (std::string)
Method to receive the latest message in the given channel.
- void [popMessage](#) (std::string)
Method to pop latest message off of the given channel.
- void [clearMessages](#) (void)
Method to clear messages from the [MessageHub](#).
- void [clear](#) (void)
Method to clear the [MessageHub](#).
- [~MessageHub](#) (void)
Destructor for the [MessageHub](#) class.

Private Attributes

- std::map< std::string, std::list< [Message](#) > > [message_map](#)
A map <string, list of [Message](#)> for sending and receiving messages. Here the key is the channel, and each channel maintains a list (history) of messages.

3.7.1 Detailed Description

A class which acts as a central hub for inter-object message traffic.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 MessageHub()

```
MessageHub::MessageHub (
    void )
```

Constructor for the [MessageHub](#) class.

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

3.7.2.2 ~MessageHub()

```
MessageHub::~MessageHub (
    void )
```

Destructor for the [MessageHub](#) class.

```
386 {
387     this->clear();
388
389     std::cout << "MessageHub at " << this << " destroyed" << std::endl;
390
391     return;
392 } /* ~MessageHub() */
```

3.7.3 Member Function Documentation

3.7.3.1 addChannel()

```
void MessageHub::addChannel (
    std::string channel )
```

Method to add channel to message map.

Parameters

<i>channel</i>	The key for the message channel being added.
----------------	--

```

97 {
98     // 1. check if channel is in map (if so, throw error)
99     if (this->message_map.count(channel) > 0) {
100         std::string error_str = "ERROR MessageHub::addChannel() channel ";
101         error_str += channel;
102         error_str += " is already in message map";
103
104         #ifdef _WIN32
105             std::cout << error_str << std::endl;
106         #endif /* _WIN32 */
107
108         throw std::runtime_error(error_str);
109     }
110
111     // 2. add channel to map
112     this->message_map[channel] = {};
113
114     return;
115 } /* addChannel() */

```

3.7.3.2 clear()

```

void MessageHub::clear (
    void )

```

Method to clear the [MessageHub](#).

```

366 {
367
368     this->clearMessages();
369     this->message_map.clear();
370
371     return;
372 } /* clear() */

```

3.7.3.3 clearMessages()

```

void MessageHub::clearMessages (
    void )

```

Method to clear messages from the [MessageHub](#).

```

340 {
341     std::map<std::string, std::list<Message>::iterator map_iter;
342     for (
343         map_iter = this->message_map.begin();
344         map_iter != this->message_map.end();
345         map_iter++
346     ) {
347         map_iter->second.clear();
348     }
349
350     return;
351 } /* clearMessages() */

```


3.7.3.4 hasTraffic()

```
bool MessageHub::hasTraffic (
    void )
```

Method to determine if there remains any message traffic.

```
67 {
68     std::map<std::string, std::list<Message>::iterator map_iter;
69     for (
70         map_iter = this->message_map.begin();
71         map_iter != this->message_map.end();
72         map_iter++
73     ) {
74         if (not map_iter->second.empty()) {
75             return true;
76         }
77     }
78     return false;
79 }
80 } /* hasTraffic() */
```

3.7.3.5 isEmpty()

```
bool MessageHub::isEmpty (
    std::string channel )
```

Method to check if channel is empty.

Parameters

<i>channel</i>	The key for the message channel being checked.
----------------	--

Returns

A boolean indicating whether the channel is empty or not.

```
207 {
208     // 1. check if channel is in map (if not, throw error)
209     if (this->message_map.count(channel) <= 0) {
210         std::string error_str = "ERROR MessageHub::isEmpty() channel ";
211         error_str += channel;
212         error_str += " is not in message map";
213
214         #ifdef _WIN32
215             std::cout << error_str << std::endl;
216         #endif /* _WIN32 */
217
218         throw std::runtime_error(error_str);
219     }
220
221     if (this->message_map[channel].empty()) {
222         return true;
223     }
224     else {
225         return false;
226     }
227 } /* isEmpty() */
```

3.7.3.6 popMessage()

```
void MessageHub::popMessage (
    std::string channel )
```

Method to pop latest message off of the given channel.

Parameters

<i>channel</i>	The key for the message channel being popped.
----------------	---

```

294 {
295     // 1. check if channel is in map (if not, throw error)
296     if (this->message_map.count(channel) <= 0) {
297         std::string error_str = "ERROR MessageHub::receiveMessage() channel ";
298         error_str += channel;
299         error_str += " is not in message map";
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. check if channel is empty (if so, throw error)
309     if (this->message_map[channel].empty()) {
310         std::string error_str = "ERROR MessageHub::receiveMessage() channel ";
311         error_str += channel;
312         error_str += " is empty";
313
314         #ifdef _WIN32
315             std::cout << error_str << std::endl;
316         #endif /* _WIN32 */
317
318         throw std::runtime_error(error_str);
319     }
320
321     // 3. pop message
322     this->message_map[channel].pop_back();
323
324     return;
325 } /* popMessage() */

```

3.7.3.7 receiveMessage()

```

Message MessageHub::receiveMessage (
    std::string channel )

```

Method to receive the latest message in the given channel.

Parameters

<i>channel</i>	The key for the message channel being received from.
----------------	--

Returns

The latest message in the given channel.

```

246 {
247     // 1. check if channel is in map (if not, throw error)
248     if (this->message_map.count(channel) <= 0) {
249         std::string error_str = "ERROR MessageHub::receiveMessage() channel ";
250         error_str += channel;
251         error_str += " is not in message map";
252
253         #ifdef _WIN32
254             std::cout << error_str << std::endl;
255         #endif /* _WIN32 */
256
257         throw std::runtime_error(error_str);
258     }
259
260     // 2. check if channel is empty (if so, throw error)
261     if (this->message_map[channel].empty()) {
262         std::string error_str = "ERROR MessageHub::receiveMessage() channel ";

```

```

263         error_str += channel;
264         error_str += " is empty";
265
266         #ifdef _WIN32
267             std::cout << error_str << std::endl;
268         #endif /* _WIN32 */
269         throw std::runtime_error(error_str);
270     }
271
272
273     // 3. receive message
274     Message message = this->message_map[channel].back();
275
276     return message;
277 } /* receiveMessage() */

```

3.7.3.8 removeChannel()

```

void MessageHub::removeChannel (
    std::string channel )

```

Method to remove channel from message map.

