Tower Defense 2

Generated by Doxygen 1.8.17

Chapter 1

Tower Defense 2

1.1 Group members

- Elias Peltokangas
- Kabir Bissessar
- Juho Poteri
- Antti Pekkanen

1.2 Overview

Goal: create a Tower Defense game, using C++

"Tower defense (or informally TD) is a subgenre of strategy video game where the goal is to defend a player's territories or possessions by obstructing enemy attackers, usually achieved by placing defensive structures on or along their path of attack."

In a tower defense game, the enemies move in waves from some position of the map to another. The goal of the player is to place towers on their path in order to block, impede, attack or destroy the enemies before they are able to reach their goal. The primary object is the survival of the base.

The theme of this game is "Wave University" - where students try to complete the assignments

1.2.1 Features of the project

1.2.1.1 Minimum Requirements

- Functioning tower defense game
- Basic graphics
- At least three different types of towers
- At least three different types of enemies
- At least five different levels, with increasing difficulty
- Game can be controlled with mouse input
- User interface which shows player information e.g. Score, Money, Health

1.2.1.2 Additional Features Implemented

- Non-hardcoded maps - maps can be read from files
- Upgradeable towers
- Level editor - maps can be edited using GUI
- High scores list
- Towers can be damaged by enemies
- Sound effects

1.3 Building instructions

This project was built and tested using Linux. It is also possible to run using WSL on Windows 11, however WSL on Windows 10 does not support it.

```
# Download the repository
git clone git@version.aalto.fi:bissesk1/tower-defense-2.git
cd tower-defense-2
# Build the project
cmake -B ./build
make -C ./build
# Start the game
./build/tower-defense
```

The library SFML was used to build this game. It can be installed using

```
sudo apt-get install libsfml-dev
```

1.4 Project Structure

- doc/ – This folder contains the documentation of the project
- images/ This folder contains media (.png .wav) files necessary for the project
- plan/ – This folder contains the project plan, created at the start of the project
- src/ – This folder contains the C++ source files for the project.
- tests/ – This folder contains C++ files used to test the project, during the development stage.

1.5 Using the Software

The game has two main phases: the build phase and wave phase. The game starts with the build phase.

During the build phase, towers can be built, upgraded and/or destroyed on the map. When "Next round" is selected, the phase is switched. If the game is in the build phase, the wave phase would then start.

During the wave phase, the enemies spawn from the start tile and follow the path to the end tile. Towers will attack the enemies, if they are in range. If the towers kill the enemies, the player gains money and the score increases. However, if enemies survive the towers and make it to the end tile, the player loses health.

The game launches on this screen.

By selecting a map and clicking "Edit map", the level editor will be opened, where the selected map can be edited using a visual interface :

When difficulty and map can be selected, the game can be started by clicking "Play now".

The game will then enter the build phase, which looks like this :

Towers can be built by selecting a tile, and then clicking the tower type.

Attack towers will deal damage to enemies.

Buff towers will enhance the damage of Attack towers.

Heal towers will heal Attack towers as they take damage, and revive them if they die.

Towers can be placed on any tile in the game, except the path tiles. When a tile is selected, the attack range of a tower can be seen by hovering over the tower type. After a tower is built on the map, it can be upgraded and/or destroyed.

When "Next round" is selected, the game will enter the wave phase, which looks like this :

The wave phase can be sped up by selecting a "Gamespeed".

Some enemies explode into multiple weaker enemies when they are killed. When these enemies explode, they will attack the tower that killed them. Towers can die as a result of this, which will cause them to stop attacking enemies.

When the player loses all their health, or when "Give up" is clicked, the game ends and the High Score screen is shown.

From there, the player can choose to save their score or go to the main menu.

1.6 Work log

Division of work among group members

- Elias Peltokangas
  - Implemented non-hardcoded maps - maps can be read from files ()
  - Implemented Level editor
  - Designed towers, enemies, map and path tiles
  - Maintained code to support Doxygen documentation
- Kabir Bissessar
  - Implemented GUI - sprites, buttons, text
  - Added sound effects
  - Documentation
- Juho Poteri
  - Implemented Enemy Factory - enemies spawned with increasing difficulty
  - Implemented Tower class - types of towers and how they interact with enemies
  - Implemented Enemy class - types of enemies and how they interact with the map/towers
  - Maintained code to support Doxygen documentation
- Antti Pekkanen
  - Wrote game core - logic that causes game to run
  - Implemented game scoring and high scores list
  - Player information displayed during game - Score, Money, Health
  - Implemented CMake build

Chapter 2

Contents

The actual project documentation in PDF format must be commited in this folder before the deadline. Separate PDF document needs to be provided also if your project uses Doxygen for inline documentation.

The document should contain the following parts:

1. **Overview:** what the software does, what it doesn't do? (this can be taken/updated from the project plan)
2. **Software structure:** overall architecture, class relationships (diagram very strongly recommended), interfaces to external libraries
3. **Instructions** for building and using the software
4. **How to compile the program** ('make' should be sufficient), as taken from git repository. If external libraries are needed, describe the requirements here
5. How to use the software: a basic user guide
6. **Testing:** how the different modules in software were tested, description of the methods and outcomes
7. **Work log:** This might be a simplified/restructured version of the weekly meeting notes file.
8. Detailed description of division of work and everyone's responsibilities
9. For each week, description of what was done and roughly how many hours were used, for each project member.

Chapter 3

LIBS directory

In this directory, you are required to place all the external libraries your project depends on. Although, in principle, you can use git submodules (and place them under this directory), for the sake of easily compiling your application, placing the source code of the open source libraries is also fine. However, this approach is not applicable to large dependencies, such as QT.

3.1 List of External Libs

1. `Project1`
2. `Project2`

If you are using already compiled library, place it in this folder, and set the linker options appropriately. The include files of the dependent library should also be placed in this folder.

Chapter 4

Meeting Notes

In this file, you are required to take notes for your weekly meetings. In each meeting, you are required to discuss:

1. What each member has done during the week?
2. Are there challenges or problems? Discuss the possible solutions
3. Plan for the next week for everyone
4. Deviations and changes to the project plan, if any

4.1 Week 1 Meeting 1 07.11.2022 18:00

Participants:

1. Elias Peltokangas
2. Kabir Bissessar
3. Juho Poteri
4. Antti Pekkanen

4.1.1 Summary of the meeting

- Introductions, discussed backgrounds in programming.
  This is the first or one of the first group programming projects for all of us.
- Agreed on "the secretary".
  Elias will be responsible for meeting notes, and possible other documentations / text submissions.
- Agreed on communication.
  Created a Telegram channel which will be used for internal communication.
- Sketched the contents for the project plan (game structure, initial feature list, work division etc.).
  Agreed on Elias finishing it.

4.1.2 Project status

The project has officially been started and a sketch for the plan has been made.

4.1.3 Next meeting

Next meeting will be on Wednesday. The project plan will be discussed and finalized as well as next steps and next meeting determined.

4.1.4 TODOs for the next meeting

- Create a draft of the full project plan (Elias)
- Create/acquire an initial idea of your part and the whole project. (Everyone)

4.2 Week 1 Meeting 2 09.11.2022 17:00

Participants:

1. Elias Peltokangas
2. Kabir Bissessar
3. Juho Poteri
4. Antti Pekkanen

4.2.1 Summary of works

1. **Elias**: Project plan draft completed.
2. **Kabir**: Initial preparations.
3. **Juho**: High-level draft of enemy and tower classes done.
4. **Antti**: Initial preparations.

4.2.2 Challenges

1. How to have enough time to build all features.

4.2.3 Actions

This week:

1. **Elias**: finalize and commit the project plan and meeting notes.

Before next meeting:

1. **Elias**: Build first version of the Map class and the map system.
2. **Kabir**: Look into GUIs and the SFML, add SFML to project, make a base/prototype window to build on.
3. **Juho**: Build first versions of Enemy and Tower classes and some subclasses, build class that generates waves of enemies.
4. **Antti**: Create first working Cmake and a basic hello world main.c, create first versions of the game class and gametime loop.

4.2.4 Project status

High level project plan is done. Some of the interfaces between software modules were preliminarily planned during this meeting.

