HelloWorld

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

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

path_2_sound	A path (either relative or absolute) to the sound file.
sound_key	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
662     //  2. clear textures
663     std::map<std::string, sf::Texture*>::iterator texture_iter;
664     for (
665         texture_iter = this->texture_map.begin();
666         texture_iter != this->texture_map.end();
667         texture_iter++
668     ) {
669         delete texture_iter->second;
670
671         std::cout « "Texture " « texture_iter->first « " deleted from texture map" «
672             std::endl;
673     }
674     this->texture_map.clear();
675
676
677     //  3. clear sound buffers
678     std::map<std::string, sf::SoundBuffer*>::iterator soundbuffer_iter;
679     for (
680         soundbuffer_iter = this->soundbuffer_map.begin();
681         soundbuffer_iter != this->soundbuffer_map.end();
682         soundbuffer_iter++
683     ) {
684         delete soundbuffer_iter->second;
685
686         std::cout « "SoundBuffer " « soundbuffer_iter->first «
687             " deleted from soundbuffer map" « std::endl;
688     }
689     this->soundbuffer_map.clear();
690
691
692     //  4. clear sounds
693     std::map<std::string, sf::Sound*>::iterator sound_iter;
694     for (
695         sound_iter = this->sound_map.begin();
696         sound_iter != this->sound_map.end();
697         sound_iter++
698     ) {
699         sound_iter->second->stop();
700         delete sound_iter->second;
701
702         std::cout « "Sound " « sound_iter->first « " deleted from sound map" «
703             std::endl;
704     }
705     this->sound_map.clear();
706
```

```
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

font_key	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

sound_key	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

sound_key	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

texture_key	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

path_2_font	A path (either relative or absolute) to the font file.
font_key	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 
152     //  2. load from file, throw error on fail
153     sf::Font* font_ptr = new sf::Font();
154 
155     if (not font_ptr->loadFromFile(path_2_font)) {
156         std::string error_str = "ERROR  AssetsManager::loadFont()  could not load ";
157         error_str += "font at ";
158         error_str += path_2_font;
159 
160         this->clear();
161 
162         #ifdef _WIN32
163             std::cout « error_str « std::endl;
164         #endif  /* _WIN32 */
165 
166         throw std::runtime_error(error_str);
167     }
168 
169 
170     //  3. insert into font map
171     this->font_map.insert(std::pair<std::string, sf::Font*>(font_key, font_ptr));
172 
173     std::cout « "Font " « font_key « " inserted into font map" « std::endl;
174 
175     return;
176 }   /* 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

path_2_sound	A path (either relative or absolute) to the sound file.
sound_key	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

path_2_texture	A path (either relative or absolute) to the texture file.
texture_key	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 
213     //  2. load from file, throw error on fail
214     sf::Texture* texture_ptr = new sf::Texture();
215 
216     if (not texture_ptr->loadFromFile(path_2_texture)) {
217         std::string error_str = "ERROR  AssetsManager::loadTexture()  could not load ";
218         error_str += "texture at ";
219         error_str += path_2_texture;
220 
221         this->clear();
222 
223         #ifdef _WIN32
224             std::cout « error_str « std::endl;
```

```
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

path_2_track	A path (either relative or absolute) to the track file.
track_key	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:

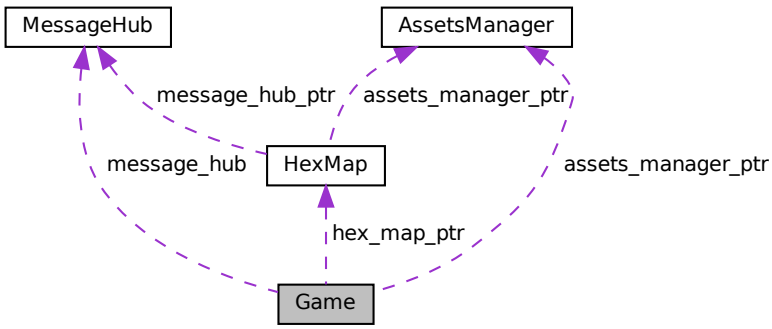
- header/ESC_core/AssetsManager.h
- source/ESC_core/AssetsManager.cpp

3.2 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 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.
- 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).

Private Member Functions

- void __toggleFrameClockOverlay (void)

  Helper method to toggle frame clock overlay.
- void __drawFrameClockOverlay (void)

  Helper method to draw frame clock overlay.
- void __processEvent (void)

  Helper method to process Game. To be called once per event.
- void __processFrame (void)

  Helper method to process Game. To be called once per frame.
- 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.

3.2.1 Detailed Description

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

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Game()

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

Constructor for the Game class.

```
186 {
187     //  1. set attributes
188 
189     //  1.1. private
190     this->render_window_ptr = render_window_ptr;
191 
192     this->assets_manager_ptr = assets_manager_ptr;
193 
194     //  1.2. public
195     this->quit_game = false;
196     this->show_frame_clock_overlay = false;
197 
198     this->frame = 0;
199     this->time_since_start_s = 0;
200 
201     this->hex_map_ptr = new HexMap(
202         6,
203         &(this->event),
204         this->render_window_ptr,
205         this->assets_manager_ptr,
206         &(this->message_hub)
207     );
208 
209     std::cout << "Game constructed at " << this << std::endl;
210 
211     return;
212 }   /* Game() */
```

3.2.2.2 ~Game()

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

Destructor for the Game class.

```
287 {
288     delete this->hex_map_ptr;
289 
290     std::cout << "Game at " << this << " destroyed" << std::endl;
291 
292     return;
293 }   /* ~Game() */
```

3.2.3 Member Function Documentation

3.2.3.1 __draw()

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

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

```
155 {
156     if (this->show_frame_clock_overlay) {
157         this->__drawFrameClockOverlay();
158     }
159 
160     return;
161 }   /* draw() */
```

3.2.3.2 __drawFrameClockOverlay()

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

Helper method to draw frame clock overlay.

```
59 {
60     std::string frame_clock_string = "FRAME: ";
61     frame_clock_string += std::to_string(this->frame);
62     frame_clock_string += "\nTIME SINCE START [s]: ";
63     frame_clock_string += std::to_string(this->time_since_start_s);
64 
65     sf::Text frame_clock_text(
66         frame_clock_string,
67         *(this->assets_manager_ptr->getFont("DroidSansMono")),
68         16
69     );
70 
71     sf::RectangleShape frame_clock_backing(
72         sf::Vector2f(
73             1.02 * frame_clock_text.getLocalBounds().width,
74             1.02 * frame_clock_text.getLocalBounds().height
75         )
76     );
77     frame_clock_backing.setFillColor(sf::Color(0, 0, 0, 255));
78 
79     this->render_window_ptr->draw(frame_clock_backing);
80     this->render_window_ptr->draw(frame_clock_text);
81 
82     return;
83 }   /* __drawFrameClockOverlay() */
```

3.2.3.3 __processEvent()

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

Helper method to process Game. To be called once per event.

```
98 {
99     if (this->event.type == sf::Event::KeyPressed) {
100         switch (this->event.key.code) {
101             case (sf::Keyboard::Tilde): {
102                 this->__toggleFrameClockOverlay();
103 
104                 break;
105             }
106 
107             default: {
108                 // do nothing!
109 
110                 break;
111             }
112         }
113     }
114 
115     if (this->event.type == sf::Event::Closed) {
116         this->render_window_ptr->close();
117         this->quit_game = true;
118     }
119 
120     return;
121 }   /* __processEvent() */
```

3.2.3.4 __processFrame()

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

Helper method to process Game. To be called once per frame.

```
136 {
137     //...
138 
139     return;
140 }   /* __processFrame() */
```

3.2.3.5 __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() */
```

3.2.3.6 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).

