Road To Zero - The Microgrid Management Game

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 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.

4.1.1 Detailed Description

A class which manages visual and sound assets.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AssetsManager()

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

Constructor for the AssetsManager class.

```
{
    //...

    std::cout « "AssetsManager constructed at " « this « std::endl;

    return;
}   /* AssetsManager() */
```

4.1.2.2 ∼AssetsManager()

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

Destructor for the AssetsManager class.

```
{
    this->clear();

    std::cout « "AssetsManager at " « this « " destroyed" « std::endl;

    return;
}   /* ~AssetsManager() */
```

4.1.3 Member Function Documentation

4.1.3.1 __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).

```
{
    //  1. check key, throw error if already in use
    if (this->soundbuffer_map.count(sound_key) > 0) {
        std::string error_str = "ERROR  AssetsManager::__loadSoundBuffer()  sound key ";
        error_str += sound_key;
        error_str += " is already in use";

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }


    //  2. load from file, throw error on fail
    sf::SoundBuffer* soundbuffer_ptr = new sf::SoundBuffer();

    if (not soundbuffer_ptr->loadFromFile(path_2_sound)) {
         std::string error_str = "ERROR  AssetsManager::__loadSoundBuffer()  could not load ";
         error_str += "soundbuffer at ";
         error_str += path_2_sound;

         this->clear();

         #ifdef _WIN32
             std::cout « error_str « std::endl;
         #endif  /* _WIN32 */

         throw std::runtime_error(error_str);
     }


    //  3. insert into soundbuffer map
    this->soundbuffer_map.insert(
        std::pair<std::string, sf::SoundBuffer*>(sound_key, soundbuffer_ptr)
    );

    std::cout « "SoundBuffer " « sound_key « " inserted into soundbuffer map" «
        std::endl;

    return;
}   /* __loadSoundBuffer() */
```

4.1.3.2 clear()

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

Method to clear all loaded assets.

```
{
    //  1. clear fonts
    std::map<std::string, sf::Font*>::iterator font_iter;
    for (
        font_iter = this->font_map.begin();
        font_iter != this->font_map.end();
        font_iter++
    ) {
        delete font_iter->second;

        std::cout « "Font " « font_iter->first « " deleted from font map" «
            std::endl;
    }
    this->font_map.clear();


    //  2. clear textures
    std::map<std::string, sf::Texture*>::iterator texture_iter;
    for (
        texture_iter = this->texture_map.begin();
        texture_iter != this->texture_map.end();
        texture_iter++
    ) {
        delete texture_iter->second;

        std::cout « "Texture " « texture_iter->first « " deleted from texture map" «
            std::endl;
    }
    this->texture_map.clear();


    //  3. clear sound buffers
    std::map<std::string, sf::SoundBuffer*>::iterator soundbuffer_iter;
    for (
        soundbuffer_iter = this->soundbuffer_map.begin();
        soundbuffer_iter != this->soundbuffer_map.end();
        soundbuffer_iter++
    ) {
        delete soundbuffer_iter->second;

        std::cout « "SoundBuffer " « soundbuffer_iter->first «
            " deleted from soundbuffer map" « std::endl;
    }
    this->soundbuffer_map.clear();


    //  4. clear sounds
    std::map<std::string, sf::Sound*>::iterator sound_iter;
    for (
        sound_iter = this->sound_map.begin();
        sound_iter != this->sound_map.end();
        sound_iter++
    ) {
        sound_iter->second->stop();
        delete sound_iter->second;

        std::cout « "Sound " « sound_iter->first « " deleted from sound map" «
            std::endl;
    }
    this->sound_map.clear();


    //  5. clear tracks
    std::map<std::string, sf::Music*>::iterator track_iter;
    for (
        track_iter = this->track_map.begin();
        track_iter != this->track_map.end();
        track_iter++
    ) {
        track_iter->second->stop();
        delete track_iter->second;

        std::cout « "Track " « track_iter->first « " deleted from track map" «
            std::endl;
    }
    this->track_map.clear();

    return;
}   /* clear() */
```

4.1.3.3 getCurrentTrackKey()

```
std::string AssetsManager::getCurrentTrackKey (
              void  )
```

Method to get track key for current track.

Returns

The track key for the current track.

```
{
    return this->current_track->first;
}   /* getCurrentTrackKey() */
```

4.1.3.4 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.

```
{
    //  1. check key, throw error if not found
    if (this->font_map.count(font_key) <= 0) {
        std::string error_str = "ERROR  AssetsManager::getFont()  font key ";
        error_str += font_key;
        error_str += " is not contained in font map";

        this->clear();

        #ifdef _WIN32
```

```
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    return this->font_map[font_key];
}   /* getFont() */
```

4.1.3.5 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.

```
{
    //  1. check key, throw error if not found
    if (this->sound_map.count(sound_key) <= 0) {
        std::string error_str = "ERROR  AssetsManager::getSound()  sound key ";
        error_str += sound_key;
        error_str += " is not contained in sound map";

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    return this->sound_map[sound_key];
}   /* getSound() */
```

4.1.3.6 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.

```
{
    //  1. check key, throw error if not found
    if (this->soundbuffer_map.count(sound_key) <= 0) {
        std::string error_str = "ERROR  AssetsManager::getSoundBuffer()  sound key ";
        error_str += sound_key;
        error_str += " is not contained in soundbuffer map";

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    return this->soundbuffer_map[sound_key];
}   /* getSoundBuffer() */
```

4.1.3.7 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.

```
{
    //  1. check key, throw error if not found
    if (this->texture_map.count(texture_key) <= 0) {
        std::string error_str = "ERROR  AssetsManager::getTexture()  texture key ";
        error_str += texture_key;
        error_str += " is not contained in texture map";

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    return this->texture_map[texture_key];
}   /* getTexture() */
```

4.1.3.8 getTrackStatus()

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

Method to get the status of the current track.

Returns

The status of the current track.

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

4.1.3.9 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).

```
{
    //  1. check key, throw error if already in use
    if (this->font_map.count(font_key) > 0) {
        std::string error_str = "ERROR  AssetsManager::loadFont()  font key ";
        error_str += font_key;
        error_str += " is already in use";

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }


    //  2. load from file, throw error on fail
    sf::Font* font_ptr = new sf::Font();

    if (not font_ptr->loadFromFile(path_2_font)) {
        std::string error_str = "ERROR  AssetsManager::loadFont()  could not load ";
        error_str += "font at ";
        error_str += path_2_font;

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }


    //  3. insert into font map
    this->font_map.insert(std::pair<std::string, sf::Font*>(font_key, font_ptr));

    std::cout « "Font " « font_key « " inserted into font map" « std::endl;

    return;
}   /* loadFont() */
```

4.1.3.10 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).

```
{
    //  1. create an associated sf::SoundBuffer
    this->__loadSoundBuffer(path_2_sound, sound_key);

    //  2. associate sf::Sound with sf::SoundBuffer
    sf::Sound* sound_ptr = new sf::Sound();
    sound_ptr->setBuffer(*(this->soundbuffer_map[sound_key]));

    //  3. insert into sound map
    this->sound_map.insert(std::pair<std::string, sf::Sound*>(sound_key, sound_ptr));

    std::cout « "Sound " « sound_key « " inserted into sound map" « std::endl;

    return;
}   /* loadSound() */
```

4.1.3.11 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).

```
{
    //  1. check key, throw error if already in use
    if (this->texture_map.count(texture_key) > 0) {
        std::string error_str = "ERROR  AssetsManager::loadTexture()  texture key ";
        error_str += texture_key;
        error_str += " is already in use";

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }


    //  2. load from file, throw error on fail
    sf::Texture* texture_ptr = new sf::Texture();

    if (not texture_ptr->loadFromFile(path_2_texture)) {
        std::string error_str = "ERROR  AssetsManager::loadTexture()  could not load ";
        error_str += "texture at ";
        error_str += path_2_texture;

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }


    //  3. insert into texture map
    this->texture_map.insert(
        std::pair<std::string, sf::Texture*>(texture_key, texture_ptr)
    );

    std::cout « "Texture " « texture_key « " inserted into texture map" « std::endl;

    return;
}   /* loadTexture() */
```

4.1.3.12 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).

```
{
    //  1. check key, throw error if already in use
    if (this->track_map.count(track_key) > 0) {
        std::string error_str = "ERROR  AssetsManager::loadTrack()  track key ";
        error_str += track_key;
        error_str += " is already in use";

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. open from file, throw error on fail
    sf::Music* track_ptr = new sf::Music();

    if (not track_ptr->openFromFile(path_2_track)) {
        std::string error_str = "ERROR  AssetsManager::loadTrack()  could not open ";
        error_str += "track at ";
        error_str += path_2_track;

        this->clear();

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  3. insert into track map
    this->track_map.insert(std::pair<std::string, sf::Music*>(track_key, track_ptr));
    this->current_track = this->track_map.begin();

    std::cout « "Track " « track_key « " inserted into track map" « std::endl;

    return;
}   /* loadTrack() */
```

4.1.3.13 nextTrack()

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

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

```
{
    // 1. stop current track
    this->stopTrack();

    // 2. increment current track
    this->current_track++;

    // 3. handle wrap around
    if (this->current_track == this->track_map.end()) {
        this->current_track = this->track_map.begin();
    }

    return;
}   /* nextTrack() */
```

4.1.3.14 pauseTrack()

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

Method to pause the current track.

```
{
    this->current_track->second->pause();

    return;
}   /* pauseTrack() */
```

4.1.3.15 playTrack()

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

Method to play the current track.

```
{
    this->current_track->second->play();

    return;
}   /* playTrack() */
```

4.1.3.16 previousTrack()

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

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

```
{
    // 1. stop current track
    this->stopTrack();

    // 2. handle wrap around
    if (this->current_track == this->track_map.begin()) {
        this->current_track = this->track_map.end();
    }

    // 3. decrement current track
    this->current_track--;

    return;
}   /* previousTrack() */
```

4.1.3.17 stopTrack()

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

Method to stop the current track.

```
{
    this->current_track->second->stop();

    return;
}   /* stopTrack() */
```

4.1.4 Member Data Documentation

4.1.4.1 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).

4.1.4.2 font_map

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

A map of pointers to loaded fonts.

4.1.4.3 sound_map

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

A map of pointers to loaded sounds.

4.1.4.4 soundbuffer_map

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

A map of pointers to sound buffers.

4.1.4.5 texture_map

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

A map of pointers to loaded textures.

4.1.4.6 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:

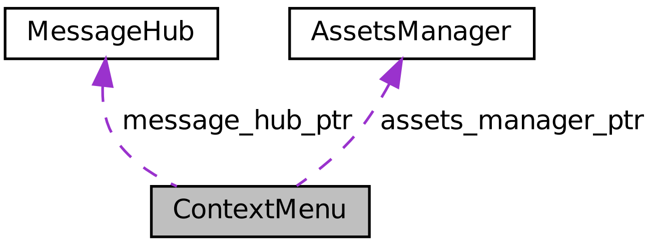
- header/ESC_core/AssetsManager.h
- source/ESC_core/AssetsManager.cpp

4.2 ContextMenu Class Reference

A class which defines a context menu for the game.

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

Collaboration diagram for ContextMenu:



Public Member Functions

- ContextMenu (sf::Event *, sf::RenderWindow *, AssetsManager *, MessageHub *)

  Constructor for the ContextMenu class.
- void processEvent (void)

  Method to processEvent ContextMenu. To be called once per event.
- void processMessage (void)

  Method to processMessage ContextMenu. To be called once per message.
- void draw (void)

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

  Destructor for the ContextMenu class.

Public Attributes

- ConsoleState console_state

  The current state of the console screen.
- bool console_string_changed

  Boolean which indicates if console string just changed.
- bool game_menu_up

  Indicates whether or not the game menu is up.
- size_t console_substring_idx

  The current final index of the console string draw.
- unsigned long long int frame

  The current frame of this object.
- double position_x

  The position of the object.
- double position_y

  The position of the object.
- std::string console_string

  The string to be printed to the console screen.
- sf::RectangleShape menu_frame

  The frame of the context menu.
- sf::RectangleShape visual_screen

  The context menu screen for visuals.
- sf::ConvexShape visual_screen_frame_top

  The top framing of the visual screen.
- sf::ConvexShape visual_screen_frame_left

  The left framing of the visual screen.
- sf::ConvexShape visual_screen_frame_bottom

  The bottom framing of the visual screen.
- sf::ConvexShape visual_screen_frame_right

  The right framing of the visual screen.
- sf::RectangleShape console_screen

  The context menu console screen (for animated text output).
- sf::ConvexShape console_screen_frame_top

  The top framing of the console screen.
- sf::ConvexShape console_screen_frame_left

  The left framing of the console screen.
- sf::ConvexShape console_screen_frame_bottom

  The bottom framing of the console screen.
- sf::ConvexShape console_screen_frame_right

  The right framing of the console screen.

Private Member Functions

- void __setUpMenuFrame (void)

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

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

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

Helper method to draw visual screen frame.
- void __setUpConsoleScreen (void)

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

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

  Helper method to draw console screen frame.
- void __setConsoleState (ConsoleState)

  Helper method to set state of console screen and update string if necessary.
- void __setConsoleString (void)

  Helper method to set console string depending on console state.
- void __drawConsoleText (void)

  Helper method to draw animated text to context menu console screen.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.
- void __sendQuitGameMessage (void)

  Helper method to format and send a quit game message.
- void __sendRestartGameMessage (void)

  Helper method to format and send a restart game message.

Private Attributes

- sf::Event ∗ event_ptr

  A pointer to the event class.
- sf::RenderWindow ∗ render_window_ptr

  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr

  A pointer to the message hub.

4.2.1 Detailed Description

A class which defines a context menu for the game.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ContextMenu()

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

Constructor for the ContextMenu class.

Parameters

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

```
{
    //  1. set attributes

    //  1.1. private
    this->event_ptr = event_ptr;
    this->render_window_ptr = render_window_ptr;

    this->assets_manager_ptr = assets_manager_ptr;
    this->message_hub_ptr = message_hub_ptr;

    //  1.2. public
    this->console_state = ConsoleState :: NONE_STATE;
    this->__setConsoleState(ConsoleState :: READY);

    this->console_string_changed = true;
    this->game_menu_up = false;

    this->frame = 0;

    this->position_x = GAME_WIDTH;
    this->position_y = 0;

    //  2. set up and position drawable attributes
    this->__setUpMenuFrame();
    this->__setUpVisualScreen();
    this->__setUpVisualScreenFrame();
    this->__setUpConsoleScreen();
    this->__setUpConsoleScreenFrame();

    std::cout « "ContextMenu constructed at " « this « std::endl;

    return;
}   /* ContextMenu() */
```

4.2.2.2 ∼ContextMenu()

```
ContextMenu::∼ContextMenu (
              void   )
```

Destructor for the ContextMenu class.

```
{
    std::cout « "ContextMenu at " « this « " destroyed" « std::endl;

    return;
}   /* ~ContextMenu() */
```

4.2.3 Member Function Documentation

4.2.3.1 __drawConsoleScreenFrame()

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

Helper method to draw console screen frame.

```
{
    this->render_window_ptr->draw(this->console_screen_frame_top);
    this->render_window_ptr->draw(this->console_screen_frame_left);
    this->render_window_ptr->draw(this->console_screen_frame_bottom);
    this->render_window_ptr->draw(this->console_screen_frame_right);

    return;
}   /* __drawContextScreenFrame() */
```

4.2.3.2 __drawConsoleText()

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

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

```
{
    //  1. set up console text (drawable)
    sf::Text console_text;

    if (this->console_string_changed) {
        this->assets_manager_ptr->getSound("console string print")->play();

        console_text.setString(this->console_string.substr(0, this->console_substring_idx));

        this->console_substring_idx++;

        while (
            (this->console_string.substr(0, this->console_substring_idx).back() == ' ') or
            (this->console_string.substr(0, this->console_substring_idx).back() == '\n')
        ) {
            this->console_substring_idx++;

            if (this->console_substring_idx >= this->console_string.size()) {
                break;
            }
        }

        if (this->console_substring_idx >= this->console_string.size()) {
            this->console_string_changed = false;
        }
    }

    else {
        console_text.setString(this->console_string);
    }

    console_text.setFont(*(this->assets_manager_ptr->getFont("Glass_TTY_VT220")));
    console_text.setCharacterSize(16);
    console_text.setFillColor(MONOCHROME_TEXT_GREEN);

    console_text.setPosition(
        this->position_x - 50 - 300 + 16,
        this->position_y + GAME_HEIGHT - 50 - 340 + 16
    );


    //  2. draw console text
    this->render_window_ptr->draw(console_text);


    //  3. assemble and draw blinking console cursor
    if ((this->frame % FRAMES_PER_SECOND) > FRAMES_PER_SECOND / 2) {
        sf::RectangleShape console_cursor(sf::Vector2f(10, 16));

        console_cursor.setFillColor(MONOCHROME_TEXT_GREEN);

        console_cursor.setPosition(
            console_text.getPosition().x,
            console_text.getPosition().y + console_text.getLocalBounds().height + 10
        );

        this->render_window_ptr->draw(console_cursor);
    }

    //  4. updating frame count if console is in menu state
    if (this->console_state == ConsoleState :: MENU) {
        std::string frame_count_string = "FRAME: ";
        frame_count_string += std::to_string(this->frame);
        sf::Text frame_count_text(
            frame_count_string,
            *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
            16
        );

        frame_count_text.setFillColor(MONOCHROME_TEXT_GREEN);

        frame_count_text.setPosition(
            console_text.getPosition().x,
            console_text.getPosition().y + console_text.getLocalBounds().height - 10
        );

        this->render_window_ptr->draw(frame_count_text);
    }

    return;
}   /* __drawConsoleText() */
```

4.2.3.3 __drawVisualScreenFrame()

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

Helper method to draw visual screen frame.

```
{
    this->render_window_ptr->draw(this->visual_screen_frame_top);
    this->render_window_ptr->draw(this->visual_screen_frame_left);
    this->render_window_ptr->draw(this->visual_screen_frame_bottom);
    this->render_window_ptr->draw(this->visual_screen_frame_right);

    return;
}   /* __drawVisualScreenFrame() */
```

4.2.3.4 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::Escape): {
            if (this->console_state == ConsoleState :: MENU) {
                this->__setConsoleState(ConsoleState :: READY);
            }

            else {
                this->__setConsoleState(ConsoleState :: MENU);
            }

            break;
        }


        case (sf::Keyboard::Q): {
            if (this->console_state == ConsoleState :: MENU) {
                this->__sendQuitGameMessage();
            }
        }


        case (sf::Keyboard::R): {
            if (this->console_state == ConsoleState :: MENU) {
                this->__sendRestartGameMessage();
            }
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.2.3.5 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.2.3.6 __sendQuitGameMessage()

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

Helper method to format and send a quit game message.

```
{
    Message quit_game_message;

    quit_game_message.channel = GAME_CHANNEL;
    quit_game_message.subject = "quit game";

    this->message_hub_ptr->sendMessage(quit_game_message);

    std::cout « "Quit game message sent by " « this « std::endl;
    return;
}   /* __sendQuitGameMessage() */
```

4.2.3.7 __sendRestartGameMessage()

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

Helper method to format and send a restart game message.

```
{
    Message restart_game_message;

    restart_game_message.channel = GAME_CHANNEL;
    restart_game_message.subject = "restart game";

    this->message_hub_ptr->sendMessage(restart_game_message);

    std::cout « "Restart game message sent by " « this « std::endl;
    return;
}   /* __sendRestartGameMessage() */
```

4.2.3.8 __setConsoleState()

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

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

Parameters

console_state	The state (ConsoleState) to set the console to.

```
{
    //  1. if no change, do nothing
    if (this->console_state == console_state) {
        return;
    }

    //  2. update console state, set console string accordingly
    this->console_state = console_state;
    this->__setConsoleString();

    return;
}   /* __setConsoleState() */
```

4.2.3.9 __setConsoleString()

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

Helper method to set console string depending on console state.

```
{
    this->console_string_changed = true;
    this->console_substring_idx = 0;

    this->console_string.clear();

    switch (this->console_state) {
        case (ConsoleState :: MENU): {
            //          32 char x 17 line console "--------------------------------\n";
            this->console_string                 = "         **** MENU ****         \n";
            this->console_string                += "                                \n";
            this->console_string                += "[R]:  RESTART                   \n";
            this->console_string                += "                                \n";
            this->console_string                += "[TAB]:  TOGGLE RESOURCE OVERLAY \n";
            this->console_string                += "[T]:  TUTORIAL                  \n";
            this->console_string                += "                                \n";
            this->console_string                += "                                \n";
            this->console_string                += "                                \n";
            this->console_string                += "                                \n";
            this->console_string                += "                                \n";
            this->console_string                += "                                \n";
            this->console_string                += "                                \n";
            this->console_string                += "[Q]:    QUIT                    \n";
            this->console_string                += "[ESC]:  CLOSE MENU              \n";
            this->console_string                += "                                \n";

            break;
        }


        case (ConsoleState :: TILE): {
            // take console string from tile state message

            break;
        }


        default: {
            //           32 char x 17 line console "--------------------------------\n";
            this->console_string                 = "  **** RTZ 64 CONTEXT V12 ****  \n";
            this->console_string                += "                                \n";
            this->console_string                += "64K RAM SYSTEM  38911 BYTES FREE\n";
            this->console_string                += "                                \n";
            this->console_string                += "[TAB]:  TOGGLE RESOURCE OVERLAY \n";
            this->console_string                += "                                \n";
            this->console_string                += "[ESC]:         MENU             \n";
            this->console_string                += "[LEFT CLICK]:  TILE INFO/OPTIONS\n";
            this->console_string                += "[RIGHT CLICK]: CLEAR SELECTION  \n";
            this->console_string                += "                                \n";
            this->console_string                += "[ENTER]:  END TURN              \n";
            this->console_string                += "                                \n";
            this->console_string                += "READY.                          ";

            break;
        }
    }

    return;
}   /* __setConsoleString() */
```

4.2.3.10 __setUpConsoleScreen()

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

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

```
{
    this->console_screen.setSize(sf::Vector2f(300, 340));
    this->console_screen.setOrigin(300, 340);
    this->console_screen.setPosition(
        this->position_x - 50,
        this->position_y + GAME_HEIGHT - 50
    );
    this->console_screen.setFillColor(MONOCHROME_SCREEN_BACKGROUND);

    return;
}   /* __setUpConsoleScreen() */
```

4.2.3.11 __setUpConsoleScreenFrame()

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

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

```
{
    int n_points = 4;

    //  1. top framing
    this->console_screen_frame_top.setPointCount(n_points);

    this->console_screen_frame_top.setPoint(
        0,
        sf::Vector2f(
            this->position_x - 50,
            this->position_y + GAME_HEIGHT - 50 - 340
        )
    );
    this->console_screen_frame_top.setPoint(
        1,
        sf::Vector2f(
            this->position_x - 50 + 16,
            this->position_y + GAME_HEIGHT - 50 - 340 - 16
        )
    );
    this->console_screen_frame_top.setPoint(
        2,
        sf::Vector2f(
            this->position_x - 350 - 16,
            this->position_y + GAME_HEIGHT - 50 - 340 - 16
        )
    );
    this->console_screen_frame_top.setPoint(
        3,
        sf::Vector2f(
            this->position_x - 350,
            this->position_y + GAME_HEIGHT - 50 - 340
        )
    );

    this->console_screen_frame_top.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->console_screen_frame_top.setOutlineThickness(2);
    this->console_screen_frame_top.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->console_screen_frame_top.move(0, -2);


    //  2. left framing
    this->console_screen_frame_left.setPointCount(n_points);

    this->console_screen_frame_left.setPoint(
        0,
        sf::Vector2f(
            this->position_x - 350,
            this->position_y + GAME_HEIGHT - 50 - 340
        )
    );
    this->console_screen_frame_left.setPoint(
        1,
        sf::Vector2f(
            this->position_x - 350 - 16,
            this->position_y + GAME_HEIGHT - 50 - 340 - 16
        )
    );
    this->console_screen_frame_left.setPoint(
        2,
        sf::Vector2f(
            this->position_x - 350 - 16,
            this->position_y + GAME_HEIGHT - 50 + 16
        )
    );
    this->console_screen_frame_left.setPoint(
        3,
        sf::Vector2f(
            this->position_x - 350,
            this->position_y + GAME_HEIGHT - 50
        )
    );

    this->console_screen_frame_left.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->console_screen_frame_left.setOutlineThickness(2);
    this->console_screen_frame_left.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->console_screen_frame_left.move(-2, 0);


    //  3. bottom framing
    this->console_screen_frame_bottom.setPointCount(n_points);

    this->console_screen_frame_bottom.setPoint(
        0,
        sf::Vector2f(
            this->position_x - 350,
            this->position_y + GAME_HEIGHT - 50
        )
    );
    this->console_screen_frame_bottom.setPoint(
        1,
        sf::Vector2f(
            this->position_x - 350 - 16,
            this->position_y + GAME_HEIGHT - 50 + 16
        )
    );
    this->console_screen_frame_bottom.setPoint(
        2,
        sf::Vector2f(
            this->position_x - 50 + 16,
            this->position_y + GAME_HEIGHT - 50 + 16
        )
    );
    this->console_screen_frame_bottom.setPoint(
        3,
        sf::Vector2f(
            this->position_x - 50,
            this->position_y + GAME_HEIGHT - 50
        )
    );

    this->console_screen_frame_bottom.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->console_screen_frame_bottom.setOutlineThickness(2);
    this->console_screen_frame_bottom.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->console_screen_frame_bottom.move(0, 2);


    //  4. right framing
    this->console_screen_frame_right.setPointCount(n_points);

    this->console_screen_frame_right.setPoint(
        0,
        sf::Vector2f(
            this->position_x - 50,
            this->position_y + GAME_HEIGHT - 50
        )
    );
    this->console_screen_frame_right.setPoint(
        1,
        sf::Vector2f(
            this->position_x - 50 + 16,
            this->position_y + GAME_HEIGHT - 50 + 16
        )
    );
    this->console_screen_frame_right.setPoint(
        2,
        sf::Vector2f(
            this->position_x - 50 + 16,
            this->position_y + GAME_HEIGHT - 50 - 340 - 16
        )
    );
    this->console_screen_frame_right.setPoint(
        3,
        sf::Vector2f(
            this->position_x - 50,
            this->position_y + GAME_HEIGHT - 50 - 340
        )
    );

    this->console_screen_frame_right.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->console_screen_frame_right.setOutlineThickness(2);
    this->console_screen_frame_right.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->console_screen_frame_right.move(2, 0);

    return;
}   /* __setUpConsoleScreenFrame() */
```

4.2.3.12 __setUpMenuFrame()

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

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

```
{
    this->menu_frame.setSize(sf::Vector2f(400, GAME_HEIGHT));
    this->menu_frame.setOrigin(400, 0);
    this->menu_frame.setPosition(this->position_x, this->position_y);
    this->menu_frame.setFillColor(MENU_FRAME_GREY);

    return;
}   /* __setUpMenuFrame() */
```

4.2.3.13 __setUpVisualScreen()

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

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

```
{
    this->visual_screen.setSize(sf::Vector2f(300, 300));
    this->visual_screen.setOrigin(300, 0);
    this->visual_screen.setPosition(this->position_x - 50, this->position_y + 50);
    this->visual_screen.setFillColor(MONOCHROME_SCREEN_BACKGROUND);

    return;
}   /* __setUpVisualScreen() */
```

4.2.3.14 __setUpVisualScreenFrame()

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

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

```
{
    int n_points = 4;

    //  1. top framing
    this->visual_screen_frame_top.setPointCount(n_points);

    this->visual_screen_frame_top.setPoint(
        0,
        sf::Vector2f(this->position_x - 50, this->position_y + 50)
    );
    this->visual_screen_frame_top.setPoint(
        1,
        sf::Vector2f(this->position_x - 50 + 16, this->position_y + 50 - 16)
    );
    this->visual_screen_frame_top.setPoint(
        2,
        sf::Vector2f(this->position_x - 350 - 16, this->position_y + 50 - 16)
    );
    this->visual_screen_frame_top.setPoint(
        3,
        sf::Vector2f(this->position_x - 350, this->position_y + 50)
    );

    this->visual_screen_frame_top.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->visual_screen_frame_top.setOutlineThickness(2);
    this->visual_screen_frame_top.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->visual_screen_frame_top.move(0, -2);


    //  2. left framing
    this->visual_screen_frame_left.setPointCount(n_points);

    this->visual_screen_frame_left.setPoint(
        0,
        sf::Vector2f(this->position_x - 350, this->position_y + 50)
    );
    this->visual_screen_frame_left.setPoint(
        1,
        sf::Vector2f(this->position_x - 350 - 16, this->position_y + 50 - 16)
    );
    this->visual_screen_frame_left.setPoint(
        2,
        sf::Vector2f(this->position_x - 350 - 16, this->position_y + 350 + 16)
    );
    this->visual_screen_frame_left.setPoint(
        3,
        sf::Vector2f(this->position_x - 350, this->position_y + 350)
    );

    this->visual_screen_frame_left.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->visual_screen_frame_left.setOutlineThickness(2);
    this->visual_screen_frame_left.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->visual_screen_frame_left.move(-2, 0);


    //  3. bottom framing
    this->visual_screen_frame_bottom.setPointCount(n_points);

    this->visual_screen_frame_bottom.setPoint(
        0,
        sf::Vector2f(this->position_x - 350, this->position_y + 350)
    );
    this->visual_screen_frame_bottom.setPoint(
        1,
        sf::Vector2f(this->position_x - 350 - 16, this->position_y + 350 + 16)
    );
    this->visual_screen_frame_bottom.setPoint(
        2,
        sf::Vector2f(this->position_x - 50 + 16, this->position_y + 350 + 16)
    );
    this->visual_screen_frame_bottom.setPoint(
        3,
        sf::Vector2f(this->position_x - 50, this->position_y + 350)
    );

    this->visual_screen_frame_bottom.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->visual_screen_frame_bottom.setOutlineThickness(2);
    this->visual_screen_frame_bottom.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->visual_screen_frame_bottom.move(0, 2);


    //  4. right framing
    this->visual_screen_frame_right.setPointCount(n_points);

    this->visual_screen_frame_right.setPoint(
        0,
        sf::Vector2f(this->position_x - 50, this->position_y + 350)
    );
    this->visual_screen_frame_right.setPoint(
        1,
        sf::Vector2f(this->position_x - 50 + 16, this->position_y + 350 + 16)
    );
    this->visual_screen_frame_right.setPoint(
        2,
        sf::Vector2f(this->position_x - 50 + 16, this->position_y + 50 - 16)
    );
    this->visual_screen_frame_right.setPoint(
        3,
        sf::Vector2f(this->position_x - 50, this->position_y + 50)
    );

    this->visual_screen_frame_right.setFillColor(VISUAL_SCREEN_FRAME_GREY);

    this->visual_screen_frame_right.setOutlineThickness(2);
    this->visual_screen_frame_right.setOutlineColor(sf::Color(0, 0, 0, 255));

    this->visual_screen_frame_right.move(2, 0);

    return;
}   /* __setUpVisualScreenFrame() */
```

4.2.3.15 draw()

```
void ContextMenu::draw (
              void  )
```

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

```
{
    //  1. menu frame
    this->render_window_ptr->draw(this->menu_frame);

    //  2. visual screen
    this->render_window_ptr->draw(this->visual_screen);
    this->__drawVisualScreenFrame();

    //  3. console screen
    this->render_window_ptr->draw(this->console_screen);
    this->__drawConsoleScreenFrame();
    this->__drawConsoleText();

    this->frame++;
    return;
}   /* draw() */
```

4.2.3.16 processEvent()

```
void ContextMenu::processEvent (
              void  )
```

Method to processEvent ContextMenu. To be called once per event.

```
{
    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.2.3.17 processMessage()

```
void ContextMenu::processMessage (
              void  )
```

Method to processMessage ContextMenu. To be called once per message.

```
{
    switch (this->console_state) {
        case (ConsoleState :: TILE): {
            //  process no tile selected
            if (not this->message_hub_ptr->isEmpty(NO_TILE_SELECTED_CHANNEL)) {
                Message no_tile_selected_message = this->message_hub_ptr->receiveMessage(
                    NO_TILE_SELECTED_CHANNEL
                );

                if (no_tile_selected_message.subject == "no tile selected") {
                    this->__setConsoleState(ConsoleState :: READY);

                    std::cout « "No tile selected message received by " « this «
                        std::endl;
                    this->message_hub_ptr->popMessage(NO_TILE_SELECTED_CHANNEL);
                }
            }

            //  process tile state
            if (not this->message_hub_ptr->isEmpty(TILE_STATE_CHANNEL)) {
                Message tile_state_message = this->message_hub_ptr->receiveMessage(
                    TILE_STATE_CHANNEL
                );

                if (tile_state_message.subject == "tile state") {
                    this->console_string = tile_state_message.string_payload["console string"];

                    this->console_string_changed = true;
                    this->console_substring_idx = 0;

                    std::cout « "Tile state message received by " « this « std::endl;
                    this->message_hub_ptr->popMessage(TILE_STATE_CHANNEL);
                }
            }

            //  process tile selected (subsequent left clicks causing program to hang)
            if (not this->message_hub_ptr->isEmpty(TILE_SELECTED_CHANNEL)) {
                this->message_hub_ptr->popMessage(TILE_SELECTED_CHANNEL);
            }

            break;
        }

        default: {
            //  process tile selected
            if (not this->message_hub_ptr->isEmpty(TILE_SELECTED_CHANNEL)) {
                Message tile_selected_message = this->message_hub_ptr->receiveMessage(
                    TILE_SELECTED_CHANNEL
                );

                if (tile_selected_message.subject == "tile selected") {
                    this->__setConsoleState(ConsoleState :: TILE);

                    std::cout « "Tile selected message received by " « this «
                        std::endl;
                    this->message_hub_ptr->popMessage(TILE_SELECTED_CHANNEL);
                }
            }

            break;
        }
    }

    return;
}   /* processMessage() */
```

4.2.4 Member Data Documentation

4.2.4.1 assets_manager_ptr

`AssetsManager* ContextMenu::assets_manager_ptr [private]`

A pointer to the assets manager.

4.2.4.2 console_screen

`sf::RectangleShape ContextMenu::console_screen`

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

4.2.4.3 console_screen_frame_bottom

```
sf::ConvexShape ContextMenu::console_screen_frame_bottom
```

The bottom framing of the console screen.

4.2.4.4 console_screen_frame_left

```
sf::ConvexShape ContextMenu::console_screen_frame_left
```

The left framing of the console screen.

4.2.4.5 console_screen_frame_right

```
sf::ConvexShape ContextMenu::console_screen_frame_right
```

The right framing of the console screen.

4.2.4.6 console_screen_frame_top

```
sf::ConvexShape ContextMenu::console_screen_frame_top
```

The top framing of the console screen.

4.2.4.7 console_state

```
ConsoleState ContextMenu::console_state
```

The current state of the console screen.

4.2.4.8 console_string

```
std::string ContextMenu::console_string
```

The string to be printed to the console screen.

4.2.4.9 console_string_changed

```
bool ContextMenu::console_string_changed
```

Boolean which indicates if console string just changed.

4.2.4.10 console_substring_idx

```
size_t ContextMenu::console_substring_idx
```

The current final index of the console string draw.

4.2.4.11 event_ptr

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

A pointer to the event class.

4.2.4.12 frame

```
unsigned long long int ContextMenu::frame
```

The current frame of this object.

4.2.4.13 game_menu_up

```
bool ContextMenu::game_menu_up
```

Indicates whether or not the game menu is up.

4.2.4.14 menu_frame

```
sf::RectangleShape ContextMenu::menu_frame
```

The frame of the context menu.

4.2.4.15 message_hub_ptr

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

A pointer to the message hub.

4.2.4.16 position_x

```
double ContextMenu::position_x
```

The position of the object.

4.2.4.17 position_y

```
double ContextMenu::position_y
```

The position of the object.

4.2.4.18 render_window_ptr

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

A pointer to the render window.

4.2.4.19 visual_screen

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

The context menu screen for visuals.

4.2.4.20 visual_screen_frame_bottom

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

The bottom framing of the visual screen.

4.2.4.21 visual_screen_frame_left

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

The left framing of the visual screen.

4.2.4.22 visual_screen_frame_right

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

The right framing of the visual screen.

4.2.4.23 visual_screen_frame_top

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

The top framing of the visual screen.

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

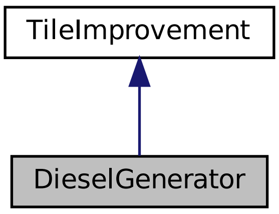
- header/ContextMenu.h
- source/ContextMenu.cpp

4.3 DieselGenerator Class Reference

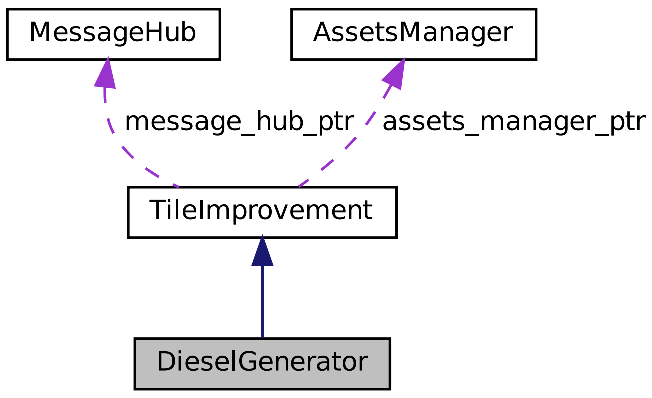
A settlement class (child class of TileImprovement).

```
#include <DieselGenerator.h>
```

Inheritance diagram for DieselGenerator:



Collaboration diagram for DieselGenerator:



Public Member Functions

- DieselGenerator (double, double, int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

  Constructor for the DieselGenerator class.
- std::string getTileOptionsSubstring (void)

  Helper method to assemble and return tile options substring.
- void setIsSelected (bool)

  Method to set the is selected attribute.
- void advanceTurn (void)

  Method to handle turn advance.
- void processEvent (void)

  Method to process DieselGenerator. To be called once per event.
- void processMessage (void)

  Method to process DieselGenerator. To be called once per message.
- void draw (void)

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

  Destructor for the DieselGenerator class.

Public Attributes

- int capacity_kW

  The rated production capacity [kW] of the diesel generator.
- int production_MWh

  The current production [MWh] of the diesel generator.
- int max_production_MWh

  The maximum production [MWh] for this turn.
- double smoke_da

  The per frame delta in smoke particle alpha value.

- double smoke_dx

  The per frame delta in smoke particle x position.
- double smoke_dy

  The per frame delta in smoke particle y position.
- double smoke_prob

  The probability of spawning a new smoke prob in any given frame.
- std::list< sf::Sprite > smoke_sprite_list

  A list of smoke sprite (for exhaust animation).
- int fuel_cost

  The fuel costs for this turn.
- int emissions_tonnes_CO2e

  The emissions for this turn.

Private Member Functions

- void __setUpTileImprovementSpriteAnimated (void)

  Helper method to set up tile improvement sprite (static).
- void __drawProductionMenu (void)

  Helper method to draw production menu assets.
- void __upgrade (void)

  Helper method to upgrade the diesel generator.
- void __computeProductionCosts (void)

  Helper method to compute production costs (fuel, O&M, emissions) based on current production level.
- void __breakdown (void)

  Helper method to trigger an equipment breakdown.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.
- void __sendImprovementStateMessage (void)

  Helper method to format and sent improvement state message.

Additional Inherited Members

4.3.1 Detailed Description

A settlement class (child class of TileImprovement).

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DieselGenerator()

```
DieselGenerator::DieselGenerator (
            double position_x,
            double position_y,
            int tile_resource,
            sf::Event * event_ptr,
            sf::RenderWindow * render_window_ptr,
            AssetsManager * assets_manager_ptr,
            MessageHub * message_hub_ptr )
```

Constructor for the DieselGenerator class.

Ref: Wikipedia [2023]

Parameters

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

```
  :
TileImprovement(
    position_x,
    position_y,
    tile_resource,
    event_ptr,
    render_window_ptr,
    assets_manager_ptr,
    message_hub_ptr
)
{
    //  1. set attributes

    //  1.1. private
    //...

    //  1.2. public
    this->tile_improvement_type = TileImprovementType :: DIESEL_GENERATOR;

    this->is_running = false;

    this->health = 100;

    this->capacity_kW = 100;
    this->upgrade_level = 1;

    this->production_MWh = 0;
    this->max_production_MWh = 72;

    this->smoke_da = 1e-8 * SECONDS_PER_FRAME;
    this->smoke_dx = 5 * SECONDS_PER_FRAME;
    this->smoke_dy = -10 * SECONDS_PER_FRAME;
    this->smoke_prob = 16 * SECONDS_PER_FRAME;

    this->smoke_sprite_list = {};

    this->fuel_cost = 0;
    this->emissions_tonnes_CO2e = 0;

    this->tile_improvement_string = "DIESEL GEN";

    this->__setUpTileImprovementSpriteAnimated();

    std::cout « "DieselGenerator constructed at " « this « std::endl;

    return;
}   /* DieselGenerator() */
```

4.3.2.2 ∼DieselGenerator()

```
DieselGenerator::∼DieselGenerator (
            void ) [virtual]
```

Destructor for the DieselGenerator class.

```
{
    std::cout « "DieselGenerator at " « this « " destroyed" « std::endl;

    return;
}   /* ~DieselGenerator() */
```

4.3.3 Member Function Documentation

4.3.3.1 __breakdown()

```
void DieselGenerator::__breakdown (
            void ) [private]
```

Helper method to trigger an equipment breakdown.

```
{
    TileImprovement :: __breakdown();

    this->production_MWh = 0;
    this->fuel_cost = 0;
    this->operation_maintenance_cost = 0;
    this->emissions_tonnes_CO2e = 0;

    return;
}   /* __breakdown() */
```

4.3.3.2 __computeProductionCosts()

```
void DieselGenerator::__computeProductionCosts (
            void ) [private]
```

Helper method to compute production costs (fuel, O&M, emissions) based on current production level.

```
{
    double litres_diesel = this->production_MWh * LITRES_DIESEL_PER_MWH_PRODUCTION;

    double fuel_cost = (litres_diesel * COST_PER_LITRE_DIESEL) / 1000;
    this->fuel_cost = round(fuel_cost);

    double emissions_tonnes_CO2e = (litres_diesel * KG_CO2E_PER_LITRE_DIESEL) / 1000;
    this->emissions_tonnes_CO2e = round(emissions_tonnes_CO2e);

    double operation_maintenance_cost =
        (this->production_MWh * DIESEL_OP_MAINT_COST_PER_MWH_PRODUCTION) / 1000;
    this->operation_maintenance_cost = round(operation_maintenance_cost);

    this->__sendTileStateRequest();

    return;
}   /* __computeProductionCosts() */
```

4.3.3.3 __drawProductionMenu()

```
void DieselGenerator::__drawProductionMenu (
              void ) [private]
```

Helper method to draw production menu assets.

```
{
    //  1. draw animated sprite (in off state)
    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
        sf::Vector2f initial_position = this->tile_improvement_sprite_animated[i].getPosition();
        this->tile_improvement_sprite_animated[i].setPosition(400 - 138, 400 + 16);

        sf::Color initial_colour = this->tile_improvement_sprite_animated[i].getColor();
        this->tile_improvement_sprite_animated[i].setColor(sf::Color(255, 255, 255, 255));

        sf::Vector2f initial_scale = this->tile_improvement_sprite_animated[i].getScale();
        this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1, 1));

        this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);

        this->tile_improvement_sprite_animated[i].setPosition(initial_position);
        this->tile_improvement_sprite_animated[i].setColor(initial_colour);
        this->tile_improvement_sprite_animated[i].setScale(initial_scale);
    }

    //  2. draw production text
    std::string production_string = "[W]:  INCREASE PRODUCTION\n";
    production_string            += "[S]:  DECREASE PRODUCTION\n";
    production_string            += "                          \n";

    production_string            += "PRODUCTION:  ";
    production_string            += std::to_string(this->production_MWh);
    production_string            += " MWh (MAX ";
    production_string            += std::to_string(this->max_production_MWh);
    production_string            += ")\n";

    production_string            += "FUEL COST:   ";
    production_string            += std::to_string(this->fuel_cost);
    production_string            += " K\n";

    production_string            += "O&M COST:    ";
    production_string            += std::to_string(this->operation_maintenance_cost);
    production_string            += " K\n";

    production_string            += "EMISSIONS:   ";
    production_string            += std::to_string(this->emissions_tonnes_CO2e);
    production_string            += " tonnes (CO2e)\n";

    sf::Text production_text(
        production_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    production_text.setOrigin(production_text.getLocalBounds().width / 2,0);
    production_text.setFillColor(MONOCHROME_TEXT_GREEN);

    production_text.setPosition(400 + 30, 400 - 55);

    this->render_window_ptr->draw(production_text);

    return;
}   /* __drawProductionMenu() */
```

4.3.3.4 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->just_built) {
        return;
    }


    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::U): {
            this->__upgrade();

            break;
        }


        case (sf::Keyboard::W): {
            if (this->production_menu_open) {
                this->production_MWh++;

                if (this->production_MWh > this->max_production_MWh) {
                    this->production_MWh = 0;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            break;
        }


        case (sf::Keyboard::S): {
            if (this->production_menu_open) {
                this->production_MWh--;

                if (this->production_MWh < 0) {
                    this->production_MWh = this->max_production_MWh;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }


    return;
}   /* __handleKeyPressEvents() */
```

4.3.3.5 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.3.3.6 __sendImprovementStateMessage()

```
void DieselGenerator::__sendImprovementStateMessage (
            void ) [private]
```

Helper method to format and sent improvement state message.

```
{
    Message improvement_state_message;

    improvement_state_message.channel = GAME_CHANNEL;
    improvement_state_message.subject = "improvement state";

    improvement_state_message.int_payload["dispatch_MWh"] = this->production_MWh;
    improvement_state_message.int_payload["fuel_cost"] = this->fuel_cost;
    improvement_state_message.int_payload["operation_maintenance_cost"] =
        this->operation_maintenance_cost;
    improvement_state_message.int_payload["emissions_tonnes_CO2e"] =
        this->emissions_tonnes_CO2e;

    this->message_hub_ptr->sendMessage(improvement_state_message);

    std::cout « "Improvement state message sent by " « this « std::endl;

    return;
}   /* __sendImprovementStateMessage() */
```

4.3.3.7 __setUpTileImprovementSpriteAnimated()

```
void DieselGenerator::__setUpTileImprovementSpriteAnimated (
            void ) [private]
```

Helper method to set up tile improvement sprite (static).

```
{
    sf::Sprite diesel_generator_sheet(
        *(this->assets_manager_ptr->getTexture("diesel generator"))
    );

    int n_elements = diesel_generator_sheet.getLocalBounds().height / 64;

    for (int i = 0; i < n_elements; i++) {
        this->tile_improvement_sprite_animated.push_back(
            sf::Sprite(
                *(this->assets_manager_ptr->getTexture("diesel generator")),
                sf::IntRect(0, i * 64, 64, 64)
            )
        );

        this->tile_improvement_sprite_animated.back().setOrigin(
            this->tile_improvement_sprite_animated.back().getLocalBounds().width / 2,
            this->tile_improvement_sprite_animated.back().getLocalBounds().height
        );

        this->tile_improvement_sprite_animated.back().setPosition(
            this->position_x,
            this->position_y - 32
        );

        this->tile_improvement_sprite_animated.back().setColor(
            sf::Color(255, 255, 255, 0)
        );
    }

    return;
}   /* __setUpTileImprovementSpriteAnimated() */
```

4.3.3.8 __upgrade()

```
void DieselGenerator::__upgrade (
              void   )  [private]
```

Helper method to upgrade the diesel generator.

```
{
    if (this->credits < DIESEL_GENERATOR_BUILD_COST) {
        std::cout « "Cannot upgrade diesel generator: insufficient credits (need "
            « DIESEL_GENERATOR_BUILD_COST « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    if (this->upgrade_level >= MAX_UPGRADE_LEVELS) {
        return;
    }

    this->is_running = false;

    this->__repair();

    this->capacity_kW += 100;
    this->upgrade_level++;

    this->production_MWh = 0;
    this->max_production_MWh += 72;

    this->just_upgraded = true;

    this->assets_manager_ptr->getSound("upgrade")->play();

    this->__sendCreditsSpentMessage(DIESEL_GENERATOR_BUILD_COST);
    this->__sendTileStateRequest();
    this->__sendGameStateRequest();

    return;
}   /* __upgrade() */
```

4.3.3.9 advanceTurn()

```
void DieselGenerator::advanceTurn (
              void   )  [virtual]
```

Method to handle turn advance.

Reimplemented from TileImprovement.

```
{
    //  1. send improvement state message
    this->__sendImprovementStateMessage();

    //  2. handle start/stop
    if ((not this->is_running) and (this->production_MWh > 0)) {
        this->is_running = true;
        this->assets_manager_ptr->getSound("diesel start")->play();
    }

    else if (this->is_running and (this->production_MWh <= 0)) {
        this->is_running = false;
        this->tile_improvement_sprite_animated[1].setScale(sf::Vector2f(1, 1));
    }

    //  3. handle equipment health
    if (this->is_running) {
        this->health--;

        if (this->health <= 0) {
            this->__breakdown();
        }
    }

    //  4. close menus
    if (this->production_menu_open) {
        this->__closeProductionMenu();
    }

    if (this->upgrade_menu_open) {
        this->__closeUpgradeMenu();
    }

    //  5. send tile state request (if selected)
    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* advanceTurn() */
```

4.3.3.10 draw()

```
void DieselGenerator::draw (
            void  )  [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    //  1. if just built, call base method and return
    if (this->just_built) {
        TileImprovement :: draw();

        return;
    }

    //  2. handle upgrade effects
    if (this->just_upgraded) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(
                    255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    255,
                    255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    255
                )
            );

            this->tile_improvement_sprite_animated[i].setScale(
                sf::Vector2f(
                    1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2)
                )
            );
        }

        this->upgrade_frame++;
    }

    if (this->upgrade_frame >= 2 * FRAMES_PER_SECOND) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(255,255,255,255)
            );

            this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1,1));
        }

        this->just_upgraded = false;
        this->upgrade_frame = 0;
    }


    //  3. draw first element of animated sprite
    this->render_window_ptr->draw(this->tile_improvement_sprite_animated[0]);


    //  4. draw second element of animated sprite
    double move_x = 0;
    double move_y = 0;

    if (this->is_running) {
        this->tile_improvement_sprite_animated[1].setScale(
            sf::Vector2f(
                1 + 0.05 * pow(cos((6 * M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                1 + 0.05 * pow(cos((6 * M_PI * this->frame) / FRAMES_PER_SECOND), 2)
            )
        );

        move_x = 1 * ((double)rand() / RAND_MAX) - 0.5;
        move_y = 1 * ((double)rand() / RAND_MAX) - 0.5;

        this->tile_improvement_sprite_animated[1].move(move_x, move_y);
    }

    this->render_window_ptr->draw(this->tile_improvement_sprite_animated[1]);

    if (this->is_running) {
        this->tile_improvement_sprite_animated[1].move(-1 * move_x, -1 * move_y);
    }


    //  5. draw smoke effects
    if (this->is_running) {
        if ((double)rand() / RAND_MAX < smoke_prob) {
            this->smoke_sprite_list.push_back(
                sf::Sprite(*(this->assets_manager_ptr->getTexture("emissions")))
            );

            this->smoke_sprite_list.back().setOrigin(
                this->smoke_sprite_list.back().getLocalBounds().width / 2,
                this->smoke_sprite_list.back().getLocalBounds().height / 2
            );

            this->smoke_sprite_list.back().setPosition(
                this->position_x + 9 + 4 * ((double)rand() / RAND_MAX) - 2,
                this->position_y - 33
            );
        }
    }

    std::list<sf::Sprite>::iterator iter = this->smoke_sprite_list.begin();

    double alpha = 255;

    while (iter != this->smoke_sprite_list.end()) {
        this->render_window_ptr->draw(*iter);

        alpha = (*iter).getColor().a;

        alpha -= this->smoke_da;

        if (alpha <= 0) {
            iter = this->smoke_sprite_list.erase(iter);
            continue;
        }

        (*iter).setColor(sf::Color(255, 255, 255, alpha));

        (*iter).move(
            this->smoke_dx + 2 * (((double)rand() / RAND_MAX) - 1) / FRAMES_PER_SECOND,
            this->smoke_dy
        );

        (*iter).rotate(((double)rand() / RAND_MAX));

        iter++;
    }


    //  6. handle dispatch illustration
    if (this->production_MWh > 0) {
        this->dispatch_text.setString(std::to_string(this->production_MWh));
        this->__drawDispatch();
    }


    //  7. draw production menu
    if (this->production_menu_open) {
        this->render_window_ptr->draw(this->production_menu_backing);
        this->render_window_ptr->draw(this->production_menu_backing_text);

        this->__drawProductionMenu();
    }


    //  8. handle broken effects
    if (this->is_broken) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
```

```
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(
                    255,
                    255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                    255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                    255
                )
            );
        }
    }

    this->frame++;
    return;
}   /* draw() */
```

4.3.3.11 getTileOptionsSubstring()

```
std::string DieselGenerator::getTileOptionsSubstring (
              void  )  [virtual]
```

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

Reimplemented from TileImprovement.

```
{
    int upgrade_cost = DIESEL_GENERATOR_BUILD_COST;

    //                   32 char x 17 line console "--------------------------------\n";
    std::string options_substring                = "CAPACITY:    ";
    options_substring                           += std::to_string(this->capacity_kW);
    options_substring                           += " kW (level ";
    options_substring                           += std::to_string(this->upgrade_level);
    options_substring                           += ")\n";

    options_substring                           += "PRODUCTION:  ";
    options_substring                           += std::to_string(this->production_MWh);
    options_substring                           += " MWh (MAX ";
    options_substring                           += std::to_string(this->max_production_MWh);
    options_substring                           += ")\n";

    options_substring                           += "HEALTH:      ";
    options_substring                           += std::to_string(this->health);
    options_substring                           += "/100";

    if (this->health <= 0) {
        options_substring                       += " ** BROKEN! **\n";
    }

    else {
        options_substring                       += "\n";
    }

    options_substring                           += "                                \n";
    options_substring                           += "  **** DIESEL GEN OPTIONS ****  \n";
    options_substring                           += "                                \n";

    options_substring                           += "     [E]:  ";

    if (this->is_broken) {
        options_substring                       += "*** BROKEN! ***\n";
    }

    else {
        options_substring                       += "OPEN PRODUCTION MENU\n";
    }

    if (this->upgrade_level < MAX_UPGRADE_LEVELS) {
        options_substring                           += "     [U]:  + 100 kW (";
        options_substring                           += std::to_string(upgrade_cost);
        options_substring                           +=" K)\n";
    }
    options_substring                            += "HOLD [P]:  SCRAP (";
    options_substring                            += std::to_string(SCRAP_COST);
    options_substring                            += " K)";

    return options_substring;
}   /* getTileOptionsSubstring() */
```

4.3.3.12 processEvent()

```
void DieselGenerator::processEvent (
            void ) [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processEvent();

    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.3.3.13 processMessage()

```
void DieselGenerator::processMessage (
            void ) [virtual]
```

Method to process DieselGenerator. To be called once per message.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processMessage();

    //...

    return;
}   /* processMessage() */
```

4.3.3.14 setIsSelected()

```
void DieselGenerator::setIsSelected (
            bool is_selected ) [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: setIsSelected(is_selected);

    if (this->is_running and this->is_selected) {
        this->assets_manager_ptr->getSound("diesel running")->play();
    }

    return;
}   /* setIsSelected() */
```

4.3.4 Member Data Documentation

4.3.4.1 capacity_kW

`int DieselGenerator::capacity_kW`

The rated production capacity [kW] of the diesel generator.

