Road To Zero

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 AssetsManager Class Reference

A class which manages visual and sound assets.

```
#include <AssetsManager.h>
```

Public Member Functions

- AssetsManager (void)

  Constructor for the AssetsManager class.
- void loadFont (std::string, std::string)

  Method to load a font and insert it into the font map.
- void loadTexture (std::string, std::string)

  Method to load a texture and insert it into the texture map.
- void loadSound (std::string, std::string)

  Method to load a sound and insert it into the sound map. Automatically creates a corresponding sf::SoundBuffer.
- void loadTrack (std::string, std::string)

  Method to load a track (sf::Music) and insert it into the track map.
- sf::Font ∗ getFont (std::string)

  Method to get font associated with given font key.
- sf::Texture ∗ getTexture (std::string)

  Method to get texture associated with given texture key.
- sf::SoundBuffer ∗ getSoundBuffer (std::string)

  Method to get soundbuffer associated with given sound key.
- sf::Sound ∗ getSound (std::string)

  Method to get sound associated with given sound key.
- void playTrack (void)

  Method to play the current track.
- void pauseTrack (void)

  Method to pause the current track.
- void stopTrack (void)

  Method to stop the current track.
- void nextTrack (void)

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

- void previousTrack (void)

  Method to return to the previous track. Wraps around if the beginning of the track map is reached.
- std::string getCurrentTrackKey (void)

  Method to get track key for current track.
- sf::SoundSource::Status getTrackStatus (void)

  Method to get the status of the current track.
- void clear (void)

  Method to clear all loaded assets.
- ∼AssetsManager (void)

  Destructor for the AssetsManager class.

Public Attributes

- std::map< std::string, sf::Font ∗ > font_map

  A map of pointers to loaded fonts.
- std::map< std::string, sf::Texture ∗ > texture_map

  A map of pointers to loaded textures.
- std::map< std::string, sf::SoundBuffer ∗ > soundbuffer_map

  A map of pointers to sound buffers.
- std::map< std::string, sf::Sound ∗ > sound_map

  A map of pointers to loaded sounds.
- std::map< std::string, sf::Music ∗ >::iterator current_track

  A map iterator which corresponds to the current track (i.e., the track currently being played).
- std::map< std::string, sf::Music ∗ > track_map

  A map of pointers to opened tracks (i.e. sf::Music).

Private Member Functions

- void __loadSoundBuffer (std::string, std::string)

  Helper method to load a soundbuffer and insert it into the soundbuffer map. Should only be called by loadSound(), to create an sf::SoundBuffer corresponding to the loaded sf::Sound.

3.1.1 Detailed Description

A class which manages visual and sound assets.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 AssetsManager()

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

Constructor for the AssetsManager class.

```
{
    //...

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

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

3.1.2.2 ∼AssetsManager()

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

Destructor for the AssetsManager class.

```
{
    this->clear();

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

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

3.1.3 Member Function Documentation

3.1.3.1 __loadSoundBuffer()

```
void AssetsManager::__loadSoundBuffer (
             std::string path_2_sound,
             std::string sound_key )  [private]
```

Helper method to load a soundbuffer and insert it into the soundbuffer map. Should only be called by loadSound(), to create an sf::SoundBuffer corresponding to the loaded sf::Sound.

Parameters

path_2_sound	A path (either relative or absolute) to the sound file.
sound_key	A key associated with the sound (for indexing into the soundbuffer map).

```
{
    //  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() */
```

3.1.3.2 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() */
```

3.1.3.3 getCurrentTrackKey()

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

Method to get track key for current track.

Returns

The track key for the current track.

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

3.1.3.4 getFont()

```
sf::Font * AssetsManager::getFont (
               std::string font_key )
```

Method to get font associated with given font key.

Parameters

font_key	A key associated with the font (for indexing into the font map).

Returns

A pointer to the corresponding font.

```
{
    //  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() */
```

3.1.3.5 getSound()

```
sf::Sound * AssetsManager::getSound (
            std::string sound_key )
```

Method to get sound associated with given sound key.

Parameters

sound_key	A key associated with the sound (for indexing into the sound map).

Returns

A pointer to the corresponding sound.

```
{
    //  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() */
```

3.1.3.6 getSoundBuffer()

```
sf::SoundBuffer * AssetsManager::getSoundBuffer (
            std::string sound_key )
```

Method to get soundbuffer associated with given sound key.

Parameters

sound_key	A key associated with the soundbuffer (for indexing into the soundbuffer map).

Returns

A pointer to the corresponding soundbuffer.

```
{
    //  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() */
```

3.1.3.7 getTexture()

```
sf::Texture * AssetsManager::getTexture (
              std::string texture_key )
```

Method to get texture associated with given texture key.

Parameters

texture_key	A key associated with the texture (for indexing into the texture map).

Returns

A pointer to the corresponding texture.

```
{
    //  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() */
```

3.1.3.8 getTrackStatus()

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

Method to get the status of the current track.

Returns

The status of the current track.

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

3.1.3.9 loadFont()

```
void AssetsManager::loadFont (
            std::string path_2_font,
            std::string font_key )
```

Method to load a font and insert it into the font map.

Parameters

path_2_font	A path (either relative or absolute) to the font file.
font_key	A key associated with the font (for indexing into the font map).

```
{
    //  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() */
```

3.1.3.10 loadSound()

```
void AssetsManager::loadSound (
            std::string path_2_sound,
            std::string sound_key )
```

Method to load a sound and insert it into the sound map. Automatically creates a corresponding sf::SoundBuffer.

Parameters

path_2_sound	A path (either relative or absolute) to the sound file.
sound_key	A key associated with the sound (for indexing into the sound map).

```
{
    //  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() */
```

3.1.3.11 loadTexture()

```
void AssetsManager::loadTexture (
            std::string path_2_texture,
            std::string texture_key )
```

Method to load a texture and insert it into the texture map.

Parameters

path_2_texture	A path (either relative or absolute) to the texture file.
texture_key	A key associated with the texture (for indexing into the texture map).

```
{
    //  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() */
```

3.1.3.12 loadTrack()

```
void AssetsManager::loadTrack (
            std::string path_2_track,
            std::string track_key )
```

Method to load a track (sf::Music) and insert it into the track map.

Parameters

path_2_track	A path (either relative or absolute) to the track file.
track_key	A key associated with the track (for indexing into the track map).

```
{
    //  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() */
```

3.1.3.13 nextTrack()

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

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

```
{
    // 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() */
```

3.1.3.14 pauseTrack()

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

Method to pause the current track.

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

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

3.1.3.15 playTrack()

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

Method to play the current track.

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

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

3.1.3.16 previousTrack()

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

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

```
{
    // 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() */
```

3.1.3.17 stopTrack()

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

Method to stop the current track.

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

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

3.1.4 Member Data Documentation

3.1.4.1 current_track

```
std::map<std::string, sf::Music*>::iterator AssetsManager::current_track
```

A map iterator which corresponds to the current track (i.e., the track currently being played).

3.1.4.2 font_map

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

A map of pointers to loaded fonts.

3.1.4.3 sound_map

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

A map of pointers to loaded sounds.

3.1.4.4 soundbuffer_map

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

A map of pointers to sound buffers.

3.1.4.5 texture_map

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

A map of pointers to loaded textures.

3.1.4.6 track_map

```
std::map<std::string, sf::Music*> AssetsManager::track_map
```

A map of pointers to opened tracks (i.e. sf::Music).

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

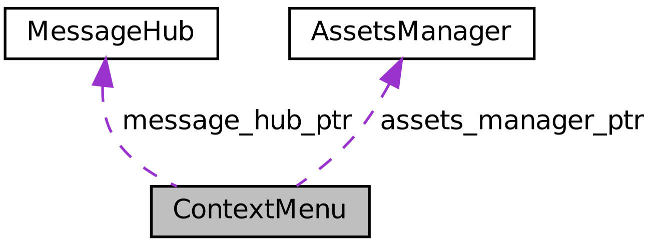
- header/ESC_core/AssetsManager.h
- source/ESC_core/AssetsManager.cpp

3.2 ContextMenu Class Reference

A class which defines a context menu for the game.

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

Collaboration diagram for ContextMenu:



Public Member Functions

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

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

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

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

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

  Destructor for the ContextMenu class.

Public Attributes

- ConsoleState console_state

  The current state of the console screen.
- bool game_menu_up

  Indicates whether or not the game menu is up.
- int frame

  The current frame of this object.
- double position_x

  The position of the object.
- double position_y

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

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

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

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

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

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

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

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

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

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

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

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

  The right framing of the console screen.

Private Member Functions

- void __setUpMenuFrame (void)

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

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

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

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

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

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

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

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

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

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

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

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

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

  Helper method to format and send a restart game message.

Private Attributes

- sf::Event ∗ event_ptr

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

  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr

  A pointer to the message hub.

3.2.1 Detailed Description

A class which defines a context menu for the game.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ContextMenu()

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

Constructor for the ContextMenu class.

Parameters

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->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() */
```

3.2.2.2 ∼ContextMenu()

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

Destructor for the ContextMenu class.