4.3 Week 2 Meeting 18.11.2022 15:00

Participants:

1. Elias Peltokangas
2. Kabir Bissessar
3. Juho Poteri
4. Antti Pekkanen

4.3.1 Summary of works

1. **Elias**: First version of map class, initialization from textfile and validation complete. Path building complete.
2. **Kabir**: Figuring out the SFML, how to run it etc.
3. **Juho**: Enemy_Factory nearly complete, enemies in progress
4. **Antti**: Gametime loop under design, cmake done, hello world done

4.3.2 Challenges

1. Running the SFML GUI has problems depending on the system
2. How the game core works (we discussed this)

4.3.3 Actions

1. **Elias**: Draw graphics for a resolution of about 1200x800, finish Map class (GetStart() method etc.).
2. **Kabir**: Get SFML working, create the drawing methods. Start working on the GUI class.
3. **Juho**: Tower Act takes the map of (x, y) -> Enemy as a parameter and enemy advance returns true or false depending.
4. **Antti**: Game class: variables, map, towerlist, enemypos and methods AdvanceEnemies and TowersAction. Work on GUI loop with Kabir.

4.3.4 Project status

Map and enemies are roughly complete. Goal of the next week: Get the wave part of the game running

4.4 Week 3 Meeting 1 25.11.2022 15:00

Participants:

1. Kabir Bissessar
2. Juho Poteri
3. Antti Pekkanen

4.4.1 Summary of works

1. **Elias**: Images done for tiles, enemies, towers and a basic background.
2. **Kabir**: GUI can now draw general shapes.
3. **Juho**: Towers have a basic logic to attack enemies etc. (towers and enemies communicate together)
4. **Antti**: Game core has functions that advance enemies and make towers attack. (Game communicates with towers and enemies)

4.4.2 Actions

1. **Elias**: Building the shop part of the game (with Kabir).
2. **Kabir**: Figuring out sprites and creating a map from maptiles (then enemies and towers).
3. **Juho**: Tower upgrade logic.
4. **Antti**: Tower building logic for the game core.

4.4.3 Project status

GUI still a work in progress. Basic structures for core and classes done. Communication between classes is partly done.
Next focusing on piecing the puzzle together and adding the rest of the features to complete the game.

4.5 Week 4 Meeting 1 30.11.2022 15:00

Participants:

1. Elias Peltokangas
2. Kabir Bissessar
3. Juho Poteri
4. Antti Pekkanen

4.5.1 Summary of works

1. **Elias**: Images for sprites.
2. **Kabir**: Figured out sprites and creating the window.
3. **Juho**: Tower upgrade stuff done.
4. **Antti**: Tower building in progress.

4.5.2 Actions

1. **Elias**: Create the abstract State class. Create all backend functionality for level editor.
2. **Kabir**: Initializing the GameState: drawing the map tiles then moving to building the round, create the loop for the building
3. **Juho**: Make the game core TowerTurn() return a list of actions ((x,y), (x,y)), ((x,y), (x, y)), Create the loop for running the wave in GameState without the drawing stuff. (Calls towerturn and enemyturn and saves the returned values for drawing the projectiles etc.)
4. **Antti**: Finish game core backend: add money + methods for increase and decrease, add game core method for create tower at x,y and remove a certain tower. Create and finish all high score backend functionality

4.5.3 Project status

Backend is close to complete, GUI and the game loops have been designed and need to be implemented.

4.6 Week 5 Meeting 1 07.12.2022 11:00

Participants:

1. Elias Peltokangas
2. Kabir Bissessar
3. Juho Poteri
4. Antti Pekkanen

4.6.1 Summary of works

1. **Elias**: LevelEditor, pseudo code for MenuState.
2. **Kabir**: Renderables, planning the GameState
3. **Juho**: Text based test to fix the game backend.
4. **Antti**: Building towers, money logic.

4.6.2 Actions

1. Working face to face together through wednesday to friday to finish the project.

4.6.3 Project status

Let's finish this before the deadline!

Chapter 5

Contents

Project plan is a PDF document describing the scope of the project, major architectural decisions, preliminary schedule and distribution of roles in the group, design rationale and so on. The document should be roughly five pages long, with a couple of diagrams illustrating the program design (for example, the planned class relationships).

You are required commit your project plan in this folder before the deadline. The plan should contain the following information:

- Scope of the work: what features and functionalities will be implemented, how is the program used, and how does it work
- High-level structure of the software: main modules, main classes (according to current understanding)
- Planned use of external libraries
- Division of work and responsibilities between the group
- Planned schedule and milestones before the final deadline of the project

It is not uncommon that as the project progresses, there may be changes relative to project plan, and that is fine. The final outcome will be described in the final documentation, that can be based on the project plan.

Chapter 6

Source content

This folder should contain only hpp/cpp files of your implementation. You can also place hpp files in a separate directory `include`.

You can create a summary of files here. It might be useful to describe file relations, and brief summary of their content.

6.1 Files:

- Map.hpp/.cpp contains the map system, including the map class.

Chapter 7

Test files

It is a common practice to do unit tests of each class before you integrate it into the project to validate its operation. In this folder, you can create your own unit test files to validate the operation of your components.

It might be a good idea to also take some notes about the tests since you are required to

report these in the final report.

7.1 Unit Tests

7.1.1 Test of EnemyFactory

Involved Classes: EnemyFactory, Assignment, Degree(TODO), Renderable

Test File: enemy_factory_test.cpp

Results: An initial check on first few rounds indicates that the logic works and no leaks were spotted using valgrind

7.1.2 Test of The general logic and interactions between towers and enemies

Involved Classes: Game, Map, all tower and enemy related classes

Test File: text_based_test.cpp

Results: Some bugs were found (mainly with loop conditions) but they were fixed. After the fixes no memory leaks were found using valgrind.

Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

Chapter 10

Class Index

10.1 Class List

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

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

Chapter 12

Namespace Documentation

12.1 UtilFunctions Namespace Reference

A namespace containing some utility functions needed in the project.

Functions

- float distance (const std::pair< int32_t, int32_t > &c1, const std::pair< int32_t, int32_t > &c2)

  Calculates the Euclidean distance between two coordinate pairds.

12.1.1 Detailed Description

A namespace containing some utility functions needed in the project.

12.1.2 Function Documentation

12.1.2.1 distance()

```
float UtilFunctions::distance (
            const std::pair< int32_t, int32_t > & c1,
            const std::pair< int32_t, int32_t > & c2 )
```

Calculates the Euclidean distance between two coordinate pairds.

Parameters

c1	coordinate 1
c2	coordinate 2

Returns

float

Chapter 13

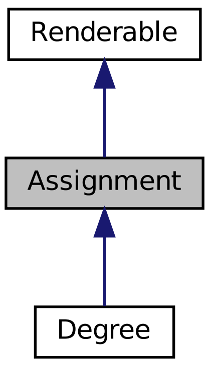
Class Documentation

13.1 Assignment Class Reference

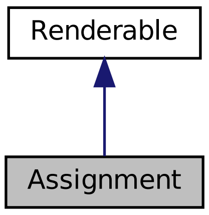
Generic Enemy class, these just die when they are killed.

```
#include <assignment.hpp>
```

Inheritance diagram for Assignment:



Collaboration diagram for Assignment:



Public Member Functions

- Assignment (uint32_t cr, uint32_t timeToMove, const std::string &name, const sf::Sprite &sprite)

  Construct a new Assignment object.
- virtual ∼Assignment ()=default

  A virtual destructor.
- bool Advance ()

  Tells if the enemy must be moved forward Returns true if the enemy moves to the next place.
- bool MovedLastTick () const

  Tells if the enemy moved during the last tick For animation purposes.
- bool IsAlive () const

  Tells whether this enemy is still alive or not.
- uint32_t CrLeft () const

  Tells the amount of health the enemy has left.
- uint32_t GetCredits () const

  Totalt amount of credits the enemy is worth.
- virtual uint32_t TakeDmg (uint32_t dmg, std::list< Assignment ∗ > &location)

  Makes the enemy take damage Splitting enemies hurt the attacking tower when they die. The return value tells the amount of this damage. The method is overridden in subclass Degree.