4.3.4.2 emissions_tonnes_CO2e

`int DieselGenerator::emissions_tonnes_CO2e`

The emissions for this turn.

4.3.4.3 fuel_cost

`int DieselGenerator::fuel_cost`

The fuel costs for this turn.

4.3.4.4 max_production_MWh

`int DieselGenerator::max_production_MWh`

The maximum production [MWh] for this turn.

4.3.4.5 production_MWh

```
int DieselGenerator::production_MWh
```

The current production [MWh] of the diesel generator.

4.3.4.6 smoke_da

```
double DieselGenerator::smoke_da
```

The per frame delta in smoke particle alpha value.

4.3.4.7 smoke_dx

```
double DieselGenerator::smoke_dx
```

The per frame delta in smoke particle x position.

4.3.4.8 smoke_dy

```
double DieselGenerator::smoke_dy
```

The per frame delta in smoke particle y position.

4.3.4.9 smoke_prob

```
double DieselGenerator::smoke_prob
```

The probability of spawning a new smoke prob in any given frame.

4.3.4.10 smoke_sprite_list

```
std::list<sf::Sprite> DieselGenerator::smoke_sprite_list
```

A list of smoke sprite (for exhaust animation).

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

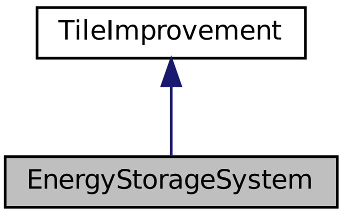
- header/DieselGenerator.h
- source/DieselGenerator.cpp

4.4 EnergyStorageSystem Class Reference

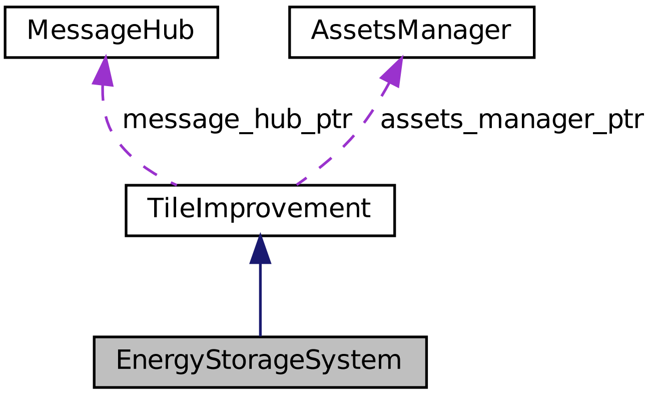
A settlement class (child class of TileImprovement).

```
#include <EnergyStorageSystem.h>
```

Inheritance diagram for EnergyStorageSystem:



Collaboration diagram for EnergyStorageSystem:



Public Member Functions

- EnergyStorageSystem (double, double, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

    Constructor for the EnergyStorageSystem class.
- void setIsSelected (bool)

    Method to set the is selected attribute.
- std::string getTileOptionsSubstring (void)

    Helper method to assemble and return tile options substring.
- void processEvent (void)

Method to process EnergyStorageSystem. To be called once per event.
- void processMessage (void)

  Method to process EnergyStorageSystem. To be called once per message.
- void draw (void)

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

  Destructor for the EnergyStorageSystem class.

Public Attributes

- int capacity_MWh

  The rated energy capacity [MWh] of the energy storage system.
- int charge_MWh

  The charge [MWh] in the energy storage system.

Private Member Functions

- void __setUpTileImprovementSpriteStatic (void)

  Helper method to set up tile improvement sprite (static).
- void __setUpProductionMenu (void)

  Helper method to set up and position production menu assets (drawable).
- void __upgrade (void)

  Helper method to upgrade the diesel generator.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.

Additional Inherited Members

4.4.1 Detailed Description

A settlement class (child class of TileImprovement).

4.4.2 Constructor & Destructor Documentation

4.4.2.1 EnergyStorageSystem()

```
EnergyStorageSystem::EnergyStorageSystem (
              double position_x,
              double position_y,
              sf::Event * event_ptr,
              sf::RenderWindow * render_window_ptr,
              AssetsManager * assets_manager_ptr,
              MessageHub * message_hub_ptr )
```

Constructor for the EnergyStorageSystem class.

Ref: Wikipedia [2023]

Parameters

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

```
  :
TileImprovement(
    position_x,
    position_y,
    event_ptr,
    render_window_ptr,
    assets_manager_ptr,
    message_hub_ptr
)
{
    //  1. set attributes

    //  1.1. private
    //...

    //  1.2. public
    this->tile_improvement_type = TileImprovementType :: ENERGY_STORAGE_SYSTEM;

    this->is_running = false;

    this->health = 100;

    this->capacity_MWh = 1;
    this->upgrade_level = 1;

    this->charge_MWh = 0;

    this->tile_improvement_string = "ENERGY STORAGE";

    this->__setUpTileImprovementSpriteStatic();
    this->__setUpProductionMenu();

    std::cout « "EnergyStorageSystem constructed at " « this « std::endl;

    return;
}   /* EnergyStorageSystem() */
```

4.4.2.2 ∼EnergyStorageSystem()

```
EnergyStorageSystem::∼EnergyStorageSystem (
              void  )  [virtual]
```

Destructor for the EnergyStorageSystem class.

```
{
    std::cout « "EnergyStorageSystem at " « this « " destroyed" « std::endl;

    return;
}   /* ~EnergyStorageSystem() */
```

4.4.3 Member Function Documentation

4.4.3.1 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::U): {
            if (this->upgrade_level < MAX_UPGRADE_LEVELS) {
                this->__upgrade();
            }

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.4.3.2 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.4.3.3 __setUpProductionMenu()

```
void EnergyStorageSystem::__setUpProductionMenu (
              void ) [private]
```

Helper method to set up and position production menu assets (drawable).

```
{
    //  1. modify production menu text
    this->production_menu_backing_text.setString("**** DISCHARGE MENU ****");
    this->production_menu_backing_text.setFont(
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220"))
    );
    this->production_menu_backing_text.setCharacterSize(16);
    this->production_menu_backing_text.setFillColor(MONOCHROME_TEXT_GREEN);
    this->production_menu_backing_text.setOrigin(
        this->production_menu_backing_text.getLocalBounds().width / 2, 0
    );
    this->production_menu_backing_text.setPosition(400, 400 - 128 + 4);

    return;
}   /* __setUpProductionMenu() */
```

4.4.3.4 __setUpTileImprovementSpriteStatic()

```
void EnergyStorageSystem::__setUpTileImprovementSpriteStatic (
              void ) [private]
```

Helper method to set up tile improvement sprite (static).

```
{
    this->tile_improvement_sprite_static.setTexture(
        *(this->assets_manager_ptr->getTexture("energy storage system"))
    );

    this->tile_improvement_sprite_static.setOrigin(
        this->tile_improvement_sprite_static.getLocalBounds().width / 2,
        this->tile_improvement_sprite_static.getLocalBounds().height
    );

    this->tile_improvement_sprite_static.setPosition(
        this->position_x,
        this->position_y - 32
    );

    this->tile_improvement_sprite_static.setColor(
        sf::Color(255, 255, 255, 0)
    );

    return;
}   /* __setUpTileImprovementSpriteStatic() */
```

4.4.3.5 __upgrade()

```
void EnergyStorageSystem::__upgrade (
              void ) [private]
```

Helper method to upgrade the diesel generator.

```
{
    /*
    int upgrade_cost = DIESEL_GENERATOR_BUILD_COST;

    if (this->credits < upgrade_cost) {
        std::cout « "Cannot upgrade diesel generator: insufficient credits (need "
            « upgrade_cost « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->is_running = false;

    this->health = 100;

    this->capacity_kW += 100;
    this->upgrade_level++;

    this->production_MWh = 0;
    this->max_production_MWh += 72;

    this->just_upgraded = true;

    this->assets_manager_ptr->getSound("upgrade")->play();

    this->__sendCreditsSpentMessage(upgrade_cost);
    this->__sendTileStateRequest();
    this->__sendGameStateRequest();
    */

    return;
}   /* __upgrade() */
```

4.4.3.6 draw()

```
void EnergyStorageSystem::draw (
            void   )  [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    //  1. if just built, call base method and return
    if (this->just_built) {
        TileImprovement :: draw();

        return;
    }


    //  2. draw static sprite
    this->render_window_ptr->draw(this->tile_improvement_sprite_static);


    //  3. draw production menu
    if (this->production_menu_open) {
        this->render_window_ptr->draw(this->production_menu_backing);
        this->render_window_ptr->draw(this->production_menu_backing_text);

        //...
    }

    this->frame++;
    return;
}   /* draw() */
```

4.4.3.7 getTileOptionsSubstring()

```
std::string EnergyStorageSystem::getTileOptionsSubstring (
            void   )  [virtual]
```

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

Reimplemented from TileImprovement.

```
{
    int upgrade_cost = ENERGY_STORAGE_SYSTEM_BUILD_COST;

    //                    32 char x 17 line console "--------------------------------\n";
    std::string options_substring                = "CAPACITY:  ";
    options_substring                           += std::to_string(this->capacity_MWh);
    options_substring                           += " MWh (level ";
    options_substring                           += std::to_string(this->upgrade_level);
    options_substring                           += ")\n";

    options_substring                           += "CHARGE:    ";
    options_substring                           += std::to_string(this->charge_MWh);
    options_substring                           += " MWh\n";

    options_substring                           += "HEALTH:    ";
    options_substring                           += std::to_string(this->health);
    options_substring                           += "/100\n";

    options_substring                           += "                                \n";
    options_substring                           += "**** ENERGY STORAGE OPTIONS ****\n";
    options_substring                           += "                                \n";
    options_substring                           += "      [E]:  OPEN DISCHARGE MENU  \n";

    if (this->upgrade_level < MAX_UPGRADE_LEVELS) {
        options_substring                           += "      [U]:  UPGRADE (";
        options_substring                           += std::to_string(upgrade_cost);
        options_substring                           +=" K)\n";
    }

    options_substring                           += "HOLD [P]:  SCRAP (";
    options_substring                           += std::to_string(SCRAP_COST);
    options_substring                           += " K)";

    return options_substring;
}   /* getTileOptionsSubstring() */
```

4.4.3.8 processEvent()

```
void EnergyStorageSystem::processEvent (
            void )  [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processEvent();

    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.4.3.9 processMessage()

```
void EnergyStorageSystem::processMessage (
             void  )  [virtual]
```

Method to process EnergyStorageSystem. To be called once per message.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processMessage();

    //...

    return;
}   /* processMessage() */
```

4.4.3.10 setIsSelected()

```
void EnergyStorageSystem::setIsSelected (
             bool is_selected )  [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: setIsSelected(is_selected);

    if (this->is_selected) {
        this->assets_manager_ptr->getSound("energy storage system")->play();
    }

    return;
}   /* setIsSelected() */
```

4.4.4 Member Data Documentation

4.4.4.1 capacity_MWh

```
int EnergyStorageSystem::capacity_MWh
```

The rated energy capacity [MWh] of the energy storage system.

4.4.4.2 charge_MWh

```
int EnergyStorageSystem::charge_MWh
```

The charge [MWh] in the energy storage system.

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

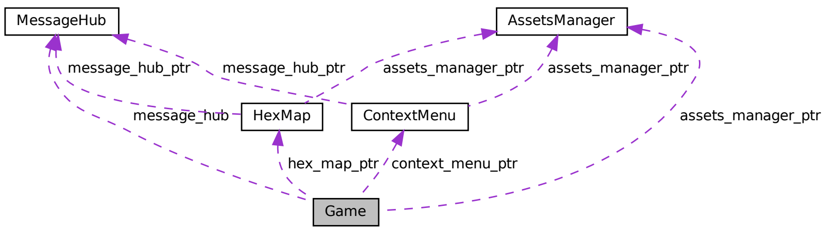
- header/EnergyStorageSystem.h
- source/EnergyStorageSystem.cpp

4.5 Game Class Reference

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

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

Collaboration diagram for Game:



Public Member Functions

- Game (sf::RenderWindow ∗, AssetsManager ∗)

    Constructor for the Game class.
- bool run (void)

    Method to run game (defines game loop).
- ∼Game (void)

    Destructor for the Game class.

Public Attributes

- GamePhase game_phase

  The current phase of the game.
- bool quit_game

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

  Boolean indicating whether or not the game loop is broken.
- bool show_frame_clock_overlay

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

  Boolean indicating whether or not to check terminating conditions.
- bool message_deadlock

  A boolean indicating whether a message deadlock has been detected.
- unsigned long long int frame

  The current frame of the game.
- double time_since_start_s

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

  Current game year.
- int month

  Current game month.
- int population

  Current population.
- int credits

  Current balance of credits.
- int demand_MWh

  Current energy demand [MWh].
- int demand_remaining_MWh

  The current remaining energy demand [MWh].
- int cumulative_emissions_tonnes

  Cumulative emissions [tonnes] (1 tonne = 1000 kg).
- int message_deadlock_frame

  A frame counter for detecting message deadlock.
- int turn = 0

  The current game turn.
- std::vector< double > demand_vec_MWh

  A vector of daily demands [MWh] for the current month.
- sf::Clock clock

  The game clock.
- sf::Event event

  The game events class.
- MessageHub message_hub

  The message hub (for inter-object message traffic).
- HexMap ∗ hex_map_ptr

  Pointer to the hex map (defines game world).
- ContextMenu ∗ context_menu_ptr

  Pointer to the context menu.

Private Member Functions

- void __toggleFrameClockOverlay (void)

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

    Helper method to check terminating conditions (i.e., loss or victory conditions).
- void __advanceTurn (void)

    Helper method to advance turn.
- void __computeCurrentDemand (void)

    Helper method to compute current energy demand.
- void __handleKeyPressEvents (void)

    Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

    Helper method to handle mouse button events.
- void __handleImprovementStateMessage (Message)

    Helper method to handle improvement state messages.
- void __processEvent (void)

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

    Helper method to process Game. To be called once per message.
- void __sendGameStateMessage (void)

    Helper method to format and send a game state message.
- void __sendTurnAdvanceMessage (void)

    Helper method to format and send a turn advance message.
- void __sendCreditsEarnedMessage (void)

    Helper method to format and send a credits earned message.
- void __insufficientCreditsAlarm (void)

    Helper method to sound and display and insufficient credits alarm.
- void __drawFrameClockOverlay (void)

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

    Helper method to heads-up display (HUD).
- void __draw (void)

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

Private Attributes

- sf::RenderWindow ∗ render_window_ptr

    A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

    A pointer to the assets manager.

4.5.1 Detailed Description

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

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Game()

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

Constructor for the Game class.

```
{
    //  1. set attributes

    //  1.1. private
    this->render_window_ptr = render_window_ptr;

    this->assets_manager_ptr = assets_manager_ptr;

    //  1.2. public
    this->game_phase = GamePhase :: BUILD_SETTLEMENT;

    this->quit_game = false;
    this->game_loop_broken = false;
    this->show_frame_clock_overlay = false;
    this->check_terminating_conditions = false;

    this->frame = 0;
    this->time_since_start_s = 0;

    this->message_deadlock = false;
    this->message_deadlock_frame = 0;

    double seconds_since_epoch = time(NULL);
    double years_since_epoch = seconds_since_epoch / SECONDS_PER_YEAR;

    this->year = 1970 + (int)years_since_epoch;
    this->month = (years_since_epoch - (int)years_since_epoch) * 12 + 1;
    while (this->month > 12) {
        this->month -= 12;
    }

    this->population = 0;
    this->credits = STARTING_CREDITS;
    this->demand_MWh = 0;
    this->demand_remaining_MWh = 0;
    this->cumulative_emissions_tonnes = 0;

    this->demand_vec_MWh.resize(30, 0);

    this->hex_map_ptr = new HexMap(
        6,
        &(this->event),
        this->render_window_ptr,
        this->assets_manager_ptr,
        &(this->message_hub)
    );

    this->context_menu_ptr = new ContextMenu(
        &(this->event),
        this->render_window_ptr,
        this->assets_manager_ptr,
        &(this->message_hub)
    );

    //  2. add message channel(s)
    this->message_hub.addChannel(GAME_CHANNEL);
    this->message_hub.addChannel(GAME_STATE_CHANNEL);

    std::cout « "Game constructed at " « this « std::endl;

    return;
}   /* Game() */
```

4.5.2.2 ∼Game()

```
Game::∼Game (
              void )
```

Destructor for the Game class.

```
{
    //  1. clean up attributes
    delete this->hex_map_ptr;
    delete this->context_menu_ptr;

    std::cout « "Game at " « this « " destroyed" « std::endl;

    return;
}   /* ~Game() */
```

4.5.3 Member Function Documentation

4.5.3.1 __advanceTurn()

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

Helper method to advance turn.

```
{
    //  1. advance turn
    this->turn++;

    //  2. advance month/year
    this->month++;
    if (this->month > 12) {
        this->year++;
        this->month = 1;
    }

    //  3. update population
    if (this->turn == 1) {
        this->population = STARTING_POPULATION;
    }

    else {
        this->population = ceil(this->population * POPULATION_MONTHLY_GROWTH_RATE);
    }

    //  4. update demand
    this->__computeCurrentDemand();

    //  5. send turn advance message
    this->__sendTurnAdvanceMessage();

}   /* __advanceTurn() */
```

4.5.3.2 __checkTerminatingConditions()

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

Helper method to check terminating conditions (i.e., loss or victory conditions).

```
{
    std::cout « "Game :: __checkTerminatingConditions()" « std::endl;

    //...

    return;
}   /* __checkTerminatingConditions() */
```

4.5.3.3 __computeCurrentDemand()

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

Helper method to compute current energy demand.

```
{
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator(seed);

    std::normal_distribution<double> normal_dist(
        MEAN_DAILY_DEMAND_RATIOS[this->month - 1],
        STDEV_DAILY_DEMAND_RATIOS[this->month - 1]
    );

    double demand_MWh = 0;

    for (int i = 0; i < 30; i++) {
        this->demand_vec_MWh[i] =
            normal_dist(generator) * MAXIMUM_DAILY_DEMAND_PER_CAPITA * this->population;

        demand_MWh += this->demand_vec_MWh[i];
    }

    this->demand_MWh = round(demand_MWh);
    this->demand_remaining_MWh = this->demand_MWh;

    return;
}   /* __computeCurrentDemand() */
```

4.5.3.4 __draw()

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

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

```
{
    this->__drawHUD();

    if (this->show_frame_clock_overlay) {
        this->__drawFrameClockOverlay();
    }

    return;
}   /* draw() */
```

4.5.3.5 __drawFrameClockOverlay()

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

Helper method to draw frame clock overlay.

```
{
    std::string frame_clock_string = "FRAME: ";
    frame_clock_string += std::to_string(this->frame);
    frame_clock_string += "\nTIME SINCE START [s]: ";
    frame_clock_string += std::to_string(this->time_since_start_s);

    sf::Text frame_clock_text(
        frame_clock_string,
        *(this->assets_manager_ptr->getFont("DroidSansMono")),
        16
    );

    sf::RectangleShape frame_clock_backing(
        sf::Vector2f(
            1.02 * frame_clock_text.getLocalBounds().width,
            1.20 * frame_clock_text.getLocalBounds().height
        )
    );
    frame_clock_backing.setFillColor(sf::Color(0, 0, 0, 255));

    this->render_window_ptr->draw(frame_clock_backing);
    this->render_window_ptr->draw(frame_clock_text);

    return;
}   /* __drawFrameClockOverlay() */
```

4.5.3.6 __drawHUD()

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

Helper method to heads-up display (HUD).

```
{
    // 1. first line (top)
    std::string HUD_string = "YEAR: ";
    HUD_string += std::to_string(this->year);

    HUD_string += "    MONTH: ";
    HUD_string += std::to_string(this->month);

    HUD_string += "    POPULATION: ";
    HUD_string += std::to_string(this->population);

    HUD_string += "    CREDITS: ";
    HUD_string += std::to_string(this->credits);
    HUD_string += " K";

    HUD_string += "    CURRENT DEMAND: ";
    HUD_string += std::to_string(this->demand_MWh);
    HUD_string += " MWh";

    sf::Text HUD_text(
        HUD_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    HUD_text.setPosition(
        (800 - HUD_text.getLocalBounds().width) / 2,
        8
    );

    HUD_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(HUD_text);


    // 2. second line (top)
    HUD_string = "CUMULATIVE EMISSIONS: ";
    HUD_string += std::to_string(this->cumulative_emissions_tonnes);
    HUD_string += " tonnes (CO2e)";

    HUD_string += "    LIFETIME LIMIT: ";
    HUD_string += std::to_string(EMISSIONS_LIFETIME_LIMIT_TONNES);
    HUD_string += " tonnes (CO2e)";

    HUD_text.setString(HUD_string);

    HUD_text.setPosition(
        (800 - HUD_text.getLocalBounds().width) / 2,
        35
    );

    this->render_window_ptr->draw(HUD_text);


    // 3. third line (bottom)
    HUD_string = "GAME PHASE: ";

    switch (this->game_phase) {
        case (GamePhase :: BUILD_SETTLEMENT): {
            HUD_string += "BUILD SETTLEMENT";

            break;
        }


        case (GamePhase :: SYSTEM_MANAGEMENT): {
            HUD_string += "SYSTEM MANAGEMENT";

            break;
        }


        case (GamePhase :: LOSS_EMISSIONS): {
            HUD_string += "LOSS (EMISSIONS)";

            break;
        }


        case (GamePhase :: LOSS_DEMAND): {
            HUD_string += "LOSS (DEMAND)";

            break;
        }


        case (GamePhase :: LOSS_CREDITS): {
            HUD_string += "LOSS (CREDITS)";

            break;
        }


        case (GamePhase :: VICTORY): {
            HUD_string += "VICTORY";

            break;
        }


        default: {
            HUD_string += "???";

            break;
        }
    }

    HUD_string += "    TURN: ";
    HUD_string += std::to_string(this->turn);

    HUD_text.setString(HUD_string);

    HUD_text.setPosition(
        (800 - HUD_text.getLocalBounds().width) / 2,
        GAME_HEIGHT - 35
    );

    this->render_window_ptr->draw(HUD_text);

    return;
}   /* __drawHUD() */
```

4.5.3.7 __handleImprovementStateMessage()

```
void Game::__handleImprovementStateMessage (
            Message improvement_state_message )  [private]
```

Helper method to handle improvement state messages.

```
{
    //  1. unpack message and update game attributes
    if (improvement_state_message.int_payload.count("dispatch_MWh") > 0) {
        this->demand_remaining_MWh -= improvement_state_message.int_payload["dispatch_MWh"];

        this->credits +=
            round(CREDITS_PER_MWH_SERVED * improvement_state_message.int_payload["dispatch_MWh"]);

        this->__sendCreditsEarnedMessage();
    }
    if (improvement_state_message.int_payload.count("fuel_cost") > 0) {
        this->credits -= improvement_state_message.int_payload["fuel_cost"];
    }

    if (improvement_state_message.int_payload.count("operation_maintenance_cost") > 0) {
        this->credits -=
            improvement_state_message.int_payload["operation_maintenance_cost"];
    }

    if (improvement_state_message.int_payload.count("emissions_tonnes_CO2e") > 0) {
        this->cumulative_emissions_tonnes +=
            improvement_state_message.int_payload["emissions_tonnes_CO2e"];
    }

    return;
}   /* __handleImprovementStateMessage() */
```

4.5.3.8 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    switch (this->event.key.code) {
        case (sf::Keyboard::Enter): {
            if (this->game_phase == GamePhase :: SYSTEM_MANAGEMENT) {
                this->__advanceTurn();
            }

            break;
        }


        case (sf::Keyboard::Tilde): {
            this->__toggleFrameClockOverlay();

            break;
        }


        case (sf::Keyboard::Tab): {
            this->hex_map_ptr->toggleResourceOverlay();

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.5.3.9 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    switch (this->event.mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.5.3.10 __insufficientCreditsAlarm()

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

Helper method to sound and display and insufficient credits alarm.

```
{
    //  1. sound buzzer
    this->assets_manager_ptr->getSound("insufficient credits")->play();

    //  2. construct alarm text and backing rectangle
    sf::Text insufficient_credits_text(
        "INSUFFICIENT CREDITS",
        (*(this->assets_manager_ptr->getFont("DroidSansMono"))),
        32
    );

    insufficient_credits_text.setOrigin(
        insufficient_credits_text.getLocalBounds().width / 2,
        insufficient_credits_text.getLocalBounds().height / 2
    );

    insufficient_credits_text.setPosition(400, GAME_HEIGHT / 2);

    sf::RectangleShape backing_rectangle(
        sf::Vector2f(
            1.1 * insufficient_credits_text.getLocalBounds().width,
            1.5 * insufficient_credits_text.getLocalBounds().height
        )
    );

    backing_rectangle.setFillColor(RESOURCE_CHIP_GREY);

    backing_rectangle.setOrigin(
        backing_rectangle.getLocalBounds().width / 2,
        backing_rectangle.getLocalBounds().height / 2
    );

    backing_rectangle.setPosition(400, (GAME_HEIGHT / 2) + 8);

    //  3. display loop (blocking ~3 seconds)
    bool red_flag = true;
    int alarm_frame = 0;
    double time_since_alarm_s = 0;

    sf::Clock alarm_clock;

    while (alarm_frame < 2.5 * FRAMES_PER_SECOND) {


        time_since_alarm_s = alarm_clock.getElapsedTime().asSeconds();

        if (time_since_alarm_s >= (alarm_frame + 1) * SECONDS_PER_FRAME) {
            while (this->render_window_ptr->pollEvent(this->event)) {
                // do nothing!
            }

            this->render_window_ptr->clear();

            this->hex_map_ptr->draw();
            this->context_menu_ptr->draw();
            this->__draw();

            if (alarm_frame % (FRAMES_PER_SECOND / 3) == 0) {
                if (red_flag) {
                    red_flag = false;
                }

                else {
                    red_flag = true;
                }
            }

            if (red_flag) {
                insufficient_credits_text.setFillColor(MONOCHROME_TEXT_RED);
            }

            else {
                insufficient_credits_text.setFillColor(sf::Color(255, 255, 255));
            }

            this->render_window_ptr->draw(backing_rectangle);
            this->render_window_ptr->draw(insufficient_credits_text);

            this->render_window_ptr->display();

            alarm_frame++;
            this->frame++;
        }

        // check track status, move to next if stopped
        if (this->assets_manager_ptr->getTrackStatus() == sf::SoundSource::Stopped) {
            this->assets_manager_ptr->nextTrack();
            this->assets_manager_ptr->playTrack();
        }
    }

    return;
}   /* __insufficientCreditsAlarm( */
```

4.5.3.11 __processEvent()

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

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

```
{
    if (this->event.type == sf::Event::Closed) {
        this->quit_game = true;
        this->game_loop_broken = true;
    }

    if (this->event.type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event.type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* __processEvent() */
```

4.5.3.12 __processMessage()

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

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

```
{
    if (not this->message_hub.isEmpty(GAME_CHANNEL)) {
        Message game_channel_message = this->message_hub.receiveMessage(GAME_CHANNEL);

        if (game_channel_message.subject == "quit game") {
            this->quit_game = true;
            this->game_loop_broken = true;

            std::cout « "Quit game message received by " « this « std::endl;
            this->message_hub.popMessage(GAME_CHANNEL);
        }

        if (game_channel_message.subject == "restart game") {
            this->game_loop_broken = true;

            std::cout « "Restart game message received by " « this « std::endl;
            this->message_hub.popMessage(GAME_CHANNEL);
        }

        if (game_channel_message.subject == "state request") {
            std::cout « "Game state request message received by " « this « std::endl;

            this->__sendGameStateMessage();
            this->message_hub.popMessage(GAME_CHANNEL);
        }

        if (game_channel_message.subject == "credits spent") {
            this->credits -= game_channel_message.int_payload["credits spent"];

            std::cout « "Credits spent message (" «
                game_channel_message.int_payload["credits spent"] « ") received by "
                « this « std::endl;

            std::cout « "Current credits (Game): " « this->credits « " K" «
                std::endl;

            this->message_hub.popMessage(GAME_CHANNEL);
        }

        if (game_channel_message.subject == "insufficient credits") {
            std::cout « "Insufficient credits message received by " « this «
                std::endl;

            this->__insufficientCreditsAlarm();

            this->message_hub.popMessage(GAME_CHANNEL);
        }

        if (game_channel_message.subject == "update game phase") {
            std::cout « "Update game phase message received by " « this « std::endl;

            if (
                game_channel_message.string_payload["game phase"] == "system management"
            ) {
                this->game_phase = GamePhase :: SYSTEM_MANAGEMENT;
                this->__advanceTurn();
            }

            else if (
                game_channel_message.string_payload["game phase"] == "loss emissions"
            ) {
                this->game_phase = GamePhase :: LOSS_EMISSIONS;
            }

            else if (
                game_channel_message.string_payload["game phase"] == "loss demand"
            ) {
                this->game_phase = GamePhase :: LOSS_DEMAND;
            }

            else if (
                game_channel_message.string_payload["game phase"] == "loss credits"
            ) {
                this->game_phase = GamePhase :: LOSS_CREDITS;
            }

            else if (
                game_channel_message.string_payload["game phase"] == "victory"
            ) {
                this->game_phase = GamePhase :: VICTORY;
            }

            this->message_hub.popMessage(GAME_CHANNEL);
        }

        if (game_channel_message.subject == "improvement state") {
            std::cout « "Improvement state message received by " « this « std::endl;

            this->__handleImprovementStateMessage(game_channel_message);

            this->message_hub.popMessage(GAME_CHANNEL);
        }
    }

    if (not this->message_hub.isEmpty(GAME_STATE_CHANNEL)) {
        Message game_state_message =
            this->message_hub.receiveMessage(GAME_STATE_CHANNEL);

        if (game_state_message.subject == "turn advance") {
            std::cout « "Turn advance message received by " « this « std::endl;
            this->message_hub.popMessage(GAME_STATE_CHANNEL);
        }
    }

    return;
}   /* __processMessage() */
```

4.5.3.13 __sendCreditsEarnedMessage()

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

Helper method to format and send a credits earned message.

```
{
    Message credits_earned_message;

    credits_earned_message.channel = SETTLEMENT_CHANNEL;
    credits_earned_message.subject = "credits earned";

    this->message_hub.sendMessage(credits_earned_message);

    std::cout « "Credits earned message sent by " « this « std::endl;
    return;
}   /* __sendCreditsEarnedMessage() */
```

4.5.3.14 __sendGameStateMessage()

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

Helper method to format and send a game state message.

```
{
    Message game_state_message;

    game_state_message.channel = GAME_STATE_CHANNEL;
    game_state_message.subject = "game state";

    game_state_message.int_payload["year"] = this->year;
    game_state_message.int_payload["month"] = this->month;
    game_state_message.int_payload["population"] = this->population;
    game_state_message.int_payload["credits"] = this->credits;
    game_state_message.int_payload["demand_MWh"] = this->demand_MWh;
    game_state_message.int_payload["cumulative_emissions_tonnes"] =
        this->cumulative_emissions_tonnes;

    switch (this->game_phase) {
        case (GamePhase :: BUILD_SETTLEMENT): {
            game_state_message.string_payload["game phase"] = "build settlement";

            break;
        }


        case (GamePhase :: SYSTEM_MANAGEMENT): {
            game_state_message.string_payload["game phase"] = "system management";

            break;
        }


        case (GamePhase :: LOSS_EMISSIONS): {
            game_state_message.string_payload["game phase"] = "loss emissions";

            break;
        }


        case (GamePhase :: LOSS_DEMAND): {
            game_state_message.string_payload["game phase"] = "loss demand";

            break;
        }


        case (GamePhase :: LOSS_CREDITS): {
            game_state_message.string_payload["game phase"] = "loss credits";

            break;
        }


        case (GamePhase :: VICTORY): {
            game_state_message.string_payload["game phase"] = "victory";

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    game_state_message.vector_payload["demand_vec_MWh"] = this->demand_vec_MWh;

    this->message_hub.sendMessage(game_state_message);

    std::cout « "Game state message sent by " « this « std::endl;
    return;
}   /* __sendGameStateMessage() */
```

4.5.3.15 __sendTurnAdvanceMessage()

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

Helper method to format and send a turn advance message.

```
{
    Message turn_advance_message;

    turn_advance_message.channel = GAME_STATE_CHANNEL;
    turn_advance_message.subject = "turn advance";

    turn_advance_message.int_payload["credits"] = this->credits;
    turn_advance_message.int_payload["month"] = this->month;
    turn_advance_message.int_payload["demand_MWh"] = this->demand_MWh;

    this->message_hub.sendMessage(turn_advance_message);

    std::cout « "Turn advance message sent by " « this « std::endl;
    return;
}   /* __sendTurnAdvanceMessage() */
```

4.5.3.16 __toggleFrameClockOverlay()

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

Helper method to toggle frame clock overlay.

```
{
    if (this->show_frame_clock_overlay) {
        this->show_frame_clock_overlay = false;
    }

    else {
        this->show_frame_clock_overlay = true;
    }

    return;
}   /* __toggleFrameClockOverlay() */
```

4.5.3.17 run()

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

Method to run game (defines game loop).

Returns

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

```
{
    //  1. play brand animation
    //...

    //  2. show splash screen
    //...

    //  3. start game loop
    while (not this->game_loop_broken) {
        this->time_since_start_s = this->clock.getElapsedTime().asSeconds();

        if (this->time_since_start_s >= (this->frame + 1) * SECONDS_PER_FRAME) {
            //  6.1. process events
            while (this->render_window_ptr->pollEvent(this->event)) {
                this->hex_map_ptr->processEvent();
                this->context_menu_ptr->processEvent();
                this->__processEvent();
            }


            //  6.2. process messages
            while (this->message_hub.hasTraffic()) {
                this->hex_map_ptr->processMessage();
                this->context_menu_ptr->processMessage();
                this->__processMessage();

                this->check_terminating_conditions = true;

                if (not this->message_deadlock) {
                    this->message_deadlock_frame++;

                    if (this->message_deadlock_frame > 5 * FRAMES_PER_SECOND) {
                        this->message_hub.printState();
                        this->message_deadlock = true;
                    }
                }
            }
            this->message_deadlock = false;
            this->message_deadlock_frame = 0;


            //  6.3. check terminating conditions
            if (this->check_terminating_conditions) {
                this->__checkTerminatingConditions();
                this->check_terminating_conditions = false;
            }


            //  6.4. draw frame
            this->render_window_ptr->clear();

            this->hex_map_ptr->draw();
            this->context_menu_ptr->draw();
            this->__draw();

            this->render_window_ptr->display();


            //  6.5. increment frame
            this->frame++;
        }

        // check track status, move to next if stopped
        if (this->assets_manager_ptr->getTrackStatus() == sf::SoundSource::Stopped) {
            this->assets_manager_ptr->nextTrack();
            this->assets_manager_ptr->playTrack();
        }

    }

    return this->quit_game;
}   /* run() */
```

4.5.4 Member Data Documentation

4.5.4.1 assets_manager_ptr

`AssetsManager* Game::assets_manager_ptr [private]`

A pointer to the assets manager.

4.5.4.2 check_terminating_conditions

`bool Game::check_terminating_conditions`

Boolean indicating whether or not to check terminating conditions.

4.5.4.3 clock

`sf::Clock Game::clock`

The game clock.

4.5.4.4 context_menu_ptr

```
ContextMenu* Game::context_menu_ptr
```

Pointer to the context menu.

4.5.4.5 credits

```
int Game::credits
```

Current balance of credits.

4.5.4.6 cumulative_emissions_tonnes

```
int Game::cumulative_emissions_tonnes
```

Cumulative emissions [tonnes] (1 tonne = 1000 kg).

4.5.4.7 demand_MWh

```
int Game::demand_MWh
```

Current energy demand [MWh].

4.5.4.8 demand_remaining_MWh

```
int Game::demand_remaining_MWh
```

The current remaining energy demand [MWh].

4.5.4.9 demand_vec_MWh

```
std::vector<double> Game::demand_vec_MWh
```

A vector of daily demands [MWh] for the current month.

4.5.4.10 event

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

The game events class.

4.5.4.11 frame

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

The current frame of the game.

4.5.4.12 game_loop_broken

```
bool Game::game_loop_broken
```

Boolean indicating whether or not the game loop is broken.

4.5.4.13 game_phase

```
GamePhase Game::game_phase
```

The current phase of the game.

4.5.4.14 hex_map_ptr

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

Pointer to the hex map (defines game world).

4.5.4.15 message_deadlock

```
bool Game::message_deadlock
```

A boolean indicating whether a message deadlock has been detected.

4.5.4.16 message_deadlock_frame

```
int Game::message_deadlock_frame
```

A frame counter for detecting message deadlock.

4.5.4.17 message_hub

```
MessageHub Game::message_hub
```

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

4.5.4.18 month

```
int Game::month
```

Current game month.

4.5.4.19 population

```
int Game::population
```

Current population.

4.5.4.20 quit_game

```
bool Game::quit_game
```

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

4.5.4.21 render_window_ptr

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

A pointer to the render window.

4.5.4.22 show_frame_clock_overlay

```
bool Game::show_frame_clock_overlay
```

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

4.5.4.23 time_since_start_s

```
double Game::time_since_start_s
```

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

4.5.4.24 turn

```
int Game::turn = 0
```

The current game turn.

4.5.4.25 year

```
int Game::year
```

Current game year.

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

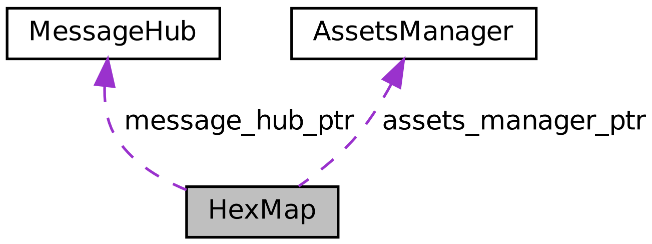
- header/Game.h
- source/Game.cpp

4.6 HexMap Class Reference

A class which defines a hex map of hex tiles.

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

Collaboration diagram for HexMap:



Public Member Functions

- HexMap (int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

  Constructor (intended) for the HexMap class.
- void assess (void)

  Method to assess the resource of the selected tile.
- void reroll (void)

  Method to re-roll the hex map.
- void toggleResourceOverlay (void)

  Method to toggle the hex map resource overlay.
- void processEvent (void)

  Method to process HexMap. To be called once per event.
- void processMessage (void)

  Method to process HexMap. To be called once per message.
- void draw (void)

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

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

  Destructor for the HexMap class.

Public Attributes

- bool show_resource

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

  A boolean which indicates if a tile is currently selected.
- int n_layers

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

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

  The current frame of this object.
- double position_x

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

  The y position of the hex map's origin (i.e. central) tile.
- sf::RectangleShape glass_screen

  To give the effect of an old glass screen over the hex map.
- std::vector< double > tile_position_x_vec

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

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

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

  A position-indexed, nested map of hex tiles.
- std::vector< HexTile ∗ > hex_draw_order_vec

  A vector of hex tiles, in drawing order.

Private Member Functions

- void __setUpGlassScreen (void)

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

  Helper method to lay the hex tiles down to generate the game world.
- void __buildDrawOrderVector (void)

  Helper method to build tile drawing order vector.
- std::vector< double > __getNoise (int, int=128)

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

  Helper method to procedurally generate tile types and set tiles accordingly.
- std::vector< double > __getValidMapIndexPositions (double, double)

  Helper method to translate given position into valid index position for a.
- std::vector< HexTile ∗ > __getNeighboursVector (HexTile ∗)

  Helper method to assemble a vector pointers to all neighbours of the given tile.
- TileType __getMajorityTileType (HexTile ∗)

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

  Helper method to smooth tile types using a majority rules approach.
- bool __isLakeTouchingOcean (HexTile ∗)
- void __enforceOceanContinuity (void)

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

  Helper method to procedurally generate tile resources and set tiles accordingly.
- void __assembleHexMap (void)

  Helper method to assemble the hex map.
- HexTile ∗ __getSelectedTile (void)

  Helper method to get pointer to selected tile.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.
- void __sendNoTileSelectedMessage (void)

  Helper method to format and send message on no tile selected.
- void __assessNeighbours (HexTile ∗)

  Helper method to assess all neighbours of the given tile.

Private Attributes

- sf::Event ∗ event_ptr

  A pointer to the event class.
- sf::RenderWindow ∗ render_window_ptr

  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr

  A pointer to the message hub.

4.6.1 Detailed Description

A class which defines a hex map of hex tiles.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 HexMap()

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

Constructor (intended) for the HexMap class.

Parameters

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

```
{
    //  1. set attributes

    //  1.1. private
    this->event_ptr = event_ptr;
    this->render_window_ptr = render_window_ptr;

    this->assets_manager_ptr = assets_manager_ptr;
    this->message_hub_ptr = message_hub_ptr;

    //  1.2. public
    this->show_resource = false;
    this->tile_selected = false;

    this->frame = 0;

    this->n_layers = n_layers;
    if (this->n_layers < 0) {
        this->n_layers = 0;
    }

    this->position_x = 400;
    this->position_y = 400;

    //  2. assemble n layer hex map
    this->__assembleHexMap();

    //  3. set up and position drawable attributes
    this->__setUpGlassScreen();

    //  4. add message channel(s)
    this->message_hub_ptr->addChannel(TILE_SELECTED_CHANNEL);
    this->message_hub_ptr->addChannel(NO_TILE_SELECTED_CHANNEL);
    this->message_hub_ptr->addChannel(TILE_STATE_CHANNEL);
    this->message_hub_ptr->addChannel(HEX_MAP_CHANNEL);

    std::cout « "HexMap constructed at " « this « std::endl;

    return;
}   /* HexMap(), intended */
```

4.6.2.2 ∼HexMap()

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

Destructor for the HexMap class.

```
{
    this->clear();

    std::cout « "HexMap at " « this « " destroyed" « std::endl;

    return;
}   /* ~HexMap() */
```

4.6.3 Member Function Documentation

4.6.3.1 __assembleHexMap()

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

Helper method to assemble the hex map.

```
{
    //  1. seed RNG (using milliseconds since 1 Jan 1970)
    unsigned long long int milliseconds_since_epoch =
        std::chrono::duration_cast<std::chrono::milliseconds>(
            std::chrono::system_clock::now().time_since_epoch()
        ).count();
    srand(milliseconds_since_epoch);

    //  2. lay tiles
    this->__layTiles();
    this->__buildDrawOrderVector();

    //  3. procedurally generate types
    this->__procedurallyGenerateTileTypes();

    //  4. procedurally generate resources
    this->__procedurallyGenerateTileResources();

    return;
}   /* __assembleHexMap() */
```

4.6.3.2 __assessNeighbours()

```
void HexMap::__assessNeighbours (
               HexTile * hex_ptr )  [private]
```

Helper method to assess all neighbours of the given tile.

Parameters

Pointer	to the tile whose neighbours are to be assessed.

```
{
    std::vector<HexTile*> neighbours_vec = this->__getNeighboursVector(hex_ptr);

    for (size_t i = 0; i < neighbours_vec.size(); i++) {
        neighbours_vec[i]->assess();
    }

    return;
}   /* __assessNeighbours() */
```

4.6.3.3 __buildDrawOrderVector()

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

Helper method to build tile drawing order vector.

```
{
    //  1. build temp list of tiles
    std::list<HexTile*> temp_list;

    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            temp_list.push_back(hex_map_iter_y->second);
        }
    }

    //  2. move elements from temp list to drawing order vector
    double min_position_y = 0;
    std::list<HexTile*>::iterator list_iter;

    while (not temp_list.empty()) {
        //  2.1. determine min y position
        min_position_y = std::numeric_limits<double>::infinity();

        for (
            list_iter = temp_list.begin();
            list_iter != temp_list.end();
            list_iter++
        ) {
            if ((*list_iter)->position_y < min_position_y) {
                min_position_y = (*list_iter)->position_y;
            }
        }

        //  2.2 move min y list elements to drawing order vec
        list_iter = temp_list.begin();
        while (list_iter != temp_list.end()) {
            if ((*list_iter)->position_y == min_position_y) {
                this->hex_draw_order_vec.push_back((*list_iter));
                list_iter = temp_list.erase(list_iter);
            }

            else {
                list_iter++;
            }
        }
    }

    return;
}   /* __buildDrawOrderVector() */
```

4.6.3.4 __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.

```
{
    std::cout « "enforcing ocean continuity ..." « std::endl;

    bool tile_changed = false;

    //  1. scan tiles and enforce (where appropriate)
    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    HexTile* hex_ptr;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_ptr = hex_map_iter_y->second;

            if (this->__isLakeTouchingOcean(hex_ptr)) {
                hex_ptr->setTileType(TileType :: OCEAN);
                tile_changed = true;
            }
        }
    }

    if (tile_changed) {
        this->__enforceOceanContinuity();
    }
    else {
        return;
    }
}   /* __enforceOceanContinuity() */
```

4.6.3.5 __getMajorityTileType()

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

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

Parameters

hex_ptr	Pointer to the given tile.

Returns

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

```
{
    //  1. init type count map
    std::map<TileType, int> type_count_map;
    type_count_map[hex_ptr->tile_type] = 1;

    //  2. survey neighbours, count type instances
    std::vector<HexTile*> neighbours_vec = this->__getNeighboursVector(hex_ptr);

    for (size_t i = 0; i < neighbours_vec.size(); i++) {
        if (type_count_map.count(neighbours_vec[i]->tile_type) <= 0) {
            type_count_map[neighbours_vec[i]->tile_type] = 1;
        }
        else {
            type_count_map[neighbours_vec[i]->tile_type] += 1;
        }
    }

    //  3. find majority tile type
    int max_count = -1 * std::numeric_limits<int>::infinity();
    TileType majority_tile_type = hex_ptr->tile_type;

    std::map<TileType, int>::iterator map_iter;
    for (
        map_iter = type_count_map.begin();
        map_iter != type_count_map.end();
        map_iter++
    ){
        if (map_iter->second > max_count) {
            max_count = map_iter->second;
            majority_tile_type = map_iter->first;
        }
    }

    //  4. detect ties
    for (
        map_iter = type_count_map.begin();
        map_iter != type_count_map.end();
        map_iter++
    ){
        if (
            map_iter->second == max_count and
            map_iter->first != majority_tile_type
        ) {
            majority_tile_type = hex_ptr->tile_type;
            break;
        }
    }

    return majority_tile_type;
}   /* __getMajorityTileType() */
```

4.6.3.6 __getNeighboursVector()

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

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

Parameters

hex_ptr	A pointer to the given tile.

Returns

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

```
{
    std::vector<HexTile*> neighbours_vec;

    //  1. build potential neighbour positions
    std::vector<double> potential_neighbour_x_vec(6, 0);
    std::vector<double> potential_neighbour_y_vec(6, 0);

    for (int i = 0; i < 6; i++) {
        potential_neighbour_x_vec[i] = hex_ptr->position_x +
            2 * hex_ptr->minor_radius * cos((60 * i) * (M_PI / 180));

        potential_neighbour_y_vec[i] = hex_ptr->position_y +
            2 * hex_ptr->minor_radius * sin((60 * i) * (M_PI / 180));
    }

    //  2. populate neighbours vector
    std::vector<double> map_index_positions;
    double potential_x = 0;
    double potential_y = 0;

    for (int i = 0; i < 6; i++) {
        potential_x = potential_neighbour_x_vec[i];
        potential_y = potential_neighbour_y_vec[i];

        map_index_positions = this->__getValidMapIndexPositions(
            potential_x,
            potential_y
        );

        if (not (map_index_positions[0] == -1)) {
            neighbours_vec.push_back(
                this->hex_map[map_index_positions[0]][map_index_positions[1]]
            );
        }
    }

    return neighbours_vec;
}   /* __getNeighbourVector() */
```

4.6.3.7 __getNoise()

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

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

Parameters

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

Returns

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

```
{
    //  1. generate random amplitude, wave number, direction, and phase vectors
    std::vector<double> random_amplitude_vec(n_components, 0);
    std::vector<double> random_wave_number_vec(n_components, 0);
    std::vector<double> random_frequency_vec(n_components, 0);
    std::vector<double> random_direction_vec(n_components, 0);
    std::vector<double> random_phase_vec(n_components, 0);

    for (int i = 0; i < n_components; i++) {
        random_amplitude_vec[i] = 10 * ((double)rand() / RAND_MAX);

        random_wave_number_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);

        random_frequency_vec[i] = ((double)rand() / RAND_MAX);

        random_direction_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);

        random_phase_vec[i] = 2 * M_PI * ((double)rand() / RAND_MAX);
    }

    //  2. generate noise vec
    double amp = 0;
    double wave_no = 0;
    double freq = 0;
    double dir = 0;
    double phase = 0;

    double x = 0;
    double y = 0;
    double t = time(NULL);

    double max_noise = -1 * std::numeric_limits<double>::infinity();
    double min_noise = std::numeric_limits<double>::infinity();

    double noise = 0;
    std::vector<double> noise_vec(n_elements, 0);

    for (int i = 0; i < n_elements; i++) {
        x = this->tile_position_x_vec[i] - this->position_x;
        y = this->tile_position_y_vec[i] - this->position_y;

        for (int j = 0; j < n_components; j++) {
            amp = random_amplitude_vec[j];
            wave_no = random_wave_number_vec[j];
            freq = random_frequency_vec[j];
            dir = random_direction_vec[j];
            phase = random_phase_vec[j];

            noise += (amp / (j + 1)) * cos(
                wave_no * (j + 1) * (x * sin(dir) + y * cos(dir)) +
                2 * M_PI * (j + 1) * freq * t +
                phase
            );
        }

        noise_vec[i] = noise;

        if (noise > max_noise) {
            max_noise = noise;
        }

        else if (noise < min_noise) {
            min_noise = noise;
        }

        noise = 0;
    }

    //  3. normalize noise vec
    for (int i = 0; i < n_elements; i++) {
        noise_vec[i] = (noise_vec[i] - min_noise) / (max_noise - min_noise);

        if (noise_vec[i] < 0) {
            noise_vec[i] = 0;
        }
        else if (noise_vec[i] > 1) {
            noise_vec[i] = 1;
        }
    }

    return noise_vec;
}   /* __getNoise() */
```

4.6.3.8 __getSelectedTile()

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

Helper method to get pointer to selected tile.

