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 HexMap Class Reference

A class which defines a hex map of hex tiles.

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

Public Member Functions

- HexMap (int)

  Constructor for the HexMap class.
- void draw (sf::RenderWindow ∗)

  Method to draw the hex map to the render window.
- 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.
- double position_x

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

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

  A vector of pointers to the tiles in the hex map.

Private Member Functions

- void __assembleHexMap (void)

  Helper method to assemble the hex map.

3.2.1 Detailed Description

A class which defines a hex map of hex tiles.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 HexMap()

```
HexMap::HexMap (
              int n_layers )
```

Constructor for the HexMap class.

Parameters

n_layers	The number of layers in the HexMap.

```
{
    //  1. set attributes
    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();

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

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

3.2.2.2 ∼HexMap()

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

Destructor for the HexMap class.

```
{
    this->clear();

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

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

3.2.3 Member Function Documentation

3.2.3.1 __assembleHexMap()

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

Helper method to assemble the hex map.

```
{
    //  1. add origin tile
    HexTile* hex_ptr = new HexTile(this->position_x, this->position_y);

    this->hex_vec.push_back(hex_ptr);


    //  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->hex_vec.push_back(hex_ptr);

        hex_ptr = new HexTile(
            this->position_x - 2 * (i + 1) * hex_ptr->minor_radius,
            this->position_y
        );

        this->hex_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 = this->hex_vec.size() - 1;
        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->hex_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->hex_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->hex_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->hex_vec.push_back(hex_ptr);
        }

        offset_count++;
    }

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

3.2.3.2 clear()

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

Method to clear the hex map.

```
{
    for (size_t i = 0; i < this->hex_vec.size(); i++) {
        delete this->hex_vec[i];
    }
    this->hex_vec.clear();

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

3.2.3.3 draw()

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

Method to draw the hex map to the render window.

Parameters

window_ptr	A pointer to the render window.

```
{
    for (size_t i = 0; i < this->hex_vec.size(); i++) {
        this->hex_vec[i]->draw(window_ptr);
    }

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

3.2.4 Member Data Documentation

3.2.4.1 hex_vec

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

A vector of pointers to the tiles in the hex map.

3.2.4.2 n_layers

```
int HexMap::n_layers
```

The number of layers in the hex map.

3.2.4.3 position_x

```
double HexMap::position_x
```

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

3.2.4.4 position_y

```
double HexMap::position_y
```

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

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

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

3.3 HexTile Class Reference

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

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

Public Member Functions

- HexTile (double, double)

  Constructor for the HexTile class.
- void draw (sf::RenderWindow ∗)

  Method to draw the hex tile to the render window.
- ∼HexTile (void)

  Destructor for the HexTile class.

Public Attributes

- bool show_node

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

  The x position of the tile.
- double position_y

  The y position of the tile.
- double major_radius

  The radius of the smallest bounding circle.
- double minor_radius

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

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

  A convex shape which represents the tile.

Private Member Functions

- void __setUpNodeSprite (void)

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

  Helper method to set up tile sprite.

3.3.1 Detailed Description

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

3.3.2 Constructor & Destructor Documentation

3.3.2.1 HexTile()

```
HexTile::HexTile (
             double position_x,
             double position_y )
```

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.

```
{
    //  1. set attributes
    this->show_node = false;

    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 the node sprite
    this->__setUpNodeSprite();

    //  3. set up and position the tile sprite
    this->__setUpTileSprite();

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

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

3.3.2.2 ∼HexTile()

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

Destructor for the HexTile class.

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

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

3.3.3 Member Function Documentation

3.3.3.1 __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() */
```

3.3.3.2 __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(2);
    this->tile_sprite.setOutlineColor(sf::Color(0, 0, 0, 255));

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

3.3.3.3 draw()

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

Method to draw the hex tile to the render window.

Parameters

window_ptr	A pointer to the render window.

```
{
    //  1. draw hex
    window_ptr->draw(this->tile_sprite);

    //  2. draw node
    if (this->show_node) {
        window_ptr->draw(this->node_sprite);
    }

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

3.3.4 Member Data Documentation

3.3.4.1 major_radius

```
double HexTile::major_radius
```

The radius of the smallest bounding circle.

3.3.4.2 minor_radius

```
double HexTile::minor_radius
```

The radius of the largest inscribed circle.

3.3.4.3 node_sprite

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

A circle shape to mark the tile node.

3.3.4.4 position_x

```
double HexTile::position_x
```

The x position of the tile.

3.3.4.5 position_y

```
double HexTile::position_y
```

The y position of the tile.