Parameters

<i>channel</i>	The key for the message channel being removed.
----------------	--

```

132 {
133     // 1. check if channel is in map (if not, throw error)
134     if (this->message_map.count(channel) <= 0) {
135         std::string error_str = "ERROR MessageHub::removeChannel() channel ";
136         error_str += channel;
137         error_str += " is not in message map";
138
139         #ifdef _WIN32
140             std::cout << error_str << std::endl;
141         #endif /* _WIN32 */
142         throw std::runtime_error(error_str);
143     }
144
145     // 2. remove channel from map
146     this->message_map[channel].clear();
147     this->message_map.erase(channel);
148
149     return;
150 } /* removeChannel() */

```

3.7.3.9 sendMessage()

```

void MessageHub::sendMessage (
    Message message )

```

Method to send a message to the message map.

Parameters

<i>message</i>	The message to be sent.
----------------	-------------------------

```

168 {
169     // 1. check if channel is in map (if not, throw error)
170     std::string channel = message.channel;

```

```
171
172     if (this->message_map.count(channel) <= 0) {
173         std::string error_str = "ERROR MessageHub::sendMessage() channel ";
174         error_str += channel;
175         error_str += " is not in message map";
176
177         #ifdef _WIN32
178             std::cout << error_str << std::endl;
179         #endif /* _WIN32 */
180
181         throw std::runtime_error(error_str);
182     }
183
184     // 2. send message to message map
185     this->message_map[channel].push_back(message);
186
187     return;
188 } /* sendMessage() */
```

3.7.4 Member Data Documentation

3.7.4.1 message_map

```
std::map<std::string, std::list<Message> > MessageHub::message_map [private]
```

A map <string, list of [Message](#)> for sending and receiving messages. Here the key is the channel, and each channel maintains a list (history) of messages.

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

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

Chapter 4

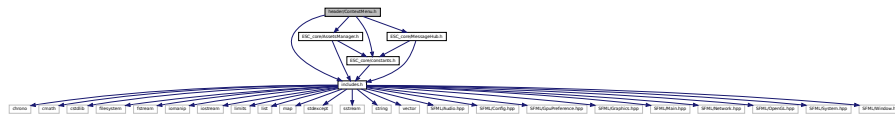
File Documentation

4.1 header/ContextMenu.h File Reference

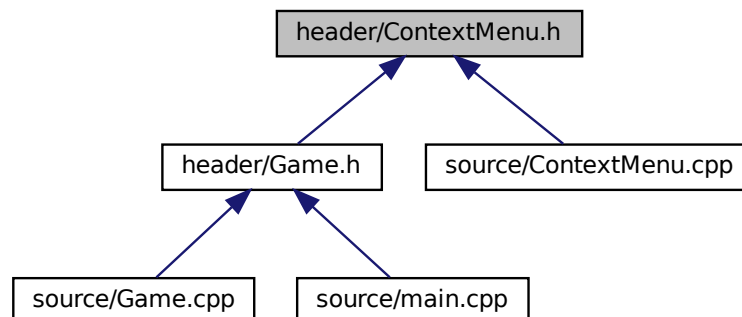
Header file for the [ContextMenu](#) class.

```
#include "ESC_core/constants.h"
#include "ESC_core/includes.h"
#include "ESC_core/AssetsManager.h"
#include "ESC_core/MessageHub.h"
```

Include dependency graph for ContextMenu.h:



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



Classes

- class [ContextMenu](#)

A class which defines a context menu for the game.

Enumerations

- enum [ConsoleState](#) {
[NONE](#) , [READY](#) , [MENU](#) , [TILE](#) ,
[N_CONSOLE_STATES](#) }

An enumeration of the different console screen states.

4.1.1 Detailed Description

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

4.1.2 Enumeration Type Documentation

4.1.2.1 ConsoleState

enum [ConsoleState](#)

An enumeration of the different console screen states.

Enumerator

NONE	None state (for initialization)
READY	Ready (default) state.
MENU	Game menu state.
TILE	Tile context state.
N_CONSOLE_STATES	A simple hack to get the number of console screen states.

```

34         {
35     NONE,
36     READY,
37     MENU,
38     TILE,
39     N_CONSOLE_STATES
40 };

```

4.2 header/ESC_core/AssetsManager.h File Reference

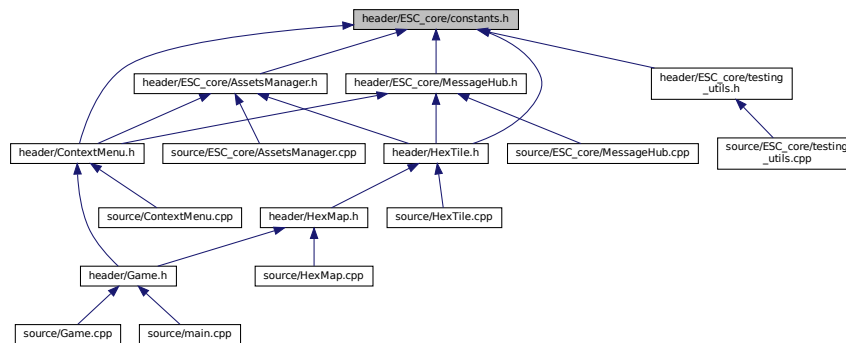
Header file for the [AssetsManager](#) class.

```

#include "constants.h"
#include "includes.h"

```


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



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.
- const sf::Color **MENU_FRAME_GREY** (185, 187, 182)
The base colour of the context menu frame.
- const sf::Color **MONOCHROME_SCREEN_BACKGROUND** (40, 40, 40)
The base colour of old monochrome screens.
- const sf::Color **VISUAL_SCREEN_FRAME_GREY** (151, 151, 143)
The base colour of the framing of the visual screen.
- const sf::Color **MONOCHROME_TEXT_GREEN** (0, 255, 102)
The base colour of old monochrome text (green).
- const sf::Color **MONOCHROME_TEXT_AMBER** (255, 176, 0)
The base colour of old monochrome text (amber).
- const sf::Color **MONOCHROME_TEXT_RED** (255, 44, 0)
The base colour of old monochrome text (red).