Returns

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

```
{
    HexTile* selected_tile_ptr = NULL;

    bool break_flag = false;
    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;

    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            if (hex_map_iter_y->second->is_selected) {
                selected_tile_ptr = hex_map_iter_y->second;
                break_flag = true;
            }

            if (break_flag) {
                break;
            }
        }

        if (break_flag) {
            break;
        }
    }

    return selected_tile_ptr;
}   /* __getSelectedTile() */
```

4.6.3.9 __getValidMapIndexPositions()

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

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

Parameters

potential←_x	The potential x position of the tile.
potential←_y	The potential y position of the tile.

Returns

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

```
{
    std::vector<double> map_index_positions = {-1, -1};

    std::map<double, std::map<double, HexTile*>>::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    HexTile* hex_ptr;

    double distance = 0;

    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_ptr = hex_map_iter_y->second;

            distance = sqrt(
                pow(hex_ptr->position_x - potential_x, 2) +
                pow(hex_ptr->position_y - potential_y, 2)
            );

            if (distance <= hex_ptr->minor_radius / 4) {
                map_index_positions = {hex_ptr->position_x, hex_ptr->position_y};
                return map_index_positions;
            }
        }
    }

    return map_index_positions;
}   /* __isInHexMap() */
```

4.6.3.10 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::Escape): {
            this->tile_selected = false;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.6.3.11 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            HexTile* hex_ptr = this->__getSelectedTile();

            if (hex_ptr != NULL) {
                this->tile_selected = true;
            }

            else if (this->tile_selected) {
                this->tile_selected = false;
                this->__sendNoTileSelectedMessage();
            }

            break;
        }


        case (sf::Mouse::Right): {
            if (this->tile_selected) {
                this->tile_selected = false;
                this->__sendNoTileSelectedMessage();
            }

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.6.3.12 __isLakeTouchingOcean()

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

```
{
    //  1. if not lake tile, return
    if (not (hex_ptr->tile_type == TileType :: LAKE)) {
        return false;
    }

    //  2. scan neighbours for ocean tiles
    std::vector<HexTile*> neighbours_vec = this->__getNeighboursVector(hex_ptr);

    for (size_t i = 0; i < neighbours_vec.size(); i++) {
        if (neighbours_vec[i]->tile_type == TileType :: OCEAN) {
            return true;
        }
    }

    return false;
}   /* __isLakeTouchingOcean() */
```

4.6.3.13 __layTiles()

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

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

```
{
    this->n_tiles = 0;

    //  1. add origin tile
    HexTile* hex_ptr = new HexTile(
        this->position_x,
        this->position_y,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
    this->tile_position_x_vec.push_back(hex_ptr->position_x);
    this->tile_position_y_vec.push_back(hex_ptr->position_y);
    this->n_tiles++;


    //  2. fill out first row (reflect across origin tile)
    for (int i = 0; i < this->n_layers; i++) {
        hex_ptr = new HexTile(
            this->position_x + 2 * (i + 1) * hex_ptr->minor_radius,
            this->position_y,
            this->event_ptr,
            this->render_window_ptr,
            this->assets_manager_ptr,
            this->message_hub_ptr
        );

        this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
        this->tile_position_x_vec.push_back(hex_ptr->position_x);
        this->tile_position_y_vec.push_back(hex_ptr->position_y);
        this->n_tiles++;

        if (i == this->n_layers - 1) {
            this->border_tiles_vec.push_back(hex_ptr);
        }

        hex_ptr = new HexTile(
            this->position_x - 2 * (i + 1) * hex_ptr->minor_radius,
            this->position_y,
            this->event_ptr,
            this->render_window_ptr,
            this->assets_manager_ptr,
            this->message_hub_ptr
        );

        this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
        this->tile_position_x_vec.push_back(hex_ptr->position_x);
        this->tile_position_y_vec.push_back(hex_ptr->position_y);
        this->n_tiles++;

        if (i == this->n_layers - 1) {
            this->border_tiles_vec.push_back(hex_ptr);
        }
    }


    //  3. fill out subsequent rows (reflect across first row)
    HexTile* first_row_left_tile = hex_ptr;

    int offset_count = 1;

    double x_offset = 0;
    double y_offset = 0;

    for (
        int row_width = 2 * this->n_layers;
        row_width > this->n_layers;
        row_width--
    ) {
        //  3.1. upper row
        x_offset = first_row_left_tile->position_x +
            2 * offset_count * first_row_left_tile->minor_radius *
            cos(60 * (M_PI / 180));

        y_offset = first_row_left_tile->position_y -
            2 * offset_count * first_row_left_tile->minor_radius *
            sin(60 * (M_PI / 180));

        hex_ptr = new HexTile(
            x_offset,
            y_offset,
            this->event_ptr,
            this->render_window_ptr,
            this->assets_manager_ptr,
            this->message_hub_ptr
        );

        this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
        this->tile_position_x_vec.push_back(hex_ptr->position_x);
        this->tile_position_y_vec.push_back(hex_ptr->position_y);
        this->n_tiles++;

        this->border_tiles_vec.push_back(hex_ptr);

        for (int i = 1; i < row_width; i++) {
            x_offset += 2 * first_row_left_tile->minor_radius;

            hex_ptr = new HexTile(
                x_offset,
                y_offset,
                this->event_ptr,
                this->render_window_ptr,
                this->assets_manager_ptr,
                this->message_hub_ptr
            );

            this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
            this->tile_position_x_vec.push_back(hex_ptr->position_x);
            this->tile_position_y_vec.push_back(hex_ptr->position_y);
            this->n_tiles++;

            if (row_width == this->n_layers + 1 or i == row_width - 1) {
                this->border_tiles_vec.push_back(hex_ptr);
            }
        }

        //  3.2. lower row
        x_offset = first_row_left_tile->position_x +
            2 * offset_count * first_row_left_tile->minor_radius *
            cos(60 * (M_PI / 180));

        y_offset = first_row_left_tile->position_y +
            2 * offset_count * first_row_left_tile->minor_radius *
            sin(60 * (M_PI / 180));

        hex_ptr = new HexTile(
            x_offset,
            y_offset,
            this->event_ptr,
            this->render_window_ptr,
            this->assets_manager_ptr,
            this->message_hub_ptr
        );

        this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
        this->tile_position_x_vec.push_back(hex_ptr->position_x);
        this->tile_position_y_vec.push_back(hex_ptr->position_y);
        this->n_tiles++;

        this->border_tiles_vec.push_back(hex_ptr);

        for (int i = 1; i < row_width; i++) {
            x_offset += 2 * first_row_left_tile->minor_radius;

            hex_ptr = new HexTile(
                x_offset,
                y_offset,
                this->event_ptr,
                this->render_window_ptr,
                this->assets_manager_ptr,
                this->message_hub_ptr
            );

            this->hex_map[hex_ptr->position_x][hex_ptr->position_y] = hex_ptr;
            this->tile_position_x_vec.push_back(hex_ptr->position_x);
            this->tile_position_y_vec.push_back(hex_ptr->position_y);
            this->n_tiles++;

            if (row_width == this->n_layers + 1 or i == row_width - 1) {
                this->border_tiles_vec.push_back(hex_ptr);
            }
        }

        offset_count++;
    }

    return;
}   /* __layTiles() */
```

4.6.3.14 __procedurallyGenerateTileResources()

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

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

```
{
    //  1. get random cosine series noise vec
    std::vector<double> noise_vec = this->__getNoise(this->n_tiles);

    //  2. set tile resources based on random cosine series noise
    int noise_idx = 0;

    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_map_iter_y->second->setTileResource(noise_vec[noise_idx]);
            noise_idx++;
        }
    }

    return;
}   /* __procedurallyGenerateTileResources() */
```

4.6.3.15 __procedurallyGenerateTileTypes()

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

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

```
{
    //  1. get random cosine series noise vec
    std::vector<double> noise_vec = this->__getNoise(this->n_tiles);

    //  2. set initial tile types based on either random cosine series noise or white
    //     noise (decided by coin toss)
    int noise_idx = 0;

    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            if ((double)rand() / RAND_MAX > 0.5) {
                hex_map_iter_y->second->setTileType(noise_vec[noise_idx]);
            }
            else {
                hex_map_iter_y->second->setTileType((double)rand() / RAND_MAX);
            }
            noise_idx++;
        }
    }

    //  3. smooth tile types (majority rules)
    this->__smoothTileTypes();

    //  4. set border tile type to ocean
    for (size_t i = 0; i < this->border_tiles_vec.size(); i++) {
        this->border_tiles_vec[i]->setTileType(TileType :: OCEAN);
    }

    //  5. enforce ocean continuity (i.e. all lake tiles touching ocean become ocean)
    this->__enforceOceanContinuity();

    //  6. decorate tiles
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_map_iter_y->second->decorateTile();
        }
    }

    return;
}   /* __procedurallyGenerateTileTypes() */
```

4.6.3.16 __sendNoTileSelectedMessage()

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

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

```
{
    Message no_tile_selected_message;

    no_tile_selected_message.channel = NO_TILE_SELECTED_CHANNEL;
    no_tile_selected_message.subject = "no tile selected";

    this->message_hub_ptr->sendMessage(no_tile_selected_message);

    std::cout « "No tile selected message sent by " « this « std::endl;
    return;
}   /* __sendNoTileSelectedMessage() */
```

4.6.3.17 __setUpGlassScreen()

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

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

```
{
    this->glass_screen.setSize(sf::Vector2f(GAME_WIDTH, GAME_HEIGHT));
    this->glass_screen.setFillColor(sf::Color(MONOCHROME_SCREEN_BACKGROUND));

    return;
}   /* __setUpGlassScreen() */
```

4.6.3.18 __smoothTileTypes()

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

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

```
{
    std::cout « "smoothing ..." « std::endl;

    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    HexTile* hex_ptr;
    TileType majority_tile_type;

    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_ptr = hex_map_iter_y->second;
            majority_tile_type = this->__getMajorityTileType(hex_ptr);

            if (majority_tile_type != hex_ptr->tile_type) {
                hex_ptr->setTileType(majority_tile_type);
            }
        }
    }

    return;
}   /* __smoothTileTypes() */
```

4.6.3.19 assess()

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

Method to assess the resource of the selected tile.

```
{
    HexTile* selected_tile_ptr = this->__getSelectedTile();
    if (selected_tile_ptr != NULL) {
        selected_tile_ptr->assess();
    }

    return;
}   /* assess() */
```

4.6.3.20 clear()

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

Method to clear the hex map.

```
{
    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            delete hex_map_iter_y->second;
        }
    }
    this->hex_map.clear();

    this->tile_position_x_vec.clear();
    this->tile_position_y_vec.clear();
    this->border_tiles_vec.clear();

    return;
}   /* clear() */
```

4.6.3.21 draw()

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

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

```
{
    //  1. draw background
    sf::Color glass_screen_colour = this->glass_screen.getFillColor();
    glass_screen_colour.a = 255;
    this->glass_screen.setFillColor(glass_screen_colour);

    this->render_window_ptr->draw(this->glass_screen);

    //  2. draw tiles (other than the selected tile) in drawing order
    for (size_t i = 0; i < this->hex_draw_order_vec.size(); i++) {
        if (not this->hex_draw_order_vec[i]->is_selected) {
            this->hex_draw_order_vec[i]->draw();
        }
    }

    //  3. draw selected tile
    HexTile* selected_tile_ptr = this->__getSelectedTile();
    if (selected_tile_ptr != NULL) {
        selected_tile_ptr->draw();
    }

    //  4. draw resource overlay text indication
    if (this->show_resource) {
        sf::Text resource_overlay_text(
            "**** RENEWABLE RESOURCE OVERLAY ****",
            *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
            16
        );

        resource_overlay_text.setPosition(
            (800 - resource_overlay_text.getLocalBounds().width) / 2,
            GAME_HEIGHT - 70
        );

        resource_overlay_text.setFillColor(MONOCHROME_TEXT_GREEN);

        this->render_window_ptr->draw(resource_overlay_text);
    }

    //  5. draw glass screen
    glass_screen_colour = this->glass_screen.getFillColor();
    glass_screen_colour.a = 40;
    this->glass_screen.setFillColor(glass_screen_colour);

    this->render_window_ptr->draw(this->glass_screen);

    this->frame++;
    return;
}   /* draw() */
```

4.6.3.22 processEvent()

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

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

```
{
    //  1. process HexTile events
    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_map_iter_y->second->processEvent();
        }
    }

    //  2. process HexMap events
    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.6.3.23 processMessage()

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

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

```
{
    //  1. process HexTile messages
    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_map_iter_y->second->processMessage();
        }
    }

    //  2. process HexMap messages
    if (not this->message_hub_ptr->isEmpty(HEX_MAP_CHANNEL)) {
        Message hex_map_message = this->message_hub_ptr->receiveMessage(
            HEX_MAP_CHANNEL
        );

        if (hex_map_message.subject == "assess neighbours") {
            HexTile* hex_ptr = this->__getSelectedTile();
            this->__assessNeighbours(hex_ptr);

            std::cout « "Assess neighbours message received by " « this « std::endl;
            this->message_hub_ptr->popMessage(HEX_MAP_CHANNEL);
        }
    }

    return;
}   /* processMessage() */
```

4.6.3.24 reroll()

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

Method to re-roll the hex map.

```
{
    this->clear();
    this->__assembleHexMap();

    return;
}   /* reroll() */
```

4.6.3.25 toggleResourceOverlay()

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

Method to toggle the hex map resource overlay.

```
{
    std::map<double, std::map<double, HexTile*»::iterator hex_map_iter_x;
    std::map<double, HexTile*>::iterator hex_map_iter_y;
    for (
        hex_map_iter_x = this->hex_map.begin();
        hex_map_iter_x != this->hex_map.end();
        hex_map_iter_x++
    ) {
        for (
            hex_map_iter_y = hex_map_iter_x->second.begin();
            hex_map_iter_y != hex_map_iter_x->second.end();
            hex_map_iter_y++
        ) {
            hex_map_iter_y->second->toggleResourceOverlay();
        }
    }

    if (this->show_resource) {
        this->show_resource = false;
        this->assets_manager_ptr->getSound("resource overlay toggle off")->play();
    }

    else {
        this->show_resource = true;
        this->assets_manager_ptr->getSound("resource overlay toggle on")->play();
    }

    return;
}   /* toggleResourceOverlay() */
```

4.6.4 Member Data Documentation

4.6.4.1 assets_manager_ptr

`AssetsManager* HexMap::assets_manager_ptr [private]`

A pointer to the assets manager.

4.6.4.2 border_tiles_vec

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

A vector of pointers to the border tiles.

4.6.4.3 event_ptr

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

A pointer to the event class.

4.6.4.4 frame

`unsigned long long int HexMap::frame`

The current frame of this object.

4.6.4.5 glass_screen

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

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

4.6.4.6 hex_draw_order_vec

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

A vector of hex tiles, in drawing order.

4.6.4.7 hex_map

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

A position-indexed, nested map of hex tiles.

4.6.4.8 message_hub_ptr

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

A pointer to the message hub.

4.6.4.9 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

4.6.4.10 n_tiles

```
int HexMap::n_tiles
```

The number of tiles in the hex map.

4.6.4.11 position_x

```
double HexMap::position_x
```

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

4.6.4.12 position_y

```
double HexMap::position_y
```

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

4.6.4.13 render_window_ptr

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

A pointer to the render window.

4.6.4.14 show_resource

```
bool HexMap::show_resource
```

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

4.6.4.15 tile_position_x_vec

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

A vector of tile x positions.

4.6.4.16 tile_position_y_vec

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

A vector of tile y position.

4.6.4.17 tile_selected

```
bool HexMap::tile_selected
```

A boolean which indicates if a tile is currently selected.

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

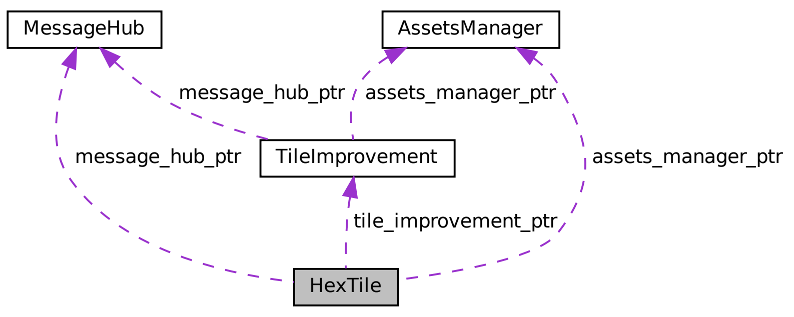
- header/HexMap.h
- source/HexMap.cpp

4.7 HexTile Class Reference

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

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

Collaboration diagram for HexTile:



Public Member Functions

- HexTile (double, double, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

    Constructor for the HexTile class.
- void setTileType (TileType)

    Method to set the tile type (by enum value).
- void setTileType (double)

    Method to set the tile type (by numeric input).
- void setTileResource (TileResource)

    Method to set the tile resource (by enum value).
- void setTileResource (double)

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

    Method to decorate tile.
- void toggleResourceOverlay (void)

    Method to toggle the tile resource overlay.

- void assess (void)

    Method to assess the tile's resource.
- void processEvent (void)

    Method to process HexTile. To be called once per event.
- void processMessage (void)

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

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

    Destructor for the HexTile class.

Public Attributes

- TileType tile_type

    The terrain type of the tile.
- TileResource tile_resource

    The renewable resource quality of the tile.
- bool show_node

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

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

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

    A boolean which triggers a resource assessment notification.
- bool is_selected

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

    A boolean which indicates whether or not to draw a tile explosion.
- bool decoration_cleared

    A boolean which indicates if the tile decoration has been cleared.
- bool has_improvement

    A boolean which indicates if tile has improvement or not.
- TileImprovement ∗ tile_improvement_ptr

    A pointer to the improvement for this tile.
- bool build_menu_open

    A boolean which indicates if the tile build menu is open.
- size_t explosion_frame

    The current frame of the explosion animation.
- unsigned long long int frame

    The current frame of this object.
- int credits

    The current balance of credits.
- int scrap_improvement_frame

    A frame for key-hold to confirm scrapping.
- 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.
- std::string game_phase

  The current phase of the game.
- sf::CircleShape node_sprite

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

  A convex shape which represents the tile.
- sf::ConvexShape select_outline_sprite

  A convex shape which outlines the tile when selected.
- sf::CircleShape resource_chip_sprite

  A circle shape which represents a resource chip.
- sf::Text resource_text

  A text representation of the resource.
- sf::Sprite tile_decoration_sprite

  A tile decoration sprite.
- sf::Sprite magnifying_glass_sprite

  A magnifying glass sprite.
- std::vector< sf::Sprite > explosion_sprite_reel

  A reel of sprites for a tile explosion animation.
- sf::RectangleShape build_menu_backing

  A backing for the tile build menu.
- sf::Text build_menu_backing_text

  A text label for the build menu.
- std::vector< std::vector< sf::Sprite > > build_menu_options_vec

  A vector of sprites for illustrating the tile build options.
- std::vector< sf::Text > build_menu_options_text_vec

  A vector of text for the tile build options.

Private Member Functions

- void __setUpNodeSprite (void)

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

  Helper method to set up tile sprite.
- void __setUpSelectOutlineSprite (void)

  Helper method to set up select outline sprite.
- void __setUpResourceChipSprite (void)

  Helper method to set up resource chip sprite.
- void __setResourceText (void)

  Helper method to set up resource text.
- void __setUpMagnifyingGlassSprite (void)

  Helper method to set up and position magnifying glass sprite.
- void __setUpTileExplosionReel (void)

  Helper method to set up tile explosion sprite reel.
- void __setUpBuildOption (std::string, std::string)

  Helper method to set up and postion the sprite and text for a build option.
- void __setUpDieselGeneratorBuildOption (void)

  Helper method to set up and position the diesel generator build option.

- void __setUpWindTurbineBuildOption (bool=false, bool=false)

  Helper method to set up and position the wind turbine build option.
- void __setUpSolarPVBuildOption (bool=false)

  Helper method to set up and position the solar PV array build option.
- void __setUpTidalTurbineBuildOption (void)

  Helper method to set up and position the tidal turbine build option.
- void __setUpWaveEnergyConverterBuildOption (void)

  Helper method to set up and position the wave energy converter build option.
- void __setUpEnergyStorageSystemBuildOption (void)

  Helper method to set up and position the wave energy converter build option.
- void __setUpBuildMenu (void)

  Helper method to set up and place build menu assets (drawable).
- void __setIsSelected (bool)

  Helper method to set the is selected attribute (of tile and improvement).
- void __clearDecoration (void)

  Helper method to clear tile decoration.
- bool __isClicked (void)

  Helper method to determine if tile was clicked on.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleKeyReleaseEvents (void)
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.
- void __openBuildMenu (void)

  Helper method to open the tile improvement build menu.
- void __closeBuildMenu (void)

  Helper method to close the tile improvement build menu.
- void __buildSettlement (void)

  Helper method to build a settlement on this tile.
- void __buildDieselGenerator (void)

  Helper method to build a diesel generator on this tile.
- void __buildSolarPV (void)

  Helper method to build a solar PV array on this tile.
- void __buildWindTurbine (void)

  Helper method to build a wind turbine on this tile.
- void __buildTidalTurbine (void)

  Helper method to build a tidal turbine on this tile.
- void __buildWaveEnergyConverter (void)

  Helper method to build a wave energy converter on this tile.
- void __buildEnergyStorage (void)

  Helper method to build an energy storage system on this tile.
- void __scrapImprovement (void)

  Helper method to scrap the tile improvement (Settlement cannot be scrapped). Requires the mapped key to be held continuously to confirm.
- void __sendTileSelectedMessage (void)

  Helper method to format and send message on tile selection.
- std::string __getTileCoordsSubstring (void)

  Helper method to assemble and return tile coordinates substring.
- std::string __getTileTypeSubstring (void)

  Helper method to assemble and return tile type substring.
- std::string __getTileResourceSubstring (void)

Helper method to assemble and return tile resource substring.
- std::string __getTileImprovementSubstring (void)

  Helper method to assemble and return the tile improvement substring.
- std::string __getTileOptionsSubstring (void)

  Helper method to assemble and return tile options substring.
- void __sendTileStateMessage (void)

  Helper method to format and send tile state message.
- void __sendAssessNeighboursMessage (void)

  Helper method to format and send assess neighbours message.
- void __sendGameStateRequest (void)

  Helper method to format and send a game state request (message).
- void __sendUpdateGamePhaseMessage (std::string)

  Helper method to format and send update game phase message.
- void __sendCreditsSpentMessage (int)

  Helper method to format and send a credits spent message.
- void __sendInsufficientCreditsMessage (void)

  Helper method to format and send an insufficient credits message.

Private Attributes

- sf::Event ∗ event_ptr

  A pointer to the event class.
- sf::RenderWindow ∗ render_window_ptr

  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr

  A pointer to the message hub.

4.7.1 Detailed Description

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

4.7.2 Constructor & Destructor Documentation

4.7.2.1 HexTile()

```
HexTile::HexTile (
            double position_x,
            double position_y,
            sf::Event * event_ptr,
            sf::RenderWindow * render_window_ptr,
            AssetsManager * assets_manager_ptr,
            MessageHub * message_hub_ptr )
```

Constructor for the HexTile class.

Ref: Wikipedia [2023]

Parameters

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

```
{
    //  1. set attributes

    //  1.1. private
    this->event_ptr = event_ptr;
    this->render_window_ptr = render_window_ptr;

    this->assets_manager_ptr = assets_manager_ptr;
    this->message_hub_ptr = message_hub_ptr;

    //  1.2. public
    this->show_node = false;
    this->show_resource = false;
    this->resource_assessed = false;
    this->resource_assessment = false;
    this->is_selected = false;
    this->draw_explosion = false;

    this->decoration_cleared = false;
    this->has_improvement = false;
    this->tile_improvement_ptr = NULL;

    this->build_menu_open = false;

    this->explosion_frame = 0;

    this->frame = 0;
    this->credits = 0;

    this->scrap_improvement_frame = 0;

    this->position_x = position_x;
    this->position_y = position_y;

    this->major_radius = 32;
    this->minor_radius = (sqrt(3) / 2) * this->major_radius;

    this->game_phase = "build settlement";

    //  2. set up and position drawable attributes
    this->__setUpNodeSprite();
    this->__setUpTileSprite();
    this->__setUpSelectOutlineSprite();
    this->__setUpResourceChipSprite();
    this->__setResourceText();
    this->__setUpMagnifyingGlassSprite();
    this->__setUpTileExplosionReel();

    //  3. set tile type and resource (default to none type and average)
    this->setTileType(TileType :: NONE_TYPE);
    this->setTileResource(TileResource :: AVERAGE);

    std::cout « "HexTile constructed at " « this « std::endl;

    return;
}   /* HexTile() */
```

4.7.2.2 ∼HexTile()

```
HexTile::∼HexTile (
                void )
```

Destructor for the HexTile class.

```
{
    if (this->tile_improvement_ptr != NULL) {
        delete this->tile_improvement_ptr;
    }

    std::cout « "HexTile at " « this « " destroyed" « std::endl;

    return;
}   /* ~HexTile() */
```

4.7.3 Member Function Documentation

4.7.3.1 __buildDieselGenerator()

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

Helper method to build a diesel generator on this tile.

```
{
    int build_cost = DIESEL_GENERATOR_BUILD_COST;

    if (this->credits < build_cost) {
        std::cout « "Cannot build diesel generator: insufficient credits (need "
            « build_cost « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->tile_improvement_ptr = new DieselGenerator(
        this->position_x,
        this->position_y,
        this->tile_resource,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->has_improvement = true;
    this->__closeBuildMenu();

    this->__sendCreditsSpentMessage(build_cost);
    this->__sendTileStateMessage();
    this->__sendGameStateRequest();

    return;
}   /* __buildDieselGenerator() */
```

4.7.3.2 __buildEnergyStorage()

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

Helper method to build an energy storage system on this tile.

```
{
    /*
    int build_cost = ENERGY_STORAGE_SYSTEM_BUILD_COST;

    if (this->credits < build_cost) {
        std::cout « "Cannot build energy storage system: insufficient credits (need "
            « build_cost « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->tile_improvement_ptr = new EnergyStorageSystem(
        this->position_x,
        this->position_y,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->has_improvement = true;
    this->__closeBuildMenu();

    this->__sendCreditsSpentMessage(build_cost);
    this->__sendTileStateMessage();
    this->__sendGameStateRequest();
    */
    return;
}   /* __buildEnergyStorage() */
```

4.7.3.3 __buildSettlement()

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

Helper method to build a settlement on this tile.

```
{
    if (this->credits < BUILD_SETTLEMENT_COST) {
        std::cout « "Cannot build settlement: insufficient credits (need "
            « BUILD_SETTLEMENT_COST « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->__clearDecoration();

    this->tile_improvement_ptr = new Settlement(
        this->position_x,
        this->position_y,
        this->tile_resource,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->has_improvement = true;

    this->assess();
    this->__sendAssessNeighboursMessage();

    this->__sendUpdateGamePhaseMessage("system management");
    this->__sendCreditsSpentMessage(BUILD_SETTLEMENT_COST);
    this->__sendTileStateMessage();
    this->__sendGameStateRequest();

    return;
}   /* __buildSettlement() */
```

4.7.3.4 __buildSolarPV()

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

Helper method to build a solar PV array on this tile.

```
{
    int build_cost = SOLAR_PV_BUILD_COST;

    if (this->tile_type == TileType :: LAKE) {
        build_cost *= SOLAR_PV_WATER_BUILD_MULTIPLIER;
    }

    if (this->credits < build_cost) {
        std::cout « "Cannot build solar PV array: insufficient credits (need "
            « build_cost « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->tile_improvement_ptr = new SolarPV(
        this->position_x,
        this->position_y,
        this->tile_resource,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->has_improvement = true;
    this->__closeBuildMenu();

    if (this->tile_type == TileType :: LAKE) {
        this->decoration_cleared = true;
        this->assets_manager_ptr->getSound("splash")->play();
    }

    this->__sendCreditsSpentMessage(build_cost);
    this->__sendTileStateMessage();
    this->__sendGameStateRequest();

    return;
}   /* __buildSolarPV() */
```

4.7.3.5 __buildTidalTurbine()

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

Helper method to build a tidal turbine on this tile.

```
{
    int build_cost = TIDAL_TURBINE_BUILD_COST;

    if (this->credits < build_cost) {
        std::cout « "Cannot build tidal turbine: insufficient credits (need "
            « build_cost « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->tile_improvement_ptr = new TidalTurbine(
        this->position_x,
        this->position_y,
        this->tile_resource,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->has_improvement = true;
    this->decoration_cleared = true;
    this->assets_manager_ptr->getSound("splash")->play();
    this->__closeBuildMenu();

    this->__sendCreditsSpentMessage(build_cost);
    this->__sendTileStateMessage();
    this->__sendGameStateRequest();

    return;
}   /* __buildTidalTurbine() */
```

4.7.3.6 __buildWaveEnergyConverter()

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

Helper method to build a wave energy converter on this tile.

```
{
    int build_cost = WAVE_ENERGY_CONVERTER_BUILD_COST;

    if (this->credits < build_cost) {
        std::cout « "Cannot build wave energy converter: insufficient credits (need "
            « build_cost « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->tile_improvement_ptr = new WaveEnergyConverter(
        this->position_x,
        this->position_y,
        this->tile_resource,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->has_improvement = true;
    this->decoration_cleared = true;
    this->assets_manager_ptr->getSound("splash")->play();
    this->__closeBuildMenu();

    this->__sendCreditsSpentMessage(build_cost);
    this->__sendTileStateMessage();
    this->__sendGameStateRequest();

    return;
}   /* __buildWaveEnergyConverter() */
```

4.7.3.7 __buildWindTurbine()

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

Helper method to build a wind turbine on this tile.

```
{
    int build_cost = WIND_TURBINE_BUILD_COST;

    if (
        (this->tile_type == TileType :: LAKE) or
        (this->tile_type == TileType :: OCEAN)
    ) {
        build_cost *= WIND_TURBINE_WATER_BUILD_MULTIPLIER;
    }

    if (this->credits < build_cost) {
        std::cout « "Cannot build wind turbine: insufficient credits (need "
            « build_cost « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    this->tile_improvement_ptr = new WindTurbine(
        this->position_x,
        this->position_y,
        this->tile_resource,
        this->event_ptr,
        this->render_window_ptr,
        this->assets_manager_ptr,
        this->message_hub_ptr
    );

    this->has_improvement = true;
    this->__closeBuildMenu();

    if (
        (this->tile_type == TileType :: LAKE) or
        (this->tile_type == TileType :: OCEAN)
    ) {
        this->decoration_cleared = true;
        this->assets_manager_ptr->getSound("splash")->play();
    }

    this->__sendCreditsSpentMessage(build_cost);
    this->__sendTileStateMessage();
    this->__sendGameStateRequest();

    return;
}   /* __buildWindTurbine() */
```

4.7.3.8 __clearDecoration()

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

Helper method to clear tile decoration.

```
{
    this->decoration_cleared = true;
    this->draw_explosion = true;

    switch (this->tile_type) {
        case (TileType :: FOREST): {
            this->assets_manager_ptr->getSound("clear non-mountains tile")->play();

            break;
        }


        case (TileType :: MOUNTAINS): {
            this->assets_manager_ptr->getSound("clear mountains tile")->play();

            break;
        }


        case (TileType :: PLAINS): {
            this->assets_manager_ptr->getSound("clear non-mountains tile")->play();

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __clearDecoration() */
```

4.7.3.9 __closeBuildMenu()

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

Helper method to close the tile improvement build menu.

```
{
    if (not this->build_menu_open) {
        return;
    }

    this->build_menu_open = false;
    this->assets_manager_ptr->getSound("build menu close")->play();

    return;
}   /* __closeBuildMenu() */
```

4.7.3.10 __getTileCoordsSubstring()

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

Helper method to assemble and return tile coordinates substring.

Returns

Tile coordinates substring.

```
{
    std::string coords_substring = "TILE COORDS:  (";
    coords_substring += std::to_string(int(this->position_x - 400));
    coords_substring += ", ";
    coords_substring += std::to_string(int(this->position_y - 400));
    coords_substring += ")\n";

    return coords_substring;
}   /* __getTileCoordsSubstring() */
```

4.7.3.11 __getTileImprovementSubstring()

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

Helper method to assemble and return the tile improvement substring.

Returns

Tile improvement substring.

```
{
    std::string improvement_substring = "TILE IMPROVEMENT:  ";

    if (this->has_improvement) {
        improvement_substring += this->tile_improvement_ptr->tile_improvement_string;
        improvement_substring += "\n";
    }

    else {
        improvement_substring += "NONE\n";
    }

    return improvement_substring;
}   /* __getTileImprovementSubstring() */
```

4.7.3.12 __getTileOptionsSubstring()

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

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

```
{
    //                   32 char x 17 line console "--------------------------------\n";
    std::string options_substring                 = "     **** TILE OPTIONS ****     \n";
    options_substring                            += "                                \n";

    if (this->game_phase == "build settlement") {
        if (
            (this->tile_type != TileType :: OCEAN) and
            (this->tile_type != TileType :: LAKE)
        ) {
            options_substring += "[B]:  BUILD SETTLEMENT (";
            options_substring += std::to_string(BUILD_SETTLEMENT_COST);
            options_substring += " K)\n";
        }
    }


    else if (this->game_phase == "system management") {
        if (this->has_improvement) {
            options_substring.clear();
            options_substring = this->tile_improvement_ptr->getTileOptionsSubstring();
        }


        else if (not this->resource_assessed) {
            options_substring += "[A]:  ASSESS RESOURCE (";
            options_substring += std::to_string(RESOURCE_ASSESSMENT_COST);
            options_substring += " K)\n";
        }


        else if (
            (not this->decoration_cleared) and
            (this->tile_type != TileType :: OCEAN) and
            (this->tile_type != TileType :: LAKE)
        ) {
            options_substring += "[C]:  CLEAR TILE (";

            switch (this->tile_type) {
                case (TileType :: FOREST): {
                    options_substring += std::to_string(CLEAR_FOREST_COST);

                    break;
                }


                case (TileType :: MOUNTAINS): {
                    options_substring += std::to_string(CLEAR_MOUNTAINS_COST);

                    break;
                }


                case (TileType :: PLAINS): {
                    options_substring += std::to_string(CLEAR_PLAINS_COST);

                    break;
                }


                default: {
                    //do nothing!

                    break;
                }
            }

            options_substring += " K)\n";
        }


        else if (
            (this->decoration_cleared) or
            (this->tile_type == TileType :: OCEAN) or
            (this->tile_type == TileType :: LAKE)
        ) {
            options_substring += "[B]: OPEN BUILD MENU\n";
        }
    }


    else if (this->game_phase == "victory") {
        options_substring                        += "       **** VICTORY ****        \n";
    }


    else {
        options_substring                             += "        **** LOSS ****          \n";
    }

    return options_substring;
}   /* __getTileOptionsString() */
```

4.7.3.13 __getTileResourceSubstring()

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

Helper method to assemble and return tile resource substring.

Returns

Tile resource substring.

```
{
    std::string resource_substring = "TILE RESOURCE:     ";

    if (this->resource_assessed) {
        switch (this->tile_resource) {
            case (TileResource :: POOR): {
                resource_substring += "POOR\n";

                break;
            }


            case (TileResource ::BELOW_AVERAGE): {
                resource_substring += "BELOW AVERAGE\n";

                break;
            }


            case (TileResource :: AVERAGE): {
                resource_substring += "AVERAGE\n";

                break;
            }


            case (TileResource :: ABOVE_AVERAGE): {
                resource_substring += "ABOVE AVERAGE\n";

                break;
            }


            case (TileResource :: GOOD): {
                resource_substring += "GOOD\n";

                break;
            }


            default: {
                resource_substring += "???\n";

                break;
            }
        }
    }

    else {
        resource_substring += "???\n";
    }

    return resource_substring;
}   /* __getTileResourceSubstring() */
```

4.7.3.14 __getTileTypeSubstring()

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

Helper method to assemble and return tile type substring.

Returns

Tile type substring.

```
{
    std::string type_substring = "TILE TYPE:         ";

    switch (this->tile_type) {
        case (TileType :: FOREST): {
            type_substring += "FOREST\n";

            break;
        }


        case (TileType :: LAKE): {
            type_substring += "LAKE\n";

            break;
        }


        case (TileType :: MOUNTAINS): {
            type_substring += "MOUNTAINS\n";

            break;
        }


        case (TileType :: OCEAN): {
            type_substring += "OCEAN\n";

            break;
        }


        case (TileType :: PLAINS): {
            type_substring += "PLAINS\n";

            break;
        }


        default: {
            type_substring += "???\n";

            break;
        }
    }

    return type_substring;
}   /* __getTileTypeSubstring() */
```

4.7.3.15 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (not this->is_selected) {
        return;
    }


    if (this->event_ptr->key.code == sf::Keyboard::Escape) {
            this->__setIsSelected(false);
        }


        if (this->build_menu_open) {
            switch (this->tile_type) {
                case (TileType :: FOREST): {
                    switch (this->event_ptr->key.code) {
                        case (sf::Keyboard::D): {
                            this->__buildDieselGenerator();

                            break;
                        }


                        case (sf::Keyboard::S): {
                            this->__buildSolarPV();

                            break;
                        }


                        case (sf::Keyboard::W): {
                            this->__buildWindTurbine();

                            break;
                        }


                        case (sf::Keyboard::E): {
                            this->__buildEnergyStorage();

                            break;
                        }


                        default: {
                            // do nothing!

                            break;
                        }
                    }

                    break;
                }


                case (TileType :: LAKE): {
                    switch (this->event_ptr->key.code) {
                        case (sf::Keyboard::S): {
                            this->__buildSolarPV();

                            break;
                        }


                        case (sf::Keyboard::W): {
                            this->__buildWindTurbine();

                            break;
                        }


                        default: {
                            // do nothing!

                            break;
                        }
                    }

                    break;
                }


                case (TileType :: MOUNTAINS): {
                    switch (this->event_ptr->key.code) {
                        case (sf::Keyboard::D): {
                            this->__buildDieselGenerator();

                            break;
                        }


                        case (sf::Keyboard::S): {
                            this->__buildSolarPV();

                            break;
                        }


                        case (sf::Keyboard::W): {
                            this->__buildWindTurbine();

                            break;
                        }


                        case (sf::Keyboard::E): {
                            this->__buildEnergyStorage();

                            break;
                        }


                        default: {
                            // do nothing!

                            break;
                        }
                    }

                    break;
                }


                case (TileType :: OCEAN): {
                    switch (this->event_ptr->key.code) {
                        case (sf::Keyboard::W): {
                             this->__buildWindTurbine();

                             break;
                         }


                         case (sf::Keyboard::T): {
                             this->__buildTidalTurbine();

                             break;
                         }


                         case (sf::Keyboard::A): {
                             this->__buildWaveEnergyConverter();

                             break;
                         }


                         default: {
                             // do nothing!

                             break;
                         }
                     }

                     break;
                 }


                 case (TileType :: PLAINS): {
                     switch (this->event_ptr->key.code) {
                         case (sf::Keyboard::D): {
                             this->__buildDieselGenerator();

                             break;
                         }


                         case (sf::Keyboard::S): {
                             this->__buildSolarPV();

                             break;
                         }


                         case (sf::Keyboard::W): {
                             this->__buildWindTurbine();

                             break;
                         }


                         case (sf::Keyboard::E): {
                             this->__buildEnergyStorage();

                             break;
                         }


                         default: {
                             // do nothing!

                             break;
                         }
                     }

                     break;
                 }


                 default: {
                     //do nothing!

                     break;
                 }
             }
         }


         if (this->game_phase == "build settlement") {
             if (
                 (this->tile_type != TileType :: OCEAN) and
                 (this->tile_type != TileType :: LAKE)
             ) {
                 if (this->event_ptr->key.code == sf::Keyboard::B) {
                     this->__buildSettlement();
                 }
             }
         }


         else if (this->game_phase == "system management") {
             if (this->has_improvement) {
                 if (this->tile_improvement_ptr->tile_improvement_type != TileImprovementType :: SETTLEMENT)
       {
                     if (this->event_ptr->key.code == sf::Keyboard::P) {
                         this->__scrapImprovement();
                     }
                 }

                 /*
                  * All other inputs will be caught and handled by
                  *   this->tile_improvement_ptr->processEvent()
                  */
             }


             else if (not this->resource_assessed) {
                 if (this->event_ptr->key.code == sf::Keyboard::A) {
                     if (this->credits < RESOURCE_ASSESSMENT_COST) {
                         std::cout « "Cannot assess resource: insufficient credits (need "
                             « RESOURCE_ASSESSMENT_COST « " K)" « std::endl;

                         this->__sendInsufficientCreditsMessage();
                     }

                     else {
                         this->assess();
                         this->__sendCreditsSpentMessage(RESOURCE_ASSESSMENT_COST);
                         this->__sendTileStateMessage();
                         this->__sendGameStateRequest();
                     }
                 }
             }


             else if (
                 (not this->decoration_cleared) and
                 (this->tile_type != TileType :: OCEAN) and
                 (this->tile_type != TileType :: LAKE)
             ) {
                 if (this->event_ptr->key.code == sf::Keyboard::C) {
                     int clear_cost = 0;

                     switch (this->tile_type) {
                         case (TileType :: FOREST): {
                             clear_cost = CLEAR_FOREST_COST;

                             break;
                         }

                         case (TileType :: MOUNTAINS): {
                             clear_cost = CLEAR_MOUNTAINS_COST;

                             break;
                         }


                         case (TileType :: PLAINS): {
                             clear_cost = CLEAR_PLAINS_COST;

                             break;
                         }


                         default: {
                             // do nothing!

                             break;
                         }
                     }

                     if (this->credits < clear_cost) {
                         std::cout « "Cannot clear tile: insufficient credits (need "
                             « clear_cost « " K)" « std::endl;

                         this->__sendInsufficientCreditsMessage();
                     }

                     else {
                         this->__clearDecoration();
                         this->__sendCreditsSpentMessage(clear_cost);
                         this->__sendTileStateMessage();
                         this->__sendGameStateRequest();
                     }
                 }
             }


             else if (
                 (this->decoration_cleared) or
                 (this->tile_type == TileType :: OCEAN) or
                 (this->tile_type == TileType :: LAKE)
             ) {
                 if (this->event_ptr->key.code == sf::Keyboard::B) {
                     this->__openBuildMenu();
                 }
             }
         }

         return;
     }   /* __handleKeyPressEvents() */
```

4.7.3.16 __handleKeyReleaseEvents()

```
void HexTile::__handleKeyReleaseEvents (
            void ) [private]
{
    if (not this->is_selected) {
        return;
    }


    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::P): {
            if (this->has_improvement) {
                this->scrap_improvement_frame = 0;

                if (
                    this->tile_improvement_ptr->tile_improvement_sprite_static.getTexture() != NULL
                ) {
                    this->tile_improvement_ptr->tile_improvement_sprite_static.setColor(
                        sf::Color(255, 255, 255, 255)
                    );
                }

                else {
                    for (
                        size_t i = 0;
                        i < this->tile_improvement_ptr->tile_improvement_sprite_animated.size();
                        i++
                    ) {
                        this->tile_improvement_ptr->tile_improvement_sprite_animated[i].setColor(
                            sf::Color(255, 255, 255, 255)
                        );
                    }
                }
            }


            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    /*
    if (this->event_ptr->key.code == sf::Keyboard::P) {

    }
    */

    return;
}   /* __handleKeyReleaseEvents() */
```

4.7.3.17 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            if (this->__isClicked()) {
                std::cout « "Tile (" « this->position_x « ", " «
                    this->position_y « ") was selected" « std::endl;

                this->__setIsSelected(true);

                this->__sendTileSelectedMessage();
                this->__sendTileStateMessage();
                this->__sendGameStateRequest();
            }

            else {
                this->__setIsSelected(false);
            }

            break;
        }


        case (sf::Mouse::Right): {
            this->__setIsSelected(false);

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.7.3.18 __isClicked()

```
bool HexTile::__isClicked (
              void ) [private]
```

Helper method to determine if tile was clicked on.

Returns

Boolean indicating whether or not tile was clicked on.

```
{
    sf::Vector2i mouse_position = sf::Mouse::getPosition(*render_window_ptr);

    double mouse_x = mouse_position.x;
    double mouse_y = mouse_position.y;

    double distance = sqrt(
        pow(this->position_x - mouse_x, 2) +
        pow(this->position_y - mouse_y, 2)
    );

    if (distance < this->minor_radius) {
        return true;
    }
    else {
        return false;
    }
}   /* __isClicked() */
```

4.7.3.19 __openBuildMenu()

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

Helper method to open the tile improvement build menu.

```
{
    if (this->build_menu_open) {
        return;
    }

    this->build_menu_open = true;
    this->assets_manager_ptr->getSound("build menu open")->play();

    return;
}   /* __openBuildMenu() */
```

4.7.3.20 __scrapImprovement()

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

Helper method to scrap the tile improvement (Settlement cannot be scrapped). Requires the mapped key to be held continuously to confirm.

```
{
    //  1. implement key hold confirmation
    if (this->scrap_improvement_frame <= FRAMES_PER_SECOND) {
        double colour_scalar =
            1 - ((double)(this->scrap_improvement_frame) / (FRAMES_PER_SECOND));

        if (
            this->tile_improvement_ptr->tile_improvement_sprite_static.getTexture() != NULL
        ) {
            this->tile_improvement_ptr->tile_improvement_sprite_static.setColor(
                sf::Color(255, 255 * colour_scalar, 255 * colour_scalar, 255)
            );
        }

        else {
            for (
                size_t i = 0;
                i < this->tile_improvement_ptr->tile_improvement_sprite_animated.size();
                i++
            ) {
                this->tile_improvement_ptr->tile_improvement_sprite_animated[i].setColor(
                    sf::Color(255, 255 * colour_scalar, 255 * colour_scalar, 255)
                );
            }
        }

        this->scrap_improvement_frame += 4;
    }


    //  2. carry out scrapping
    else {
        this->draw_explosion = true;
        this->assets_manager_ptr->getSound("clear non-mountains tile")->play();

        if (this->tile_improvement_ptr->production_menu_open) {
            this->tile_improvement_ptr->production_menu_open = false;
            this->assets_manager_ptr->getSound("build menu close")->play();
        }

        delete this->tile_improvement_ptr;
        this->tile_improvement_ptr = NULL;

        this->has_improvement = false;

        this->scrap_improvement_frame = 0;

        if (
            (this->tile_type == TileType :: LAKE) or
            (this->tile_type == TileType :: OCEAN)
        ) {
            this->decoration_cleared = false;
        }

        this->__sendCreditsSpentMessage(SCRAP_COST);
        this->__sendTileStateMessage();
        this->__sendGameStateRequest();
    }

    return;
}   /* __scrapImprovement() */
```

4.7.3.21 __sendAssessNeighboursMessage()

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

Helper method to format and send assess neighbours message.

```
{
    Message assess_neighbours_message;

    assess_neighbours_message.channel = HEX_MAP_CHANNEL;
    assess_neighbours_message.subject = "assess neighbours";

    this->message_hub_ptr->sendMessage(assess_neighbours_message);

    std::cout « "Assess neighbours message sent by " « this « std::endl;

    return;
}   /* __sendAssessNeighboursMessage() */
```

4.7.3.22 __sendCreditsSpentMessage()

```
void HexTile::__sendCreditsSpentMessage (
            int credits_spent ) [private]
```

Helper method to format and send a credits spent message.

Parameters

credits_spent	The number of credits that were spent.

```
{
    Message credits_spent_message;

    credits_spent_message.channel = GAME_CHANNEL;
    credits_spent_message.subject = "credits spent";

    credits_spent_message.int_payload["credits spent"] = credits_spent;

    this->message_hub_ptr->sendMessage(credits_spent_message);

    std::cout « "Credits spent (" « credits_spent « ") message sent by " « this
        « std::endl;
    return;
}   /* __sendCreditsSpentMessage() */
```

4.7.3.23 __sendGameStateRequest()

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

Helper method to format and send a game state request (message).

```
{
    Message game_state_request;

    game_state_request.channel = GAME_CHANNEL;
    game_state_request.subject = "state request";

    this->message_hub_ptr->sendMessage(game_state_request);

    std::cout « "Game state request message sent by " « this « std::endl;
    return;
}   /* __sendGameStateRequest() */
```

4.7.3.24 __sendInsufficientCreditsMessage()

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

Helper method to format and send an insufficient credits message.

```
{
    Message insufficient_credits_message;

    insufficient_credits_message.channel = GAME_CHANNEL;
    insufficient_credits_message.subject = "insufficient credits";

    this->message_hub_ptr->sendMessage(insufficient_credits_message);

    std::cout « "Insufficient credits message sent by " « this « std::endl;

    return;
}   /* __sendInsufficientCreditsMessage() */
```

4.7.3.25 __sendTileSelectedMessage()

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

Helper method to format and send message on tile selection.

```
{
    Message tile_selected_message;

    tile_selected_message.channel = TILE_SELECTED_CHANNEL;
    tile_selected_message.subject = "tile selected";

    this->message_hub_ptr->sendMessage(tile_selected_message);

    return;
}   /* __sendTileSelectedMessage() */
```

4.7.3.26 __sendTileStateMessage()

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

Helper method to format and send tile state message.

```
{
    Message tile_state_message;

    tile_state_message.channel = TILE_STATE_CHANNEL;
    tile_state_message.subject = "tile state";


    //                   32 char x 17 line console "---------------------------------\n";
    std::string console_string                   = "       **** TILE INFO ****       \n";

    console_string                            += this->__getTileCoordsSubstring();
    console_string                            += "                                 \n";

    console_string                            += this->__getTileTypeSubstring();
    console_string                            += this->__getTileResourceSubstring();
    console_string                            += this->__getTileImprovementSubstring();
    console_string                            += "                                 \n";

    console_string                            += this->__getTileOptionsSubstring();

    tile_state_message.string_payload["console string"] = console_string;

    this->message_hub_ptr->sendMessage(tile_state_message);

    std::cout « "Tile state message sent by " « this « std::endl;
    return;
}   /* __sendTileStateMessage() */
```

4.7.3.27 __sendUpdateGamePhaseMessage()

```
void HexTile::__sendUpdateGamePhaseMessage (
                std::string game_phase )  [private]
```

Helper method to format and send update game phase message.

Parameters

game_phase	The updated game phase.

```
{
    Message update_game_phase_message;

    update_game_phase_message.channel = GAME_CHANNEL;
    update_game_phase_message.subject = "update game phase";

    update_game_phase_message.string_payload["game phase"] = game_phase;

    this->message_hub_ptr->sendMessage(update_game_phase_message);

    std::cout « "Update game phase message sent by " « this « std::endl;

    return;
}   /* __sendUpdateGamePhaseMessage() */
```

4.7.3.28 __setIsSelected()

```
void HexTile::__setIsSelected (
            bool is_selected )  [private]
```

Helper method to set the is selected attribute (of tile and improvement).