```
230 {
231     //  1. play brand animation
232     //...
233 
234     //  2. show splash screen
235     //...
236 
237     //  3. start game loop
238     while (this->render_window_ptr->isOpen()) {
239         this->time_since_start_s = this->clock.getElapsedTime().asSeconds();
240 
241         if (this->time_since_start_s >= (this->frame + 1) * SECONDS_PER_FRAME) {
242             //  6.1. process events
243             while (this->render_window_ptr->pollEvent(this->event)) {
244                 this->hex_map_ptr->processEvent();
245 
246                 this->__processEvent();
247             }
248 
249 
250             //  6.2. process frame
251             this->hex_map_ptr->processFrame();
252 
```

```
253             this->__processFrame();
254
255
256             //  6.3. draw frame
257             this->render_window_ptr->clear();
258
259             this->hex_map_ptr->draw();
260
261             this->__draw();
262
263             this->render_window_ptr->display();
264
265
266             //  6.4. increment frame
267             this->frame++;
268         }
269     }
270
271     return this->quit_game;
272 }   /*  */
```

3.2.4 Member Data Documentation

3.2.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.2.4.2 clock

```
sf::Clock Game::clock
```

The game clock.

3.2.4.3 event

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

The game events class.

3.2.4.4 frame

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

The current frame of the game.

3.2.4.5 hex_map_ptr

```
HexMap* Game::hex_map_ptr
```

Pointer to the hex map (defines game world).

3.2.4.6 message_hub

```
MessageHub Game::message_hub
```

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

3.2.4.7 quit_game

```
bool Game::quit_game
```

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

3.2.4.8 render_window_ptr

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

A pointer to the render window.

3.2.4.9 show_frame_clock_overlay

```
bool Game::show_frame_clock_overlay
```

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

3.2.4.10 time_since_start_s

```
double Game::time_since_start_s
```

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

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

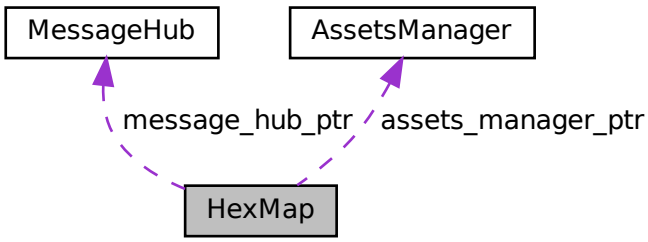
- header/Game.h
- source/Game.cpp

3.3 HexMap Class Reference

A class which defines a hex map of hex tiles.

`#include <HexMap.h>`

Collaboration diagram for HexMap:



Public Member Functions

- HexMap (void)

  Constructor (dummy) for the HexMap class.
- 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 processFrame (void)

  Method to process HexMap. To be called once per frame.
- 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

- 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.

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.

3.3.1 Detailed Description

A class which defines a hex map of hex tiles.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 HexMap() [1/2]

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

Constructor (dummy) for the HexMap class.

```
845 {
846     //...
847
848     std::cout « "HexMap dummy constructed at " « this « std::endl;
849
850     return;
851 }   /* HexMap(), dummy */
```

3.3.2.2 HexMap() [2/2]

```
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

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

```
888 {
889     //  1. set attributes
890
891     //  1.1. private
892     this->event_ptr = event_ptr;
893     this->render_window_ptr = render_window_ptr;
894
895     this->assets_manager_ptr = assets_manager_ptr;
896     this->message_hub_ptr = message_hub_ptr;
897
898     //  1.2. public
899     this->frame = 0;
900
901     this->n_layers = n_layers;
902     if (this->n_layers < 0) {
903         this->n_layers = 0;
904     }
905
906     this->position_x = 400;
907     this->position_y = 400;
908
909     //  2. assemble n layer hex map
910     this->__assembleHexMap();
911
912     //  3. set up and position drawable attributes
913     this->__setUpGlassScreen();
914
915     std::cout « "HexMap constructed at " « this « std::endl;
916
917     return;
918 }   /* HexMap(), intended */
```

3.3.2.3 ~HexMap()

```
HexMap::~HexMap (
            void  )
```

Destructor for the HexMap class.

```
1162 {
1163     this->clear();
1164
1165     std::cout « "HexMap at " « this « " destroyed" « std::endl;
1166
1167     return;
1168 }   /* ~HexMap() */
```

3.3.3 Member Function Documentation

3.3.3.1 __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(milliseconds_since_epoch);
765
766     //  2. lay tiles
767     this->__layTiles();
768
```

```
769     //  3. procedurally generate types
770     this->__proceduralllyGenerateTileTypes();
771
772     //  4. procedurally generate resources
773     this->__proceduralllyGenerateTileResources();
774
775     return;
776 }   /* __assembleHexMap() */
```

3.3.3.2 __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() */
```

3.3.3.3 __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

hex_ptr	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() */
```

3.3.3.4 __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

hex_ptr	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() */
```

3.3.3.5 __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

n_elements	The number of elements in the generated noise vector.
n_components	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() */
```

3.3.3.6 __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() */
```

3.3.3.7 __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

potential_x	The potential x position of the tile.
potential_y	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() */
```

3.3.3.8 __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() */
```

3.3.3.9 __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[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
68     this->tile_position_x_vec.push_back(hex_ptr->position_x);
69     this->tile_position_y_vec.push_back(hex_ptr->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() */
```

3.3.3.10 __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() */
```

3.3.3.11 __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() */
```

3.3.3.12 __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(40, 40, 40, 40));
37 
38     return;
39 }   /* __setUpGlassScreen() */
```

3.3.3.13 __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() */
```

3.3.3.14 assess()

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

Method to assess the resource of the selected tile.

```
933 {
934     HexTile* selected_tile_ptr = this->__getSelectedTile();
935     if (selected_tile_ptr != NULL) {
936         selected_tile_ptr->assess();
937     }
938 
939     return;
940 }   /* assess() */
```

3.3.3.15 clear()

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

Method to clear the hex map.