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

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

3.2.3 Member Function Documentation

3.2.3.1 __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() */
```

3.2.3.2 __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(
        this->console_string,
        *(this->assets_manager_ptr->getFont("Glass_TTY_VT220")),
        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() */
```

3.2.3.3 __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() */
```

3.2.3.4 __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() */
```

3.2.3.5 __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() */
```

3.2.3.6 __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() */
```

3.2.3.7 __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() */
```

3.2.3.8 __setConsoleState()

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

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

Parameters

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() */
```

3.2.3.9 __setConsoleString()

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

Helper method to set console string depending on console state.

```
{
    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                += "                                \n";
            this->console_string                += "READY.                          ";

            break;
        }
    }

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

3.2.3.10 __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() */
```

3.2.3.11 __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() */
```

3.2.3.12 __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() */
```

3.2.3.13 __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() */
```

3.2.3.14 __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() */
```

3.2.3.15 draw()

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

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

```
{
    //  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() */
```

3.2.3.16 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() */
```

3.2.3.17 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;

                    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() */
```

3.2.4 Member Data Documentation

3.2.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.2.4.2 console_screen

```
sf::RectangleShape ContextMenu::console_screen
```

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

3.2.4.3 console_screen_frame_bottom

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

The bottom framing of the console screen.

3.2.4.4 console_screen_frame_left

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

The left framing of the console screen.

3.2.4.5 console_screen_frame_right

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

The right framing of the console screen.

3.2.4.6 console_screen_frame_top

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

The top framing of the console screen.

3.2.4.7 console_state

```
ConsoleState ContextMenu::console_state
```

The current state of the console screen.

3.2.4.8 console_string

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

The string to be printed to the console screen.

3.2.4.9 event_ptr

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

A pointer to the event class.

3.2.4.10 frame

```
int ContextMenu::frame
```

The current frame of this object.

3.2.4.11 game_menu_up

```
bool ContextMenu::game_menu_up
```

Indicates whether or not the game menu is up.

3.2.4.12 menu_frame

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

The frame of the context menu.

3.2.4.13 message_hub_ptr

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

A pointer to the message hub.

3.2.4.14 position_x

```
double ContextMenu::position_x
```

The position of the object.

3.2.4.15 position_y

```
double ContextMenu::position_y
```

The position of the object.

3.2.4.16 render_window_ptr

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

A pointer to the render window.

3.2.4.17 visual_screen

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

The context menu screen for visuals.

3.2.4.18 visual_screen_frame_bottom

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

The bottom framing of the visual screen.

3.2.4.19 visual_screen_frame_left

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

The left framing of the visual screen.

3.2.4.20 visual_screen_frame_right

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

The right framing of the visual screen.

3.2.4.21 visual_screen_frame_top

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

The top framing of the visual screen.

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

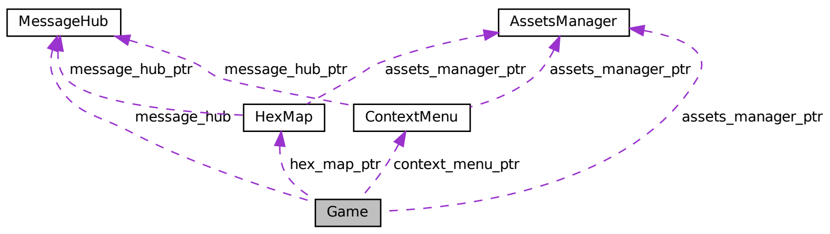
- header/ContextMenu.h
- source/ContextMenu.cpp

3.3 Game Class Reference

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

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

Collaboration diagram for Game:



Public Member Functions

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

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

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

  Destructor for the Game class.

Public Attributes

- bool quit_game

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

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

  Boolean indicating whether or not to show frame and clock overlay.
- unsigned long long int frame

  The current frame of the game.
- double time_since_start_s

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

  Cumulative emissions [tonnes] (1 tonne = 1000 kg).
- sf::Clock clock

  The game clock.
- sf::Event event

  The game events class.
- MessageHub message_hub

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

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

  Pointer to the context menu.

Private Member Functions

- void __toggleFrameClockOverlay (void)

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

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

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

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

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

  Helper method to format and send a game state 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.

3.3.1 Detailed Description

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

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Game()

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

Constructor for the Game class.

```
{
    //  1. set attributes

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

    this->assets_manager_ptr = assets_manager_ptr;

    //  1.2. public
    this->quit_game = false;
    this->game_loop_broken = false;
    this->show_frame_clock_overlay = false;

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

    this->population = 0;
    this->credits = 250;
    this->demand_MWh = 0;
    this->cumulative_emissions_tonnes = 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() */
```

3.3.2.2 ∼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() */
```

3.3.3 Member Function Documentation

3.3.3.1 __draw()

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

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

```
{
    this->__drawHUD();

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

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

3.3.3.2 __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() */
```

3.3.3.3 __drawHUD()

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

Helper method to heads-up display (HUD).

```
{
    //  1. first line
    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
    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);

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

3.3.3.4 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    switch (this->event.key.code) {
        case (sf::Keyboard::Tilde): {
            this->__toggleFrameClockOverlay();

            break;
        }


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

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

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

3.3.3.5 __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() */
```

3.3.3.6 __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. blocking display loop (~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) {
            this->render_window_ptr->clear();

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

            if (alarm_frame % 20 == 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++;
        }
    }

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

3.3.3.7 __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() */
```

3.3.3.8 __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);
        }
    }

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

3.3.3.9 __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;

    this->message_hub.sendMessage(game_state_message);

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

3.3.3.10 __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() */
```

3.3.3.11 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();
            }


            //  6.3. draw frame
            this->render_window_ptr->clear();

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

            this->render_window_ptr->display();


            //  6.4. increment frame
            this->frame++;
        }
    }

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

3.3.4 Member Data Documentation

3.3.4.1 assets_manager_ptr

`AssetsManager* Game::assets_manager_ptr [private]`

A pointer to the assets manager.

3.3.4.2 clock

`sf::Clock Game::clock`

The game clock.

3.3.4.3 context_menu_ptr

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

Pointer to the context menu.

3.3.4.4 credits

```
int Game::credits
```

Current balance of credits.

3.3.4.5 cumulative_emissions_tonnes

```
int Game::cumulative_emissions_tonnes
```

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

3.3.4.6 demand_MWh

```
int Game::demand_MWh
```

Current energy demand [MWh].

3.3.4.7 event

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

The game events class.

3.3.4.8 frame

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

The current frame of the game.

3.3.4.9 game_loop_broken

```
bool Game::game_loop_broken
```

Boolean indicating whether or not the game loop is broken.

3.3.4.10 hex_map_ptr

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

Pointer to the hex map (defines game world).

3.3.4.11 message_hub

```
MessageHub Game::message_hub
```

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

3.3.4.12 month

```
int Game::month
```

Current game month.

3.3.4.13 population

```
int Game::population
```

Current population.

3.3.4.14 quit_game

```
bool Game::quit_game
```

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

3.3.4.15 render_window_ptr

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

A pointer to the render window.

3.3.4.16 show_frame_clock_overlay

```
bool Game::show_frame_clock_overlay
```

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

3.3.4.17 time_since_start_s

```
double Game::time_since_start_s
```

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

3.3.4.18 year

```
int Game::year
```

Current game year.

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

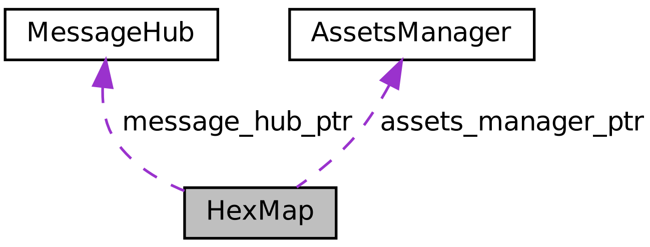
- header/Game.h
- source/Game.cpp

3.4 HexMap Class Reference

A class which defines a hex map of hex tiles.

`#include <HexMap.h>`

Collaboration diagram for HexMap:



Public Member Functions

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

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

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

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

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

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

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

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

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

  Destructor for the HexMap class.

Public Attributes

- bool tile_selected

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

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

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

  The current frame of this object.
- double position_x

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

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

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

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

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

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

  A position-indexed, nested map of hex tiles.
- 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.

Private Attributes

- sf::Event ∗ event_ptr

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

  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr

  A pointer to the message hub.

3.4.1 Detailed Description

A class which defines a hex map of hex tiles.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 HexMap()

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

Constructor (intended) for the HexMap class.

Parameters

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->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);

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

    return;
}   /* HexMap(), intended */
```