Variables

- const double **FLOAT_TOLERANCE** = 1e-6
Tolerance for floating point equality tests.
- const unsigned long long int **SECONDS_PER_YEAR** = 31537970
- const unsigned long long int **SECONDS_PER_MONTH** = 2628164
- 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).*
- const int `GAME_WIDTH` = 1200
Width of the game space.
- const int `GAME_HEIGHT` = 800
Height of the game space.
- const std::vector< double > `TILE_TYPE_CUMULATIVE_PROBABILITIES`
Cumulative probabilities for each tile type (to support procedural generation).
- const std::vector< double > `TILE_RESOURCE_CUMULATIVE_PROBABILITIES`
Cumulative probabilities for each tile resource (to support procedural generation).
- const std::string `TILE_SELECTED_CHANNEL` = "TILE SELECTED CHANNEL"
A message channel for tile selection messages.
- const std::string `NO_TILE_SELECTED_CHANNEL` = "NO TILE SELECTED CHANNEL"
A message channel for no tile selected messages.
- const std::string `TILE_STATE_CHANNEL` = "TILE STATE CHANNEL"
A message channel for tile state messages.
- const std::string `GAME_CHANNEL` = "GAME CHANNEL"
A message channel for game messages.

4.3.1 Detailed Description

Header file for various constants.

4.3.2 Function Documentation

4.3.2.1 FOREST_GREEN()

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

The base colour of a forest tile.

4.3.2.2 LAKE_BLUE()

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

The base colour of a lake (water) tile.

4.3.2.3 MENU_FRAME_GREY()

```
const sf::Color MENU_FRAME_GREY (
    185 ,
    187 ,
    182 )
```

The base colour of the context menu frame.

4.3.2.4 MONOCHROME_SCREEN_BACKGROUND()

```
const sf::Color MONOCHROME_SCREEN_BACKGROUND (
    40 ,
    40 ,
    40 )
```

The base colour of old monochrome screens.

4.3.2.5 MONOCHROME_TEXT_AMBER()

```
const sf::Color MONOCHROME_TEXT_AMBER (
    255 ,
    176 ,
    0 )
```

The base colour of old monochrome text (amber).

4.3.2.6 MONOCHROME_TEXT_GREEN()

```
const sf::Color MONOCHROME_TEXT_GREEN (
    0 ,
    255 ,
    102 )
```

The base colour of old monochrome text (green).

4.3.2.7 MONOCHROME_TEXT_RED()

```
const sf::Color MONOCHROME_TEXT_RED (
    255 ,
    44 ,
    0 )
```

The base colour of old monochrome text (red).

4.3.2.8 MOUNTAINS_GREY()

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

The base colour of a mountains tile.

4.3.2.9 OCEAN_BLUE()

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

The base colour of an ocean (water) tile.

4.3.2.10 PLAINS_YELLOW()

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

The base colour of a plains tile.

4.3.2.11 VISUAL_SCREEN_FRAME_GREY()

```
const sf::Color VISUAL_SCREEN_FRAME_GREY (
    151 ,
    151 ,
    143 )
```

The base colour of the framing of the visual screen.

4.3.3 Variable Documentation

4.3.3.1 FLOAT_TOLERANCE

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

Tolerance for floating point equality tests.

4.3.3.2 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

4.3.3.3 GAME_CHANNEL

```
const std::string GAME_CHANNEL = "GAME CHANNEL"
```

A message channel for game messages.

4.3.3.4 GAME_HEIGHT

```
const int GAME_HEIGHT = 800
```

Height of the game space.

4.3.3.5 GAME_WIDTH

```
const int GAME_WIDTH = 1200
```

Width of the game space.

4.3.3.6 NO_TILE_SELECTED_CHANNEL

```
const std::string NO_TILE_SELECTED_CHANNEL = "NO TILE SELECTED CHANNEL"
```

A message channel for no tile selected messages.

4.3.3.7 SECONDS_PER_FRAME

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

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

4.3.3.8 SECONDS_PER_MONTH

```
const unsigned long long int SECONDS_PER_MONTH = 2628164
```

4.3.3.9 SECONDS_PER_YEAR

```
const unsigned long long int SECONDS_PER_YEAR = 31537970
```

4.3.3.10 TILE_RESOURCE_CUMULATIVE_PROBABILITIES

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

Initial value:

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

Cumulative probabilities for each tile resource (to support procedural generation).

4.3.3.11 TILE_SELECTED_CHANNEL

```
const std::string TILE_SELECTED_CHANNEL = "TILE SELECTED CHANNEL"
```

A message channel for tile selection messages.

4.3.3.12 TILE_STATE_CHANNEL

```
const std::string TILE_STATE_CHANNEL = "TILE STATE CHANNEL"
```

A message channel for tile state messages.

4.3.3.13 TILE_TYPE_CUMULATIVE_PROBABILITIES

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

Initial value:

```
= {  
    0.25,  
    0.50,  
    0.75,  
    1.00  
}
```

Cumulative probabilities for each tile type (to support procedural generation).

4.4 header/ESC_core/doxygen_cite.h File Reference

Header file which simply cites the doxygen tool.

4.4.1 Detailed Description

Header file which simply cites the doxygen tool.

Ref: [van Heesch. \[2023\]](#)

4.5 header/ESC_core/includes.h File Reference

Header file for various includes.

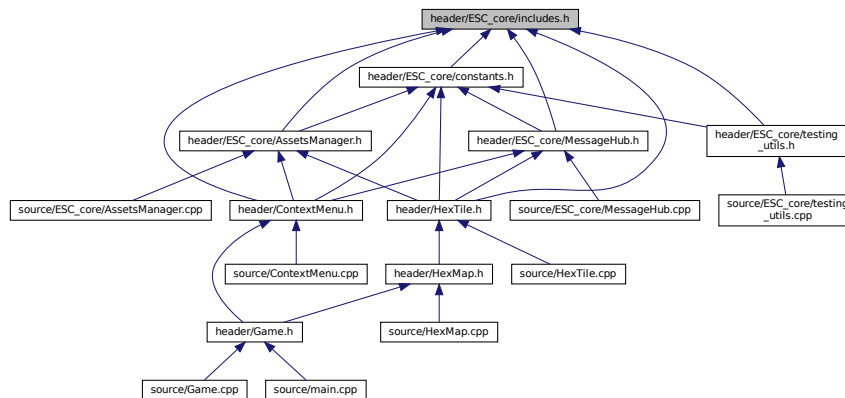
```
#include <chrono>  
#include <cmath>  
#include <cstdlib>  
#include <filesystem>  
#include <fstream>  
#include <iomanip>  
#include <iostream>  
#include <limits>  
#include <list>  
#include <map>  
#include <stdexcept>  
#include <sstream>  
#include <string>  
#include <vector>  
#include <SFML/Audio.hpp>  
#include <SFML/Config.hpp>  
#include <SFML/GpuPreference.hpp>  
#include <SFML/Graphics.hpp>  
#include <SFML/Main.hpp>  
#include <SFML/Network.hpp>  
#include <SFML/OpenGL.hpp>  
#include <SFML/System.hpp>
```

```
#include <SFML/Window.hpp>
```

Include dependency graph for includes.h:



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



4.5.1 Detailed Description

Header file for various includes.