Parameters

is_selected	The value to set the is selected attribute to.

```
{
    this->is_selected = is_selected;

    if (this->tile_improvement_ptr != NULL) {
        this->tile_improvement_ptr->setIsSelected(is_selected);
        this->tile_improvement_ptr->update();
    }

    if ((not is_selected) and this->build_menu_open) {
        this->__closeBuildMenu();
    }

    return;
}   /* __setIsSelected() */
```

4.7.3.29 __setResourceText()

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

Helper method to set up resource text.

```
{
    this->resource_text.setFont(*(assets_manager_ptr->getFont("DroidSansMono")));

    this->resource_text.setFillColor(sf::Color(0, 0, 0, 255));

    if (this->resource_assessed) {
        switch (this->tile_resource) {
            case (TileResource :: POOR): {
                this->resource_text.setString("-2");
                this->resource_text.setFillColor(MONOCHROME_TEXT_RED);

                break;
            }

            case (TileResource :: BELOW_AVERAGE): {
                this->resource_text.setString("-1");
                this->resource_text.setFillColor(MONOCHROME_TEXT_RED);

                break;
            }

            case (TileResource :: AVERAGE): {
                this->resource_text.setString("+0");

                break;
            }

            case (TileResource :: ABOVE_AVERAGE): {
                this->resource_text.setString("+1");
                this->resource_text.setFillColor(MONOCHROME_TEXT_GREEN);

                break;
            }

            case (TileResource :: GOOD): {
                this->resource_text.setString("+2");
                this->resource_text.setFillColor(MONOCHROME_TEXT_GREEN);

                break;
            }

            default: {
                this->resource_text.setString("");

                break;
            }
        }
    }

    else {
        this->resource_text.setString("");
    }

    this->resource_text.setCharacterSize(20);

    this->resource_text.setOrigin(
        this->resource_text.getLocalBounds().width / 2,
        this->resource_text.getLocalBounds().height / 2
    );

    this->resource_text.setPosition(
        this->position_x,
        this->position_y - 4
    );

    this->resource_text.setOutlineThickness(1);
    this->resource_text.setOutlineColor(sf::Color(0, 0, 0, 255));

    return;
}   /* __setResourceText() */
```

4.7.3.30 __setUpBuildMenu()

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

Helper method to set up and place build menu assets (drawable).

```
{
    this->build_menu_options_vec.clear();
    this->build_menu_options_text_vec.clear();

    //  1. set up and place build menu backing and text
    this->build_menu_backing.setSize(sf::Vector2f(600, 256));
    this->build_menu_backing.setOrigin(300, 128);
    this->build_menu_backing.setPosition(400, 400);
    this->build_menu_backing.setFillColor(MONOCHROME_SCREEN_BACKGROUND);
    this->build_menu_backing.setOutlineColor(MENU_FRAME_GREY);
    this->build_menu_backing.setOutlineThickness(4);

    this->build_menu_backing_text.setString("**** BUILD MENU ****");
    this->build_menu_backing_text.setFont(
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220"))
    );
    this->build_menu_backing_text.setCharacterSize(16);
    this->build_menu_backing_text.setFillColor(MONOCHROME_TEXT_GREEN);
    this->build_menu_backing_text.setOrigin(
        this->build_menu_backing_text.getLocalBounds().width / 2, 0
    );
    this->build_menu_backing_text.setPosition(400, 400 - 128 + 4);

    //  2. set up and place build menu option sprites and text
    switch (this->tile_type) {
        case (TileType :: FOREST): {
            this->__setUpDieselGeneratorBuildOption();
            this->__setUpSolarPVBuildOption();
            this->__setUpWindTurbineBuildOption();
            //this->__setUpEnergyStorageSystemBuildOption();

            break;
        }


        case (TileType :: LAKE): {
            this->__setUpSolarPVBuildOption(true);
            this->__setUpWindTurbineBuildOption(true);

            break;
        }


        case (TileType :: MOUNTAINS): {
            this->__setUpDieselGeneratorBuildOption();
            this->__setUpSolarPVBuildOption();
            this->__setUpWindTurbineBuildOption();
            //this->__setUpEnergyStorageSystemBuildOption();

            break;
        }


        case (TileType :: OCEAN): {
            this->__setUpWindTurbineBuildOption(false, true);
            this->__setUpTidalTurbineBuildOption();
            this->__setUpWaveEnergyConverterBuildOption();

            break;
        }


        case (TileType :: PLAINS): {
            this->__setUpDieselGeneratorBuildOption();
            this->__setUpSolarPVBuildOption();
            this->__setUpWindTurbineBuildOption();
            //this->__setUpEnergyStorageSystemBuildOption();

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __setUpBuildMenu() */
```

4.7.3.31 __setUpBuildOption()

```
void HexTile::__setUpBuildOption (
        std::string texture_key,
        std::string option_string )  [private]
```

Helper method to set up and postion the sprite and text for a build option.

Parameters

texture_key	The key for the appropriate illustration asset for the build option.
option_string	A string for the build option.

```
{
    size_t n_options = this->build_menu_options_vec.size();

    //  1. set up option sprite(s)
    this->build_menu_options_vec.push_back({});

    if (not texture_key.empty()) {
        sf::Sprite texture_sheet(
            *(this->assets_manager_ptr->getTexture(texture_key))
        );

        int sheet_height = texture_sheet.getLocalBounds().height;
        int n_subrects = sheet_height / 64;

        for (int i = 0; i < n_subrects; i++) {
            this->build_menu_options_vec.back().push_back(
                sf::Sprite(
                    *(this->assets_manager_ptr->getTexture(texture_key)),
                    sf::IntRect(0, i * 64, 64, 64)
                )
            );

            this->build_menu_options_vec.back().back().setOrigin(
                this->build_menu_options_vec.back().back().getLocalBounds().width / 2,
                this->build_menu_options_vec.back().back().getLocalBounds().height
            );

            this->build_menu_options_vec.back().back().setPosition(
                400 - 300 + 75 + n_options * 150,
                400 - 32
            );
        }
    }

    else {
        this->build_menu_options_vec.back().push_back(sf::Sprite());
    }


    //  2. set up option text
    this->build_menu_options_text_vec.push_back(
        sf::Text(
            option_string,
            *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
            16
        )
    );

    this->build_menu_options_text_vec.back().setOrigin(
        this->build_menu_options_text_vec.back().getLocalBounds().width / 2,
        0
    );

    this->build_menu_options_text_vec.back().setPosition(
        400 - 300 + 75 + n_options * 150,
        400 - 16 - 4
    );

    this->build_menu_options_text_vec.back().setFillColor(MONOCHROME_TEXT_GREEN);

    return;
}   /* __setUpBuildOption() */
```

4.7.3.32 __setUpDieselGeneratorBuildOption()

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

Helper method to set up and position the diesel generator build option.

```
{
    //  1. set up option sprite(s)
    std::string texture_key = "diesel generator";

    //  2. set up option string (up to 16 chars wide)
    //                                 "----------------\n"
    std::string diesel_generator_string = "DIESEL GENERATOR\n";
    diesel_generator_string            += "                \n";
    diesel_generator_string            += "CAPACITY: 100 kW\n";
    diesel_generator_string             += "COST:      ";
    diesel_generator_string             += std::to_string(DIESEL_GENERATOR_BUILD_COST);
    diesel_generator_string             += " K\n\n\n";
    diesel_generator_string             += "BUILD:     [D]    \n";

    //  3. call general method
    this->__setUpBuildOption(texture_key, diesel_generator_string);

    return;
}   /* __setUpDieselGeneratorBuildOption() */
```

4.7.3.33 __setUpEnergyStorageSystemBuildOption()

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

Helper method to set up and position the wave energy converter build option.

```
{
    /*
    //  1. set up option sprite(s)
    std::string texture_key = "energy storage system";

    //  2. set up option string (up to 16 chars wide)
    //                                          "----------------\n"
    std::string energy_storage_system_string    = " ENERGY STORAGE \n";
    energy_storage_system_string                += "                \n";
    energy_storage_system_string                += "CAPCTY:    1 MWh\n";
    energy_storage_system_string                += "COST:      ";
    energy_storage_system_string                += std::to_string(ENERGY_STORAGE_SYSTEM_BUILD_COST);
    energy_storage_system_string                += " K\n\n\n";
    energy_storage_system_string                += "BUILD:     [E]    \n";

    //  3. call general method
    this->__setUpBuildOption(texture_key, energy_storage_system_string);
    */
    return;
}   /* __setUpEnergyStorageSystemBuildOption() */
```

4.7.3.34 __setUpMagnifyingGlassSprite()

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

Helper method to set up and position magnifying glass sprite.

```
{
    this->magnifying_glass_sprite.setTexture(
        *(this->assets_manager_ptr->getTexture("magnifying_glass_64x64_1"))
    );

    this->magnifying_glass_sprite.setOrigin(
        this->magnifying_glass_sprite.getLocalBounds().width / 2,
        this->magnifying_glass_sprite.getLocalBounds().height / 2
    );

    this->magnifying_glass_sprite.setPosition(
        this->position_x,
        this->position_y
    );

    return;
}   /* __setUpMagnifyingGlassSprite() */
```

4.7.3.35 __setUpNodeSprite()

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

Helper method to set up node sprite.

```
{
    this->node_sprite.setRadius(4);

    this->node_sprite.setOrigin(
        this->node_sprite.getLocalBounds().width / 2,
        this->node_sprite.getLocalBounds().height / 2
    );

    this->node_sprite.setPosition(this->position_x, this->position_y);

    this->node_sprite.setFillColor(sf::Color(255, 0, 0, 255));

    return;
}   /* __setUpNodeSprite() */
```

4.7.3.36 __setUpResourceChipSprite()

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

Helper method to set up resource chip sprite.

```
{
    this->resource_chip_sprite.setRadius(2 * this->minor_radius / 3);

    this->resource_chip_sprite.setOrigin(
        this->resource_chip_sprite.getLocalBounds().width / 2,
        this->resource_chip_sprite.getLocalBounds().height / 2
    );

    this->resource_chip_sprite.setPosition(this->position_x, this->position_y);

    this->resource_chip_sprite.setFillColor(RESOURCE_CHIP_GREY);

    return;
}   /* __setUpResourceChip() */
```

4.7.3.37 __setUpSelectOutlineSprite()

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

Helper method to set up select outline sprite.

```
{
    int n_points = 6;

    this->select_outline_sprite.setPointCount(n_points);

    for (int i = 0; i < n_points; i++) {
        this->select_outline_sprite.setPoint(
            i,
            sf::Vector2f(
                this->position_x + this->major_radius * cos((30 + 60 * i) * (M_PI / 180)),
                this->position_y + this->major_radius * sin((30 + 60 * i) * (M_PI / 180))
            )
        );
    }

    this->select_outline_sprite.setOutlineThickness(4);
    this->select_outline_sprite.setOutlineColor(MONOCHROME_TEXT_RED);

    this->select_outline_sprite.setFillColor(sf::Color(0, 0, 0, 0));

    return;
}   /* __setUpSelectOutline() */
```

4.7.3.38 __setUpSolarPVBuildOption()

```
void HexTile::__setUpSolarPVBuildOption (
              bool is_lake = false )  [private]
```

Helper method to set up and position the solar PV array build option.

Parameters

is_lake	If being built on a lake.

```
{
    //  1. set up option sprite(s)
    std::string texture_key = "solar PV array";

    //  2. set up option string (up to 16 chars wide)
    int build_cost = SOLAR_PV_BUILD_COST;
    if (is_lake) {
        build_cost *= SOLAR_PV_WATER_BUILD_MULTIPLIER;
    }

    //                                    "----------------\n"
    std::string solar_PV_string         = " SOLAR PV ARRAY \n";
    solar_PV_string                    += "                \n";
    solar_PV_string                    += "CAPACITY: 100 kW\n";
    solar_PV_string                    += "COST:      ";
    solar_PV_string                    += std::to_string(build_cost);
    solar_PV_string                    += " K";

    if (is_lake) {
        solar_PV_string += "\n** LAKE BUILD **\n\n";
    }
    else {
        solar_PV_string += "\n\n\n";
    }

    solar_PV_string                    += "BUILD:    [S]   \n";

    //  3. call general method
    this->__setUpBuildOption(texture_key, solar_PV_string);

    return;
}   /* __setUpSolarPVBuildOption() */
```

4.7.3.39 __setUpTidalTurbineBuildOption()

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

Helper method to set up and position the tidal turbine build option.

```
{
    //  1. set up option sprite(s)
    std::string texture_key = "tidal turbine";

    //  2. set up option string (up to 16 chars wide)
    //                                    "----------------\n"
    std::string tidal_turbine_string    = " TIDAL TURBINE  \n";
    tidal_turbine_string               += "                \n";
    tidal_turbine_string               += "CAPACITY: 100 kW\n";
    tidal_turbine_string               += "COST:      ";
    tidal_turbine_string               += std::to_string(TIDAL_TURBINE_BUILD_COST);
    tidal_turbine_string               += " K\n\n\n";
    tidal_turbine_string               += "BUILD:    [T]   \n";

    //  3. call general method
    this->__setUpBuildOption(texture_key, tidal_turbine_string);

    return;
}   /* __setUpTidalTurbineBuildOption() */
```

4.7.3.40 __setUpTileExplosionReel()

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

Helper method to set up tile explosion sprite reel.

```
{
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            this->explosion_sprite_reel.push_back(
                sf::Sprite(
                    *(this->assets_manager_ptr->getTexture("tile clear explosion")),
                    sf::IntRect(j * 64, i * 64, 64, 64)
                )
            );

            this->explosion_sprite_reel.back().setOrigin(
                this->explosion_sprite_reel.back().getLocalBounds().width / 2,
                this->explosion_sprite_reel.back().getLocalBounds().height / 2
            );

            this->explosion_sprite_reel.back().setPosition(
                this->position_x,
                this->position_y
            );
        }
    }

    return;
}   /* __setUpTileExplosionReel() */
```

4.7.3.41 __setUpTileSprite()

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

Helper method to set up tile sprite.

```
{
    int n_points = 6;

    this->tile_sprite.setPointCount(n_points);

     for (int i = 0; i < n_points; i++) {
         this->tile_sprite.setPoint(
             i,
             sf::Vector2f(
                 this->position_x + this->major_radius * cos((30 + 60 * i) * (M_PI / 180)),
                 this->position_y + this->major_radius * sin((30 + 60 * i) * (M_PI / 180))
             )
         );
     }

     this->tile_sprite.setOutlineThickness(1);
     this->tile_sprite.setOutlineColor(sf::Color(175, 175, 175, 255));

     return;
}   /* __setUpTileSprite() */
```

4.7.3.42 __setUpWaveEnergyConverterBuildOption()

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

Helper method to set up and position the wave energy converter build option.

```
{
    //  1. set up option sprite(s)
    std::string texture_key = "wave energy converter";

    //  2. set up option string (up to 16 chars wide)
    //                                            "----------------\n"
    std::string wave_energy_converter_string     = "WAVE ENERGY CVTR\n";
    wave_energy_converter_string                += "                \n";
    wave_energy_converter_string                += "CAPACITY: 100 kW\n";
    wave_energy_converter_string                += "COST:      ";
    wave_energy_converter_string                += std::to_string(WAVE_ENERGY_CONVERTER_BUILD_COST);
    wave_energy_converter_string                += " K\n\n\n";
    wave_energy_converter_string                += "BUILD:     [A]    \n";

    //  3. call general method
    this->__setUpBuildOption(texture_key, wave_energy_converter_string);

    return;
}   /* __setUpWaveEnergyConverterBuildOption() */
```

4.7.3.43 __setUpWindTurbineBuildOption()

```
void HexTile::__setUpWindTurbineBuildOption (
            bool is_lake = false,
            bool is_ocean = false )  [private]
```

Helper method to set up and position the wind turbine build option.

Parameters

is_lake	If being built on a lake tile.
is_ocean	If being built on an ocean tile.

```
{
    //  1. set up option sprite(s)
    std::string texture_key = "wind turbine";

    //  2. set up option string (up to 16 chars wide)
    int build_cost = WIND_TURBINE_BUILD_COST;
    if (is_lake or is_ocean) {
        build_cost *= WIND_TURBINE_WATER_BUILD_MULTIPLIER;
    }

    //                                    "----------------\n"
    std::string wind_turbine_string      = "  WIND TURBINE  \n";
    wind_turbine_string                 += "                \n";
    wind_turbine_string                 += "CAPACITY: 100 kW\n";
    wind_turbine_string                 += "COST:      ";
    wind_turbine_string                 += std::to_string(build_cost);
    wind_turbine_string                 += " K";

    if (is_lake) {
        wind_turbine_string += "\n** LAKE BUILD **\n\n";
    }
    else if (is_ocean) {
        wind_turbine_string += "\n* OCEAN BUILD * \n\n";
    }
    else {
        wind_turbine_string += "\n\n\n";
    }

    wind_turbine_string                 += "BUILD:     [W]    \n";

    //  3. call general method
    this->__setUpBuildOption(texture_key, wind_turbine_string);

    return;
}   /* __setUpWindTurbineBuildOption() */
```

4.7.3.44 assess()

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

Method to assess the tile's resource.

```
{
    this->resource_assessed = true;
    this->resource_assessment = true;

    this->assets_manager_ptr->getSound("resource assessment")->play();

    this->__setResourceText();
    this->__sendTileStateMessage();

    return;
}   /* assess() */
```

4.7.3.45 decorateTile()

```
void HexTile::decorateTile (
            void )
```

Method to decorate tile.

```
{
    switch (this->tile_type) {
        case (TileType :: FOREST): {
            this->tile_decoration_sprite.setTexture(
                *(this->assets_manager_ptr->getTexture("pine_tree_64x64_1"))
            );

            break;
        }

        case (TileType :: LAKE): {
            this->tile_decoration_sprite.setTexture(
                *(this->assets_manager_ptr->getTexture("water_shimmer_64x64_1"))
            );

            break;
        }

        case (TileType :: MOUNTAINS): {
            this->tile_decoration_sprite.setTexture(
                *(this->assets_manager_ptr->getTexture("mountain_64x64_1"))
            );

            break;
        }

        case (TileType :: OCEAN): {
            this->tile_decoration_sprite.setTexture(
                *(this->assets_manager_ptr->getTexture("water_waves_64x64_1"))
            );

            break;
        }

        case (TileType :: PLAINS): {
            this->tile_decoration_sprite.setTexture(
                *(this->assets_manager_ptr->getTexture("wheat_64x64_1"))
            );

            break;
        }

        default: {
            // do nothing!

            break;
        }
    }


    if (this->tile_type == TileType :: OCEAN or this->tile_type == TileType :: LAKE) {
        this->tile_decoration_sprite.setOrigin(
            this->tile_decoration_sprite.getLocalBounds().width / 2,
            this->tile_decoration_sprite.getLocalBounds().height / 2
        );

        this->tile_decoration_sprite.setPosition(
            this->position_x,
            this->position_y
        );

        if ((double)rand() / RAND_MAX > 0.5) {
            this->tile_decoration_sprite.setScale(sf::Vector2f(-1, 1));
        }
    }

    else {
        this->tile_decoration_sprite.setOrigin(
            this->tile_decoration_sprite.getLocalBounds().width / 2,
            this->tile_decoration_sprite.getLocalBounds().height
        );

        this->tile_decoration_sprite.setPosition(
            this->position_x,
            this->position_y + 12
        );

        if ((double)rand() / RAND_MAX > 0.5) {
            this->tile_decoration_sprite.setScale(sf::Vector2f(-1, 1));
        }
    }

    return;
}   /* decorateTile(void) */
```

4.7.3.46 draw()

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

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

```
{
    //  1. draw hex
    this->render_window_ptr->draw(this->tile_sprite);

    //  2. draw node
    if (this->show_node) {
        this->render_window_ptr->draw(this->node_sprite);
    }

    //  3. draw tile decoration
    if (not this->decoration_cleared) {
        this->render_window_ptr->draw(this->tile_decoration_sprite);
    }

    //  4. draw selection outline
    if (this->is_selected) {
        sf::Color outline_colour = this->select_outline_sprite.getOutlineColor();

        outline_colour.a =
            255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2);

        this->select_outline_sprite.setOutlineColor(outline_colour);

        this->render_window_ptr->draw(this->select_outline_sprite);
    }

    //  5. draw tile improvement
    if (this->has_improvement) {
        if (not this->tile_improvement_ptr->just_built) {
            this->tile_improvement_ptr->draw();
        }
    }

    //  6. draw resource
    if (this->show_resource) {
        this->render_window_ptr->draw(this->resource_chip_sprite);
        this->render_window_ptr->draw(this->resource_text);
    }

    //  7. draw resource assessment notification
    if (this->resource_assessment) {
        int alpha = this->magnifying_glass_sprite.getColor().a;

        alpha -= 0.05 * FRAMES_PER_SECOND;
        if (alpha < 0) {
            alpha = 0;
            this->resource_assessment = false;
        }

        this->magnifying_glass_sprite.setColor(
            sf::Color(255, 255, 255, alpha)
        );

        this->render_window_ptr->draw(this->magnifying_glass_sprite);
    }

    //  8. draw explosion, then settlement placement
    if (this->draw_explosion) {
        this->render_window_ptr->draw(this->explosion_sprite_reel[this->explosion_frame]);

        if (this->frame % (FRAMES_PER_SECOND / 20) == 0) {
            this->explosion_frame++;
        }

        if (this->explosion_frame >= this->explosion_sprite_reel.size()) {
            this->draw_explosion = false;
            this->explosion_frame = 0;
        }
    }

    else if (this->has_improvement) {
        if (this->tile_improvement_ptr->just_built) {
            this->tile_improvement_ptr->draw();
        }
    }

    //  9. build menu
    if (this->build_menu_open) {
        this->render_window_ptr->draw(this->build_menu_backing);
        this->render_window_ptr->draw(this->build_menu_backing_text);

        for (size_t i = 0; i < this->build_menu_options_vec.size(); i++) {
            for (size_t j = 0; j < this->build_menu_options_vec[i].size(); j++) {
                this->render_window_ptr->draw(this->build_menu_options_vec[i][j]);
            }
            this->render_window_ptr->draw(this->build_menu_options_text_vec[i]);
        }
    }

    this->frame++;
    return;
}   /* draw() */
```

4.7.3.47 processEvent()

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

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

```
{
    //  1. process TileImprovement events
    if (
        this->is_selected and
        this->tile_improvement_ptr != NULL
    ) {
        this->tile_improvement_ptr->processEvent();
    }

    //  2. process HexTile events
    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::KeyReleased) {
        this->__handleKeyReleaseEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.7.3.48 processMessage()

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

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

```
{
    //  1. process TileImprovement messages
    if (this->tile_improvement_ptr != NULL) {
        this->tile_improvement_ptr->processMessage();
    }

    //  2. process HexTile messages
    if (this->is_selected) {
        if (not this->message_hub_ptr->isEmpty(GAME_STATE_CHANNEL)) {
            Message game_state_message = this->message_hub_ptr->receiveMessage(
                GAME_STATE_CHANNEL
            );

            if (game_state_message.subject == "game state") {
                this->credits = game_state_message.int_payload["credits"];
                this->game_phase = game_state_message.string_payload["game phase"];

                if (this->tile_improvement_ptr != NULL) {
                    this->tile_improvement_ptr->credits = this->credits;
                    this->tile_improvement_ptr->game_phase = this->game_phase;

                    this->tile_improvement_ptr->month =
                        game_state_message.int_payload["month"];

                    this->tile_improvement_ptr->demand_MWh =
                        game_state_message.int_payload["demand_MWh"];

                    this->tile_improvement_ptr->demand_vec_MWh =
                        game_state_message.vector_payload["demand_vec_MWh"];

                    this->tile_improvement_ptr->update();
                }

                std::cout « "Game state message received by " « this « std::endl;
                this->__sendTileStateMessage();
                this->message_hub_ptr->popMessage(GAME_STATE_CHANNEL);
            }
        }

        if (not this->message_hub_ptr->isEmpty(TILE_STATE_CHANNEL)) {
            Message tile_state_message = this->message_hub_ptr->receiveMessage(
                TILE_STATE_CHANNEL
            );

            if (tile_state_message.subject == "state request") {
                this->__sendTileStateMessage();

                std::cout « "Tile state request received by " « this « std::endl;
                this->message_hub_ptr->popMessage(TILE_STATE_CHANNEL);
            }
        }

        std::cout « "Current credits (HexTile): " « this->credits « " K" «
            std::endl;
    }

    return;
}   /* processMessage() */
```

4.7.3.49 setTileResource() [1/2]

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

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

Parameters

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

```
{
    //  1. check input
    if (input_value < 0 or input_value > 1) {
        std::string error_str = "ERROR  HexTile::setTileResource()  given input value is ";
        error_str += "not in the closed interval [0, 1]";

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. convert input value to tile resource
    TileResource tile_resource;

    if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[0]) {
        tile_resource = TileResource :: POOR;
    }
    else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[1]) {
        tile_resource = TileResource :: BELOW_AVERAGE;
    }
    else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[2]) {
        tile_resource = TileResource :: AVERAGE;
    }
    else if (input_value <= TILE_RESOURCE_CUMULATIVE_PROBABILITIES[3]) {
        tile_resource = TileResource :: ABOVE_AVERAGE;
    }
    else {
        tile_resource = TileResource :: GOOD;
    }

    //  3. call alternate method
    this->setTileResource(tile_resource);

    return;
}   /* setTileResource(double) */
```

4.7.3.50 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.

```
{
    this->tile_resource = tile_resource;
    this->__setResourceText();

    return;
}   /* setTileResource(TileResource) */
```

4.7.3.51 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].

```
{
    //  1. check input
    if (input_value < 0 or input_value > 1) {
        std::string error_str = "ERROR  HexTile::setTileType()  given input value is ";
        error_str += "not in the closed interval [0, 1]";

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. convert input value to tile type
    TileType tile_type;

    if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[0]) {
        tile_type = TileType :: LAKE;
    }
    else if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[1]) {
        tile_type = TileType :: PLAINS;
    }
    else if (input_value <= TILE_TYPE_CUMULATIVE_PROBABILITIES[2]) {
        tile_type = TileType :: FOREST;
    }
    else {
        tile_type = TileType :: MOUNTAINS;
    }

    //  3. call alternate method
    this->setTileType(tile_type);

    return;
}   /* setTileType(double) */
```

4.7.3.52 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.

```
{
    this->tile_type = tile_type;

    switch (this->tile_type) {
        case (TileType :: FOREST): {
            this->tile_sprite.setFillColor(FOREST_GREEN);

            break;
        }

        case (TileType :: LAKE): {
            this->tile_sprite.setFillColor(LAKE_BLUE);

            break;
        }

        case (TileType :: MOUNTAINS): {
            this->tile_sprite.setFillColor(MOUNTAINS_GREY);

            break;
        }

        case (TileType :: OCEAN): {
            this->tile_sprite.setFillColor(OCEAN_BLUE);

            break;
        }

        case (TileType :: PLAINS): {
            this->tile_sprite.setFillColor(PLAINS_YELLOW);

            break;
        }

        default: {
            // do nothing!

            break;
        }
    }

    this->__setUpBuildMenu();

    return;
}   /* setTileType(TileType) */
```

4.7.3.53 toggleResourceOverlay()

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

Method to toggle the tile resource overlay.

```
{
    if (this->show_resource) {
        this->show_resource = false;
    }
    else {
        this->show_resource = true;
    }

    return;
}   /* toggleResourceOverlay() */
```

4.7.4 Member Data Documentation

4.7.4.1 assets_manager_ptr

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

A pointer to the assets manager.

4.7.4.2 build_menu_backing

```
sf::RectangleShape HexTile::build_menu_backing
```

A backing for the tile build menu.

4.7.4.3 build_menu_backing_text

```
sf::Text HexTile::build_menu_backing_text
```

A text label for the build menu.

4.7.4.4 build_menu_open

```
bool HexTile::build_menu_open
```

A boolean which indicates if the tile build menu is open.

4.7.4.5 build_menu_options_text_vec

```
std::vector<sf::Text> HexTile::build_menu_options_text_vec
```

A vector of text for the tile build options.

4.7.4.6 build_menu_options_vec

```
std::vector<std::vector<sf::Sprite> > HexTile::build_menu_options_vec
```

A vector of sprites for illustrating the tile build options.

4.7.4.7 credits

```
int HexTile::credits
```

The current balance of credits.

4.7.4.8 decoration_cleared

```
bool HexTile::decoration_cleared
```

A boolean which indicates if the tile decoration has been cleared.

4.7.4.9 draw_explosion

```
bool HexTile::draw_explosion
```

A boolean which indicates whether or not to draw a tile explosion.

4.7.4.10 event_ptr

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

A pointer to the event class.

4.7.4.11 explosion_frame

```
size_t HexTile::explosion_frame
```

The current frame of the explosion animation.

4.7.4.12 explosion_sprite_reel

```
std::vector<sf::Sprite> HexTile::explosion_sprite_reel
```

A reel of sprites for a tile explosion animation.

4.7.4.13 frame

```
unsigned long long int HexTile::frame
```

The current frame of this object.

4.7.4.14 game_phase

```
std::string HexTile::game_phase
```

The current phase of the game.

4.7.4.15 has_improvement

```
bool HexTile::has_improvement
```

A boolean which indicates if tile has improvement or not.

4.7.4.16 is_selected

```
bool HexTile::is_selected
```

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

4.7.4.17 magnifying_glass_sprite

```
sf::Sprite HexTile::magnifying_glass_sprite
```

A magnifying glass sprite.

4.7.4.18 major_radius

```
double HexTile::major_radius
```

The radius of the smallest bounding circle.

4.7.4.19 message_hub_ptr

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

A pointer to the message hub.

4.7.4.20 minor_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

4.7.4.21 node_sprite

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

A circle shape to mark the tile node.

4.7.4.22 position_x

```
double HexTile::position_x
```

The x position of the tile.

4.7.4.23 position_y

```
double HexTile::position_y
```

The y position of the tile.

4.7.4.24 render_window_ptr

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

A pointer to the render window.

4.7.4.25 resource_assessed

```
bool HexTile::resource_assessed
```

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

4.7.4.26 resource_assessment

```
bool HexTile::resource_assessment
```

A boolean which triggers a resource assessment notification.

4.7.4.27 resource_chip_sprite

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

A circle shape which represents a resource chip.

4.7.4.28 resource_text

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

A text representation of the resource.

4.7.4.29 scrap_improvement_frame

```
int HexTile::scrap_improvement_frame
```

A frame for key-hold to confirm scrapping.

4.7.4.30 select_outline_sprite

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

A convex shape which outlines the tile when selected.

4.7.4.31 show_node

```
bool HexTile::show_node
```

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

4.7.4.32 show_resource

```
bool HexTile::show_resource
```

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

4.7.4.33 tile_decoration_sprite

```
sf::Sprite HexTile::tile_decoration_sprite
```

A tile decoration sprite.

4.7.4.34 tile_improvement_ptr

```
TileImprovement* HexTile::tile_improvement_ptr
```

A pointer to the improvement for this tile.

4.7.4.35 tile_resource

```
TileResource HexTile::tile_resource
```

The renewable resource quality of the tile.

4.7.4.36 tile_sprite

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

A convex shape which represents the tile.

4.7.4.37 tile_type

```
TileType HexTile::tile_type
```

The terrain type of the tile.

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

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

4.8 Message Struct Reference

A structure which defines a standard message format.

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

Public Attributes

- std::string channel = ""

  A string identifying the appropriate channel for this message.
- std::string subject = ""

  A string describing the message subject.
- std::map< std::string, bool > bool_payload = {}

  A boolean payload.
- std::map< std::string, int > int_payload = {}

  An int payload.
- std::map< std::string, double > double_payload = {}

  A double payload.
- std::map< std::string, std::vector< double > > vector_payload = {}

  A vector (double) payload.
- std::map< std::string, std::string > string_payload = {}

  A string payload.

4.8.1 Detailed Description

A structure which defines a standard message format.

4.8.2 Member Data Documentation

4.8.2.1 bool_payload

```
std::map<std::string, bool> Message::bool_payload = {}
```

A boolean payload.

4.8.2.2 channel

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

A string identifying the appropriate channel for this message.

4.8.2.3 double_payload

```
std::map<std::string, double> Message::double_payload = {}
```

A double payload.

4.8.2.4 int_payload

```
std::map<std::string, int> Message::int_payload = {}
```

An int payload.

4.8.2.5 string_payload

```
std::map<std::string, std::string> Message::string_payload = {}
```

A string payload.

4.8.2.6 subject

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

A string describing the message subject.

4.8.2.7 vector_payload

```
std::map<std::string, std::vector<double> > Message::vector_payload = {}
```

A vector (double) payload.

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

- header/ESC_core/MessageHub.h

4.9 MessageHub Class Reference

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

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

Public Member Functions

- MessageHub (void)

    Constructor for the MessageHub class.
- bool hasTraffic (void)

    Method to determine if there remains any message traffic.
- void addChannel (std::string)

    Method to add channel to message map.
- void removeChannel (std::string)

    Method to remove channel from message map.
- void printState (void)

    Method for printing message hub state information (mostly for troubleshooting message deadlocks).
- void sendMessage (Message)

    Method to send a message to the message map. Channels are implemented in a first in, first out manner (i.e. message queue).
- bool isEmpty (std::string)

    Method to check if channel is empty.
- Message receiveMessage (std::string)

    Method to receive the first message in the channel. Channels are implemented in a first in, first out manner (i.e. message queue).
- void popMessage (std::string)

    Method to pop first message off of the given channel. Channels are implemented in a first in, first out manner (i.e. message queue).
- void clearMessages (void)

    Method to clear messages from the MessageHub.
- void clear (void)

    Method to clear the MessageHub.
- ∼MessageHub (void)

    Destructor for the MessageHub class.

Private Attributes

- std::map< std::string, std::list< Message > > message_map

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

4.9.1 Detailed Description

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

4.9.2 Constructor & Destructor Documentation

4.9.2.1 MessageHub()

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

Constructor for the MessageHub class.

```
{
    //...

    std::cout « "MessageHub constructed at " « this « std::endl;

    return;
}   /* MessageHub() */
```

4.9.2.2 ∼MessageHub()

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

Destructor for the MessageHub class.

```
{
    this->clear();

    std::cout « "MessageHub at " « this « " destroyed" « std::endl;

    return;
}   /* ~MessageHub() */
```

4.9.3 Member Function Documentation

4.9.3.1 addChannel()

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

Method to add channel to message map.

Parameters

channel	The key for the message channel being added.

```
{
    //  1. check if channel is in map (if so, throw error)
    if (this->message_map.count(channel) > 0) {
        std::string error_str = "ERROR  MessageHub::addChannel()  channel ";
        error_str += channel;
        error_str += " is already in message map";

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. add channel to map
    this->message_map[channel] = {};
    std::cout « "Channel " « channel « " added to message hub" « std::endl;

    return;
}   /* addChannel() */
```

4.9.3.2 clear()

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

Method to clear the MessageHub.

```
{

    this->clearMessages();
    this->message_map.clear();

    return;
}   /* clear() */
```

4.9.3.3 clearMessages()

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

Method to clear messages from the MessageHub.

```
{
    std::map<std::string, std::list<Message»::iterator map_iter;
    for (
        map_iter = this->message_map.begin();
        map_iter != this->message_map.end();
        map_iter++
    ) {
        map_iter->second.clear();
    }

    return;
}   /* clearMessages() */
```

4.9.3.4 hasTraffic()

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

Method to determine if there remains any message traffic.

```
{
    std::map<std::string, std::list<Message»::iterator map_iter;
    for (
        map_iter = this->message_map.begin();
        map_iter != this->message_map.end();
        map_iter++
    ) {
        if (not map_iter->second.empty()) {
            return true;
        }
    }

    return false;
}   /* hasTraffic() */
```

4.9.3.5 isEmpty()

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

Method to check if channel is empty.

Parameters

channel	The key for the message channel being checked.

Returns

A boolean indicating whether the channel is empty or not.

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

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    if (this->message_map[channel].empty()) {
        return true;
    }
    else {
        return false;
    }
}   /* isEmpty() */
```

4.9.3.6 popMessage()

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

Method to pop first message off of the given channel. Channels are implemented in a first in, first out manner (i.e. message queue).

Parameters

channel	The key for the message channel being popped.

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

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. check if channel is empty (if so, throw error)
    if (this->message_map[channel].empty()) {
        std::string error_str = "ERROR  MessageHub::receiveMessage()  channel ";
        error_str += channel;
        error_str += " is empty";

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  3. pop message
    this->message_map[channel].pop_front();

    return;
}   /* popMessage() */
```

4.9.3.7 printState()

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

Method for printing message hub state information (mostly for troubleshooting message deadlocks).

```
{
    std::cout « "\n\n    **** MESSAGE HUB STATE ****    \n" « std::endl;

    std::map<std::string, std::list<Message»::iterator channel_iterator;

    for (
        channel_iterator = this->message_map.begin();
        channel_iterator != this->message_map.end();
        channel_iterator++
    ) {
        std::string channel = channel_iterator->first;
        std::list<Message> message_queue = channel_iterator->second;

        std::cout « "--------------------------------" «std::endl;
        std::cout « "\tCHANNEL: " « channel « std::endl;
        std::cout « "\tMESSAGE QUEUE LENGTH: " « message_queue.size() « std::endl;
        std::cout « std::endl;

        std::list<Message>::iterator message_queue_iterator;

        for (
            message_queue_iterator = message_queue.begin();
            message_queue_iterator != message_queue.end();
            message_queue_iterator++
        ) {
            std::cout « "\tSUBJECT: " « (*message_queue_iterator).subject «
                std::endl;
        }

        std::cout « std::endl;
    }

    std::cout « std::endl;

    return;
}   /* printState() */
```

4.9.3.8 receiveMessage()

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

Method to receive the first message in the channel. Channels are implemented in a first in, first out manner (i.e. message queue).

Parameters

channel	The key for the message channel being received from.

Returns

The first message in the given channel.

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

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. check if channel is empty (if so, throw error)
    if (this->message_map[channel].empty()) {
        std::string error_str = "ERROR  MessageHub::receiveMessage()  channel ";
        error_str += channel;
        error_str += " is empty";

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  3. receive message
    Message message = this->message_map[channel].front();

    return message;
}   /* receiveMessage() */
```

4.9.3.9 removeChannel()

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

Method to remove channel from message map.

Parameters

channel	The key for the message channel being removed.

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

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. remove channel from map
    this->message_map[channel].clear();
    this->message_map.erase(channel);

    std::cout « "Channel " « channel « " removed from message hub" « std::endl;

    return;
}   /* removeChannel() */
```

4.9.3.10 sendMessage()

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

Method to send a message to the message map. Channels are implemented in a first in, first out manner (i.e. message queue).

Parameters

message	The message to be sent.

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

    if (this->message_map.count(channel) <= 0) {
        std::string error_str = "ERROR  MessageHub::sendMessage()  channel ";
        error_str += channel;
        error_str += " is not in message map";

        #ifdef _WIN32
            std::cout « error_str « std::endl;
        #endif  /* _WIN32 */

        throw std::runtime_error(error_str);
    }

    //  2. send message to message map
    this->message_map[channel].push_back(message);

    return;
}   /* sendMessage() */
```

4.9.4 Member Data Documentation

4.9.4.1 message_map

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

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

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

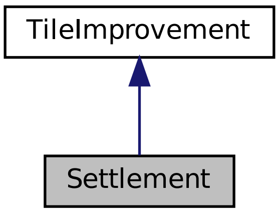
- header/ESC_core/MessageHub.h
- source/ESC_core/MessageHub.cpp

4.10 Settlement Class Reference

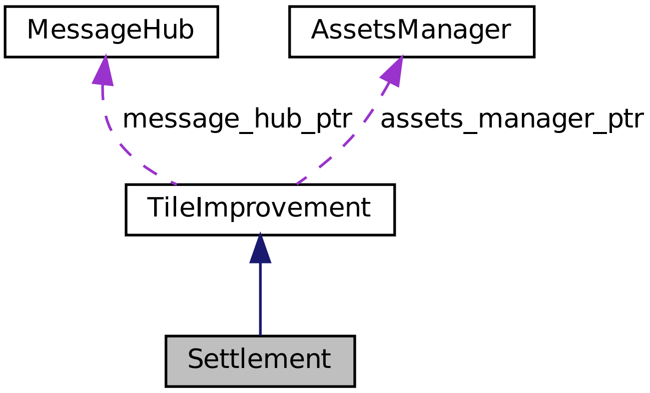
A settlement class (child class of TileImprovement).

```
#include <Settlement.h>
```

Inheritance diagram for Settlement:



Collaboration diagram for Settlement:



Public Member Functions

- Settlement (double, double, int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

  Constructor for the Settlement class.
- void setIsSelected (bool)

  Method to set the is selected attribute.
- std::string getTileOptionsSubstring (void)

  Helper method to assemble and return tile options substring.
- void processEvent (void)

  Method to process Settlement. To be called once per event.
- void processMessage (void)

  Method to process Settlement. To be called once per message.
- void draw (void)

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

  Destructor for the Settlement class.

Public Attributes

- bool draw_coin

  Boolean indicating whether or not to draw credits earned coin.
- double smoke_da

  The per frame delta in smoke particle alpha value.
- double smoke_dx

  The per frame delta in smoke particle x position.
- double smoke_dy

  The per frame delta in smoke particle y position.
- double smoke_prob

  The probability of spawning a new smoke prob in any given frame.
- std::list< sf::Sprite > smoke_sprite_list

  A list of smoke sprite (for chimney animation).
- sf::Sprite coin_sprite

  A coin sprite (for credits earned animation).

Private Member Functions

- void __setUpTileImprovementSpriteStatic (void)

  Helper method to set up tile improvement sprite (static).
- void __setUpCoinSprite (void)

  Helper method to set up and place coin sprite.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.

Additional Inherited Members

4.10.1 Detailed Description

A settlement class (child class of TileImprovement).

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Settlement()

```
Settlement::Settlement (
            double position_x,
            double position_y,
            int tile_resource,
            sf::Event * event_ptr,
            sf::RenderWindow * render_window_ptr,
            AssetsManager * assets_manager_ptr,
            MessageHub * message_hub_ptr )
```

Constructor for the Settlement class.

Ref: Wikipedia [2023]

Parameters

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

```
  :
TileImprovement(
    position_x,
    position_y,
    tile_resource,
    event_ptr,
    render_window_ptr,
    assets_manager_ptr,
    message_hub_ptr
)
{
    //  1. set attributes

    //  1.1. private
    //...

    //  1.2. public
    this->tile_improvement_type = TileImprovementType :: SETTLEMENT;

    this->draw_coin = false;

    this->smoke_da = SECONDS_PER_FRAME / 4;
    this->smoke_dx = 5 * SECONDS_PER_FRAME;
    this->smoke_dy = -10 * SECONDS_PER_FRAME;
    this->smoke_prob = 3 * SECONDS_PER_FRAME;

    this->smoke_sprite_list = {};

    this->tile_improvement_string = "SETTLEMENT";

    this->__setUpTileImprovementSpriteStatic();
    this->__setUpCoinSprite();

    this->message_hub_ptr->addChannel(SETTLEMENT_CHANNEL);

    std::cout « "Settlement constructed at " « this « std::endl;

    return;
}   /* Settlement() */
```

4.10.2.2 ∼Settlement()

```
Settlement::∼Settlement (
              void  )  [virtual]
```

Destructor for the Settlement class.

```
{
    std::cout « "Settlement at " « this « " destroyed" « std::endl;

    return;
}   /* ~Settlement() */
```

4.10.3 Member Function Documentation

4.10.3.1 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->key.code) {
        //...


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.10.3.2 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.10.3.3 __setUpCoinSprite()

```
void Settlement::__setUpCoinSprite (
              void ) [private]
```

Helper method to set up and place coin sprite.

```
{
    this->coin_sprite.setTexture(
        *(this->assets_manager_ptr->getTexture("coin"))
    );

    this->coin_sprite.setOrigin(
        this->coin_sprite.getLocalBounds().width / 2,
        this->coin_sprite.getLocalBounds().height / 2
    );

    this->coin_sprite.setPosition(this->position_x, this->position_y);

    return;
}   /* __setUpCoinSprite() */
```

4.10.3.4 __setUpTileImprovementSpriteStatic()

```
void Settlement::__setUpTileImprovementSpriteStatic (
              void ) [private]
```

Helper method to set up tile improvement sprite (static).

```
{
    this->tile_improvement_sprite_static.setTexture(
        *(this->assets_manager_ptr->getTexture("brick_house_64x64_1"))
    );

    this->tile_improvement_sprite_static.setOrigin(
        this->tile_improvement_sprite_static.getLocalBounds().width / 2,
        this->tile_improvement_sprite_static.getLocalBounds().height
    );

    this->tile_improvement_sprite_static.setPosition(
        this->position_x,
        this->position_y - 32
    );

    this->tile_improvement_sprite_static.setColor(
        sf::Color(255, 255, 255, 0)
    );

    return;
}   /* __setUpTileImprovementSpriteStatic() */
```

4.10.3.5 draw()

```
void Settlement::draw (
              void ) [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    //  1. if just built, call base method and return
    if (this->just_built) {
        TileImprovement :: draw();

        return;
    }

    //  2. draw static sprite and chimney smoke effects
    this->render_window_ptr->draw(this->tile_improvement_sprite_static);

    std::list<sf::Sprite>::iterator iter = this->smoke_sprite_list.begin();

    double alpha = 255;

    while (iter != this->smoke_sprite_list.end()) {
        this->render_window_ptr->draw(*iter);

        alpha = (*iter).getColor().a;

        alpha -= this->smoke_da;

        if (alpha <= 0) {
            iter = this->smoke_sprite_list.erase(iter);
            continue;
        }

        (*iter).setColor(sf::Color(255, 255, 255, alpha));

        (*iter).move(
            this->smoke_dx + 2 * (((double)rand() / RAND_MAX) - 1) / FRAMES_PER_SECOND,
            this->smoke_dy
        );

        (*iter).rotate(((double)rand() / RAND_MAX));

        iter++;
    }


    if ((double)rand() / RAND_MAX < smoke_prob) {
        this->smoke_sprite_list.push_back(
            sf::Sprite(*(this->assets_manager_ptr->getTexture("emissions")))
        );

        this->smoke_sprite_list.back().setOrigin(
            this->smoke_sprite_list.back().getLocalBounds().width / 2,
            this->smoke_sprite_list.back().getLocalBounds().height / 2
        );

        this->smoke_sprite_list.back().setPosition(
            this->position_x + 9 + 4 * ((double)rand() / RAND_MAX) - 2,
            this->position_y - 33
        );
    }



    //  4. draw coin
    if (this->draw_coin) {
        double alpha = this->coin_sprite.getColor().a;

        alpha -= this->smoke_da;

        if (alpha <= 0) {
            this->coin_sprite.setColor(sf::Color(255, 255, 255, 255));
            this->coin_sprite.setPosition(this->position_x, this->position_y);
            this->draw_coin = false;
        }

        this->coin_sprite.move(0, this->smoke_dy);
        this->coin_sprite.setColor(sf::Color(255, 255, 255, alpha));

        this->render_window_ptr->draw(this->coin_sprite);
    }

    this->frame++;
    return;
}   /* draw() */
```

4.10.3.6 getTileOptionsSubstring()

```
std::string Settlement::getTileOptionsSubstring (
            void  )  [virtual]
```

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

Reimplemented from TileImprovement.

```
{
    //                   32 char x 17 line console "--------------------------------\n";
    std::string options_substring                = "  **** SETTLEMENT OPTIONS ****  \n";
    options_substring                           += "                                \n";
    options_substring                           += "                                \n";
    options_substring                           += "                                \n";
    options_substring                           += "                                \n";
    options_substring                           += "                                \n";
    options_substring                           += "                                \n";
    options_substring                           += "                                \n";

    return options_substring;
}   /* getTileOptionsSubstring() */
```

4.10.3.7 processEvent()

```
void Settlement::processEvent (
              void   )   [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processEvent();

    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.10.3.8 processMessage()

```
void Settlement::processMessage (
              void   )   [virtual]
```

Method to process Settlement. To be called once per message.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processMessage();

    if (not this->message_hub_ptr->isEmpty(SETTLEMENT_CHANNEL)) {
        Message settlement_message = this->message_hub_ptr->receiveMessage(
            SETTLEMENT_CHANNEL
        );

        if (settlement_message.subject == "credits earned") {
            this->draw_coin = true;
            this->assets_manager_ptr->getSound("coin ring")->play();

            std::cout « "Credits earned message received by " « this « std::endl;
            this->message_hub_ptr->popMessage(SETTLEMENT_CHANNEL);
        }
    }

    return;
}   /* processMessage() */
```

4.10.3.9 setIsSelected()

```
void Settlement::setIsSelected (
            bool is_selected )  [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: setIsSelected(is_selected);

    if (this->is_selected) {
        this->assets_manager_ptr->getSound("people and children")->play();
    }

    return;
}   /* setIsSelected() */
```

4.10.4 Member Data Documentation

4.10.4.1 coin_sprite

```
sf::Sprite Settlement::coin_sprite
```

A coin sprite (for credits earned animation).

4.10.4.2 draw_coin

```
bool Settlement::draw_coin
```

Boolean indicating whether or not to draw credits earned coin.

4.10.4.3 smoke_da

```
double Settlement::smoke_da
```

The per frame delta in smoke particle alpha value.

4.10.4.4 smoke_dx

```
double Settlement::smoke_dx
```

The per frame delta in smoke particle x position.

4.10.4.5 smoke_dy

```
double Settlement::smoke_dy
```

The per frame delta in smoke particle y position.

4.10.4.6 smoke_prob

```
double Settlement::smoke_prob
```

The probability of spawning a new smoke prob in any given frame.

4.10.4.7 smoke_sprite_list

```
std::list<sf::Sprite> Settlement::smoke_sprite_list
```

A list of smoke sprite (for chimney animation).

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

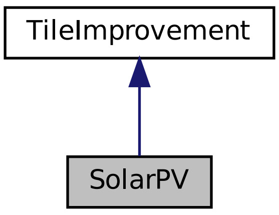
- header/Settlement.h
- source/Settlement.cpp

4.11 SolarPV Class Reference

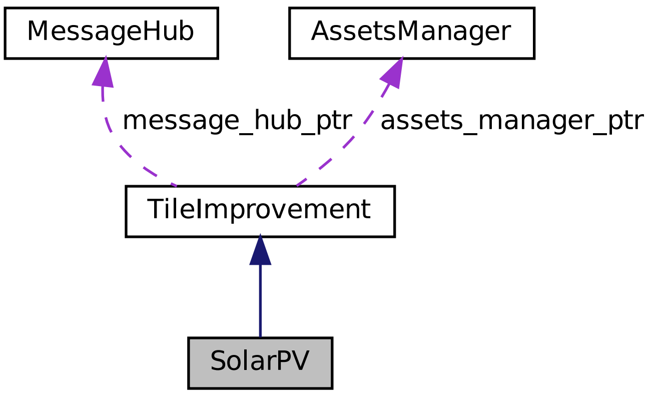
A settlement class (child class of TileImprovement).

```
#include <SolarPV.h>
```

Inheritance diagram for SolarPV:



Collaboration diagram for SolarPV:



Public Member Functions

- SolarPV (double, double, int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

  Constructor for the SolarPV class.
- std::string getTileOptionsSubstring (void)

  Helper method to assemble and return tile options substring.
- void setIsSelected (bool)

  Method to set the is selected attribute.
- void advanceTurn (void)

Method to handle turn advance.

- void update (void)

  Method to trigger production and dispatchable updates.
- void processEvent (void)

  Method to process SolarPV. To be called once per event.
- void processMessage (void)

  Method to process SolarPV. To be called once per message.
- void draw (void)

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

  Destructor for the SolarPV class.

Public Attributes

- int capacity_kW

  The rated production capacity [kW] of the solar PV array.
- int production_MWh

  The current production [MWh] of the solar PV array.
- int dispatch_MWh

  The current dispatch [MWh] of the solar PV array.
- int dispatchable_MWh

  The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).
- double max_daily_production_MWh

  The maximum daily production [MWh] of the solar PV array.
- std::vector< double > capacity_factor_vec

  A vector of daily capacity factors for the current month.
- std::vector< double > production_vec_MWh

  A vector of daily production [MWh] for the current month.
- std::vector< double > dispatch_vec_MWh

  A vector of daily dispatch [MWh] for the current month.

Private Member Functions

- void __setUpTileImprovementSpriteStatic (void)

  Helper method to set up tile improvement sprite (static).
- void __drawProductionMenu (void)

  Helper method to draw production menu assets.
- void __upgradePowerCapacity (void)

  Helper method to upgrade power capacity.
- void __computeProductionCosts (void)

  Helper method to compute production costs (O&M) based on current production level.
- void __breakdown (void)

  Helper method to trigger an equipment breakdown.
- void __computeCapacityFactors (void)

  Helper method to compute capacity factors.
- void __computeProduction (void)

  Helper method to compute production values.
- void __computeDispatch (void)

  Helper method to compute dispatch values.

- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.
- void __drawUpgradeOptions (void)

  Helper method to set up and draw upgrade options.
- void __sendImprovementStateMessage (void)

  Helper method to format and sent improvement state message.

Additional Inherited Members

4.11.1 Detailed Description

A settlement class (child class of TileImprovement).

4.11.2 Constructor & Destructor Documentation

4.11.2.1 SolarPV()

```
SolarPV::SolarPV (
            double position_x,
            double position_y,
            int tile_resource,
            sf::Event * event_ptr,
            sf::RenderWindow * render_window_ptr,
            AssetsManager * assets_manager_ptr,
            MessageHub * message_hub_ptr )
```

Constructor for the SolarPV class.

Ref: Wikipedia [2023]

Parameters

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

```
  :
TileImprovement(
    position_x,
    position_y,
    tile_resource,
    event_ptr,
    render_window_ptr,
    assets_manager_ptr,
    message_hub_ptr
)
{
    //  1. set attributes

    //  1.1. private
    //...

    //  1.2. public
    this->tile_improvement_type = TileImprovementType :: SOLAR_PV;

    this->is_running = false;

    this->health = 100;

    this->capacity_kW = 100;
    this->upgrade_level = 1;

    this->storage_kWh = 0;
    this->storage_level = 0;

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->capacity_factor_vec.resize(30, 0);
    this->production_vec_MWh.resize(30, 0);
    this->dispatch_vec_MWh.resize(30, 0);

    this->tile_improvement_string = "SOLAR PV ARRAY";

    this->__setUpTileImprovementSpriteStatic();
    this->__computeCapacityFactors();
    this->update();

    std::cout « "SolarPV constructed at " « this « std::endl;

    return;
}   /* SolarPV() */
```

4.11.2.2 ∼SolarPV()

```
SolarPV::∼SolarPV (
              void   )  [virtual]
```

Destructor for the SolarPV class.

```
{
    std::cout « "SolarPV at " « this « " destroyed" « std::endl;

    return;
}   /* ~SolarPV() */
```

4.11.3 Member Function Documentation

4.11.3.1 __breakdown()

```
void SolarPV::__breakdown (
              void   )  [private]
```

Helper method to trigger an equipment breakdown.

```
{
    TileImprovement :: __breakdown();

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;
    this->operation_maintenance_cost = 0;

    return;
}   /* __breakdown() */
```

4.11.3.2 __computeCapacityFactors()

```
void SolarPV::__computeCapacityFactors (
            void ) [private]
```

Helper method to compute capacity factors.

```
{
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator(seed);

    double mean =
        this->tile_resource_scalar * MEAN_DAILY_SOLAR_CAPACITY_FACTORS[this->month - 1];

    double stdev = STDEV_DAILY_SOLAR_CAPACITY_FACTORS[this->month - 1];

    if (this->tile_resource_scalar > 1) {
        stdev /= this->tile_resource_scalar;
    }

    std::normal_distribution<double> normal_dist(mean, stdev);

    double capacity_factor = 0;

    for (int i = 0; i < 30; i++) {
        capacity_factor = normal_dist(generator);

        if (capacity_factor < 0) {
            capacity_factor = 0;
        }

        this->capacity_factor_vec[i] = capacity_factor;
    }

    return;
}   /* __computeCapacityFactors() */
```

4.11.3.3 __computeDispatch()

```
void SolarPV::__computeDispatch (
            void ) [private]
```

Helper method to compute dispatch values.

```
{
    double stored_energy_MWh = 0;
    double storage_capacity_MWh = (double)(this->storage_kWh) / 1000;

    double demand_MWh = 0;
    double production_MWh = 0;
    double dispatchable_MWh = 0;
    double difference_MWh = 0;

    double room_MWh = 0;

    for (int i = 0; i < 30; i++) {
        demand_MWh = this->demand_vec_MWh[i];
        production_MWh = this->production_vec_MWh[i];

        if (production_MWh <= demand_MWh) {
            this->dispatch_vec_MWh[i] = production_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = demand_MWh - production_MWh;

            if ((storage_capacity_MWh > 0) and (stored_energy_MWh > 0)) {
                if (difference_MWh > stored_energy_MWh) {
                    this->dispatch_vec_MWh[i] += stored_energy_MWh;
                    dispatchable_MWh += stored_energy_MWh;
                    stored_energy_MWh = 0;
                }

                else {
                    this->dispatch_vec_MWh[i] += difference_MWh;
                    dispatchable_MWh += difference_MWh;
                    stored_energy_MWh -= difference_MWh;
                }
            }
        }

        else {
            this->dispatch_vec_MWh[i] = demand_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = production_MWh - demand_MWh;

            if (
                (storage_capacity_MWh > 0) and
                (stored_energy_MWh < storage_capacity_MWh)
            ) {
                room_MWh = storage_capacity_MWh - stored_energy_MWh;

                if (difference_MWh > room_MWh) {
                    stored_energy_MWh += room_MWh;
                }

                else {
                    stored_energy_MWh += difference_MWh;
                }
            }
        }
    }

    this->dispatchable_MWh = round(dispatchable_MWh);

    if (this->dispatch_MWh > this->dispatchable_MWh) {
        this->dispatch_MWh = this->dispatchable_MWh;
    }

    return;
}   /* __computeDispatch() */
```

4.11.3.4 __computeProduction()

```
void SolarPV::__computeProduction (
              void   )  [private]
```

Helper method to compute production values.

```
{
    double production_MWh = 0;

    for (int i = 0; i < 30; i++) {
        this->production_vec_MWh[i] =
            this->max_daily_production_MWh * this->capacity_factor_vec[i];

        production_MWh += this->production_vec_MWh[i];
    }

    this->production_MWh = round(production_MWh);

    return;
}   /* __computeProduction() */
```

4.11.3.5 __computeProductionCosts()

```
void SolarPV::__computeProductionCosts (
            void ) [private]
```

Helper method to compute production costs (O&M) based on current production level.

```
{
    double operation_maintenance_cost =
        (this->production_MWh * SOLAR_OP_MAINT_COST_PER_MWH_PRODUCTION) / 1000;
    this->operation_maintenance_cost = round(operation_maintenance_cost);

    return;
}   /* __computeProductionCosts() */
```

4.11.3.6 __drawProductionMenu()

```
void SolarPV::__drawProductionMenu (
            void ) [private]
```

Helper method to draw production menu assets.

```
{
    //  1. draw static sprite
    sf::Vector2f initial_position = this->tile_improvement_sprite_static.getPosition();
    this->tile_improvement_sprite_static.setPosition(400 - 138, 400 + 16);

    sf::Color initial_colour = this->tile_improvement_sprite_static.getColor();
    this->tile_improvement_sprite_static.setColor(sf::Color(255, 255, 255, 255));

    sf::Vector2f initial_scale = this->tile_improvement_sprite_static.getScale();
    this->tile_improvement_sprite_static.setScale(sf::Vector2f(1, 1));

    this->render_window_ptr->draw(this->tile_improvement_sprite_static);

    this->tile_improvement_sprite_static.setPosition(initial_position);
    this->tile_improvement_sprite_static.setColor(initial_colour);
    this->tile_improvement_sprite_static.setScale(initial_scale);

    //  2. draw production text
    std::string production_string = "[W]:  INCREASE DISPATCH\n";
    production_string            += "[S]:  DECREASE DISPATCH\n";
    production_string            += "                         \n";

    production_string            += "DISPATCH:  ";
    production_string            += std::to_string(this->dispatch_MWh);
    production_string            += " MWh (MAX ";
    production_string            += std::to_string(this->dispatchable_MWh);
    production_string            += ")\n";

    production_string            += "O&M COST:  ";
    production_string            += std::to_string(this->operation_maintenance_cost);
    production_string            += " K\n";

    sf::Text production_text(
        production_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    production_text.setOrigin(production_text.getLocalBounds().width / 2,0);
    production_text.setFillColor(MONOCHROME_TEXT_GREEN);

    production_text.setPosition(400 + 30, 400 - 45);

    this->render_window_ptr->draw(production_text);

    return;
}   /* __drawProductionMenu() */
```

4.11.3.7 __drawUpgradeOptions()

```
void SolarPV::__drawUpgradeOptions (
            void ) [private]
```

Helper method to set up and draw upgrade options.

```
{
    //  1. draw power capacity upgrade sprite
    sf::Vector2f initial_position = this->tile_improvement_sprite_static.getPosition();
    this->tile_improvement_sprite_static.setPosition(400 - 100, 400 - 32);

    sf::Color initial_colour = this->tile_improvement_sprite_static.getColor();
    this->tile_improvement_sprite_static.setColor(sf::Color(255, 255, 255, 255));

    sf::Vector2f initial_scale = this->tile_improvement_sprite_static.getScale();
    this->tile_improvement_sprite_static.setScale(sf::Vector2f(1, 1));

    this->render_window_ptr->draw(this->tile_improvement_sprite_static);

    this->tile_improvement_sprite_static.setPosition(initial_position);
    this->tile_improvement_sprite_static.setColor(initial_colour);
    this->tile_improvement_sprite_static.setScale(initial_scale);

    this->render_window_ptr->draw(this->upgrade_arrow_sprite);


    //  2. draw power capacity upgrade text
    //                  16 char line = "                \n"
    std::string power_upgrade_string = "POWER CAPACITY  \n";
    power_upgrade_string            += "                \n";

    power_upgrade_string            += "CAPACITY:  ";
    power_upgrade_string            += std::to_string(this->capacity_kW);
    power_upgrade_string            += " kW\n";

    power_upgrade_string            += "LEVEL:     ";
    power_upgrade_string            += std::to_string(this->upgrade_level);
    power_upgrade_string            += "\n\n";

    if (this->upgrade_level < MAX_UPGRADE_LEVELS) {
        power_upgrade_string        += "[W]: + 100 kW (";
        power_upgrade_string        +=  std::to_string(SOLAR_PV_BUILD_COST);
        power_upgrade_string        += " K)\n";
    }

    else {
        power_upgrade_string        += " * MAX LEVEL *  \n";
    }

    sf::Text power_upgrade_text = sf::Text(
        power_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    power_upgrade_text.setOrigin(power_upgrade_text.getLocalBounds().width / 2, 0);
    power_upgrade_text.setPosition(400 - 100, 400 - 32 + 16);
    power_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(power_upgrade_text);


    //  3. draw energy capacity (storage) upgrade sprite
    this->render_window_ptr->draw(this->storage_upgrade_sprite);
    this->render_window_ptr->draw(this->upgrade_plus_sprite);


    //  4. draw energy capacity (storage) upgrade text
    //                   16 char line = "                \n"
    std::string energy_upgrade_string = "ENERGY CAPACITY \n";
    energy_upgrade_string            += "                \n";

    energy_upgrade_string            += "CAPACITY:  ";
    energy_upgrade_string            += std::to_string(this->storage_level * 200);
    energy_upgrade_string            += " kWh\n";

    energy_upgrade_string            += "LEVEL:     ";
    energy_upgrade_string            += std::to_string(this->storage_level);
    energy_upgrade_string            += "\n\n";

    if (this->storage_level < MAX_STORAGE_LEVELS) {
        energy_upgrade_string        += "[D]: + 200 kWh (";
        energy_upgrade_string        +=  std::to_string(ENERGY_STORAGE_SYSTEM_BUILD_COST);
        energy_upgrade_string        += " K)\n";
    }

    else {
        energy_upgrade_string += " * MAX LEVEL *  \n";
    }

    sf::Text energy_upgrade_text = sf::Text(
        energy_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    energy_upgrade_text.setOrigin(energy_upgrade_text.getLocalBounds().width / 2, 0);
    energy_upgrade_text.setPosition(400 + 100, 400 - 32 + 16);
    energy_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(energy_upgrade_text);

    return;
}   /* __drawUpgradeOptions() */
```

4.11.3.8 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::U): {
            this->__openUpgradeMenu();

            break;
        }


        case (sf::Keyboard::W): {
            if (this->production_menu_open) {
                this->dispatch_MWh++;

                if (this->dispatch_MWh > this->dispatchable_MWh) {
                    this->dispatch_MWh = 0;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            else if (this->upgrade_menu_open) {
                this->__upgradePowerCapacity();
            }

            break;
        }


        case (sf::Keyboard::S): {
            if (this->production_menu_open) {
                this->dispatch_MWh--;

                if (this->dispatch_MWh < 0) {
                    this->dispatch_MWh = this->dispatchable_MWh;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            break;
        }


        case (sf::Keyboard::D): {
            if (this->upgrade_menu_open) {
                this->__upgradeStorageCapacity();
                this->__computeProduction();
                this->__computeDispatch();
            }

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.11.3.9 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.11.3.10 __sendImprovementStateMessage()

```
void SolarPV::__sendImprovementStateMessage (
                void ) [private]
```

Helper method to format and sent improvement state message.

```
{
    Message improvement_state_message;

    improvement_state_message.channel = GAME_CHANNEL;
    improvement_state_message.subject = "improvement state";

    improvement_state_message.int_payload["dispatch_MWh"] = this->dispatch_MWh;
    improvement_state_message.int_payload["operation_maintenance_cost"] =
        this->operation_maintenance_cost;

    this->message_hub_ptr->sendMessage(improvement_state_message);

    std::cout « "Improvement state message sent by " « this « std::endl;

    return;
}   /* __sendImprovementStateMessage() */
```

4.11.3.11 __setUpTileImprovementSpriteStatic()

```
void SolarPV::__setUpTileImprovementSpriteStatic (
              void ) [private]
```

Helper method to set up tile improvement sprite (static).

```
{
    this->tile_improvement_sprite_static.setTexture(
        *(this->assets_manager_ptr->getTexture("solar PV array"))
    );

    this->tile_improvement_sprite_static.setOrigin(
        this->tile_improvement_sprite_static.getLocalBounds().width / 2,
        this->tile_improvement_sprite_static.getLocalBounds().height
    );

    this->tile_improvement_sprite_static.setPosition(
        this->position_x,
        this->position_y - 32
    );

    this->tile_improvement_sprite_static.setColor(
        sf::Color(255, 255, 255, 0)
    );

    return;
}   /* __setUpTileImprovementSpriteStatic() */
```

4.11.3.12 __upgradePowerCapacity()

```
void SolarPV::__upgradePowerCapacity (
              void ) [private]
```

Helper method to upgrade power capacity.

```
{
    if (this->credits < SOLAR_PV_BUILD_COST) {
        std::cout « "Cannot upgrade solar PV: insufficient credits (need "
            « SOLAR_PV_BUILD_COST « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    if (this->upgrade_level >= MAX_UPGRADE_LEVELS) {
        return;
    }

    this->health = 100;

    this->capacity_kW += 100;
    this->upgrade_level++;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->__computeProduction();
    this->__computeDispatch();

    this->just_upgraded = true;

    this->assets_manager_ptr->getSound("upgrade")->play();
    this->__sendCreditsSpentMessage(SOLAR_PV_BUILD_COST);
    this->__sendTileStateRequest();
    this->__sendGameStateRequest();

    return;
}   /* __upgradePowerCapacity() */
```

4.11.3.13 advanceTurn()

```
void SolarPV::advanceTurn (
            void ) [virtual]
```

Method to handle turn advance.

Reimplemented from TileImprovement.

```
{
    // 1. update
    this->__computeCapacityFactors();
    this->update();

    // 2. send improvement state message
    this->__sendImprovementStateMessage();

    // 3. handle start/stop
    if ((not this->is_running) and (this->dispatch_MWh > 0)) {
        this->is_running = true;
    }

    else if (this->is_running and (this->dispatch_MWh <= 0)) {
        this->is_running = false;
    }

    // 4. handle equipment health
    if (this->is_running) {
        this->health--;

        if (this->health <= 0) {
            this->__breakdown();
        }
    }

    // 5. send tile state request (if selected)
    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* advanceTurn() */
```

4.11.3.14 draw()

```
void SolarPV::draw (
            void ) [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    // 1. if just built, call base method and return
    if (this->just_built) {
        TileImprovement :: draw();

        return;
    }


    //  2. handle upgrade effects
    if (this->just_upgraded) {
        this->tile_improvement_sprite_static.setColor(
            sf::Color(
                255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                 255,
                 255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                 255
             )
         );

         this->tile_improvement_sprite_static.setScale(
             sf::Vector2f(
                 1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                 1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2)
             )
         );

         this->upgrade_frame++;
     }

     if (this->upgrade_frame >= 2 * FRAMES_PER_SECOND) {
         this->tile_improvement_sprite_static.setColor(
             sf::Color(255,255,255,255)
         );

         this->tile_improvement_sprite_static.setScale(sf::Vector2f(1,1));

         this->just_upgraded = false;
         this->upgrade_frame = 0;
     }


     //  3. draw static sprite
     this->render_window_ptr->draw(this->tile_improvement_sprite_static);


     //  4. draw storage upgrades
     for (size_t i = 0; i < this->storage_upgrade_sprite_vec.size(); i++) {
         this->render_window_ptr->draw(this->storage_upgrade_sprite_vec[i]);
     }


     //  5. handle dispatch illustration
     if (this->dispatch_MWh > 0) {
         this->dispatch_text.setString(std::to_string(this->dispatch_MWh));
         this->__drawDispatch();
     }


     //  6. draw production menu
     if (this->production_menu_open) {
         this->render_window_ptr->draw(this->production_menu_backing);
         this->render_window_ptr->draw(this->production_menu_backing_text);

         this->__drawProductionMenu();
     }


     //  7. draw upgrade menu
     if (this->upgrade_menu_open) {
         this->render_window_ptr->draw(this->upgrade_menu_backing);
         this->render_window_ptr->draw(this->upgrade_menu_backing_text);

         this->__drawUpgradeOptions();
     }


     //  10. handle broken effects
     if (this->is_broken) {
         this->tile_improvement_sprite_static.setColor(
             sf::Color(
                 255,
                 255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                 255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                 255
             )
         );
     }

     this->frame++;
     return;
 }   /* draw() */
```

4.11.3.15 getTileOptionsSubstring()

```
std::string SolarPV::getTileOptionsSubstring (
            void ) [virtual]
```

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

Reimplemented from TileImprovement.

```
{
    //                  32 char x 17 line console "--------------------------------\n";
    std::string options_substring                = "CAPACITY:      ";
    options_substring                           += std::to_string(this->capacity_kW);
    options_substring                           += " kW (level ";
    options_substring                           += std::to_string(this->upgrade_level);
    options_substring                           += ")\n";

    options_substring                           += "PRODUCTION:    ";
    options_substring                           += std::to_string(this->production_MWh);
    options_substring                           += " MWh\n";

    options_substring                           += "DISPATCHABLE:  ";
    options_substring                           += std::to_string(this->dispatchable_MWh);
    options_substring                           += " MWh\n";

    options_substring                           += "HEALTH:      ";
    options_substring                           += std::to_string(this->health);
    options_substring                           += "/100";

    if (this->health <= 0) {
        options_substring                       += " ** BROKEN! **\n";
    }

    else {
        options_substring                       += "\n";
    }

    options_substring                           += "                                \n";
    options_substring                           += "   **** SOLAR PV OPTIONS ****   \n";
    options_substring                           += "                                \n";

    options_substring                           += "     [E]:  ";

    if (this->is_broken) {
        options_substring                       += "*** BROKEN! ***\n";
    }

    else {
        options_substring                       += "OPEN PRODUCTION MENU\n";
    }

    options_substring                           += "     [U]:  OPEN UPGRADE MENU    \n";
    options_substring                           += "HOLD [P]:  SCRAP (";
    options_substring                           += std::to_string(SCRAP_COST);
    options_substring                           += " K)";

    return options_substring;
}   /* getTileOptionsSubstring() */
```

4.11.3.16 processEvent()

```
void SolarPV::processEvent (
            void ) [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processEvent();

    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.11.3.17 processMessage()

```
void SolarPV::processMessage (
             void  )  [virtual]
```

Method to process SolarPV. To be called once per message.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processMessage();

    //...

    return;
}   /* processMessage() */
```

4.11.3.18 setIsSelected()

```
void SolarPV::setIsSelected (
             bool is_selected )  [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: setIsSelected(is_selected);

    if (this->is_running and this->is_selected) {
        this->assets_manager_ptr->getSound("solar hum")->play();
    }

    return;
}   /* setIsSelected() */
```

4.11.3.19 update()

```
void SolarPV::update (
             void  )  [virtual]
```

Method to trigger production and dispatchable updates.

Reimplemented from TileImprovement.

```
{
    this->__computeProduction();
    this->__computeProductionCosts();
    this->__computeDispatch();

    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* update() */
```

4.11.4 Member Data Documentation

4.11.4.1 capacity_factor_vec

`std::vector<double> SolarPV::capacity_factor_vec`

A vector of daily capacity factors for the current month.

4.11.4.2 capacity_kW

`int SolarPV::capacity_kW`

The rated production capacity [kW] of the solar PV array.

4.11.4.3 dispatch_MWh

`int SolarPV::dispatch_MWh`

The current dispatch [MWh] of the solar PV array.

4.11.4.4 dispatch_vec_MWh

`std::vector<double> SolarPV::dispatch_vec_MWh`

A vector of daily dispatch [MWh] for the current month.

4.11.4.5 dispatchable_MWh

```
int SolarPV::dispatchable_MWh
```

The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).

4.11.4.6 max_daily_production_MWh

```
double SolarPV::max_daily_production_MWh
```

The maximum daily production [MWh] of the solar PV array.

4.11.4.7 production_MWh

```
int SolarPV::production_MWh
```

The current production [MWh] of the solar PV array.

4.11.4.8 production_vec_MWh

```
std::vector<double> SolarPV::production_vec_MWh
```

A vector of daily production [MWh] for the current month.

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

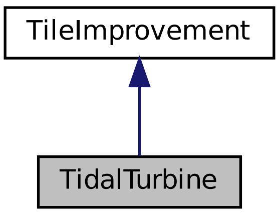
- header/SolarPV.h
- source/SolarPV.cpp

4.12 TidalTurbine Class Reference

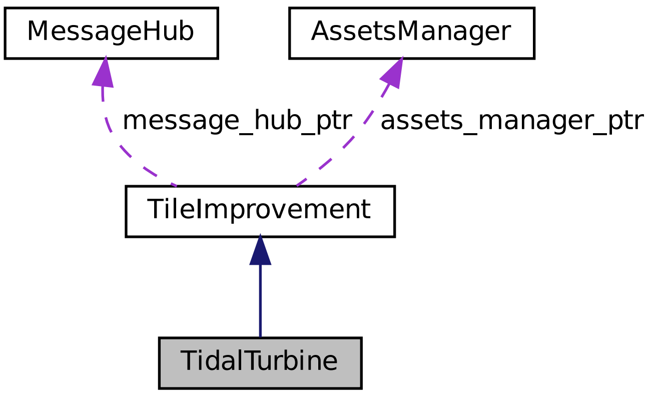
A settlement class (child class of TileImprovement).

```
#include <TidalTurbine.h>
```

Inheritance diagram for TidalTurbine:



Collaboration diagram for TidalTurbine:



Public Member Functions

- TidalTurbine (double, double, int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

  Constructor for the TidalTurbine class.
- std::string getTileOptionsSubstring (void)

  Helper method to assemble and return tile options substring.
- void setIsSelected (bool)

  Method to set the is selected attribute.
- void advanceTurn (void)

Method to handle turn advance.
- void update (void)

  Method to trigger production and dispatchable updates.
- void processEvent (void)

  Method to process TidalTurbine. To be called once per event.
- void processMessage (void)

  Method to process TidalTurbine. To be called once per message.
- void draw (void)

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

  Destructor for the TidalTurbine class.

Public Attributes

- int capacity_kW

  The rated production capacity [kW] of the solar PV array.
- int production_MWh

  The current production [MWh] of the solar PV array.
- int dispatch_MWh

  The current dispatch [MWh] of the solar PV array.
- int dispatchable_MWh

  The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).
- double max_daily_production_MWh

  The maximum daily production [MWh] of the solar PV array.
- double rotor_drotation

  The rotation rate of the rotor.
- double bobbing_y

  The bobbing extent of the tidal turbine.
- std::vector< double > capacity_factor_vec

  A vector of daily capacity factors for the current month.
- std::vector< double > production_vec_MWh

  A vector of daily production [MWh] for the current month.
- std::vector< double > dispatch_vec_MWh

  A vector of daily dispatch [MWh] for the current month.

Private Member Functions

- void __setUpTileImprovementSpriteAnimated (void)

  Helper method to set up tile improvement sprite (static).
- void __drawProductionMenu (void)

  Helper method to draw production menu assets.
- void __upgradePowerCapacity (void)

  Helper method to upgrade power capacity.
- void __computeProductionCosts (void)

  Helper method to compute production costs (O&M) based on current production level.
- void __breakdown (void)

  Helper method to trigger an equipment breakdown.
- void __computeCapacityFactors (void)

  Helper method to compute capacity factors.

- void __computeProduction (void)

  Helper method to compute production values.
- void __computeDispatch (void)

  Helper method to compute dispatch values.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.
- void __drawUpgradeOptions (void)

  Helper method to set up and draw upgrade options.
- void __sendImprovementStateMessage (void)

  Helper method to format and sent improvement state message.

Additional Inherited Members

4.12.1 Detailed Description

A settlement class (child class of TileImprovement).

4.12.2 Constructor & Destructor Documentation

4.12.2.1 TidalTurbine()

```
TidalTurbine::TidalTurbine (
            double position_x,
            double position_y,
            int tile_resource,
            sf::Event * event_ptr,
            sf::RenderWindow * render_window_ptr,
            AssetsManager * assets_manager_ptr,
            MessageHub * message_hub_ptr )
```

Constructor for the TidalTurbine class.

Ref: Wikipedia [2023]

Parameters

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

```
  :
TileImprovement(
    position_x,
    position_y,
    tile_resource,
    event_ptr,
    render_window_ptr,
    assets_manager_ptr,
    message_hub_ptr
)
{
    //  1. set attributes

    //  1.1. private
    //...

    //  1.2. public
    this->tile_improvement_type = TileImprovementType :: TIDAL_TURBINE;

    this->is_running = false;

    this->health = 100;

    this->capacity_kW = 100;
    this->upgrade_level = 1;

    this->storage_kWh = 0;
    this->storage_level = 0;

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->rotor_drotation = 64 * SECONDS_PER_FRAME;
    this->bobbing_y = 4;

    this->capacity_factor_vec.resize(30, 0);
    this->production_vec_MWh.resize(30, 0);
    this->dispatch_vec_MWh.resize(30, 0);

    this->tile_improvement_string = "TIDAL TURBINE";

    this->__setUpTileImprovementSpriteAnimated();
    this->__computeCapacityFactors();
    this->update();

    std::cout « "TidalTurbine constructed at " « this « std::endl;

    return;
}   /* TidalTurbine() */
```

4.12.2.2 ∼TidalTurbine()

```
TidalTurbine::∼TidalTurbine (
              void   )  [virtual]
```

Destructor for the TidalTurbine class.

```
{
    std::cout « "TidalTurbine at " « this « " destroyed" « std::endl;

    return;
}   /* ~TidalTurbine() */
```

4.12.3 Member Function Documentation

4.12.3.1 __breakdown()

```
void TidalTurbine::__breakdown (
            void ) [private]
```

Helper method to trigger an equipment breakdown.

```
{
    TileImprovement :: __breakdown();

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;
    this->operation_maintenance_cost = 0;

    return;
}   /* __breakdown() */
```

4.12.3.2 __computeCapacityFactors()

```
void TidalTurbine::__computeCapacityFactors (
            void ) [private]
```

Helper method to compute capacity factors.

```
{
    for (int i = 0; i < 30; i++) {
        this->capacity_factor_vec[i] =
            this->tile_resource_scalar * DAILY_TIDAL_CAPACITY_FACTOR;
    }

    return;
}   /* __computeCapacityFactors() */
```

4.12.3.3 __computeDispatch()

```
void TidalTurbine::__computeDispatch (
            void ) [private]
```

Helper method to compute dispatch values.

```
{
    double stored_energy_MWh = 0;
    double storage_capacity_MWh = (double)(this->storage_kWh) / 1000;

    double demand_MWh = 0;
    double production_MWh = 0;
    double dispatchable_MWh = 0;
    double difference_MWh = 0;

    double room_MWh = 0;

    for (int i = 0; i < 30; i++) {
        demand_MWh = this->demand_vec_MWh[i];
        production_MWh = this->production_vec_MWh[i];

        if (production_MWh <= demand_MWh) {
            this->dispatch_vec_MWh[i] = production_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = demand_MWh - production_MWh;

            if ((storage_capacity_MWh > 0) and (stored_energy_MWh > 0)) {
                if (difference_MWh > stored_energy_MWh) {
                    this->dispatch_vec_MWh[i] += stored_energy_MWh;
                    dispatchable_MWh += stored_energy_MWh;
                    stored_energy_MWh = 0;
                }

                else {
                    this->dispatch_vec_MWh[i] += difference_MWh;
                    dispatchable_MWh += difference_MWh;
                    stored_energy_MWh -= difference_MWh;
                }
            }
        }

        else {
            this->dispatch_vec_MWh[i] = demand_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = production_MWh - demand_MWh;

            if (
                (storage_capacity_MWh > 0) and
                (stored_energy_MWh < storage_capacity_MWh)
            ) {
                room_MWh = storage_capacity_MWh - stored_energy_MWh;

                if (difference_MWh > room_MWh) {
                    stored_energy_MWh += room_MWh;
                }

                else {
                    stored_energy_MWh += difference_MWh;
                }
            }
        }
    }

    this->dispatchable_MWh = round(dispatchable_MWh);

    if (this->dispatch_MWh > this->dispatchable_MWh) {
        this->dispatch_MWh = this->dispatchable_MWh;
    }

    return;
}   /* __computeDispatch() */
```

4.12.3.4 __computeProduction()

```
void TidalTurbine::__computeProduction (
              void  )  [private]
```

Helper method to compute production values.

```
{
    double production_MWh = 0;

    for (int i = 0; i < 30; i++) {
        this->production_vec_MWh[i] =
            this->max_daily_production_MWh * this->capacity_factor_vec[i];

        production_MWh += this->production_vec_MWh[i];
    }

    this->production_MWh = round(production_MWh);

    return;
}   /* __computeProduction() */
```

4.12.3.5 __computeProductionCosts()

```
void TidalTurbine::__computeProductionCosts (
              void  )  [private]
```

Helper method to compute production costs (O&M) based on current production level.

```
{
    double operation_maintenance_cost =
        (this->production_MWh * TIDAL_OP_MAINT_COST_PER_MWH_PRODUCTION) / 1000;
    this->operation_maintenance_cost = round(operation_maintenance_cost);

    return;
}   /* __computeProductionCosts() */
```

4.12.3.6 __drawProductionMenu()

```
void TidalTurbine::__drawProductionMenu (
            void ) [private]
```

Helper method to draw production menu assets.

```
{
    //  1. draw static sprite
    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
        sf::Vector2f initial_position = this->tile_improvement_sprite_animated[i].getPosition();
        this->tile_improvement_sprite_animated[i].setPosition(400 - 138, 400 + 16);

        sf::Color initial_colour = this->tile_improvement_sprite_animated[i].getColor();
        this->tile_improvement_sprite_animated[i].setColor(sf::Color(255, 255, 255, 255));

        sf::Vector2f initial_scale = this->tile_improvement_sprite_animated[i].getScale();
        this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1, 1));

        double initial_rotation = this->tile_improvement_sprite_animated[i].getRotation();
        this->tile_improvement_sprite_animated[i].setRotation(0);

        this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);

        this->tile_improvement_sprite_animated[i].setPosition(initial_position);
        this->tile_improvement_sprite_animated[i].setColor(initial_colour);
        this->tile_improvement_sprite_animated[i].setScale(initial_scale);
        this->tile_improvement_sprite_animated[i].setRotation(initial_rotation);
    }

    //  2. draw production text
    std::string production_string = "[W]:  INCREASE DISPATCH\n";
    production_string           += "[S]:  DECREASE DISPATCH\n";
    production_string           += "                        \n";

    production_string           += "DISPATCH:  ";
    production_string           += std::to_string(this->dispatch_MWh);
    production_string           += " MWh (MAX ";
    production_string           += std::to_string(this->dispatchable_MWh);
    production_string           += ")\n";

    production_string           += "O&M COST:  ";
    production_string           += std::to_string(this->operation_maintenance_cost);
    production_string           += " K\n";

    sf::Text production_text(
        production_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    production_text.setOrigin(production_text.getLocalBounds().width / 2,0);
    production_text.setFillColor(MONOCHROME_TEXT_GREEN);

    production_text.setPosition(400 + 30, 400 - 45);

    this->render_window_ptr->draw(production_text);

    return;
}   /* __drawProductionMenu() */
```

4.12.3.7 __drawUpgradeOptions()

```
void TidalTurbine::__drawUpgradeOptions (
            void ) [private]
```

Helper method to set up and draw upgrade options.

```
{
    //  1. draw power capacity upgrade sprite
    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
        sf::Vector2f initial_position = this->tile_improvement_sprite_animated[i].getPosition();
        this->tile_improvement_sprite_animated[i].setPosition(400 - 100, 400 - 32 - 8);

        sf::Color initial_colour = this->tile_improvement_sprite_animated[i].getColor();
        this->tile_improvement_sprite_animated[i].setColor(sf::Color(255, 255, 255, 255));

        sf::Vector2f initial_scale = this->tile_improvement_sprite_animated[i].getScale();
        this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1, 1));

        double initial_rotation = this->tile_improvement_sprite_animated[i].getRotation();
        this->tile_improvement_sprite_animated[i].setRotation(0);

        this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);

        this->tile_improvement_sprite_animated[i].setPosition(initial_position);
        this->tile_improvement_sprite_animated[i].setColor(initial_colour);
        this->tile_improvement_sprite_animated[i].setScale(initial_scale);
        this->tile_improvement_sprite_animated[i].setRotation(initial_rotation);
    }

    this->render_window_ptr->draw(this->upgrade_arrow_sprite);


    //  2. draw power capacity upgrade text
    //                  16 char line = "                \n"
    std::string power_upgrade_string = "POWER CAPACITY  \n";
    power_upgrade_string            += "                \n";

    power_upgrade_string            += "CAPACITY:  ";
    power_upgrade_string            += std::to_string(this->capacity_kW);
    power_upgrade_string            += " kW\n";

    power_upgrade_string            += "LEVEL:     ";
    power_upgrade_string            += std::to_string(this->upgrade_level);
    power_upgrade_string            += "\n\n";

    if (this->upgrade_level < MAX_UPGRADE_LEVELS) {
        power_upgrade_string        += "[W]: + 100 kW (";
        power_upgrade_string        +=  std::to_string(TIDAL_TURBINE_BUILD_COST);
        power_upgrade_string        += " K)\n";
    }

    else {
        power_upgrade_string        += " * MAX LEVEL *  \n";
    }

    sf::Text power_upgrade_text = sf::Text(
        power_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    power_upgrade_text.setOrigin(power_upgrade_text.getLocalBounds().width / 2, 0);
    power_upgrade_text.setPosition(400 - 100, 400 - 32 + 16);
    power_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(power_upgrade_text);


    //  3. draw energy capacity (storage) upgrade sprite
    this->render_window_ptr->draw(this->storage_upgrade_sprite);
    this->render_window_ptr->draw(this->upgrade_plus_sprite);


    //  4. draw energy capacity (storage) upgrade text
    //                   16 char line = "                \n"
    std::string energy_upgrade_string = "ENERGY CAPACITY \n";
    energy_upgrade_string            += "                \n";

    energy_upgrade_string            += "CAPACITY:  ";
    energy_upgrade_string            += std::to_string(this->storage_level * 200);
    energy_upgrade_string            += " kWh\n";

    energy_upgrade_string            += "LEVEL:     ";
    energy_upgrade_string            += std::to_string(this->storage_level);
    energy_upgrade_string            += "\n\n";

    if (this->storage_level < MAX_STORAGE_LEVELS) {
        energy_upgrade_string        += "[D]: + 200 kWh (";
        energy_upgrade_string        +=  std::to_string(ENERGY_STORAGE_SYSTEM_BUILD_COST);
        energy_upgrade_string        += " K)\n";
    }

    else {
        energy_upgrade_string += " * MAX LEVEL *  \n";
    }

    sf::Text energy_upgrade_text = sf::Text(
        energy_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    energy_upgrade_text.setOrigin(energy_upgrade_text.getLocalBounds().width / 2, 0);
    energy_upgrade_text.setPosition(400 + 100, 400 - 32 + 16);
    energy_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(energy_upgrade_text);

    return;
}   /* __drawUpgradeOptions() */
```

4.12.3.8 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::U): {
            this->__openUpgradeMenu();

            break;
        }


        case (sf::Keyboard::W): {
            if (this->production_menu_open) {
                this->dispatch_MWh++;

                if (this->dispatch_MWh > this->dispatchable_MWh) {
                    this->dispatch_MWh = 0;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            else if (this->upgrade_menu_open) {
                this->__upgradePowerCapacity();
            }

            break;
        }


        case (sf::Keyboard::S): {
            if (this->production_menu_open) {
                this->dispatch_MWh--;

                if (this->dispatch_MWh < 0) {
                    this->dispatch_MWh = this->dispatchable_MWh;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            break;
        }


        case (sf::Keyboard::D): {
            if (this->upgrade_menu_open) {
                this->__upgradeStorageCapacity();
                this->__computeProduction();
                this->__computeDispatch();
            }

            break;
        }


        default: {
            // do nothing!

```

```
                break;
            }
        }

        return;
    }   /* __handleKeyPressEvents() */
```

4.12.3.9 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.12.3.10 __sendImprovementStateMessage()

```
void TidalTurbine::__sendImprovementStateMessage (
                void  )  [private]
```

Helper method to format and sent improvement state message.

```
{
    Message improvement_state_message;

    improvement_state_message.channel = GAME_CHANNEL;
    improvement_state_message.subject = "improvement state";

    improvement_state_message.int_payload["dispatch_MWh"] = this->dispatch_MWh;
    improvement_state_message.int_payload["operation_maintenance_cost"] =
        this->operation_maintenance_cost;

    this->message_hub_ptr->sendMessage(improvement_state_message);

    std::cout « "Improvement state message sent by " « this « std::endl;

    return;
}   /* __sendImprovementStateMessage() */
```

4.12.3.11 __setUpTileImprovementSpriteAnimated()

```
void TidalTurbine::__setUpTileImprovementSpriteAnimated (
            void ) [private]
```

Helper method to set up tile improvement sprite (static).

```
{
    sf::Sprite diesel_generator_sheet(
        *(this->assets_manager_ptr->getTexture("tidal turbine"))
    );

    int n_elements = diesel_generator_sheet.getLocalBounds().height / 64;

    for (int i = 0; i < n_elements; i++) {
        this->tile_improvement_sprite_animated.push_back(
            sf::Sprite(
                *(this->assets_manager_ptr->getTexture("tidal turbine")),
                sf::IntRect(0, i * 64, 64, 64)
            )
        );

        this->tile_improvement_sprite_animated.back().setOrigin(
            this->tile_improvement_sprite_animated.back().getLocalBounds().width / 2,
            this->tile_improvement_sprite_animated.back().getLocalBounds().height
        );

        this->tile_improvement_sprite_animated.back().setPosition(
            this->position_x,
            this->position_y - 32
        );

        this->tile_improvement_sprite_animated.back().setColor(
            sf::Color(255, 255, 255, 0)
        );
    }

    return;
}   /* __setUpTileImprovementSpriteAnimated() */
```

4.12.3.12 __upgradePowerCapacity()

```
void TidalTurbine::__upgradePowerCapacity (
            void ) [private]
```

Helper method to upgrade power capacity.

```
{
    if (this->credits < TIDAL_TURBINE_BUILD_COST) {
        std::cout « "Cannot upgrade tidal turbine: insufficient credits (need "
            « TIDAL_TURBINE_BUILD_COST « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    if (this->upgrade_level >= MAX_UPGRADE_LEVELS) {
        return;
    }

    this->health = 100;

    this->capacity_kW += 100;
    this->upgrade_level++;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->__computeProduction();
    this->__computeDispatch();

    this->just_upgraded = true;

    this->assets_manager_ptr->getSound("upgrade")->play();

    this->__sendCreditsSpentMessage(TIDAL_TURBINE_BUILD_COST);
    this->__sendTileStateRequest();
    this->__sendGameStateRequest();

    return;
}   /* __upgradePowerCapacity() */
```

4.12.3.13 advanceTurn()

```
void TidalTurbine::advanceTurn (
              void  )  [virtual]
```

Method to handle turn advance.

Reimplemented from TileImprovement.

```
{
    //  1. update
    this->__computeCapacityFactors();
    this->update();

    //  2. send improvement state message
    this->__sendImprovementStateMessage();

    //  3. handle start/stop
    if ((not this->is_running) and (this->dispatch_MWh > 0)) {
        this->is_running = true;
    }

    else if (this->is_running and (this->dispatch_MWh <= 0)) {
        this->is_running = false;
    }

    //  4. handle equipment health
    if (this->is_running) {
        this->health--;

        if (this->health <= 0) {
            this->__breakdown();
        }
    }

    //  5. send tile state request (if selected)
    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* advanceTurn() */
```

4.12.3.14 draw()

```
void TidalTurbine::draw (
              void  )  [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    //  1. if just built, call base method and return
    if (this->just_built) {
        TileImprovement :: draw();

        return;
    }


     //  2. handle upgrade effects
     if (this->just_upgraded) {
         for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
             this->tile_improvement_sprite_animated[i].setColor(
                 sf::Color(
                     255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                     255,
                     255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                     255
                 )
             );

             this->tile_improvement_sprite_animated[i].setScale(
                 sf::Vector2f(
                     1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                     1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2)
                 )
             );
         }

         this->upgrade_frame++;
     }

     if (this->upgrade_frame >= 2 * FRAMES_PER_SECOND) {
         for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
             this->tile_improvement_sprite_animated[i].setColor(
                 sf::Color(255,255,255,255)
             );

             this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1,1));
         }

         this->just_upgraded = false;
         this->upgrade_frame = 0;
     }


     //  3. handle bobbing
     for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
         this->tile_improvement_sprite_animated[i].setPosition(
             this->position_x,
             this->position_y + this->bobbing_y * cos(
                 (double)(0.4 * M_PI * this->frame) / FRAMES_PER_SECOND
             )
         );
     }


     //  4. draw first element of animated sprite
     this->render_window_ptr->draw(this->tile_improvement_sprite_animated[0]);


     //  5. draw second element of animated sprite
     if (this->is_running) {
         this->tile_improvement_sprite_animated[1].rotate(this->rotor_drotation);
     }

     this->render_window_ptr->draw(this->tile_improvement_sprite_animated[1]);


     //  6. draw storage upgrades
     for (size_t i = 0; i < this->storage_upgrade_sprite_vec.size(); i++) {
         this->render_window_ptr->draw(this->storage_upgrade_sprite_vec[i]);
     }


     //  7. handle dispatch illustration
     if (this->dispatch_MWh > 0) {
         this->dispatch_text.setString(std::to_string(this->dispatch_MWh));
         this->__drawDispatch();
     }


     //  8. draw production menu
     if (this->production_menu_open) {
         this->render_window_ptr->draw(this->production_menu_backing);
         this->render_window_ptr->draw(this->production_menu_backing_text);

         this->__drawProductionMenu();
     }


     //  9. draw upgrade menu
     if (this->upgrade_menu_open) {
         this->render_window_ptr->draw(this->upgrade_menu_backing);
         this->render_window_ptr->draw(this->upgrade_menu_backing_text);

         this->__drawUpgradeOptions();
     }


     //  10. handle broken effects
     if (this->is_broken) {
         for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
             this->tile_improvement_sprite_animated[i].setColor(
                 sf::Color(
                     255,
                     255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                     255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                     255
                 )
             );
         }
     }

     this->frame++;
     return;
 }   /* draw() */
```

4.12.3.15 getTileOptionsSubstring()

```
std::string TidalTurbine::getTileOptionsSubstring (
            void ) [virtual]
```

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

Reimplemented from TileImprovement.

```
{
    //                  32 char x 17 line console "--------------------------------\n";
    std::string options_substring                  = "CAPACITY:      ";
    options_substring                             += std::to_string(this->capacity_kW);
    options_substring                             += " kW (level ";
    options_substring                             += std::to_string(this->upgrade_level);
    options_substring                             += ")\n";

    options_substring                             += "PRODUCTION:    ";
    options_substring                             += std::to_string(this->production_MWh);
    options_substring                             += " MWh\n";

    options_substring                             += "DISPATCHABLE:  ";
    options_substring                             += std::to_string(this->dispatchable_MWh);
    options_substring                             += " MWh\n";

    options_substring                             += "HEALTH:      ";
    options_substring                             += std::to_string(this->health);
    options_substring                             += "/100";

    if (this->health <= 0) {
        options_substring                         += " ** BROKEN! **\n";
    }

    else {
        options_substring                         += "\n";
    }

    options_substring                             += "                                \n";
    options_substring                             += "**** TIDAL TURBINE OPTIONS **** \n";
    options_substring                             += "                                \n";

    options_substring                             += "      [E]:  ";

    if (this->is_broken) {
        options_substring                         += "*** BROKEN! ***\n";
    }

    else {
        options_substring                         += "OPEN PRODUCTION MENU\n";
    }

    options_substring                             += "      [U]:  OPEN UPGRADE MENU    \n";
    options_substring                             += "HOLD [P]:  SCRAP (";
    options_substring                             += std::to_string(SCRAP_COST);
    options_substring                             += " K)";

    return options_substring;
}   /* getTileOptionsSubstring() */
```

4.12.3.16 processEvent()

```
void TidalTurbine::processEvent (
            void  )  [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processEvent();

    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.12.3.17 processMessage()

```
void TidalTurbine::processMessage (
            void  )  [virtual]
```

Method to process TidalTurbine. To be called once per message.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processMessage();

    //...

    return;
}   /* processMessage() */
```

4.12.3.18 setIsSelected()

```
void TidalTurbine::setIsSelected (
            bool is_selected )  [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: setIsSelected(is_selected);

    if (this->is_running and this->is_selected) {
        this->assets_manager_ptr->getSound("water flow")->play();
    }

    return;
} /* setIsSelected() */
```

4.12.3.19 update()

```
void TidalTurbine::update (
            void ) [virtual]
```

Method to trigger production and dispatchable updates.

Reimplemented from TileImprovement.

```
{
    this->__computeProduction();
    this->__computeProductionCosts();
    this->__computeDispatch();

    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
} /* update() */
```

4.12.4 Member Data Documentation

4.12.4.1 bobbing_y

```
double TidalTurbine::bobbing_y
```

The bobbing extent of the tidal turbine.

4.12.4.2 capacity_factor_vec

```
std::vector<double> TidalTurbine::capacity_factor_vec
```

A vector of daily capacity factors for the current month.

4.12.4.3 capacity_kW

```
int TidalTurbine::capacity_kW
```

The rated production capacity [kW] of the solar PV array.

4.12.4.4 dispatch_MWh

```
int TidalTurbine::dispatch_MWh
```

The current dispatch [MWh] of the solar PV array.

4.12.4.5 dispatch_vec_MWh

```
std::vector<double> TidalTurbine::dispatch_vec_MWh
```

A vector of daily dispatch [MWh] for the current month.

4.12.4.6 dispatchable_MWh

```
int TidalTurbine::dispatchable_MWh
```

The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).

4.12.4.7 max_daily_production_MWh

```
double TidalTurbine::max_daily_production_MWh
```

The maximum daily production [MWh] of the solar PV array.

4.12.4.8 production_MWh

```
int TidalTurbine::production_MWh
```

The current production [MWh] of the solar PV array.

4.12.4.9 production_vec_MWh

```
std::vector<double> TidalTurbine::production_vec_MWh
```

A vector of daily production [MWh] for the current month.

4.12.4.10 rotor_drotation

```
double TidalTurbine::rotor_drotation
```

The rotation rate of the rotor.

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

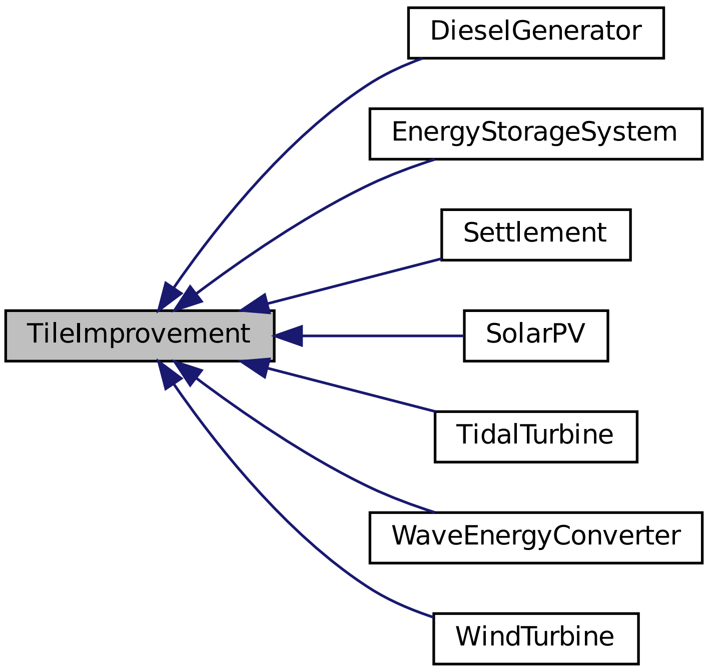
- header/TidalTurbine.h
- source/TidalTurbine.cpp

4.13 TileImprovement Class Reference

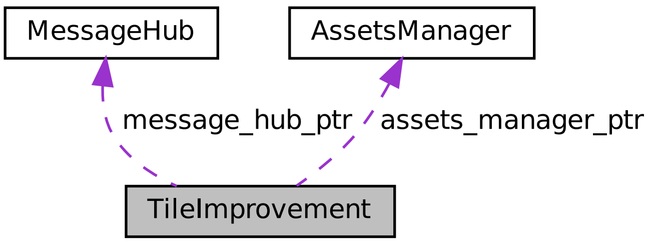
A base class for the tile improvement hierarchy.