3.4.2.2 ∼HexMap()

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

Destructor for the HexMap class.

```
{
    this->clear();

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

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

3.4.3 Member Function Documentation

3.4.3.1 __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() */
```

3.4.3.2 __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() */
```

3.4.3.3 __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() */
```

3.4.3.4 __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() */
```

3.4.3.5 __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() */
```

3.4.3.6 __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() */
```

3.4.3.7 __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() */
```

3.4.3.8 __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() */
```

3.4.3.9 __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() */
```

3.4.3.10 __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() */
```

3.4.3.11 __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() */
```

3.4.3.12 __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() */
```

3.4.3.13 __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() */
```

3.4.3.14 __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() */
```

3.4.3.15 __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() */
```

3.4.3.16 __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() */
```

3.4.3.17 __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() */
```

3.4.3.18 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() */
```

3.4.3.19 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() */
```

3.4.3.20 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 in drawing order
    for (size_t i = 0; i < this->hex_draw_order_vec.size(); i++) {
        this->hex_draw_order_vec[i]->draw();
    }

    //  3. redraw selected tile
    HexTile* selected_tile_ptr = this->__getSelectedTile();
    if (selected_tile_ptr != NULL) {
        selected_tile_ptr->draw();
    }

    //  4. 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() */
```

3.4.3.21 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() */
```

3.4.3.22 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
    //...

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

3.4.3.23 reroll()

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

Method to re-roll the hex map.

```
{
    this->clear();
    this->__assembleHexMap();

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

3.4.3.24 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();
        }
    }

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

3.4.4 Member Data Documentation

3.4.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.4.4.2 border_tiles_vec

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

A vector of pointers to the border tiles.

3.4.4.3 event_ptr

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

A pointer to the event class.

3.4.4.4 frame

```
int HexMap::frame
```

The current frame of this object.

3.4.4.5 glass_screen

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

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

3.4.4.6 hex_draw_order_vec

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

A vector of hex tiles, in drawing order.

3.4.4.7 hex_map

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

A position-indexed, nested map of hex tiles.

3.4.4.8 message_hub_ptr

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

A pointer to the message hub.

3.4.4.9 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

3.4.4.10 n_tiles

```
int HexMap::n_tiles
```

The number of tiles in the hex map.

3.4.4.11 position_x

```
double HexMap::position_x
```

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

3.4.4.12 position_y

```
double HexMap::position_y
```

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

3.4.4.13 render_window_ptr

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

A pointer to the render window.

3.4.4.14 tile_position_x_vec

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

A vector of tile x positions.

3.4.4.15 tile_position_y_vec

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

A vector of tile y position.

3.4.4.16 tile_selected

```
bool HexMap::tile_selected
```

A boolean which indicates if a tile is currently selected.

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

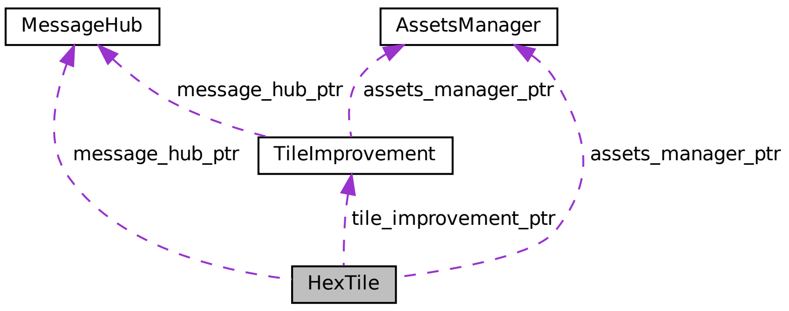
- header/HexMap.h
- source/HexMap.cpp

3.5 HexTile Class Reference

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

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

Collaboration diagram for HexTile:



Public Member Functions

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

  Constructor for the HexTile class.
- void setTileType (TileType)

  Method to set the tile type (by enum value).
- void setTileType (double)

  Method to set the tile type (by numeric input).
- void setTileResource (TileResource)

  Method to set the tile resource (by enum value).
- void setTileResource (double)

  Method to set the tile resource (by numeric input).
- void 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
- TileResource tile_resource
- bool show_node

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

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

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

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

    A boolean which indicates if a settlement has been built yet or not.
- 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.
- int frame

    The current frame of this object.
- int credits

    The current balance of credits.
- double position_x

    The x position of the tile.
- double position_y

    The y position of the tile.
- double major_radius

    The radius of the smallest bounding circle.
- double minor_radius

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

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

    A convex shape which represents the tile.
- sf::ConvexShape select_outline_sprite

    A convex shape which outlines the tile when selected.
- sf::CircleShape resource_chip_sprite

    A circle shape which represents a resource chip.
- sf::Text resource_text

    A text representation of the resource.
- sf::Sprite tile_decoration_sprite

    A tile decoration sprite.

Private Member Functions

- void __setUpNodeSprite (void)

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

  Helper method to set up tile sprite.
- void __setUpSelectOutlineSprite (void)

  Helper method to set up select outline sprite.
- void __setUpResourceChipSprite (void)

  Helper method to set up resource chip sprite.
- void __setResourceText (void)

  Helper method to set up resource text.
- bool __isClicked (void)

  Helper method to determine if tile was clicked on.
- void __handleKeyPressEvents (void)

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

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

  Helper method to format and send message on tile selection.
- std::string __getTileCoordsSubstring (void)

  Helper method to assemble and return tile coordinates substring.
- std::string __getTileTypeSubstring (void)

  Helper method to assemble and return tile type substring.
- std::string __getTileResourceSubstring (void)

  Helper method to assemble and return tile resource substring.
- std::string __getTileImprovementSubstring (void)

  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 __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.

Private Attributes

- sf::Event ∗ event_ptr

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

  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr

  A pointer to the message hub.

3.5.1 Detailed Description

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

3.5.2 Constructor & Destructor Documentation

3.5.2.1 HexTile()

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

Constructor for the HexTile class.

Ref: Wikipedia [2023]

Parameters

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->is_selected = false;

    this->has_improvement = false;
    this->settlement_built = false;
    this->tile_improvement_ptr = NULL;

    this->frame = 0;
    this->credits = 0;

    this->position_x = position_x;
    this->position_y = position_y;

    this->major_radius = 32;
    this->minor_radius = (sqrt(3) / 2) * this->major_radius;

    //  2. set up and position drawable attributes
    this->__setUpNodeSprite();
    this->__setUpTileSprite();
    this->__setUpSelectOutlineSprite();
    this->__setUpResourceChipSprite();
    this->__setResourceText();

    //  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() */
```

3.5.2.2 ∼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() */
```

3.5.3 Member Function Documentation

3.5.3.1 __getTileCoordsSubstring()

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

Helper method to assemble and return tile coordinates substring.

Returns

Tile coordinates substring.

