HelloWorld

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 AssetsManager Class Reference

A class which manages visual and sound assets.

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

Public Member Functions

- AssetsManager (void)

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

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

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

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

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

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

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

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

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

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

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

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

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

- void previousTrack (void)

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

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

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

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

  Destructor for the AssetsManager class.

Public Attributes

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

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

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

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

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

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

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

Private Member Functions

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

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

3.1.1 Detailed Description

A class which manages visual and sound assets.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 AssetsManager()

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

Constructor for the AssetsManager class.

```
{
    //...

    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:

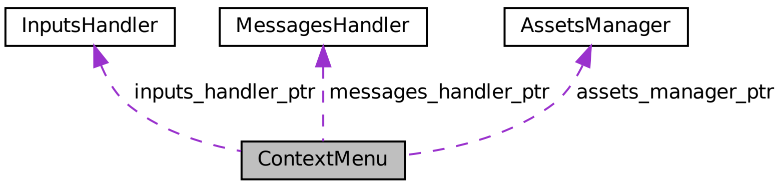
- header/ESC_core/AssetsManager.h
- source/ESC_core/AssetsManager.cpp

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 (AssetsManager ∗, InputsHandler ∗, MessagesHandler ∗, sf::RenderWindow ∗)
  Constructor for the ContextMenu class.
- void process (void)
  Method to process ContextMenu. To be called once per event.
- 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.

Private Attributes

- unsigned long long int address_int

    An int representation of the memory address of this object.
- std::string address_string

    A string representation of the hex address of this object.
- AssetsManager ∗ assets_manager_ptr

    A pointer to the assets manager.
- InputsHandler ∗ inputs_handler_ptr

    A pointer to the inputs handler.
- MessagesHandler ∗ messages_handler_ptr

    A pointer to the messages handler.
- sf::RenderWindow ∗ render_window_ptr

    A pointer to the render window.

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 (
             AssetsManager * assets_manager_ptr,
             InputsHandler * inputs_handler_ptr,
             MessagesHandler * messages_handler_ptr,
             sf::RenderWindow * render_window_ptr )
```

Constructor for the ContextMenu class.

Parameters

assets_manager_ptr	Pointer to the assets manager.
inputs_handler_ptr	Pointer to the inputs handler.
messages_handler_ptr	Pointer to the messages handler.
render_window_ptr	Pointer to the render window.

```
{
    //  1. set attributes
    this->address_int = (unsigned long long int)this;

    std::stringstream ss;
    ss « std::hex « this;
    this->address_string = ss.str();

    this->assets_manager_ptr = assets_manager_ptr;
    this->inputs_handler_ptr = inputs_handler_ptr;
    this->messages_handler_ptr = messages_handler_ptr;
    this->render_window_ptr = render_window_ptr;

    this->console_state = ConsoleState :: NONE;
    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 « " (" « this->address_int
        « ")" « std::endl;

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

3.2.2.2 ∼ContextMenu()

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

Destructor for the ContextMenu class.

```
{
    std::cout « "ContextMenu at " « this « " (" « this->address_int
        « ") 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,
        *(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,
            *(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 __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() */
```

3.2.3.5 __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                += "[T]:  TUTORIAL                  \n";
            this->console_string                += "                                \n";
            this->console_string                += "[R]:  RESTART                   \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                += "                                \n";
            this->console_string                += "[ESC]:  CLOSE MENU              \n";
            this->console_string                += "                                \n";

            break;
        }


        case (ConsoleState :: TILE): {
            // console string set from tile 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                += "[ESC]:        MENU              \n";
            this->console_string                += "[LEFT CLICK]: TILE INFO/OPTIONS \n";
            this->console_string                  += "                                  \n";
            this->console_string                  += "READY.                            ";

            break;
        }
    }

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

3.2.3.6 __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() */
```

3.2.3.7 __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() */
```

3.2.3.8 __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() */
```

3.2.3.9 __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() */
```

3.2.3.10 __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() */
```

3.2.3.11 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() */
```

3.2.3.12 process()

```
void ContextMenu::process (
            void  )
```

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

```
{
    //  1. handle inputs
    if (this->inputs_handler_ptr->key_pressed_once_vec[sf::Keyboard::Escape]) {
        switch (this->console_state) {
            case (ConsoleState :: MENU): {
                this->__setConsoleState(ConsoleState :: READY);

                break;
            }


            default: {
                this->__setConsoleState(ConsoleState :: MENU);

                break;
            }
        }
    }


    if (this->inputs_handler_ptr->key_pressed_once_vec[sf::Keyboard::Q]) {
        switch (this->console_state) {
            case (ConsoleState :: MENU): {
                this->render_window_ptr->close();

                break;
            }


            default: {
                // do nothing!

                break;
            }
        }
    }


    if (inputs_handler_ptr->mouse_left_click) {
        if (not this->messages_handler_ptr->isEmpty(MESSAGE_CHANNEL_TILE)) {
            Message selected_message = this->messages_handler_ptr->receiveMessage(
                MESSAGE_CHANNEL_TILE
            );

            if (selected_message.subject == "DUMMY") {
                this->__setConsoleState(ConsoleState :: READY);
            }

            else {
                this->__setConsoleState(ConsoleState :: TILE);
                this->console_string = selected_message.string_payload;
            }
        }
    }


    if (this->inputs_handler_ptr->mouse_right_click) {
        this->__setConsoleState(ConsoleState :: READY);
    }

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

3.2.4 Member Data Documentation

3.2.4.1 address_int

```
unsigned long long int ContextMenu::address_int  [private]
```

An int representation of the memory address of this object.

3.2.4.2 address_string

```
std::string ContextMenu::address_string  [private]
```

A string representation of the hex address of this object.

3.2.4.3 assets_manager_ptr

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

A pointer to the assets manager.

3.2.4.4 console_screen

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

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

3.2.4.5 console_screen_frame_bottom

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

The bottom framing of the console screen.

3.2.4.6 console_screen_frame_left

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

The left framing of the console screen.

3.2.4.7 console_screen_frame_right

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

The right framing of the console screen.

3.2.4.8 console_screen_frame_top

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

The top framing of the console screen.

3.2.4.9 console_state

```
ConsoleState ContextMenu::console_state
```

The current state of the console screen.

3.2.4.10 console_string

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

The string to be printed to the console screen.

3.2.4.11 frame

```
int ContextMenu::frame
```

The current frame of this object.

3.2.4.12 game_menu_up

```
bool ContextMenu::game_menu_up
```

Indicates whether or not the game menu is up.

3.2.4.13 inputs_handler_ptr

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

A pointer to the inputs handler.

3.2.4.14 menu_frame

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

The frame of the context menu.

3.2.4.15 messages_handler_ptr

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

A pointer to the messages handler.

3.2.4.16 position_x

```
double ContextMenu::position_x
```

The position of the object.

3.2.4.17 position_y

```
double ContextMenu::position_y
```

The position of the object.

3.2.4.18 render_window_ptr

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

A pointer to the render window.

3.2.4.19 visual_screen

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

The context menu screen for visuals.

3.2.4.20 visual_screen_frame_bottom

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

The bottom framing of the visual screen.

3.2.4.21 visual_screen_frame_left

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

The left framing of the visual screen.

3.2.4.22 visual_screen_frame_right

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

The right framing of the visual screen.

3.2.4.23 visual_screen_frame_top

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

The top framing of the visual screen.

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

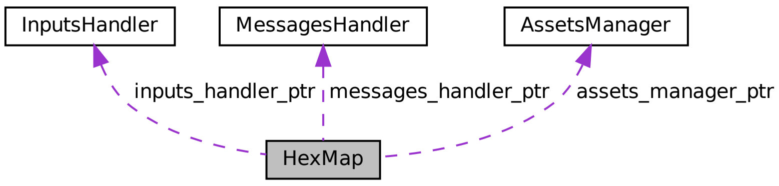
- header/ContextMenu/ContextMenu.h
- source/ContextMenu/ContextMenu.cpp

3.3 HexMap Class Reference

A class which defines a hex map of hex tiles.

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

Collaboration diagram for HexMap:



Public Member Functions

- HexMap (int, AssetsManager ∗, InputsHandler ∗, MessagesHandler ∗, sf::RenderWindow ∗)

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

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

    Method to format and send a tile message on certain events.
- void process (void)

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

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

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

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

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

    Destructor for the HexMap class.

Public Attributes

- int n_layers

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

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

    The current frame of this object.
- double position_x

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

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

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

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

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

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

    A position-indexed, nested map of hex tiles.

Private Member Functions

- void __setUpGlassScreen (void)

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

  Helper method to lay the hex tiles down to generate the game world.
- std::vector< double > __getNoise (int, int=128)

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

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

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

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

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

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

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

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

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

  Helper method to get pointer to selected tile.

Private Attributes

- unsigned long long int address_int

  An int representation of the memory address of this object.
- std::string address_string

  A string representation of the hex address of this object.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- InputsHandler ∗ inputs_handler_ptr

  A pointer to the inputs handler.
- MessagesHandler ∗ messages_handler_ptr

  A pointer to the messages handler.
- sf::RenderWindow ∗ render_window_ptr

  A pointer to the render window.

3.3.1 Detailed Description

A class which defines a hex map of hex tiles.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 HexMap()

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

Constructor for the HexMap class.