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

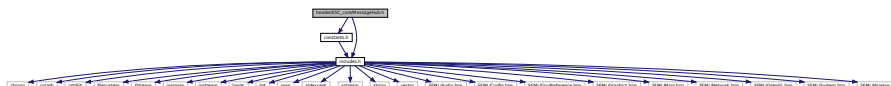
4.6 header/ESC_core/MessageHub.h File Reference

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

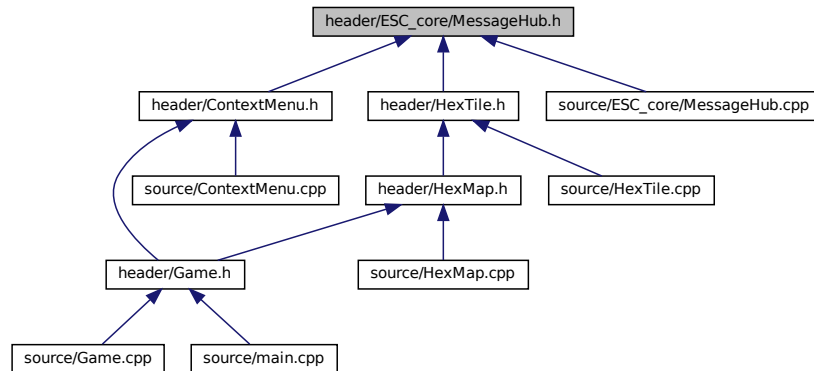
```
#include "constants.h"
```

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

Include dependency graph for MessageHub.h:



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



Classes

- struct [Message](#)
A structure which defines a standard message format.
- class [MessageHub](#)
A class which acts as a central hub for inter-object message traffic.

4.6.1 Detailed Description

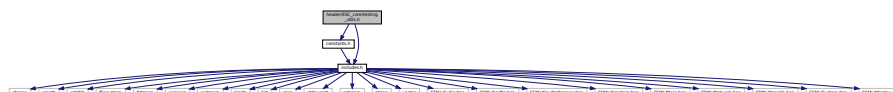
Header file for the [MessageHub](#) class.

4.7 header/ESC_core/testing_utils.h File Reference

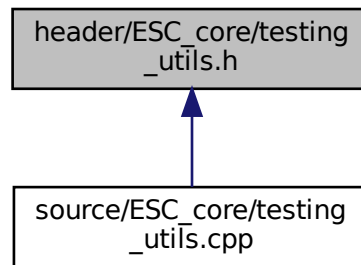
Header file for various testing utilities.

```
#include "constants.h"
#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.

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

```

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

```

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

Parameters

<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() */

```

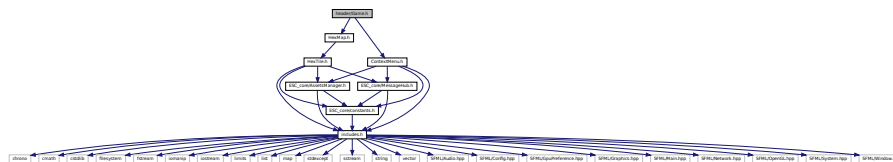
4.8 header/Game.h File Reference

```

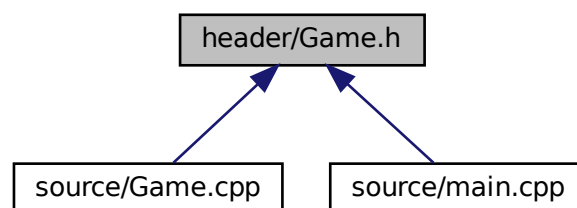
#include "HexMap.h"
#include "ContextMenu.h"

```

Include dependency graph for Game.h:



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



Classes

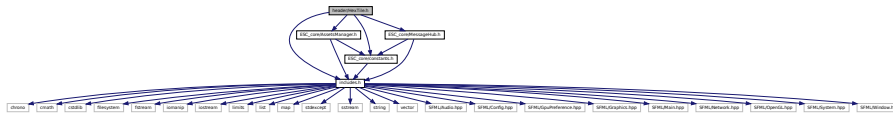
- class [Game](#)

A class which acts as the central class for the game, by containing all other classes and implementing the game loop.

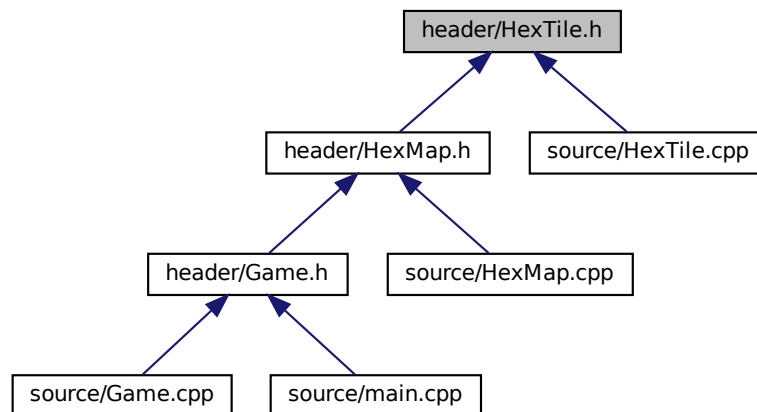
4.10 header/HexTile.h File Reference

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

```
#include "ESC_core/constants.h"
#include "ESC_core/includes.h"
#include "ESC_core/AssetsManager.h"
#include "ESC_core/MessageHub.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.

4.10.1 Detailed Description

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

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

4.10.2 Enumeration Type Documentation

4.10.2.1 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.

```

50         {
51     POOR,
52     BELOW_AVERAGE,
53     AVERAGE,
54     ABOVE_AVERAGE,
55     GOOD,
56     N_TILE_RESOURCES
57 }; /* TileResource */

```

4.10.2.2 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 TileType.

