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:

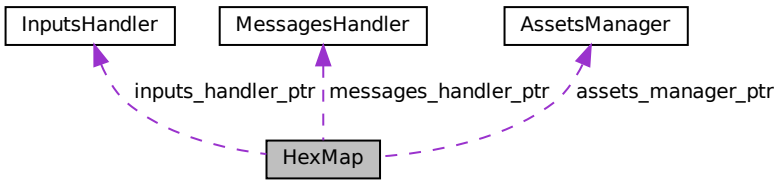
- header/ESC_core/AssetsManager.h
- source/ESC_core/AssetsManager.cpp

3.2 HexMap Class Reference

A class which defines a hex map of hex tiles.

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

Collaboration diagram for HexMap:



Public Member Functions

- HexMap (int, AssetsManager *, InputsHandler *, MessagesHandler *)

  Constructor for the HexMap class.
- void process (void)

  Method to process HexMap. To be called once per frame;.
- void reroll (void)

  Method to re-roll the hex map.
- void draw (sf::RenderWindow *)

  Method to draw the hex map to the render window. To be called only once per frame!
- void clear (void)

  Method to clear the hex map.
- ~HexMap (void)

  Destructor for the HexMap class.

Public Attributes

- int n_layers

  The number of layers in the hex map.
- int n_tiles

  The number of tiles in the hex map.
- int frame

  The current frame of this object.
- double position_x

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

  The y position of the hex map's origin (i.e. central) tile.
- std::vector< double > tile_position_x_vec

  A vector of tile x positions.
- std::vector< double > tile_position_y_vec

  A vector of tile y position.
- std::vector< HexTile * > border_tiles_vec

  A vector of pointers to the border tiles.
- std::map< double, std::map< double, HexTile * > > hex_map

  A position-indexed, nested map of hex tiles.

Private Member Functions

- void __layTiles (void)

  Helper method to lay the hex tiles down to generate the game world.
- std::vector< double > __getNoise (int, int=64)

  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)
- 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)
- void __assembleHexMap (void)

  Helper method to assemble the hex map.

Private Attributes

- AssetsManager * assets_manager_ptr

  A pointer to the assets manager.
- InputsHandler * inputs_handler_ptr

  A pointer to the inputs handler.
- MessagesHandler * messages_handler_ptr

  A pointer to the messages handler.

3.2.1 Detailed Description

A class which defines a hex map of hex tiles.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 HexMap()

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

Constructor for the HexMap class.

Parameters

n_layers	The number of layers in the HexMap.
assets_manager_ptr	Pointer to the assets manager.
inputs_handler_ptr	Pointer to the inputs handler.
messages_handler_ptr	Pointer to the messages handler.

```
573 {
574     //  1. set attributes
575     this->assets_manager_ptr = assets_manager_ptr;
576     this->inputs_handler_ptr = inputs_handler_ptr;
577     this->messages_handler_ptr = messages_handler_ptr;
578 
579     this->frame = 0;
580 
581     this->n_layers = n_layers;
582     if (this->n_layers < 0) {
583         this->n_layers = 0;
584     }
585 
586     this->position_x = 400;
587     this->position_y = 400;
588 
589     //  2. assemble n layer hex map
590     this->__assembleHexMap();
591 
592     std::cout « "HexMap constructed at " « this « std::endl;
593 
594     return;
595 }   /* HexMap() */
```

3.2.2.2 ~HexMap()

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

Destructor for the HexMap class.

```
744 {
745     this->clear();
```

```
746
747     std::cout « "HexMap at " « this « " destroyed" « std::endl;
748
749     return;
750 }   /* ~HexMap() */
```

3.2.3 Member Function Documentation

3.2.3.1 __assembleHexMap()

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

Helper method to assemble the hex map.

```
518 {
519     //  1. seed RNG
520     unsigned long long int milliseconds_since_epoch =
521         std::chrono::duration_cast<std::chrono::milliseconds>(
522             std::chrono::system_clock::now().time_since_epoch()
523         ).count();
524     srand(milliseconds_since_epoch);
525
526     //  2. lay tiles
527     this->__layTiles();
528
529     //  3. procedurally generate types
530     this->__proceduralllyGenerateTileTypes();
531
532     //  4. procedurally generate resources
533     //...
534
535     return;
536 }   /* __assembleHexMap() */
```

3.2.3.2 __enforceOceanContinuity()

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

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

```
469 {
470     std::cout « "enforcing ..." « std::endl;
471
472     bool tile_changed = false;
473
474     //  1. scan tiles and enforce (where appropriate)
475     std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
476     std::map<double, HexTile*>::iterator hex_map_iter_y;
477     HexTile* hex_ptr;
478     for (
479         hex_map_iter_x = this->hex_map.begin();
480         hex_map_iter_x != this->hex_map.end();
481         hex_map_iter_x++
482     ) {
483         for (
484             hex_map_iter_y = hex_map_iter_x->second.begin();
485             hex_map_iter_y != hex_map_iter_x->second.end();
486             hex_map_iter_y++
487         ) {
488             hex_ptr = hex_map_iter_y->second;
489
490             if (this->__isLakeTouchingOcean(hex_ptr)) {
491                 hex_ptr->setTileType(TileType :: OCEAN);
492                 tile_changed = true;
```

```
493             }
494         }
495     }
496
497     if (tile_changed) {
498         this->__enforceOceanContinuity();
499     }
500     else {
501         return;
502     }
503 }   /* __enforceOceanContinuity() */
```

3.2.3.3 __getNoise()

```
std::vector< double > HexMap::__getNoise (
            int n_elements,
            int n_components = 64 )  [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].

```
220 {
221     //  1. generate random amplitude, wave number, direction, and phase vectors
222     std::vector<double> random_amplitude_vec(n_components, 0);
223     std::vector<double> random_wave_number_vec(n_components, 0);
224     std::vector<double> random_direction_vec(n_components, 0);
225     std::vector<double> random_phase_vec(n_components, 0);
226
227     for (int i = 0; i < n_components; i++) {
228         random_amplitude_vec[i] = AMPLITUDE_BASE * (double)rand() / RAND_MAX;
229
230         random_wave_number_vec[i] = WAVE_NUMBER_BASE * ((double)rand() / RAND_MAX);
231
232         random_direction_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
233
234         random_phase_vec[i] = PHASE_BASE * ((double)rand() / RAND_MAX);
235     }
236
237     //  2. generate noise vec
238     double amp = 0;
239     double wave_no = 0;
240     double dir = 0;
241     double phase = 0;
242
243     double x = 0;
244     double y = 0;
245
246     double max_noise = -1 * std::numeric_limits<double>::infinity();
247     double min_noise = std::numeric_limits<double>::infinity();
248
249     double noise = 0;
250     std::vector<double> noise_vec(n_elements, 0);
251
252     for (int i = 0; i < n_elements; i++) {
253         x = this->tile_position_x_vec[i] - this->position_x;
254         y = this->tile_position_y_vec[i] - this->position_y;
255
256         for (int j = 0; j < n_components; j++) {
257             amp = random_amplitude_vec[j];
258             wave_no = random_wave_number_vec[j];
259             dir = random_direction_vec[j];
```