```
#include <TileImprovement.h>
```

Inheritance diagram for TileImprovement:



Collaboration diagram for TileImprovement:



Public Member Functions

- TileImprovement (double, double, int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

    Constructor for the TileImprovement class.
- virtual void setIsSelected (bool)

    Method to set the is selected attribute.
- virtual void advanceTurn (void)
- virtual void update (void)
- virtual std::string getTileOptionsSubstring (void)
- virtual void processEvent (void)

    Method to process TileImprovement. To be called once per event.
- virtual void processMessage (void)

    Method to process TileImprovement. To be called once per message.
- virtual void draw (void)

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

    Destructor for the TileImprovement class.

Public Attributes

- TileImprovementType tile_improvement_type

    The type of the tile improvement.
- bool is_running

    A boolean which indicates whether or not the improvement is running.
- bool is_selected

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

    A boolean which indicates that the improvement was just built.
- bool just_upgraded

    A boolean which indicates that the improvement was just upgraded.
- bool production_menu_open

    A boolean which indicates whether or not the production menu is open.
- bool upgrade_menu_open

    A boolean which indicates whether or not the build menu is open.

- bool is_broken

    A boolean which indicated whether or not improvement is broken.
- unsigned long long int frame

    The current frame of this object.
- int credits

    The current balance of credits.
- int month

    The current month of play.
- int demand_MWh

    The current demand [MWh].
- int health

    The health of the improvement.
- int upgrade_level

    The upgrade level of the improvement.
- int upgrade_frame

    The frame of the upgrade animation.
- int storage_kWh

    The rated energy capacity [kWh] of the storage.
- int storage_level

    The level of storage installed alongside the tile improvement.
- int operation_maintenance_cost

    The operation and maintenance costs for this turn.
- int tile_resource

    The renewable resource quality of the tile.
- double tile_resource_scalar

    A scalar associated with the renewable resource quality.
- double position_x

    The x position of the tile improvement.
- double position_y

    The y position of the tile improvement.
- std::vector< double > demand_vec_MWh

    A vector of daily demands [MWh] for the current month.
- std::string game_phase

    The current phase of the game.
- std::string tile_improvement_string

    A string representation of the tile improvement type.
- sf::Sprite tile_improvement_sprite_static

    A static sprite, for decorating the tile.
- std::vector< sf::Sprite > tile_improvement_sprite_animated

    An animated sprite, for the ContextMenu visual screen.
- sf::RectangleShape production_menu_backing

    A backing for the production menu.
- sf::Text production_menu_backing_text

    Text for the production menu backing.
- sf::RectangleShape upgrade_menu_backing

    A backing for the upgrade menu.
- sf::Text upgrade_menu_backing_text

    Text for the upgrade menu backing.
- sf::Sprite storage_upgrade_sprite

    A sprite for illustrating storage (in upgrade menu).
- std::vector< sf::Sprite > storage_upgrade_sprite_vec

A vector of sprites for illustrating the storage upgrade level (on tile).
- sf::Sprite upgrade_arrow_sprite

  An upgrade arrow sprite.
- sf::Sprite upgrade_plus_sprite

  An upgrade plus sprite.
- sf::CircleShape dispatch_backing

  A backing circle for dispatch text illustration.
- sf::Text dispatch_text

  Text for illustrating dispatch.

Protected Member Functions

- void __setUpProductionMenu (void)

  Helper method to set up and position production menu assets (drawable).
- void __setUpUpgradeMenu (void)

  Helper method to set up and position upgrade menu assets (drawable).
- void __setUpDispatchIllustration (void)

  Helper method to set up and position dispatch assets (drawable).
- void __upgradeStorageCapacity (void)

  Helper method to upgrade storage capacity.
- void __handleKeyPressEvents (void)

  Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

  Helper method to handle mouse button events.
- void __openProductionMenu (void)

  Helper method to open the production menu.
- void __closeProductionMenu (void)

  Helper method to close the production menu.
- void __breakdown (void)

  Helper method to trigger an equipment breakdown.
- void __repair (void)

  Helper method to repair a tile improvement.
- void __openUpgradeMenu (void)

  Helper method to open the upgrade menu.
- void __closeUpgradeMenu (void)

  Helper method to close the build menu.
- void __sendTileStateRequest (void)

  Helper method to format and send a request for the parent HexTile to send a tile state message.
- void __sendGameStateRequest (void)

  Helper method to format and send a game state request (message).
- void __sendCreditsSpentMessage (int)

  Helper method to format and send a credits spent message.
- void __sendInsufficientCreditsMessage (void)

  Helper method to format and send an insufficient credits message.
- void __drawDispatch (void)

  Helper method to draw dispatch illustration.

Protected Attributes

- sf::Event ∗ event_ptr

  A pointer to the event class.
- sf::RenderWindow ∗ render_window_ptr

  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr

  A pointer to the message hub.

4.13.1 Detailed Description

A base class for the tile improvement hierarchy.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 TileImprovement()

```
TileImprovement::TileImprovement (
            double position_x,
            double position_y,
            int tile_resource,
            sf::Event * event_ptr,
            sf::RenderWindow * render_window_ptr,
            AssetsManager * assets_manager_ptr,
            MessageHub * message_hub_ptr )
```

Constructor for the TileImprovement class.

Ref: Wikipedia [2023]

Parameters

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

```
{
    //  1. set attributes

    //  1.1. protected
    this->event_ptr = event_ptr;
    this->render_window_ptr = render_window_ptr;

    this->assets_manager_ptr = assets_manager_ptr;
    this->message_hub_ptr = message_hub_ptr;

    //  1.2. public
    this->is_selected = true;
    this->just_built = true;
    this->production_menu_open = false;
    this->upgrade_menu_open = false;
    this->is_broken = false;

    this->just_upgraded = false;
    this->upgrade_frame = 0;

    this->frame = 0;
    this->credits = 0;
    this->month = 1;
    this->demand_MWh = 0;

    this->demand_vec_MWh.resize(30, 0);

    this->operation_maintenance_cost = 0;

    this->tile_resource = tile_resource;

    switch (this->tile_resource) {
        case (0): {
            this->tile_resource_scalar = 0.7;

            break;
        }


        case (1): {
            this->tile_resource_scalar = 0.85;

            break;
        }


        case (2): {
            this->tile_resource_scalar = 1;

            break;
        }


        case (3): {
            this->tile_resource_scalar = 1.15;

            break;
        }


        case (4): {
            this->tile_resource_scalar = 1.3;

            break;
        }


        default: {
            this->tile_resource_scalar = 1;
        }
    }

    this->position_x = position_x;
    this->position_y = position_y;

    this->game_phase = "build settlement";

    this->__setUpProductionMenu();
    this->__setUpUpgradeMenu();
    this->__setUpDispatchIllustration();

    std::cout « "TileImprovement constructed at " « this « std::endl;

    return;
}   /* TileImprovement() */
```

4.13.2.2 ∼TileImprovement()

```
TileImprovement::∼TileImprovement (
            void  )  [virtual]
```

Destructor for the TileImprovement class.

```
{
    std::cout « "TileImprovement at " « this « " destroyed" « std::endl;

    return;
}   /* ~TileImprovement() */
```

4.13.3 Member Function Documentation

4.13.3.1 __breakdown()

```
void TileImprovement::__breakdown (
            void  )  [protected]
```

Helper method to trigger an equipment breakdown.

```
{
    this->is_broken = true;
    this->is_running = false;
    this->assets_manager_ptr->getSound("breakdown")->play();

    return;
}   /* __breakdown() */
```

4.13.3.2 __closeProductionMenu()

```
void TileImprovement::__closeProductionMenu (
            void  )  [protected]
```

Helper method to close the production menu.

```
{
    if (not this->production_menu_open) {
        return;
    }

    this->production_menu_open = false;
    this->assets_manager_ptr->getSound("build menu close")->play();

    return;
}   /* __closeProductionMenu() */
```

4.13.3.3 __closeUpgradeMenu()

```
void TileImprovement::__closeUpgradeMenu (
            void  )  [protected]
```

Helper method to close the build menu.

```
{
    if (not this->upgrade_menu_open) {
        return;
    }

    this->upgrade_menu_open = false;
    this->assets_manager_ptr->getSound("build menu close")->play();

    return;
}   /* __closeUpgradeMenu() */
```

4.13.3.4 __drawDispatch()

```
void TileImprovement::__drawDispatch (
            void ) [protected]
```

Helper method to draw dispatch illustration.

```
{
    double alpha = 255 * pow(cos((0.5 * M_PI * this->frame) / FRAMES_PER_SECOND), 2);


    //  1. dispatch backing
    sf::Color backing_colour = this->dispatch_backing.getFillColor();

    backing_colour.a = alpha;

    this->dispatch_backing.setFillColor(backing_colour);
    this->dispatch_backing.setOutlineColor(sf::Color(0, 0, 0, alpha));

    this->render_window_ptr->draw(this->dispatch_backing);


    //  2. dispatch text
    this->dispatch_text.setOrigin(
        this->dispatch_text.getLocalBounds().width / 2,
        this->dispatch_text.getLocalBounds().height / 2
    );

    this->dispatch_text.setFillColor(
        sf::Color(0, 0, 0, alpha)
    );

    this->render_window_ptr->draw(this->dispatch_text);

    return;
}   /* __drawDispatch() */
```

4.13.3.5 __handleKeyPressEvents()

```
void TileImprovement::__handleKeyPressEvents (
            void ) [protected]
```

Helper method to handle key press events.

```
{
    if (this->tile_improvement_type == TileImprovementType :: SETTLEMENT) {
        return;
    }

    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::E): {
            this->__openProductionMenu();

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.13.3.6 __handleMouseButtonEvents()

```
void TileImprovement::__handleMouseButtonEvents (
              void ) [protected]
```

Helper method to handle mouse button events.

```
{
    if (this->tile_improvement_type == TileImprovementType :: SETTLEMENT) {
        return;
    }

    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.13.3.7 __openProductionMenu()

```
void TileImprovement::__openProductionMenu (
              void ) [protected]
```

Helper method to open the production menu.

```
{
    if (this->is_broken) {
        this->assets_manager_ptr->getSound("breakdown")->play();
        return;
    }

    if (this->production_menu_open) {
        return;
    }

    if (this->upgrade_menu_open) {
        this->__closeUpgradeMenu();
    }

    this->production_menu_open = true;
    this->assets_manager_ptr->getSound("build menu open")->play();

    return;
}   /* __openProductionMenu() */
```

4.13.3.8 __openUpgradeMenu()

```
void TileImprovement::__openUpgradeMenu (
            void ) [protected]
```

Helper method to open the upgrade menu.

```
{
    if (this->upgrade_menu_open) {
        return;
    }

    if (this->production_menu_open) {
        this->__closeProductionMenu();
    }

    this->upgrade_menu_open = true;
    this->assets_manager_ptr->getSound("build menu open")->play();

    return;
}   /* __openUpgradeMenu() */
```

4.13.3.9 __repair()

```
void TileImprovement::__repair (
            void ) [protected]
```

Helper method to repair a tile improvement.

```
{
    this->health = 100;
    if (this->is_broken) {
        this->is_broken = false;
        this->assets_manager_ptr->getSound("positive notification")->play();
    }


    if (this->tile_improvement_sprite_static.getTexture() != NULL) {
        this->tile_improvement_sprite_static.setColor(sf::Color(255, 255, 255, 255));
    }

    else {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(255, 255, 255, 255)
            );
        }
    }

    return;
}   /* __repair() */
```

4.13.3.10 __sendCreditsSpentMessage()

```
void TileImprovement::__sendCreditsSpentMessage (
            int credits_spent ) [protected]
```

Helper method to format and send a credits spent message.

Parameters

credits_spent	The number of credits that were spent.

```
{
    Message credits_spent_message;

    credits_spent_message.channel = GAME_CHANNEL;
    credits_spent_message.subject = "credits spent";

    credits_spent_message.int_payload["credits spent"] = credits_spent;

    this->message_hub_ptr->sendMessage(credits_spent_message);

    std::cout « "Credits spent (" « credits_spent « ") message sent by " « this
        « std::endl;
    return;
}   /* __sendCreditsSpentMessage() */
```

4.13.3.11 __sendGameStateRequest()

```
void TileImprovement::__sendGameStateRequest (
              void ) [protected]
```

Helper method to format and send a game state request (message).

```
{
    Message game_state_request;

    game_state_request.channel = GAME_CHANNEL;
    game_state_request.subject = "state request";

    this->message_hub_ptr->sendMessage(game_state_request);

    std::cout « "Game state request message sent by " « this « std::endl;
    return;
}   /* __sendGameStateRequest() */
```

4.13.3.12 __sendInsufficientCreditsMessage()

```
void TileImprovement::__sendInsufficientCreditsMessage (
              void ) [protected]
```

Helper method to format and send an insufficient credits message.

```
{
    Message insufficient_credits_message;

    insufficient_credits_message.channel = GAME_CHANNEL;
    insufficient_credits_message.subject = "insufficient credits";

    this->message_hub_ptr->sendMessage(insufficient_credits_message);

    std::cout « "Insufficient credits message sent by " « this « std::endl;

    return;
}   /* __sendInsufficientCreditsMessage() */
```

4.13.3.13 __sendTileStateRequest()

```
void TileImprovement::__sendTileStateRequest (
              void ) [protected]
```

Helper method to format and send a request for the parent HexTile to send a tile state message.

```
{
    Message tile_state_request;

    tile_state_request.channel = TILE_STATE_CHANNEL;
    tile_state_request.subject = "state request";

    this->message_hub_ptr->sendMessage(tile_state_request);

    std::cout « "Tile state request sent by " « this « std::endl;
    return;
}   /* __sendTileStateRequest() */
```

4.13.3.14 __setUpDispatchIllustration()

```
void TileImprovement::__setUpDispatchIllustration (
            void ) [protected]
```

Helper method to set up and position dispatch assets (drawable).

```
{
    // 1. set up backing
    this->dispatch_backing.setRadius(16);

    this->dispatch_backing.setOrigin(
        this->dispatch_backing.getLocalBounds().width / 2,
        this->dispatch_backing.getLocalBounds().height / 2
    );

    this->dispatch_backing.setPosition(
        this->position_x,
        this->position_y
    );

    this->dispatch_backing.setFillColor(RESOURCE_CHIP_GREY);
    this->dispatch_backing.setOutlineThickness(1);
    this->dispatch_backing.setOutlineColor(sf::Color(0, 0, 0, 255));


    // 2. set up text
    this->dispatch_text.setFont(*(assets_manager_ptr->getFont("DroidSansMono")));
    this->dispatch_text.setFillColor(sf::Color(0, 0, 0, 255));
    this->dispatch_text.setCharacterSize(16);
    this->dispatch_text.setPosition(
        this->position_x,
        this->position_y - 4
    );

    return;
}   /* __setUpDispatchIllustration() */
```

4.13.3.15 __setUpProductionMenu()

```
void TileImprovement::__setUpProductionMenu (
            void ) [protected]
```

Helper method to set up and position production menu assets (drawable).

```
{
    // 1. set up and place production menu backing and text
    this->production_menu_backing.setSize(sf::Vector2f(400, 256));
    this->production_menu_backing.setOrigin(200, 128);
    this->production_menu_backing.setPosition(400, 400);
    this->production_menu_backing.setFillColor(MONOCHROME_SCREEN_BACKGROUND);
    this->production_menu_backing.setOutlineColor(MENU_FRAME_GREY);
    this->production_menu_backing.setOutlineThickness(4);

    this->production_menu_backing_text.setString("**** PRODUCTION MENU ****");
    this->production_menu_backing_text.setFont(
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220"))
    );
    this->production_menu_backing_text.setCharacterSize(16);
    this->production_menu_backing_text.setFillColor(MONOCHROME_TEXT_GREEN);
    this->production_menu_backing_text.setOrigin(
        this->production_menu_backing_text.getLocalBounds().width / 2, 0
    );
    this->production_menu_backing_text.setPosition(400, 400 - 128 + 4);

    return;
}   /* __setUpProductionMenu() */
```

4.13.3.16 __setUpUpgradeMenu()

```
void TileImprovement::__setUpUpgradeMenu (
            void ) [protected]
```

Helper method to set up and position upgrade menu assets (drawable).

```
{
    //  1. set up and place upgrade menu backing and text
    this->upgrade_menu_backing.setSize(sf::Vector2f(400, 256));
    this->upgrade_menu_backing.setOrigin(200, 128);
    this->upgrade_menu_backing.setPosition(400, 400);
    this->upgrade_menu_backing.setFillColor(MONOCHROME_SCREEN_BACKGROUND);
    this->upgrade_menu_backing.setOutlineColor(MENU_FRAME_GREY);
    this->upgrade_menu_backing.setOutlineThickness(4);

    this->upgrade_menu_backing_text.setString("**** UPGRADE MENU ****");
    this->upgrade_menu_backing_text.setFont(
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220"))
    );
    this->upgrade_menu_backing_text.setCharacterSize(16);
    this->upgrade_menu_backing_text.setFillColor(MONOCHROME_TEXT_GREEN);
    this->upgrade_menu_backing_text.setOrigin(
        this->upgrade_menu_backing_text.getLocalBounds().width / 2, 0
    );
    this->upgrade_menu_backing_text.setPosition(400, 400 - 128 + 4);


    //  2. set up and place storage upgrade sprite (with upgrade plus)
    this->storage_upgrade_sprite = sf::Sprite(
        *(this->assets_manager_ptr->getTexture("energy storage system"))
    );

    this->storage_upgrade_sprite.setOrigin(
        this->storage_upgrade_sprite.getLocalBounds().width / 2,
        this->storage_upgrade_sprite.getLocalBounds().height
    );

    this->storage_upgrade_sprite.setPosition(400 + 100, 400 - 32);

    this->upgrade_plus_sprite = sf::Sprite(
        *(this->assets_manager_ptr->getTexture("upgrade plus"))
    );

    this->upgrade_plus_sprite.setOrigin(
        this->upgrade_plus_sprite.getLocalBounds().width / 2,
        this->upgrade_plus_sprite.getLocalBounds().height / 2
    );

    this->upgrade_plus_sprite.setPosition(400 + 130, 400 - 64);


    //  3. set up and place upgrade arrow sprite
    this->upgrade_arrow_sprite = sf::Sprite(
        *(this->assets_manager_ptr->getTexture("upgrade arrow"))
    );

    this->upgrade_arrow_sprite.setOrigin(
        this->upgrade_arrow_sprite.getLocalBounds().width / 2,
        this->upgrade_arrow_sprite.getLocalBounds().height / 2
    );

    this->upgrade_arrow_sprite.setPosition(400 - 64, 400 - 64);


    return;
}   /* __setUpUpgradeMenu() */
```

4.13.3.17 __upgradeStorageCapacity()

```
void TileImprovement::__upgradeStorageCapacity (
            void ) [protected]
```

Helper method to upgrade storage capacity.

```
{
    if (this->credits < ENERGY_STORAGE_SYSTEM_BUILD_COST) {
        std::cout « "Cannot add energy storage: insufficient credits (need "
            « ENERGY_STORAGE_SYSTEM_BUILD_COST « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    if (this->storage_level >= MAX_STORAGE_LEVELS) {
        return;
    }

    this->storage_level++;
    this->storage_kWh += 200;

    this->storage_upgrade_sprite_vec.push_back(
        sf::Sprite(
            *(this->assets_manager_ptr->getTexture("storage level"))
        )
    );

    this->storage_upgrade_sprite_vec.back().setOrigin(
        this->storage_upgrade_sprite_vec.back().getLocalBounds().width / 2,
        this->storage_upgrade_sprite_vec.back().getLocalBounds().height
    );

    this->storage_upgrade_sprite_vec.back().setPosition(
        this->position_x + 18,
        this->position_y + 25 - 7 * this->storage_upgrade_sprite_vec.size()
    );

    this->just_upgraded = true;

    this->assets_manager_ptr->getSound("upgrade")->play();

    this->__sendCreditsSpentMessage(ENERGY_STORAGE_SYSTEM_BUILD_COST);
    this->__sendTileStateRequest();

    return;
}   /* __upgradeStorageCapacity() */
```

4.13.3.18 advanceTurn()

```
virtual void TileImprovement::advanceTurn (
            void )  [inline], [virtual]
```

Reimplemented in WindTurbine, WaveEnergyConverter, TidalTurbine, SolarPV, and DieselGenerator.

```
{return;}
```

4.13.3.19 draw()

```
void TileImprovement::draw (
            void )  [virtual]
```

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

Reimplemented in WindTurbine, WaveEnergyConverter, TidalTurbine, SolarPV, Settlement, EnergyStorageSystem, and DieselGenerator.

```
{
    if (this->tile_improvement_sprite_static.getTexture() != NULL) {
        int alpha = this->tile_improvement_sprite_static.getColor().a;

        alpha += 0.08 * FRAMES_PER_SECOND;

        this->tile_improvement_sprite_static.setColor(
            sf::Color(255, 255, 255, alpha)
        );

        this->tile_improvement_sprite_static.move(0, 50 * SECONDS_PER_FRAME);

        if (
            (alpha >= 255) or
            (this->tile_improvement_sprite_static.getPosition().y >= this->position_y + 12)
        ) {
            this->tile_improvement_sprite_static.setColor(
                sf::Color(255, 255, 255, 255)
            );

            this->tile_improvement_sprite_static.setPosition(
                this->position_x,
                this->position_y + 12
            );

            this->just_built = false;
            this->assets_manager_ptr->getSound("place improvement")->play();
        }

        this->render_window_ptr->draw(this->tile_improvement_sprite_static);
    }


    else {
        int alpha = 0;

        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            alpha = this->tile_improvement_sprite_animated[i].getColor().a;

            alpha += 0.08 * FRAMES_PER_SECOND;

            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(255, 255, 255, alpha)
            );

            this->tile_improvement_sprite_animated[i].move(0, 50 * SECONDS_PER_FRAME);

            if (
                (alpha >= 255) or
                (this->tile_improvement_sprite_animated[i].getPosition().y >= this->position_y + 12)
            ) {
                this->tile_improvement_sprite_animated[i].setColor(
                    sf::Color(255, 255, 255, 255)
                );

                this->tile_improvement_sprite_animated[i].setPosition(
                    this->position_x,
                    this->position_y + 12
                );
            }

            this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);
        }

        if (
            (alpha >= 255) or
            (this->tile_improvement_sprite_animated[0].getPosition().y >= this->position_y + 12)
        ) {
            this->just_built = false;
            this->assets_manager_ptr->getSound("place improvement")->play();

            switch (this->tile_improvement_type) {
                case (TileImprovementType :: WIND_TURBINE): {
                    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
                        this->tile_improvement_sprite_animated[i].setOrigin(32, 32);
                        this->tile_improvement_sprite_animated[i].move(0, -32);
                    }

                    break;
                }


                case (TileImprovementType :: TIDAL_TURBINE): {
                    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
                        this->tile_improvement_sprite_animated[i].setOrigin(32, 45);
                        this->tile_improvement_sprite_animated[i].move(0, -19);
                    }

                    break;
                }


                case (TileImprovementType :: WAVE_ENERGY_CONVERTER): {
                    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
                        this->tile_improvement_sprite_animated[i].setOrigin(32, 32);
                        this->tile_improvement_sprite_animated[i].move(0, -32);
                    }

                    break;
                }


                default: {
                    // do nothing!

                    break;
                }
            }
        }
    }


    this->frame++;
    return;
}   /* draw() */
```

4.13.3.20 getTileOptionsSubstring()

```
virtual std::string TileImprovement::getTileOptionsSubstring (
            void ) [inline], [virtual]
```

Reimplemented in WindTurbine, WaveEnergyConverter, TidalTurbine, SolarPV, Settlement, EnergyStorageSystem, and DieselGenerator.

```
{return "";}
```

4.13.3.21 processEvent()

```
void TileImprovement::processEvent (
            void ) [virtual]
```

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

Reimplemented in WindTurbine, WaveEnergyConverter, TidalTurbine, SolarPV, Settlement, EnergyStorageSystem, and DieselGenerator.

```
{
    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.13.3.22 processMessage()

```
void TileImprovement::processMessage (
            void ) [virtual]
```

Method to process TileImprovement. To be called once per message.

Reimplemented in WindTurbine, WaveEnergyConverter, TidalTurbine, SolarPV, Settlement, EnergyStorageSystem, and DieselGenerator.

```
{
    if (not this->message_hub_ptr->isEmpty(GAME_STATE_CHANNEL)) {
        Message game_state_message = this->message_hub_ptr->receiveMessage(
            GAME_STATE_CHANNEL
        );

        if (game_state_message.subject == "turn advance") {
            this->credits = game_state_message.int_payload["credits"];
            this->month = game_state_message.int_payload["month"];
            this->demand_MWh = game_state_message.int_payload["demand_MWh"];

            this->advanceTurn();

            std::cout « "Turn advance message read and passed by " « this « std::endl;
        }
    }

    return;
}   /* processMessage() */
```

4.13.3.23 setIsSelected()

```
void TileImprovement::setIsSelected (
            bool is_selected ) [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented in WindTurbine, WaveEnergyConverter, TidalTurbine, SolarPV, Settlement, EnergyStorageSystem, and DieselGenerator.

```
{
    this->is_selected = is_selected;

    if ((not is_selected) and this->production_menu_open) {
        this->__closeProductionMenu();
    }

    if ((not is_selected) and this->upgrade_menu_open) {
        this->__closeUpgradeMenu();
    }

    return;
}   /* setIsSelected() */
```

4.13.3.24 update()

```
virtual void TileImprovement::update (
            void ) [inline], [virtual]
```

Reimplemented in WindTurbine, WaveEnergyConverter, TidalTurbine, and SolarPV.

```
{return;}
```

4.13.4 Member Data Documentation

4.13.4.1 assets_manager_ptr

```
AssetsManager* TileImprovement::assets_manager_ptr  [protected]
```

A pointer to the assets manager.

4.13.4.2 credits

```
int TileImprovement::credits
```

The current balance of credits.

4.13.4.3 demand_MWh

```
int TileImprovement::demand_MWh
```

The current demand [MWh].

4.13.4.4 demand_vec_MWh

```
std::vector<double> TileImprovement::demand_vec_MWh
```

A vector of daily demands [MWh] for the current month.

4.13.4.5 dispatch_backing

```
sf::CircleShape TileImprovement::dispatch_backing
```

A backing circle for dispatch text illustration.

4.13.4.6 dispatch_text

```
sf::Text TileImprovement::dispatch_text
```

Text for illustrating dispatch.

4.13.4.7 event_ptr

```
sf::Event* TileImprovement::event_ptr  [protected]
```

A pointer to the event class.

4.13.4.8 frame

```
unsigned long long int TileImprovement::frame
```

The current frame of this object.

4.13.4.9 game_phase

```
std::string TileImprovement::game_phase
```

The current phase of the game.

4.13.4.10 health

```
int TileImprovement::health
```

The health of the improvement.

4.13.4.11 is_broken

```
bool TileImprovement::is_broken
```

A boolean which indicated whether or not improvement is broken.

4.13.4.12 is_running

`bool TileImprovement::is_running`

A boolean which indicates whether or not the improvement is running.

4.13.4.13 is_selected

`bool TileImprovement::is_selected`

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

4.13.4.14 just_built

`bool TileImprovement::just_built`

A boolean which indicates that the improvement was just built.

4.13.4.15 just_upgraded

`bool TileImprovement::just_upgraded`

A boolean which indicates that the improvement was just upgraded.

4.13.4.16 message_hub_ptr

`MessageHub* TileImprovement::message_hub_ptr [protected]`

A pointer to the message hub.

4.13.4.17 month

`int TileImprovement::month`

The current month of play.

4.13.4.18 operation_maintenance_cost

```
int TileImprovement::operation_maintenance_cost
```

The operation and maintenance costs for this turn.

4.13.4.19 position_x

```
double TileImprovement::position_x
```

The x position of the tile improvement.

4.13.4.20 position_y

```
double TileImprovement::position_y
```

The y position of the tile improvement.

4.13.4.21 production_menu_backing

```
sf::RectangleShape TileImprovement::production_menu_backing
```

A backing for the production menu.

4.13.4.22 production_menu_backing_text

```
sf::Text TileImprovement::production_menu_backing_text
```

Text for the production menu backing.

4.13.4.23 production_menu_open

```
bool TileImprovement::production_menu_open
```

A boolean which indicates whether or not the production menu is open.

4.13.4.24 render_window_ptr

```
sf::RenderWindow* TileImprovement::render_window_ptr  [protected]
```

A pointer to the render window.

4.13.4.25 storage_kWh

```
int TileImprovement::storage_kWh
```

The rated energy capacity [kWh] of the storage.

4.13.4.26 storage_level

```
int TileImprovement::storage_level
```

The level of storage installed alongside the tile improvement.

4.13.4.27 storage_upgrade_sprite

```
sf::Sprite TileImprovement::storage_upgrade_sprite
```

A sprite for illustrating storage (in upgrade menu).

4.13.4.28 storage_upgrade_sprite_vec

```
std::vector<sf::Sprite> TileImprovement::storage_upgrade_sprite_vec
```

A vector of sprites for illustrating the storage upgrade level (on tile).

4.13.4.29 tile_improvement_sprite_animated

```
std::vector<sf::Sprite> TileImprovement::tile_improvement_sprite_animated
```

An animated sprite, for the ContextMenu visual screen.

4.13.4.30 tile_improvement_sprite_static

```
sf::Sprite TileImprovement::tile_improvement_sprite_static
```

A static sprite, for decorating the tile.

4.13.4.31 tile_improvement_string

```
std::string TileImprovement::tile_improvement_string
```

A string representation of the tile improvement type.

4.13.4.32 tile_improvement_type

```
TileImprovementType TileImprovement::tile_improvement_type
```

The type of the tile improvement.

4.13.4.33 tile_resource

```
int TileImprovement::tile_resource
```

The renewable resource quality of the tile.

4.13.4.34 tile_resource_scalar

```
double TileImprovement::tile_resource_scalar
```

A scalar associated with the renewable resource quality.

4.13.4.35 upgrade_arrow_sprite

```
sf::Sprite TileImprovement::upgrade_arrow_sprite
```

An upgrade arrow sprite.

4.13.4.36 upgrade_frame

```
int TileImprovement::upgrade_frame
```

The frame of the upgrade animation.

4.13.4.37 upgrade_level

```
int TileImprovement::upgrade_level
```

The upgrade level of the improvement.

4.13.4.38 upgrade_menu_backing

```
sf::RectangleShape TileImprovement::upgrade_menu_backing
```

A backing for the upgrade menu.

4.13.4.39 upgrade_menu_backing_text

```
sf::Text TileImprovement::upgrade_menu_backing_text
```

Text for the upgrade menu backing.

4.13.4.40 upgrade_menu_open

```
bool TileImprovement::upgrade_menu_open
```

A boolean which indicates whether or not the build menu is open.

4.13.4.41 upgrade_plus_sprite

```
sf::Sprite TileImprovement::upgrade_plus_sprite
```

An upgrade plus sprite.

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

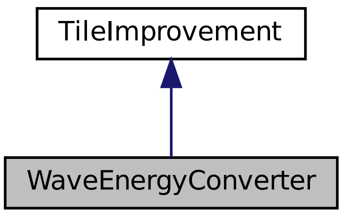
- header/TileImprovement.h
- source/TileImprovement.cpp

4.14 WaveEnergyConverter Class Reference

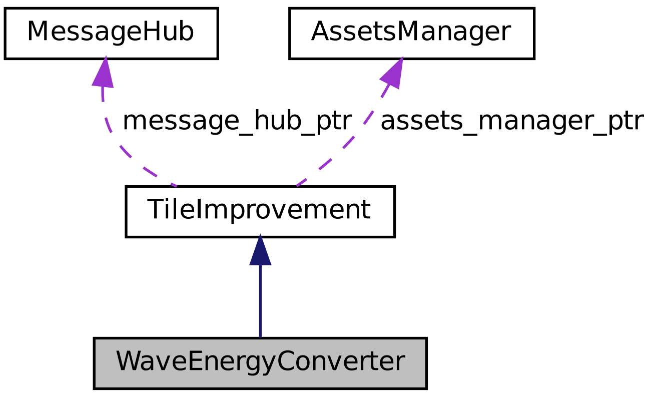
A settlement class (child class of TileImprovement).

`#include <WaveEnergyConverter.h>`

Inheritance diagram for WaveEnergyConverter:



Collaboration diagram for WaveEnergyConverter:



Public Member Functions

- WaveEnergyConverter (double, double, int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

    Constructor for the WaveEnergyConverter class.
- std::string getTileOptionsSubstring (void)

    Helper method to assemble and return tile options substring.
- void setIsSelected (bool)

    Method to set the is selected attribute.

- void advanceTurn (void)

    Method to handle turn advance.
- void update (void)

    Method to trigger production and dispatchable updates.
- void processEvent (void)

    Method to process WaveEnergyConverter. To be called once per event.
- void processMessage (void)

    Method to process WaveEnergyConverter. To be called once per message.
- void draw (void)

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

    Destructor for the WaveEnergyConverter class.

Public Attributes

- int capacity_kW

    The rated production capacity [kW] of the solar PV array.
- int production_MWh

    The current production [MWh] of the solar PV array.
- int dispatch_MWh

    The current dispatch [MWh] of the solar PV array.
- int dispatchable_MWh

    The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).
- double max_daily_production_MWh

    The maximum daily production [MWh] of the solar PV array.
- double bobbing_y

    The bobbing extent of the wave energy converter.
- std::vector< double > capacity_factor_vec

    A vector of daily capacity factors for the current month.
- std::vector< double > production_vec_MWh

    A vector of daily production [MWh] for the current month.
- std::vector< double > dispatch_vec_MWh

    A vector of daily dispatch [MWh] for the current month.

Private Member Functions

- void __setUpTileImprovementSpriteAnimated (void)

    Helper method to set up tile improvement sprite (static).
- void __drawProductionMenu (void)

    Helper method to draw production menu assets.
- void __upgradePowerCapacity (void)

    Helper method to upgrade power capacity.
- void __computeProductionCosts (void)

    Helper method to compute production costs (O&M) based on current production level.
- void __breakdown (void)

    Helper method to trigger an equipment breakdown.
- void __computeCapacityFactors (void)

    Helper method to compute capacity factors.
- void __computeProduction (void)

Helper method to compute production values.

- void __computeDispatch (void)

    Helper method to compute dispatch values.
- void __handleKeyPressEvents (void)

    Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

    Helper method to handle mouse button events.
- void __drawUpgradeOptions (void)

    Helper method to set up and draw upgrade options.
- void __sendImprovementStateMessage (void)

    Helper method to format and sent improvement state message.

Additional Inherited Members

4.14.1 Detailed Description

A settlement class (child class of TileImprovement).

4.14.2 Constructor & Destructor Documentation

4.14.2.1 WaveEnergyConverter()

```
WaveEnergyConverter::WaveEnergyConverter (
            double position_x,
            double position_y,
            int tile_resource,
            sf::Event * event_ptr,
            sf::RenderWindow * render_window_ptr,
            AssetsManager * assets_manager_ptr,
            MessageHub * message_hub_ptr )
```

Constructor for the WaveEnergyConverter class.

Ref: Wikipedia [2023]

Parameters

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

```
  :
TileImprovement(
    position_x,
    position_y,
    tile_resource,
    event_ptr,
    render_window_ptr,
    assets_manager_ptr,
    message_hub_ptr
)
{
    //  1. set attributes

    //  1.1. private
    //...

    //  1.2. public
    this->tile_improvement_type = TileImprovementType :: WAVE_ENERGY_CONVERTER;

    this->is_running = false;

    this->health = 100;

    this->capacity_kW = 100;
    this->upgrade_level = 1;

    this->storage_kWh = 0;
    this->storage_level = 0;

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->bobbing_y = 4;

    this->capacity_factor_vec.resize(30, 0);
    this->production_vec_MWh.resize(30, 0);
    this->dispatch_vec_MWh.resize(30, 0);

    this->tile_improvement_string = "WAVE ENERGY";

    this->__setUpTileImprovementSpriteAnimated();
    this->__computeCapacityFactors();
    this->update();

    std::cout « "WaveEnergyConverter constructed at " « this « std::endl;

    return;
}   /* WaveEnergyConverter() */
```

4.14.2.2 ∼WaveEnergyConverter()

```
WaveEnergyConverter::∼WaveEnergyConverter (
               void  )  [virtual]
```

Destructor for the WaveEnergyConverter class.

```
{
    std::cout « "WaveEnergyConverter at " « this « " destroyed" « std::endl;

    return;
}   /* ~WaveEnergyConverter() */
```

4.14.3 Member Function Documentation

4.14.3.1 __breakdown()

```
void WaveEnergyConverter::__breakdown (
              void ) [private]
```

Helper method to trigger an equipment breakdown.

```
{
    TileImprovement :: __breakdown();

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;
    this->operation_maintenance_cost = 0;

    return;
}   /* __breakdown() */
```

4.14.3.2 __computeCapacityFactors()

```
void WaveEnergyConverter::__computeCapacityFactors (
              void ) [private]
```

Helper method to compute capacity factors.

```
{
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator(seed);

    double mean =
        this->tile_resource_scalar * MEAN_DAILY_WAVE_CAPACITY_FACTORS[this->month - 1];

    double stdev = STDEV_DAILY_WAVE_CAPACITY_FACTORS[this->month - 1];

    if (this->tile_resource_scalar > 1) {
        stdev /= this->tile_resource_scalar;
    }

    std::normal_distribution<double> normal_dist(mean, stdev);

    double capacity_factor = 0;

    for (int i = 0; i < 30; i++) {
        capacity_factor = normal_dist(generator);

        if (capacity_factor < 0) {
            capacity_factor = 0;
        }

        this->capacity_factor_vec[i] = capacity_factor;
    }

    return;
}   /* __computeCapacityFactors() */
```

4.14.3.3 __computeDispatch()

```
void WaveEnergyConverter::__computeDispatch (
              void ) [private]
```

Helper method to compute dispatch values.

```
{
    double stored_energy_MWh = 0;
    double storage_capacity_MWh = (double)(this->storage_kWh) / 1000;

    double demand_MWh = 0;
    double production_MWh = 0;
    double dispatchable_MWh = 0;
    double difference_MWh = 0;

    double room_MWh = 0;

    for (int i = 0; i < 30; i++) {
        demand_MWh = this->demand_vec_MWh[i];
        production_MWh = this->production_vec_MWh[i];

        if (production_MWh <= demand_MWh) {
            this->dispatch_vec_MWh[i] = production_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = demand_MWh - production_MWh;

            if ((storage_capacity_MWh > 0) and (stored_energy_MWh > 0)) {
                if (difference_MWh > stored_energy_MWh) {
                    this->dispatch_vec_MWh[i] += stored_energy_MWh;
                    dispatchable_MWh += stored_energy_MWh;
                    stored_energy_MWh = 0;
                }

                else {
                    this->dispatch_vec_MWh[i] += difference_MWh;
                    dispatchable_MWh += difference_MWh;
                    stored_energy_MWh -= difference_MWh;
                }
            }
        }

        else {
            this->dispatch_vec_MWh[i] = demand_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = production_MWh - demand_MWh;

            if (
                (storage_capacity_MWh > 0) and
                (stored_energy_MWh < storage_capacity_MWh)
            ) {
                room_MWh = storage_capacity_MWh - stored_energy_MWh;

                if (difference_MWh > room_MWh) {
                    stored_energy_MWh += room_MWh;
                }

                else {
                    stored_energy_MWh += difference_MWh;
                }
            }
        }
    }

    this->dispatchable_MWh = round(dispatchable_MWh);

    if (this->dispatch_MWh > this->dispatchable_MWh) {
        this->dispatch_MWh = this->dispatchable_MWh;
    }

    return;
}   /* __computeDispatch() */
```

4.14.3.4 __computeProduction()

```
void WaveEnergyConverter::__computeProduction (
            void ) [private]
```

Helper method to compute production values.

```
{
    double production_MWh = 0;

    for (int i = 0; i < 30; i++) {
        this->production_vec_MWh[i] =
            this->max_daily_production_MWh * this->capacity_factor_vec[i];

        production_MWh += this->production_vec_MWh[i];
    }

    this->production_MWh = round(production_MWh);
    return;
} /* __computeProduction() */
```

4.14.3.5 __computeProductionCosts()

```
void WaveEnergyConverter::__computeProductionCosts (
            void ) [private]
```

Helper method to compute production costs (O&M) based on current production level.

```
{
    double operation_maintenance_cost =
        (this->production_MWh * WAVE_OP_MAINT_COST_PER_MWH_PRODUCTION) / 1000;
    this->operation_maintenance_cost = round(operation_maintenance_cost);

    return;
} /* __computeProductionCosts() */
```

4.14.3.6 __drawProductionMenu()

```
void WaveEnergyConverter::__drawProductionMenu (
            void ) [private]
```

Helper method to draw production menu assets.

```
{
    // 1. draw static sprite
    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
        sf::Vector2f initial_position = this->tile_improvement_sprite_animated[i].getPosition();
        this->tile_improvement_sprite_animated[i].setPosition(400 - 138, 400 + 16);

        sf::Color initial_colour = this->tile_improvement_sprite_animated[i].getColor();
        this->tile_improvement_sprite_animated[i].setColor(sf::Color(255, 255, 255, 255));

        sf::Vector2f initial_scale = this->tile_improvement_sprite_animated[i].getScale();
        this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1, 1));

        double initial_rotation = this->tile_improvement_sprite_animated[i].getRotation();
        this->tile_improvement_sprite_animated[i].setRotation(0);

        this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);

        this->tile_improvement_sprite_animated[i].setPosition(initial_position);
        this->tile_improvement_sprite_animated[i].setColor(initial_colour);
        this->tile_improvement_sprite_animated[i].setScale(initial_scale);
        this->tile_improvement_sprite_animated[i].setRotation(initial_rotation);
    }

    // 2. draw production text
    std::string production_string = "[W]:  INCREASE DISPATCH\n";
    production_string            += "[S]:  DECREASE DISPATCH\n";
    production_string            += "                         \n";

    production_string            += "DISPATCH:  ";
    production_string            += std::to_string(this->dispatch_MWh);
    production_string            += " MWh (MAX ";
    production_string            += std::to_string(this->dispatchable_MWh);
    production_string            += ")\n";

    production_string            += "O&M COST:  ";
    production_string            += std::to_string(this->operation_maintenance_cost);
    production_string            += " K\n";

    sf::Text production_text(
        production_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    production_text.setOrigin(production_text.getLocalBounds().width / 2,0);
    production_text.setFillColor(MONOCHROME_TEXT_GREEN);

    production_text.setPosition(400 + 30, 400 - 45);

    this->render_window_ptr->draw(production_text);

    return;
}   /* __drawProductionMenu() */
```

4.14.3.7 __drawUpgradeOptions()

```
void WaveEnergyConverter::__drawUpgradeOptions (
            void ) [private]
```

Helper method to set up and draw upgrade options.

```
{
    //  1. draw power capacity upgrade sprite
    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
        sf::Vector2f initial_position = this->tile_improvement_sprite_animated[i].getPosition();
        this->tile_improvement_sprite_animated[i].setPosition(400 - 100, 400 - 32 - 20);

        sf::Color initial_colour = this->tile_improvement_sprite_animated[i].getColor();
        this->tile_improvement_sprite_animated[i].setColor(sf::Color(255, 255, 255, 255));

        sf::Vector2f initial_scale = this->tile_improvement_sprite_animated[i].getScale();
        this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1, 1));

        this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);

        this->tile_improvement_sprite_animated[i].setPosition(initial_position);
        this->tile_improvement_sprite_animated[i].setColor(initial_colour);
        this->tile_improvement_sprite_animated[i].setScale(initial_scale);
    }

    this->render_window_ptr->draw(this->upgrade_arrow_sprite);


    //  2. draw power capacity upgrade text
    //                  16 char line = "                \n"
    std::string power_upgrade_string = "POWER CAPACITY  \n";
    power_upgrade_string            += "                \n";

    power_upgrade_string            += "CAPACITY:  ";
    power_upgrade_string            += std::to_string(this->capacity_kW);
    power_upgrade_string            += " kW\n";

    power_upgrade_string            += "LEVEL:     ";
    power_upgrade_string            += std::to_string(this->upgrade_level);
    power_upgrade_string            += "\n\n";

    if (this->upgrade_level < MAX_UPGRADE_LEVELS) {
        power_upgrade_string        += "[W]: + 100 kW (";
        power_upgrade_string        +=  std::to_string(WAVE_ENERGY_CONVERTER_BUILD_COST);
        power_upgrade_string        += " K)\n";
    }

    else {
        power_upgrade_string        += " * MAX LEVEL *  \n";
    }

    sf::Text power_upgrade_text = sf::Text(
        power_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    power_upgrade_text.setOrigin(power_upgrade_text.getLocalBounds().width / 2, 0);
    power_upgrade_text.setPosition(400 - 100, 400 - 32 + 16);
    power_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(power_upgrade_text);


    //  3. draw energy capacity (storage) upgrade sprite
    this->render_window_ptr->draw(this->storage_upgrade_sprite);
    this->render_window_ptr->draw(this->upgrade_plus_sprite);


    //  4. draw energy capacity (storage) upgrade text
    //                  16 char line = "                \n"
    std::string energy_upgrade_string = "ENERGY CAPACITY \n";
    energy_upgrade_string            += "                \n";

    energy_upgrade_string            += "CAPACITY:  ";
    energy_upgrade_string            += std::to_string(this->storage_level * 200);
    energy_upgrade_string            += " kWh\n";

    energy_upgrade_string            += "LEVEL:     ";
    energy_upgrade_string            += std::to_string(this->storage_level);
    energy_upgrade_string            += "\n\n";

    if (this->storage_level < MAX_STORAGE_LEVELS) {
        energy_upgrade_string        += "[D]: + 200 kWh (";
        energy_upgrade_string        +=  std::to_string(ENERGY_STORAGE_SYSTEM_BUILD_COST);
        energy_upgrade_string        += " K)\n";
    }

    else {
        energy_upgrade_string += " * MAX LEVEL *  \n";
    }

    sf::Text energy_upgrade_text = sf::Text(
        energy_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    energy_upgrade_text.setOrigin(energy_upgrade_text.getLocalBounds().width / 2, 0);
    energy_upgrade_text.setPosition(400 + 100, 400 - 32 + 16);
    energy_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(energy_upgrade_text);

    return;
}   /* __drawUpgradeOptions() */
```

4.14.3.8 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::U): {
            this->__openUpgradeMenu();

            break;
        }


        case (sf::Keyboard::W): {
            if (this->production_menu_open) {
                this->dispatch_MWh++;

                if (this->dispatch_MWh > this->dispatchable_MWh) {
                    this->dispatch_MWh = 0;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            else if (this->upgrade_menu_open) {
                this->__upgradePowerCapacity();
            }

            break;
        }


        case (sf::Keyboard::S): {
            if (this->production_menu_open) {
                this->dispatch_MWh--;

                if (this->dispatch_MWh < 0) {
                    this->dispatch_MWh = this->dispatchable_MWh;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            break;
        }


        case (sf::Keyboard::D): {
            if (this->upgrade_menu_open) {
                this->__upgradeStorageCapacity();
                this->__computeProduction();
                this->__computeDispatch();
            }

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.14.3.9 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    if (this->just_built) {
        return;
    }
    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.14.3.10 __sendImprovementStateMessage()

```
void WaveEnergyConverter::__sendImprovementStateMessage (
            void ) [private]
```

Helper method to format and sent improvement state message.

```
{
    Message improvement_state_message;

    improvement_state_message.channel = GAME_CHANNEL;
    improvement_state_message.subject = "improvement state";

    improvement_state_message.int_payload["dispatch_MWh"] = this->dispatch_MWh;
    improvement_state_message.int_payload["operation_maintenance_cost"] =
        this->operation_maintenance_cost;

    this->message_hub_ptr->sendMessage(improvement_state_message);

    std::cout « "Improvement state message sent by " « this « std::endl;

    return;
}   /* __sendImprovementStateMessage() */
```

4.14.3.11 __setUpTileImprovementSpriteAnimated()

```
void WaveEnergyConverter::__setUpTileImprovementSpriteAnimated (
            void ) [private]
```

Helper method to set up tile improvement sprite (static).

```
{
    sf::Sprite diesel_generator_sheet(
        *(this->assets_manager_ptr->getTexture("wave energy converter"))
    );

    int n_elements = diesel_generator_sheet.getLocalBounds().height / 64;

    for (int i = 0; i < n_elements; i++) {
        this->tile_improvement_sprite_animated.push_back(
            sf::Sprite(
                *(this->assets_manager_ptr->getTexture("wave energy converter")),
                sf::IntRect(0, i * 64, 64, 64)
            )
        );

        this->tile_improvement_sprite_animated.back().setOrigin(
            this->tile_improvement_sprite_animated.back().getLocalBounds().width / 2,
            this->tile_improvement_sprite_animated.back().getLocalBounds().height
        );

        this->tile_improvement_sprite_animated.back().setPosition(
            this->position_x,
            this->position_y - 32
        );

        this->tile_improvement_sprite_animated.back().setColor(
            sf::Color(255, 255, 255, 0)
        );
    }

    return;
}   /* __setUpTileImprovementSpriteAnimated() */
```

4.14.3.12 __upgradePowerCapacity()

```
void WaveEnergyConverter::__upgradePowerCapacity (
              void ) [private]
```

Helper method to upgrade power capacity.

```
{
    if (this->credits < WAVE_ENERGY_CONVERTER_BUILD_COST) {
        std::cout « "Cannot upgrade wave energy converter: insufficient credits (need "
            « WAVE_ENERGY_CONVERTER_BUILD_COST « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    if (this->upgrade_level >= MAX_UPGRADE_LEVELS) {
        return;
    }

    this->health = 100;

    this->capacity_kW += 100;
    this->upgrade_level++;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->__computeProduction();
    this->__computeDispatch();

    this->just_upgraded = true;

    this->assets_manager_ptr->getSound("upgrade")->play();

    this->__sendCreditsSpentMessage(WAVE_ENERGY_CONVERTER_BUILD_COST);
    this->__sendTileStateRequest();
    this->__sendGameStateRequest();

    return;
}   /* __upgradePowerCapacity() */
```

4.14.3.13 advanceTurn()

```
void WaveEnergyConverter::advanceTurn (
              void ) [virtual]
```

Method to handle turn advance.

Reimplemented from TileImprovement.

```
{
    //  1. update
    this->__computeCapacityFactors();
    this->update();

    //  2. send improvement state message
    this->__sendImprovementStateMessage();

    //  3. handle start/stop
    if ((not this->is_running) and (this->dispatch_MWh > 0)) {
        this->is_running = true;
    }

    else if (this->is_running and (this->dispatch_MWh <= 0)) {
        this->is_running = false;
    }

    //  4. handle equipment health
    if (this->is_running) {
        this->health--;

        if (this->health <= 0) {
            this->__breakdown();
        }
    }

    //  5. send tile state request (if selected)
    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* advanceTurn() */
```

4.14.3.14 draw()

```
void WaveEnergyConverter::draw (
            void  )  [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    //  1. if just built, call base method and return
    if (this->just_built) {
        TileImprovement :: draw();

        return;
    }


    //  2. handle upgrade effects
    if (this->just_upgraded) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(
                    255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    255,
                    255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    255
                )
            );

            this->tile_improvement_sprite_animated[i].setScale(
                sf::Vector2f(
                    1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2)
                )
            );
        }

        this->upgrade_frame++;
    }

    if (this->upgrade_frame >= 2 * FRAMES_PER_SECOND) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(255,255,255,255)
            );

            this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1,1));
        }

        this->just_upgraded = false;
        this->upgrade_frame = 0;
    }


    //  3. draw first element of animated sprite
    this->render_window_ptr->draw(this->tile_improvement_sprite_animated[0]);


    //  4. draw second element of animated sprite
    if (this->is_running) {
        this->tile_improvement_sprite_animated[0].setPosition(
            this->position_x,
            this->position_y + this->bobbing_y * cos(
                (double)(0.4 * M_PI * this->frame) / FRAMES_PER_SECOND
            )
        );

        this->tile_improvement_sprite_animated[1].setPosition(
            this->position_x,
            this->position_y + 1.25 * this->bobbing_y * sin(
                (double)(0.4 * M_PI * this->frame) / FRAMES_PER_SECOND
            )
        );
    }

    else {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setPosition(
                this->position_x,
                this->position_y + this->bobbing_y * cos(
                    (double)(0.4 * M_PI * this->frame) / FRAMES_PER_SECOND
                )
            );
        }
    }

    this->render_window_ptr->draw(this->tile_improvement_sprite_animated[1]);


    //  5. draw storage upgrades
    for (size_t i = 0; i < this->storage_upgrade_sprite_vec.size(); i++) {
        this->render_window_ptr->draw(this->storage_upgrade_sprite_vec[i]);
    }


    //  6. handle dispatch illustration
    if (this->dispatch_MWh > 0) {
        this->dispatch_text.setString(std::to_string(this->dispatch_MWh));
        this->__drawDispatch();
    }


    //  7. draw production menu
    if (this->production_menu_open) {
        this->render_window_ptr->draw(this->production_menu_backing);
        this->render_window_ptr->draw(this->production_menu_backing_text);

        this->__drawProductionMenu();
    }


    //  8. draw upgrade menu
    if (this->upgrade_menu_open) {
        this->render_window_ptr->draw(this->upgrade_menu_backing);
        this->render_window_ptr->draw(this->upgrade_menu_backing_text);

        this->__drawUpgradeOptions();
    }


    //  9. handle broken effects
    if (this->is_broken) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(
                    255,
                    255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                    255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                    255
                )
            );
        }
    }

    this->frame++;
    return;
}   /* draw() */
```

4.14.3.15 getTileOptionsSubstring()

```
std::string WaveEnergyConverter::getTileOptionsSubstring (
            void ) [virtual]
```

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

Reimplemented from TileImprovement.