3.3.4.6 show_node

```
bool HexTile::show_node
```

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

3.3.4.7 tile_sprite

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

A convex shape which represents the tile.

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

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

3.4 InputsHandler Class Reference

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

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

Public Member Functions

- InputsHandler (void)
  Constructor for the InputsHandler class.
- void process (sf::Event ∗)
- void printKeysPressed (void)
  Method to print out which keys are currently pressed.
- void reset (void)
  Method to reset InputsHandler. To be called once per frame (at end of frame!).
- ∼InputsHandler (void)
  Destructor for the InputsHandler class.

Public Attributes

- std::vector< bool > key_pressed_once_vec
  A vector (bool) which indicates which keys have been pressed once. Useful for discrete inputs.
- std::vector< bool > key_press_vec
  A vector <bool> which indicates which keys are currently pressed. Useful for smooth movement.
- std::map< sf::Keyboard::Key, std::string > key_code_map
  A map from key codes to corresponding string representations.

Private Member Functions

- void __constructKeyCodeMap (void)
  Helper method to construct a map from sf::Keyboard::Key to a string representation of the corresponding key.

3.4.1 Detailed Description

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

3.4.2 Constructor & Destructor Documentation

3.4.2.1 InputsHandler()

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

Constructor for the InputsHandler class.

```
{
    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() */
```

3.4.2.2 ∼InputsHandler()

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

Destructor for the InputsHandler class.

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

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

3.4.3 Member Function Documentation

3.4.3.1 __constructKeyCodeMap()

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

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

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

3.4.3.2 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() */
```

3.4.3.3 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;

            break;
        }

        case (sf::Event::KeyReleased): {
            this->key_pressed_once_vec[event_ptr->key.code] = false;
            this->key_press_vec[event_ptr->key.code] = false;

            break;
        }

        default: {
            //  do nothing!

            break;
        }
    }

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

3.4.3.4 reset()

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

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

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

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

3.4.4 Member Data Documentation

3.4.4.1 key_code_map

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

A map from key codes to corresponding string representations.

3.4.4.2 key_press_vec

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

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

3.4.4.3 key_pressed_once_vec

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

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

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

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

Chapter 4

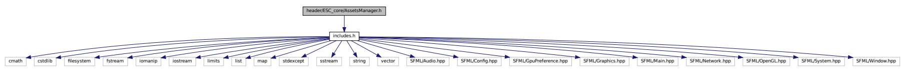
File Documentation

4.1 header/ESC_core/AssetsManager.h File Reference

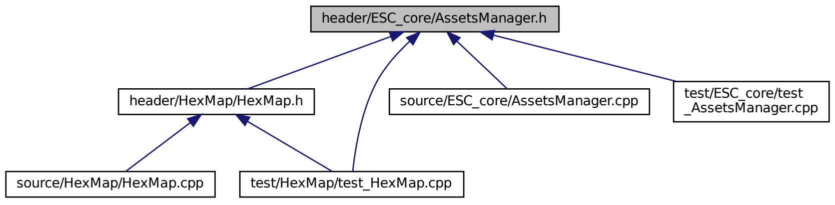
Header file for the AssetsManager class.

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

Include dependency graph for AssetsManager.h:



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


header/ESC_core/AssetsManager.h
header/HexMap/HexMap.h
source/ESC_core/AssetsManager.cpp
test/ESC_core/test_AssetsManager.cpp
source/HexMap/HexMap.cpp
test/HexMap/test_HexMap.cpp

Classes

- class AssetsManager

  A class which manages visual and sound assets.

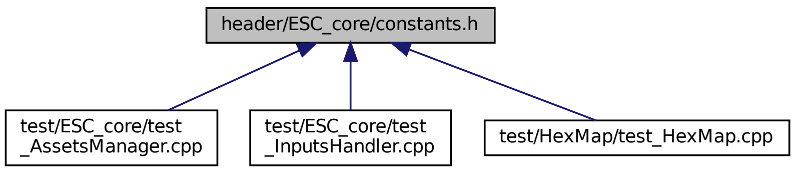
4.1.1 Detailed Description

Header file for the AssetsManager class.

4.2 header/ESC_core/constants.h File Reference

Header file for various constants.

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


header/ESC_core/constants.h
test/ESC_core/test_AssetsManager.cpp
test/ESC_core/test_InputsHandler.cpp
test/HexMap/test_HexMap.cpp

Variables

- const int FRAMES_PER_SECOND = 60

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

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

4.2.1 Detailed Description

Header file for various constants.

4.2.2 Variable Documentation

4.2.2.1 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

Target frames per second.

4.2.2.2 SECONDS_PER_FRAME

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

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

4.3 header/ESC_core/doxygen_cite.h File Reference

Header file which simply cites the doxygen tool.

4.3.1 Detailed Description

Header file which simply cites the doxygen tool.

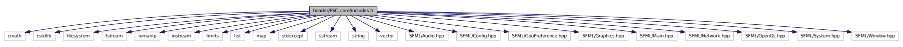
Ref: van Heesch. [2023]

4.4 header/ESC_core/includes.h File Reference

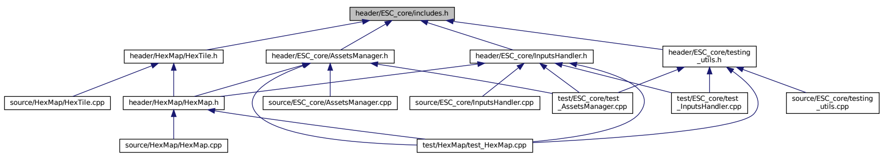
Header file for various includes.