```
1124 {
1125     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
1126     std::map<double, HexTile*>::iterator hex_map_iter_y;
1127     for (
1128         hex_map_iter_x = this->hex_map.begin();
1129         hex_map_iter_x != this->hex_map.end();
1130         hex_map_iter_x++
1131     ) {
1132         for (
1133             hex_map_iter_y = hex_map_iter_x->second.begin();
1134             hex_map_iter_y != hex_map_iter_x->second.end();
1135             hex_map_iter_y++
1136         ) {
1137             delete hex_map_iter_y->second;
1138         }
1139     }
1140     this->hex_map.clear();
1141
1142     this->tile_position_x_vec.clear();
1143     this->tile_position_y_vec.clear();
1144     this->border_tiles_vec.clear();
1145
1146     return;
1147 }   /* clear() */
```

3.3.3.16 draw()

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

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

```
1080 {
1081     //  1. draw all tiles in order
1082     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
1083     std::map<double, HexTile*>::iterator hex_map_iter_y;
1084     for (
1085         hex_map_iter_x = this->hex_map.begin();
1086         hex_map_iter_x != this->hex_map.end();
1087         hex_map_iter_x++
1088     ) {
1089         for (
1090             hex_map_iter_y = hex_map_iter_x->second.begin();
1091             hex_map_iter_y != hex_map_iter_x->second.end();
1092             hex_map_iter_y++
1093         ) {
1094             hex_map_iter_y->second->draw();
1095         }
1096     }
1097
1098     //  2. redraw selected tile
1099     HexTile* selected_tile_ptr = this->__getSelectedTile();
1100     if (selected_tile_ptr != NULL) {
1101         selected_tile_ptr->draw();
1102     }
1103
1104     //  3. draw glass screen
1105     this->render_window_ptr->draw(this->glass_screen);
1106
1107     this->frame++;
1108     return;
1109 }   /* draw() */
```

3.3.3.17 processEvent()

```
void HexMap::processEvent (
            void  )
```

Method to process HexMap. To be called once per event.

```
1008 {
1009     //  1. process tiles
1010     std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
1011     std::map<double, HexTile*>::iterator hex_map_iter_y;
1012     for (
1013         hex_map_iter_x = this->hex_map.begin();
1014         hex_map_iter_x != this->hex_map.end();
1015         hex_map_iter_x++
1016     ) {
1017         for (
1018             hex_map_iter_y = hex_map_iter_x->second.begin();
1019             hex_map_iter_y != hex_map_iter_x->second.end();
1020             hex_map_iter_y++
1021         ) {
1022             hex_map_iter_y->second->processEvent();
1023         }
1024     }
1025
1026     //  2. handle inputs
1027     //...
1028
1029     return;
1030 }   /* processEvent() */
```

3.3.3.18 processFrame()

```
void HexMap::processFrame (
            void  )
```

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

```
1045 {
1046     //  1. process tiles
1047     std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
1048     std::map<double, HexTile*>::iterator hex_map_iter_y;
1049     for (
1050         hex_map_iter_x = this->hex_map.begin();
1051         hex_map_iter_x != this->hex_map.end();
1052         hex_map_iter_x++
1053     ) {
1054         for (
1055             hex_map_iter_y = hex_map_iter_x->second.begin();
1056             hex_map_iter_y != hex_map_iter_x->second.end();
1057             hex_map_iter_y++
1058         ) {
1059             hex_map_iter_y->second->processFrame();
1060         }
1061     }
1062
1063     return;
1064 }   /* processFrame() */
```

3.3.3.19 reroll()

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

Method to re-roll the hex map.

```
955 {
956     this->clear();
957     this->__assembleHexMap();
958
959     return;
960 }   /* reroll() */
```

3.3.3.20 toggleResourceOverlay()

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

Method to toggle the hex map resource overlay.

```
975 {
976     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
977     std::map<double, HexTile*>::iterator hex_map_iter_y;
978     for (
979         hex_map_iter_x = this->hex_map.begin();
980         hex_map_iter_x != this->hex_map.end();
981         hex_map_iter_x++
982     ) {
983         for (
984             hex_map_iter_y = hex_map_iter_x->second.begin();
985             hex_map_iter_y != hex_map_iter_x->second.end();
986             hex_map_iter_y++
987         ) {
988             hex_map_iter_y->second->toggleResourceOverlay();
989         }
990     }
991
992     return;
993 }   /* toggleResourceOverlay() */
```

3.3.4 Member Data Documentation

3.3.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.3.4.2 border_tiles_vec

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

A vector of pointers to the border tiles.

3.3.4.3 event_ptr

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

A pointer to the event class.

3.3.4.4 frame

```
int HexMap::frame
```

The current frame of this object.

3.3.4.5 glass_screen

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

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

3.3.4.6 hex_map

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

A position-indexed, nested map of hex tiles.

3.3.4.7 message_hub_ptr

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

A pointer to the message hub.

3.3.4.8 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

3.3.4.9 n_tiles

```
int HexMap::n_tiles
```

The number of tiles in the hex map.

3.3.4.10 position_x

```
double HexMap::position_x
```

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

3.3.4.11 position_y

```
double HexMap::position_y
```

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

3.3.4.12 render_window_ptr

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

A pointer to the render window.

3.3.4.13 tile_position_x_vec

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

A vector of tile x positions.

3.3.4.14 tile_position_y_vec

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

A vector of tile y position.

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

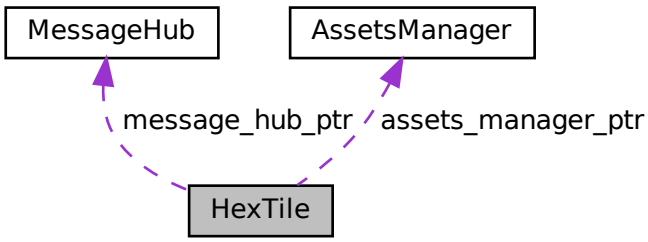
- header/HexMap.h
- source/HexMap.cpp

3.4 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 processFrame (void)

  Method to process HexTile. To be called once per frame.
- void draw (void)

  Method to draw the hex tile to the render window. To be called 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.
- 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.

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 tile of the hex map.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 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

position_x	The x position of the tile.
position_y	The y position of the tile.
event_ptr	Pointer to the event class.
render_window_ptr	Pointer to the render window.
assets_manager_ptr	Pointer to the assets manager.
message_hub_ptr	Pointer to the message hub.

```
300 {
301     //  1. set attributes
302
303     //  1.1. private
304     this->event_ptr = event_ptr;
305     this->render_window_ptr = render_window_ptr;
306
307     this->assets_manager_ptr = assets_manager_ptr;
308     this->message_hub_ptr = message_hub_ptr;
309
```

```
310     //  1.2. public
311     this->show_node = false;
312     this->show_resource = false;
313     this->resource_assessed = false;
314     this->is_selected = false;
315 
316     this->frame = 0;
317 
318     this->position_x = position_x;
319     this->position_y = position_y;
320 
321     this->major_radius = 32;
322     this->minor_radius = (sqrt(3) / 2) * this->major_radius;
323 
324     //  2. set up and position drawable attributes
325     this->__setUpNodeSprite();
326     this->__setUpTileSprite();
327     this->__setUpSelectOutlineSprite();
328     this->__setUpResourceChipSprite();
329     this->__setResourceText();
330 
331     //  3. set tile type and resource (default to forest and average)
332     this->setTileType(TileType :: FOREST);
333     this->setTileResource(TileResource :: AVERAGE);
334 
335     std::cout « "HexTile constructed at " « this « std::endl;
336 
337     return;
338 }   /* HexTile() */
```

3.4.2.2 ~HexTile()

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

Destructor for the HexTile class.