```
260             phase = random_phase_vec[j];
261
262             noise += amp * cos(wave_no * (x * sin(dir) + y * cos(dir)) + phase);
263         }
264
265         noise_vec[i] = noise;
266
267         if (noise > max_noise) {
268             max_noise = noise;
269         }
270
271         else if (noise < min_noise) {
272             min_noise = noise;
273         }
274
275         noise = 0;
276     }
277
278     //  3. normalize noise vec
279     for (int i = 0; i < n_elements; i++) {
280         noise_vec[i] = (noise_vec[i] - min_noise) / (max_noise - min_noise);
281
282         if (noise_vec[i] < 0) {
283             noise_vec[i] = 0;
284         }
285         else if (noise_vec[i] > 1) {
286             noise_vec[i] = 1;
287         }
288     }
289
290     return noise_vec;
291 }   /* __getNoise() */
```

3.2.3.4 __getValidMapIndexPositions()

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

```
354 {
355     std::vector<double> map_index_positions = {-1, -1};
356
357     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
358     std::map<double, HexTile*>::iterator hex_map_iter_y;
359     HexTile* hex_ptr;
360
361     double distance = 0;
362
363     for (
364         hex_map_iter_x = this->hex_map.begin();
365         hex_map_iter_x != this->hex_map.end();
366         hex_map_iter_x++
367     ) {
368         for (
369             hex_map_iter_y = hex_map_iter_x->second.begin();
370             hex_map_iter_y != hex_map_iter_x->second.end();
371             hex_map_iter_y++
372         ) {
373             hex_ptr = hex_map_iter_y->second;
374
375             distance = sqrt(
376                 pow(hex_ptr->position_x - potential_x, 2) +
377                 pow(hex_ptr->position_y - potential_y, 2)
378             );
379
380             if (distance <= hex_ptr->minor_radius / 4) {
381                 map_index_positions = {hex_ptr->position_x, hex_ptr->position_y};
382                 return map_index_positions;
383             }
384         }
385     }
386
387     return map_index_positions;
388 }   /* __isInHexMap() */
```

3.2.3.5 __isLakeTouchingOcean()

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

```
407 {
408     //  1. if not lake tile, return
409     if (not (hex_ptr->tile_type == TileType :: LAKE)) {
410         return false;
411     }
412 
413     //  2. build potential neighbour positions
414     std::vector<double> potential_neighbour_x_vec(6, 0);
415     std::vector<double> potential_neighbour_y_vec(6, 0);
416 
417     for (int i = 0; i < 6; i++) {
418         potential_neighbour_x_vec[i] = hex_ptr->position_x +
419             2 * hex_ptr->minor_radius * cos((60 * i) * (M_PI / 180));
420 
421         potential_neighbour_y_vec[i] = hex_ptr->position_y +
422             2 * hex_ptr->minor_radius * sin((60 * i) * (M_PI / 180));
423     }
424 
425     //  3. scan neighbours for ocean tiles
426     double potential_x = 0;
427     double potential_y = 0;
428     std::vector<double> map_index_positions = {-1, -1};
429 
430     for (int i = 0; i < 6; i++) {
431         potential_x = potential_neighbour_x_vec[i];
432         potential_y = potential_neighbour_y_vec[i];
433 
434         map_index_positions = this->__getValidMapIndexPositions(
435             potential_x,
436             potential_y
437         );
438 
439         if (map_index_positions[0] == -1) {
440             continue;
441         }
442 
443         if (
444             this->hex_map[map_index_positions[0]][map_index_positions[1]]->tile_type ==
445             TileType :: OCEAN
446         ) {
447             return true;
448         }
449     }
450 
451     return false;
452 }   /* __isLakeTouchingOcean() */
```

3.2.3.6 __layTiles()

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

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

```
34 {
35     this->n_tiles = 0;
36 
37     //  1. add origin tile
38     HexTile* hex_ptr = new HexTile(
39         this->position_x,
40         this->position_y,
41         this->assets_manager_ptr,
42         this->inputs_handler_ptr,
43         this->messages_handler_ptr
44     );
45 
46     this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
47     this->tile_position_x_vec.push_back(hex_ptr->position_x);
48     this->tile_position_y_vec.push_back(hex_ptr->position_y);
49     this->n_tiles++;
50 
51 
52     //  2. fill out first row (reflect across origin tile)
```

```
53      for (int i = 0; i < this->n_layers; i++) {
54          hex_ptr = new HexTile(
55              this->position_x + 2 * (i + 1) * hex_ptr->minor_radius,
56              this->position_y,
57              this->assets_manager_ptr,
58              this->inputs_handler_ptr,
59              this->messages_handler_ptr
60          );
61
62          this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
63          this->tile_position_x_vec.push_back(hex_ptr->position_x);
64          this->tile_position_y_vec.push_back(hex_ptr->position_y);
65          this->n_tiles++;
66
67          if (i == this->n_layers - 1) {
68              this->border_tiles_vec.push_back(hex_ptr);
69          }
70
71          hex_ptr = new HexTile(
72              this->position_x - 2 * (i + 1) * hex_ptr->minor_radius,
73              this->position_y,
74              this->assets_manager_ptr,
75              this->inputs_handler_ptr,
76              this->messages_handler_ptr
77          );
78
79          this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
80          this->tile_position_x_vec.push_back(hex_ptr->position_x);
81          this->tile_position_y_vec.push_back(hex_ptr->position_y);
82          this->n_tiles++;
83
84          if (i == this->n_layers - 1) {
85              this->border_tiles_vec.push_back(hex_ptr);
86          }
87      }
88
89
90      //  3. fill out subsequent rows (reflect across first row)
91      HexTile* first_row_left_tile = hex_ptr;
92
93      int offset_count = 1;
94
95      double x_offset = 0;
96      double y_offset = 0;
97
98      for (
99          int row_width = 2 * this->n_layers;
100          row_width > this->n_layers;
101          row_width--
102      ) {
103          //  3.1. upper row
104          x_offset = first_row_left_tile->position_x +
105              2 * offset_count * first_row_left_tile->minor_radius *
106              cos(60 * (M_PI / 180));
107
108          y_offset = first_row_left_tile->position_y -
109              2 * offset_count * first_row_left_tile->minor_radius *
110              sin(60 * (M_PI / 180));
111
112          hex_ptr = new HexTile(
113              x_offset,
114              y_offset,
115              this->assets_manager_ptr,
116              this->inputs_handler_ptr,
117              this->messages_handler_ptr
118          );
119
120          this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
121          this->tile_position_x_vec.push_back(hex_ptr->position_x);
122          this->tile_position_y_vec.push_back(hex_ptr->position_y);
123          this->n_tiles++;
124
125          this->border_tiles_vec.push_back(hex_ptr);
126
127          for (int i = 1; i < row_width; i++) {
128              x_offset += 2 * first_row_left_tile->minor_radius;
129
130              hex_ptr = new HexTile(
131                  x_offset,
132                  y_offset,
133                  this->assets_manager_ptr,
134                  this->inputs_handler_ptr,
135                  this->messages_handler_ptr
136              );
137
138              this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
139              this->tile_position_x_vec.push_back(hex_ptr->position_x);
```

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

3.2.3.7 __proceduralllyGenerateTileResources()

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

3.2.3.8 __proceduralllyGenerateTileTypes()

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

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

```
306 {
307     //  1. get noise vec
308     std::vector<double> noise_vec = this->__getNoise(this->n_tiles);
309 
310     //  2. set tile types based on noise
311     int noise_idx = 0;
312 
313     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
314     std::map<double, HexTile*>::iterator hex_map_iter_y;
315     for (
316         hex_map_iter_x = this->hex_map.begin();
317         hex_map_iter_x != this->hex_map.end();
318         hex_map_iter_x++
319     ) {
320         for (
321             hex_map_iter_y = hex_map_iter_x->second.begin();
322             hex_map_iter_y != hex_map_iter_x->second.end();
323             hex_map_iter_y++
324         ) {
325             hex_map_iter_y->second->setTileType(noise_vec[noise_idx]);
326             noise_idx++;
327         }
328     }
329 
330     //  3. set border tile type to ocean
331     for (size_t i = 0; i < this->border_tiles_vec.size(); i++) {
332         this->border_tiles_vec[i]->setTileType(TileType :: OCEAN);
333     }
334 
335     //  4. enforce ocean continuity (i.e. all lake tiles touching ocean become ocean)
336     this->__enforceOceanContinuity();
337 
338     return;
339 }   /* __procedurallyGenerateTileTypes() */
```

3.2.3.9 clear()

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

Method to clear the hex map.