Protected Attributes

- uint32_t m_maxCr
- uint32_t m_curCr
- uint32_t m_timeToMove
- uint32_t m_timeRemainder
- bool m_movedLastTick

Friends

- std::ostream & operator<< (std::ostream &os, const Assignment &as)

  Overload for stream operator << for debugging.

Additional Inherited Members

13.1.1 Detailed Description

Generic Enemy class, these just die when they are killed.

13.1.2 Constructor & Destructor Documentation

13.1.2.1 Assignment()

```
Assignment::Assignment (
             uint32_t cr,
             uint32_t timeToMove,
             const std::string & name,
             const sf::Sprite & sprite )
```

Construct a new Assignment object.

Parameters

cr	The "health" of the enemy, and also the amount of credits it rewards the player for killing it
timeToMove	The inverse of speed for the enemy, basically the amount of game ticks it takes to advance
name	The name of this enemy
sprite	The sprite used for this enemy

13.1.2.2 ∼Assignment()

```
virtual Assignment::∼Assignment ( )  [virtual], [default]
```

A virtual destructor.

13.1.3 Member Function Documentation

13.1.3.1 Advance()

```
bool Assignment::Advance ( )
```

Tells if the enemy must be moved forward Returns true if the enemy moves to the next place.

Returns

bool

13.1.3.2 CrLeft()

```
uint32_t Assignment::CrLeft ( ) const
```

Tells the amount of health the enemy has left.

Returns

uint32_t

13.1.3.3 GetCredits()

```
uint32_t Assignment::GetCredits ( ) const
```

Totalt amount of credits the enemy is worth.

Returns

uint32_t

13.1.3.4 IsAlive()

```
bool Assignment::IsAlive ( ) const
```

Tells whether this enemy is still alive or not.

Returns

bool

13.1.3.5 MovedLastTick()

```
bool Assignment::MovedLastTick ( ) const
```

Tells if the enemy moved during the last tick For animation purposes.

Returns

bool

13.1.3.6 TakeDmg()

```
uint32_t Assignment::TakeDmg (
             uint32_t dmg,
             std::list< Assignment * > & location ) [virtual]
```

Makes the enemy take damage Splitting enemies hurt the attacking tower when they die. The return value tells the amount of this damage. The method is overridden in subclass Degree.

Parameters

dmg	The amount of damage taken
location	A reference to the list of enemies at the location of this enemy. Used by Degree to spawn its descendants there

Returns

uint32_t

Reimplemented in Degree.

13.1.4 Friends And Related Function Documentation

13.1.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const Assignment & as )  [friend]
```

Overload for stream operator << for debugging.

13.1.5 Member Data Documentation

13.1.5.1 m_curCr

```
uint32_t Assignment::m_curCr  [protected]
```

13.1.5.2 m_maxCr

```
uint32_t Assignment::m_maxCr  [protected]
```

13.1.5.3 m_movedLastTick

```
bool Assignment::m_movedLastTick  [protected]
```

13.1.5.4 m_timeRemainder

```
uint32_t Assignment::m_timeRemainder  [protected]
```

13.1.5.5 m_timeToMove

```
uint32_t Assignment::m_timeToMove  [protected]
```

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

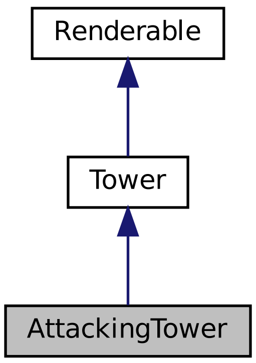
- src/assignment.hpp
- src/assignment.cpp

13.2 AttackingTower Class Reference

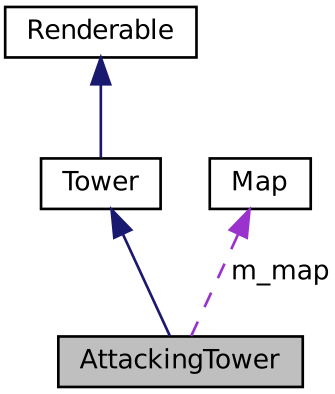
A class for the offensive towers.

```
#include <attacking_tower.hpp>
```

Inheritance diagram for AttackingTower:



Collaboration diagram for AttackingTower:



Public Member Functions

- AttackingTower (uint32_t power, uint32_t range, uint32_t health, uint32_t upgCost, const std::pair< int32_t, int32_t > &coords, const Map &map, const std::string &name, const std::vector< sf::Sprite > &sprites)

  Construct a new Attacking Tower object.
- ∼AttackingTower ()=default

  Default destructor.
- void Attack (std::vector< std::list< Assignment ∗ >> &enemies, std::list< std::pair< std::pair< int32_↩ t, int32_t >, std::pair< int32_t, int32_t >>> &attackCollection)

  Performs an attack against one enemy The tower goes through the possible targetable locations in m_inRangeInd, starting from the one closest to end and when it finds a living enemy in one of the locations, it attacks that. After performing an attack, it clears the buffs It also adds the attacks which it performs to a collection given as a reference.
- void ApplyBuff (float b)

  Used by the buffing towers to apply a buff In case multiple buffing towers buff a single tower, the buffs stack additively, not multiplicatively.
- bool IsFunctional () const

  Tells whether the tower is functional or destroyed.
- void Heal (uint32_t h)

  Used by the healing towers to heal other towers Will not heal over the maximum health.
- bool IsUpgradeable (uint32_t money) const

  Tells whether or not this tower can be upgraded with the corrent amount of money Also checks that this tower is not already max level.
- uint32_t GetUpgradeCost () const

  Tells the upgrade cost.
- uint32_t Upgrade ()

  Can be used to upgrade the tower to the next level. Assumes that tower is upgradeable.

Static Public Member Functions

- static AttackingTower ∗ Freshman (const std::pair< int32_t, int32_t > &coords, const Map &map)

  Static function to create a specific tower.
- static AttackingTower ∗ Teekkari (const std::pair< int32_t, int32_t > &coords, const Map &map)

  Static function to create a specific tower.
- static AttackingTower ∗ Bachelor (const std::pair< int32_t, int32_t > &coords, const Map &map)

  Static function to create a specific tower.
- static AttackingTower ∗ Master (const std::pair< int32_t, int32_t > &coords, const Map &map)

  Static function to create a specific tower.
- static AttackingTower ∗ Doctor (const std::pair< int32_t, int32_t > &coords, const Map &map)

  Static function to create a specific tower.

Private Member Functions

- void Priv_UpdateRange (uint32_t newRange)

  Private. Used to update the m_inRangeInd.

Private Attributes

- uint32_t m_basePower
- uint32_t m_maxHealth
- uint32_t m_health
- uint32_t m_upgCost
- uint32_t m_level
- float m_buffs
- std::vector< uint32_t > m_inRangeInd
- const Map & m_map

Friends

- std::ostream & operator<< (std::ostream &os, const AttackingTower &at)

  Overload for the stream output operator.

Additional Inherited Members

13.2.1 Detailed Description

A class for the offensive towers.

13.2.2 Constructor & Destructor Documentation

13.2.2.1 AttackingTower()

```
AttackingTower::AttackingTower (
             uint32_t power,
             uint32_t range,
             uint32_t health,
             uint32_t upgCost,
             const std::pair< int32_t, int32_t > & coords,
             const Map & map,
             const std::string & name,
             const std::vector< sf::Sprite > & sprites )
```

Construct a new Attacking Tower object.

Parameters

power	The amount of damage this tower can do to an enemy
range	The basic range
health	The health of this tower
upgCost	The cost of upgrading this tower
coords	The coordinates of the tower
map	A reference to the map used for the game, this is needed to know the enemy path and find attackable locations
name	The name of the tower
sprites	A collection of the sprites of different levels of this tower

13.2.2.2 ∼AttackingTower()

```
AttackingTower::∼AttackingTower ( )  [default]
```

Default destructor.

13.2.3 Member Function Documentation

13.2.3.1 ApplyBuff()

```
void AttackingTower::ApplyBuff (
             float b )
```

Used by the buffing towers to apply a buff In case multiple buffing towers buff a single tower, the buffs stack additively, not multiplicatively.

Parameters

b	The buff amount as a decimal number (e.g. 20% buff is 0.2f)

13.2.3.2 Attack()