Parameters

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

```
{
    //  1. set attributes
    this->address_int = (unsigned long long int)this;

    std::stringstream ss;
    ss « std::hex « this;
    this->address_string = ss.str();

    this->assets_manager_ptr = assets_manager_ptr;
    this->inputs_handler_ptr = inputs_handler_ptr;
    this->messages_handler_ptr = messages_handler_ptr;
    this->render_window_ptr = render_window_ptr;

    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->messages_handler_ptr->addChannel(MESSAGE_CHANNEL_TILE);

    std::cout « "HexMap constructed at " « this « " (" « this->address_int
        « ")" « std::endl;

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

3.3.2.2 ∼HexMap()

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

Destructor for the HexMap class.

```
{
    this->clear();

    std::cout « "HexMap at " « this « " (" « this->address_int
        « ") destroyed" « std::endl;

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

3.3.3 Member Function Documentation

3.3.3.1 __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();

    //  3. procedurally generate types
    this->__procedurallyGenerateTileTypes();

    //  4. procedurally generate resources
    this->__procedurallyGenerateTileResources();

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

3.3.3.2 __enforceOceanContinuity()

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

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

```
{
    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() */
```

3.3.3.3 __getMajorityTileType()

```
TileType HexMap::__getMajorityTileType (
              HexTile * hex_ptr )  [private]
```

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

Parameters

hex_ptr	Pointer to the given tile.

Returns

The majority tile type of the tile and its neighbours. If no clear majority type, then the type of the given tile is simply returned.

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

3.3.3.4 __getNeighboursVector()

```
std::vector< HexTile * > HexMap::__getNeighboursVector (
            HexTile * hex_ptr )  [private]
```

Helper method to assemble a vector pointers to all neighbours of the given tile.

Parameters

hex_ptr	A pointer to the given tile.

Returns

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

```
{
    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() */
```

3.3.3.5 __getNoise()

```
std::vector< double > HexMap::__getNoise (
            int n_elements,
            int n_components = 128 )  [private]
```

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

Parameters

n_elements	The number of elements in the generated noise vector.
n_components	The number of components to use in the random cosine series. Defaults to 64.

Returns

A vector of noise, with values mapped to the closed interval [0, 1].

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

3.3.3.6 __getSelectedTile()

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

Helper method to get pointer to selected tile.

Returns

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

```
{
    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() */
```

3.3.3.7 __getValidMapIndexPositions()

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

Helper method to translate given position into valid index position for a.

Parameters

potential↵ _x	The potential x position of the tile.
potential↵ _y	The potential y position of the tile.

Returns

A vector of positions, either valid for indexing into the hex map, or sentinel values (-1) if invalid.

```
{
    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() */
```

3.3.3.8 __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() */
```

3.3.3.9 __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->assets_manager_ptr,
        this->inputs_handler_ptr,
        this->messages_handler_ptr,
        this->render_window_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->assets_manager_ptr,
            this->inputs_handler_ptr,
            this->messages_handler_ptr,
            this->render_window_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->assets_manager_ptr,
            this->inputs_handler_ptr,
            this->messages_handler_ptr,
            this->render_window_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->assets_manager_ptr,
            this->inputs_handler_ptr,
            this->messages_handler_ptr,
            this->render_window_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->assets_manager_ptr,
                this->inputs_handler_ptr,
                this->messages_handler_ptr,
                this->render_window_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->assets_manager_ptr,
            this->inputs_handler_ptr,
            this->messages_handler_ptr,
            this->render_window_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->assets_manager_ptr,
                this->inputs_handler_ptr,
                this->messages_handler_ptr,
                this->render_window_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() */
```

3.3.3.10 __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() */
```

3.3.3.11 __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();

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

3.3.3.12 __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(40, 40, 40, 40));

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

3.3.3.13 __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() */
```

3.3.3.14 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() */
```

3.3.3.15 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() */
```

3.3.3.16 draw()

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

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

```
{
    //  1. draw all tiles in order
    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->draw();
        }
    }

    //  2. redraw selected tile
    HexTile* selected_tile_ptr = this->__getSelectedTile();
    if (selected_tile_ptr != NULL) {
        selected_tile_ptr->draw();
    }

    //  3. draw glass screen
    this->render_window_ptr->draw(this->glass_screen);

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

3.3.3.17 process()

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

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

```
{
    //  1. process tiles
    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->process();
        }
    }

    //  2. handle inputs
    if (inputs_handler_ptr->mouse_left_click) {
        HexTile* selected_hex_ptr = __getSelectedTile();

        if (selected_hex_ptr != NULL) {
            selected_hex_ptr->sendMessage();
        }
        else {
            this->sendMessage();
        }
    }

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

3.3.3.18 reroll()

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

Method to re-roll the hex map.

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

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

3.3.3.19 sendMessage()

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

Method to format and send a tile message on certain events.

```
{
    //  1. check if last message sent was dummy (if so, do nothing)
    if (not this->messages_handler_ptr->isEmpty(MESSAGE_CHANNEL_TILE)) {
        Message message = this->messages_handler_ptr->receiveMessage(
            MESSAGE_CHANNEL_TILE
        );

        if (message.subject == "DUMMY") {
            return;
        }
    }

    //  2. format message header
    Message dummy_message;

    dummy_message.sender_name = "HexMap";
    dummy_message.sender_address = this->address_int;
    dummy_message.subject = "DUMMY";
    dummy_message.channel = MESSAGE_CHANNEL_TILE;

    //  3. send message
    this->messages_handler_ptr->sendMessage(dummy_message);

    std::cout « "HexMap at " « this « " sent a message" « std::endl;

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

3.3.3.20 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() */
```

3.3.4 Member Data Documentation

3.3.4.1 address_int

```
unsigned long long int HexMap::address_int  [private]
```

An int representation of the memory address of this object.

3.3.4.2 address_string

```
std::string HexMap::address_string  [private]
```

A string representation of the hex address of this object.

3.3.4.3 assets_manager_ptr

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

A pointer to the assets manager.

3.3.4.4 border_tiles_vec

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

A vector of pointers to the border tiles.

3.3.4.5 frame

```
int HexMap::frame
```

The current frame of this object.

3.3.4.6 glass_screen

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

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

3.3.4.7 hex_map

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

A position-indexed, nested map of hex tiles.

3.3.4.8 inputs_handler_ptr

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

A pointer to the inputs handler.

3.3.4.9 messages_handler_ptr

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

A pointer to the messages handler.

3.3.4.10 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

3.3.4.11 n_tiles

```
int HexMap::n_tiles
```

The number of tiles in the hex map.

3.3.4.12 position_x

```
double HexMap::position_x
```

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

3.3.4.13 position_y

```
double HexMap::position_y
```

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

3.3.4.14 render_window_ptr

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

A pointer to the render window.

3.3.4.15 tile_position_x_vec

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

A vector of tile x positions.

3.3.4.16 tile_position_y_vec

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

A vector of tile y position.

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

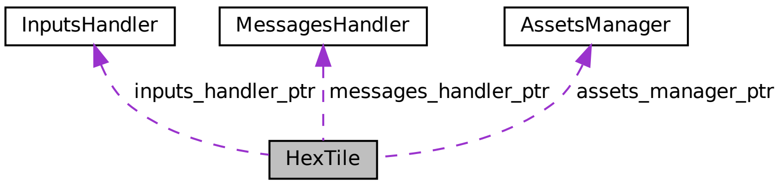
- header/HexMap/HexMap.h
- source/HexMap/HexMap.cpp

3.4 HexTile Class Reference

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

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

Collaboration diagram for HexTile:



Public Member Functions

- HexTile (double, double, AssetsManager ∗, InputsHandler ∗, MessagesHandler ∗, sf::RenderWindow ∗)

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

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

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

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

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

  Method to toggle the tile resource overlay.
- void assess (void)

  Method to assess the tile's resource.
- void sendMessage (void)

  Method to format and send a tile message on certain events.
- void process (void)

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

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

  Destructor for the HexTile class.

Public Attributes

- TileType tile_type
- TileResource tile_resource
- bool show_node

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

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

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

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

  The current frame of this object.
- double position_x

  The x position of the tile.
- double position_y

  The y position of the tile.
- double major_radius

  The radius of the smallest bounding circle.
- double minor_radius

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

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

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

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

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

  A text representation of the resource.

Private Member Functions

- void __setUpNodeSprite (void)

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

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

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

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

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

  Helper method to determine if tile was clicked on.
- std::string __assembleMessageStringPayload (void)

  Helpe method to assemble string payload of tile message.

Private Attributes

- unsigned long long int address_int

  An int representation of the memory address of this object.
- std::string address_string

  A string representation of the memory address of this object.
- AssetsManager ∗ assets_manager_ptr

  A pointer to the assets manager.
- InputsHandler ∗ inputs_handler_ptr

  A pointer to the inputs handler.
- MessagesHandler ∗ messages_handler_ptr

  A pointer to the messages handler.
- sf::RenderWindow ∗ render_window_ptr

  A pointer to the render window.

3.4.1 Detailed Description

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

3.4.2 Constructor & Destructor Documentation

3.4.2.1 HexTile()

```
HexTile::HexTile (
             double position_x,
             double position_y,
             AssetsManager ∗ assets_manager_ptr,
             InputsHandler ∗ inputs_handler_ptr,
             MessagesHandler ∗ messages_handler_ptr,
             sf::RenderWindow ∗ render_window_ptr )
```

Constructor for the HexTile class.