```
706 {
707     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
708     std::map<double, HexTile*>::iterator hex_map_iter_y;
709     for (
710         hex_map_iter_x = this->hex_map.begin();
711         hex_map_iter_x != this->hex_map.end();
712         hex_map_iter_x++
713     ) {
714         for (
715             hex_map_iter_y = hex_map_iter_x->second.begin();
716             hex_map_iter_y != hex_map_iter_x->second.end();
717             hex_map_iter_y++
718         ) {
719             delete hex_map_iter_y->second;
720         }
721     }
722     this->hex_map.clear();
723 
724     this->tile_position_x_vec.clear();
725     this->tile_position_y_vec.clear();
726     this->border_tiles_vec.clear();
727 
728     return;
729 }   /* clear() */
```

3.2.3.10 draw()

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

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

Parameters

window_ptr	A pointer to the render window.

```
672 {
673     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
674     std::map<double, HexTile*>::iterator hex_map_iter_y;
675     for (
676         hex_map_iter_x = this->hex_map.begin();
677         hex_map_iter_x != this->hex_map.end();
678         hex_map_iter_x++
679     ) {
680         for (
681             hex_map_iter_y = hex_map_iter_x->second.begin();
682             hex_map_iter_y != hex_map_iter_x->second.end();
683             hex_map_iter_y++
684         ) {
685             hex_map_iter_y->second->draw(window_ptr);
686         }
687     }
688
689     this->frame++;
690     return;
691 }   /* draw() */
```

3.2.3.11 process()

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

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

```
610 {
611     //  1. handle inputs
612     if (inputs_handler_ptr->key_pressed_once_vec[sf::Keyboard::R]) {
613         this->reroll();
614     }
615
616     //  2. process tiles
617     std::map<double, std::map<double, HexTile*>::iterator hex_map_iter_x;
618     std::map<double, HexTile*>::iterator hex_map_iter_y;
619     for (
620         hex_map_iter_x = this->hex_map.begin();
621         hex_map_iter_x != this->hex_map.end();
622         hex_map_iter_x++
623     ) {
624         for (
625             hex_map_iter_y = hex_map_iter_x->second.begin();
626             hex_map_iter_y != hex_map_iter_x->second.end();
627             hex_map_iter_y++
628         ) {
629             hex_map_iter_y->second->process();
630         }
631     }
632
633     return;
634 }   /* process() */
```

3.2.3.12 reroll()

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

Method to re-roll the hex map.

```
649 {
650     this->clear();
651     this->__assembleHexMap();
652
653     return;
654 }   /* reroll() */
```

3.2.4 Member Data Documentation

3.2.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.2.4.2 border_tiles_vec

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

A vector of pointers to the border tiles.

3.2.4.3 frame

```
int HexMap::frame
```

The current frame of this object.

3.2.4.4 hex_map

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

A position-indexed, nested map of hex tiles.

3.2.4.5 inputs_handler_ptr

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

A pointer to the inputs handler.

3.2.4.6 messages_handler_ptr

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

A pointer to the messages handler.

3.2.4.7 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

3.2.4.8 n_tiles

```
int HexMap::n_tiles
```

The number of tiles in the hex map.

3.2.4.9 position_x

```
double HexMap::position_x
```

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

3.2.4.10 position_y

```
double HexMap::position_y
```

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

3.2.4.11 tile_position_x_vec

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

A vector of tile x positions.

3.2.4.12 tile_position_y_vec

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

A vector of tile y position.

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

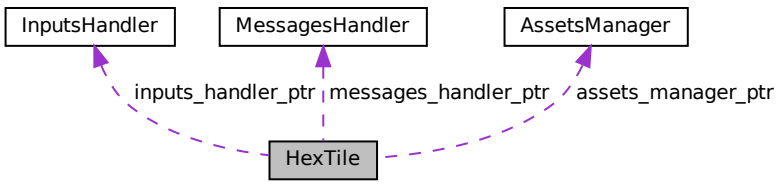
- header/HexMap/HexMap.h
- source/HexMap/HexMap.cpp

3.3 HexTile Class Reference

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

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

Collaboration diagram for HexTile:



Public Member Functions

- HexTile (double, double, AssetsManager ∗, InputsHandler ∗, MessagesHandler ∗)

  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)
- void process (void)

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

  Method to draw the hex tile to the render window. To be called only once per frame!
- ∼HexTile (void)

  Destructor for the HexTile class.

Public Attributes

- TileType tile_type
- TileResource tile_resource
- bool show_node

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

  The current frame of this object.
- double position_x

  The x position of the tile.
- double position_y

  The y position of the tile.
- double major_radius

  The radius of the smallest bounding circle.
- double minor_radius

  The radius of the largest inscribed circle.
- sf::CircleShape node_sprite

  A circle shape to mark the tile node.
- sf::ConvexShape tile_sprite

  A convex shape which represents the tile.

Private Member Functions

- void __setUpNodeSprite (void)

  Helper method to set up node sprite.
- void __setUpTileSprite (void)

  Helper method to set up tile sprite.

Private Attributes

- AssetsManager * assets_manager_ptr

  A pointer to the assets manager.
- InputsHandler * inputs_handler_ptr

  A pointer to the inputs handler.
- MessagesHandler * messages_handler_ptr

  A pointer to the messages handler.

3.3.1 Detailed Description

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

3.3.2 Constructor & Destructor Documentation

3.3.2.1 HexTile()

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

Constructor for the HexTile class.

Ref: Wikipedia [2023]

Parameters

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

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

3.3.2.2 ~HexTile()

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

Destructor for the HexTile class.

```
345 {
346     std::cout « "HexTile at " « this « " destroyed" « std::endl;
347 
348     return;
349 }   /* ~HexTile() */
```

3.3.3 Member Function Documentation

3.3.3.1 __setUpNodeSprite()

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

Helper method to set up node sprite.

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

3.3.3.2 __setUpTileSprite()

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

Helper method to set up tile sprite.

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

3.3.3.3 draw()

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

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

Parameters

window_ptr	A pointer to the render window.

```
319 {
320     //  1. draw hex
321     window_ptr->draw(this->tile_sprite);
322 
323     //  2. draw node
324     if (this->show_node) {
325         window_ptr->draw(this->node_sprite);
326     }
327 
328     this->frame++;
329     return;
330 }   /* draw() */
```

3.3.3.4 process()

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

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

```
297 {
298     //...
299 
300     return;
301 }   /* process() */
```

3.3.3.5 setTileResource() [1/2]

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

3.3.3.6 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.