```
void AttackingTower::Attack (
            std::vector< std::list< Assignment * >> & enemies,
            std::list< std::pair< std::pair< int32_t, int32_t >, std::pair< int32_t, int32←
_t >>> & attackCollection )
```

Performs an attack against one enemy The tower goes through the possible targetable locations in m_inRangeInd, starting from the one closest to end and when it finds a living enemy in one of the locations, it attacks that. After performing an attack, it clears the buffs It also adds the attacks which it performs to a collection given as a reference.

Parameters

enemies	A reference to the map of enemies in different coordinates
attackCollection	A reference to the collection where the attacks happening during the round are stored

13.2.3.3 Bachelor()

```
AttackingTower * AttackingTower::Bachelor (
            const std::pair< int32_t, int32_t > & coords,
            const Map & map ) [static]
```

Static function to create a specific tower.

Parameters

coords	Where the tower is placed
map	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.4 Doctor()

```
AttackingTower * AttackingTower::Doctor (
            const std::pair< int32_t, int32_t > & coords,
            const Map & map ) [static]
```

Static function to create a specific tower.

Parameters

coords	Where the tower is placed
map	A const ref to the map the tower is placed on

Returns

AttackingTower∗

13.2.3.5 Freshman()

```
AttackingTower * AttackingTower::Freshman (
            const std::pair< int32_t, int32_t > & coords,
            const Map & map )  [static]
```

Static function to create a specific tower.

Parameters

coords	Where the tower is placed
map	A const ref to the map the tower is placed on

Returns

AttackingTower∗

13.2.3.6 GetUpgradeCost()

```
uint32_t AttackingTower::GetUpgradeCost ( ) const
```

Tells the upgrade cost.

Returns

uint32_t

13.2.3.7 Heal()

```
void AttackingTower::Heal (
            uint32_t h )
```

Used by the healing towers to heal other towers Will not heal over the maximum health.

Parameters

h	The heal amount

13.2.3.8 IsFunctional()

```
bool AttackingTower::IsFunctional ( ) const
```

Tells whether the tower is functional or destroyed.

Returns

bool

13.2.3.9 IsUpgradeable()

```
bool AttackingTower::IsUpgradeable (
            uint32_t money ) const  [virtual]
```

Tells whether or not this tower can be upgraded with the corrent amount of money Also checks that this tower is not already max level.

Returns

bool

Implements Tower.

13.2.3.10 Master()

```
AttackingTower * AttackingTower::Master (
            const std::pair< int32_t, int32_t > & coords,
            const Map & map )  [static]
```

Static function to create a specific tower.

Parameters

coords	Where the tower is placed
map	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.11 Priv_UpdateRange()

```
void AttackingTower::Priv_UpdateRange (
             uint32_t newRange )  [private]
```

Private. Used to update the m_inRangeInd.

Parameters

newRange	The new range of the tower

13.2.3.12 Teekkari()

```
AttackingTower * AttackingTower::Teekkari (
             const std::pair< int32_t, int32_t > & coords,
             const Map & map )  [static]
```

Static function to create a specific tower.

Parameters

coords	Where the tower is placed
map	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.13 Upgrade()

```
uint32_t AttackingTower::Upgrade ( )
```

Can be used to upgrade the tower to the next level. Assumes that tower is upgradeable.

Returns

uint32_t The cost of the upgrade

13.2.4 Friends And Related Function Documentation

13.2.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const AttackingTower & at )  [friend]
```

Overload for the stream output operator.

13.2.5 Member Data Documentation

13.2.5.1 m_basePower

```
uint32_t AttackingTower::m_basePower  [private]
```

13.2.5.2 m_buffs

```
float AttackingTower::m_buffs  [private]
```

13.2.5.3 m_health

```
uint32_t AttackingTower::m_health  [private]
```

13.2.5.4 m_inRangeInd

```
std::vector<uint32_t> AttackingTower::m_inRangeInd  [private]
```

13.2.5.5 m_level

```
uint32_t AttackingTower::m_level  [private]
```

13.2.5.6 m_map

```
const Map& AttackingTower::m_map  [private]
```

13.2.5.7 m_maxHealth

```
uint32_t AttackingTower::m_maxHealth  [private]
```

13.2.5.8 m_upgCost

```
uint32_t AttackingTower::m_upgCost  [private]
```

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

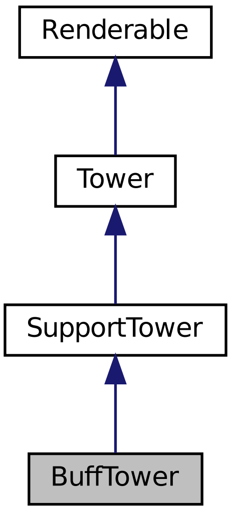
- src/attacking_tower.hpp
- src/attacking_tower.cpp

13.3 BuffTower Class Reference

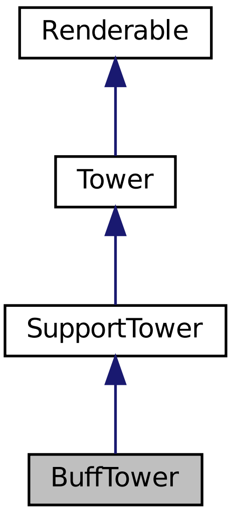
A tower which buffs attacking towers making the do more damage.

```
#include <support_towers.hpp>
```

Inheritance diagram for BuffTower:



Collaboration diagram for BuffTower:



Public Member Functions

- BuffTower (uint32_t range, const std::pair< int32_t, int32_t > &coords, float buffStrength, const std::string &name, const std::vector< sf::Sprite > &sprites)

    Construct a new Buff Tower object.
- ∼BuffTower ()=default

    Default destructor.
- void Act (std::list< AttackingTower ∗ > &towers)

    Goes trough the attacking towers in the game and applies buff to them.

Private Attributes

- float m_buffStrength

Additional Inherited Members

13.3.1 Detailed Description

A tower which buffs attacking towers making the do more damage.

13.3.2 Constructor & Destructor Documentation

13.3.2.1 BuffTower()

```
BuffTower::BuffTower (
            uint32_t range,
            const std::pair< int32_t, int32_t > & coords,
            float buffStrength,
            const std::string & name,
            const std::vector< sf::Sprite > & sprites )
```

Construct a new Buff Tower object.

Parameters

range	The basic range
coords	The coordinates of the tower
buffStrength	The amount of buff the tower gives, for example 20% buff is 0.2f
name	The name of the tower
sprites	Support towers cannot be upgraded, so the collection only has one sprite for them

13.3.2.2 ∼BuffTower()

```
BuffTower::∼BuffTower ( )  [default]
```

Default destructor.

13.3.3 Member Function Documentation

13.3.3.1 Act()

```
void BuffTower::Act (
            std::list< AttackingTower * > & towers )  [virtual]
```

Goes trough the attacking towers in the game and applies buff to them.

Parameters

towers	The towers on the game board

Implements SupportTower.

13.3.4 Member Data Documentation

13.3.4.1 m_buffStrength

```
float BuffTower::m_buffStrength [private]
```

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

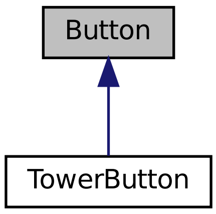
- src/support_towers.hpp
- src/support_towers.cpp

13.4 Button Class Reference

A class for the buttons in the game.

```
#include <button.hpp>
```

Inheritance diagram for Button:



Public Member Functions

- Button (std::string text, int32_t x, int32_t y, sf::Font &font)

  Construct a new Button object.
- virtual ∼Button ()=default

  A virtual destructor.
- void addHighlight ()

  Adds a highlighted border for the button.
- void removeHighlight ()

  Removes the highlight.
- virtual void disableButton ()

  Makes the button grayed out.
- virtual void enableButton ()

  Restores the original look of the button.
- virtual void changeText (std::string text)

  Changes the text shown in the button.
- virtual void drawButton (sf::RenderWindow &window)

  Draws the button on the window.
- sf::FloatRect getGlobalBounds ()

  Get the bounds of this button.

Protected Attributes

- sf::RectangleShape m_button
- sf::Text m_text
- sf::Font & m_font

13.4.1 Detailed Description

A class for the buttons in the game.