```

34         {

```

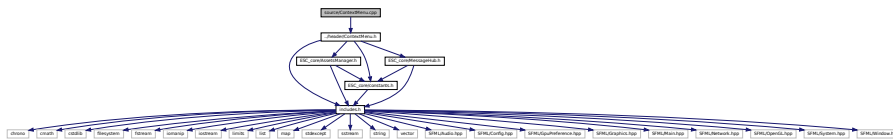
```
35     FOREST,
36     LAKE,
37     MOUNTAINS,
38     OCEAN,
39     PLAINS,
40     N_TILE_TYPES
41 }; /* TileType */
```

4.11 source/ContextMenu.cpp File Reference

Implementation file for the `ContextMenu` class.

```
#include "../header/ContextMenu.h"
```

Include dependency graph for ContextMenu.cpp:



4.11.1 Detailed Description

Implementation file for the `ContextMenu` class.

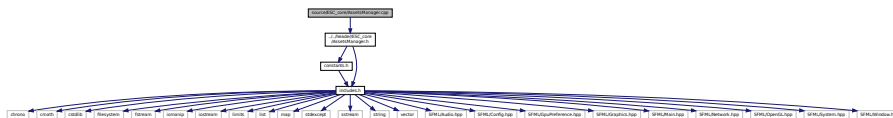
A class which defines a context menu for the game.

4.12 source/ESC_core/AssetsManager.cpp File Reference

Implementation file for the `AssetsManager` class.

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

Include dependency graph for AssetsManager.cpp:



4.12.1 Detailed Description

Implementation file for the `AssetsManager` class.

A class which manages visual and sound assets.

4.14.1 Detailed Description

Implementation file for various testing utilities.

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

4.14.2 Function Documentation

4.14.2.1 `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() */
```

4.14.2.2 `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() */
```

4.14.2.3 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() */
```

4.14.2.4 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() */
```

4.14.2.5 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() */

```

4.14.2.6 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() */

```

4.14.2.7 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() */
```

4.14.2.8 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() */

```

4.14.2.9 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() */

```

4.14.2.10 testTruth()

```

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

```

Tests if the given statement is true.

4.18.1 Detailed Description

Implementation file for `main()` for Road To Zero.

4.18.2 Function Documentation

4.18.2.1 `constructRenderWindow()`

```
sf::RenderWindow * constructRenderWindow (
    void )
```

Helper function to construct render window.

Returns

Pointer to the render window.

```
54 {
55     sf::RenderWindow* render_window_ptr = new sf::RenderWindow(
56         sf::VideoMode(GAME_WIDTH, GAME_HEIGHT),
57         "Road To Zero"
58     );
59
60     return render_window_ptr;
61 } /* constructRenderWindow() */
```

4.18.2.2 `loadAssets()`

```
void loadAssets (
    AssetsManager * assets_manager_ptr )
```

Helper function to load game assets.

Parameters

<code>assets_manager_ptr</code>	Pointer to the assets manager.
---------------------------------	--------------------------------

```
32 {
33     // 1. load font assets
34     assets_manager_ptr->loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
35     assets_manager_ptr->loadFont("assets/fonts/Glass_TTY_VT220.ttf", "Glass_TTY_VT220");
36
37     return;
38 } /* loadAssets() */
```

4.18.2.3 `main()`

```
int main (
    int argc,
    char ** argv )
```

```
70 {
71     // 1. load assets
72     AssetsManager assets_manager;
73     loadAssets(&assets_manager);
74
75     // 2. construct render window
76     sf::RenderWindow* render_window_ptr = constructRenderWindow();
77
78     // 3. start game loop
79     bool quit_game = false;
80
81     while (not quit_game) {
82         Game game(render_window_ptr, &assets_manager);
83         quit_game = game.run();
84     }
85
86     // 4. clean up
87     render_window_ptr->close();
88     delete render_window_ptr;
89
90     return 0;
91 } /* main() */
```


Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL <https://www.sfml-dev.org/>. 101

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL <https://www.doxygen.nl>. 100

Wikipedia. Hexagon, 2023. URL <https://en.wikipedia.org/wiki/Hexagon>. 67

Index

- __assembleHexMap
 - HexMap, [47](#)
- __draw
 - Game, [37](#)
- __drawConsoleScreenFrame
 - ContextMenu, [20](#)
- __drawConsoleText
 - ContextMenu, [21](#)
- __drawFrameClockOverlay
 - Game, [37](#)
- __drawHUD
 - Game, [37](#)
- __drawVisualScreenFrame
 - ContextMenu, [21](#)
- __enforceOceanContinuity
 - HexMap, [48](#)
- __getMajorityTileType
 - HexMap, [48](#)
- __getNeighboursVector
 - HexMap, [49](#)
- __getNoise
 - HexMap, [50](#)
- __getSelectedTile
 - HexMap, [51](#)
- __getTileCoordsSubstring
 - HexTile, [68](#)
- __getTileImprovementSubstring
 - HexTile, [68](#)
- __getTileResourceSubstring
 - HexTile, [68](#)
- __getTileTypeSubstring
 - HexTile, [69](#)
- __getValidMapIndexPositions
 - HexMap, [52](#)
- __handleKeyPressEvents
 - ContextMenu, [22](#)
 - Game, [38](#)
 - HexMap, [53](#)
 - HexTile, [70](#)
- __handleMouseButtonEvents
 - ContextMenu, [22](#)
 - Game, [39](#)
 - HexMap, [53](#)
 - HexTile, [70](#)
- __isClicked
 - HexTile, [71](#)
- __isLakeTouchingOcean
 - HexMap, [54](#)
- __layTiles
 - HexMap, [54](#)
- __loadSoundBuffer
 - AssetsManager, [7](#)
- __procedurallyGenerateTileResources
 - HexMap, [56](#)
- __procedurallyGenerateTileTypes
 - HexMap, [57](#)
- __processEvent
 - Game, [39](#)
- __processMessage
 - Game, [39](#)
- __sendNoTileSelectedMessage
 - HexMap, [57](#)
- __sendQuitGameMessage
 - ContextMenu, [23](#)
- __sendRestartGameMessage
 - ContextMenu, [23](#)
- __sendTileSelectedMessage
 - HexTile, [71](#)
- __sendTileStateMessage
 - HexTile, [71](#)
- __setConsoleState
 - ContextMenu, [23](#)
- __setConsoleString
 - ContextMenu, [24](#)
- __setResourceText
 - HexTile, [72](#)
- __setUpConsoleScreen
 - ContextMenu, [24](#)
- __setUpConsoleScreenFrame
 - ContextMenu, [25](#)
- __setUpGlassScreen
 - HexMap, [58](#)
- __setUpMenuFrame
 - ContextMenu, [27](#)
- __setUpNodeSprite
 - HexTile, [73](#)
- __setUpResourceChipSprite
 - HexTile, [73](#)
- __setUpSelectOutlineSprite
 - HexTile, [74](#)
- __setUpTileSprite
 - HexTile, [74](#)
- __setUpVisualScreen
 - ContextMenu, [27](#)
- __setUpVisualScreenFrame
 - ContextMenu, [27](#)
- __smoothTileTypes
 - HexMap, [58](#)