```
278 {
279     this->tile_resource = tile_resource;
280 
281     return;
282 }   /* setTileType() */
```

3.3.3.7 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].

```
227 {
228     //  1. check input
229     if (input_value < 0 or input_value > 1) {
230         std::string error_str = "ERROR  HexTile::setTileType()  given input value is ";
231         error_str += "not in the closed interval [0, 1]";
232 
233         #ifdef _WIN32
234             std::cout « error_str « std::endl;
235         #endif  /* _WIN32 */
236 
237         throw std::runtime_error(error_str);
238     }
239 
240     //  2. convert input value to tile type
241     TileType tile_type;
242 
243     std::cout « input_value « std::endl;
244     if (input_value <= tile_type_cumulative_probabilities[0]) {
245         tile_type = TileType :: LAKE;
246     }
247     else if (input_value <= tile_type_cumulative_probabilities[1]) {
248         tile_type = TileType :: PLAINS;
249     }
250     else if (input_value <= tile_type_cumulative_probabilities[2]) {
251         tile_type = TileType :: FOREST;
252     }
253     else {
254         tile_type = TileType :: MOUNTAINS;
255     }
256 
257     //  3. call alternate method
258     this->setTileType(tile_type);
259 
260     return;
261 }   /* setTileType(double) */
```

3.3.3.8 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.

```
168 {
169     this->tile_type = tile_type;
170 
171     switch (this->tile_type) {
172         case (TileType :: FOREST): {
173             this->tile_sprite.setFillColor(FOREST_GREEN);
174 
175             break;
176         }
177 
```

```
178         case (TileType :: LAKE): {
179             this->tile_sprite.setFillColor(LAKE_BLUE);
180 
181             break;
182         }
183 
184         case (TileType :: MOUNTAINS): {
185             this->tile_sprite.setFillColor(MOUNTAINS_GREY);
186 
187             break;
188         }
189 
190         case (TileType :: OCEAN): {
191             this->tile_sprite.setFillColor(OCEAN_BLUE);
192 
193             break;
194         }
195 
196         case (TileType :: PLAINS): {
197             this->tile_sprite.setFillColor(PLAINS_YELLOW);
198 
199             break;
200         }
201 
202         default: {
203             // do nothing!
204 
205             break;
206         }
207     }
208 
209     return;
210 }   /* setTileType(TileType) */
```

3.3.4 Member Data Documentation

3.3.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.3.4.2 frame

```
int HexTile::frame
```

The current frame of this object.

3.3.4.3 inputs_handler_ptr

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

A pointer to the inputs handler.

3.3.4.4 major_radius

```
double HexTile::major_radius
```

The radius of the smallest bounding circle.

3.3.4.5 messages_handler_ptr

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

A pointer to the messages handler.

3.3.4.6 minor_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

3.3.4.7 node_sprite

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

A circle shape to mark the tile node.

3.3.4.8 position_x

```
double HexTile::position_x
```

The x position of the tile.

3.3.4.9 position_y

```
double HexTile::position_y
```

The y position of the tile.

3.3.4.10 show_node

```
bool HexTile::show_node
```

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

3.3.4.11 tile_resource

```
TileResource HexTile::tile_resource
```

3.3.4.12 tile_sprite

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

A convex shape which represents the tile.

3.3.4.13 tile_type

```
TileType HexTile::tile_type
```

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

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

3.4 InputsHandler Class Reference

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

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

Public Member Functions

- InputsHandler (void)

  Constructor for the InputsHandler class.
- void process (sf::Event *)
- void printKeysPressed (void)

  Method to print out which keys are currently pressed.
- void reset (void)

  Method to reset InputsHandler. To be called once per frame (at end of frame!).
- ~InputsHandler (void)

  Destructor for the InputsHandler class.

Public Attributes

- std::vector< bool > key_pressed_once_vec

  A vector (bool) which indicates which keys have been pressed once. Useful for discrete inputs.
- std::vector< bool > key_press_vec

  A vector <bool> which indicates which keys are currently pressed. Useful for smooth movement.
- std::map< sf::Keyboard::Key, std::string > key_code_map

  A map from key codes to corresponding string representations.

Private Member Functions

- void __constructKeyCodeMap (void)

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

3.4.1 Detailed Description

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

3.4.2 Constructor & Destructor Documentation

3.4.2.1 InputsHandler()

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

Constructor for the InputsHandler class.

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

3.4.2.2 ~InputsHandler()

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

Destructor for the InputsHandler class.

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

3.4.3 Member Function Documentation

3.4.3.1 __constructKeyCodeMap()

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

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

```
35 {
36     //  1. unknown keys
37     this->key_code_map.insert(
38         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Unknown, "Unknown")
39     );
40
41
42     //  2. alpha keys
43     this->key_code_map.insert(
44         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::A, "A")
45     );
46     this->key_code_map.insert(
47         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::B, "B")
48     );
49     this->key_code_map.insert(
50         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::C, "C")
51     );
52     this->key_code_map.insert(
53         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::D, "D")
54     );
55     this->key_code_map.insert(
56         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::E, "E")
57     );
58     this->key_code_map.insert(
59         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F, "F")
60     );
61     this->key_code_map.insert(
62         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::G, "G")
63     );
64     this->key_code_map.insert(
65         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::H, "H")
66     );
67     this->key_code_map.insert(
68         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::I, "I")
69     );
70     this->key_code_map.insert(
71         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::J, "J")
72     );
73     this->key_code_map.insert(
74         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::K, "K")
75     );
76     this->key_code_map.insert(
77         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::L, "L")
78     );
79     this->key_code_map.insert(
80         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::M, "M")
81     );
82     this->key_code_map.insert(
83         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::N, "N")
84     );
85     this->key_code_map.insert(
86         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::O, "O")
87     );
88     this->key_code_map.insert(
89         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::P, "P")
90     );
91     this->key_code_map.insert(
92         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Q, "Q")
93     );
94     this->key_code_map.insert(
95         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::R, "R")
96     );
97     this->key_code_map.insert(
98         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::S, "S")
99     );
100     this->key_code_map.insert(
101         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::T, "T")
102     );
103     this->key_code_map.insert(
```

```
104         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::U, "U")
105     );
106     this->key_code_map.insert(
107         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::V, "V")
108     );
109     this->key_code_map.insert(
110         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::W, "W")
111     );
112     this->key_code_map.insert(
113         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::X, "X")
114     );
115     this->key_code_map.insert(
116         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Y, "Y")
117     );
118     this->key_code_map.insert(
119         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Z, "Z")
120     );
121 
122 
123     //  3. numeric keys
124     this->key_code_map.insert(
125         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num0, "0")
126     );
127     this->key_code_map.insert(
128         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num1, "1")
129     );
130     this->key_code_map.insert(
131         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num2, "2")
132     );
133     this->key_code_map.insert(
134         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num3, "3")
135     );
136     this->key_code_map.insert(
137         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num4, "4")
138     );
139     this->key_code_map.insert(
140         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num5, "5")
141     );
142     this->key_code_map.insert(
143         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num6, "6")
144     );
145     this->key_code_map.insert(
146         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num7, "7")
147     );
148     this->key_code_map.insert(
149         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num8, "8")
150     );
151     this->key_code_map.insert(
152         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num9, "9")
153     );
154     this->key_code_map.insert(
155         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad0, "0")
156     );
157     this->key_code_map.insert(
158         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad1, "1")
159     );
160     this->key_code_map.insert(
161         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad2, "2")
162     );
163     this->key_code_map.insert(
164         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad3, "3")
165     );
166     this->key_code_map.insert(
167         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad4, "4")
168     );
169     this->key_code_map.insert(
170         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad5, "5")
171     );
172     this->key_code_map.insert(
173         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad6, "6")
174     );
175     this->key_code_map.insert(
176         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad7, "7")
177     );
178     this->key_code_map.insert(
179         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad8, "8")
180     );
181     this->key_code_map.insert(
182         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad9, "9")
183     );
184 
185 
186     //  4. direction keys
187     this->key_code_map.insert(
188         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Left, "Left")
189     );
190     this->key_code_map.insert(
```