```
662 {
663     std::cout « "HexTile at " « this « " destroyed" « std::endl;
664 
665     return;
666 }   /* ~HexTile() */
```

3.4.3 Member Function Documentation

3.4.3.1 __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() */
```

3.4.3.2 __setResourceText()

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

Helper method to set up resource text.

```
159 {
160     this->resource_text.setFont(*(assets_manager_ptr->getFont("Glass_TTY_VT220")));
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 
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() */
```

3.4.3.3 __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() */
```

3.4.3.4 __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() */
```

3.4.3.5 __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(MONOCHROME_TEXT_RED);
113
114     this->select_outline_sprite.setFillColor(sf::Color(0, 0, 0, 0));
115
116     return;
117 }   /* __setUpSelectOutline() */
```

3.4.3.6 __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() */
```

3.4.3.7 assess()

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

Method to assess the tile's resource.

```
559 {
560     this->resource_assessed = true;
561     this->__setResourceText();
562
563     return;
564 }   /* assess() */
```

3.4.3.8 draw()

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

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

```
618 {
619     //  1. draw hex
620     this->render_window_ptr->draw(this->tile_sprite);
621 
622     //  2. draw node
623     if (this->show_node) {
624         this->render_window_ptr->draw(this->node_sprite);
625     }
626 
627     //  3. draw resource
628     if (this->show_resource) {
629         this->render_window_ptr->draw(this->resource_chip_sprite);
630         this->render_window_ptr->draw(this->resource_text);
631     }
632 
633     //  4. draw selection outline
634     if (this->is_selected) {
635         sf::Color outline_colour = this->select_outline_sprite.getOutlineColor();
636 
637         outline_colour.a =
638             255 * pow(cos((M_PI * this->frame) / (1.5 * FRAMES_PER_SECOND)), 2);
639 
640         this->select_outline_sprite.setOutlineColor(outline_colour);
641 
642         this->render_window_ptr->draw(this->select_outline_sprite);
643     }
644 
645     this->frame++;
646     return;
647 }   /* draw() */
```

3.4.3.9 processEvent()

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

Method to process HexTile. To be called once per event.

```
579 {
580     //...
581 
582     return;
583 }   /* processEvent() */
```

3.4.3.10 processFrame()

```
void HexTile::processFrame (
            void  )
```

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

```
598 {
599     //...
600 
601     return;
602 }   /* processFrame() */
```

3.4.3.11 setTileResource() [1/2]

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

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

Parameters

input_value	A numerical input in the closed interval [0, 1].

```
484 {
485     //  1. check input
486     if (input_value < 0 or input_value > 1) {
487         std::string error_str = "ERROR  HexTile::setTileResource()  given input value is ";
488         error_str += "not in the closed interval [0, 1]";
489 
490         #ifdef _WIN32
491             std::cout « error_str « std::endl;
492         #endif  /* _WIN32 */
493 
494         throw std::runtime_error(error_str);
495     }
496 
497     //  2. convert input value to tile resource
498     TileResource tile_resource;
499 
500     if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[0]) {
501         tile_resource = TileResource :: POOR;
502     }
503     else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[1]) {
504         tile_resource = TileResource :: BELOW_AVERAGE;
505     }
506     else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[2]) {
507         tile_resource = TileResource :: AVERAGE;
508     }
509     else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[3]) {
510         tile_resource = TileResource :: ABOVE_AVERAGE;
511     }
512     else {
513         tile_resource = TileResource :: GOOD;
514     }
515 
516     //  3. call alternate method
517     this->setTileResource(tile_resource);
518 
519     return;
520 }   /* setTileResource(double) */
```

3.4.3.12 setTileResource() [2/2]

```
void HexTile::setTileResource (
            TileResource tile_resource )
```

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

Parameters

tile_resource	The resource (TileResource) value to attribute to the tile.

```
462 {
463     this->tile_resource = tile_resource;
464     this->__setResourceText();
465 
466     return;
467 }   /* setTileResource(TileResource) */
```

3.4.3.13 setTileType() [1/2]

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

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

Parameters

input_value	A numerical input in the closed interval [0, 1].

```
412 {
413     //  1. check input
414     if (input_value < 0 or input_value > 1) {
415         std::string error_str = "ERROR  HexTile::setTileType()  given input value is ";
416         error_str += "not in the closed interval [0, 1]";
417 
418         #ifdef _WIN32
419             std::cout « error_str « std::endl;
420         #endif  /* _WIN32 */
421 
422         throw std::runtime_error(error_str);
423     }
424 
425     //  2. convert input value to tile type
426     TileType tile_type;
427 
428     if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[0]) {
429         tile_type = TileType :: LAKE;
430     }
431     else if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[1]) {
432         tile_type = TileType :: PLAINS;
433     }
434     else if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[2]) {
435         tile_type = TileType :: FOREST;
436     }
437     else {
438         tile_type = TileType :: MOUNTAINS;
439     }
440 
441     //  3. call alternate method
442     this->setTileType(tile_type);
443 
444     return;
445 }   /* setTileType(double) */
```

3.4.3.14 setTileType() [2/2]

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

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

Parameters

tile_type	The type (TileType) to set the tile to.

```
353 {
354     this->tile_type = tile_type;
355 
356     switch (this->tile_type) {
357         case (TileType :: FOREST): {
358             this->tile_sprite.setFillColor(FOREST_GREEN);
359 
360             break;
361         }
362 
363         case (TileType :: LAKE): {
364             this->tile_sprite.setFillColor(LAKE_BLUE);
365 
366             break;
367         }
368 
369         case (TileType :: MOUNTAINS): {
370             this->tile_sprite.setFillColor(MOUNTAINS_GREY);
371 
372             break;
373         }
374 
375         case (TileType :: OCEAN): {
```

```
376             this->tile_sprite.setFillColor(OCEAN_BLUE);
377
378             break;
379         }
380
381         case (TileType :: PLAINS): {
382             this->tile_sprite.setFillColor(PLAINS_YELLOW);
383
384             break;
385         }
386
387         default: {
388             // do nothing!
389
390             break;
391         }
392     }
393
394     return;
395 }   /* setTileType(TileType) */
```

3.4.3.15 toggleResourceOverlay()

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

Method to toggle the tile resource overlay.

```
535 {
536     if (this->show_resource) {
537         this->show_resource = false;
538     }
539     else {
540         this->show_resource = true;
541     }
542
543     return;
544 }   /* toggleResourceOverlay() */
```

3.4.4 Member Data Documentation

3.4.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.4.4.2 event_ptr

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

A pointer to the event class.

3.4.4.3 frame

```
int HexTile::frame
```

The current frame of this object.

3.4.4.4 is_selected

```
bool HexTile::is_selected
```

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

3.4.4.5 major_radius

```
double HexTile::major_radius
```

The radius of the smallest bounding circle.

3.4.4.6 message_hub_ptr

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

A pointer to the message hub.

3.4.4.7 minor_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

3.4.4.8 node_sprite

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

A circle shape to mark the tile node.

3.4.4.9 position_x

```
double HexTile::position_x
```

The x position of the tile.

3.4.4.10 position_y

```
double HexTile::position_y
```

The y position of the tile.

3.4.4.11 render_window_ptr

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

A pointer to the render window.

3.4.4.12 resource_assessed

```
bool HexTile::resource_assessed
```

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

3.4.4.13 resource_chip_sprite

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

A circle shape which represents a resource chip.

3.4.4.14 resource_text

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

A text representation of the resource.

3.4.4.15 select_outline_sprite

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

A convex shape which outlines the tile when selected.