Ref: Wikipedia [2023]

Parameters

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

```
{
    //  1. set attributes
    this->address_int = (unsigned long long int)this;

    std::stringstream ss;
    ss « std::hex « this;
    this->address_string = ss.str();

    this->assets_manager_ptr = assets_manager_ptr;
    this->inputs_handler_ptr = inputs_handler_ptr;
    this->messages_handler_ptr = messages_handler_ptr;
    this->render_window_ptr = render_window_ptr;

    this->show_node = false;
    this->show_resource = false;
    this->resource_assessed = false;
    this->is_selected = false;

    this->frame = 0;

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

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

    //  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 forest and average)
    this->setTileType(TileType :: FOREST);
    this->setTileResource(TileResource :: AVERAGE);

    std::cout « "HexTile constructed at " « this « " (" « this->address_int
        « ")" « std::endl;

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

3.4.2.2 ∼HexTile()

```
HexTile::∼HexTile (
              void )
```

Destructor for the HexTile class.

```
{
    std::cout « "HexTile at " « this « " (" « this->address_int
        « ") destroyed" « std::endl;

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

3.4.3 Member Function Documentation

3.4.3.1 __assembleMessageStringPayload()

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

Helpe method to assemble string payload of tile message.

Returns

String payload of tile message.

```
{
    //                   32 char x 17 line console "--------------------------------\n";
    std::string payload                          = "  **** TILE INFO/OPTIONS ****   \n";
    payload                                     += "                                \n";


    payload                                     += "TYPE:     ";

    switch (this->tile_type) {
        case (TileType :: FOREST): {
            payload +=                                        "FOREST               \n";

            break;
        }

        case (TileType :: LAKE): {
            payload +=                                        "LAKE                 \n";

            break;
        }

        case (TileType :: MOUNTAINS): {
            payload +=                                        "MOUNTAINS            \n";

            break;
        }

        case (TileType :: OCEAN): {
            payload +=                                        "OCEAN                \n";

            break;
        }

        case (TileType :: PLAINS): {
            payload +=                                        "PLAINS               \n";

            break;
        }

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

            break;
        }
    }


    payload                                     += "RESOURCE: ";

    if (not this->resource_assessed) {
        payload +=                                            "[A]: ASSESS RESOURCE \n";
    }

    else {
        switch (this->tile_resource) {
            //...

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

                break;
            }
        }
    }


    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "                                \n";
    payload                                     += "[ESC]: MENU                     ";

    return payload;
}   /* __assembleMessageStringPayload() */
```

3.4.3.2 __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() */
```

3.4.3.3 __setResourceText()

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

Helper method to set up resource text.

```
{
    this->resource_text.setFont(*(assets_manager_ptr->getFont("DroidSansMono")));

    switch (this->tile_resource) {
        case (TileResource :: POOR): {
            this->resource_text.setString("-2");

            break;
        }

        case (TileResource :: BELOW_AVERAGE): {
            this->resource_text.setString("-1");

            break;
        }

        case (TileResource :: AVERAGE): {
            this->resource_text.setString("0");

            break;
        }

        case (TileResource :: ABOVE_AVERAGE): {
            this->resource_text.setString("+1");

            break;
        }

        case (TileResource :: GOOD): {
            this->resource_text.setString("+2");

            break;
        }

        default: {
            this->resource_text.setString("?");

```

```
                break;
            }
        }

        if (not this->resource_assessed) {
            this->resource_text.setString("?");
        }

        this->resource_text.setCharacterSize(16);

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

        this->resource_text.setFillColor(sf::Color(0, 0, 0, 255));

        this->resource_text.setPosition(
            this->position_x,
            this->position_y - 4
        );

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

3.4.3.4 __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() */
```

3.4.3.5 __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(sf::Color(175, 175, 175, 175));

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

3.4.3.6 __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() */
```

3.4.3.7 __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() */
```

3.4.3.8 assess()

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

Method to assess the tile's resource.

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

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

3.4.3.9 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 resource
    if (this->show_resource) {
        this->render_window_ptr->draw(this->resource_chip_sprite);
        this->render_window_ptr->draw(this->resource_text);
    }

    //  4. draw selection outline
    if (this->is_selected) {
        sf::Color outline_colour = this->select_outline_sprite.getOutlineColor();

        outline_colour.a =
            255 * pow(cos((M_PI * this->frame) / (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() */
```

3.4.3.10 process()

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

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

```
{
    //  1. handle inputs
    if (this->inputs_handler_ptr->key_pressed_once_vec[sf::Keyboard::Escape]) {
        this->is_selected = false;
    }

    if (inputs_handler_ptr->mouse_left_click) {
        this->is_selected = false;

        if (this->__isClicked()) {
            std::cout « "Tile (" « this->position_x « ", " « this->position_y «
                ") was selected" « std::endl;

            this->is_selected = true;
        }
    }

    if (inputs_handler_ptr->mouse_right_click) {
        this->is_selected = false;
    }

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

3.4.3.11 sendMessage()

```
void HexTile::sendMessage (
            void )
```

Method to format and send a tile message on certain events.

```
{
    //  1. format message header
    Message selected_message;

    selected_message.sender_name = "HexTile";
    selected_message.sender_address = this->address_int;
    selected_message.subject = "Tile selected";
    selected_message.channel = MESSAGE_CHANNEL_TILE;

    //  2. assemble message payload
    selected_message.string_payload = this->__assembleMessageStringPayload();

    //  3. send message
    this->messages_handler_ptr->sendMessage(selected_message);

    std::cout « "HexTile at " « this « " sent a message" « std::endl;

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

3.4.3.12 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) */
```

3.4.3.13 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) */
```

3.4.3.14 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) */
```

3.4.3.15 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) */
```

3.4.3.16 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() */
```

3.4.4 Member Data Documentation

3.4.4.1 address_int

`unsigned long long int HexTile::address_int  [private]`

An int representation of the memory address of this object.

3.4.4.2 address_string

`std::string HexTile::address_string  [private]`

A string representation of the memory address of this object.

3.4.4.3 assets_manager_ptr

`AssetsManager* HexTile::assets_manager_ptr  [private]`

A pointer to the assets manager.

3.4.4.4 frame

`int HexTile::frame`

The current frame of this object.

3.4.4.5 inputs_handler_ptr

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

A pointer to the inputs handler.

3.4.4.6 is_selected

```
bool HexTile::is_selected
```

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

3.4.4.7 major_radius

```
double HexTile::major_radius
```

The radius of the smallest bounding circle.

3.4.4.8 messages_handler_ptr

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

A pointer to the messages handler.

3.4.4.9 minor_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

3.4.4.10 node_sprite

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

A circle shape to mark the tile node.

3.4.4.11 position_x

```
double HexTile::position_x
```

The x position of the tile.

3.4.4.12 position_y

```
double HexTile::position_y
```

The y position of the tile.

3.4.4.13 render_window_ptr

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

A pointer to the render window.

3.4.4.14 resource_assessed

```
bool HexTile::resource_assessed
```

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

3.4.4.15 resource_chip_sprite

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

A circle shape which represents a resource chip.

3.4.4.16 resource_text

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

A text representation of the resource.

3.4.4.17 select_outline_sprite

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

A convex shape which outlines the tile when selected.

3.4.4.18 show_node

```
bool HexTile::show_node
```

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

3.4.4.19 show_resource

```
bool HexTile::show_resource
```

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

3.4.4.20 tile_resource

```
TileResource HexTile::tile_resource
```

3.4.4.21 tile_sprite

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

A convex shape which represents the tile.

3.4.4.22 tile_type

```
TileType HexTile::tile_type
```

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

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

3.5 InputsHandler Class Reference

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

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

Public Member Functions

- InputsHandler (void)

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

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

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

  Destructor for the InputsHandler class.

Public Attributes

- bool any_key_once

  A boolean which indicates if any key has just been pressed/clicked once.
- bool mouse_left_click

  A boolean which indicates if the mouse left button has been clicked.
- bool mouse_right_click

  A boolean which indicates if the mouse right button has been clicked.
- std::vector< bool > key_pressed_once_vec

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

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

  A map from key codes to corresponding string representations.

Private Member Functions

- void __constructKeyCodeMap (void)

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

3.5.1 Detailed Description

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

3.5.2 Constructor & Destructor Documentation

3.5.2.1 InputsHandler()

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

Constructor for the InputsHandler class.

```
{
    this->any_key_once = false;

    this->mouse_left_click = false;
    this->mouse_right_click = false;

    this->key_pressed_once_vec.resize(sf::Keyboard::KeyCount, false);
    this->key_press_vec.resize(sf::Keyboard::KeyCount, false);

    this->__constructKeyCodeMap();

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

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

3.5.2.2 ∼InputsHandler()

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

Destructor for the InputsHandler class.