```
{
    //                    32 char x 17 line console "--------------------------------\n";
    std::string options_substring                   = "CAPACITY:      ";
    options_substring                              += std::to_string(this->capacity_kW);
    options_substring                              += " kW (level ";
    options_substring                              += std::to_string(this->upgrade_level);
    options_substring                              += ")\n";

    options_substring                              += "PRODUCTION:    ";
    options_substring                              += std::to_string(this->production_MWh);
    options_substring                              += " MWh\n";

    options_substring                              += "DISPATCHABLE:  ";
    options_substring                              += std::to_string(this->dispatchable_MWh);
    options_substring                              += " MWh\n";

    options_substring                              += "HEALTH:      ";
    options_substring                              += std::to_string(this->health);
    options_substring                              += "/100";

    if (this->health <= 0) {
        options_substring                          += " ** BROKEN! **\n";
    }

    else {
        options_substring                          += "\n";
    }

    options_substring                              += "                                \n";
    options_substring                              += " **** WAVE ENERGY OPTIONS ****  \n";
    options_substring                              += "                                \n";

    options_substring                              += "     [E]:  ";

    if (this->is_broken) {
        options_substring                          += "*** BROKEN! ***\n";
    }

    else {
        options_substring                          += "OPEN PRODUCTION MENU\n";
    }

    options_substring                              += "     [U]:  OPEN UPGRADE MENU    \n";
    options_substring                              += "HOLD [P]:  SCRAP (";
    options_substring                              += std::to_string(SCRAP_COST);
    options_substring                              += " K)";

    return options_substring;
}   /* getTileOptionsSubstring() */
```

4.14.3.16 processEvent()

```
void WaveEnergyConverter::processEvent (
            void ) [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processEvent();

    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.14.3.17 processMessage()

```
void WaveEnergyConverter::processMessage (
            void ) [virtual]
```

Method to process WaveEnergyConverter. To be called once per message.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processMessage();

    //...

    return;
}   /* processMessage() */
```

4.14.3.18 setIsSelected()

```
void WaveEnergyConverter::setIsSelected (
            bool is_selected ) [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: setIsSelected(is_selected);

    if (this->is_running and this->is_selected) {
        this->assets_manager_ptr->getSound("ocean waves")->play();
    }

    return;
}   /* setIsSelected() */
```

4.14.3.19 update()

```
void WaveEnergyConverter::update (
            void ) [virtual]
```

Method to trigger production and dispatchable updates.

Reimplemented from TileImprovement.

```
{
    this->__computeProduction();
    this->__computeProductionCosts();
    this->__computeDispatch();

    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* update() */
```

4.14.4 Member Data Documentation

4.14.4.1 bobbing_y

```
double WaveEnergyConverter::bobbing_y
```

The bobbing extent of the wave energy converter.

4.14.4.2 capacity_factor_vec

```
std::vector<double> WaveEnergyConverter::capacity_factor_vec
```

A vector of daily capacity factors for the current month.

4.14.4.3 capacity_kW

```
int WaveEnergyConverter::capacity_kW
```

The rated production capacity [kW] of the solar PV array.

4.14.4.4 dispatch_MWh

```
int WaveEnergyConverter::dispatch_MWh
```

The current dispatch [MWh] of the solar PV array.

4.14.4.5 dispatch_vec_MWh

```
std::vector<double> WaveEnergyConverter::dispatch_vec_MWh
```

A vector of daily dispatch [MWh] for the current month.

4.14.4.6 dispatchable_MWh

```
int WaveEnergyConverter::dispatchable_MWh
```

The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).

4.14.4.7 max_daily_production_MWh

```
double WaveEnergyConverter::max_daily_production_MWh
```

The maximum daily production [MWh] of the solar PV array.

4.14.4.8 production_MWh

```
int WaveEnergyConverter::production_MWh
```

The current production [MWh] of the solar PV array.

4.14.4.9 production_vec_MWh

```
std::vector<double> WaveEnergyConverter::production_vec_MWh
```

A vector of daily production [MWh] for the current month.

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

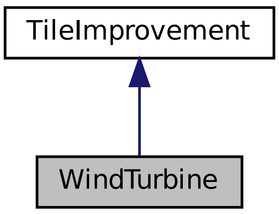
- header/WaveEnergyConverter.h
- source/WaveEnergyConverter.cpp

4.15 WindTurbine Class Reference

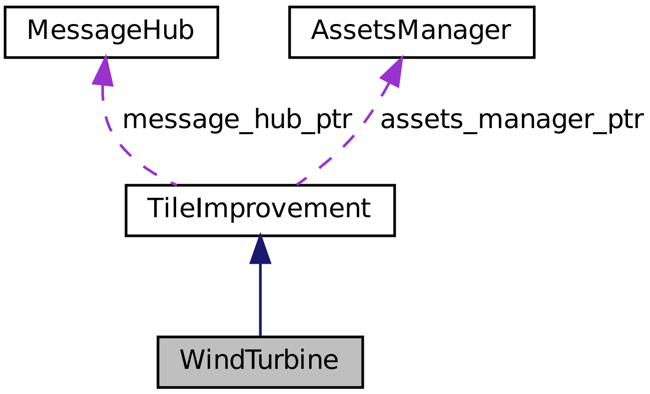
A settlement class (child class of TileImprovement).

```
#include <WindTurbine.h>
```

Inheritance diagram for WindTurbine:



Collaboration diagram for WindTurbine:



Public Member Functions

- WindTurbine (double, double, int, sf::Event ∗, sf::RenderWindow ∗, AssetsManager ∗, MessageHub ∗)

    Constructor for the WindTurbine class.
- std::string getTileOptionsSubstring (void)

    Helper method to assemble and return tile options substring.
- void setIsSelected (bool)

    Method to set the is selected attribute.
- void advanceTurn (void)

Method to handle turn advance.
- void update (void)

  Method to trigger production and dispatchable updates.
- void processEvent (void)

  Method to process WindTurbine. To be called once per event.
- void processMessage (void)

  Method to process WindTurbine. To be called once per message.
- void draw (void)

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

  Destructor for the WindTurbine class.

Public Attributes

- int capacity_kW

  The rated production capacity [kW] of the solar PV array.
- int production_MWh

  The current production [MWh] of the solar PV array.
- int dispatch_MWh

  The current dispatch [MWh] of the solar PV array.
- int dispatchable_MWh

  The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).
- double max_daily_production_MWh

  The maximum daily production [MWh] of the solar PV array.
- double rotor_drotation

  The rotation rate of the rotor.
- std::vector< double > capacity_factor_vec

  A vector of daily capacity factors for the current month.
- std::vector< double > production_vec_MWh

  A vector of daily production [MWh] for the current month.
- std::vector< double > dispatch_vec_MWh

  A vector of daily dispatch [MWh] for the current month.

Private Member Functions

- void __setUpTileImprovementSpriteAnimated (void)

  Helper method to set up tile improvement sprite (static).
- void __drawProductionMenu (void)

  Helper method to draw production menu assets.
- void __upgradePowerCapacity (void)

  Helper method to upgrade the power capacity.
- void __computeProductionCosts (void)

  Helper method to compute production costs (O&M) based on current production level.
- void __breakdown (void)

  Helper method to trigger an equipment breakdown.
- void __computeCapacityFactors (void)

  Helper method to compute capacity factors.
- void __computeProduction (void)

  Helper method to compute production values.

- void __computeDispatch (void)

    Helper method to compute dispatch values.
- void __handleKeyPressEvents (void)

    Helper method to handle key press events.
- void __handleMouseButtonEvents (void)

    Helper method to handle mouse button events.
- void __drawUpgradeOptions (void)

    Helper method to set up and draw upgrade options.
- void __sendImprovementStateMessage (void)

    Helper method to format and sent improvement state message.

Additional Inherited Members

4.15.1 Detailed Description

A settlement class (child class of TileImprovement).

4.15.2 Constructor & Destructor Documentation

4.15.2.1 WindTurbine()

```
WindTurbine::WindTurbine (
                double position_x,
                double position_y,
                int tile_resource,
                sf::Event * event_ptr,
                sf::RenderWindow * render_window_ptr,
                AssetsManager * assets_manager_ptr,
                MessageHub * message_hub_ptr )
```

Constructor for the WindTurbine class.

Ref: Wikipedia [2023]

Parameters

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

```
735    :
736 TileImprovement(
    position_x,
    position_y,
    tile_resource,
    event_ptr,
    render_window_ptr,
    assets_manager_ptr,
    message_hub_ptr
)
{
    //  1. set attributes

    //  1.1. private
    //...

    //  1.2. public
    this->tile_improvement_type = TileImprovementType :: WIND_TURBINE;

    this->is_running = false;

    this->health = 100;

    this->capacity_kW = 100;
    this->upgrade_level = 1;

    this->storage_kWh = 0;
    this->storage_level = 0;

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->rotor_drotation = 256 * SECONDS_PER_FRAME;

    this->capacity_factor_vec.resize(30, 0);
    this->production_vec_MWh.resize(30, 0);
    this->dispatch_vec_MWh.resize(30, 0);

    this->tile_improvement_string = "WIND TURBINE";

    this->__setUpTileImprovementSpriteAnimated();
    this->__computeCapacityFactors();
    this->update();

    std::cout « "WindTurbine constructed at " « this « std::endl;

    return;
}   /* WindTurbine() */
```

4.15.2.2 ∼WindTurbine()

```
WindTurbine::∼WindTurbine (
              void  )  [virtual]
```

Destructor for the WindTurbine class.

```
{
    std::cout « "WindTurbine at " « this « " destroyed" « std::endl;

    return;
}   /* ~WindTurbine() */
```

4.15.3 Member Function Documentation

4.15.3.1 __breakdown()

```
void WindTurbine::__breakdown (
            void ) [private]
```

Helper method to trigger an equipment breakdown.

```
{
    TileImprovement :: __breakdown();

    this->production_MWh = 0;
    this->dispatch_MWh = 0;
    this->dispatchable_MWh = 0;
    this->operation_maintenance_cost = 0;

    return;
}   /* __breakdown() */
```

4.15.3.2 __computeCapacityFactors()

```
void WindTurbine::__computeCapacityFactors (
            void ) [private]
```

Helper method to compute capacity factors.

```
{
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator(seed);

    double mean =
        this->tile_resource_scalar * MEAN_DAILY_WIND_CAPACITY_FACTORS[this->month - 1];

    double stdev = STDEV_DAILY_WIND_CAPACITY_FACTORS[this->month - 1];

    if (this->tile_resource_scalar > 1) {
        stdev /= this->tile_resource_scalar;
    }

    std::normal_distribution<double> normal_dist(mean, stdev);

    double capacity_factor = 0;

    for (int i = 0; i < 30; i++) {
        capacity_factor = normal_dist(generator);

        if (capacity_factor < 0) {
            capacity_factor = 0;
        }

        this->capacity_factor_vec[i] = capacity_factor;
    }

    return;
}   /* __computeCapacityFactors() */
```

4.15.3.3 __computeDispatch()

```
void WindTurbine::__computeDispatch (
            void ) [private]
```

Helper method to compute dispatch values.

```
{
    double stored_energy_MWh = 0;
    double storage_capacity_MWh = (double)(this->storage_kWh) / 1000;

    double demand_MWh = 0;
    double production_MWh = 0;
    double dispatchable_MWh = 0;
    double difference_MWh = 0;

    double room_MWh = 0;

    for (int i = 0; i < 30; i++) {
        demand_MWh = this->demand_vec_MWh[i];
        production_MWh = this->production_vec_MWh[i];

        if (production_MWh <= demand_MWh) {
            this->dispatch_vec_MWh[i] = production_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = demand_MWh - production_MWh;

            if ((storage_capacity_MWh > 0) and (stored_energy_MWh > 0)) {
                if (difference_MWh > stored_energy_MWh) {
                    this->dispatch_vec_MWh[i] += stored_energy_MWh;
                    dispatchable_MWh += stored_energy_MWh;
                    stored_energy_MWh = 0;
                }

                else {
                    this->dispatch_vec_MWh[i] += difference_MWh;
                    dispatchable_MWh += difference_MWh;
                    stored_energy_MWh -= difference_MWh;
                }
            }
        }

        else {
            this->dispatch_vec_MWh[i] = demand_MWh;
            dispatchable_MWh += this->dispatch_vec_MWh[i];

            difference_MWh = production_MWh - demand_MWh;

            if (
                (storage_capacity_MWh > 0) and
                (stored_energy_MWh < storage_capacity_MWh)
            ) {
                room_MWh = storage_capacity_MWh - stored_energy_MWh;

                if (difference_MWh > room_MWh) {
                    stored_energy_MWh += room_MWh;
                }

                else {
                    stored_energy_MWh += difference_MWh;
                }
            }
        }
    }

    this->dispatchable_MWh = round(dispatchable_MWh);

    if (this->dispatch_MWh > this->dispatchable_MWh) {
        this->dispatch_MWh = this->dispatchable_MWh;
    }

    return;
}   /* __computeDispatch() */
```

4.15.3.4 __computeProduction()

```
void WindTurbine::__computeProduction (
            void ) [private]
```

Helper method to compute production values.

```
{
    double production_MWh = 0;

    for (int i = 0; i < 30; i++) {
        this->production_vec_MWh[i] =
            this->max_daily_production_MWh * this->capacity_factor_vec[i];

        production_MWh += this->production_vec_MWh[i];
    }

    this->production_MWh = round(production_MWh);

    return;
}   /* __computeProduction() */
```

4.15.3.5 __computeProductionCosts()

```
void WindTurbine::__computeProductionCosts (
            void ) [private]
```

Helper method to compute production costs (O&M) based on current production level.

```
{
    double operation_maintenance_cost =
        (this->production_MWh * WIND_OP_MAINT_COST_PER_MWH_PRODUCTION) / 1000;
    this->operation_maintenance_cost = round(operation_maintenance_cost);

    return;
}   /* __computeProductionCosts() */
```

4.15.3.6 __drawProductionMenu()

```
void WindTurbine::__drawProductionMenu (
            void ) [private]
```

Helper method to draw production menu assets.

```
{
    //  1. draw static sprite
    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
        sf::Vector2f initial_position = this->tile_improvement_sprite_animated[i].getPosition();
        this->tile_improvement_sprite_animated[i].setPosition(400 - 138, 400 + 16);

        sf::Color initial_colour = this->tile_improvement_sprite_animated[i].getColor();
        this->tile_improvement_sprite_animated[i].setColor(sf::Color(255, 255, 255, 255));

        sf::Vector2f initial_scale = this->tile_improvement_sprite_animated[i].getScale();
        this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1, 1));

        double initial_rotation = this->tile_improvement_sprite_animated[i].getRotation();
        this->tile_improvement_sprite_animated[i].setRotation(0);

        this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);

        this->tile_improvement_sprite_animated[i].setPosition(initial_position);
        this->tile_improvement_sprite_animated[i].setColor(initial_colour);
        this->tile_improvement_sprite_animated[i].setScale(initial_scale);
        this->tile_improvement_sprite_animated[i].setRotation(initial_rotation);
    }

    //  2. draw production text
    std::string production_string = "[W]:  INCREASE DISPATCH\n";
    production_string            += "[S]:  DECREASE DISPATCH\n";
    production_string            += "                         \n";

    production_string            += "DISPATCH:  ";
    production_string            += std::to_string(this->dispatch_MWh);
    production_string            += " MWh (MAX ";
    production_string            += std::to_string(this->dispatchable_MWh);
    production_string            += ")\n";

    production_string            += "O&M COST:  ";
    production_string            += std::to_string(this->operation_maintenance_cost);
    production_string            += " K\n";

    sf::Text production_text(
        production_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    production_text.setOrigin(production_text.getLocalBounds().width / 2,0);
    production_text.setFillColor(MONOCHROME_TEXT_GREEN);

    production_text.setPosition(400 + 30, 400 - 45);

    this->render_window_ptr->draw(production_text);

    return;
}   /* __drawProductionMenu() */
```

4.15.3.7 __drawUpgradeOptions()

```
void WindTurbine::__drawUpgradeOptions (
            void ) [private]
```

Helper method to set up and draw upgrade options.

```
{
    //  1. draw power capacity upgrade sprite
    for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
        sf::Vector2f initial_position = this->tile_improvement_sprite_animated[i].getPosition();
        this->tile_improvement_sprite_animated[i].setPosition(400 - 100, 400 - 56);

        sf::Color initial_colour = this->tile_improvement_sprite_animated[i].getColor();
        this->tile_improvement_sprite_animated[i].setColor(sf::Color(255, 255, 255, 255));

        sf::Vector2f initial_scale = this->tile_improvement_sprite_animated[i].getScale();
        this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1, 1));

        double initial_rotation = this->tile_improvement_sprite_animated[i].getRotation();
        this->tile_improvement_sprite_animated[i].setRotation(0);

        this->render_window_ptr->draw(this->tile_improvement_sprite_animated[i]);

        this->tile_improvement_sprite_animated[i].setPosition(initial_position);
        this->tile_improvement_sprite_animated[i].setColor(initial_colour);
        this->tile_improvement_sprite_animated[i].setScale(initial_scale);
        this->tile_improvement_sprite_animated[i].setRotation(initial_rotation);
    }

    this->render_window_ptr->draw(this->upgrade_arrow_sprite);


    //  2. draw power capacity upgrade text
    //                  16 char line = "                \n"
    std::string power_upgrade_string = "POWER CAPACITY  \n";
    power_upgrade_string            += "                \n";

    power_upgrade_string            += "CAPACITY:  ";
    power_upgrade_string            += std::to_string(this->capacity_kW);
    power_upgrade_string            += " kW\n";

    power_upgrade_string            += "LEVEL:     ";
    power_upgrade_string            += std::to_string(this->upgrade_level);
    power_upgrade_string            += "\n\n";

    if (this->upgrade_level < MAX_UPGRADE_LEVELS) {
        power_upgrade_string        += "[W]: + 100 kW (";
        power_upgrade_string        +=  std::to_string(WIND_TURBINE_BUILD_COST);
        power_upgrade_string        += " K)\n";
    }

    else {
        power_upgrade_string        += " * MAX LEVEL *  \n";
    }

    sf::Text power_upgrade_text = sf::Text(
        power_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    power_upgrade_text.setOrigin(power_upgrade_text.getLocalBounds().width / 2, 0);
    power_upgrade_text.setPosition(400 - 100, 400 - 32 + 16);
    power_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(power_upgrade_text);


    //  3. draw energy capacity (storage) upgrade sprite
    this->render_window_ptr->draw(this->storage_upgrade_sprite);
    this->render_window_ptr->draw(this->upgrade_plus_sprite);


    //  4. draw energy capacity (storage) upgrade text
    //                    16 char line = "                \n"
    std::string energy_upgrade_string = "ENERGY CAPACITY \n";
    energy_upgrade_string            += "                \n";

    energy_upgrade_string            += "CAPACITY:  ";
    energy_upgrade_string            += std::to_string(this->storage_level * 200);
    energy_upgrade_string            += " kWh\n";

    energy_upgrade_string            += "LEVEL:     ";
    energy_upgrade_string            += std::to_string(this->storage_level);
    energy_upgrade_string            += "\n\n";

    if (this->storage_level < MAX_STORAGE_LEVELS) {
        energy_upgrade_string        += "[D]: + 200 kWh (";
        energy_upgrade_string        +=  std::to_string(ENERGY_STORAGE_SYSTEM_BUILD_COST);
        energy_upgrade_string        += " K)\n";
    }

    else {
        energy_upgrade_string += " * MAX LEVEL *  \n";
    }

    sf::Text energy_upgrade_text = sf::Text(
        energy_upgrade_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        16
    );

    energy_upgrade_text.setOrigin(energy_upgrade_text.getLocalBounds().width / 2, 0);
    energy_upgrade_text.setPosition(400 + 100, 400 - 32 + 16);
    energy_upgrade_text.setFillColor(MONOCHROME_TEXT_GREEN);

    this->render_window_ptr->draw(energy_upgrade_text);

    return;
}   /* __drawUpgradeOptions() */
```

4.15.3.8 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->key.code) {
        case (sf::Keyboard::U): {
            this->__openUpgradeMenu();

            break;
        }


        case (sf::Keyboard::W): {
            if (this->production_menu_open) {
                this->dispatch_MWh++;

                if (this->dispatch_MWh > this->dispatchable_MWh) {
                    this->dispatch_MWh = 0;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            else if (this->upgrade_menu_open) {
                this->__upgradePowerCapacity();
            }

            break;
        }


        case (sf::Keyboard::S): {
            if (this->production_menu_open) {
                this->dispatch_MWh--;

                if (this->dispatch_MWh < 0) {
                    this->dispatch_MWh = this->dispatchable_MWh;
                }

                this->__computeProductionCosts();
                this->assets_manager_ptr->getSound("interface click")->play();
            }

            break;
        }


        case (sf::Keyboard::D): {
            if (this->upgrade_menu_open) {
                this->__upgradeStorageCapacity();
                this->__computeProduction();
                this->__computeDispatch();
            }

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleKeyPressEvents() */
```

4.15.3.9 __handleMouseButtonEvents()

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

Helper method to handle mouse button events.

```
{
    if (this->just_built) {
        return;
    }

    switch (this->event_ptr->mouseButton.button) {
        case (sf::Mouse::Left): {
            //...

            break;
        }


        case (sf::Mouse::Right): {
            //...

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

    return;
}   /* __handleMouseButtonEvents() */
```

4.15.3.10 __sendImprovementStateMessage()

```
void WindTurbine::__sendImprovementStateMessage (
            void ) [private]
```

Helper method to format and sent improvement state message.

```
{
    Message improvement_state_message;

    improvement_state_message.channel = GAME_CHANNEL;
    improvement_state_message.subject = "improvement state";

    improvement_state_message.int_payload["dispatch_MWh"] = this->dispatch_MWh;
    improvement_state_message.int_payload["operation_maintenance_cost"] =
        this->operation_maintenance_cost;

    this->message_hub_ptr->sendMessage(improvement_state_message);

    std::cout « "Improvement state message sent by " « this « std::endl;

    return;
}   /* __sendImprovementStateMessage() */
```

4.15.3.11 __setUpTileImprovementSpriteAnimated()

```
void WindTurbine::__setUpTileImprovementSpriteAnimated (
            void ) [private]
```

Helper method to set up tile improvement sprite (static).

```
{
    sf::Sprite diesel_generator_sheet(
        *(this->assets_manager_ptr->getTexture("wind turbine"))
    );

    int n_elements = diesel_generator_sheet.getLocalBounds().height / 64;

    for (int i = 0; i < n_elements; i++) {
        this->tile_improvement_sprite_animated.push_back(
            sf::Sprite(
                *(this->assets_manager_ptr->getTexture("wind turbine")),
                sf::IntRect(0, i * 64, 64, 64)
            )
        );

        this->tile_improvement_sprite_animated.back().setOrigin(
            this->tile_improvement_sprite_animated.back().getLocalBounds().width / 2,
            this->tile_improvement_sprite_animated.back().getLocalBounds().height
        );

        this->tile_improvement_sprite_animated.back().setPosition(
            this->position_x,
            this->position_y - 32
        );

        this->tile_improvement_sprite_animated.back().setColor(
            sf::Color(255, 255, 255, 0)
        );
    }

    return;
}   /* __setUpTileImprovementSpriteAnimated() */
```

4.15.3.12 __upgradePowerCapacity()

```
void WindTurbine::__upgradePowerCapacity (
              void   )  [private]
```

Helper method to upgrade the power capacity.

```
{
    if (this->credits < WIND_TURBINE_BUILD_COST) {
        std::cout « "Cannot upgrade wind turbine: insufficient credits (need "
            « WIND_TURBINE_BUILD_COST « " K)" « std::endl;

        this->__sendInsufficientCreditsMessage();
        return;
    }

    if (this->upgrade_level >= MAX_UPGRADE_LEVELS) {
        return;
    }

    this->health = 100;

    this->capacity_kW += 100;
    this->upgrade_level++;

    this->max_daily_production_MWh = (double)(24 * this->capacity_kW) / 1000;

    this->__computeProduction();
    this->__computeDispatch();

    this->just_upgraded = true;

    this->assets_manager_ptr->getSound("upgrade")->play();

    this->__sendCreditsSpentMessage(WIND_TURBINE_BUILD_COST);
    this->__sendTileStateRequest();
    this->__sendGameStateRequest();

    return;
}   /* __upgradePowerCapacity() */
```

4.15.3.13 advanceTurn()

```
void WindTurbine::advanceTurn (
              void   )  [virtual]
```

Method to handle turn advance.

Reimplemented from TileImprovement.

```
{
    //  1. update
    this->__computeCapacityFactors();
    this->update();

    //  2. send improvement state message
    this->__sendImprovementStateMessage();

    //  3. handle start/stop
    if ((not this->is_running) and (this->dispatch_MWh > 0)) {
        this->is_running = true;
    }

    else if (this->is_running and (this->dispatch_MWh <= 0)) {
        this->is_running = false;
    }

    //  4. handle equipment health
    if (this->is_running) {
        this->health--;

        if (this->health <= 0) {
            this->__breakdown();
        }
    }

    //  5. send tile state request (if selected)
    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* advanceTurn() */
```

4.15.3.14 draw()

```
void WindTurbine::draw (
            void ) [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    //  1. if just built, call base method and return
    if (this->just_built) {
        TileImprovement :: draw();

        return;
    }


    //  2. handle upgrade effects
    if (this->just_upgraded) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(
                    255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    255,
                    255 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    255
                )
            );

            this->tile_improvement_sprite_animated[i].setScale(
                sf::Vector2f(
                    1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2),
                    1 + 0.2 * pow(cos((M_PI * this->upgrade_frame) / FRAMES_PER_SECOND), 2)
                )
            );
        }

        this->upgrade_frame++;
    }

    if (this->upgrade_frame >= 2 * FRAMES_PER_SECOND) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(255,255,255,255)
            );

            this->tile_improvement_sprite_animated[i].setScale(sf::Vector2f(1,1));
        }

        this->just_upgraded = false;
        this->upgrade_frame = 0;
    }


    //  3. draw first element of animated sprite
    this->render_window_ptr->draw(this->tile_improvement_sprite_animated[0]);


    //  4. draw second element of animated sprite
    if (this->is_running) {
        this->tile_improvement_sprite_animated[1].rotate(this->rotor_drotation);
    }

    this->render_window_ptr->draw(this->tile_improvement_sprite_animated[1]);


    //  5. draw storage upgrades
    for (size_t i = 0; i < this->storage_upgrade_sprite_vec.size(); i++) {
        this->render_window_ptr->draw(this->storage_upgrade_sprite_vec[i]);
    }


    //  6. handle dispatch illustration
    if (this->dispatch_MWh > 0) {
        this->dispatch_text.setString(std::to_string(this->dispatch_MWh));
        this->__drawDispatch();
    }


    //  7. draw production menu
    if (this->production_menu_open) {
        this->render_window_ptr->draw(this->production_menu_backing);
        this->render_window_ptr->draw(this->production_menu_backing_text);

        this->__drawProductionMenu();
    }


    //  8. draw upgrade menu
    if (this->upgrade_menu_open) {
        this->render_window_ptr->draw(this->upgrade_menu_backing);
        this->render_window_ptr->draw(this->upgrade_menu_backing_text);

        this->__drawUpgradeOptions();
    }


    //  9. handle broken effects
    if (this->is_broken) {
        for (size_t i = 0; i < this->tile_improvement_sprite_animated.size(); i++) {
            this->tile_improvement_sprite_animated[i].setColor(
                sf::Color(
                    255,
                    255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                    255 * pow(cos((M_PI * this->frame) / FRAMES_PER_SECOND), 2),
                    255
                )
            );
        }
    }

    this->frame++;
    return;
}   /* draw() */
```

4.15.3.15 getTileOptionsSubstring()

```
std::string WindTurbine::getTileOptionsSubstring (
              void  )  [virtual]
```

Helper method to assemble and return tile options substring.

Returns

Tile options substring.

Reimplemented from TileImprovement.

```
{
    //                    32 char x 17 line console "--------------------------------\n";
    std::string options_substring                  = "CAPACITY:      ";
    options_substring                             += std::to_string(this->capacity_kW);
    options_substring                             += " kW (level ";
    options_substring                             += std::to_string(this->upgrade_level);
    options_substring                             += ")\n";

    options_substring                             += "PRODUCTION:    ";
    options_substring                             += std::to_string(this->production_MWh);
    options_substring                             += " MWh\n";

    options_substring                             += "DISPATCHABLE:  ";
    options_substring                             += std::to_string(this->dispatchable_MWh);
    options_substring                             += " MWh\n";

    options_substring                              += "HEALTH:      ";
    options_substring                              += std::to_string(this->health);
    options_substring                              += "/100";

    if (this->health <= 0) {
        options_substring                          += " ** BROKEN! **\n";
    }

    else {
        options_substring                          += "\n";
    }

    options_substring                              += "                                 \n";
    options_substring                              += " **** WIND TURBINE OPTIONS **** \n";
    options_substring                              += "                                 \n";

    options_substring                              += "      [E]:  ";

    if (this->is_broken) {
        options_substring                          += "*** BROKEN! ***\n";
    }

    else {
        options_substring                          += "OPEN PRODUCTION MENU\n";
    }

    options_substring                              += "      [U]:  OPEN UPGRADE MENU    \n";
    options_substring                              += "HOLD [P]:  SCRAP (";
    options_substring                              += std::to_string(SCRAP_COST);
    options_substring                              += " K)";

    return options_substring;
}   /* getTileOptionsSubstring() */
```

4.15.3.16 processEvent()

```
void WindTurbine::processEvent (
              void  )  [virtual]
```

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

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processEvent();

    if (this->event_ptr->type == sf::Event::KeyPressed) {
        this->__handleKeyPressEvents();
    }

    if (this->event_ptr->type == sf::Event::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

    return;
}   /* processEvent() */
```

4.15.3.17 processMessage()

```
void WindTurbine::processMessage (
              void  )  [virtual]
```

Method to process WindTurbine. To be called once per message.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: processMessage();

    //...

    return;
}   /* processMessage() */
```

4.15.3.18 setIsSelected()

```
void WindTurbine::setIsSelected (
            bool is_selected )  [virtual]
```

Method to set the is selected attribute.

Parameters

is_selected	The value to set the is selected attribute to.

Reimplemented from TileImprovement.

```
{
    TileImprovement :: setIsSelected(is_selected);

    if (this->is_running and this->is_selected) {
        this->assets_manager_ptr->getSound("wind turbine running")->play();
    }

    return;
}   /* setIsSelected() */
```

4.15.3.19 update()

```
void WindTurbine::update (
            void  )  [virtual]
```

Method to trigger production and dispatchable updates.

Reimplemented from TileImprovement.

```
{
    this->__computeProduction();
    this->__computeProductionCosts();
    this->__computeDispatch();

    if (this->is_selected) {
        this->__sendTileStateRequest();
    }

    return;
}   /* update() */
```

4.15.4 Member Data Documentation

4.15.4.1 capacity_factor_vec

```
std::vector<double> WindTurbine::capacity_factor_vec
```

A vector of daily capacity factors for the current month.

4.15.4.2 capacity_kW

```
int WindTurbine::capacity_kW
```

The rated production capacity [kW] of the solar PV array.

4.15.4.3 dispatch_MWh

```
int WindTurbine::dispatch_MWh
```

The current dispatch [MWh] of the solar PV array.

4.15.4.4 dispatch_vec_MWh

```
std::vector<double> WindTurbine::dispatch_vec_MWh
```

A vector of daily dispatch [MWh] for the current month.

4.15.4.5 dispatchable_MWh

```
int WindTurbine::dispatchable_MWh
```

The amount of production that is directly dispatchable to the grid (i.e. production correlated with demand).

4.15.4.6 max_daily_production_MWh

```
double WindTurbine::max_daily_production_MWh
```

The maximum daily production [MWh] of the solar PV array.

4.15.4.7 production_MWh

```
int WindTurbine::production_MWh
```

The current production [MWh] of the solar PV array.

4.15.4.8 production_vec_MWh

```
std::vector<double> WindTurbine::production_vec_MWh
```

A vector of daily production [MWh] for the current month.

4.15.4.9 rotor_drotation

```
double WindTurbine::rotor_drotation
```

The rotation rate of the rotor.

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

- header/WindTurbine.h
- source/WindTurbine.cpp

Chapter 5

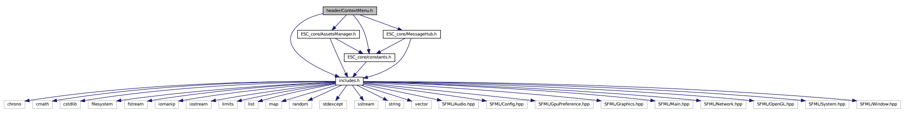
File Documentation

5.1 header/ContextMenu.h File Reference

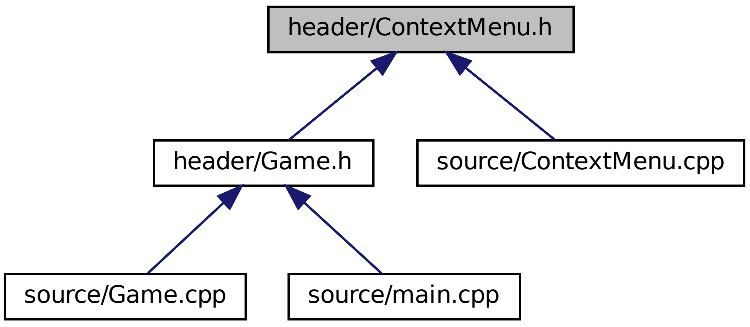
Header file for the ContextMenu class.

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

Include dependency graph for ContextMenu.h:



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



Classes

- class ContextMenu

  A class which defines a context menu for the game.

Enumerations

- enum ConsoleState {
  NONE_STATE , READY , MENU , TILE ,
  N_CONSOLE_STATES }

  An enumeration of the different console screen states.

5.1.1 Detailed Description

Header file for the ContextMenu class.

5.1.2 Enumeration Type Documentation

5.1.2.1 ConsoleState

```
enum ConsoleState
```

An enumeration of the different console screen states.

Enumerator

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

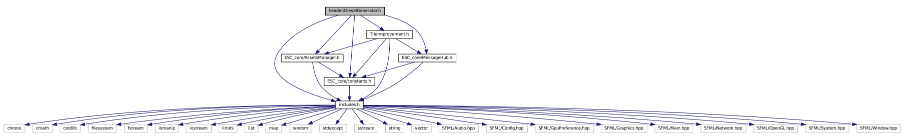
```
                {
    NONE_STATE,
    READY,
    MENU,
    TILE,
    N_CONSOLE_STATES
};
```

5.2 header/DieselGenerator.h File Reference

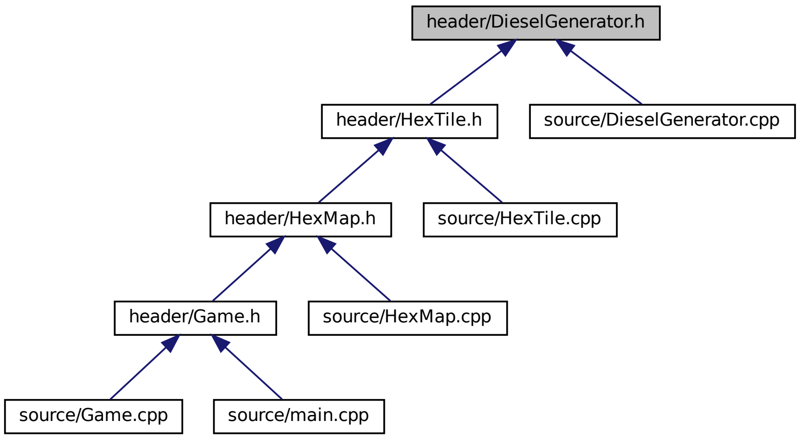
Header file for the DieselGenerator class.

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

```
#include "ESC_core/MessageHub.h"
#include "TileImprovement.h"
```
Include dependency graph for DieselGenerator.h:



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



Classes

- class DieselGenerator

  A settlement class (child class of TileImprovement).

5.2.1 Detailed Description

Header file for the DieselGenerator class.

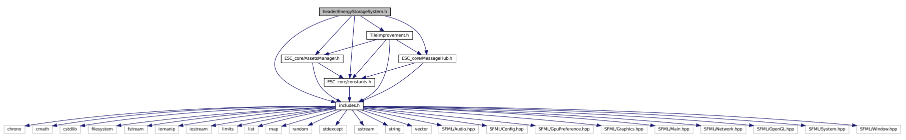
5.3 header/EnergyStorageSystem.h File Reference

Header file for the EnergyStorageSystem class.

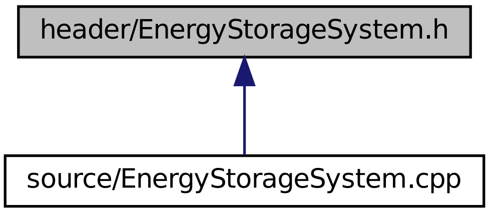
```
#include "ESC_core/constants.h"
#include "ESC_core/includes.h"
#include "ESC_core/AssetsManager.h"
#include "ESC_core/MessageHub.h"
```

```
#include "TileImprovement.h"
```

Include dependency graph for EnergyStorageSystem.h:



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



Classes

- class EnergyStorageSystem

  A settlement class (child class of TileImprovement).

5.3.1 Detailed Description

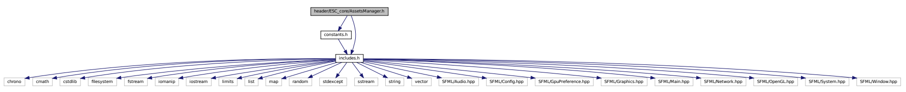
Header file for the EnergyStorageSystem class.

5.4 header/ESC_core/AssetsManager.h File Reference

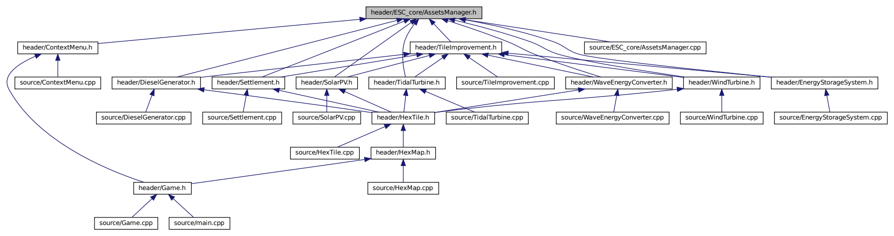
Header file for the AssetsManager class.

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

Include dependency graph for AssetsManager.h:



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



Classes

- class AssetsManager

  A class which manages visual and sound assets.

5.4.1 Detailed Description

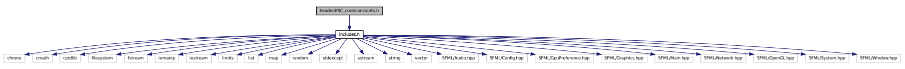
Header file for the AssetsManager class.

5.5 header/ESC_core/constants.h File Reference

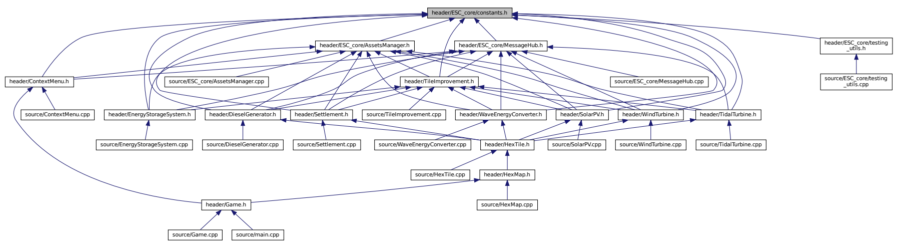
Header file for various constants.

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

Include dependency graph for constants.h:



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


header/ESC_core/constants.h
header/ESC_core/AssetsManager.h
header/ESC_core/MessageHub.h
header/ESC_core/testing_utils.h
source/ESC_core/testing_utils.cpp
header/ContextMenu.h
source/ESC_core/AssetsManager.cpp
header/TileImprovement.h
source/ESC_core/MessageHub.cpp
source/ContextMenu.cpp
header/EnergyStorageSystem.h
header/DieselGenerator.h
header/Settlement.h
source/TileImprovement.cpp
header/WaveEnergyConverter.h
header/SolarPV.h
header/WindTurbine.h
header/TidalTurbine.h
source/EnergyStorageSystem.cpp
source/DieselGenerator.cpp
source/Settlement.cpp
source/WaveEnergyConverter.cpp
header/HexTile.h
source/SolarPV.cpp
source/WindTurbine.cpp
source/TidalTurbine.cpp
source/HexTile.cpp
header/HexMap.h
header/Game.h
source/HexMap.cpp
source/Game.cpp
source/main.cpp

Functions

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

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

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

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

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

  The base colour of a plains tile.
- const sf::Color RESOURCE_CHIP_GREY (175, 175, 175, 250)

  The base colour of the resource chip (backing).
- const sf::Color MENU_FRAME_GREY (185, 187, 182)

  The base colour of the context menu frame.
- const sf::Color MONOCHROME_SCREEN_BACKGROUND (40, 40, 40)

  The base colour of old monochrome screens.
- const sf::Color VISUAL_SCREEN_FRAME_GREY (151, 151, 143)

  The base colour of the framing of the visual screen.
- const sf::Color MONOCHROME_TEXT_GREEN (0, 255, 102)

  The base colour of old monochrome text (green).
- const sf::Color MONOCHROME_TEXT_AMBER (255, 176, 0)

  The base colour of old monochrome text (amber).
- const sf::Color MONOCHROME_TEXT_RED (255, 44, 0)

  The base colour of old monochrome text (red).

Variables

- const double FLOAT_TOLERANCE = 1e-6

  Tolerance for floating point equality tests.
- const unsigned long long int SECONDS_PER_YEAR = 31537970
- const unsigned long long int SECONDS_PER_MONTH = 2628164
- const int FRAMES_PER_SECOND = 60

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

  Target seconds per frame (just reciprocal of target frames per second).
- const int GAME_WIDTH = 1200

  Width of the game space.
- const int GAME_HEIGHT = 800

  Height of the game space.
- const std::vector< double > TILE_TYPE_CUMULATIVE_PROBABILITIES

  Cumulative probabilities for each tile type (to support procedural generation).
- const std::vector< double > TILE_RESOURCE_CUMULATIVE_PROBABILITIES

  Cumulative probabilities for each tile resource (to support procedural generation).
- const std::string TILE_SELECTED_CHANNEL = "TILE SELECTED CHANNEL"

  A message channel for tile selection messages.
- const std::string NO_TILE_SELECTED_CHANNEL = "NO TILE SELECTED CHANNEL"

  A message channel for no tile selected messages.
- const std::string TILE_STATE_CHANNEL = "TILE STATE CHANNEL"

A message channel for tile state messages.

- const std::string HEX_MAP_CHANNEL = "HEX MAP CHANNEL"

  A message channel for hex map messages.
- const std::string SETTLEMENT_CHANNEL = "SETTLEMENT CHANNEL"

  A message channel for the settlement.
- const int CLEAR_FOREST_COST = 40

  The cost of clearing a forest tile.
- const int CLEAR_MOUNTAINS_COST = 250

  The cost of clearing a mountains tile.
- const int CLEAR_PLAINS_COST = 20

  The cost of clearing a plains tile.
- const int DIESEL_GENERATOR_BUILD_COST = 100

  The cost of building (or ugrading) a diesel generator in 100 kW increments.
- const int WIND_TURBINE_BUILD_COST = 400

  The cost of building (or upgrading) a wind turbine in 100 kW increments.
- const double WIND_TURBINE_WATER_BUILD_MULTIPLIER = 1.25

  The additional cost of building on water.
- const int SOLAR_PV_BUILD_COST = 300

  The cost of building (or upgrading) a solar PV array in 100 kW increments.
- const double SOLAR_PV_WATER_BUILD_MULTIPLIER = 1.5

  The additional cost of building on water.
- const int TIDAL_TURBINE_BUILD_COST = 600

  The cost of building (or upgrading) a tidal turbine in 100 kW increments.
- const int WAVE_ENERGY_CONVERTER_BUILD_COST = 800

  The cost of building (or upgrading) a wave energy converter in 100 kW increments.
- const int ENERGY_STORAGE_SYSTEM_BUILD_COST = 160

  The cost of adding energy storage in 200 kWh increments.
- const int SCRAP_COST = 50

  The cost of scrapping a tile improvement (other than settlement).
- const int MAX_UPGRADE_LEVELS = 5

  The maximum upgrade level of any tile improvement.
- const int MAX_STORAGE_LEVELS = 5

  The maximum storage level of any tile improvement.
- const int STARTING_CREDITS = 999999
- const double CREDITS_PER_MWH_SERVED = 1

  The number of credits (x1000) earned.
- const int EMISSIONS_LIFETIME_LIMIT_TONNES = 1500

  The CO2-equivalent mass of emissions that would result from burning 1,000,000 L of diesel fuel.
- const int RESOURCE_ASSESSMENT_COST = 20

  The cost of doing a resource assessment.
- const int BUILD_SETTLEMENT_COST = 250

  The cost of building a settlement.
- const int STARTING_POPULATION = 100

  The starting population of a settlement.
- const double POPULATION_MONTHLY_GROWTH_RATE = 1.005

  The monthly population growth rate.
- const double LITRES_DIESEL_PER_MWH_PRODUCTION = 373.175

  The litres of diesel consumed in producing 1 MWh (assumes higher heating value and constant thermal efficiency of 0.25).
- const double COST_PER_LITRE_DIESEL = 1.70

  The cost of a litre of diesel.

- const double KG_CO2E_PER_LITRE_DIESEL = 3.1596

    The CO2-equivalent mass of emissions that result from burning one litre of diesel fuel.
- const double DIESEL_OP_MAINT_COST_PER_MWH_PRODUCTION = 50

    The operation and maintenace cost of running a diesel generator (assumed 0.05 credits per kWh produced).
- const double SOLAR_OP_MAINT_COST_PER_MWH_PRODUCTION = 10

    The operation and maintenance cost of running a solar PV array (assumed 0.01 credits per kWh produced).
- const double TIDAL_OP_MAINT_COST_PER_MWH_PRODUCTION = 50

    The operation and maintenance cost of running a tidal turbine (assumed 0.05 credits per kWh produced).
- const double WAVE_OP_MAINT_COST_PER_MWH_PRODUCTION = 50

    The operation and maintenance cost of running a wave energy converter (assumed 0.05 credits per kWh produced).
- const double WIND_OP_MAINT_COST_PER_MWH_PRODUCTION = 50

    The operation and maintenance cost of running a wind turbine (assumed 0.05 credits per kWh produced).
- const std::vector< double > MEAN_DAILY_DEMAND_RATIOS

    The mean daily demand ratio for each month, where demand ratio is demand [MWh] divided by maximum daily demand [MWh]. Maximum daily demand is simply (24)(max load [kW]) / 1000.
- const std::vector< double > STDEV_DAILY_DEMAND_RATIOS

    The standard deviation in daily demand ratio for each month, where demand ratio is demand [MWh] divided by maximum daily demand [MWh]. Maximum daily demand is simply (24)(max load [kW]) / 1000.
- const double MAXIMUM_DAILY_DEMAND_PER_CAPITA = 0.0475

    The maximum daily demand [MWh] (at any point in the year) per capita.
- const std::vector< double > MEAN_DAILY_SOLAR_CAPACITY_FACTORS

    The mean daily solar capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.
- const std::vector< double > STDEV_DAILY_SOLAR_CAPACITY_FACTORS

    The standard deviation in daily solar capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.
- const double DAILY_TIDAL_CAPACITY_FACTOR = 0.2175

    The daily tidal capacity factor, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000. The tides are not a random process, and are not very sensitive to season.
- const std::vector< double > MEAN_DAILY_WAVE_CAPACITY_FACTORS

    The mean daily wave capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.
- const std::vector< double > STDEV_DAILY_WAVE_CAPACITY_FACTORS

    The standard deviation in daily wave capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.
- const std::vector< double > MEAN_DAILY_WIND_CAPACITY_FACTORS

    The mean daily wind capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.
- const std::vector< double > STDEV_DAILY_WIND_CAPACITY_FACTORS

    The standard deviation in daily wind capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.
- const std::string GAME_CHANNEL = "GAME CHANNEL"

    A message channel for game messages.
- const std::string GAME_STATE_CHANNEL = "GAME STATE CHANNEL"

    A message channel for game state messages.

5.5.1 Detailed Description

Header file for various constants.

5.5.2 Function Documentation

5.5.2.1 FOREST_GREEN()

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

The base colour of a forest tile.

5.5.2.2 LAKE_BLUE()

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

The base colour of a lake (water) tile.

5.5.2.3 MENU_FRAME_GREY()

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

The base colour of the context menu frame.

5.5.2.4 MONOCHROME_SCREEN_BACKGROUND()

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

The base colour of old monochrome screens.

5.5.2.5 MONOCHROME_TEXT_AMBER()

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

The base colour of old monochrome text (amber).

5.5.2.6 MONOCHROME_TEXT_GREEN()

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

The base colour of old monochrome text (green).

5.5.2.7 MONOCHROME_TEXT_RED()

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

The base colour of old monochrome text (red).

5.5.2.8 MOUNTAINS_GREY()

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

The base colour of a mountains tile.

5.5.2.9 OCEAN_BLUE()

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

The base colour of an ocean (water) tile.

5.5.2.10 PLAINS_YELLOW()

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

The base colour of a plains tile.

5.5.2.11 RESOURCE_CHIP_GREY()

```
const sf::Color RESOURCE_CHIP_GREY (
            175 ,
            175 ,
            175 ,
            250  )
```

The base colour of the resource chip (backing).

5.5.2.12 VISUAL_SCREEN_FRAME_GREY()

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

The base colour of the framing of the visual screen.

5.5.3 Variable Documentation

5.5.3.1 BUILD_SETTLEMENT_COST

```
const int BUILD_SETTLEMENT_COST = 250
```

The cost of building a settlement.

5.5.3.2 CLEAR_FOREST_COST

```
const int CLEAR_FOREST_COST = 40
```

The cost of clearing a forest tile.

5.5.3.3 CLEAR_MOUNTAINS_COST

```
const int CLEAR_MOUNTAINS_COST = 250
```

The cost of clearing a mountains tile.

5.5.3.4 CLEAR_PLAINS_COST

```
const int CLEAR_PLAINS_COST = 20
```

The cost of clearing a plains tile.

5.5.3.5 COST_PER_LITRE_DIESEL

```
const double COST_PER_LITRE_DIESEL = 1.70
```

The cost of a litre of diesel.

5.5.3.6 CREDITS_PER_MWH_SERVED

```
const double CREDITS_PER_MWH_SERVED = 1
```

The number of credits (x1000) earned.

5.5.3.7 DAILY_TIDAL_CAPACITY_FACTOR

```
const double DAILY_TIDAL_CAPACITY_FACTOR = 0.2175
```

The daily tidal capacity factor, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000. The tides are not a random process, and are not very sensitive to season.

5.5.3.8 DIESEL_GENERATOR_BUILD_COST

```
const int DIESEL_GENERATOR_BUILD_COST = 100
```

The cost of building (or ugrading) a diesel generator in 100 kW increments.

5.5.3.9 DIESEL_OP_MAINT_COST_PER_MWH_PRODUCTION

```
const double DIESEL_OP_MAINT_COST_PER_MWH_PRODUCTION = 50
```

The operation and maintenace cost of running a diesel generator (assumed 0.05 credits per kWh produced).

5.5.3.10 EMISSIONS_LIFETIME_LIMIT_TONNES

```
const int EMISSIONS_LIFETIME_LIMIT_TONNES = 1500
```

The CO2-equivalent mass of emissions that would result from burning 1,000,000 L of diesel fuel.

5.5.3.11 ENERGY_STORAGE_SYSTEM_BUILD_COST

```
const int ENERGY_STORAGE_SYSTEM_BUILD_COST = 160
```

The cost of adding energy storage in 200 kWh increments.

5.5.3.12 FLOAT_TOLERANCE

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

Tolerance for floating point equality tests.

5.5.3.13 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

5.5.3.14 GAME_CHANNEL

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

A message channel for game messages.

5.5.3.15 GAME_HEIGHT

```
const int GAME_HEIGHT = 800
```

Height of the game space.

5.5.3.16 GAME_STATE_CHANNEL

```
const std::string GAME_STATE_CHANNEL = "GAME STATE CHANNEL"
```

A message channel for game state messages.

5.5.3.17 GAME_WIDTH

```
const int GAME_WIDTH = 1200
```

Width of the game space.

5.5.3.18 HEX_MAP_CHANNEL

```
const std::string HEX_MAP_CHANNEL = "HEX MAP CHANNEL"
```

A message channel for hex map messages.

5.5.3.19 KG_CO2E_PER_LITRE_DIESEL

```
const double KG_CO2E_PER_LITRE_DIESEL = 3.1596
```

The CO2-equivalent mass of emissions that result from burning one litre of diesel fuel.

5.5.3.20 LITRES_DIESEL_PER_MWH_PRODUCTION

```
const double LITRES_DIESEL_PER_MWH_PRODUCTION = 373.175
```

The litres of diesel consumed in producing 1 MWh (assumes higher heating value and constant thermal efficiency of 0.25).

5.5.3.21 MAX_STORAGE_LEVELS

```
const int MAX_STORAGE_LEVELS = 5
```

The maximum storage level of any tile improvement.