3.4.4.16 show_node

```
bool HexTile::show_node
```

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

3.4.4.17 show_resource

```
bool HexTile::show_resource
```

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

3.4.4.18 tile_resource

```
TileResource HexTile::tile_resource
```

3.4.4.19 tile_sprite

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

A convex shape which represents the tile.

3.4.4.20 tile_type

```
TileType HexTile::tile_type
```

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

- header/HexTile.h
- source/HexTile.cpp

3.5 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.

3.5.1 Detailed Description

A structure which defines a standard message format.

3.5.2 Member Data Documentation

3.5.2.1 bool_payload_vec

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

A vector <bool> payload.

3.5.2.2 channel

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

A string identifying the appropriate channel for this message.

3.5.2.3 double_payload_vec

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

A vector <double> payload.

3.5.2.4 int_payload_vec

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

A vector <int> payload.

3.5.2.5 string_payload

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

A string payload.

3.5.2.6 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

3.6 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.
- 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 process (void)

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

3.6.1 Detailed Description

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

3.6.2 Constructor & Destructor Documentation

3.6.2.1 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() */
```

3.6.2.2 ~MessageHub()

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

Destructor for the MessageHub class.

```
310 {
311     this->clear();
312
313     std::cout « "MessageHub at " « this « " destroyed" « std::endl;
314
315     return;
316 }   /* ~MessageHub() */
```

3.6.3 Member Function Documentation

3.6.3.1 addChannel()

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

Method to add channel to message map.

Parameters

channel	The key for the message channel being added.

```
69 {
70     //  1. check if channel is in map (if so, throw error)
71     if (this->message_map.count(channel) > 0) {
72         std::string error_str = "ERROR  MessageHub::addChannel()  channel ";
73         error_str += channel;
74         error_str += " is already in message map";
75
76         #ifdef _WIN32
77             std::cout « error_str « std::endl;
78         #endif  /* _WIN32 */
79
80         throw std::runtime_error(error_str);
81     }
82
83     //  2. add channel to map
84     this->message_map[channel] = {};
85
86     return;
87 }   /* addChannel() */
```

3.6.3.2 clear()

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

Method to clear the MessageHub.

```
283 {
284
285     std::map<std::string, std::list<Message»::iterator map_iter;
286     for (
287         map_iter = this->message_map.begin();
```

```
288         map_iter != this->message_map.end();
289         map_iter++
290     ) {
291         map_iter->second.clear();
292     }
293     this->message_map.clear();
294 
295     return;
296 }   /* clear() */
```

3.6.3.3 isEmpty()

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

Method to check if channel is empty.

Parameters

channel	The key for the message channel being checked.

Returns

A boolean indicating whether the channel is empty or not.

```
179 {
180     //  1. check if channel is in map (if not, throw error)
181     if (this->message_map.count(channel) <= 0) {
182         std::string error_str = "ERROR  MessageHub::isEmpty()  channel ";
183         error_str += channel;
184         error_str += " is not in message map";
185 
186         #ifdef _WIN32
187             std::cout « error_str « std::endl;
188         #endif  /* _WIN32 */
189 
190         throw std::runtime_error(error_str);
191     }
192 
193     if (this->message_map[channel].empty()) {
194         return true;
195     }
196     else {
197         return false;
198     }
199 }   /* isEmpty() */
```

3.6.3.4 process()

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

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

```
264 {
265     //...
266 
267     return;
268 }   /* process() */
```

3.6.3.5 receiveMessage()

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

Method to receive the latest message in the given channel.

Parameters

channel	The key for the message channel being received from.

Returns

The latest message in the given channel.

```
218 {
219     //  1. check if channel is in map (if not, throw error)
220     if (this->message_map.count(channel) <= 0) {
221         std::string error_str = "ERROR  MessageHub::receiveMessage()  channel ";
222         error_str += channel;
223         error_str += " is not in message map";
224 
225         #ifdef _WIN32
226             std::cout << error_str << std::endl;
227         #endif  /* _WIN32 */
228 
229         throw std::runtime_error(error_str);
230     }
231 
232     //  2. check if channel is empty (if so, throw error)
233     if (this->message_map[channel].empty()) {
234         std::string error_str = "ERROR  MessageHub::receiveMessage()  channel ";
235         error_str += channel;
236         error_str += " is empty";
237 
238         #ifdef _WIN32
239             std::cout << error_str << std::endl;
240         #endif  /* _WIN32 */
241 
242         throw std::runtime_error(error_str);
243     }
244 
245     //  3. receive message
246     Message message = this->message_map[channel].back();
247 
248     return message;
249 }   /* receiveMessage() */
```

3.6.3.6 removeChannel()

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

Method to remove channel from message map.

Parameters

channel	The key for the message channel being removed.

```
104 {
105     //  1. check if channel is in map (if not, throw error)
106     if (this->message_map.count(channel) <= 0) {
107         std::string error_str = "ERROR  MessageHub::removeChannel()  channel ";
108         error_str += channel;
109         error_str += " is not in message map";
```

```
110
111         #ifdef _WIN32
112             std::cout « error_str « std::endl;
113         #endif  /* _WIN32 */
114
115         throw std::runtime_error(error_str);
116     }
117
118     //  2. remove channel from map
119     this->message_map[channel].clear();
120     this->message_map.erase(channel);
121
122     return;
123 }   /* removeChannel() */
```

3.6.3.7 sendMessage()

```
void MessageHub::sendMessage (
            Message message )
```

Method to send a message to the message map.

Parameters

message	The message to be sent.