13.4.2 Constructor & Destructor Documentation

13.4.2.1 Button()

```
Button::Button (
            std::string text,
            int32_t x,
            int32_t y,
            sf::Font & font )
```

Construct a new Button object.

Parameters

text	What do display on the button
x	The x-coordinate of the upper left corner
y	The y-coordinate of the upper left corner
font	The font used by the text

13.4.2.2 ∼Button()

```
virtual Button::∼Button ( ) [virtual], [default]
```

A virtual destructor.

13.4.3 Member Function Documentation

13.4.3.1 addHighlight()

```
void Button::addHighlight ( )
```

Adds a highlighted border for the button.

13.4.3.2 changeText()

```
void Button::changeText (
             std::string text )  [virtual]
```

Changes the text shown in the button.

Parameters

text	The new text

13.4.3.3 disableButton()

```
void Button::disableButton ( )  [virtual]
```

Makes the button grayed out.

Reimplemented in TowerButton.

13.4.3.4 drawButton()

```
void Button::drawButton (
             sf::RenderWindow & window )  [virtual]
```

Draws the button on the window.

Parameters

window	A ref to the window where to draw

Reimplemented in TowerButton.

13.4.3.5 enableButton()

```
void Button::enableButton ( )  [virtual]
```

Restores the original look of the button.

Reimplemented in TowerButton.

13.4.3.6 getGlobalBounds()

```
sf::FloatRect Button::getGlobalBounds ( )
```

Get the bounds of this button.

Returns

sf::FloatRect

13.4.3.7 removeHighlight()

```
void Button::removeHighlight ( )
```

Removes the highlight.

13.4.4 Member Data Documentation

13.4.4.1 m_button

```
sf::RectangleShape Button::m_button  [protected]
```

13.4.4.2 m_font

```
sf::Font& Button::m_font  [protected]
```

13.4.4.3 m_text

```
sf::Text Button::m_text  [protected]
```

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

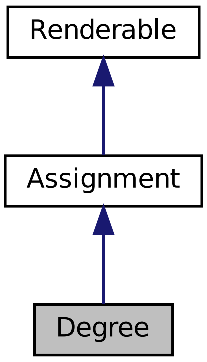
- src/button.hpp
- src/button.cpp

13.5 Degree Class Reference

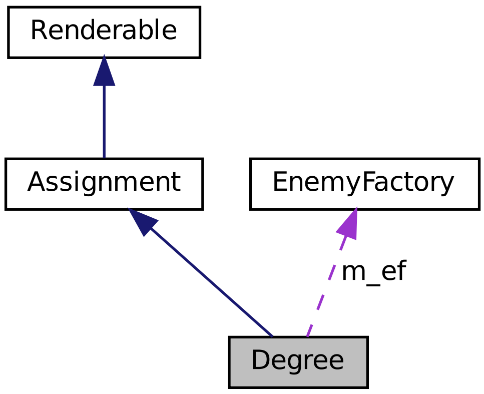
A more advanced type of enemy Splits into multiple other enemies upon death The descendants are stored in the m_descendants collection as pairs where the first one is the type and the second one is the amount.

```
#include <degree.hpp>
```

Inheritance diagram for Degree:



Collaboration diagram for Degree:



Public Member Functions

- Degree (uint32_t cr, uint32_t timeToMove, const std::string &name, const sf::Sprite &sprite, const EnemyFactory &ef, const std::list< std::pair< Enemy, uint32_t >> &decendants)

Construct a new Degree object.
- ∼Degree ()=default

  Default destructor.
- uint32_t TakeDmg (uint32_t dmg, std::list< Assignment ∗ > &location)

  Makes the enemy take damage.

Private Attributes

- const std::list< std::pair< Enemy, uint32_t > > m_decendants
- const EnemyFactory & m_ef

Additional Inherited Members

13.5.1 Detailed Description

A more advanced type of enemy Splits into multiple other enemies upon death The descendants are stored in the m_descendants collection as pairs where the first one is the type and the second one is the amount.

13.5.2 Constructor & Destructor Documentation

13.5.2.1 Degree()

```
Degree::Degree (
            uint32_t cr,
            uint32_t timeToMove,
            const std::string & name,
            const sf::Sprite & sprite,
            const EnemyFactory & ef,
            const std::list< std::pair< Enemy, uint32_t >> & decendants )
```

Construct a new Degree object.

Parameters

cr	The "health" of the enemy, and also the amount of credits it rewards the player for killing it
timeToMove	The inverse of speed for the enemy, basically the amount of game ticks it takes to advance
name	The name of this enemy
ef	The enemyfactory used for creating this enemy. Is used to spawn the descendants when this dies
decendants	The types and amounts of descendants

13.5.2.2 ∼Degree()

```
Degree::∼Degree ( ) [default]
```

Default destructor.

13.5.3 Member Function Documentation

13.5.3.1 TakeDmg()

```
uint32_t Degree::TakeDmg (
            uint32_t dmg,
            std::list< Assignment * > & location )  [virtual]
```

Makes the enemy take damage.

Parameters

dmg	Amount of damage the enemy takes
location	A ref to the location of this enemy, spawns the descendants there

Returns

uint32_t The amount of damage the enemy gives the tower upon splitting (== m_maxCr / 4)

Reimplemented from Assignment.

13.5.4 Member Data Documentation

13.5.4.1 m_decendants

```
const std::list<std::pair<Enemy, uint32_t> > Degree::m_decendants  [private]
```

13.5.4.2 m_ef

```
const EnemyFactory& Degree::m_ef  [private]
```

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

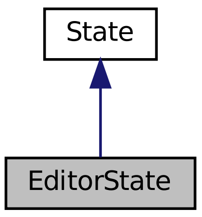
- src/degree.hpp
- src/degree.cpp

13.6 EditorState Class Reference

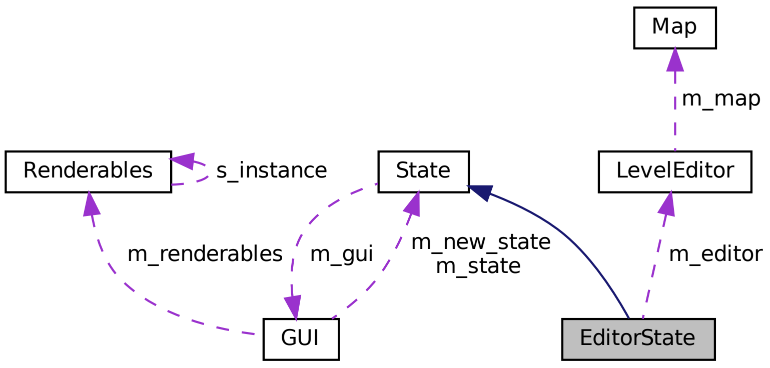
A gamestate run by GUI corresponding to the level editor.

```
#include <editorstate.hpp>
```

Inheritance diagram for EditorState:



Collaboration diagram for EditorState:



Public Member Functions

- EditorState (GUI &gui, sf::RenderWindow &window, std::string &mapPath)

  Construct a new Menu State object.
- ∼EditorState ()

  Destroy the Menu State object Frees the buttons.
- void Run ()

  Run and draw the Menu state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Has options to select difficulty and map file. Has buttons to game editor and to play.) Exits the loop and calls GUIs MoveToGameState(int score) when the user presses "Play".

Private Member Functions

- void Priv_PollEvents ()

  Polls the events that have happened in GUI.
- void Priv_Draw ()

  Draws the frame to the screen.

Private Attributes

- LevelEditor m_editor
- std::vector< sf::Sprite > m_mapTileSprites
- std::map< int, Button * > m_buttons
- sf::RectangleShape m_selectedShape
- sf::Text m_validated
- sf::Text m_unvalidated
- sf::Text m_instructions
- std::string m_mapPath
- int32_t m_selX
- int32_t m_selY
- bool m_drawSelectedShape
- int32_t m_selectedButton

Additional Inherited Members

13.6.1 Detailed Description

A gamestate run by GUI corresponding to the level editor.

13.6.2 Constructor & Destructor Documentation

13.6.2.1 EditorState()

```
EditorState::EditorState (
            GUI & gui,
            sf::RenderWindow & window,
            std::string & mapPath )
```

Construct a new Menu State object.