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

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

3.5.3 Member Function Documentation

3.5.3.1 __constructKeyCodeMap()

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

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

```
{
    //  1. unknown keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Unknown, "Unknown")
    );


    //  2. alpha keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::A, "A")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::B, "B")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::C, "C")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::D, "D")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::E, "E")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F, "F")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::G, "G")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::H, "H")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::I, "I")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::J, "J")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::K, "K")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::L, "L")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::M, "M")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::N, "N")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::O, "O")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::P, "P")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Q, "Q")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::R, "R")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::S, "S")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::T, "T")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::U, "U")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::V, "V")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::W, "W")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::X, "X")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Y, "Y")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Z, "Z")
    );


    //  3. numeric keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num0, "0")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num1, "1")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num2, "2")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num3, "3")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num4, "4")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num5, "5")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num6, "6")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num7, "7")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num8, "8")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num9, "9")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad0, "0")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad1, "1")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad2, "2")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad3, "3")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad4, "4")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad5, "5")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad6, "6")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad7, "7")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad8, "8")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad9, "9")
    );


    //  4. direction keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Left, "Left")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Right, "Right")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Up, "Up")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Down, "Down")
    );


    //  5. function keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F1, "F1")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F2, "F2")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F3, "F3")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F4, "F4")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F5, "F5")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F6, "F6")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F7, "F7")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F8, "F8")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F9, "F9")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F10, "F10")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F11, "F11")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F12, "F12")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F13, "F13")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F14, "F14")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F15, "F15")
    );


    //  6. other keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Escape, "Escape")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LControl, "LCtrl")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LShift, "LShift")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LAlt, "LAlt")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LSystem, "LSystem")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RControl, "RCtrl")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RShift, "RShift")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RAlt, "RAlt")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RSystem, "RSystem")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Menu, "Menu")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LBracket, "LBracket")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RBracket, "RBracket")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Semicolon, "Semicolon")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Comma, "Comma")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Period, "Period")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Quote, "Quote")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Slash, "Slash")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backslash, "Backslash")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tilde, "Tilde")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Equal, "Equal")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Hyphen, "Hyphen")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Space, "Space")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Enter, "Enter")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backspace, "Backspace")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tab, "Tab")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageUp, "PageUp")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageDown, "PageDown")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::End, "End")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Home, "Home")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Insert, "Insert")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Delete, "Delete")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Add, "Add")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Subtract, "Subtract")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Multiply, "Multiply")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Divide, "Divide")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Pause, "Pause")
    );

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

3.5.3.2 printKeysPressed()

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

Method to print out which keys are currently pressed.

```
{
    std::string print_str = "";

    for (size_t i = 0; i < this->key_press_vec.size(); i++) {
        if (this->key_press_vec[i]) {
            print_str += this->key_code_map[sf::Keyboard::Key(i)];
            print_str += ", ";
        }
    }

    if (not print_str.empty()) {
        std::cout « "Keys pressed: " « print_str « std::endl;
    }

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

3.5.3.3 process()

```
void InputsHandler::process (
              sf::Event * event_ptr )
{
    //  1. update state of key press vectors
    switch (event_ptr->type) {
        case (sf::Event::KeyPressed): {
            if (not this->key_press_vec[event_ptr->key.code]) {
                this->key_pressed_once_vec[event_ptr->key.code] = true;
            }

            this->key_press_vec[event_ptr->key.code] = true;

            if (not this->any_key_once) {
                this->any_key_once = true;
            }

            break;
        }

        case (sf::Event::KeyReleased): {
            this->any_key_once = false;

            this->key_pressed_once_vec[event_ptr->key.code] = false;
            this->key_press_vec[event_ptr->key.code] = false;

            break;
        }

        case (sf::Event::MouseButtonPressed): {
            if (not this->any_key_once) {
                this->any_key_once = true;
            }

            if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
            {
                this->mouse_left_click = true;

                std::cout « "left click        " « std::endl;
            }

            if (sf::Mouse::isButtonPressed(sf::Mouse::Right))
            {
                this->mouse_right_click = true;

                std::cout « "right click        " « std::endl;
            }

            break;
        }

        case (sf::Event::MouseButtonReleased): {
            this->any_key_once = false;

            this->mouse_left_click = false;
            this->mouse_right_click = false;

            break;
        }

        default: {
            //  do nothing!

            break;
        }
    }

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

3.5.3.4 reset()

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

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

```
{
    this->any_key_once = false;

    this->mouse_left_click = false;
    this->mouse_right_click = false;

    for (size_t i = 0; i < this->key_press_vec.size(); i++) {
        this->key_pressed_once_vec[i] = false;
    }

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

3.5.4 Member Data Documentation

3.5.4.1 any_key_once

`bool InputsHandler::any_key_once`

A boolean which indicates if any key has just been pressed/clicked once.

3.5.4.2 key_code_map

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

A map from key codes to corresponding string representations.

3.5.4.3 key_press_vec

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

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

3.5.4.4 key_pressed_once_vec

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

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

3.5.4.5 mouse_left_click

```
bool InputsHandler::mouse_left_click
```

A boolean which indicates if the mouse left button has been clicked.

3.5.4.6 mouse_right_click

```
bool InputsHandler::mouse_right_click
```

A boolean which indicates if the mouse right button has been clicked.

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

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

3.6 Message Struct Reference

A structure which defines a standard message format.

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

Public Attributes

- std::string sender_name = ""

    A string representation of the sender's class.
- unsigned long long int sender_address = 0
- std::string subject = ""

    An int representation of the sender's memory address.
- std::string channel = ""

    A string identifying the appropriate channel for this message.
- std::vector< bool > bool_payload_vec = {}

    A vector <bool> payload.
- std::vector< int > int_payload_vec = {}

    A vector <int> payload.
- std::vector< double > double_payload_vec = {}

    A vector <double> payload.
- std::string string_payload = ""

    A string payload.

3.6.1 Detailed Description

A structure which defines a standard message format.

3.6.2 Member Data Documentation

3.6.2.1 bool_payload_vec

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

A vector <bool> payload.

3.6.2.2 channel

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

A string identifying the appropriate channel for this message.

3.6.2.3 double_payload_vec

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

A vector <double> payload.

3.6.2.4 int_payload_vec

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

A vector <int> payload.

3.6.2.5 sender_address

```
unsigned long long int Message::sender_address = 0
```

3.6.2.6 sender_name

```
std::string Message::sender_name = ""
```

A string representation of the sender's class.

3.6.2.7 string_payload

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

A string payload.

3.6.2.8 subject

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

An int representation of the sender's memory address.

A string describing the message subject.

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

- header/ESC_core/MessagesHandler.h

3.7 MessagesHandler Class Reference

A class which handles message traffic between game objects.

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

Public Member Functions

- MessagesHandler (void)

  Constructor for the MessagesHandler class.
- void addChannel (std::string)

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

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

  Method to send a message to the message map.
- bool isEmpty (std::string)

  Method to check if channel is empty.
- Message receiveMessage (std::string)

  Method to receive the latest message in the given channel.
- void process (void)

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

  Method to clear the MessagesHandler.
- ∼MessagesHandler (void)

  Destructor for the MessagesHandler 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 handles message traffic between game objects.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 MessagesHandler()

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

Constructor for the MessagesHandler class.

```
{
    //...

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

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

3.7.2.2 ∼MessagesHandler()

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

Destructor for the MessagesHandler class.

```
{
    this->clear();

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

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

3.7.3 Member Function Documentation

3.7.3.1 addChannel()

```
void MessagesHandler::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  MessagesHandler::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] = {};

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

3.7.3.2 clear()

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

Method to clear the MessagesHandler.

```
{

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

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

3.7.3.3 isEmpty()

```
bool MessagesHandler::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  MessagesHandler::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() */
```

3.7.3.4 process()

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

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

```
{
    //...

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

3.7.3.5 receiveMessage()

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

Method to receive the latest message in the given channel.

Parameters

channel	The key for the message channel being received from.

Returns

The latest message in the given channel.

```
{
    //  1. check if channel is in map (if not, throw error)
    if (this->message_map.count(channel) <= 0) {
        std::string error_str = "ERROR  MessagesHandler::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  MessagesHandler::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].back();

    return message;
}   /* receiveMessage() */
```

3.7.3.6 removeChannel()

```
void MessagesHandler::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  MessagesHandler::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);

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

3.7.3.7 sendMessage()

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

Method to send a message to the message map.

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  MessagesHandler::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> > MessagesHandler::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/MessagesHandler.h
- source/ESC_core/MessagesHandler.cpp

Chapter 4

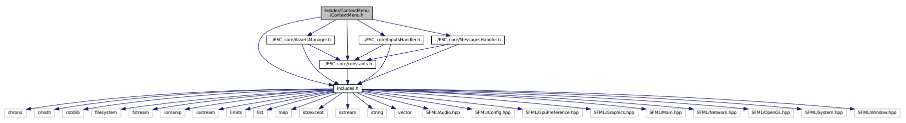
File Documentation

4.1 header/ContextMenu/ContextMenu.h File Reference

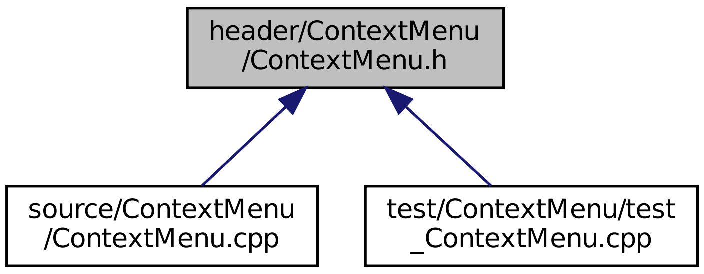
Header file for the ContextMenu class.

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

Include dependency graph for ContextMenu.h:



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


header/ContextMenu /ContextMenu.h
source/ContextMenu /ContextMenu.cpp
test/ContextMenu/test _ContextMenu.cpp

Classes

- class ContextMenu

    A class which defines a context menu for the game.

Enumerations

- enum ConsoleState {
  NONE , READY , MENU , TILE ,
  N_CONSOLE_STATES }

    An enumeration of the different console screen states.

4.1.1 Detailed Description

Header file for the ContextMenu class.

4.1.2 Enumeration Type Documentation

4.1.2.1 ConsoleState

```
enum ConsoleState
```

An enumeration of the different console screen states.