```
{
    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() */
```

3.5.3.2 __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) {
        //...
    }

    else {
        improvement_substring += "NONE\n";
    }

    return improvement_substring;
}   /* __getTileImprovementSubstring() */
```

3.5.3.3 __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 (
        (not this->has_improvement) and
        (not this->settlement_built) and
        (this->tile_type != TileType :: OCEAN) and
        (this->tile_type != TileType :: LAKE)
    ) {
        options_substring                      += "[B]:  BUILD SETTLEMENT          ";
    }

    else if (not this->has_improvement) {
        switch (this->tile_type) {
            //...


            default: {
                // do nothing!

                break;
            }
        }
    }

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

3.5.3.4 __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) {
            //...


            default: {
                resource_substring += "???\n";

                break;
            }
        }
    }

    else {
        resource_substring += "[A]: ASSESS\n";
    }

    return resource_substring;
}   /* __getTileResourceSubstring() */
```

3.5.3.5 __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() */
```

3.5.3.6 __handleKeyPressEvents()

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

Helper method to handle key press events.

```
{
    if (this->event_ptr->key.code == sf::Keyboard::Escape) {
        this->is_selected = false;
    }

    if (this->is_selected) {
        switch (this->event_ptr->key.code) {
            case (sf::Keyboard::A): {
                if (this->resource_assessed) {
                    std::cout « "Cannot assess resource: already assessed" «
                        std::endl;
                }

                else 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->__sendGameStateRequest();
                }

                break;
            }

            default: {
                // do nothing!

                break;
            }
        }
    }


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

3.5.3.7 __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->is_selected = true;

                this->__sendTileSelectedMessage();
                this->__sendTileStateMessage();
                this->__sendGameStateRequest();
            }

            else {
                this->is_selected = false;
            }

            break;
        }


        case (sf::Mouse::Right): {
            this->is_selected = false;

            break;
        }


        default: {
            // do nothing!

            break;
        }
    }

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

3.5.3.8 __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() */
```

3.5.3.9 __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() */
```

3.5.3.10 __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() */
```

3.5.3.11 __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() */
```

3.5.3.12 __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() */
```

3.5.3.13 __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 string_payload                  = "      **** TILE INFO ****       \n";
    string_payload                             += "                                \n";

    string_payload                             += this->__getTileCoordsSubstring();
    string_payload                             += "                                \n";

    string_payload                             += this->__getTileTypeSubstring();
    string_payload                             += this->__getTileResourceSubstring();
    string_payload                             += this->__getTileImprovementSubstring();
    string_payload                             += "                                \n";

    string_payload                             += this->__getTileOptionsSubstring();


    tile_state_message.string_payload = string_payload;

    this->message_hub_ptr->sendMessage(tile_state_message);

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

3.5.3.14 __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() */
```

3.5.3.15 __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() */
```

3.5.3.16 __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() */
```

3.5.3.17 __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() */
```

3.5.3.18 __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() */
```

3.5.3.19 assess()

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

Method to assess the tile's resource.

```
{
    this->resource_assessed = true;
    this->__setResourceText();
    this->__sendTileStateMessage();

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

3.5.3.20 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) */
```

3.5.3.21 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
    this->render_window_ptr->draw(this->tile_decoration_sprite);

    //  4. draw resource
    if (this->show_resource) {
        this->render_window_ptr->draw(this->resource_chip_sprite);
        this->render_window_ptr->draw(this->resource_text);
    }

    //  5. 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) / (1.5 * FRAMES_PER_SECOND)), 2);

        this->select_outline_sprite.setOutlineColor(outline_colour);

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

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

3.5.3.22 processEvent()

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

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

```
{
    //  1. process TileImprovement events
    if (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::MouseButtonPressed) {
        this->__handleMouseButtonEvents();
    }

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

3.5.3.23 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"];

                if (this->tile_improvement_ptr != NULL) {
                    this->tile_improvement_ptr->credits = this->credits;
                }

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

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

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

3.5.3.24 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) */
```

3.5.3.25 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) */
```

3.5.3.26 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) */
```

3.5.3.27 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;
        }
    }

    return;
}   /* setTileType(TileType) */
```

3.5.3.28 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() */
```

3.5.4 Member Data Documentation

3.5.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.5.4.2 credits

```
int HexTile::credits
```

The current balance of credits.

3.5.4.3 event_ptr

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

A pointer to the event class.

3.5.4.4 frame

```
int HexTile::frame
```

The current frame of this object.

3.5.4.5 has_improvement

```
bool HexTile::has_improvement
```

A boolean which indicates if tile has improvement or not.

3.5.4.6 is_selected

```
bool HexTile::is_selected
```

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

3.5.4.7 major_radius

```
double HexTile::major_radius
```

The radius of the smallest bounding circle.

3.5.4.8 message_hub_ptr

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

A pointer to the message hub.

3.5.4.9 minor_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

3.5.4.10 node_sprite

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

A circle shape to mark the tile node.

3.5.4.11 position_x

```
double HexTile::position_x
```

The x position of the tile.

3.5.4.12 position_y

```
double HexTile::position_y
```

The y position of the tile.

3.5.4.13 render_window_ptr

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

A pointer to the render window.

3.5.4.14 resource_assessed

```
bool HexTile::resource_assessed
```

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

3.5.4.15 resource_chip_sprite

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

A circle shape which represents a resource chip.

3.5.4.16 resource_text

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

A text representation of the resource.

3.5.4.17 select_outline_sprite

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

A convex shape which outlines the tile when selected.

3.5.4.18 settlement_built

```
bool HexTile::settlement_built
```

A boolean which indicates if a settlement has been built yet or not.

3.5.4.19 show_node

```
bool HexTile::show_node
```

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

3.5.4.20 show_resource

```
bool HexTile::show_resource
```

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

3.5.4.21 tile_decoration_sprite

```
sf::Sprite HexTile::tile_decoration_sprite
```

A tile decoration sprite.

3.5.4.22 tile_improvement_ptr

```
TileImprovement* HexTile::tile_improvement_ptr
```

A pointer to the improvement for this tile.

3.5.4.23 tile_resource

```
TileResource HexTile::tile_resource
```

3.5.4.24 tile_sprite

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

A convex shape which represents the tile.

3.5.4.25 tile_type

```
TileType HexTile::tile_type
```

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

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

3.6 Message Struct Reference

A structure which defines a standard message format.

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

Public Attributes

- std::string channel = ""

  A string identifying the appropriate channel for this message.
- std::string subject = ""

  A string describing the message subject.
- std::map< std::string, bool > bool_payload = {}

  A boolean payload.
- std::map< std::string, int > int_payload = {}

  A vector payload.
- std::map< std::string, double > double_payload = {}

  A vector payload.
- std::string string_payload = ""

  A string payload.

3.6.1 Detailed Description

A structure which defines a standard message format.

3.6.2 Member Data Documentation

3.6.2.1 bool_payload

```
std::map<std::string, bool> Message::bool_payload = {}
```

A boolean payload.

3.6.2.2 channel

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

A string identifying the appropriate channel for this message.

3.6.2.3 double_payload

```
std::map<std::string, double> Message::double_payload = {}
```

A vector payload.

3.6.2.4 int_payload

```
std::map<std::string, int> Message::int_payload = {}
```

A vector payload.

3.6.2.5 string_payload

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

A string payload.

3.6.2.6 subject

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

A string describing the message subject.

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

- header/ESC_core/MessageHub.h

3.7 MessageHub Class Reference

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

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

Public Member Functions

- MessageHub (void)

  Constructor for the MessageHub class.
- bool hasTraffic (void)

  Method to determine if there remains any message traffic.
- void addChannel (std::string)

  Method to add channel to message map.
- void removeChannel (std::string)

  Method to remove channel from message map.
- void sendMessage (Message)

  Method to send a message to the message map. 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.

3.7.1 Detailed Description

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

3.7.2 Constructor & Destructor Documentation

3.7.2.1 MessageHub()

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

Constructor for the MessageHub class.

```
{
    //...

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

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

3.7.2.2 ∼MessageHub()

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

Destructor for the MessageHub class.

```
{
    this->clear();

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

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

3.7.3 Member Function Documentation

3.7.3.1 addChannel()

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

Method to add channel to message map.

Parameters

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() */
```

3.7.3.2 clear()

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

Method to clear the MessageHub.

```
{

    this->clearMessages();
    this->message_map.clear();

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

3.7.3.3 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() */
```

3.7.3.4 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() */
```

3.7.3.5 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() */
```

3.7.3.6 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() */
```

3.7.3.7 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() */
```

3.7.3.8 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() */
```

3.7.3.9 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() */
```

3.7.4 Member Data Documentation

3.7.4.1 message_map

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

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

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

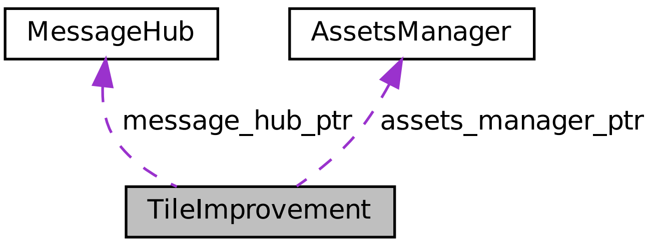
- header/ESC_core/MessageHub.h
- source/ESC_core/MessageHub.cpp

3.8 TileImprovement Class Reference

A base class for the tile improvement hierarchy.

```
#include <TileImprovement.h>
```

Collaboration diagram for TileImprovement:



Public Member Functions

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

    Constructor for the TileImprovement class.
- 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

- int frame
  The current frame of this object.
- int credits
  The current balance of credits.
- double position_x
  The x position of the tile improvement.
- double position_y
  The y position of the tile improvement.
- 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.

Private Member Functions

- virtual void __handleKeyPressEvents (void)
  Helper method to handle key press events.
- virtual void __handleMouseButtonEvents (void)
  Helper method to handle mouse button events.