Parameters

gui	A ref to the GUI used
window	A ref to the window used
mapPath	The map file which is edited

13.6.2.2 ∼EditorState()

```
EditorState::∼EditorState ( )
```

Destroy the Menu State object Frees the buttons.

13.6.3 Member Function Documentation

13.6.3.1 Priv_Draw()

```
void EditorState::Priv_Draw ( )  [private]
```

Draws the frame to the screen.

13.6.3.2 Priv_PollEvents()

```
void EditorState::Priv_PollEvents ( )  [private]
```

Polls the events that have happened in GUI.

13.6.3.3 Run()

```
void EditorState::Run ( )  [virtual]
```

Run and draw the Menu state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Has options to select difficulty and map file. Has buttons to game editor and to play.) Exits the loop and calls GUIs MoveToGameState(int score) when the user presses "Play".

Implements State.

13.6.4 Member Data Documentation

13.6.4.1 m_buttons

```
std::map<int, Button*> EditorState::m_buttons  [private]
```

13.6.4.2 m_drawSelectedShape

```
bool EditorState::m_drawSelectedShape  [private]
```

13.6.4.3 m_editor

```
LevelEditor EditorState::m_editor  [private]
```

13.6.4.4 m_instructions

```
sf::Text EditorState::m_instructions  [private]
```

13.6.4.5 m_mapPath

```
std::string EditorState::m_mapPath  [private]
```

13.6.4.6 m_mapTileSprites

```
std::vector<sf::Sprite> EditorState::m_mapTileSprites  [private]
```

13.6.4.7 m_selectedButton

```
int32_t EditorState::m_selectedButton  [private]
```

13.6.4.8 m_selectedShape

```
sf::RectangleShape EditorState::m_selectedShape  [private]
```

13.6.4.9 m_selX

```
int32_t EditorState::m_selX  [private]
```

13.6.4.10 m_selY

```
int32_t EditorState::m_selY  [private]
```

13.6.4.11 m_unvalidated

```
sf::Text EditorState::m_unvalidated  [private]
```

13.6.4.12 m_validated

```
sf::Text EditorState::m_validated  [private]
```

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

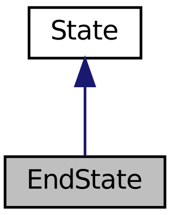
- src/states/editorstate.hpp
- src/states/editorstate.cpp

13.7 EndState Class Reference

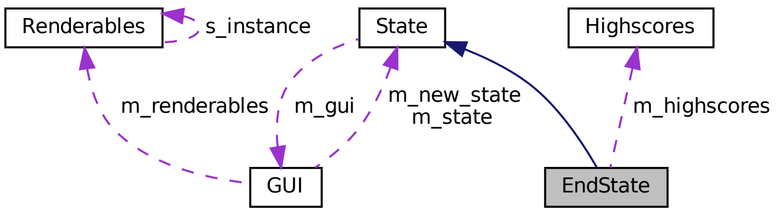
A gamestate run by GUI corresponding to the game over screen.

`#include <endstate.hpp>`

Inheritance diagram for EndState:



Collaboration diagram for EndState:



Public Member Functions

- EndState (GUI &gui, sf::RenderWindow &window, uint32_t score, Difficulty difficulty)

  Construct a new End State object.
- ∼EndState ()

  Destroy the End State object Frees the buttons.
- void Run ()

  Run and draw the Game Over state of the software. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Allows saving the score, displays top scores and has a button to main menu.) Exits the loop and calls GUIs MoveToMenuState(int score) when the user presses "Main menu".

Private Member Functions

- void Priv_PollEvents ()

  Polls the events that have happened in GUI.
- void Priv_Draw ()

  Draws the frame to the screen.

Private Attributes

- uint32_t m_score
- Difficulty m_difficulty
- std::map< int32_t, Button ∗ > m_buttons
- Highscores m_highscores
- std::string m_input
- sf::Text m_text_score
- sf::Text m_text_name
- sf::Text m_text_highscores
- sf::Font m_font

Additional Inherited Members

13.7.1 Detailed Description

A gamestate run by GUI corresponding to the game over screen.

13.7.2 Constructor & Destructor Documentation

13.7.2.1 EndState()

```
EndState::EndState (
            GUI & gui,
            sf::RenderWindow & window,
            uint32_t score,
            Difficulty difficulty )
```

Construct a new End State object.

Parameters

gui	A ref to the GUI
window	A ref to the window to draw on
score	The score that was achieved in game
difficulty	The difficulty that the game was played on

13.7.2.2 ∼EndState()

```
EndState::∼EndState ( )
```

Destroy the End State object Frees the buttons.

13.7.3 Member Function Documentation

13.7.3.1 Priv_Draw()

```
void EndState::Priv_Draw ( )  [private]
```

Draws the frame to the screen.

13.7.3.2 Priv_PollEvents()

```
void EndState::Priv_PollEvents ( )  [private]
```

Polls the events that have happened in GUI.

13.7.3.3 Run()

```
void EndState::Run ( )  [virtual]
```

Run and draw the Game Over state of the software. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Allows saving the score, displays top scores and has a button to main menu.) Exits the loop and calls GUIs MoveToMenuState(int score) when the user presses "Main menu".

Implements State.

13.7.4 Member Data Documentation

13.7.4.1 m_buttons

`std::map<int32_t, Button*> EndState::m_buttons [private]`

13.7.4.2 m_difficulty

`Difficulty EndState::m_difficulty [private]`

13.7.4.3 m_font

`sf::Font EndState::m_font [private]`

13.7.4.4 m_highscores

`Highscores EndState::m_highscores [private]`

13.7.4.5 m_input

`std::string EndState::m_input [private]`

13.7.4.6 m_score

`uint32_t EndState::m_score [private]`

13.7.4.7 m_text_highscores

`sf::Text EndState::m_text_highscores [private]`

13.7.4.8 m_text_name

```
sf::Text EndState::m_text_name  [private]
```

13.7.4.9 m_text_score

```
sf::Text EndState::m_text_score  [private]
```

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

- src/states/endstate.hpp
- src/states/endstate.cpp

13.8 EnemyFactory Class Reference

A class which handles the logic about what enemies come and how much of during each round.

```
#include <enemy_factory.hpp>
```

Public Member Functions

- EnemyFactory (Difficulty diff)

  Constructs a new Enemy Factory object.
- EnemyFactory (const EnemyFactory &other)=delete

  Delete copy constructor.
- EnemyFactory & operator= (const EnemyFactory &other)=delete

  Delete assignment operator.
- ∼EnemyFactory ()

  Destroys the Enemy Factory object, if there were some enemies left in the storage, frees them.
- uint32_t NextRoundInit ()

  Initializes the enemies of the next round After calling this, the enemies which come each tick can be obtained using NextTick() Also frees the previously allocated enemies if the previous round for some reason did not finnish.
- std::list< Assignment ∗ > NextTick ()

  Used to get the enemies which appear on the next game tick Removes them from the objects own collection, so the resposibility is transferred to the object which calls the function The max amound of enemies during the tick for round n is approximately base-2-log(a_n) + 1 where a_n is the element of the sequence used to determine the types present.
- bool EnemiesLeft () const

  Tells if there are still enemies left which have not been handed to the Game logic unit.
- Assignment ∗ CreateEnemy (Enemy e) const

  Creates an Enemy object Can be used either by this class itself or by the game core to spawn the additional enemies at some location.
- Difficulty GetDifficulty () const

  Tells the set difficulty.

Private Member Functions

- uint32_t Priv_NextNum ()

  Private. Uses m_nums to calculate the next one in the sequence.
- void Priv_Free ()

  Private Used to free the enemies which are left.

Private Attributes

- Difficulty m_diff
- uint32_t m_round
- uint32_t m_nums [3] = {1, 1, 1}
- std::list< Assignment ∗ > m_roundEnemies
- uint32_t m_batchSizes [9] = {0}
- uint32_t m_batchSizeDeltas [9] = {10, 5, 3, 2, 2, 2, 1, 1, 1}

Friends

- std::ostream & operator<< (std::ostream &os, const EnemyFactory &ef)

  An overload for the stream operator for debugging purposes.

13.8.1 Detailed Description

A class which handles the logic about what enemies come and how much of during each round.