```
#include <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/HexMap/HexTile.h
header/ESC_core/AssetsManager.h
header/ESC_core/InputsHandler.h
header/ESC_core/testing_utils.h
source/HexMap/HexTile.cpp
header/HexMap/HexMap.h
source/ESC_core/AssetsManager.cpp
source/ESC_core/InputsHandler.cpp
test/ESC_core/test_AssetsManager.cpp
test/ESC_core/test_InputsHandler.cpp
source/ESC_core/testing_utils.cpp
source/HexMap/HexMap.cpp
test/HexMap/test_HexMap.cpp

4.4.1 Detailed Description

Header file for various includes.

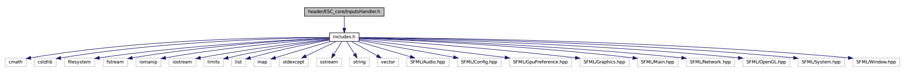
Ref: Gomila [2023]

4.5 header/ESC_core/InputsHandler.h File Reference

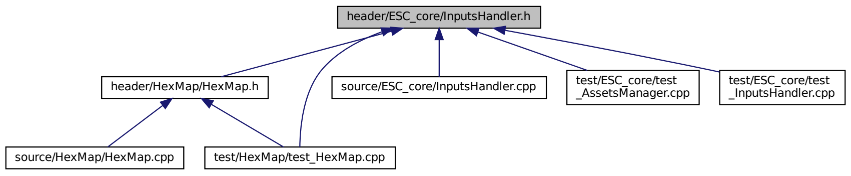
Header file for the InputsHandler class.

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

Include dependency graph for InputsHandler.h:



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


header/ESC_core/InputsHandler.h
header/HexMap/HexMap.h
source/ESC_core/InputsHandler.cpp
test/ESC_core/test_AssetsManager.cpp
test/ESC_core/test_InputsHandler.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).

4.5.1 Detailed Description

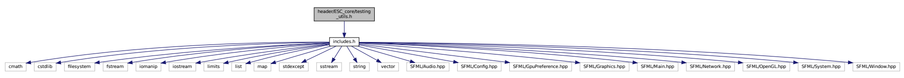
Header file for the InputsHandler class.

4.6 header/ESC_core/testing_utils.h File Reference

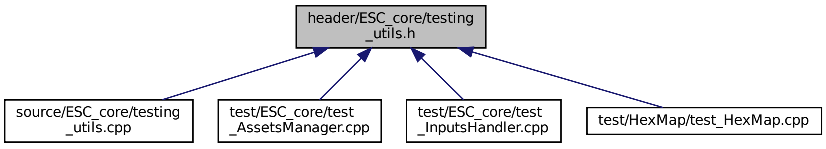
Header file for various testing utilities.

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

Include dependency graph for testing_utils.h:



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


header/ESC_core/testing_utils.h
source/ESC_core/testing_utils.cpp
test/ESC_core/test_AssetsManager.cpp
test/ESC_core/test_InputsHandler.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.

Variables

- const double FLOAT_TOLERANCE = 1e-6

    Tolerance for floating point equality tests.

4.6.1 Detailed Description

Header file for various testing utilities.

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

4.6.2 Function Documentation

4.6.2.1 expectedErrorNotDetected()

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

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

Parameters

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

```
{
    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() */
```

4.6.2.2 printGold()

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

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

Parameters

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

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

4.6.2.3 printGreen()

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

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

Parameters

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

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

4.6.2.4 printRed()

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

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

Parameters

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

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

4.6.2.5 testFloatEquals()

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

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

Parameters

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

```
{
    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() */
```

4.6.2.6 testGreaterThan()

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

Tests if x > y.