Private Attributes

- sf::Event ∗ event_ptr
  A pointer to the event class.
- sf::RenderWindow ∗ render_window_ptr
  A pointer to the render window.
- AssetsManager ∗ assets_manager_ptr
  A pointer to the assets manager.
- MessageHub ∗ message_hub_ptr
  A pointer to the message hub.

3.8.1 Detailed Description

A base class for the tile improvement hierarchy.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 TileImprovement()

```
TileImprovement::TileImprovement (
             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 TileImprovement 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->frame = 0;
    this->credits = 0;

    this->position_x = position_x;
    this->position_y = position_y;

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

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

3.8.2.2 ∼TileImprovement()

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

Destructor for the TileImprovement class.

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

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

3.8.3 Member Function Documentation

3.8.3.1 __handleKeyPressEvents()

```
void TileImprovement::__handleKeyPressEvents (
               void  )  [private], [virtual]
```

Helper method to handle key press events.

```
{
    switch (this->event_ptr->key.code) {
        //...


        default: {
            // do nothing!

            break;
        }
    }

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

3.8.3.2 __handleMouseButtonEvents()

```
void TileImprovement::__handleMouseButtonEvents (
            void ) [private], [virtual]
```

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() */
```

3.8.3.3 draw()

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

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

```
{
    //...

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

3.8.3.4 processEvent()

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

Method to process TileImprovement. 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() */
```

3.8.3.5 processMessage()

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

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

```
{
    //...

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

3.8.4 Member Data Documentation

3.8.4.1 assets_manager_ptr

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

A pointer to the assets manager.

3.8.4.2 credits

```
int TileImprovement::credits
```

The current balance of credits.

3.8.4.3 event_ptr

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

A pointer to the event class.

3.8.4.4 frame

```
int TileImprovement::frame
```

The current frame of this object.

3.8.4.5 message_hub_ptr

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

A pointer to the message hub.

3.8.4.6 position_x

```
double TileImprovement::position_x
```

The x position of the tile improvement.

3.8.4.7 position_y

```
double TileImprovement::position_y
```

The y position of the tile improvement.

3.8.4.8 render_window_ptr

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

A pointer to the render window.

3.8.4.9 tile_improvement_sprite_animated

```
std::vector<sf::Sprite> TileImprovement::tile_improvement_sprite_animated
```

An animated sprite, for the ContextMenu visual screen.

3.8.4.10 tile_improvement_sprite_static

```
sf::Sprite TileImprovement::tile_improvement_sprite_static
```

A static sprite, for decorating the tile.

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

- header/TileImprovement.h
- source/TileImprovement.cpp

Chapter 4

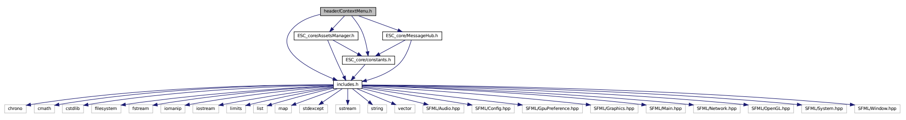
File Documentation

4.1 header/ContextMenu.h File Reference

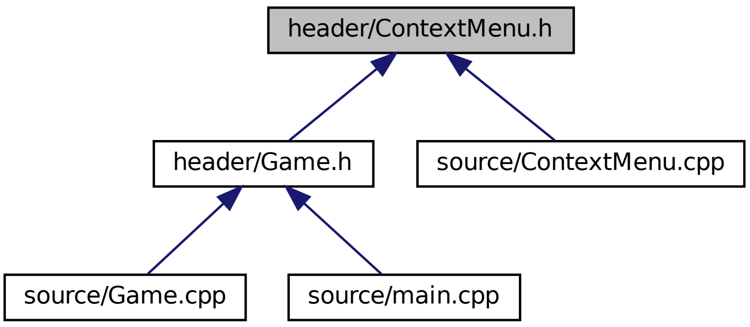
Header file for the ContextMenu class.

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

Include dependency graph for ContextMenu.h:



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



Classes

- class ContextMenu

  A class which defines a context menu for the game.

Enumerations

- enum ConsoleState {
  NONE_STATE , READY , MENU , TILE ,
  N_CONSOLE_STATES }

  An enumeration of the different console screen states.

4.1.1 Detailed Description

Header file for the ContextMenu class.

4.1.2 Enumeration Type Documentation

4.1.2.1 ConsoleState

```
enum ConsoleState
```

An enumeration of the different console screen states.

Enumerator

NONE_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.

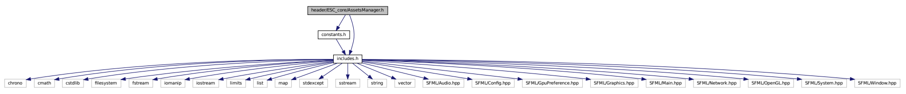
```
                     {
    NONE_STATE,
    READY,
    MENU,
    TILE,
    N_CONSOLE_STATES
};
```

4.2 header/ESC_core/AssetsManager.h File Reference

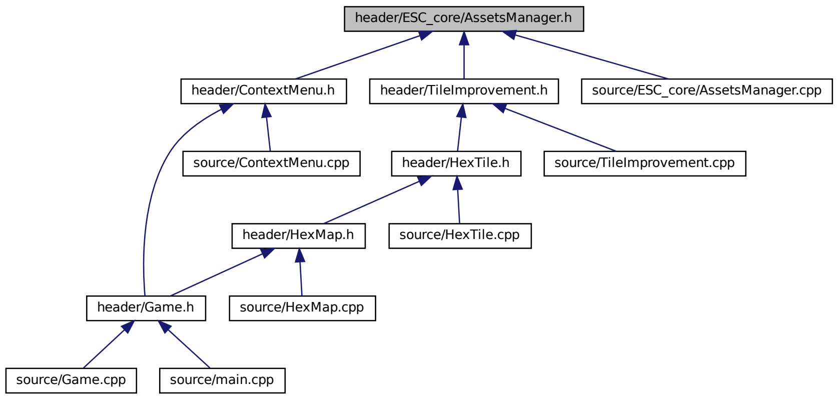
Header file for the AssetsManager class.

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

Include dependency graph for AssetsManager.h:



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



Classes

- class AssetsManager

    A class which manages visual and sound assets.

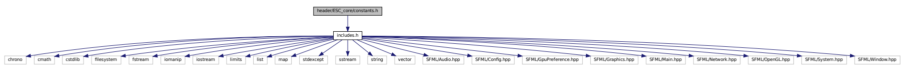
4.2.1 Detailed Description

Header file for the AssetsManager class.

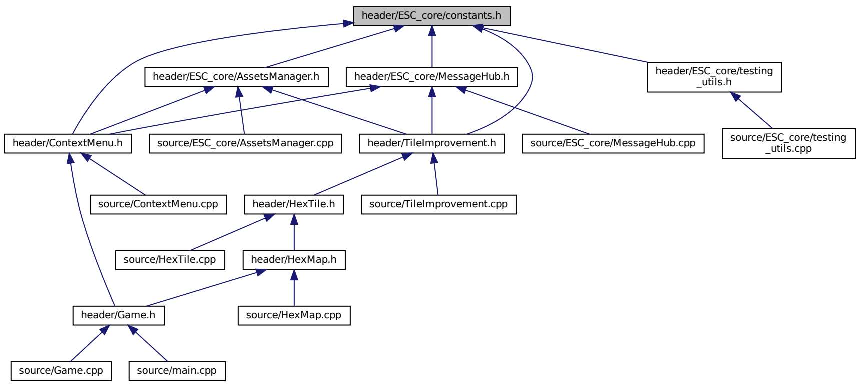
4.3 header/ESC_core/constants.h File Reference

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
header/ContextMenu.h
source/ESC_core/AssetsManager.cpp
header/TileImprovement.h
source/ESC_core/MessageHub.cpp
source/ESC_core/testing_utils.cpp
source/ContextMenu.cpp
header/HexTile.h
source/TileImprovement.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 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 double CO2E_KG_PER_LITRE_DIESEL = 3.1596

  The CO2-equivalent mass of emissions that result from burning one litre of diesel fuel.
- 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.

4.3.1 Detailed Description

Header file for various constants.

4.3.2 Function Documentation

4.3.2.1 FOREST_GREEN()

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

The base colour of a forest tile.

4.3.2.2 LAKE_BLUE()

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

The base colour of a lake (water) tile.

4.3.2.3 MENU_FRAME_GREY()

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

The base colour of the context menu frame.