Enumerator

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

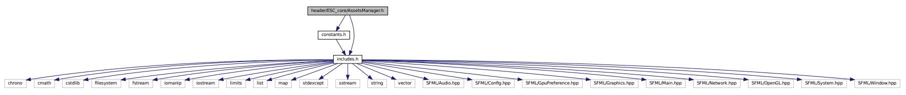
```
                 {
    NONE,
    READY,
    MENU,
    TILE,
    N_CONSOLE_STATES
};
```

4.2 header/ESC_core/AssetsManager.h File Reference

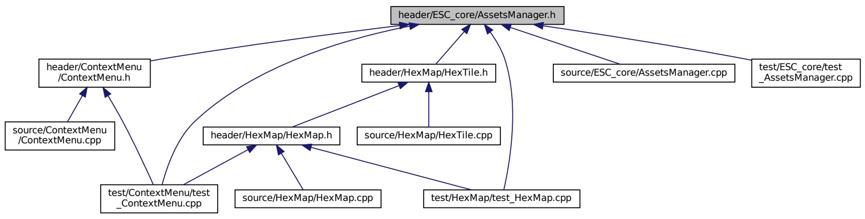
Header file for the AssetsManager class.

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

Include dependency graph for AssetsManager.h:



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


header/ESC_core/AssetsManager.h
header/ContextMenu /ContextMenu.h
header/HexMap/HexTile.h
source/ESC_core/AssetsManager.cpp
test/ESC_core/test _AssetsManager.cpp
source/ContextMenu /ContextMenu.cpp
header/HexMap/HexMap.h
source/HexMap/HexTile.cpp
test/ContextMenu/test _ContextMenu.cpp
source/HexMap/HexMap.cpp
test/HexMap/test_HexMap.cpp

Classes

- class AssetsManager

  A class which manages visual and sound assets.

4.2.1 Detailed Description

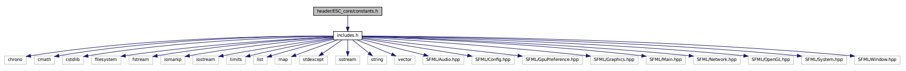
Header file for the AssetsManager class.

4.3 header/ESC_core/constants.h File Reference

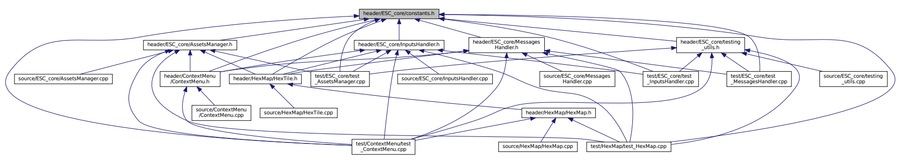
Header file for various constants.

```
#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/InputsHandler.h
header/ESC_core/Messages Handler.h
header/ESC_core/testing _utils.h
source/ESC_core/AssetsManager.cpp
header/ContextMenu /ContextMenu.h
header/HexMap/HexTile.h
test/ESC_core/test _AssetsManager.cpp
source/ESC_core/InputsHandler.cpp
source/ESC_core/Messages Handler.cpp
test/ESC_core/test _InputsHandler.cpp
test/ESC_core/test _MessagesHandler.cpp
source/ESC_core/testing _utils.cpp
source/ContextMenu /ContextMenu.cpp
source/HexMap/HexTile.cpp
header/HexMap/HexMap.h
test/ContextMenu/test _ContextMenu.cpp
source/HexMap/HexMap.cpp
test/HexMap/test_HexMap.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 MENU_FRAME_GREY (185, 187, 182)

  The base colour of the context menu frame.
- const sf::Color MONOCHROME_SCREEN_BACKGROUND (40, 40, 40)

  The base colour of old monochrome screens.
- const sf::Color VISUAL_SCREEN_FRAME_GREY (151, 151, 143)

  The base colour of the framing of the visual screen.
- const sf::Color MONOCHROME_TEXT_GREEN (0, 255, 102)

  The base colour of old monochrome text (green).
- const sf::Color MONOCHROME_TEXT_AMBER (255, 176, 0)

  The base colour of old monochrome text (amber).
- const sf::Color MONOCHROME_TEXT_RED (255, 44, 0)

  The base colour of old monochrome text (red).

Variables

- const double FLOAT_TOLERANCE = 1e-6

  Tolerance for floating point equality tests.
- const int FRAMES_PER_SECOND = 60

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

  Target seconds per frame (just reciprocal of target frames per second).
- const int GAME_WIDTH = 1200

  Width of the game space.
- const int GAME_HEIGHT = 800

  Height of the game space.
- const std::vector< double > TILE_TYPE_CUMULATIVE_PROBABILITIES

  Cumulative probabilities for each tile type (to support procedural generation).
- const std::vector< double > TILE_RESOURCE_CUMULATIVE_PROBABILITIES

  Cumulative probabilities for each tile resource (to support procedural generation).
- const std::string MESSAGE_CHANNEL_TILE = "MESSAGE_CHANNEL_TILE"

  A channel for tile messages (for indexing into message map).

4.3.1 Detailed Description

Header file for various constants.

4.3.2 Function Documentation

4.3.2.1 FOREST_GREEN()

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

The base colour of a forest tile.

4.3.2.2 LAKE_BLUE()

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

The base colour of a lake (water) tile.

4.3.2.3 MENU_FRAME_GREY()

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

The base colour of the context menu frame.

4.3.2.4 MONOCHROME_SCREEN_BACKGROUND()

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

The base colour of old monochrome screens.

4.3.2.5 MONOCHROME_TEXT_AMBER()

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

The base colour of old monochrome text (amber).

4.3.2.6 MONOCHROME_TEXT_GREEN()

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

The base colour of old monochrome text (green).

4.3.2.7 MONOCHROME_TEXT_RED()

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

The base colour of old monochrome text (red).

4.3.2.8 MOUNTAINS_GREY()

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

The base colour of a mountains tile.

4.3.2.9 OCEAN_BLUE()

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

The base colour of an ocean (water) tile.

4.3.2.10 PLAINS_YELLOW()

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

The base colour of a plains tile.

4.3.2.11 VISUAL_SCREEN_FRAME_GREY()

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

The base colour of the framing of the visual screen.

4.3.3 Variable Documentation

4.3.3.1 FLOAT_TOLERANCE

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

Tolerance for floating point equality tests.

4.3.3.2 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

4.3.3.3 GAME_HEIGHT

```
const int GAME_HEIGHT = 800
```

Height of the game space.

4.3.3.4 GAME_WIDTH

```
const int GAME_WIDTH = 1200
```

Width of the game space.

4.3.3.5 MESSAGE_CHANNEL_TILE

```
const std::string MESSAGE_CHANNEL_TILE = "MESSAGE_CHANNEL_TILE"
```

A channel for tile messages (for indexing into message map).

4.3.3.6 SECONDS_PER_FRAME

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

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

4.3.3.7 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).

4.3.3.8 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.

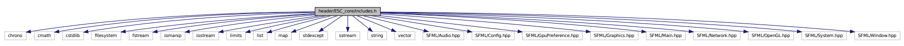
Ref: van Heesch. [2023]

4.5 header/ESC_core/includes.h File Reference

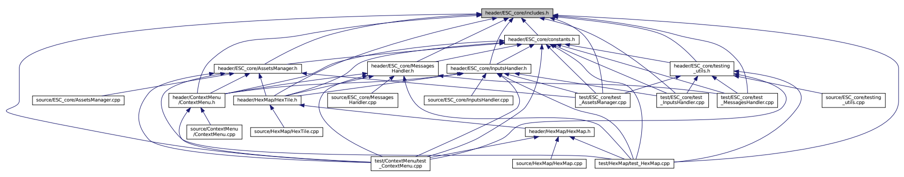
Header file for various includes.

```
#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/Messages Handler.h
header/ESC_core/InputsHandler.h
header/ESC_core/testing _utils.h
source/ESC_core/AssetsManager.cpp
header/ContextMenu /ContextMenu.h
header/HexMap/HexTile.h
source/ESC_core/Messages Handler.cpp
source/ESC_core/InputsHandler.cpp
test/ESC_core/test _AssetsManager.cpp
test/ESC_core/test _InputsHandler.cpp
test/ESC_core/test _MessagesHandler.cpp
source/ESC_core/testing _utils.cpp
source/ContextMenu /ContextMenu.cpp
source/HexMap/HexTile.cpp
header/HexMap/HexMap.h
test/ContextMenu/test _ContextMenu.cpp
source/HexMap/HexMap.cpp
test/HexMap/test_HexMap.cpp

4.5.1 Detailed Description

Header file for various includes.

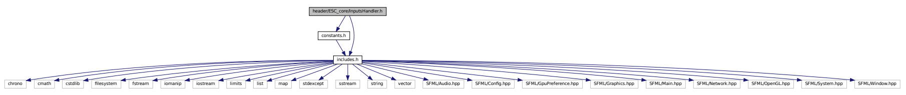
Ref: Gomila [2023]

4.6 header/ESC_core/InputsHandler.h File Reference

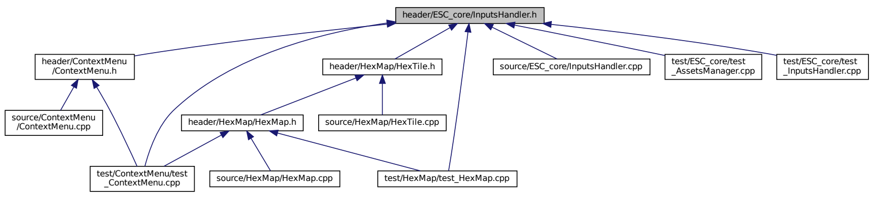
Header file for the InputsHandler class.