```
140 {
141     //  1. check if channel is in map (if not, throw error)
142     std::string channel = message.channel;
143
144     if (this->message_map.count(channel) <= 0) {
145         std::string error_str = "ERROR  MessageHub::sendMessage()  channel ";
146         error_str += channel;
147         error_str += " is not in message map";
148
149         #ifdef _WIN32
150             std::cout « error_str « std::endl;
151         #endif  /* _WIN32 */
152
153         throw std::runtime_error(error_str);
154     }
155
156     //  2. send message to message map
157     this->message_map[channel].push_back(message);
158
159     return;
160 }   /* sendMessage() */
```

3.6.4 Member Data Documentation

3.6.4.1 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

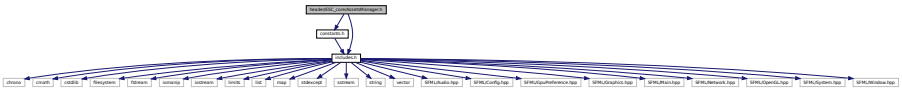
File Documentation

4.1 header/ESC_core/AssetsManager.h File Reference

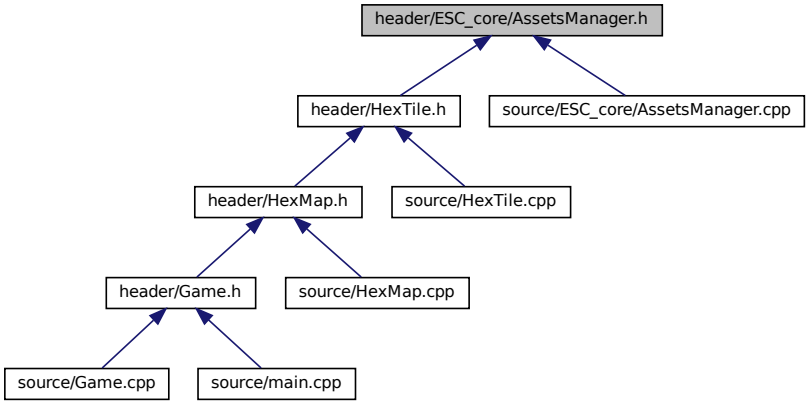
Header file for the AssetsManager class.

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

Include dependency graph for AssetsManager.h:



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



Classes

- class AssetsManager

  A class which manages visual and sound assets.

4.1.1 Detailed Description

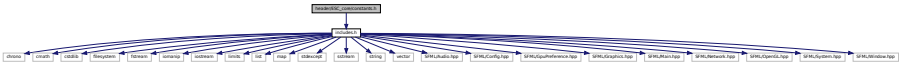
Header file for the AssetsManager class.

4.2 header/ESC_core/constants.h File Reference

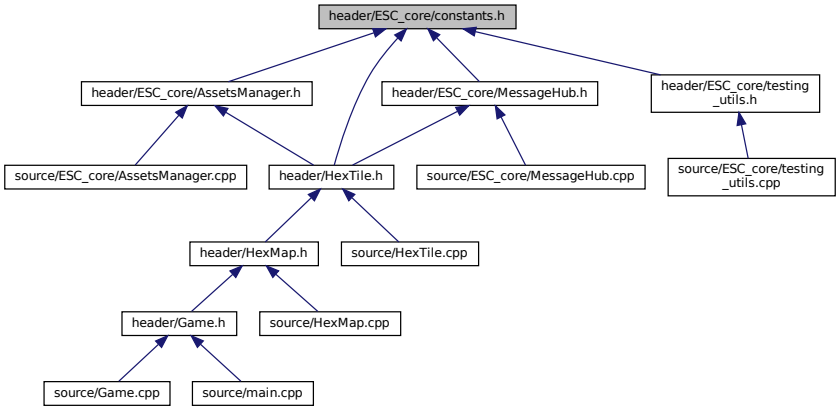
Header file for various constants.

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

Include dependency graph for constants.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 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).

4.2.1 Detailed Description

Header file for various constants.

4.2.2 Function Documentation

4.2.2.1 FOREST_GREEN()

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

The base colour of a forest tile.

4.2.2.2 LAKE_BLUE()

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

The base colour of a lake (water) tile.

4.2.2.3 MENU_FRAME_GREY()

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

The base colour of the context menu frame.

4.2.2.4 MONOCHROME_SCREEN_BACKGROUND()

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

The base colour of old monochrome screens.

4.2.2.5 MONOCHROME_TEXT_AMBER()

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

The base colour of old monochrome text (amber).

4.2.2.6 MONOCHROME_TEXT_GREEN()

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

The base colour of old monochrome text (green).

4.2.2.7 MONOCHROME_TEXT_RED()

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

The base colour of old monochrome text (red).

4.2.2.8 MOUNTAINS_GREY()

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

The base colour of a mountains tile.

4.2.2.9 OCEAN_BLUE()

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

The base colour of an ocean (water) tile.

4.2.2.10 PLAINS_YELLOW()

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

The base colour of a plains tile.

4.2.2.11 VISUAL_SCREEN_FRAME_GREY()

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

The base colour of the framing of the visual screen.

4.2.3 Variable Documentation

4.2.3.1 FLOAT_TOLERANCE

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

Tolerance for floating point equality tests.

4.2.3.2 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

4.2.3.3 GAME_HEIGHT

```
const int GAME_HEIGHT = 800
```

Height of the game space.

4.2.3.4 GAME_WIDTH

```
const int GAME_WIDTH = 1200
```

Width of the game space.

4.2.3.5 SECONDS_PER_FRAME

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

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

4.2.3.6 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).

4.2.3.7 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).

4.3 header/ESC_core/doxygen_cite.h File Reference

Header file which simply cites the doxygen tool.

4.3.1 Detailed Description

Header file which simply cites the doxygen tool.

Ref: van Heesch. [2023]

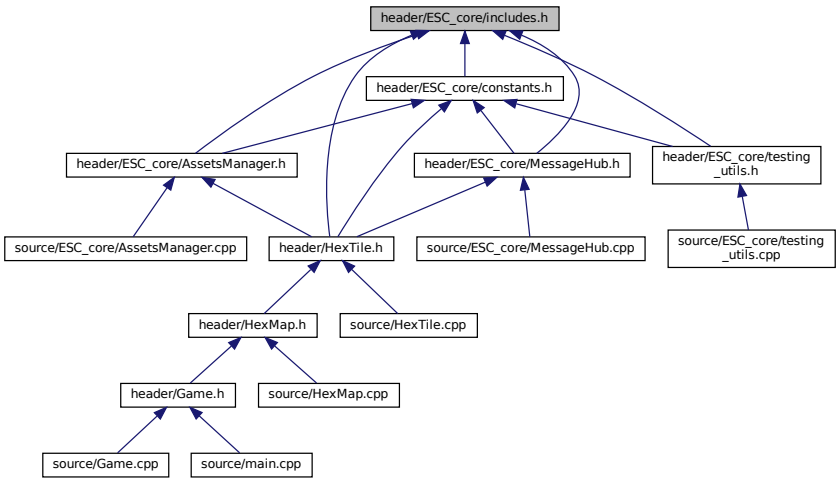
4.4 header/ESC_core/includes.h File Reference

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:



4.4.1 Detailed Description

Header file for various includes.

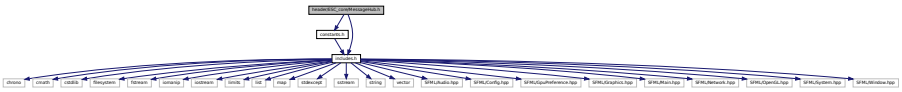
Ref: Gomila [2023]

4.5 header/ESC_core/MessageHub.h File Reference

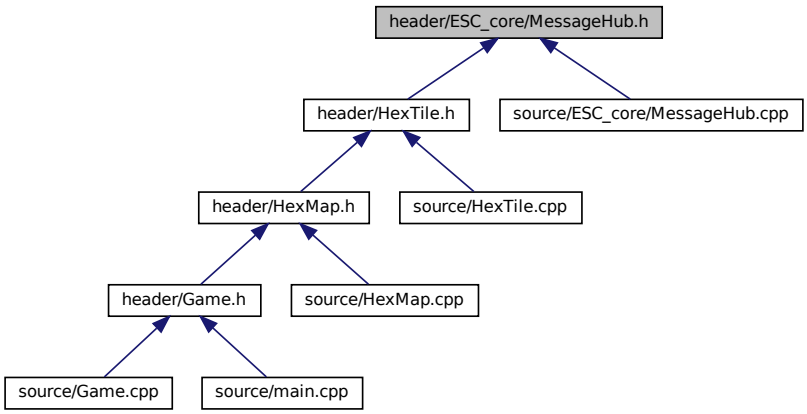
Header file for the MessageHub class.

```
#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.