4.3.2.4 MONOCHROME_SCREEN_BACKGROUND()

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

The base colour of old monochrome screens.

4.3.2.5 MONOCHROME_TEXT_AMBER()

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

The base colour of old monochrome text (amber).

4.3.2.6 MONOCHROME_TEXT_GREEN()

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

The base colour of old monochrome text (green).

4.3.2.7 MONOCHROME_TEXT_RED()

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

The base colour of old monochrome text (red).

4.3.2.8 MOUNTAINS_GREY()

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

The base colour of a mountains tile.

4.3.2.9 OCEAN_BLUE()

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

The base colour of an ocean (water) tile.

4.3.2.10 PLAINS_YELLOW()

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

The base colour of a plains tile.

4.3.2.11 RESOURCE_CHIP_GREY()

```
const sf::Color RESOURCE_CHIP_GREY (
             175 ,
             175 ,
             175 ,
             250  )
```

The base colour of the resource chip (backing).

4.3.2.12 VISUAL_SCREEN_FRAME_GREY()

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

The base colour of the framing of the visual screen.

4.3.3 Variable Documentation

4.3.3.1 CO2E_KG_PER_LITRE_DIESEL

```
const double CO2E_KG_PER_LITRE_DIESEL = 3.1596
```

The CO2-equivalent mass of emissions that result from burning one litre of diesel fuel.

4.3.3.2 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.

4.3.3.3 FLOAT_TOLERANCE

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

Tolerance for floating point equality tests.

4.3.3.4 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

4.3.3.5 GAME_CHANNEL

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

A message channel for game messages.

4.3.3.6 GAME_HEIGHT

```
const int GAME_HEIGHT = 800
```

Height of the game space.

4.3.3.7 GAME_STATE_CHANNEL

```
const std::string GAME_STATE_CHANNEL = "GAME STATE CHANNEL"
```

A message channel for game state messages.

4.3.3.8 GAME_WIDTH

```
const int GAME_WIDTH = 1200
```

Width of the game space.

4.3.3.9 NO_TILE_SELECTED_CHANNEL

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

A message channel for no tile selected messages.

4.3.3.10 RESOURCE_ASSESSMENT_COST

```
const int RESOURCE_ASSESSMENT_COST = 20
```

The cost of doing a resource assessment.

4.3.3.11 SECONDS_PER_FRAME

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

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

4.3.3.12 SECONDS_PER_MONTH

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

4.3.3.13 SECONDS_PER_YEAR

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

4.3.3.14 TILE_RESOURCE_CUMULATIVE_PROBABILITIES

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

Initial value:

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

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

4.3.3.15 TILE_SELECTED_CHANNEL

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

A message channel for tile selection messages.

4.3.3.16 TILE_STATE_CHANNEL

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

A message channel for tile state messages.

4.3.3.17 TILE_TYPE_CUMULATIVE_PROBABILITIES

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

Initial value:

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

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

4.4 header/ESC_core/doxygen_cite.h File Reference

Header file which simply cites the doxygen tool.

4.4.1 Detailed Description

Header file which simply cites the doxygen tool.

Ref: van Heesch. [2023]

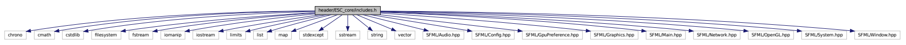
4.5 header/ESC_core/includes.h File Reference

Header file for various includes.

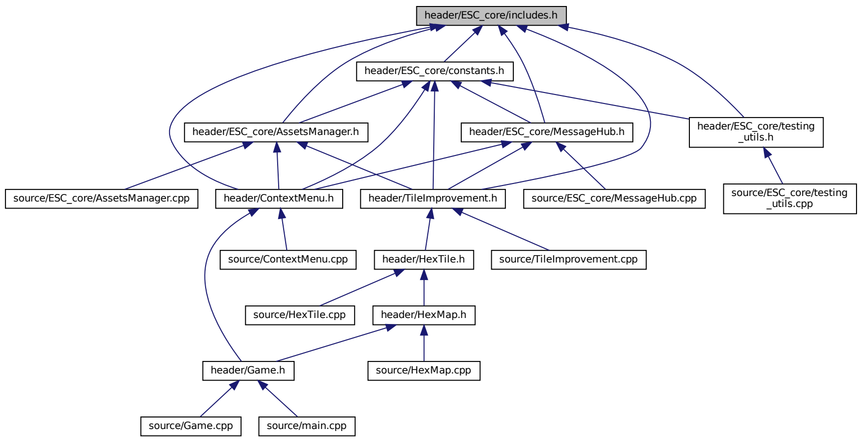
```
#include <chrono>
#include <cmath>
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <limits>
#include <list>
#include <map>
#include <stdexcept>
#include <sstream>
#include <string>
#include <vector>
#include <SFML/Audio.hpp>
#include <SFML/Config.hpp>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

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

Include dependency graph for includes.h:



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


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
source/ESC_core/AssetsManager.cpp
header/ContextMenu.h
header/TileImprovement.h
source/ESC_core/MessageHub.cpp
source/ESC_core/testing _utils.cpp
source/ContextMenu.cpp
header/HexTile.h
source/TileImprovement.cpp
source/HexTile.cpp
header/HexMap.h
header/Game.h
source/HexMap.cpp
source/Game.cpp
source/main.cpp

4.5.1 Detailed Description

Header file for various includes.

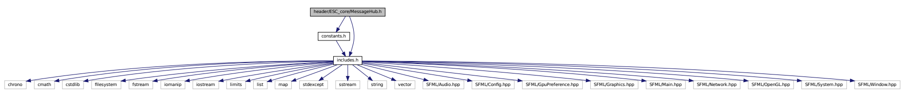
Ref: Gomila [2023]

4.6 header/ESC_core/MessageHub.h File Reference

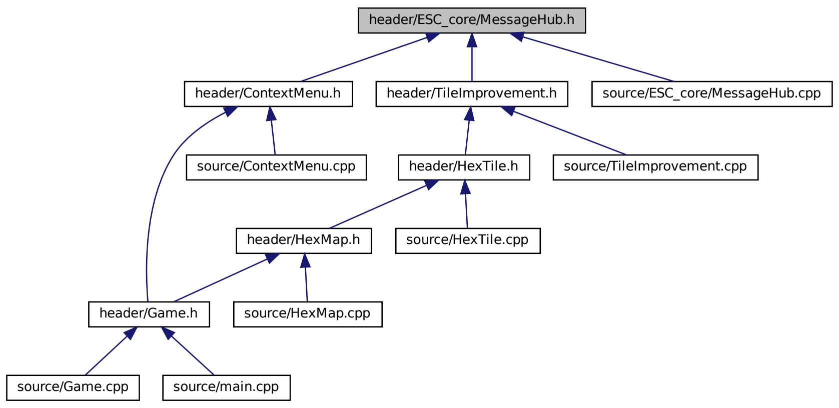
Header file for the MessageHub class.

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

Include dependency graph for MessageHub.h:



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



Classes

- struct Message

    A structure which defines a standard message format.

- class MessageHub

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

4.6.1 Detailed Description

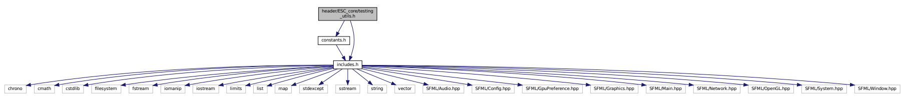
Header file for the MessageHub class.

4.7 header/ESC_core/testing_utils.h File Reference

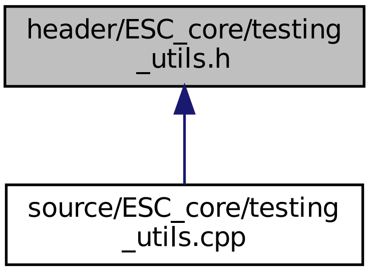
Header file for various testing utilities.

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

Include dependency graph for testing_utils.h:



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


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.

4.7.1 Detailed Description

Header file for various testing utilities.

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

4.7.2 Function Documentation

4.7.2.1 expectedErrorNotDetected()

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

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

Parameters

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

```
{
    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() */
```

4.7.2.2 printGold()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[33m" « input_str « "\033[0m";
    return;
}   /* printGold() */
```

4.7.2.3 printGreen()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[32m" « input_str « "\033[0m";
    return;
}   /* printGreen() */
```

4.7.2.4 printRed()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[31m" « input_str « "\033[0m";
    return;
}   /* printRed() */
```

4.7.2.5 testFloatEquals()

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

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

Parameters

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

```
{
    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() */
```

4.7.2.6 testGreaterThan()

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

Tests if x > y.