```
191         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Right, "Right")
192     );
193     this->key_code_map.insert(
194         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Up, "Up")
195     );
196     this->key_code_map.insert(
197         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Down, "Down")
198     );
199 
200 
201     //  5. function keys
202     this->key_code_map.insert(
203         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F1, "F1")
204     );
205     this->key_code_map.insert(
206         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F2, "F2")
207     );
208     this->key_code_map.insert(
209         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F3, "F3")
210     );
211     this->key_code_map.insert(
212         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F4, "F4")
213     );
214     this->key_code_map.insert(
215         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F5, "F5")
216     );
217     this->key_code_map.insert(
218         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F6, "F6")
219     );
220     this->key_code_map.insert(
221         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F7, "F7")
222     );
223     this->key_code_map.insert(
224         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F8, "F8")
225     );
226     this->key_code_map.insert(
227         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F9, "F9")
228     );
229     this->key_code_map.insert(
230         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F10, "F10")
231     );
232     this->key_code_map.insert(
233         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F11, "F11")
234     );
235     this->key_code_map.insert(
236         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F12, "F12")
237     );
238     this->key_code_map.insert(
239         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F13, "F13")
240     );
241     this->key_code_map.insert(
242         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F14, "F14")
243     );
244     this->key_code_map.insert(
245         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F15, "F15")
246     );
247 
248 
249     //  6. other keys
250     this->key_code_map.insert(
251         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Escape, "Escape")
252     );
253     this->key_code_map.insert(
254         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LControl, "LCtrl")
255     );
256     this->key_code_map.insert(
257         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LShift, "LShift")
258     );
259     this->key_code_map.insert(
260         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LAlt, "LAlt")
261     );
262     this->key_code_map.insert(
263         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LSystem, "LSystem")
264     );
265     this->key_code_map.insert(
266         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RControl, "RCtrl")
267     );
268     this->key_code_map.insert(
269         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RShift, "RShift")
270     );
271     this->key_code_map.insert(
272         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RAlt, "RAlt")
273     );
274     this->key_code_map.insert(
275         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RSystem, "RSystem")
276     );
277     this->key_code_map.insert(
```

```
278         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Menu, "Menu")
279     );
280     this->key_code_map.insert(
281         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LBracket, "LBracket")
282     );
283     this->key_code_map.insert(
284         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RBracket, "RBracket")
285     );
286     this->key_code_map.insert(
287         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Semicolon, "Semicolon")
288     );
289     this->key_code_map.insert(
290         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Comma, "Comma")
291     );
292     this->key_code_map.insert(
293         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Period, "Period")
294     );
295     this->key_code_map.insert(
296         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Quote, "Quote")
297     );
298     this->key_code_map.insert(
299         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Slash, "Slash")
300     );
301     this->key_code_map.insert(
302         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backslash, "Backslash")
303     );
304     this->key_code_map.insert(
305         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tilde, "Tilde")
306     );
307     this->key_code_map.insert(
308         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Equal, "Equal")
309     );
310     this->key_code_map.insert(
311         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Hyphen, "Hyphen")
312     );
313     this->key_code_map.insert(
314         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Space, "Space")
315     );
316     this->key_code_map.insert(
317         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Enter, "Enter")
318     );
319     this->key_code_map.insert(
320         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backspace, "Backspace")
321     );
322     this->key_code_map.insert(
323         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tab, "Tab")
324     );
325     this->key_code_map.insert(
326         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageUp, "PageUp")
327     );
328     this->key_code_map.insert(
329         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageDown, "PageDown")
330     );
331     this->key_code_map.insert(
332         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::End, "End")
333     );
334     this->key_code_map.insert(
335         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Home, "Home")
336     );
337     this->key_code_map.insert(
338         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Insert, "Insert")
339     );
340     this->key_code_map.insert(
341         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Delete, "Delete")
342     );
343     this->key_code_map.insert(
344         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Add, "Add")
345     );
346     this->key_code_map.insert(
347         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Subtract, "Subtract")
348     );
349     this->key_code_map.insert(
350         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Multiply, "Multiply")
351     );
352     this->key_code_map.insert(
353         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Divide, "Divide")
354     );
355     this->key_code_map.insert(
356         std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Pause, "Pause")
357     );
358
359     return;
360 }   /* __constructKeyCodeMap() */
```

3.4.3.2 printKeysPressed()

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

Method to print out which keys are currently pressed.

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

3.4.3.3 process()

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

3.4.3.4 reset()

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

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

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

3.4.4 Member Data Documentation

3.4.4.1 key_code_map

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

A map from key codes to corresponding string representations.

3.4.4.2 key_press_vec

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

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

3.4.4.3 key_pressed_once_vec

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

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

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

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

3.5 MessagesHandler Class Reference

A class which handles message traffic between game objects.

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

Public Member Functions

- MessagesHandler (void)

  Constructor for the MessagesHandler class.
- void process (void)

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

  Destructor for the MessagesHandler class.

3.5.1 Detailed Description

A class which handles message traffic between game objects.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 MessagesHandler()

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

Constructor for the MessagesHandler class.

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

3.5.2.2 ~MessagesHandler()

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

Destructor for the MessagesHandler class.

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

3.5.3 Member Function Documentation

3.5.3.1 process()

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

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

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

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

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

Chapter 4

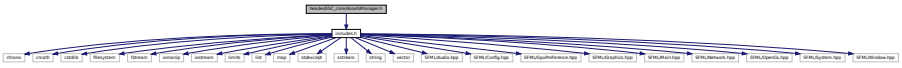
File Documentation

4.1 header/ESC_core/AssetsManager.h File Reference

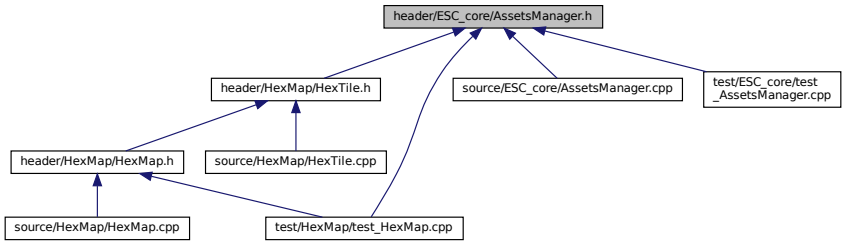
Header file for the AssetsManager class.

```
#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.