Parameters

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

```
{
    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() */
```

4.6.2.7 testGreaterThanOrEqualTo()

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

Tests if x >= y.

Parameters

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

```
{
    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() */
```

4.6.2.8 testLessThan()

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

Tests if x < y.

Parameters

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

```
{
    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() */
```

4.6.2.9 testLessThanOrEqualTo()

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

Tests if x <= y.

Parameters

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

```
{
    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() */
```

4.6.2.10 testTruth()

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

Tests if the given statement is true.

Parameters

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

```
{
    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() */
```

4.6.3 Variable Documentation

4.6.3.1 FLOAT_TOLERANCE

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

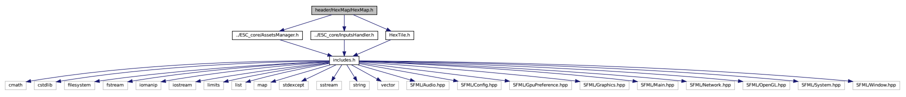
Tolerance for floating point equality tests.

4.7 header/HexMap/HexMap.h File Reference

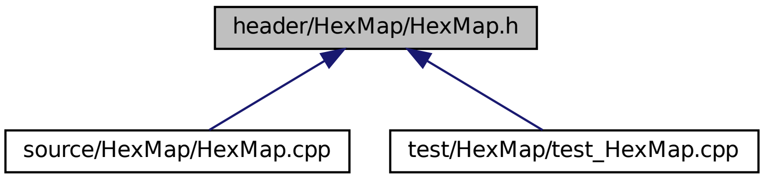
Header file for the HexMap class.

```
#include "../ESC_core/AssetsManager.h"
#include "../ESC_core/InputsHandler.h"
#include "HexTile.h"
```

Include dependency graph for HexMap.h:



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



Classes

- class HexMap

  A class which defines a hex map of hex tiles.

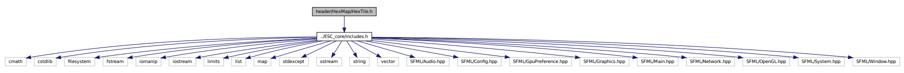
4.7.1 Detailed Description

Header file for the HexMap class.

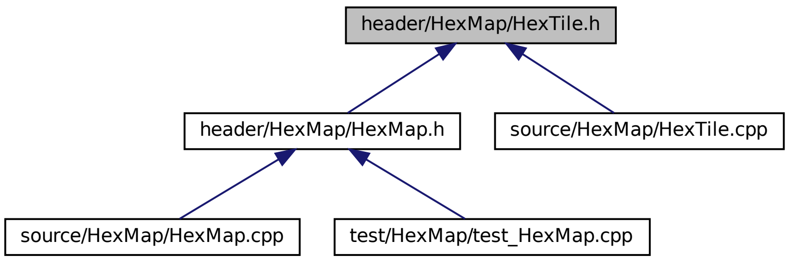
4.8 header/HexMap/HexTile.h File Reference

Header file for the HexTile class.

```
#include "../ESC_core/includes.h"
```
Include dependency graph for HexTile.h:



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



Classes

- class HexTile

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

4.8.1 Detailed Description

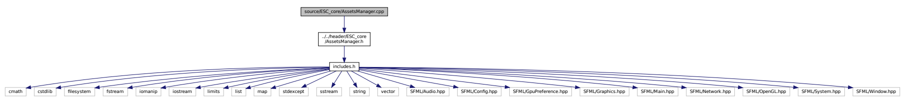
Header file for the HexTile class.

4.9 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:



4.9.1 Detailed Description

Implementation file for the AssetsManager class.

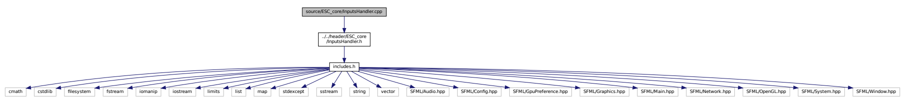
A class which manages visual and sound assets.

4.10 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:



4.10.1 Detailed Description

Implementation file for the InputsHandler class.

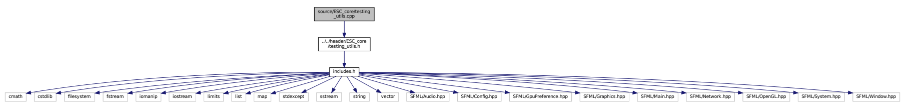
A class which handles inputs from peripherals (i.e., keyboard and mouse).

4.11 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.