Parameters

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

```
{
    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() */
```

4.7.2.7 testGreaterThanOrEqualTo()

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

Tests if x >= y.

Parameters

x	The first of two numbers to test.

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() */
```

4.7.2.8 testLessThan()

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

Tests if x $<$ y.

Parameters

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

```
{
    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() */
```

4.7.2.9 testLessThanOrEqualTo()

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

Tests if x <= y.

Parameters

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

```
{
    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() */
```

4.7.2.10 testTruth()

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

Tests if the given statement is true.

Parameters

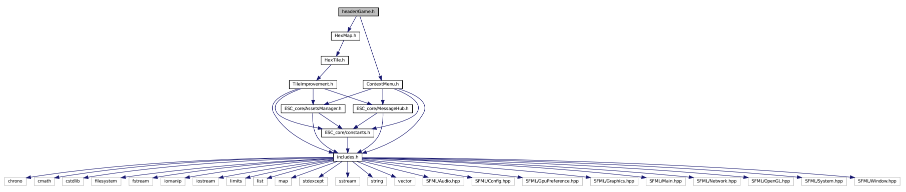
statement	The statement whose truth is to be tested ("1 == 0", for example).
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    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() */
```

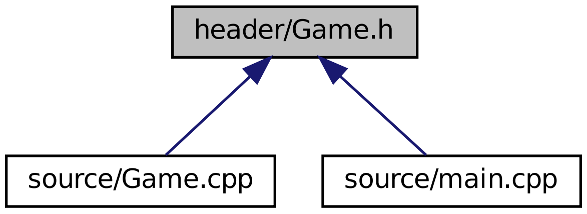
4.8 header/Game.h File Reference

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

Include dependency graph for Game.h:



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



Classes

- class Game

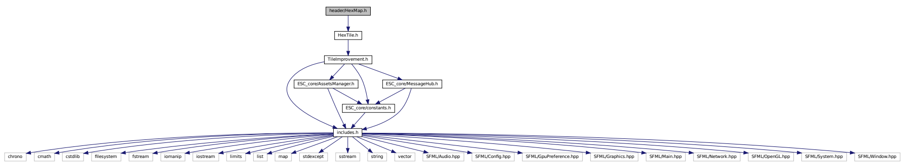
  A class which acts as the central class for the game, by containing all other classes and implementing the game loop.

4.9 header/HexMap.h File Reference

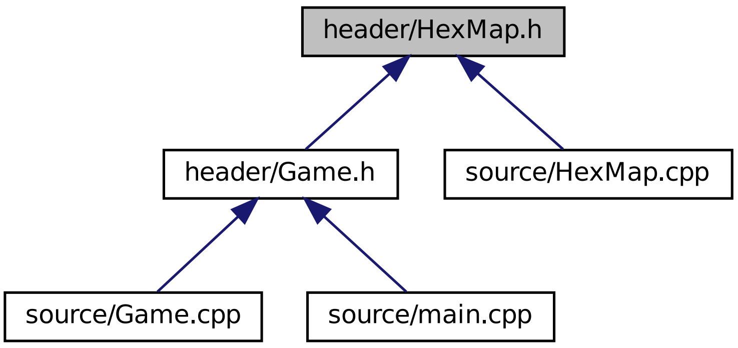
Header file for the HexMap class.

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

Include dependency graph for HexMap.h:



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



Classes

- class HexMap

  A class which defines a hex map of hex tiles.

4.9.1 Detailed Description

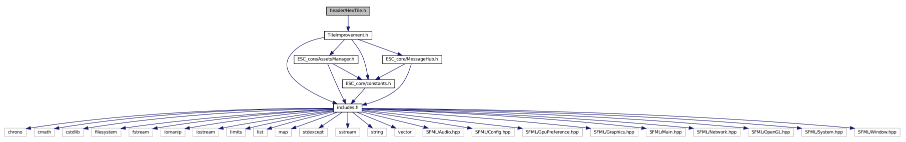
Header file for the HexMap class.

4.10 header/HexTile.h File Reference

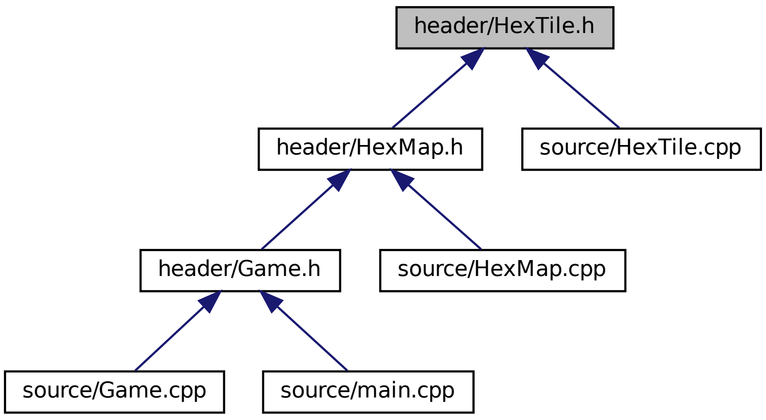
Header file for the Game class.

```
#include "TileImprovement.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.

4.10.1 Detailed Description

Header file for the Game class.

Header file for the HexTile class.

4.10.2 Enumeration Type Documentation

4.10.2.1 TileResource

```
enum TileResource
```

An enumeration of the different tile resource values.

Enumerator

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

```
                    {
    POOR,
    BELOW_AVERAGE,
    AVERAGE,
    ABOVE_AVERAGE,
    GOOD,
    N_TILE_RESOURCES
};  /* TileResource */
```

4.10.2.2 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.

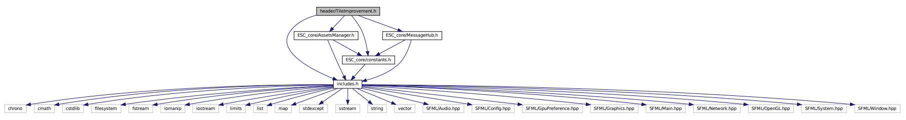
```
                {
    NONE_TYPE,
    FOREST,
    LAKE,
    MOUNTAINS,
    OCEAN,
    PLAINS,
    N_TILE_TYPES
}; /* TileType */
```

4.11 header/TileImprovement.h File Reference

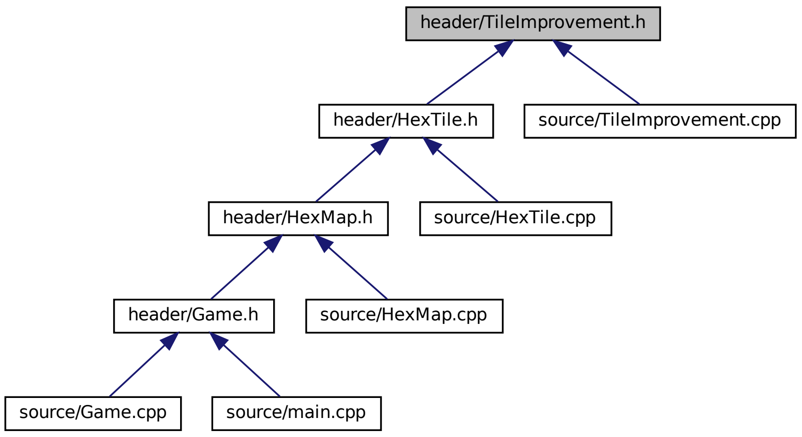
Header file for the TileImprovement class.

```
#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 , SOLAR_PV , WIND_TURBINE , TIDAL_TURBINE ,
  WAVE_ENERGY_CONVERTER , ENERGY_STORAGE_SYSTEM , N_TILE_IMPROVEMENT_TYPES }

  An enumeration of the different tile improvement types.

4.11.1 Detailed Description

Header file for the TileImprovement class.

4.11.2 Enumeration Type Documentation

4.11.2.1 TileImprovementType

`enum TileImprovementType`

An enumeration of the different tile improvement types.