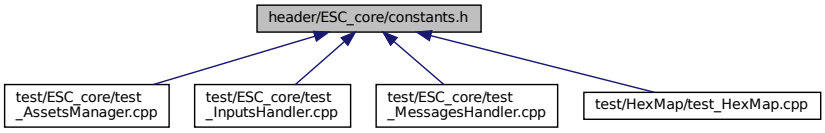
4.1.1 Detailed Description

Header file for the AssetsManager class.

4.2 header/ESC_core/constants.h File Reference

Header file for various constants.

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



Variables

- const int FRAMES_PER_SECOND = 60

  Target frames per second.
- const double SECONDS_PER_FRAME = 1.0 / 60

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

4.2.1 Detailed Description

Header file for various constants.

4.2.2 Variable Documentation

4.2.2.1 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

4.2.2.2 SECONDS_PER_FRAME

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

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

4.3 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]

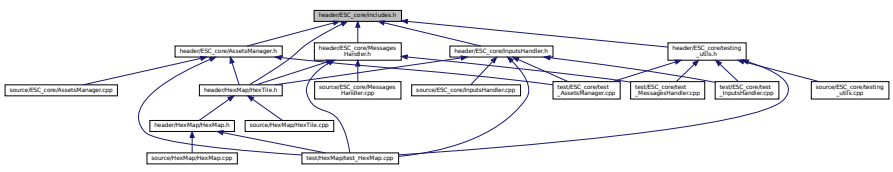
4.4 header/ESC_core/includes.h File Reference

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

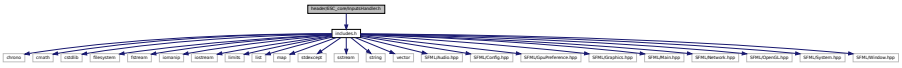
Header file for various includes.

Ref: Gomila [2023]

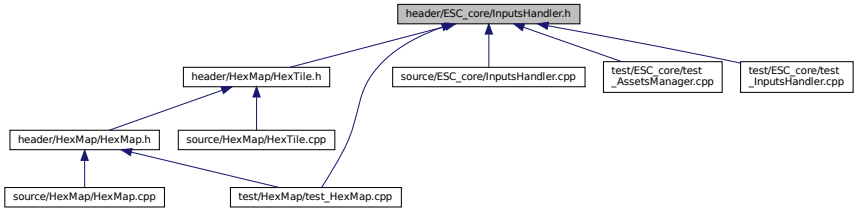
4.5 header/ESC_core/InputsHandler.h File Reference

Header file for the InputsHandler class.

`#include "includes.h"`
Include dependency graph for InputsHandler.h:



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



Classes

- class InputsHandler

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

4.5.1 Detailed Description

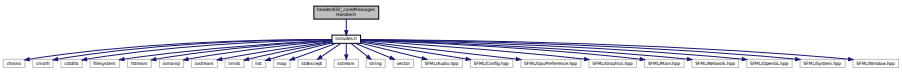
Header file for the InputsHandler class.

4.6 header/ESC_core/MessagesHandler.h File Reference

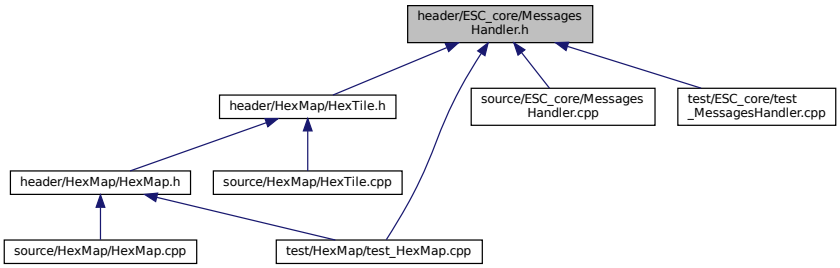
Header file for the MessagesHandler class.

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

Include dependency graph for MessagesHandler.h:



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



Classes

- class MessagesHandler

  A class which handles message traffic between game objects.

4.6.1 Detailed Description

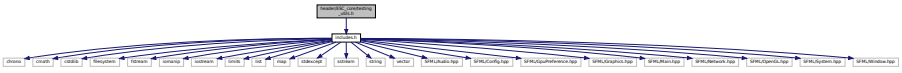
Header file for the MessagesHandler class.

4.7 header/ESC_core/testing_utils.h File Reference

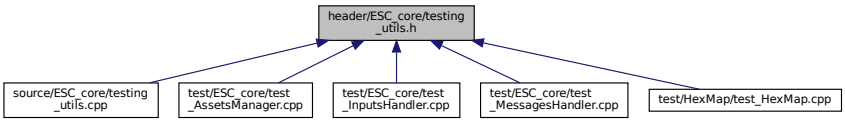
Header file for various testing utilities.

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

Include dependency graph for testing_utils.h:



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



Functions

- void printGreen (std::string)

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

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

  A function that sends red text to std::cout.
- void testFloatEquals (double, double, std::string, int)

  Tests for the equality of two floating point numbers x and y (to within FLOAT_TOLERANCE).
- void testGreaterThan (double, double, std::string, int)

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

  Tests if x >= 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.

Variables

- const double FLOAT_TOLERANCE = 1e-6

  Tolerance for floating point equality tests.

4.7.1 Detailed Description

Header file for various testing utilities.

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

4.7.2 Function Documentation

4.7.2.1 expectedErrorNotDetected()

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

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

Parameters

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() */
```

4.7.2.2 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() */
```

4.7.2.3 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() */
```

4.7.2.4 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() */
```

4.7.2.5 testFloatEquals()

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

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

Parameters

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() */
```

4.7.2.6 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() */
```

4.7.2.7 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() */
```

4.7.2.8 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() */
```

4.7.2.9 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() */
```

4.7.2.10 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() */
```

4.7.3 Variable Documentation

4.7.3.1 FLOAT_TOLERANCE

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

Tolerance for floating point equality tests.

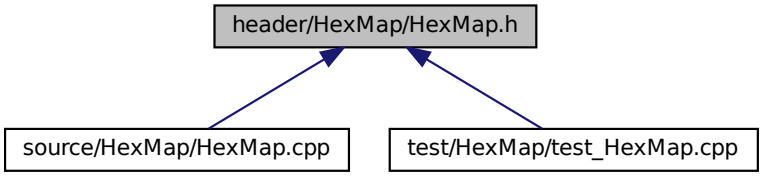
4.8 header/HexMap/HexMap.h File Reference

Header file for the HexMap class.

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

Include dependency graph for HexMap.h:

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



Classes

- class HexMap

  A class which defines a hex map of hex tiles.

Variables

- const double AMPLITUDE_BASE = 1

  A base amplitude value to use in procedural generation (based on random cosine series).
- const double WAVE_NUMBER_BASE = (2 ∗ M_PI) / 32

  A base wave number to use in procedural generation(based on random cosine series).
- const double PHASE_BASE = 64

  A base phase to use in procedural generation(based on random cosine series).

4.8.1 Detailed Description

Header file for the HexMap class.

4.8.2 Variable Documentation

4.8.2.1 AMPLITUDE_BASE

```
const double AMPLITUDE_BASE = 1
```

A base amplitude value to use in procedural generation (based on random cosine series).

4.8.2.2 PHASE_BASE

```
const double PHASE_BASE = 64
```

A base phase to use in procedural generation(based on random cosine series).