- __toggleFrameClockOverlay
 - Game, 40
- ~AssetsManager
 - AssetsManager, 6
- ~ContextMenu
 - ContextMenu, 20
- ~Game
 - Game, 36
- ~HexMap
 - HexMap, 47
- ~HexTile
 - HexTile, 68
- ~MessageHub
 - MessageHub, 85
- ABOVE_AVERAGE
 - HexTile.h, 112
- addChannel
 - MessageHub, 85
- assess
 - HexMap, 58
 - HexTile, 74
- assets_manager_ptr
 - ContextMenu, 30
 - Game, 41
 - HexMap, 61
 - HexTile, 78
- 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, 112
- BELOW_AVERAGE
 - HexTile.h, 112
- bool_payload_vec
 - Message, 83
- border_tiles_vec
 - HexMap, 62
- channel
 - Message, 83
- clear
 - AssetsManager, 8
 - HexMap, 59
 - MessageHub, 86
- clearMessages
 - MessageHub, 86
- clock
 - Game, 41
- console_screen
 - ContextMenu, 31
- console_screen_frame_bottom
 - ContextMenu, 31
- console_screen_frame_left
 - ContextMenu, 31
- console_screen_frame_right
 - ContextMenu, 31
- console_screen_frame_top
 - ContextMenu, 31
- console_state
 - ContextMenu, 31
- console_string
 - ContextMenu, 32
- ConsoleState
 - ContextMenu.h, 92
- constants.h
 - FLOAT_TOLERANCE, 97
 - FOREST_GREEN, 95
 - FRAMES_PER_SECOND, 98
 - GAME_CHANNEL, 98
 - GAME_HEIGHT, 98
 - GAME_WIDTH, 98
 - LAKE_BLUE, 95
 - MENU_FRAME_GREY, 95
 - MONOCHROME_SCREEN_BACKGROUND, 96
 - MONOCHROME_TEXT_AMBER, 96
 - MONOCHROME_TEXT_GREEN, 96
 - MONOCHROME_TEXT_RED, 96
 - MOUNTAINS_GREY, 96
 - NO_TILE_SELECTED_CHANNEL, 98
 - OCEAN_BLUE, 97
 - PLAINS_YELLOW, 97
 - SECONDS_PER_FRAME, 98
 - SECONDS_PER_MONTH, 99
 - SECONDS_PER_YEAR, 99
 - TILE_RESOURCE_CUMULATIVE_PROBABILITIES, 99
 - TILE_SELECTED_CHANNEL, 99
 - TILE_STATE_CHANNEL, 99
 - TILE_TYPE_CUMULATIVE_PROBABILITIES, 99
 - VISUAL_SCREEN_FRAME_GREY, 97
- constructRenderWindow
 - main.cpp, 122
- context_menu_ptr

- Game, 41
- ContextMenu, 17
 - __drawConsoleScreenFrame, 20
 - __drawConsoleText, 21
 - __drawVisualScreenFrame, 21
 - __handleKeyPressEvents, 22
 - __handleMouseButtonEvents, 22
 - __sendQuitGameMessage, 23
 - __sendRestartGameMessage, 23
 - __setConsoleState, 23
 - __setConsoleString, 24
 - __setUpConsoleScreen, 24
 - __setUpConsoleScreenFrame, 25
 - __setUpMenuFrame, 27
 - __setUpVisualScreen, 27
 - __setUpVisualScreenFrame, 27
- ~ContextMenu, 20
- assets_manager_ptr, 30
- console_screen, 31
- console_screen_frame_bottom, 31
- console_screen_frame_left, 31
- console_screen_frame_right, 31
- console_screen_frame_top, 31
- console_state, 31
- console_string, 32
- ContextMenu, 19
- draw, 29
- event_ptr, 32
- frame, 32
- game_menu_up, 32
- menu_frame, 32
- message_hub_ptr, 32
- position_x, 33
- position_y, 33
- processEvent, 29
- processMessage, 30
- render_window_ptr, 33
- visual_screen, 33
- visual_screen_frame_bottom, 33
- visual_screen_frame_left, 33
- visual_screen_frame_right, 34
- visual_screen_frame_top, 34
- ContextMenu.h
 - ConsoleState, 92
 - MENU, 92
 - N_CONSOLE_STATES, 92
 - NONE, 92
 - READY, 92
 - TILE, 92
- current_track
 - AssetsManager, 16
- double_payload_vec
 - Message, 83
- draw
 - ContextMenu, 29
 - HexMap, 59
 - HexTile, 75
- event
 - Game, 42
- event_ptr
 - ContextMenu, 32
 - HexMap, 62
 - HexTile, 79
- expectedErrorNotDetected
 - testing_utils.cpp, 115
 - testing_utils.h, 103
- FLOAT_TOLERANCE
 - constants.h, 97
- font_map
 - AssetsManager, 16
- FOREST
 - HexTile.h, 112
- FOREST_GREEN
 - constants.h, 95
- frame
 - ContextMenu, 32
 - Game, 42
 - HexMap, 62
 - HexTile, 79
- FRAMES_PER_SECOND
 - constants.h, 98
- Game, 34
 - __draw, 37
 - __drawFrameClockOverlay, 37
 - __drawHUD, 37
 - __handleKeyPressEvents, 38
 - __handleMouseButtonEvents, 39
 - __processEvent, 39
 - __processMessage, 39
 - __toggleFrameClockOverlay, 40
 - ~Game, 36
 - assets_manager_ptr, 41
 - clock, 41
 - context_menu_ptr, 41
 - event, 42
 - frame, 42
 - Game, 36
 - game_loop_broken, 42
 - hex_map_ptr, 42
 - message_hub, 42
 - month, 42
 - quit_game, 43
 - render_window_ptr, 43
 - run, 40
 - show_frame_clock_overlay, 43
 - time_since_start_s, 43
 - year, 43
- GAME_CHANNEL
 - constants.h, 98
- GAME_HEIGHT
 - constants.h, 98
- game_loop_broken
 - Game, 42
- game_menu_up