Enumerator

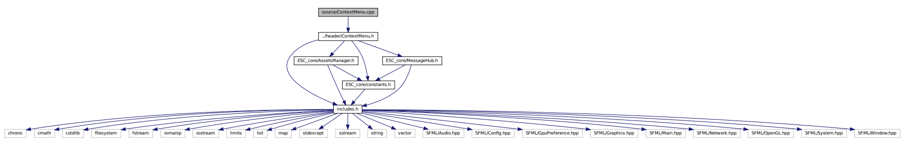
SETTLEMENT	A settlement.
SOLAR_PV	A solar PV array.
WIND_TURBINE	A wind turbine.
TIDAL_TURBINE	A tidal turbine.
WAVE_ENERGY_CONVERTER	A wave energy converter.
ENERGY_STORAGE_SYSTEM	An energy storage system.
N_TILE_IMPROVEMENT_TYPES	A simple hack to get the number of elements in TileImprovementType.

```
                        {
    SETTLEMENT,
    SOLAR_PV,
    WIND_TURBINE,
    TIDAL_TURBINE,
    WAVE_ENERGY_CONVERTER,
    ENERGY_STORAGE_SYSTEM,
    N_TILE_IMPROVEMENT_TYPES
};  /* TileImprovementType */
```

4.12 source/ContextMenu.cpp File Reference

Implementation file for the ContextMenu class.

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



4.12.1 Detailed Description

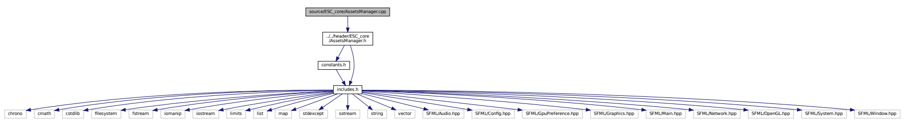
Implementation file for the ContextMenu class.

A class which defines a context menu for the game.

4.13 source/ESC_core/AssetsManager.cpp File Reference

Implementation file for the AssetsManager class.

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



4.13.1 Detailed Description

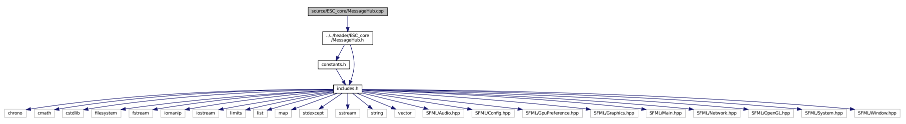
Implementation file for the AssetsManager class.

A class which manages visual and sound assets.

4.14 source/ESC_core/MessageHub.cpp File Reference

Implementation file for the MessageHub class.

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



4.14.1 Detailed Description

Implementation file for the MessageHub class.

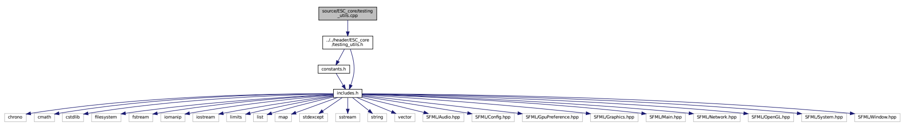
A class which acts as a central hub for inter-object message traffic.

4.15 source/ESC_core/testing_utils.cpp File Reference

Implementation file for various testing utilities.

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

Include dependency graph for testing_utils.cpp:



Functions

- void printGreen (std::string input_str)

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

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

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

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

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

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

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

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

  Tests if the given statement is true.
- void expectedErrorNotDetected (std::string file, int line)

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

4.15.1 Detailed Description

Implementation file for various testing utilities.

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

4.15.2 Function Documentation

4.15.2.1 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() */
```

4.15.2.2 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() */
```

4.15.2.3 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() */
```

4.15.2.4 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() */
```

4.15.2.5 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() */
```

4.15.2.6 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() */
```

4.15.2.7 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() */
```

4.15.2.8 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() */
```

4.15.2.9 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() */
```

4.15.2.10 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__").

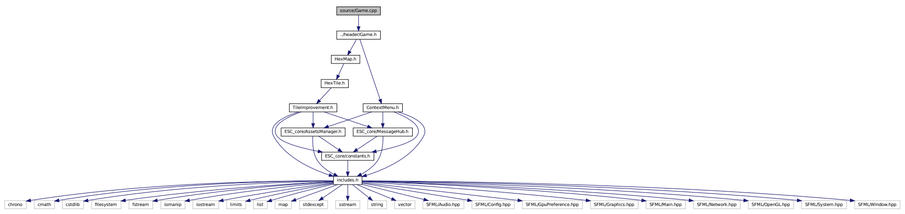
```
{
    if (statement) {
        return;
    }

    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() */
```

4.16 source/Game.cpp File Reference

Implementation file for the Game class.

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

Include dependency graph for Game.cpp:



4.16.1 Detailed Description

Implementation file for the Game class.

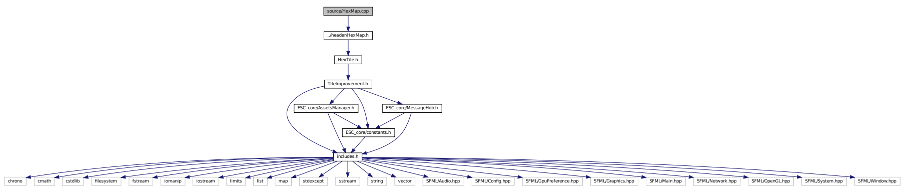
A class which defines a tile of a hex map.

4.17 source/HexMap.cpp File Reference

Implementation file for the HexMap class.

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

Include dependency graph for HexMap.cpp:



4.17.1 Detailed Description

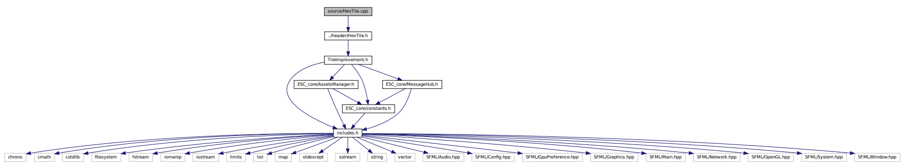
Implementation file for the HexMap class.

A class which defines a hex map of hex tiles.

4.18 source/HexTile.cpp File Reference

Implementation file for the HexTile class.

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



4.18.1 Detailed Description

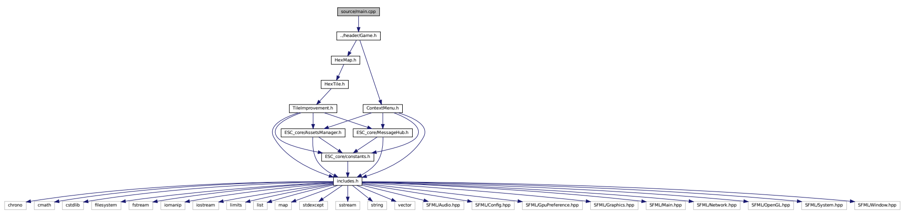
Implementation file for the HexTile class.

A class which defines a tile of a hex map.

4.19 source/main.cpp File Reference

Implementation file for main() for Road To Zero.

```
#include "../header/Game.h"
```
Include dependency graph for main.cpp:



Functions

- void loadAssets (AssetsManager ∗assets_manager_ptr)

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

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

4.19.1 Detailed Description

Implementation file for main() for Road To Zero.

4.19.2 Function Documentation

4.19.2.1 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() */
```

4.19.2.2 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.png",
        "pine_tree_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/wheat_64x64_1.png",
        "wheat_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/mountain_64x64_1.png",
        "mountain_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/water_waves_64x64_1.png",
        "water_waves_64x64_1"
    );

    assets_manager_ptr->loadTexture(
        "assets/tile_sheets/water_shimmer_64x64_1.png",
        "water_shimmer_64x64_1"
    );


    //  3. load sounds
    assets_manager_ptr->loadSound(
        "assets/audio/samples/mixkit-apartment-buzzer-bell-press-932.ogg",
        "insufficient credits"
    );

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

4.19.2.3 main()

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

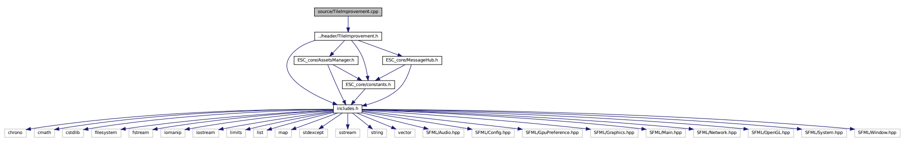
```
{
    //  1. load assets
    AssetsManager assets_manager;
    loadAssets(&assets_manager);

    //  2. construct render window
    sf::RenderWindow* render_window_ptr = constructRenderWindow();

    //  3. start game loop
    bool quit_game = false;

    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() */
```

4.20 source/TileImprovement.cpp File Reference

Implementation file for the TileImprovement class.

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

Include dependency graph for TileImprovement.cpp:



4.20.1 Detailed Description

Implementation file for the TileImprovement 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/. 118

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL https://www.doxygen.nl. 117

Wikipedia. Hexagon, 2023. URL https://en.wikipedia.org/wiki/Hexagon. 72, 102

Index