4.8.2.3 WAVE_NUMBER_BASE

```
const double WAVE_NUMBER_BASE = (2 * M_PI) / 32
```

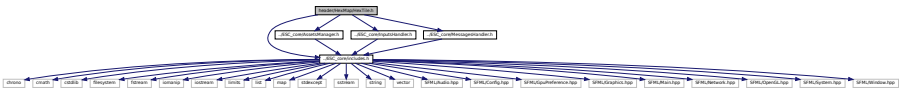
A base wave number to use in procedural generation(based on random cosine series).

4.9 header/HexMap/HexTile.h File Reference

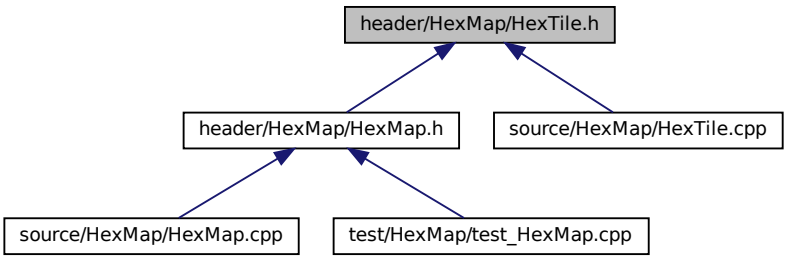
Header file for the HexTile class.

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

Include dependency graph for HexTile.h:



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



Classes

- class HexTile

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

Enumerations

- enum TileType {
  FOREST , LAKE , MOUNTAINS , OCEAN ,
  PLAINS , N_TILE_TYPES }

  An enumeration of the different tile types.
- enum TileResource {
  POOR , BELOW_AVERAGE , AVERAGE , ABOVE_AVERAGE ,
  GOOD , N_TILE_RESOURCES }

  An enumeration of the different tile resource values.

Functions

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

  The base colour of a forest tile.
- const sf::Color LAKE_BLUE (0, 102, 204)

  The base colour of a lake (water) tile.
- const sf::Color MOUNTAINS_GREY (97, 110, 113)

  The base colour of a mountains tile.
- const sf::Color OCEAN_BLUE (0, 51, 102)

  The base colour of an ocean (water) tile.
- const sf::Color PLAINS_YELLOW (245, 222, 133)

  The base colour of a plains tile.

Variables

- const std::vector< double > tile_type_cumulative_probabilities

4.9.1 Detailed Description

Header file for the HexTile class.

4.9.2 Enumeration Type Documentation

4.9.2.1 TileResource

```
enum TileResource
```

An enumeration of the different tile resource values.

Enumerator

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

```
63                  {
64     POOR,
65     BELOW_AVERAGE,
66     AVERAGE,
67     ABOVE_AVERAGE,
68     GOOD,
69     N_TILE_RESOURCES
70 };
```

4.9.2.2 TileType

```
enum TileType
```

An enumeration of the different tile types.

Enumerator

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

```
34              {
35     FOREST,
36     LAKE,
37     MOUNTAINS,
```

```
38     OCEAN,
39     PLAINS,
40     N_TILE_TYPES
41 };
```

4.9.3 Function Documentation

4.9.3.1 FOREST_GREEN()

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

The base colour of a forest tile.

4.9.3.2 LAKE_BLUE()

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

The base colour of a lake (water) tile.

4.9.3.3 MOUNTAINS_GREY()

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

The base colour of a mountains tile.

4.9.3.4 OCEAN_BLUE()

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

The base colour of an ocean (water) tile.

4.9.3.5 PLAINS_YELLOW()

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

The base colour of a plains tile.

4.9.4 Variable Documentation

4.9.4.1 tile_type_cumulative_probabilities

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

Initial value:

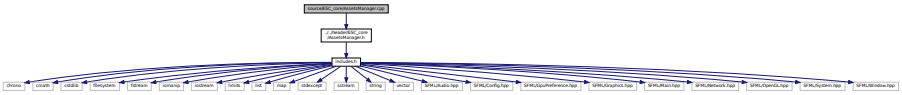
```
= {
    0.30,
    0.50,
    0.75,
    1.00
}
```

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

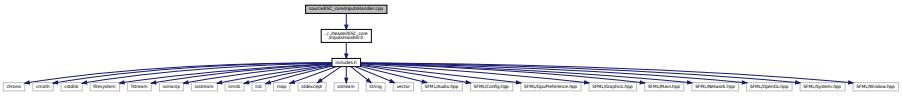
Implementation file for the AssetsManager class.

A class which manages visual and sound assets.

4.11 source/ESC_core/InputsHandler.cpp File Reference

Implementation file for the InputsHandler class.

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



4.11.1 Detailed Description

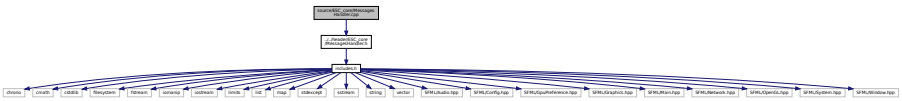
Implementation file for the InputsHandler class.

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

4.12 source/ESC_core/MessagesHandler.cpp File Reference

Implementation file for the MessagesHandler class.

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



4.12.1 Detailed Description

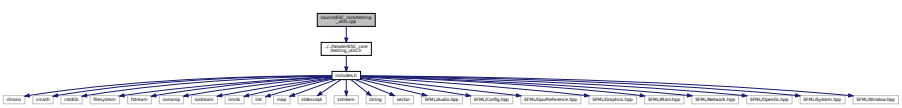
Implementation file for the MessagesHandler class.

A class which handles message traffic between game objects.

4.13 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.

4.13.1 Detailed Description

Implementation file for various testing utilities.

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

4.13.2 Function Documentation

4.13.2.1 expectedErrorNotDetected()

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

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

Parameters

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() */
```

4.13.2.2 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() */
```

4.13.2.3 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() */
```

4.13.2.4 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() */
```

4.13.2.5 testFloatEquals()

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

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

Parameters

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() */
```

4.13.2.6 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() */
```

4.13.2.7 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() */
```

4.13.2.8 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() */
```

4.13.2.9 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() */
```

4.13.2.10 testTruth()

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

Tests if the given statement is true.

Parameters

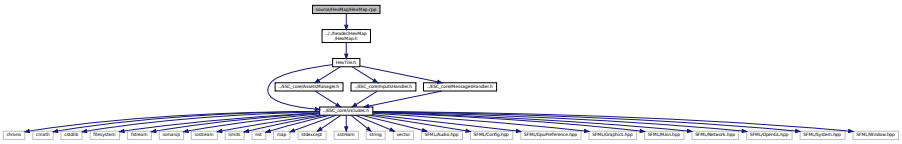
statement	The statement whose truth is to be tested ("1 == 0", for example).
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
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() */
```

4.14 source/HexMap/HexMap.cpp File Reference

Implementation file for the HexMap class.

```
#include "../../header/HexMap/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/HexMap/HexTile.cpp File Reference

Implementation file for the HexTile class.

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

4.15.1 Detailed Description

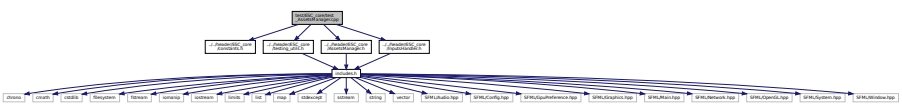
Implementation file for the HexTile class.

A class which defines a tile of a hex map.

4.16 test/ESC_core/test_AssetsManager.cpp File Reference

Suite of tests for the AssetsManager class.

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



Functions

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

4.16.1 Detailed Description

Suite of tests for the AssetsManager class.

A suite of tests for the AssetsManager class.

4.16.2 Function Documentation

4.16.2.1 main()

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