The logic as in "Which enemies are present in the round?" is as follows: In addition to the round number m_round, we have a Fibonacci-like sequence, where a_1 = a_2 = a_3 = 1 and after that a_n = a_(n-2) + a_(n-3). Then, from the binary representation of a_(m_round) we look at which bits are set, starting from the least significant bits, and the i:th bit being set means that the i:th enemy type is present. For example on 6th round a_6 = 3, which is ...0011 in binary, so enemies 0 (Homework) and 1 (Essay) are present. The numbers a_n in the sequence may also overflow at some point, but that is ok. The logic as in "How many enemies come during the round?" is based on the array m_batchSizes. After each round an enemy type is present, the batch size will also be incremented according to the deltas.

This class allocates the enemies dynamically, and then the game logic class is responsible for freeing the memory when enemies either reach the end or die

13.8.2 Constructor & Destructor Documentation

13.8.2.1 EnemyFactory() [1/2]

```
EnemyFactory::EnemyFactory (
             Difficulty diff )
```

Constructs a new Enemy Factory object.

Parameters

diff	The difficulty of the game, scales the HP of the enemies

13.8.2.2 EnemyFactory() [2/2]

```
EnemyFactory::EnemyFactory (
            const EnemyFactory & other )   [delete]
```

Delete copy constructor.

13.8.2.3 ∼EnemyFactory()

```
EnemyFactory::∼EnemyFactory ( )
```

Destroys the Enemy Factory object, if there were some enemies left in the storage, frees them.

13.8.3 Member Function Documentation

13.8.3.1 CreateEnemy()

```
Assignment * EnemyFactory::CreateEnemy (
            Enemy e ) const
```

Creates an Enemy object Can be used either by this class itself or by the game core to spawn the additional enemies at some location.

Returns

Assignment∗ A dynamically allocated enemy

13.8.3.2 EnemiesLeft()

```
bool EnemyFactory::EnemiesLeft ( ) const
```

Tells if there are still enemies left which have not been handed to the Game logic unit.

Returns

bool

13.8.3.3 GetDifficulty()

```
Difficulty EnemyFactory::GetDifficulty ( ) const
```

Tells the set difficulty.

Returns

Difficulty

13.8.3.4 NextRoundInit()

```
uint32_t EnemyFactory::NextRoundInit ( )
```

Initializes the enemies of the next round After calling this, the enemies which come each tick can be obtained using NextTick() Also frees the previously allocated enemies if the previous round for some reason did not finnish.

Returns

uint32_t The round number which is starting

13.8.3.5 NextTick()

```
std::list< Assignment * > EnemyFactory::NextTick ( )
```

Used to get the enemies which appear on the next game tick Removes them from the objects own collection, so the resposibility is transferred to the object which calls the function The max amound of enemies during the tick for round n is approximately base-2-log(a_n) + 1 where a_n is the element of the sequence used to determine the types present.

Returns

std::list<Assignment*>

13.8.3.6 operator=()

```
EnemyFactory& EnemyFactory::operator= (
             const EnemyFactory & other )   [delete]
```

Delete assignment operator.

13.8.3.7 Priv_Free()

```
void EnemyFactory::Priv_Free ( )  [private]
```

Private Used to free the enemies which are left.

13.8.3.8 Priv_NextNum()

```
uint32_t EnemyFactory::Priv_NextNum ( )  [private]
```

Private. Uses m_nums to calculate the next one in the sequence.

Returns

uint32_t The next number in the sequence

13.8.4 Friends And Related Function Documentation

13.8.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const EnemyFactory & ef )  [friend]
```

An overload for the stream operator for debugging purposes.

13.8.5 Member Data Documentation

13.8.5.1 m_batchSizeDeltas

```
uint32_t EnemyFactory::m_batchSizeDeltas[9] = {10, 5, 3, 2, 2, 2, 1, 1, 1}  [private]
```

13.8.5.2 m_batchSizes

```
uint32_t EnemyFactory::m_batchSizes[9] = {0}  [private]
```

13.8.5.3 m_diff

```
Difficulty EnemyFactory::m_diff  [private]
```

13.8.5.4 m_nums

```
uint32_t EnemyFactory::m_nums[3] = {1, 1, 1}  [private]
```

13.8.5.5 m_round

```
uint32_t EnemyFactory::m_round  [private]
```

13.8.5.6 m_roundEnemies

```
std::list<Assignment*> EnemyFactory::m_roundEnemies  [private]
```

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

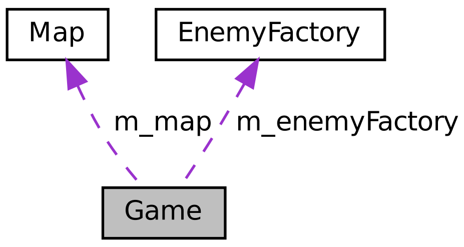
- src/enemy_factory.hpp
- src/enemy_factory.cpp

13.9 Game Class Reference

A class which holds the logic of the game.

```
#include <game.hpp>
```

Collaboration diagram for Game:



Public Member Functions

- Game (uint32_t mapWidth, uint32_t mapHeigth, const std::string &filename, Difficulty difficulty)

  Construct a new Game object.
- ∼Game ()

  Destroy the Game object In case of a game over / quit the enemies and towers present must be freed.
- Game (const Game &other)=delete

  Delete copy constructor.
- Game & operator= (const Game &other)=delete

  Delete assignment operator.
- uint32_t StartNextRound ()

  Used to start the next round Calls the enemyfactory to initialize its own state such that the round can start.
- bool EnemyTurn ()

  Advances the enemies.
- bool TowerTurn ()

  Makes the towers attack the enemies If enemies died during the attacking, returns true (for sound effects)
- bool RoundIsFinished ()

  Used to check if the round is still ongoing. SHOULD be called only AFTER StartNextRound()!
- std::vector< std::list< Assignment ∗ > > & GetEnemies ()

  Get the Enemies in the game now Method for getting enemies and their coordinates for GUI to draw.
- const std::list< std::pair< std::pair< int32_t, int32_t >, std::pair< int32_t, int32_t > > > & GetAttacks () const

  Gives information about which attacks happened during the turn.
- const Map & GetMap () const

  Gets a const ref version of the map used.
- bool AddTower (Tower ∗t)

  For adding a tower to the Game. Only for the text_based_test.cpp.
- bool IsActionPossible (const std::pair< int32_t, int32_t > &coords, Action a) const

  Used by GUI states to check what can be done.
- void CreateTower (const std::pair< int32_t, int32_t > &coords, TowerType t)

  Create a Tower object in the game Uses the coordinates and enumeration to place a tower on the playing field Reduces the players money.
- void UpgradeTower (const std::pair< int32_t, int32_t > &coords)

  Upgrades tower and reduces player's money.
- void DestroyTower (const std::pair< int32_t, int32_t > &coords)

  Destroys tower.
- const std::list< AttackingTower ∗ > & GetAttackingTowers () const

  Get a ref to the Attacking Towers for drawing them.
- const std::list< SupportTower ∗ > & GetSupportTowers () const

  Get a ref to the Support Towers for drawing them.
- const Tower ∗ GetTower (const std::pair< int32_t, int32_t > &coords) const

  Get a pointer to Tower in a cell The function will return nullptr if no tower is at the desired location.
- uint32_t GetScore () const

  Calculates player's score. Total money earned is multiplied by 100 and then divided by the length of the path. When the path is shorter there is shorter time to defeat enemies so player get more points.
- uint32_t GetMoney () const

  Tells the player's current money.
- uint32_t GetHealth () const

  Tells the player's current health.
- Difficulty GetDifficulty () const

  Tells the difficulty of the current game.

Private Attributes

- uint32_t m_playerHealth
- uint32_t m_score
- uint32_t m_money
- Map m_map
- EnemyFactory m_enemyFactory
- std::list< AttackingTower * > m_attakingTowers
- std::list< SupportTower * > m_supportingTowers
- std::vector< std::list< Assignment * > > m_enemies
- std::list< std::pair< std::pair< int32_t, int32_t >, std::pair< int32_t, int32_t > > > m_tickAttacks

Friends

- std::ostream & operator<< (std::ostream &os, const Game &game)

  Overload for the stream output operator.

13.9.1 Detailed Description

A class which holds the logic of the game.

13.9.2 Constructor & Destructor Documentation

13.9.2.1 Game() [1/2]