5.5.3.22 MAX_UPGRADE_LEVELS

```
const int MAX_UPGRADE_LEVELS = 5
```

The maximum upgrade level of any tile improvement.

5.5.3.23 MAXIMUM_DAILY_DEMAND_PER_CAPITA

```
const double MAXIMUM_DAILY_DEMAND_PER_CAPITA = 0.0475
```

The maximum daily demand [MWh] (at any point in the year) per capita.

5.5.3.24 MEAN_DAILY_DEMAND_RATIOS

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

Initial value:

```
= {
    0.702, 0.704, 0.652,
    0.546, 0.445, 0.362,
    0.261, 0.261, 0.379,
    0.518, 0.622, 0.716
}
```

The mean daily demand ratio for each month, where demand ratio is demand [MWh] divided by maximum daily demand [MWh]. Maximum daily demand is simply (24)(max load [kW]) / 1000.

5.5.3.25 MEAN_DAILY_SOLAR_CAPACITY_FACTORS

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

Initial value:

```
= {
    0.022, 0.046, 0.088,
    0.138, 0.171, 0.175,
    0.164, 0.139, 0.104,
    0.061, 0.030, 0.016
}
```

The mean daily solar capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.

5.5.3.26 MEAN_DAILY_WAVE_CAPACITY_FACTORS

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

Initial value:

```
= {
    0.742, 0.694, 0.618,
    0.467, 0.366, 0.292,
    0.280, 0.293, 0.374,
    0.424, 0.662, 0.600
}
```

The mean daily wave capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.

5.5.3.27 MEAN_DAILY_WIND_CAPACITY_FACTORS

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

Initial value:

```
= {
    0.591, 0.594, 0.627,
    0.629, 0.579, 0.537,
    0.442, 0.507, 0.587,
    0.618, 0.611, 0.580
}
```

The mean daily wind capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.

5.5.3.28 NO_TILE_SELECTED_CHANNEL

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

A message channel for no tile selected messages.

5.5.3.29 POPULATION_MONTHLY_GROWTH_RATE

```
const double POPULATION_MONTHLY_GROWTH_RATE = 1.005
```

The monthly population growth rate.

5.5.3.30 RESOURCE_ASSESSMENT_COST

```
const int RESOURCE_ASSESSMENT_COST = 20
```

The cost of doing a resource assessment.

5.5.3.31 SCRAP_COST

```
const int SCRAP_COST = 50
```

The cost of scrapping a tile improvement (other than settlement).

5.5.3.32 SECONDS_PER_FRAME

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

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

5.5.3.33 SECONDS_PER_MONTH

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

5.5.3.34 SECONDS_PER_YEAR

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

5.5.3.35 SETTLEMENT_CHANNEL

```
const std::string SETTLEMENT_CHANNEL = "SETTLEMENT CHANNEL"
```

A message channel for the settlement.

5.5.3.36 SOLAR_OP_MAINT_COST_PER_MWH_PRODUCTION

```
const double SOLAR_OP_MAINT_COST_PER_MWH_PRODUCTION = 10
```

The operation and maintenance cost of running a solar PV array (assumed 0.01 credits per kWh produced).

5.5.3.37 SOLAR_PV_BUILD_COST

```
const int SOLAR_PV_BUILD_COST = 300
```

The cost of building (or upgrading) a solar PV array in 100 kW increments.

5.5.3.38 SOLAR_PV_WATER_BUILD_MULTIPLIER

```
const double SOLAR_PV_WATER_BUILD_MULTIPLIER = 1.5
```

The additional cost of building on water.

5.5.3.39 STARTING_CREDITS

```
const int STARTING_CREDITS = 999999
```

5.5.3.40 STARTING_POPULATION

```
const int STARTING_POPULATION = 100
```

The starting population of a settlement.

5.5.3.41 STDEV_DAILY_DEMAND_RATIOS

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

Initial value:

```
= {
    0.069, 0.074, 0.072,
    0.072, 0.063, 0.060,
    0.012, 0.031, 0.040,
    0.049, 0.063, 0.053
}
```

The standard deviation in daily demand ratio for each month, where demand ratio is demand [MWh] divided by maximum daily demand [MWh]. Maximum daily demand is simply (24)(max load [kW]) / 1000.

5.5.3.42 STDEV_DAILY_SOLAR_CAPACITY_FACTORS

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

Initial value:
```
= {
    0.013, 0.024, 0.043,
    0.049, 0.072, 0.072,
    0.076, 0.065, 0.048,
    0.026, 0.018, 0.009
}
```

The standard deviation in daily solar capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.

5.5.3.43 STDEV_DAILY_WAVE_CAPACITY_FACTORS

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

Initial value:
```
= {
    0.146, 0.135, 0.163,
    0.145, 0.158, 0.106,
    0.086, 0.058, 0.145,
    0.171, 0.184, 0.309
}
```

The standard deviation in daily wave capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.

5.5.3.44 STDEV_DAILY_WIND_CAPACITY_FACTORS

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

Initial value:
```
= {
    0.147, 0.142, 0.198,
    0.154, 0.162, 0.202,
    0.180, 0.217, 0.198,
    0.168, 0.141, 0.168
}
```

The standard deviation in daily wind capacity factors for each month, where capacity factor is daily production [MWh] divided by maximum daily production [MWh]. Maximum daily production is simply (24)(power capacity [kW]) / 1000.

5.5.3.45 TIDAL_OP_MAINT_COST_PER_MWH_PRODUCTION

```
const double TIDAL_OP_MAINT_COST_PER_MWH_PRODUCTION = 50
```

The operation and maintenance cost of running a tidal turbine (assumed 0.05 credits per kWh produced).

5.5.3.46 TIDAL_TURBINE_BUILD_COST

```
const int TIDAL_TURBINE_BUILD_COST = 600
```

The cost of building (or upgrading) a tidal turbine in 100 kW increments.

5.5.3.47 TILE_RESOURCE_CUMULATIVE_PROBABILITIES

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

Initial value:

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

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

5.5.3.48 TILE_SELECTED_CHANNEL

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

A message channel for tile selection messages.

5.5.3.49 TILE_STATE_CHANNEL

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

A message channel for tile state messages.

5.5.3.50 TILE_TYPE_CUMULATIVE_PROBABILITIES

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

Initial value:

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

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

5.5.3.51 WAVE_ENERGY_CONVERTER_BUILD_COST

```
const int WAVE_ENERGY_CONVERTER_BUILD_COST = 800
```

The cost of building (or upgrading) a wave energy converter in 100 kW increments.

5.5.3.52 WAVE_OP_MAINT_COST_PER_MWH_PRODUCTION

```
const double WAVE_OP_MAINT_COST_PER_MWH_PRODUCTION = 50
```

The operation and maintenance cost of running a wave energy converter (assumed 0.05 credits per kWh produced).

5.5.3.53 WIND_OP_MAINT_COST_PER_MWH_PRODUCTION

```
const double WIND_OP_MAINT_COST_PER_MWH_PRODUCTION = 50
```

The operation and maintenance cost of running a wind turbine (assumed 0.05 credits per kWh produced).

5.5.3.54 WIND_TURBINE_BUILD_COST

```
const int WIND_TURBINE_BUILD_COST = 400
```

The cost of building (or upgrading) a wind turbine in 100 kW increments.

5.5.3.55 WIND_TURBINE_WATER_BUILD_MULTIPLIER

```
const double WIND_TURBINE_WATER_BUILD_MULTIPLIER = 1.25
```

The additional cost of building on water.

5.6 header/ESC_core/doxygen_cite.h File Reference

Header file which simply cites the doxygen tool.

5.6.1 Detailed Description

Header file which simply cites the doxygen tool.

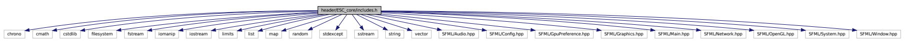
Ref: van Heesch. [2023]

5.7 header/ESC_core/includes.h File Reference

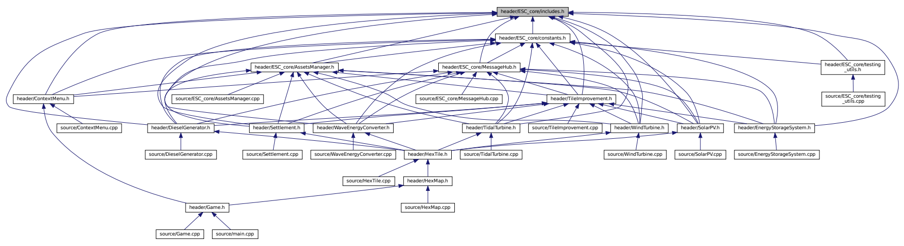
Header file for various includes.

```
#include <chrono>
#include <cmath>
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <limits>
#include <list>
#include <map>
#include <random>
#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:


header/ESC_core/includes.h
header/ESC_core/constants.h
header/ESC_core/AssetsManager.h
header/ESC_core/MessageHub.h
header/ESC_core/testing_utils.h
header/ContextMenu.h
source/ESC_core/AssetsManager.cpp
source/ESC_core/MessageHub.cpp
header/TileImprovement.h
source/ESC_core/testing_utils.cpp
source/ContextMenu.cpp
header/DieselGenerator.h
header/Settlement.h
header/WaveEnergyConverter.h
header/TidalTurbine.h
source/TileImprovement.cpp
header/WindTurbine.h
header/SolarPV.h
header/EnergyStorageSystem.h
source/DieselGenerator.cpp
source/Settlement.cpp
source/WaveEnergyConverter.cpp
header/HexTile.h
source/TidalTurbine.cpp
source/WindTurbine.cpp
source/SolarPV.cpp
source/EnergyStorageSystem.cpp
source/HexTile.cpp
header/HexMap.h
header/Game.h
source/HexMap.cpp
source/Game.cpp
source/main.cpp

5.7.1 Detailed Description

Header file for various includes.

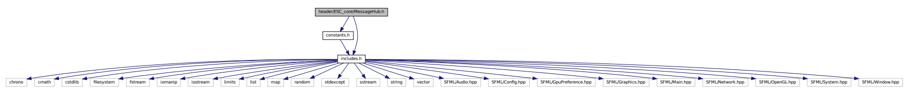
Ref: Gomila [2023]

5.8 header/ESC_core/MessageHub.h File Reference

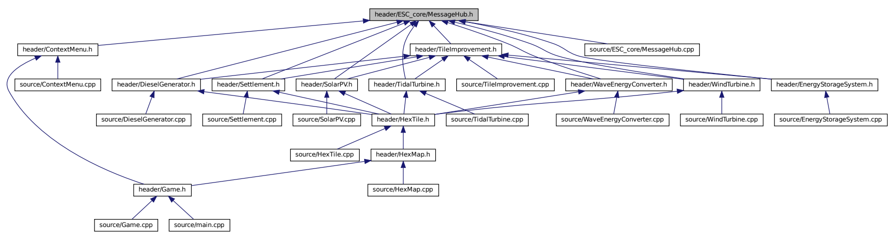
Header file for the MessageHub class.

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

Include dependency graph for MessageHub.h:



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



Classes

- struct Message

  A structure which defines a standard message format.
- class MessageHub

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

5.8.1 Detailed Description

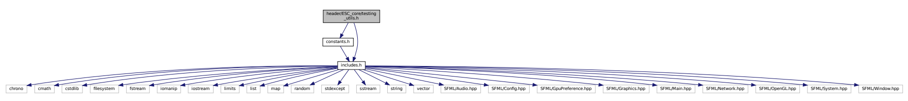
Header file for the MessageHub class.

5.9 header/ESC_core/testing_utils.h File Reference

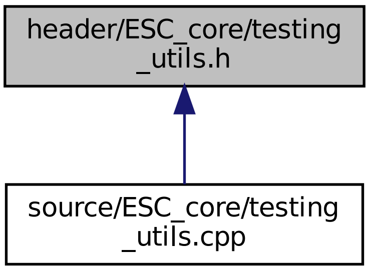
Header file for various testing utilities.

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

Include dependency graph for testing_utils.h:



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


header/ESC_core/testing_utils.h
source/ESC_core/testing_utils.cpp

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.

5.9.1 Detailed Description

Header file for various testing utilities.

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

5.9.2 Function Documentation

5.9.2.1 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__").

```
{
    std::string error_str = "\n ERROR  failed to throw expected error prior to line ";
    error_str += std::to_string(line);
    error_str += " of ";
    error_str += file;

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* expectedErrorNotDetected() */
```

5.9.2.2 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.

```
{
    std::cout « "\x1B[33m" « input_str « "\033[0m";
    return;
}   /* printGold() */
```

5.9.2.3 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.

```
{
    std::cout « "\x1B[32m" « input_str « "\033[0m";
    return;
}   /* printGreen() */
```

5.9.2.4 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.

```
{
    std::cout « "\x1B[31m" « input_str « "\033[0m";
    return;
}   /* printRed() */
```

5.9.2.5 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__").

```
{
    if (fabs(x - y) <= FLOAT_TOLERANCE) {
        return;
    }

    std::string error_str = "ERROR: testFloatEquals():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " and ";
    error_str += std::to_string(y);
    error_str += " are not equal to within +/- ";
    error_str += std::to_string(FLOAT_TOLERANCE);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testFloatEquals() */
```

5.9.2.6 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__").

```
{
    if (x > y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThan() */
```

5.9.2.7 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.

Parameters

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__").

```
{
    if (x >= y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThanOrEqualTo() */
```

5.9.2.8 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__").

```
{
    if (x < y) {
        return;
    }

    std::string error_str = "ERROR: testLessThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThan() */
```

5.9.2.9 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__").

```
{
    if (x <= y) {
        return;
    }

    std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThanOrEqualTo() */
```

5.9.2.10 testTruth()

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

Tests if the given statement is true.

Parameters

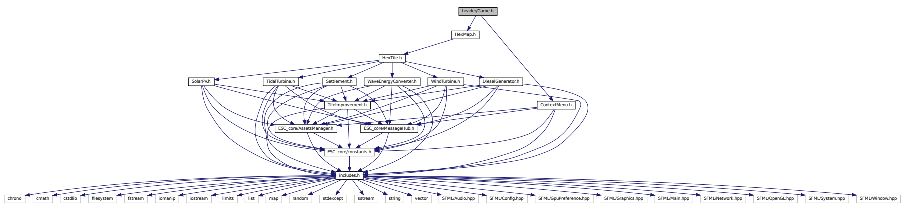
statement	The statement whose truth is to be tested ("1 == 0", for example).
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (statement) {
        return;
    }

    std::string error_str = "ERROR: testTruth():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += "Given statement is not true";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testTruth() */
```

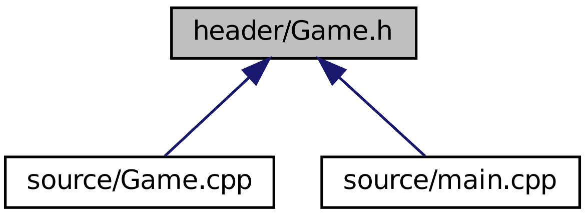
5.10 header/Game.h File Reference

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

Include dependency graph for Game.h:



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



Classes

- class Game

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

Enumerations

- enum GamePhase {
  BUILD_SETTLEMENT , SYSTEM_MANAGEMENT , LOSS_EMISSIONS , LOSS_DEMAND ,
  LOSS_CREDITS , VICTORY , N_GAME_PHASES }

  An enumeration of the various game phases.

5.10.1 Enumeration Type Documentation

5.10.1.1 GamePhase

```
enum GamePhase
```

An enumeration of the various game phases.

Enumerator

BUILD_SETTLEMENT	The settlement building phase.
SYSTEM_MANAGEMENT	The system management phase (main phase of play).
LOSS_EMISSIONS	A loss due to excessive emissions.
LOSS_DEMAND	A loss due to failing to meet the demand.
LOSS_CREDITS	A loss due to running out of credits.
VICTORY	A victory (12 consecutive months of zero emissions).
N_GAME_PHASES	A simple hack to get the number of elements in GamePhase.

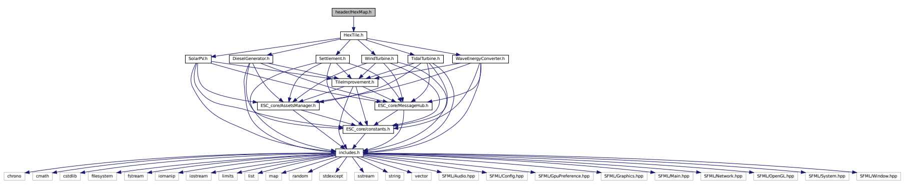
```
                {
    BUILD_SETTLEMENT,
    SYSTEM_MANAGEMENT,
    LOSS_EMISSIONS,
    LOSS_DEMAND,
    LOSS_CREDITS,
    VICTORY,
    N_GAME_PHASES
};  /* GamePhase */
```

5.11 header/HexMap.h File Reference

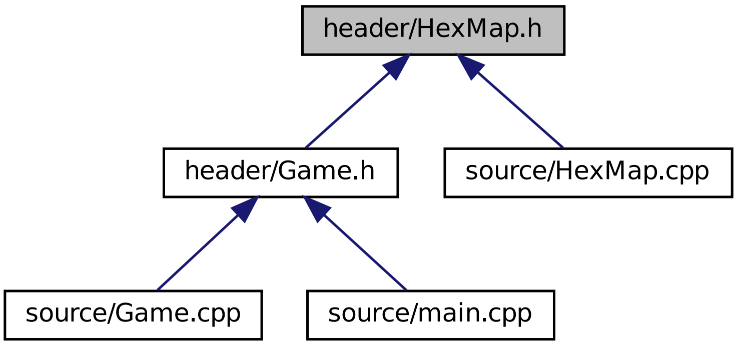
Header file for the HexMap class.

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

Include dependency graph for HexMap.h:



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



Classes

- class HexMap

  A class which defines a hex map of hex tiles.

5.11.1 Detailed Description

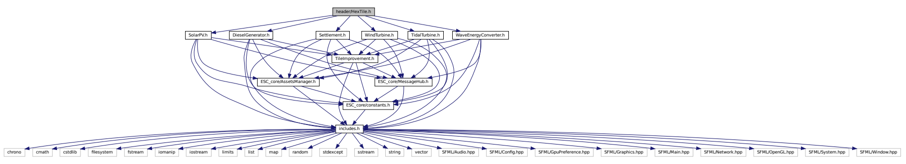
Header file for the HexMap class.

5.12 header/HexTile.h File Reference

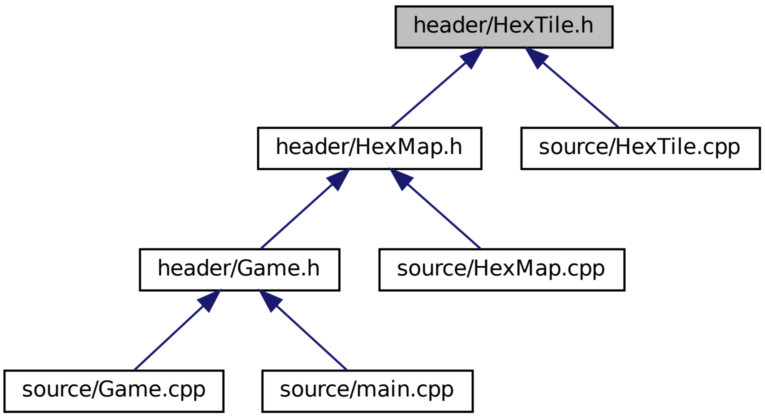
Header file for the Game class.

```
#include "DieselGenerator.h"
#include "Settlement.h"
#include "SolarPV.h"
#include "TidalTurbine.h"
#include "WaveEnergyConverter.h"
#include "WindTurbine.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 {
  NONE_TYPE , 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.

5.12.1 Detailed Description

Header file for the Game class.

Header file for the HexTile class.

5.12.2 Enumeration Type Documentation

5.12.2.1 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.

```
               {
    POOR,
    BELOW_AVERAGE,
    AVERAGE,
    ABOVE_AVERAGE,
    GOOD,
    N_TILE_RESOURCES
};  /* TileResource */
```

5.12.2.2 TileType

```
enum TileType
```

An enumeration of the different tile types.

Enumerator

NONE_TYPE	A dummy tile (for initialization).
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.

```
           {
    NONE_TYPE,
    FOREST,
    LAKE,
    MOUNTAINS,
    OCEAN,
    PLAINS,
    N_TILE_TYPES
}; /* TileType */
```

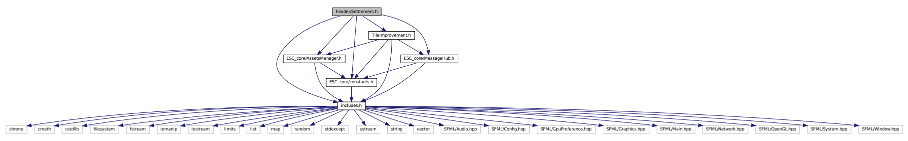
5.13 header/Settlement.h File Reference

Header file for the Settlement class.

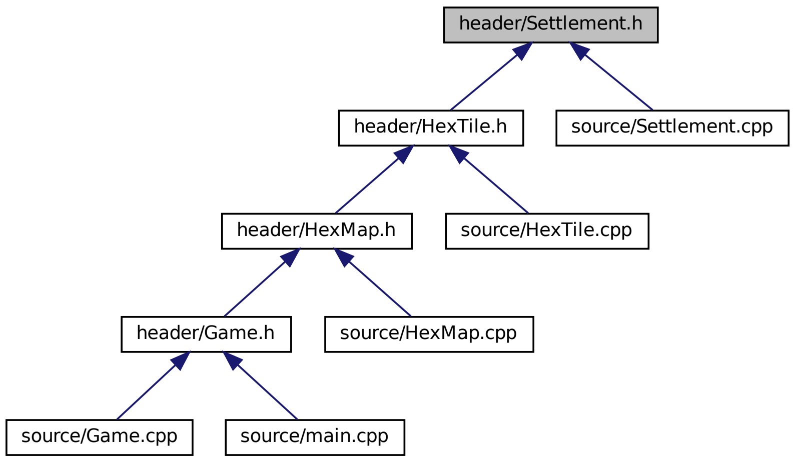
```
#include "ESC_core/constants.h"
#include "ESC_core/includes.h"
#include "ESC_core/AssetsManager.h"
#include "ESC_core/MessageHub.h"
```

```
#include "TileImprovement.h"
```

Include dependency graph for Settlement.h:



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



Classes

- class Settlement

    A settlement class (child class of TileImprovement).

5.13.1 Detailed Description

Header file for the Settlement class.

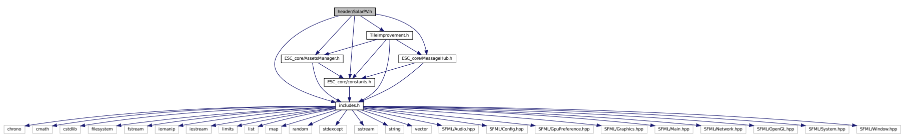
5.14 header/SolarPV.h File Reference

Header file for the SolarPV class.

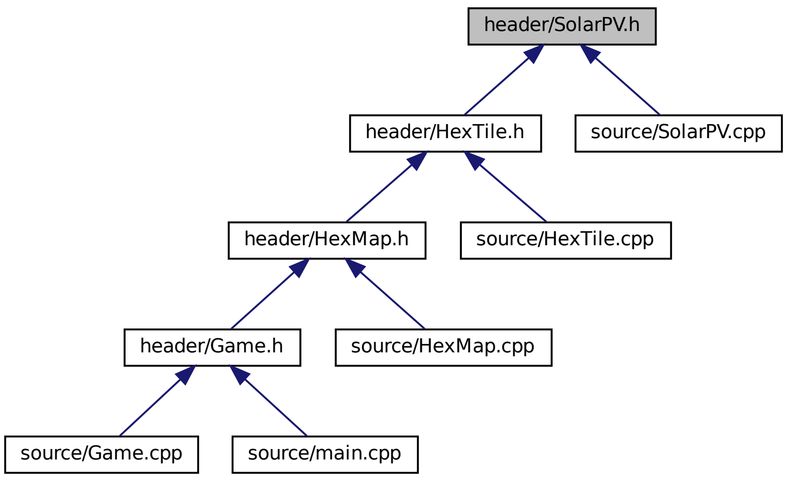
```
#include "ESC_core/constants.h"
#include "ESC_core/includes.h"
#include "ESC_core/AssetsManager.h"
#include "ESC_core/MessageHub.h"
```

```
#include "TileImprovement.h"
```

Include dependency graph for SolarPV.h:



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



Classes

- class SolarPV

  A settlement class (child class of TileImprovement).

5.14.1 Detailed Description

Header file for the SolarPV class.

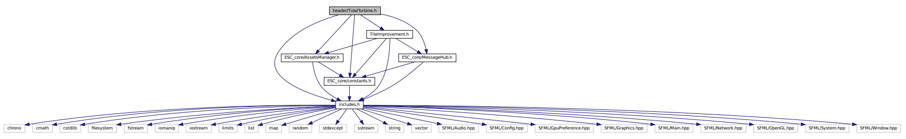
5.15 header/TidalTurbine.h File Reference

Header file for the TidalTurbine class.

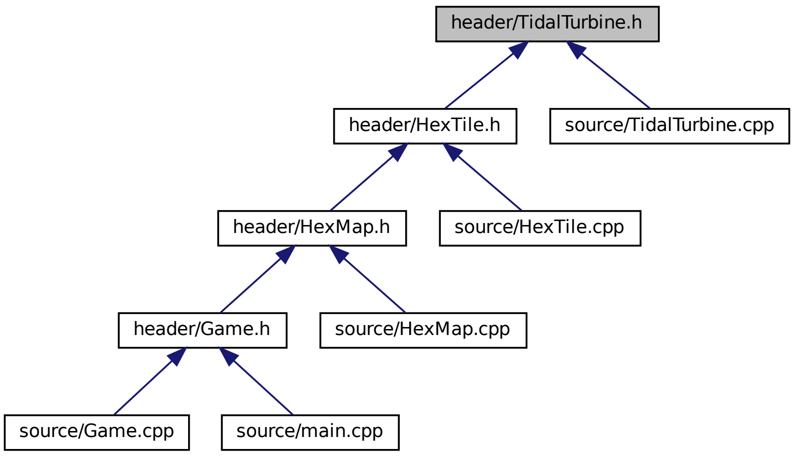
```
#include "ESC_core/constants.h"
#include "ESC_core/includes.h"
#include "ESC_core/AssetsManager.h"
#include "ESC_core/MessageHub.h"
```

```
#include "TileImprovement.h"
```

Include dependency graph for TidalTurbine.h:



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



Classes

- class TidalTurbine

  A settlement class (child class of TileImprovement).

5.15.1 Detailed Description

Header file for the TidalTurbine class.

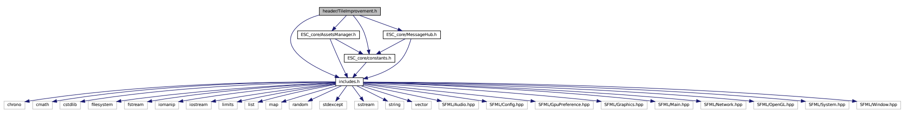
5.16 header/TileImprovement.h File Reference

Header file for the TileImprovement class.

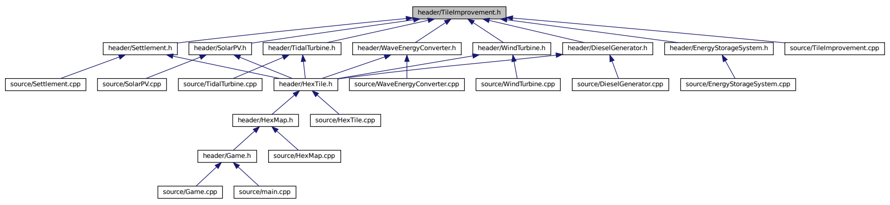
```
#include "ESC_core/constants.h"
#include "ESC_core/includes.h"
#include "ESC_core/AssetsManager.h"
```

```
#include "ESC_core/MessageHub.h"
```

Include dependency graph for TileImprovement.h:



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



Classes

- class TileImprovement

  A base class for the tile improvement hierarchy.

Enumerations

- enum TileImprovementType {
  SETTLEMENT , DIESEL_GENERATOR , SOLAR_PV , WIND_TURBINE ,
  TIDAL_TURBINE , WAVE_ENERGY_CONVERTER , ENERGY_STORAGE_SYSTEM , N_TILE_IMPROVEMENT_TYPES
  }

  An enumeration of the different tile improvement types.

5.16.1 Detailed Description

Header file for the TileImprovement class.

5.16.2 Enumeration Type Documentation

5.16.2.1 TileImprovementType

```
enum TileImprovementType
```

An enumeration of the different tile improvement types.

Enumerator

SETTLEMENT	A settlement.
DIESEL_GENERATOR	A diesel generator.
SOLAR_PV	A solar PV array.
WIND_TURBINE	A wind turbine.
TIDAL_TURBINE	A tidal turbine.
WAVE_ENERGY_CONVERTER	A wave energy converter.
ENERGY_STORAGE_SYSTEM	An energy storage system.
N_TILE_IMPROVEMENT_TYPES	A simple hack to get the number of elements in TileImprovementType.

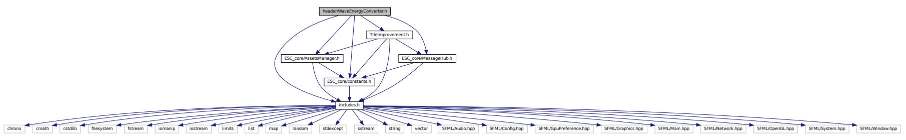
```
                        {
    SETTLEMENT,
    DIESEL_GENERATOR,
    SOLAR_PV,
    WIND_TURBINE,
    TIDAL_TURBINE,
    WAVE_ENERGY_CONVERTER,
    ENERGY_STORAGE_SYSTEM,
    N_TILE_IMPROVEMENT_TYPES
};  /* TileImprovementType */
```

5.17 header/WaveEnergyConverter.h File Reference

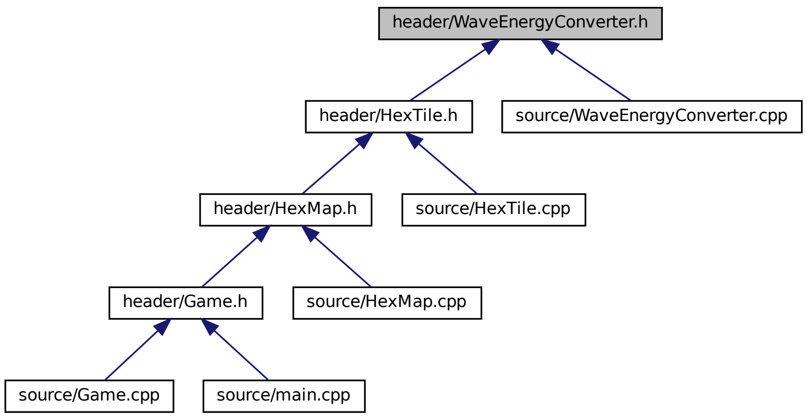
Header file for the WaveEnergyConverter class.

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

Include dependency graph for WaveEnergyConverter.h:



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



Classes

- class WaveEnergyConverter

  A settlement class (child class of TileImprovement).

5.17.1 Detailed Description

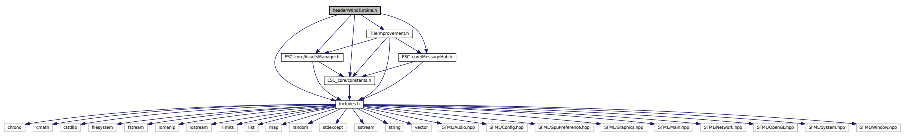
Header file for the WaveEnergyConverter class.

5.18 header/WindTurbine.h File Reference

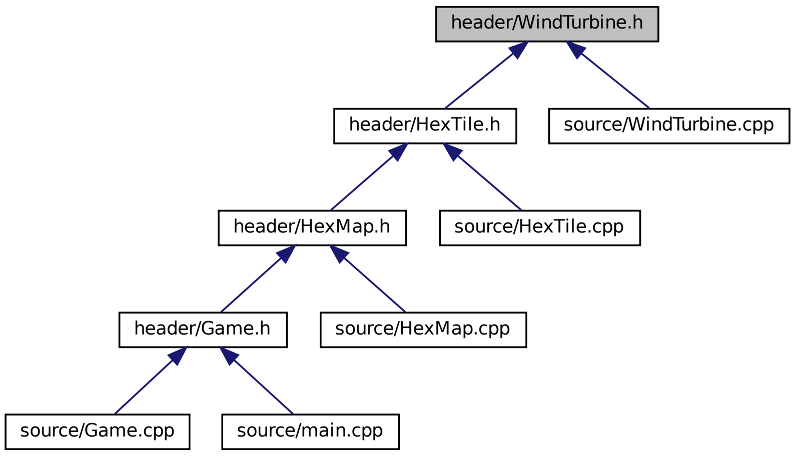
Header file for the WindTurbine class.

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

Include dependency graph for WindTurbine.h:



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



Classes

- class WindTurbine

  A settlement class (child class of TileImprovement).

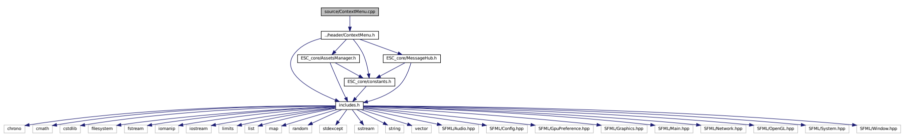
5.18.1 Detailed Description

Header file for the WindTurbine class.

5.19 source/ContextMenu.cpp File Reference

Implementation file for the ContextMenu class.

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



5.19.1 Detailed Description

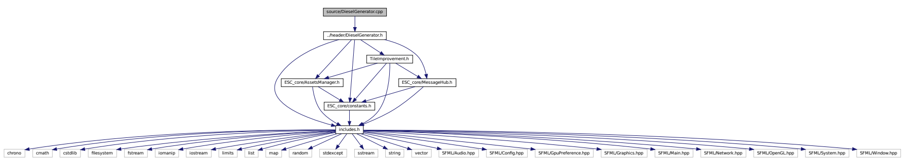
Implementation file for the ContextMenu class.

A class which defines a context menu for the game.

5.20 source/DieselGenerator.cpp File Reference

Implementation file for the DieselGenerator class.

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



5.20.1 Detailed Description

Implementation file for the DieselGenerator class.

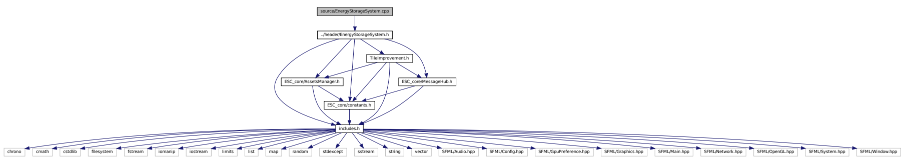
A base class for the tile improvement hierarchy.

5.21 source/EnergyStorageSystem.cpp File Reference

Implementation file for the EnergyStorageSystem class.

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

Include dependency graph for EnergyStorageSystem.cpp:



5.21.1 Detailed Description

Implementation file for the EnergyStorageSystem class.

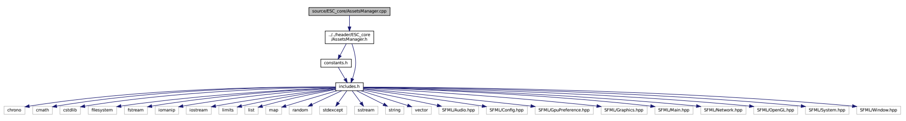
A base class for the tile improvement hierarchy.

5.22 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:



5.22.1 Detailed Description

Implementation file for the AssetsManager class.

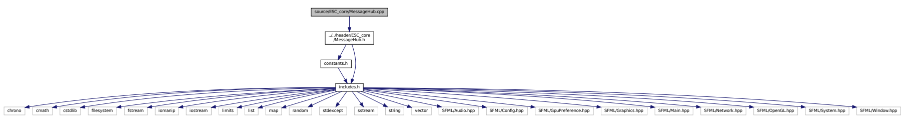
A class which manages visual and sound assets.

5.23 source/ESC_core/MessageHub.cpp File Reference

Implementation file for the MessageHub class.

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

Include dependency graph for MessageHub.cpp:



5.23.1 Detailed Description

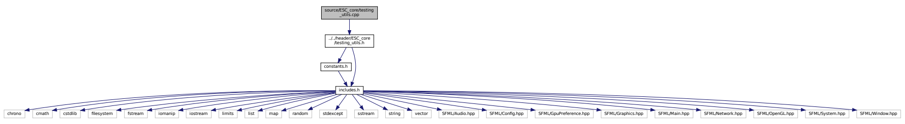
Implementation file for the MessageHub class.

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

5.24 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.

5.24.1 Detailed Description

Implementation file for various testing utilities.

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

5.24.2 Function Documentation

5.24.2.1 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__").

```
{
    std::string error_str = "\n ERROR  failed to throw expected error prior to line ";
    error_str += std::to_string(line);
    error_str += " of ";
    error_str += file;

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* expectedErrorNotDetected() */
```

5.24.2.2 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.

```
{
    std::cout « "\x1B[33m" « input_str « "\033[0m";
    return;
}   /* printGold() */
```

5.24.2.3 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.

```
{
    std::cout « "\x1B[32m" « input_str « "\033[0m";
    return;
}   /* printGreen() */
```

5.24.2.4 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.

```
{
    std::cout « "\x1B[31m" « input_str « "\033[0m";
    return;
}   /* printRed() */
```

5.24.2.5 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__").

```
{
    if (fabs(x - y) <= FLOAT_TOLERANCE) {
        return;
    }

    std::string error_str = "ERROR: testFloatEquals():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " and ";
    error_str += std::to_string(y);
    error_str += " are not equal to within +/- ";
    error_str += std::to_string(FLOAT_TOLERANCE);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testFloatEquals() */
```

5.24.2.6 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__").

```
{
    if (x > y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThan() */
```

5.24.2.7 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__").

```
{
    if (x >= y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThanOrEqualTo() */
```

5.24.2.8 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__").

```
{
    if (x < y) {
        return;
    }

    std::string error_str = "ERROR: testLessThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThan() */
```

5.24.2.9 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__").

```
{
    if (x <= y) {
        return;
    }

    std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThanOrEqualTo() */
```

5.24.2.10 testTruth()

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

Tests if the given statement is true.

Parameters

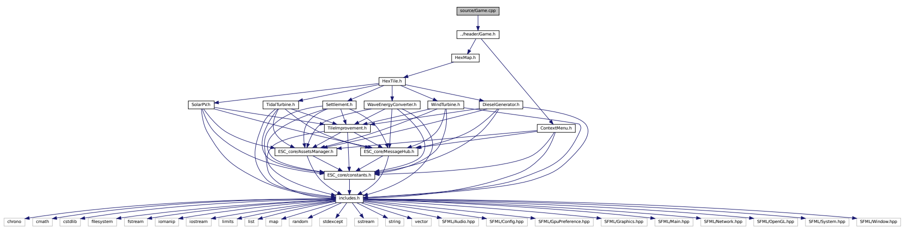
statement	The statement whose truth is to be tested ("1 == 0", for example).
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (statement) {
        return;
    }

    std::string error_str = "ERROR: testTruth():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += "Given statement is not true";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testTruth() */
```

5.25 source/Game.cpp File Reference

Implementation file for the Game class.

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



5.25.1 Detailed Description

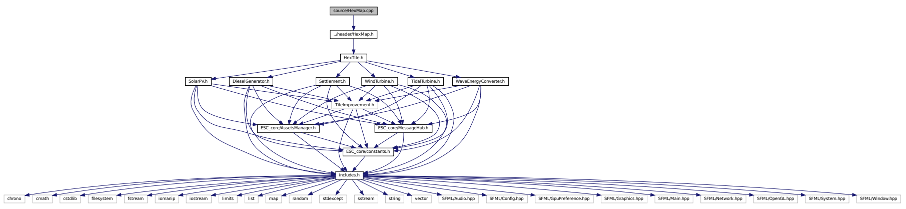
Implementation file for the Game class.

A class which defines a tile of a hex map.

5.26 source/HexMap.cpp File Reference

Implementation file for the HexMap class.

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



5.26.1 Detailed Description

Implementation file for the HexMap class.

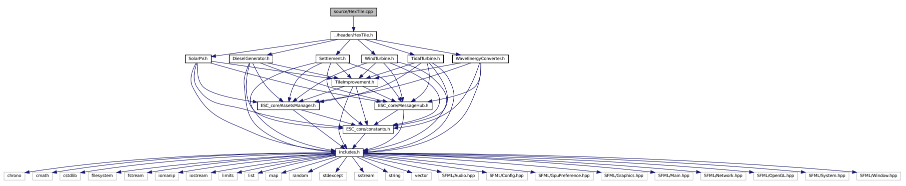
A class which defines a hex map of hex tiles.

5.27 source/HexTile.cpp File Reference

Implementation file for the HexTile class.

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

Include dependency graph for HexTile.cpp:



5.27.1 Detailed Description

Implementation file for the HexTile class.

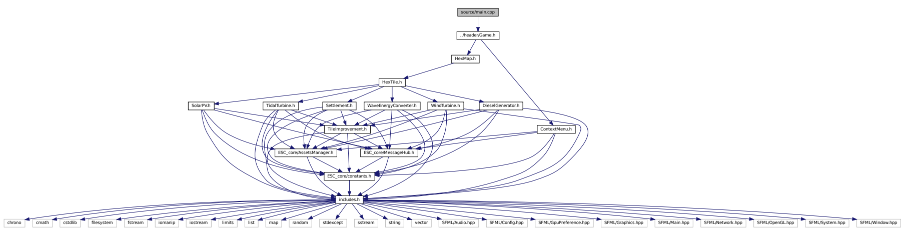
A class which defines a tile of a hex map.

5.28 source/main.cpp File Reference

Implementation file for main() for Road To Zero.

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

Include dependency graph for main.cpp:



Functions

- void loadAssets (AssetsManager ∗assets_manager_ptr)

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

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

5.28.1 Detailed Description

Implementation file for main() for Road To Zero.

5.28.2 Function Documentation

5.28.2.1 constructRenderWindow()

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

Helper function to construct render window.

Returns

Pointer to the render window.

```
{
    sf::RenderWindow* render_window_ptr = new sf::RenderWindow(
        sf::VideoMode(GAME_WIDTH, GAME_HEIGHT),
        "Road To Zero"
    );

    return render_window_ptr;
}   /* constructRenderWindow() */
```

5.28.2.2 loadAssets()

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

Helper function to load game assets.

Parameters

assets_manager_ptr	Pointer to the assets manager.

```
{
    //  1. load font assets
    assets_manager_ptr->loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
    assets_manager_ptr->loadFont("assets/fonts/Glass_TTY_VT220.ttf", "Glass_TTY_VT220");


    //  2. load tile sheets
    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/pine_tree_64x64_1_CC-BY.png",
        "pine_tree_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/wheat_64x64_1_CC-BY.png",
        "wheat_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/mountain_64x64_1_CC-BY.png",
        "mountain_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/water_waves_64x64_1_CC-BY.png",
        "water_waves_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/water_shimmer_64x64_1_CC-BY.png",
        "water_shimmer_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/brick_house_64x64_1_CC-BY.png",
        "brick_house_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/magnifying_glass_64x64_1_CC-BY.png",
        "magnifying_glass_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/exp2_0_CC0.png",
        "tile clear explosion"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/emissions_8x8_1_CC-BY.png",
        "emissions"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/diesel_generator_64x64_2_CC-BY.png",
        "diesel generator"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/solar_PV_64x64_1_CC-BY.png",
        "solar PV array"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/wind_turbine_64x64_2_CC-BY.png",
        "wind turbine"
    );

     assets_manager_ptr->loadTexture(
        "assets/tile_sheets/energy_storage_system_64x64_1_CC-BY.png",
        "energy storage system"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/tidal_turbine_64x64_2_CC-BY.png",
        "tidal turbine"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/wave_energy_converter_64x64_2_CC-BY.png",
        "wave energy converter"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/upgrade_arrow_16x16_1_CC-BY.png",
        "upgrade arrow"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/upgrade_plus_16x16_1_CC-BY.png",
        "upgrade plus"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/energy_storage_16x16_1_CC-BY.png",
        "storage level"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/coin_16x16_1_CC-BY.png",
        "coin"
    );


    //  3. load sounds
    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-magical-coin-win-1936_MixkitFree.ogg",
        "coin ring"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-positive-notification-951_MixkitFree.ogg",
        "positive notification"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-sci-fi-click-900_MixkitFree.ogg",
        "sci-fi click"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-apartment-buzzer-bell-press-932_MixkitFree.ogg",
        "insufficient credits"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-data-scanner-2487_MixkitFree.ogg",
        "resource assessment"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-interface-click-1126_MixkitFree.ogg",
        "console string print"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-video-game-retro-click-237_MixkitFree.ogg",
        "resource overlay toggle on"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-video-game-retro-click-237_REVERSED_MixkitFree.ogg",
        "resource overlay toggle off"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-explosion-with-rocks-debris-1703_MixkitFree.ogg",
        "clear mountains tile"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-arcade-game-explosion-2759_MixkitFree.ogg",
        "clear non-mountains tile"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-electronic-retro-block-hit-2185_MixkitFree.ogg",
        "place improvement"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-video-game-lock-2851_REVERSED_MixkitFree.ogg",
        "build menu open"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-video-game-lock-2851_MixkitFree.ogg",
        "build menu close"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-jump-into-the-water-1180_MixkitFree.ogg",
        "splash"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/505316__nuncaconoci__diesel_CC0.ogg",
        "diesel running"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/33460__pempi__320d_2_CC-BY.ogg",
        "diesel start"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/132724__andy_gardner__wind-turbine-blades_CC-BY.ogg",
        "wind turbine running"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/58416__darren1979__oceanwaves_CC-SAMPLING.ogg",
        "ocean waves"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/369927__mephisto_egmont__water-flowing-in-tubes_CC-BY.ogg",
        "water flow"
    );

    assets_manager_ptr->loadSound(
"assets/audio/samples/647663__jotraing__electric-train-motor-idle-loop-new-generation-rollingstock_CC0.ogg",
        "solar hum"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-epic-futuristic-movie-accent-2913_MixkitFree.ogg",
        "game title screen"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-calm-park-with-people-and-children_MixkitFree.ogg",
        "people and children"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-magical-coin-win-1936_MixkitFree.ogg",
        "upgrade"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-cool-interface-click-tone-2568_MixkitFree.ogg",
        "interface click"
    );

    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-factory-metal-hard-hit-2980_MixkitFree.ogg",
        "breakdown"
    );


    //  4. load tracks
    assets_manager_ptr->loadTrack(
        "assets/audio/tracks/TreeStarMoon_Dobranoc_CC0.ogg",
        "Tree Star Moon - Dobranoc"
    );

    assets_manager_ptr->loadTrack(
        "assets/audio/tracks/TreeStarMoon_Lighthouse_CC0.ogg",
        "Tree Star Moon - Lighthouse"
    );

    assets_manager_ptr->loadTrack(
        "assets/audio/tracks/TreeStarMoon_SkyFarm_CC0.ogg",
        "Tree Star Moon - Sky Farm"
    );

    return;
}   /* loadAssets() */
```

5.28.2.3 main()

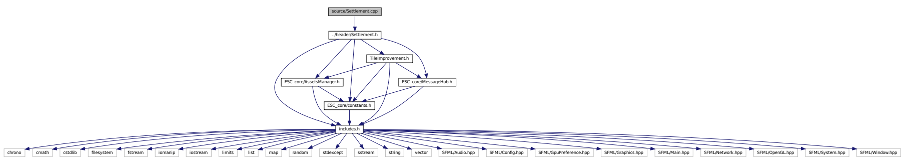
```
int main (
              int argc,
              char ** argv )
```

```
{
    //  1. load assets
    AssetsManager assets_manager;
    loadAssets(&assets_manager);

    //  2. construct render window
    sf::RenderWindow* render_window_ptr = constructRenderWindow();

    //  3. start game loop
    bool quit_game = false;
    assets_manager.playTrack();

    while (not quit_game) {
        Game game(render_window_ptr, &assets_manager);
        quit_game = game.run();
    }

    //  4. clean up
    render_window_ptr->close();
    delete render_window_ptr;

    return 0;
}   /* main() */
```

5.29 source/Settlement.cpp File Reference

Implementation file for the Settlement class.

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



5.29.1 Detailed Description

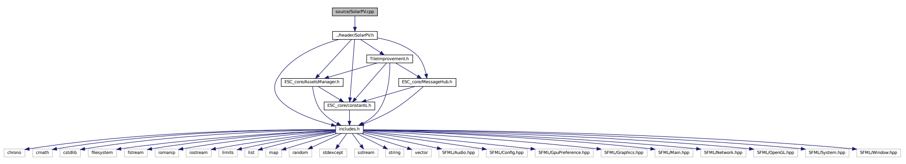
Implementation file for the Settlement class.

A base class for the tile improvement hierarchy.

5.30 source/SolarPV.cpp File Reference

Implementation file for the SolarPV class.

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



5.30.1 Detailed Description

Implementation file for the SolarPV class.

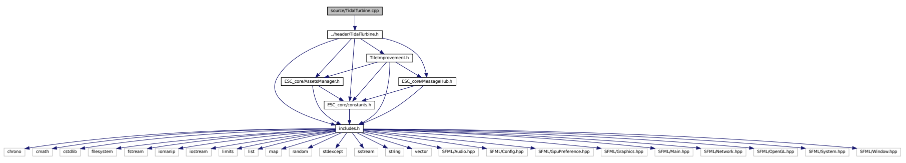
A base class for the tile improvement hierarchy.

5.31 source/TidalTurbine.cpp File Reference

Implementation file for the TidalTurbine class.

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

Include dependency graph for TidalTurbine.cpp:



5.31.1 Detailed Description

Implementation file for the TidalTurbine class.

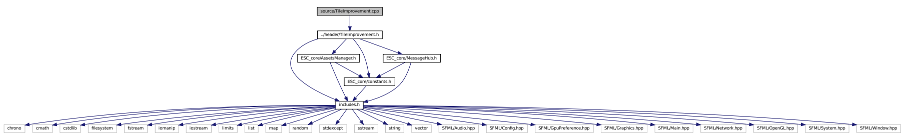
A base class for the tile improvement hierarchy.

5.32 source/TileImprovement.cpp File Reference

Implementation file for the TileImprovement class.

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

Include dependency graph for TileImprovement.cpp:



5.32.1 Detailed Description

Implementation file for the TileImprovement class.

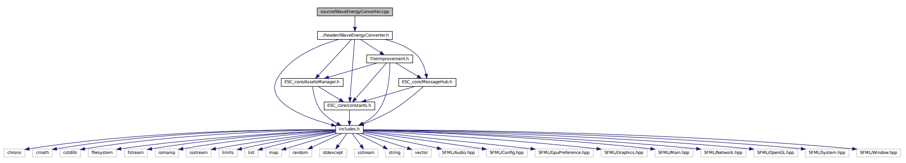
A base class for the tile improvement hierarchy.

5.33 source/WaveEnergyConverter.cpp File Reference

Implementation file for the WaveEnergyConverter class.

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

Include dependency graph for WaveEnergyConverter.cpp:



5.33.1 Detailed Description

Implementation file for the WaveEnergyConverter class.

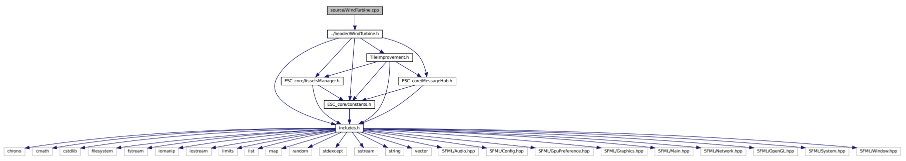
A base class for the tile improvement hierarchy.

5.34 source/WindTurbine.cpp File Reference

Implementation file for the WindTurbine class.

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

Include dependency graph for WindTurbine.cpp:



5.34.1 Detailed Description

Implementation file for the WindTurbine class.

A base class for the tile improvement hierarchy.

Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL https://www.sfml-dev.org/. 282

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL https://www.doxygen.nl. 281

Wikipedia. Hexagon, 2023. URL https://en.wikipedia.org/wiki/Hexagon. 40, 53, 106, 159, 169, 186, 205, 227, 245

Index