```
37 {
38     #ifdef _WIN32
39         activateVirtualTerminal();
40     #endif  /* _WIN32 */
41 
42     printGold("\tTesting AssetsManager");
43     std::cout « std::endl;
44 
45     srand(time(NULL));
46     int n_dots = 8;
47 
48 
49     try {
50         //  1. construct
51         InputsHandler inputs_handler;
52         AssetsManager assets_manager;
53 
54 
55         //  2. load/open some test assets
56         assets_manager.loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
57         assets_manager.loadTexture(
58             "assets/ESC_brand/ESC_key_98x81.png",
59             "ESC_key_98x81"
60         );
61         assets_manager.loadSound("assets/ESC_brand/key_press.ogg", "key_press");
62         assets_manager.loadTrack(
63             "assets/audio/tracks/AlexanderBlu_BackgroundElectronicModernMusic.ogg",
64             "AlexanderBlu_BackgroundElectronicModernMusic"
65         );
66 
67 
68         //  3. test game loop
69         sf::Clock clock;
70         sf::Event event;
71         sf::RenderWindow window(sf::VideoMode(800, 600), "Testing AssetsManager");
72 
73         double screen_width = window.getSize().x;
74         double screen_height = window.getSize().y;
75 
76         testFloatEquals(
77             screen_width,
78             800,
79             __FILE__,
80             __LINE__
81         );
82 
83         testFloatEquals(
84             screen_height,
85             600,
86             __FILE__,
87             __LINE__
```

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

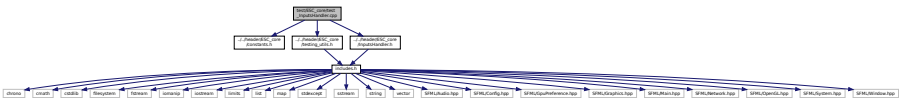
```
175
176                 window.draw(ESC_key);
177                 window.draw(click_text);
178
179                 window.display();
180
181                 alpha -= 8;
182                 if (alpha < 0) {
183                     alpha = 0;
184                 }
185
186                 click_text.setFillColor(sf::Color(255, 255, 255, alpha));
187
188                 std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
189                 frame++;
190             }
191         }
192     }
193
194
195     catch (...) {
196         //...
197
198         printGold(" ");
199         for (int i = 0; i < n_dots; i++) {
200             printGold(".");
201         }
202         printGold(" ");
203         printRed("FAIL");
204         std::cout « std::endl;
205         throw;
206     }
207
208
209     //...
210
211     printGold(" ");
212     for (int i = 0; i < n_dots; i++) {
213         printGold(".");
214     }
215     printGold(" ");
216     printGreen("PASS");
217     std::cout « std::endl;
218
219     return 0;
220 }   /* main() */
```

4.17 test/ESC_core/test_InputsHandler.cpp File Reference

Suite of tests for the InputsHandler class.

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

Include dependency graph for test_InputsHandler.cpp:



Functions

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

4.17.1 Detailed Description

Suite of tests for the InputsHandler class.

A suite of tests for the InputsHandler class.

4.17.2 Function Documentation

4.17.2.1 main()

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

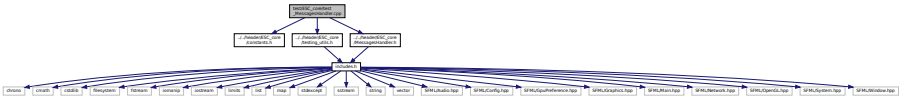
```
107                     inputs_handler.process(&event);
108
109                     if (event.type == sf::Event::Closed) {
110                         window.close();
111                     }
112                 }
113
114                 window.clear();
115                 window.display();
116
117                 inputs_handler.printKeysPressed();
118                 if (inputs_handler.key_pressed_once_vec[sf::Keyboard::Enter]) {
119                     std::cout « "Enter" « std::endl;
120                 }
121
122                 inputs_handler.reset();
123
124                 std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
125                 frame++;
126             }
127         }
128     }
129
130
131     catch (...) {
132         //...
133
134         printGold(" ");
135         for (int i = 0; i < n_dots; i++) {
136             printGold(".");
137         }
138         printGold(" ");
139         printRed("FAIL");
140         std::cout « std::endl;
141         throw;
142     }
143
144
145     //...
146
147     printGold(" ");
148     for (int i = 0; i < n_dots; i++) {
149         printGold(".");
150     }
151     printGold(" ");
152     printGreen("PASS");
153     std::cout « std::endl;
154
155     return 0;
156 }   /* main() */
```

4.18 test/ESC_core/test_MessagesHandler.cpp File Reference

Suite of tests for the MessagesHandler class.

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

Include dependency graph for test_MessagesHandler.cpp:



Functions

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

4.18.1 Detailed Description

Suite of tests for the MessagesHandler class.

A suite of tests for the MessagesHandler class.

4.18.2 Function Documentation

4.18.2.1 main()

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

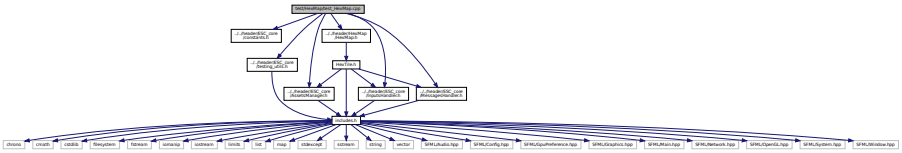
```
95
96                  std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
97                  frame++;
98              }
99          }
100     }
101
102
103     catch (...) {
104         //...
105
106         printGold(" ");
107         for (int i = 0; i < n_dots; i++) {
108             printGold(".");
109         }
110         printGold(" ");
111         printRed("FAIL");
112         std::cout « std::endl;
113         throw;
114     }
115
116
117     //...
118
119     printGold(" ");
120     for (int i = 0; i < n_dots; i++) {
121         printGold(".");
122     }
123     printGold(" ");
124     printGreen("PASS");
125     std::cout « std::endl;
126
127     return 0;
128 }   /* main() */
```

4.19 test/HexMap/test_HexMap.cpp File Reference

Suite of tests for the HexMap class.

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

Include dependency graph for test_HexMap.cpp:



Functions

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

4.19.1 Detailed Description

Suite of tests for the HexMap class.

A suite of tests for the HexMap class.

4.19.2 Function Documentation

4.19.2.1 main()

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

```
111
112                 inputs_handler.reset();
113
114                 std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
115                 frame++;
116             }
117         }
118     }
119
120
121     catch (...) {
122         //...
123
124         printGold(" ");
125         for (int i = 0; i < n_dots; i++) {
126             printGold(".");
127         }
128         printGold(" ");
129         printRed("FAIL");
130         std::cout « std::endl;
131         throw;
132     }
133
134
135     //...
136
137     printGold(" ");
138     for (int i = 0; i < n_dots; i++) {
139         printGold(".");
140     }
141     printGold(" ");
142     printGreen("PASS");
143     std::cout « std::endl;
144
145     return 0;
146 }   /* main() */
```

Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL https://www.sfml-dev.org/. 50

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL https://www.doxygen.nl. 49

Wikipedia. Hexagon, 2023. URL https://en.wikipedia.org/wiki/Hexagon. 32

Index