4.5.1 Detailed Description

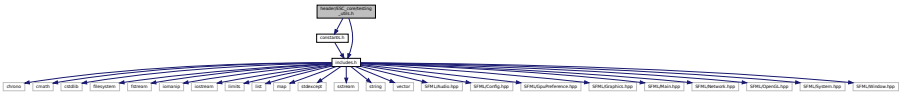
Header file for the MessageHub class.

4.6 header/ESC_core/testing_utils.h File Reference

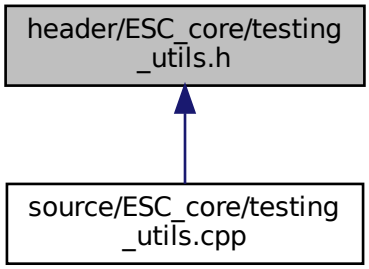
Header file for various testing utilities.

```
#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 >= y.
- void testLessThan (double, double, std::string, int)

  Tests if x < y.
- void testLessThanOrEqualTo (double, double, std::string, int)

  Tests if x <= 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.

4.6.1 Detailed Description

Header file for various testing utilities.

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

4.6.2 Function Documentation

4.6.2.1 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

file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	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() */
```

4.6.2.2 printGold()

```
void printGold (
            std::string input_str )
```

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

Parameters

input_str	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() */
```

4.6.2.3 printGreen()

```
void printGreen (
            std::string input_str )
```

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

Parameters

input_str	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() */
```

4.6.2.4 printRed()

```
void printRed (
            std::string input_str )
```

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

Parameters

input_str	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() */
```

4.6.2.5 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

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

4.6.2.6 testGreaterThan()

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

Tests if x > y.

Parameters

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

4.6.2.7 testGreaterThanOrEqualTo()

```
void testGreaterThanOrEqualTo (
             double x,
```

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

Tests if x >= y.

Parameters

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

4.6.2.8 testLessThan()

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

Tests if x < y.

Parameters

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

4.6.2.9 testLessThanOrEqualTo()

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

Tests if x <= y.

Parameters

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

4.6.2.10 testTruth()

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

Tests if the given statement is true.

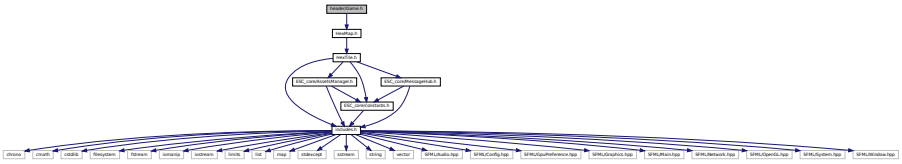
Parameters

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

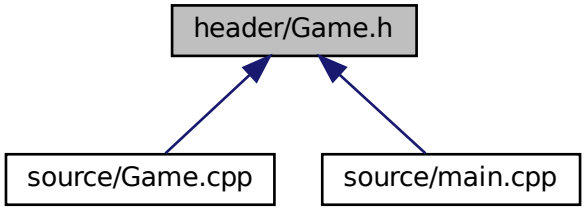
```
390 {
391     if (statement) {
392         return;
393     }
394
395     std::string error_str = "ERROR: testTruth():\t in ";
396     error_str += file;
397     error_str += "\tline ";
398     error_str += std::to_string(line);
399     error_str += ":\t\n";
400     error_str += "Given statement is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

4.7 header/Game.h File Reference

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



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



Classes

- class Game

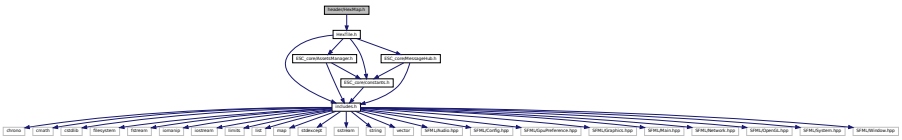
  A class which acts as the central class for the game, by containing all other classes and implementing the game loop.

4.8 header/HexMap.h File Reference

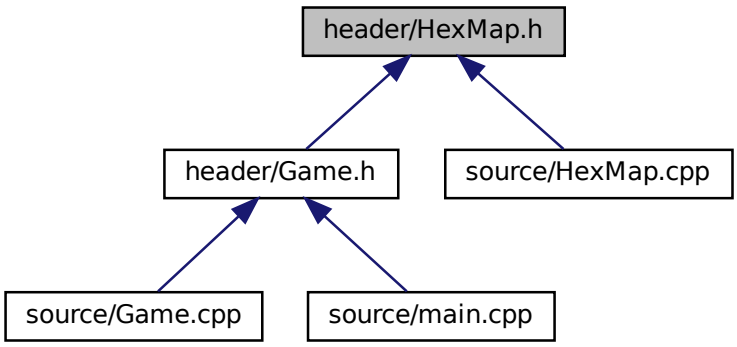
Header file for the HexMap class.

```
#include "HexTile.h"
```

Include dependency graph for HexMap.h:



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



Classes

- class HexMap

  A class which defines a hex map of hex tiles.

4.8.1 Detailed Description

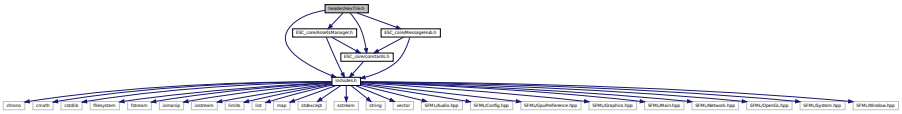
Header file for the HexMap class.

4.9 header/HexTile.h File Reference

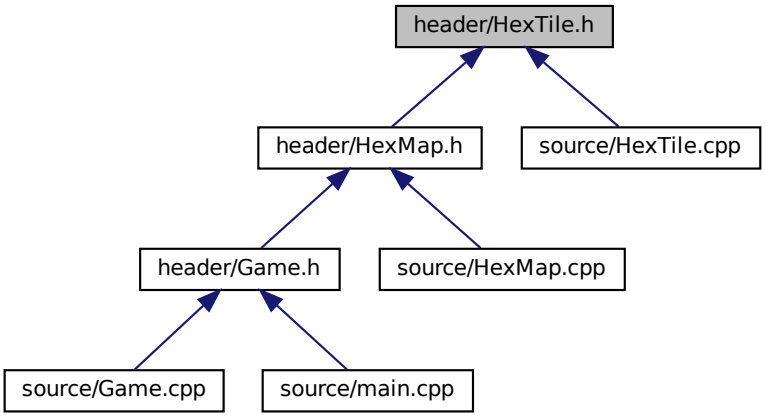
Header file for the Game class.

```
#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.

4.9.1 Detailed Description

Header file for the Game class.

Header file for the HexTile class.

4.9.2 Enumeration Type Documentation

4.9.2.1 TileResource

```
enum TileResource
```

An enumeration of the different tile resource values.

Enumerator

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

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

4.9.2.2 TileType