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

Include dependency graph for InputsHandler.h:



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


header/ESC_core/InputsHandler.h
header/ContextMenu /ContextMenu.h
header/HexMap/HexTile.h
source/ESC_core/InputsHandler.cpp
test/ESC_core/test _AssetsManager.cpp
test/ESC_core/test _InputsHandler.cpp
source/ContextMenu /ContextMenu.cpp
header/HexMap/HexMap.h
source/HexMap/HexTile.cpp
test/ContextMenu/test _ContextMenu.cpp
source/HexMap/HexMap.cpp
test/HexMap/test_HexMap.cpp

Classes

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

4.6.1 Detailed Description

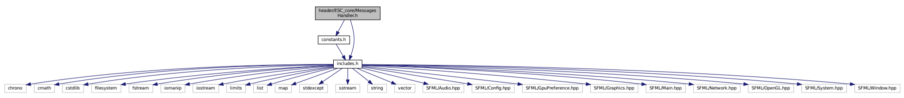
Header file for the InputsHandler class.

4.7 header/ESC_core/MessagesHandler.h File Reference

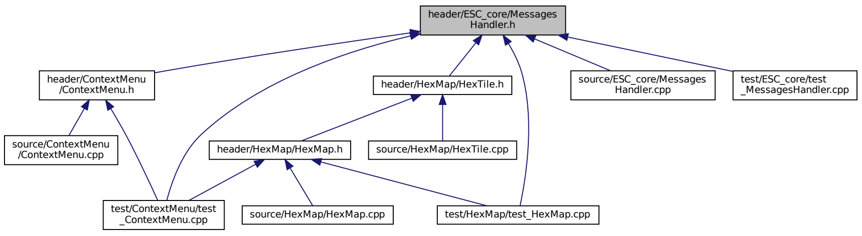
Header file for the MessagesHandler class.

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

Include dependency graph for MessagesHandler.h:



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



Classes

- struct Message

  A structure which defines a standard message format.
- class MessagesHandler

  A class which handles message traffic between game objects.

4.7.1 Detailed Description

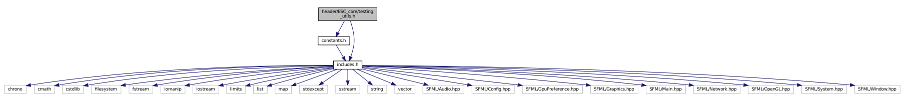
Header file for the MessagesHandler class.

4.8 header/ESC_core/testing_utils.h File Reference

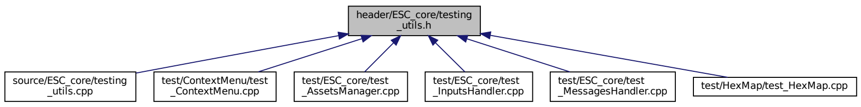
Header file for various testing utilities.

```
#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
test/ContextMenu/test_ContextMenu.cpp
test/ESC_core/test_AssetsManager.cpp
test/ESC_core/test_InputsHandler.cpp
test/ESC_core/test_MessagesHandler.cpp
test/HexMap/test_HexMap.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.

4.8.1 Detailed Description

Header file for various testing utilities.

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

4.8.2 Function Documentation

4.8.2.1 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() */
```

4.8.2.2 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() */
```

4.8.2.3 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() */
```

4.8.2.4 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() */
```

4.8.2.5 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() */
```

4.8.2.6 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() */
```

4.8.2.7 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() */
```

4.8.2.8 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() */
```

4.8.2.9 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() */
```

4.8.2.10 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__").

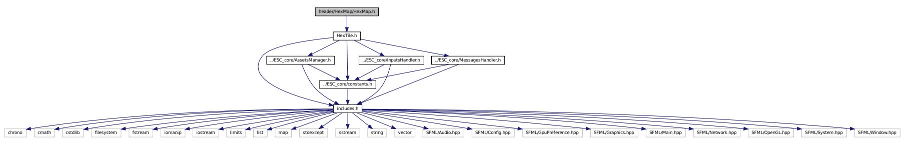
```
{
    if (statement) {
        return;
    }

    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() */
```

4.9 header/HexMap/HexMap.h File Reference

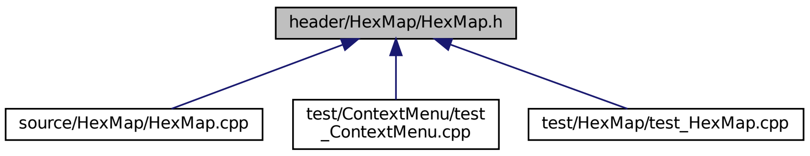
Header file for the HexMap class.

`#include "HexTile.h"`

Include dependency graph for HexMap.h:



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


header/HexMap/HexMap.h
source/HexMap/HexMap.cpp
test/ContextMenu/test_ContextMenu.cpp
test/HexMap/test_HexMap.cpp

Classes

- class HexMap

  A class which defines a hex map of hex tiles.

4.9.1 Detailed Description

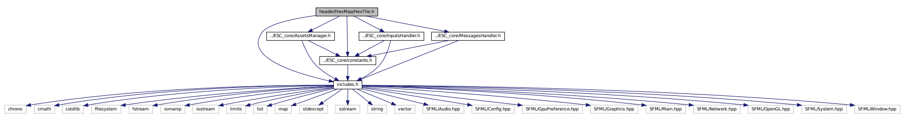
Header file for the HexMap class.

4.10 header/HexMap/HexTile.h File Reference

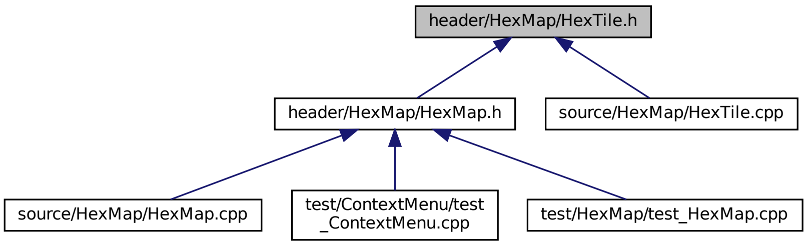
Header file for the HexTile class.

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

Include dependency graph for HexTile.h:



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


header/HexMap/HexTile.h
header/HexMap/HexMap.h
source/HexMap/HexTile.cpp
source/HexMap/HexMap.cpp
test/ContextMenu/test_ContextMenu.cpp
test/HexMap/test_HexMap.cpp

Classes

- class HexTile

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

Enumerations

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

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

  An enumeration of the different tile resource values.