- ContextMenu, 32
- GAME_WIDTH
 - constants.h, 98
- getCurrentTrackKey
 - AssetsManager, 9
- getFont
 - AssetsManager, 9
- getSound
 - AssetsManager, 10
- getSoundBuffer
 - AssetsManager, 10
- getTexture
 - AssetsManager, 11
- getTrackStatus
 - AssetsManager, 11
- glass_screen
 - HexMap, 62
- GOOD
 - HexTile.h, 112
- has_improvement
 - HexTile, 79
- hasTraffic
 - MessageHub, 86
- header/ContextMenu.h, 91
- header/ESC_core/AssetsManager.h, 92
- header/ESC_core/constants.h, 93
- header/ESC_core/doxygen_cite.h, 100
- header/ESC_core/includes.h, 100
- header/ESC_core/MessageHub.h, 101
- header/ESC_core/testing_utils.h, 102
- header/Game.h, 109
- header/HexMap.h, 110
- header/HexTile.h, 111
- hex_map
 - HexMap, 62
- hex_map_ptr
 - Game, 42
- HexMap, 44
 - __assembleHexMap, 47
 - __enforceOceanContinuity, 48
 - __getMajorityTileType, 48
 - __getNeighboursVector, 49
 - __getNoise, 50
 - __getSelectedTile, 51
 - __getValidMapIndexPositions, 52
 - __handleKeyPressEvents, 53
 - __handleMouseButtonEvents, 53
 - __isLakeTouchingOcean, 54
 - __layTiles, 54
 - __procedurallyGenerateTileResources, 56
 - __procedurallyGenerateTileTypes, 57
 - __sendNoTileSelectedMessage, 57
 - __setUpGlassScreen, 58
 - __smoothTileTypes, 58
 - ~HexMap, 47
 - assess, 58
 - assets_manager_ptr, 61
 - border_tiles_vec, 62
 - clear, 59
 - draw, 59
 - event_ptr, 62
 - frame, 62
 - glass_screen, 62
 - hex_map, 62
 - HexMap, 46
 - message_hub_ptr, 62
 - n_layers, 63
 - n_tiles, 63
 - position_x, 63
 - position_y, 63
 - processEvent, 60
 - processMessage, 60
 - render_window_ptr, 63
 - reroll, 61
 - tile_position_x_vec, 63
 - tile_position_y_vec, 64
 - tile_selected, 64
 - toggleResourceOverlay, 61
- HexTile, 64
 - __getTileCoordsSubstring, 68
 - __getTileImprovementSubstring, 68
 - __getTileResourceSubstring, 68
 - __getTileTypeSubstring, 69
 - __handleKeyPressEvents, 70
 - __handleMouseButtonEvents, 70
 - __isClicked, 71
 - __sendTileSelectedMessage, 71
 - __sendTileStateMessage, 71
 - __setResourceText, 72
 - __setUpNodeSprite, 73
 - __setUpResourceChipSprite, 73
 - __setUpSelectOutlineSprite, 74
 - __setUpTileSprite, 74
 - ~HexTile, 68
 - assess, 74
 - assets_manager_ptr, 78
 - draw, 75
 - event_ptr, 79
 - frame, 79
 - has_improvement, 79
 - HexTile, 67
 - is_selected, 79
 - major_radius, 79
 - message_hub_ptr, 79
 - minor_radius, 80
 - node_sprite, 80
 - position_x, 80
 - position_y, 80
 - processEvent, 75
 - processMessage, 75
 - render_window_ptr, 80
 - resource_assessed, 80
 - resource_chip_sprite, 81
 - resource_text, 81
 - select_outline_sprite, 81
 - setTileResource, 76

- setTileType, 77
- show_node, 81
- show_resource, 81
- tile_resource, 81
- tile_sprite, 82
- tile_type, 82
- toggleResourceOverlay, 78
- HexTile.h
 - ABOVE_AVERAGE, 112
 - AVERAGE, 112
 - BELOW_AVERAGE, 112
 - FOREST, 112
 - GOOD, 112
 - LAKE, 112
 - MOUNTAINS, 112
 - N_TILE_RESOURCES, 112
 - N_TILE_TYPES, 112
 - OCEAN, 112
 - PLAINS, 112
 - POOR, 112
 - TileResource, 112
 - TileType, 112
- int_payload_vec
 - Message, 83
- is_selected
 - HexTile, 79
- isEmpty
 - MessageHub, 87
- LAKE
 - HexTile.h, 112
- LAKE_BLUE
 - constants.h, 95
- loadAssets
 - main.cpp, 122
- loadFont
 - AssetsManager, 12
- loadSound
 - AssetsManager, 12
- loadTexture
 - AssetsManager, 13
- loadTrack
 - AssetsManager, 14
- main
 - main.cpp, 122
- main.cpp
 - constructRenderWindow, 122
 - loadAssets, 122
 - main, 122
- major_radius
 - HexTile, 79
- MENU
 - ContextMenu.h, 92
- menu_frame
 - ContextMenu, 32
- MENU_FRAME_GREY
 - constants.h, 95
- Message, 82
 - bool_payload_vec, 83
 - channel, 83
 - double_payload_vec, 83
 - int_payload_vec, 83
 - string_payload, 83
 - subject, 83
- message_hub
 - Game, 42
- message_hub_ptr
 - ContextMenu, 32
 - HexMap, 62
 - HexTile, 79
- message_map
 - MessageHub, 90
- MessageHub, 84
 - ~MessageHub, 85
 - addChannel, 85
 - clear, 86
 - clearMessages, 86
 - hasTraffic, 86
 - isEmpty, 87
 - message_map, 90
 - MessageHub, 85
 - popMessage, 87
 - receiveMessage, 88
 - removeChannel, 89
 - sendMessage, 89
- minor_radius
 - HexTile, 80
- MONOCHROME_SCREEN_BACKGROUND
 - constants.h, 96
- MONOCHROME_TEXT_AMBER
 - constants.h, 96
- MONOCHROME_TEXT_GREEN
 - constants.h, 96
- MONOCHROME_TEXT_RED
 - constants.h, 96
- month
 - Game, 42
- MOUNTAINS
 - HexTile.h, 112
- MOUNTAINS_GREY
 - constants.h, 96
- N_CONSOLE_STATES
 - ContextMenu.h, 92
- n_layers
 - HexMap, 63
- N_TILE_RESOURCES
 - HexTile.h, 112
- N_TILE_TYPES
 - HexTile.h, 112
- n_tiles
 - HexMap, 63
- nextTrack
 - AssetsManager, 14
- NO_TILE_SELECTED_CHANNEL
 - constants.h, 98