```
enum TileType
```

An enumeration of the different tile types.

Enumerator

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

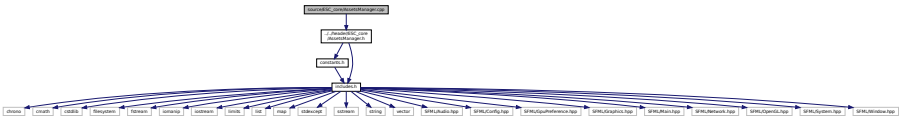
```
34              {
```

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

4.10 source/ESC_core/AssetsManager.cpp File Reference

Implementation file for the AssetsManager class.

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



4.10.1 Detailed Description

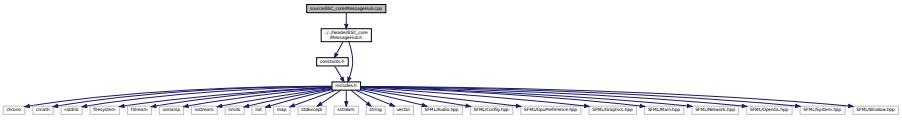
Implementation file for the AssetsManager class.

A class which manages visual and sound assets.

4.11 source/ESC_core/MessageHub.cpp File Reference

Implementation file for the MessageHub class.

`#include "../../header/ESC_core/MessageHub.h"`
Include dependency graph for MessageHub.cpp:



4.11.1 Detailed Description

Implementation file for the MessageHub class.

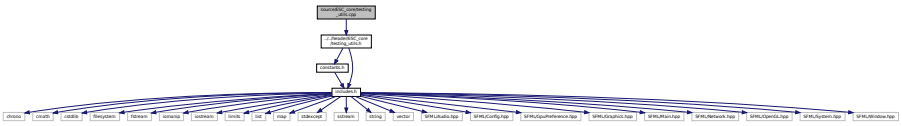
A class which acts as a central hub for inter-object message traffic.

4.12 source/ESC_core/testing_utils.cpp File Reference

Implementation file for various testing utilities.

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

Include dependency graph for testing_utils.cpp:



Functions

- void printGreen (std::string input_str)

  A function that sends green text to std::cout.
- void printGold (std::string input_str)

  A function that sends gold text to std::cout.
- void printRed (std::string input_str)

  A function that sends red text to std::cout.
- 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).
- void testGreaterThan (double x, double y, std::string file, int line)

  Tests if x > y.
- void testGreaterThanOrEqualTo (double x, double y, std::string file, int line)

  Tests if x >= y.
- void testLessThan (double x, double y, std::string file, int line)

  Tests if x < y.
- void testLessThanOrEqualTo (double x, double y, std::string file, int line)

  Tests if x <= y.
- void testTruth (bool statement, std::string file, int line)

  Tests if the given statement is true.
- 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.

4.12.1 Detailed Description

Implementation file for various testing utilities.

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

4.12.2 Function Documentation

4.12.2.1 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

file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	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() */
```

4.12.2.2 printGold()

```
void printGold (
             std::string input_str )
```

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

Parameters

input_str	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() */
```

4.12.2.3 printGreen()

```
void printGreen (
             std::string input_str )
```

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

Parameters

input_str	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() */
```

4.12.2.4 printRed()

```
void printRed (
```

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

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

Parameters

input_str	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() */
```

4.12.2.5 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

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

4.12.2.6 testGreaterThan()

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

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

Tests if x > y.

Parameters

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

4.12.2.7 testGreaterThanOrEqualTo()

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

Tests if x >= y.

Parameters

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

4.12.2.8 testLessThan()

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

Tests if x < y.

Parameters

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

4.12.2.9 testLessThanOrEqualTo()

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

Tests if x <= y.

Parameters

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

4.12.2.10 testTruth()

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

Tests if the given statement is true.

Parameters

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

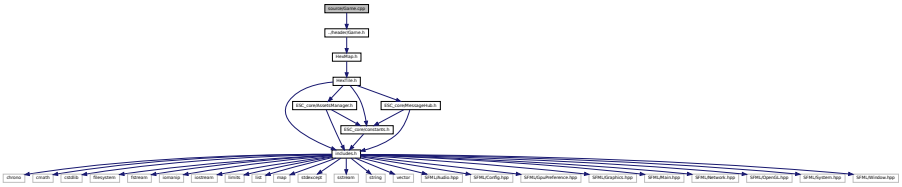
```
390 {
391     if (statement) {
392         return;
393     }
394
395     std::string error_str = "ERROR: testTruth():\t in ";
396     error_str += file;
397     error_str += "\tline ";
398     error_str += std::to_string(line);
399     error_str += ":\t\n";
400     error_str += "Given statement is 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() */
```

4.13 source/Game.cpp File Reference

Implementation file for the Game class.

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

Include dependency graph for Game.cpp:



4.13.1 Detailed Description

Implementation file for the Game class.

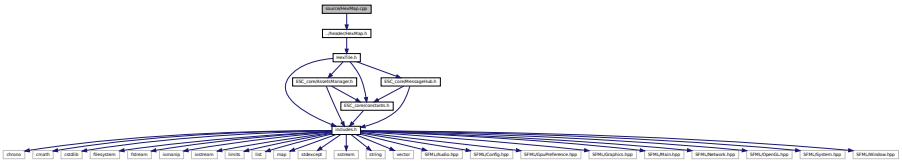
A class which defines a tile of a hex map.

4.14 source/HexMap.cpp File Reference

Implementation file for the HexMap class.

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

Include dependency graph for HexMap.cpp:



4.14.1 Detailed Description

Implementation file for the HexMap class.

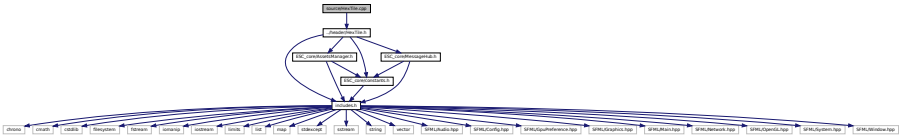
A class which defines a hex map of hex tiles.

4.15 source/HexTile.cpp File Reference

Implementation file for the HexTile class.

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

Include dependency graph for HexTile.cpp:



4.15.1 Detailed Description

Implementation file for the HexTile class.

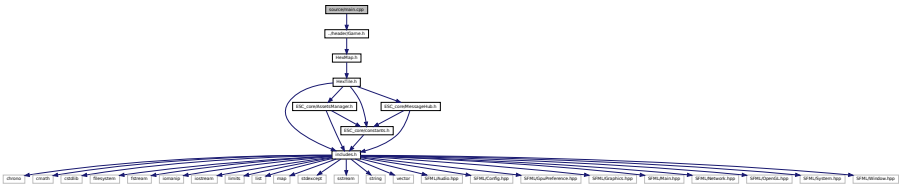
A class which defines a tile of a hex map.

4.16 source/main.cpp File Reference

Implementation file for main() for Road To Zero.

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

Include dependency graph for main.cpp:



Functions

- void loadAssets (AssetsManager *assets_manager_ptr)

  Helper function to load game assets.
- sf::RenderWindow * constructRenderWindow (void)

  Helper function to construct render window.
- int main (int argc, char **argv)

4.16.1 Detailed Description

Implementation file for main() for Road To Zero.

4.16.2 Function Documentation

4.16.2.1 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() */
```

4.16.2.2 loadAssets()

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

Helper function to load game assets.

Parameters

assets_manager_ptr	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() */
```

4.16.2.3 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     delete render_window_ptr;
88
89     return 0;
90 }   /* main() */
```

Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL https://www.sfml-dev.org/. 73

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL https://www.doxygen.nl. 71

Wikipedia. Hexagon, 2023. URL https://en.wikipedia.org/wiki/Hexagon. 45

Index