4.10.1 Detailed Description

Header file for the 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
};
```

4.10.2.2 TileType

`enum TileType`

An enumeration of the different tile types.

Enumerator

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

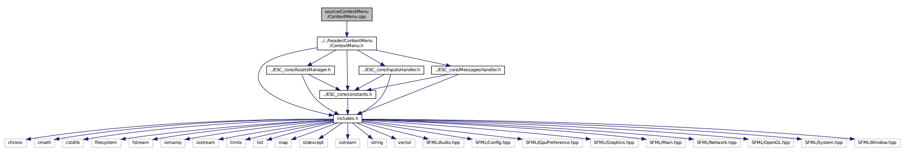
```
            {
    FOREST,
    LAKE,
    MOUNTAINS,
    OCEAN,
    PLAINS,
    N_TILE_TYPES
};
```

4.11 source/ContextMenu/ContextMenu.cpp File Reference

Implementation file for the ContextMenu class.

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

Include dependency graph for ContextMenu.cpp:



4.11.1 Detailed Description

Implementation file for the ContextMenu class.

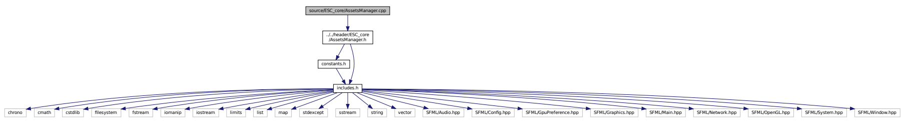
A class which defines a context menu for the game.

4.12 source/ESC_core/AssetsManager.cpp File Reference

Implementation file for the AssetsManager class.

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

Include dependency graph for AssetsManager.cpp:



4.12.1 Detailed Description

Implementation file for the AssetsManager class.

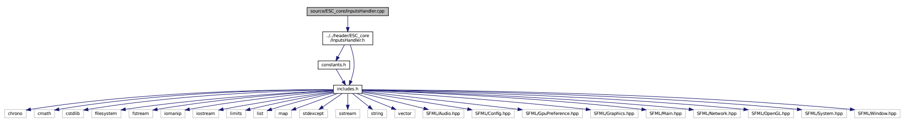
A class which manages visual and sound assets.

4.13 source/ESC_core/InputsHandler.cpp File Reference

Implementation file for the InputsHandler class.

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

Include dependency graph for InputsHandler.cpp:



4.13.1 Detailed Description

Implementation file for the InputsHandler class.

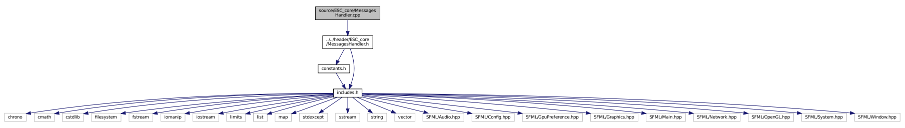
A class which handles inputs from peripherals (i.e., keyboard and mouse).

4.14 source/ESC_core/MessagesHandler.cpp File Reference

Implementation file for the MessagesHandler class.

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

Include dependency graph for MessagesHandler.cpp:



4.14.1 Detailed Description

Implementation file for the MessagesHandler class.

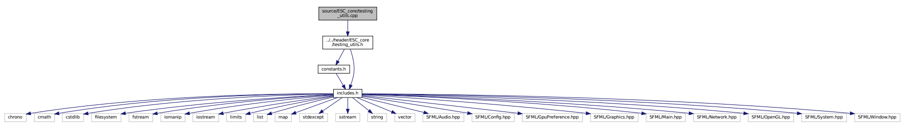
A class which handles message traffic between game objects.

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

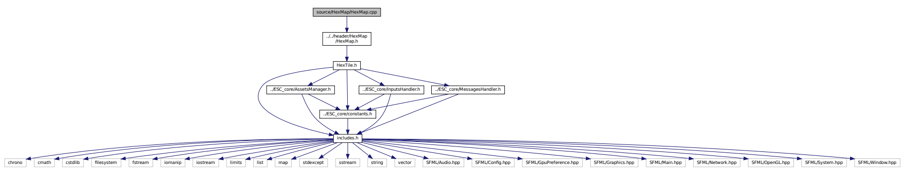
statement	The statement whose truth is to be tested ("1 == 0", for example).
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    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/HexMap/HexMap.cpp File Reference

Implementation file for the HexMap class.

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



4.16.1 Detailed Description

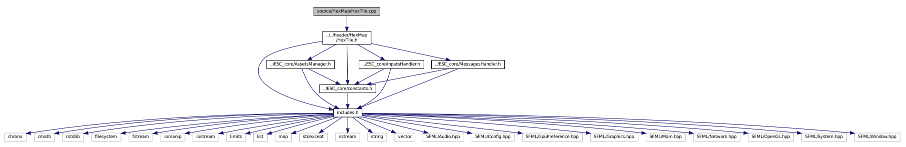
Implementation file for the HexMap class.

A class which defines a hex map of hex tiles.

4.17 source/HexMap/HexTile.cpp File Reference

Implementation file for the HexTile class.

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



4.17.1 Detailed Description

Implementation file for the HexTile class.

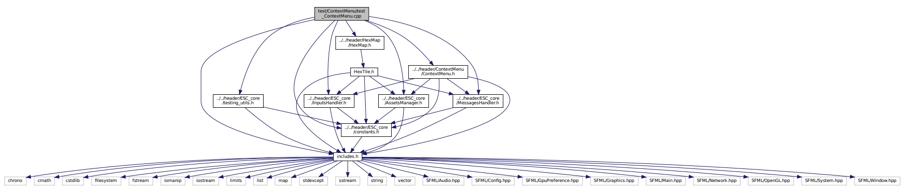
A class which defines a tile of a hex map.

4.18 test/ContextMenu/test_ContextMenu.cpp File Reference

Suite of tests for the ContextMenu class.

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

Include dependency graph for test_ContextMenu.cpp:


test/ContextMenu/test_ContextMenu.cpp
../../header/HexMap/HexMap.h
HexTile.h
../../header/ContextMenu/ContextMenu.h
../../header/ESC_core/testing_utils.h
../../header/ESC_core/InputsHandler.h
../../header/ESC_core/AssetsManager.h
../../header/ESC_core/MessagesHandler.h
../../header/ESC_core/constants.h
includes.h
chrono
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
list
map
stdexcept
sstream
string
vector
SFML/Audio.hpp
SFML/Config.hpp
SFML/GpuPreference.hpp
SFML/Graphics.hpp
SFML/Main.hpp
SFML/Network.hpp
SFML/OpenGL.hpp
SFML/System.hpp
SFML/Window.hpp

Functions

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

4.18.1 Detailed Description

Suite of tests for the ContextMenu class.

A suite of tests for the ContextMenu class.

4.18.2 Function Documentation

4.18.2.1 main()

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

```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting ContextMenu");
    std::cout « std::endl;

    srand(time(NULL));
    int n_dots = 8;


    try {
        //  1. construct, load/open some test assets
        AssetsManager assets_manager;
        InputsHandler inputs_handler;
        MessagesHandler messages_handler;

        assets_manager.loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
        assets_manager.loadFont("assets/fonts/Glass_TTY_VT220.ttf", "Glass_TTY_VT220");


        //  2. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(
            sf::VideoMode(GAME_WIDTH, GAME_HEIGHT),
            "Testing ContextMenu"
        );

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            1200,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            800,
            __FILE__,
            __LINE__
        );

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

        ContextMenu context_menu(
            &assets_manager,
            &inputs_handler,
            &messages_handler,
            &window
        );

        HexMap hex_map(
             6,
             &assets_manager,
             &inputs_handler,
             &messages_handler,
             &window
         );

         while (window.isOpen()) {
             time_since_run_s = clock.getElapsedTime().asSeconds();

             if (
                 time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
             ) {
                 while (window.pollEvent(event))
                 {
                     inputs_handler.process(&event);

                     if (event.type == sf::Event::Closed) {
                         window.close();
                     }
                 }

```

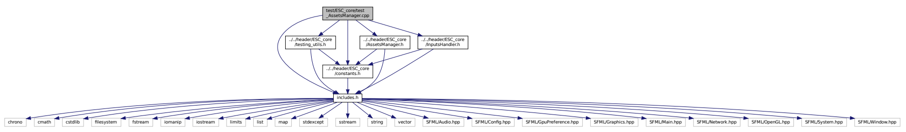
```
                messages_handler.process();

                hex_map.process();
                context_menu.process();

                if (inputs_handler.key_pressed_once_vec[sf::Keyboard::Q]) {
                    std::cout « "Q" « std::endl;
                    hex_map.reroll();
                }

                if (inputs_handler.key_pressed_once_vec[sf::Keyboard::R]) {
                    std::cout « "R" « std::endl;
                    hex_map.toggleResourceOverlay();
                }

                if (inputs_handler.key_pressed_once_vec[sf::Keyboard::A]) {
                    std::cout « "A" « std::endl;
                    hex_map.assess();
                }

                window.clear();

                hex_map.draw(); // draw hex map before context menu!
                context_menu.draw();

                window.display();

                inputs_handler.reset();

                std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
                frame++;
            }
        }
    }


    catch (...) {
        //...

        printGold(" ");
        for (int i = 0; i < n_dots; i++) {
            printGold(".");
        }
        printGold(" ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    //...

    printGold(" ");
    for (int i = 0; i < n_dots; i++) {
        printGold(".");
    }
    printGold(" ");
    printGreen("PASS");
    std::cout « std::endl;

    return 0;
}   /* main() */
```

4.19 test/ESC_core/test_AssetsManager.cpp File Reference

Suite of tests for the AssetsManager class.

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

Include dependency graph for test_AssetsManager.cpp:



Functions

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

4.19.1 Detailed Description

Suite of tests for the AssetsManager class.

A suite of tests for the AssetsManager class.

4.19.2 Function Documentation

4.19.2.1 main()

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

```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting AssetsManager");
    std::cout « std::endl;

    srand(time(NULL));
    int n_dots = 8;


    try {
        //  1. construct
        InputsHandler inputs_handler;
        AssetsManager assets_manager;


        //  2. load/open some test assets
        assets_manager.loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");
        assets_manager.loadTexture(
            "assets/ESC_brand/ESC_key_98x81.png",
            "ESC_key_98x81"
        );
        assets_manager.loadSound("assets/ESC_brand/key_press.ogg", "key_press");
        assets_manager.loadTrack(
            "assets/audio/tracks/AlexanderBlu_BackgroundElectronicModernMusic.ogg",
            "AlexanderBlu_BackgroundElectronicModernMusic"
        );


        //  3. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(800, 600), "Testing AssetsManager");
```

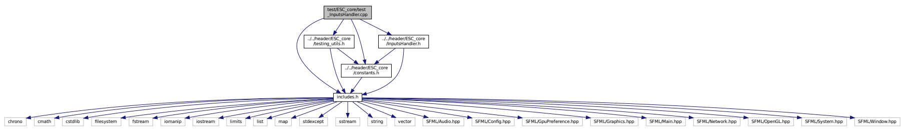
```cpp

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            800,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            600,
            __FILE__,
            __LINE__
        );

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

        assets_manager.playTrack();

        sf::Sprite ESC_key(*(assets_manager.getTexture("ESC_key_98x81")));

        double sprite_width = ESC_key.getLocalBounds().width;
        double sprite_height = ESC_key.getLocalBounds().height;

        double sprite_velocity_x = 256 * (2 * ((double)rand() / RAND_MAX) - 1);
        double sprite_velocity_y = 256 * (2 * ((double)rand() / RAND_MAX) - 1);

        ESC_key.setOrigin(sprite_width / 2, sprite_height / 2);
        ESC_key.setPosition(
            (screen_width - sprite_width) * ((double)rand() / RAND_MAX) + sprite_width / 2,
            (screen_height - sprite_height) * ((double)rand() / RAND_MAX) + sprite_height / 2
        );

        sf::Text click_text(
            "CLICK!",
            *(assets_manager.getFont("DroidSansMono")),
            16
        );

        double text_width = click_text.getLocalBounds().width;
        double text_height = click_text.getLocalBounds().height;

        click_text.setOrigin(text_width / 2, text_height / 2);

        int alpha = 255;

        click_text.setFillColor(sf::Color(255, 255, 255, alpha));

        while (window.isOpen()) {
            time_since_run_s = clock.getElapsedTime().asSeconds();

            if (
                time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
            ) {
                while (window.pollEvent(event))
                {
                    //...