- node_sprite
 - HexTile, [80](#)
- NONE
 - ContextMenu.h, [92](#)
- OCEAN
 - HexTile.h, [112](#)
- OCEAN_BLUE
 - constants.h, [97](#)
- pauseTrack
 - AssetsManager, [15](#)
- PLAINS
 - HexTile.h, [112](#)
- PLAINS_YELLOW
 - constants.h, [97](#)
- playTrack
 - AssetsManager, [15](#)
- POOR
 - HexTile.h, [112](#)
- popMessage
 - MessageHub, [87](#)
- position_x
 - ContextMenu, [33](#)
 - HexMap, [63](#)
 - HexTile, [80](#)
- position_y
 - ContextMenu, [33](#)
 - HexMap, [63](#)
 - HexTile, [80](#)
- previousTrack
 - AssetsManager, [15](#)
- printGold
 - testing_utils.cpp, [115](#)
 - testing_utils.h, [104](#)
- printGreen
 - testing_utils.cpp, [115](#)
 - testing_utils.h, [104](#)
- printRed
 - testing_utils.cpp, [116](#)
 - testing_utils.h, [105](#)
- processEvent
 - ContextMenu, [29](#)
 - HexMap, [60](#)
 - HexTile, [75](#)
- processMessage
 - ContextMenu, [30](#)
 - HexMap, [60](#)
 - HexTile, [75](#)
- quit_game
 - Game, [43](#)
- READY
 - ContextMenu.h, [92](#)
- receiveMessage
 - MessageHub, [88](#)
- removeChannel
 - MessageHub, [89](#)
- render_window_ptr
 - ContextMenu, [33](#)
 - Game, [43](#)
 - HexMap, [63](#)
 - HexTile, [80](#)
- reroll
 - HexMap, [61](#)
- resource_assessed
 - HexTile, [80](#)
- resource_chip_sprite
 - HexTile, [81](#)
- resource_text
 - HexTile, [81](#)
- run
 - Game, [40](#)
- SECONDS_PER_FRAME
 - constants.h, [98](#)
- SECONDS_PER_MONTH
 - constants.h, [99](#)
- SECONDS_PER_YEAR
 - constants.h, [99](#)
- select_outline_sprite
 - HexTile, [81](#)
- sendMessage
 - MessageHub, [89](#)
- setTileResource
 - HexTile, [76](#)
- setTileType
 - HexTile, [77](#)
- show_frame_clock_overlay
 - Game, [43](#)
- show_node
 - HexTile, [81](#)
- show_resource
 - HexTile, [81](#)
- sound_map
 - AssetsManager, [16](#)
- soundbuffer_map
 - AssetsManager, [16](#)
- source/ContextMenu.cpp, [113](#)
- source/ESC_core/AssetsManager.cpp, [113](#)
- source/ESC_core/MessageHub.cpp, [114](#)
- source/ESC_core/testing_utils.cpp, [114](#)
- source/Game.cpp, [120](#)
- source/HexMap.cpp, [120](#)
- source/HexTile.cpp, [121](#)
- source/main.cpp, [121](#)
- stopTrack
 - AssetsManager, [15](#)
- string_payload
 - Message, [83](#)
- subject
 - Message, [83](#)
- testFloatEquals
 - testing_utils.cpp, [116](#)
 - testing_utils.h, [105](#)
- testGreaterThan

- testing_utils.cpp, 117
- testing_utils.h, 106
- testGreaterThanOrEqualTo
 - testing_utils.cpp, 117
 - testing_utils.h, 106
- testing_utils.cpp
 - expectedErrorNotDetected, 115
 - printGold, 115
 - printGreen, 115
 - printRed, 116
 - testFloatEquals, 116
 - testGreaterThan, 117
 - testGreaterThanOrEqualTo, 117
 - testLessThan, 118
 - testLessThanOrEqualTo, 119
 - testTruth, 119
- testing_utils.h
 - expectedErrorNotDetected, 103
 - printGold, 104
 - printGreen, 104
 - printRed, 105
 - testFloatEquals, 105
 - testGreaterThan, 106
 - testGreaterThanOrEqualTo, 106
 - testLessThan, 107
 - testLessThanOrEqualTo, 108
 - testTruth, 108
- testLessThan
 - testing_utils.cpp, 118
 - testing_utils.h, 107
- testLessThanOrEqualTo
 - testing_utils.cpp, 119
 - testing_utils.h, 108
- testTruth
 - testing_utils.cpp, 119
 - testing_utils.h, 108
- texture_map
 - AssetsManager, 16
- TILE
 - ContextMenu.h, 92
- tile_position_x_vec
 - HexMap, 63
- tile_position_y_vec
 - HexMap, 64
- tile_resource
 - HexTile, 81
- TILE_RESOURCE_CUMULATIVE_PROBABILITIES
 - constants.h, 99
- tile_selected
 - HexMap, 64
- TILE_SELECTED_CHANNEL
 - constants.h, 99
- tile_sprite
 - HexTile, 82
- TILE_STATE_CHANNEL
 - constants.h, 99
- tile_type
 - HexTile, 82
- TILE_TYPE_CUMULATIVE_PROBABILITIES
 - constants.h, 99
- TileResource
 - HexTile.h, 112
- TileType
 - HexTile.h, 112
- time_since_start_s
 - Game, 43
- toggleResourceOverlay
 - HexMap, 61
 - HexTile, 78
- track_map
 - AssetsManager, 17
- visual_screen
 - ContextMenu, 33
- visual_screen_frame_bottom
 - ContextMenu, 33
- VISUAL_SCREEN_FRAME_GREY
 - constants.h, 97
- visual_screen_frame_left
 - ContextMenu, 33
- visual_screen_frame_right
 - ContextMenu, 34
- visual_screen_frame_top
 - ContextMenu, 34
- year
 - Game, 43