4.11.1 Detailed Description

Implementation file for various testing utilities.

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

4.11.2 Function Documentation

4.11.2.1 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() */
```

4.11.2.2 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() */
```

4.11.2.3 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() */
```

4.11.2.4 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() */
```

4.11.2.5 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() */
```

4.11.2.6 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() */
```

4.11.2.7 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() */
```

4.11.2.8 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() */
```

4.11.2.9 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() */
```

4.11.2.10 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__").

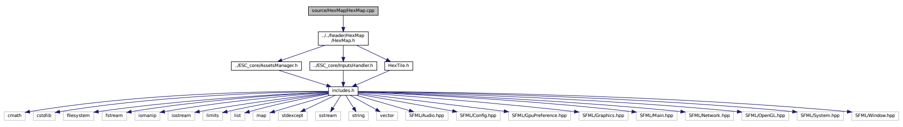
```
{
    if (statement) {
        return;
    }

    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() */
```

4.12 source/HexMap/HexMap.cpp File Reference

Implementation file for the HexMap class.

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

Include dependency graph for HexMap.cpp:



4.12.1 Detailed Description

Implementation file for the HexMap class.

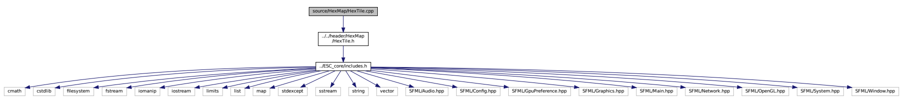
A class which defines a hex map of hex tiles.

4.13 source/HexMap/HexTile.cpp File Reference

Implementation file for the HexTile class.

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

Include dependency graph for HexTile.cpp:



4.13.1 Detailed Description

Implementation file for the HexTile class.

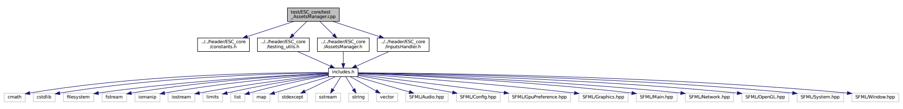
A class which defines a tile of a hex map.

4.14 test/ESC_core/test_AssetsManager.cpp File Reference

Suite of tests for the AssetsManager class.

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

Include dependency graph for test_AssetsManager.cpp:



Functions

- int main (int argc, char ∗∗argv)

4.14.1 Detailed Description

Suite of tests for the AssetsManager class.

A suite of tests for the AssetsManager class.

4.14.2 Function Documentation

4.14.2.1 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");

        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();
```

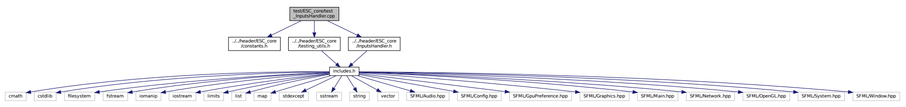
```

                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() */
```

4.15 test/ESC_core/test_InputsHandler.cpp File Reference

Suite of tests for the InputsHandler class.

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

Include dependency graph for test_InputsHandler.cpp:



Functions

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

4.15.1 Detailed Description

Suite of tests for the InputsHandler class.

A suite of tests for the InputsHandler class.

4.15.2 Function Documentation

4.15.2.1 main()

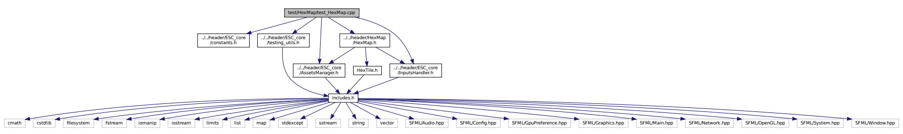
```
int main (
            int argc,
            char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    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() */
```

4.16 test/HexMap/test_HexMap.cpp File Reference

Suite of tests for the HexMap class.

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

Include dependency graph for test_HexMap.cpp:



Functions

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

4.16.1 Detailed Description

Suite of tests for the HexMap class.

A suite of tests for the HexMap class.

4.16.2 Function Documentation

4.16.2.1 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
        InputsHandler inputs_handler;
        AssetsManager assets_manager;
        HexMap hex_map(6);

        //  2. ...


        //  3. test game loop
        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(1200, 800), "Testing AssetsManager");

        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;

        //...

        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();

                hex_map.draw(&window);

                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() */
```

Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL https://www.sfml-dev.org/. 38

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL https://www.doxygen.nl. 37

Wikipedia. Hexagon, 2023. URL https://en.wikipedia.org/wiki/Hexagon. 22

Index