                    if (event.type == sf::Event::Closed) {
                        window.close();
                    }
                }

                ESC_key.move(
                    sprite_velocity_x * SECONDS_PER_FRAME,
                    sprite_velocity_y * SECONDS_PER_FRAME
                );

                if (
                    ESC_key.getPosition().x <= sprite_width / 2 or
                    ESC_key.getPosition().x >= screen_width - sprite_width / 2
                ) {
                    sprite_velocity_x *= -1;

                    assets_manager.getSound("key_press")->play();

                    alpha = 255;
                    click_text.setPosition(
                        ESC_key.getPosition().x,
                        ESC_key.getPosition().y
                    );
                }

                if (
                    ESC_key.getPosition().y <= sprite_height / 2 or
                    ESC_key.getPosition().y >= screen_height - sprite_height / 2
                ) {
                    sprite_velocity_y *= -1;

                    assets_manager.getSound("key_press")->play();

                    alpha = 255;
                    click_text.setPosition(
                        ESC_key.getPosition().x,
                        ESC_key.getPosition().y
                    );
                }

                window.clear();

                window.draw(ESC_key);
                window.draw(click_text);

                window.display();

                alpha -= 8;
                if (alpha < 0) {
                    alpha = 0;
                }

                click_text.setFillColor(sf::Color(255, 255, 255, alpha));

                std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
                frame++;
            }
        }
    }


    catch (...) {
        //...

        printGold(" ");
        for (int i = 0; i < n_dots; i++) {
            printGold(".");
        }
        printGold(" ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    //...

    printGold(" ");
    for (int i = 0; i < n_dots; i++) {
        printGold(".");
    }
    printGold(" ");
    printGreen("PASS");
    std::cout « std::endl;

    return 0;
}   /* main() */
```

4.20 test/ESC_core/test_InputsHandler.cpp File Reference

Suite of tests for the InputsHandler class.

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

Include dependency graph for test_InputsHandler.cpp:



Functions

- int main (int argc, char ∗∗argv)

4.20.1 Detailed Description

Suite of tests for the InputsHandler class.

A suite of tests for the InputsHandler class.

4.20.2 Function Documentation

4.20.2.1 main()

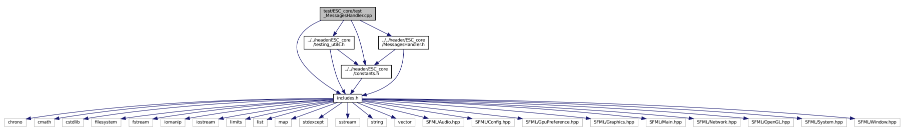
```
int main (
              int argc,
              char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting InputsHandler");
    std::cout « std::endl;

    srand(time(NULL));
    int n_dots = 8;


    try {
        //  1. construct and spot check attributes
        InputsHandler inputs_handler;

        testFloatEquals(
            int(sf::Keyboard::KeyCount),
            101,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            inputs_handler.key_press_vec.size(),
            int(sf::Keyboard::KeyCount),
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            inputs_handler.key_pressed_once_vec.size(),
            int(sf::Keyboard::KeyCount),
            __FILE__,
            __LINE__
        );


        //  2. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(800, 600), "Testing InputsHandler");

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            800,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            600,
            __FILE__,
            __LINE__
        );

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

         while (window.isOpen()) {
             time_since_run_s = clock.getElapsedTime().asSeconds();

             if (
                 time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
             ) {
                 while (window.pollEvent(event))
                 {
                     inputs_handler.process(&event);

                     if (event.type == sf::Event::Closed) {
                         window.close();
                     }
                 }

                 window.clear();
                 window.display();

                 inputs_handler.printKeysPressed();

                 if (inputs_handler.key_pressed_once_vec[sf::Keyboard::Enter]) {
                     std::cout « "Enter" « std::endl;
                 }



                 inputs_handler.reset();

                 std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
                 frame++;
             }
         }
    }


    catch (...) {
        //...

        printGold(" ");
        for (int i = 0; i < n_dots; i++) {
            printGold(".");
        }
        printGold(" ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    //...

    printGold(" ");
    for (int i = 0; i < n_dots; i++) {
        printGold(".");
    }
    printGold(" ");
    printGreen("PASS");
    std::cout « std::endl;

    return 0;
}   /* main() */
```

4.21 test/ESC_core/test_MessagesHandler.cpp File Reference

Suite of tests for the MessagesHandler class.

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

Include dependency graph for test_MessagesHandler.cpp:



Functions

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

4.21.1 Detailed Description

Suite of tests for the MessagesHandler class.

A suite of tests for the MessagesHandler class.

4.21.2 Function Documentation

4.21.2.1 main()

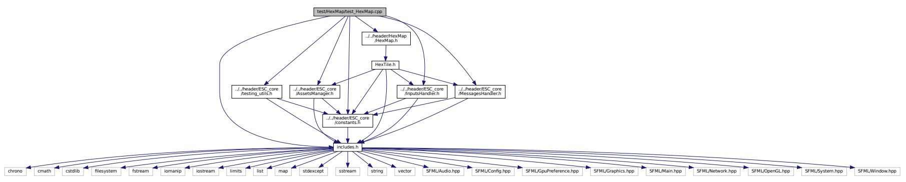
```
int main (
            int argc,
            char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting MessagesHandler");
    std::cout « std::endl;

    srand(time(NULL));
    int n_dots = 8;


    try {
        //  1. construct
        MessagesHandler messages_handler;


        //  2. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(800, 600), "Testing MessagesHandler");

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            800,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            600,
            __FILE__,
            __LINE__
        );

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

        while (window.isOpen()) {
            time_since_run_s = clock.getElapsedTime().asSeconds();

            if (
                time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
            ) {
                while (window.pollEvent(event))
                {
                    //...

                    if (event.type == sf::Event::Closed) {
                        window.close();
                    }
                }

                window.clear();
                window.display();

                std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
                frame++;
            }
        }
    }


    catch (...) {
        //...

        printGold(" ");
        for (int i = 0; i < n_dots; i++) {
            printGold(".");
        }
        printGold(" ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    //...

    printGold(" ");
    for (int i = 0; i < n_dots; i++) {
        printGold(".");
    }
    printGold(" ");
    printGreen("PASS");
    std::cout « std::endl;

    return 0;
}   /* main() */
```

4.22 test/HexMap/test_HexMap.cpp File Reference

Suite of tests for the HexMap class.

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

Include dependency graph for test_HexMap.cpp:



Functions

- int main (int argc, char ∗∗argv)

4.22.1 Detailed Description

Suite of tests for the HexMap class.

A suite of tests for the HexMap class.

4.22.2 Function Documentation

4.22.2.1 main()

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

```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting HexMap");
    std::cout « std::endl;

    srand(time(NULL));
    int n_dots = 8;


    try {
        //  1. construct, load/open some test assets
        AssetsManager assets_manager;
        InputsHandler inputs_handler;
        MessagesHandler messages_handler;

        assets_manager.loadFont("assets/fonts/DroidSansMono.ttf", "DroidSansMono");


        //  2. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(
            sf::VideoMode(GAME_WIDTH, GAME_HEIGHT),
            "Testing HexMap"
        );

        double screen_width = window.getSize().x;
        double screen_height = window.getSize().y;

        testFloatEquals(
            screen_width,
            1200,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            screen_height,
            800,
            __FILE__,
            __LINE__
        );

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

        HexMap hex_map(
            6,
            &assets_manager,
            &inputs_handler,
            &messages_handler,
            &window
        );

        while (window.isOpen()) {
            time_since_run_s = clock.getElapsedTime().asSeconds();

             if (
                time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
             ) {
                while (window.pollEvent(event))
                {
                    inputs_handler.process(&event);

                    if (event.type == sf::Event::Closed) {
                        window.close();
                    }
                }

                hex_map.process();

                if (inputs_handler.key_pressed_once_vec[sf::Keyboard::Q]) {
                    std::cout « "Q" « std::endl;
                    hex_map.reroll();
                }

                if (inputs_handler.key_pressed_once_vec[sf::Keyboard::R]) {
                    std::cout « "R" « std::endl;
                    hex_map.toggleResourceOverlay();
                }

                if (inputs_handler.key_pressed_once_vec[sf::Keyboard::A]) {
                    std::cout « "A" « std::endl;
                    hex_map.assess();
                }

                window.clear();

                hex_map.draw();

                window.display();

                inputs_handler.reset();

                std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;
                frame++;
             }
         }
    }


    catch (...) {
        //...

        printGold(" ");
        for (int i = 0; i < n_dots; i++) {
            printGold(".");
        }
        printGold(" ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    //...

    printGold(" ");
    for (int i = 0; i < n_dots; i++) {
        printGold(".");
    }
    printGold(" ");
    printGreen("PASS");
    std::cout « std::endl;

    return 0;
}   /* main() */
```

Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL https://www.sfml-dev.org/. 94

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL https://www.doxygen.nl. 93

Wikipedia. Hexagon, 2023. URL https://en.wikipedia.org/wiki/Hexagon. 54

Index