```
Game::Game (
            uint32_t mapWidth,
            uint32_t mapHeigth,
            const std::string & filename,
            Difficulty difficulty )
```

Construct a new Game object.

Parameters

mapWidth	The width of the map (should be 30)
mapHeigth	The height of the map (should be 20)
filename	The file where the map's text representation is
difficulty	The difficulty for the game

13.9.2.2 ∼Game()

```
Game::∼Game ( )
```

Destroy the Game object In case of a game over / quit the enemies and towers present must be freed.

13.9.2.3 Game() [2/2]

```
Game::Game (
            const Game & other )   [delete]
```

Delete copy constructor.

13.9.3 Member Function Documentation

13.9.3.1 AddTower()

```
bool Game::AddTower (
            Tower * t )
```

For adding a tower to the Game. Only for the text_based_test.cpp.

Parameters

t	A pointer to the dynamically allocated tower Will fail if the Tower is not an instance of Attacking or supporting Tower

Returns

bool Whether the adding was successfull

13.9.3.2 CreateTower()

```
void Game::CreateTower (
            const std::pair< int32_t, int32_t > & coords,
            TowerType t )
```

Create a Tower object in the game Uses the coordinates and enumeration to place a tower on the playing field Reduces the players money.

Parameters

coords	The position where the tower needs to be created
t	An enumeration of the desired tower to build

13.9.3.3 DestroyTower()

```
void Game::DestroyTower (
             const std::pair< int32_t, int32_t > & coords )
```

Destroys tower.

Parameters

coords	position of tower to destroy

13.9.3.4 EnemyTurn()

```
bool Game::EnemyTurn ( )
```

Advances the enemies.

Returns

true: No game over

false: Game over

13.9.3.5 GetAttackingTowers()

```
const std::list< AttackingTower * > & Game::GetAttackingTowers ( ) const
```

Get a ref to the Attacking Towers for drawing them.

Returns

const std::list<AttackingTower∗>&

13.9.3.6 GetAttacks()

```
const std::list< std::pair< std::pair< int32_t, int32_t >, std::pair< int32_t, int32_t > >
> & Game::GetAttacks ( ) const
```

Gives information about which attacks happened during the turn.

Returns

const std::list<std::pair<std::pair<int32_t, int32_t>,std::pair<int32_t, int32_t>>>&

13.9.3.7 GetDifficulty()

```
Difficulty Game::GetDifficulty ( ) const
```

Tells the difficulty of the current game.

Returns

Difficulty

13.9.3.8 GetEnemies()

```
std::vector< std::list< Assignment * > > & Game::GetEnemies ( )
```

Get the Enemies in the game now Method for getting enemies and their coordinates for GUI to draw.

Returns

std::vector<std::list<Assignment*>>&

13.9.3.9 GetHealth()

```
uint32_t Game::GetHealth ( ) const
```

Tells the player's current health.

Returns

uint32_t

13.9.3.10 GetMap()

```
const Map & Game::GetMap ( ) const
```

Gets a const ref version of the map used.

Returns

const Map&

13.9.3.11 GetMoney()

```
uint32_t Game::GetMoney ( ) const
```

Tells the player's current money.

Returns

uint32_t

13.9.3.12 GetScore()

```
uint32_t Game::GetScore ( ) const
```

Calculates player's score. Total money earned is multiplied by 100 and then divided by the length of the path. When the path is shorter there is shorter time to defeat enemies so player get more points.

Returns

uint32_t player's score

13.9.3.13 GetSupportTowers()

```
const std::list< SupportTower * > & Game::GetSupportTowers ( ) const
```

Get a ref to the Support Towers for drawing them.

Returns

const std::list<SupportTower*>&

13.9.3.14 GetTower()

```
const Tower * Game::GetTower (
            const std::pair< int32_t, int32_t > & coords ) const
```

Get a pointer to Tower in a cell The function will return nullptr if no tower is at the desired location.

Parameters

coords	The xy-coordinates where we want to look

Returns

const Tower*

13.9.3.15 IsActionPossible()

```
bool Game::IsActionPossible (
            const std::pair< int32_t, int32_t > & coords,
            Action a ) const
```

Used by GUI states to check what can be done.

Parameters

coords	The grid coordinates
a	Enumeration telling the desired action

Returns

bool

13.9.3.16 operator=()

```
Game& Game::operator= (
            const Game & other )  [delete]
```

Delete assignment operator.

13.9.3.17 RoundIsFinished()

```
bool Game::RoundIsFinished ( )
```

Used to check if the round is still ongoing. SHOULD be called only AFTER StartNextRound()!

Returns

bool

13.9.3.18 StartNextRound()

```
uint32_t Game::StartNextRound ( )
```

Used to start the next round Calls the enemyfactory to initialize its own state such that the round can start.

Returns

uint32_t The number of the round starting

13.9.3.19 TowerTurn()

```
bool Game::TowerTurn ( )
```

Makes the towers attack the enemies If enemies died during the attacking, returns true (for sound effects)

Returns

bool

13.9.3.20 UpgradeTower()

```
void Game::UpgradeTower (
             const std::pair< int32_t, int32_t > & coords )
```

Upgrades tower and reduces player's money.

Parameters

coords	position of tower to upgrade

13.9.4 Friends And Related Function Documentation

13.9.4.1 operator<<

```
std::ostream& operator<< (
             std::ostream & os,
             const Game & game )   [friend]
```

Overload for the stream output operator.

13.9.5 Member Data Documentation

13.9.5.1 m_attakingTowers

`std::list<AttackingTower*> Game::m_attakingTowers [private]`

13.9.5.2 m_enemies

`std::vector<std::list<Assignment*> > Game::m_enemies [private]`

13.9.5.3 m_enemyFactory

`EnemyFactory Game::m_enemyFactory [private]`

13.9.5.4 m_map

`Map Game::m_map [private]`

13.9.5.5 m_money

`uint32_t Game::m_money [private]`

13.9.5.6 m_playerHealth

`uint32_t Game::m_playerHealth [private]`

13.9.5.7 m_score

`uint32_t Game::m_score [private]`

13.9.5.8 m_supportingTowers

```
std::list<SupportTower*> Game::m_supportingTowers [private]
```

13.9.5.9 m_tickAttacks

```
std::list<std::pair<std::pair<int32_t, int32_t>, std::pair<int32_t, int32_t> > > Game::m_↩
tickAttacks [private]
```

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

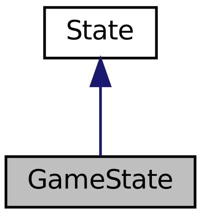
- src/game.hpp
- src/game.cpp

13.10 GameState Class Reference

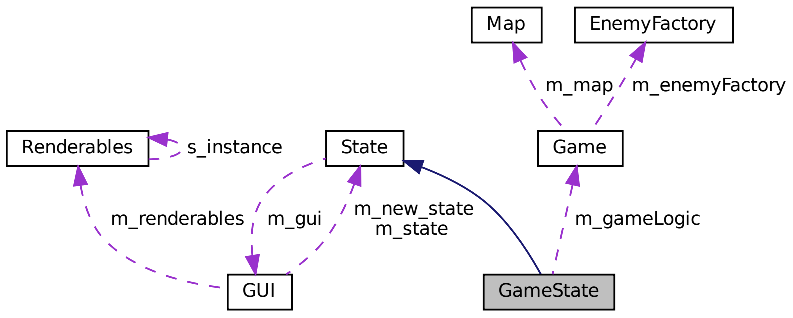
GameState class runs and draws the game part of the software.

```
#include <gamestate.hpp>
```

Inheritance diagram for GameState:



Collaboration diagram for GameState:



Public Member Functions

- GameState (GUI &gui, sf::RenderWindow &window, Difficulty difficulty, const std::string &filename)

  Construct a new Game State object.
- ∼GameState ()

  Destroy the Game State object Frees the buttons.
- void Run ()

  Run and draw the game state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Alternates between the building/upgrading phase and running the waves.) Exits the loop and calls GUIs MoveToEndState(int score) when the game is over.

Private Member Functions

- void Priv_PollEvents ()

  Polls the events that have happened in GUI.
- void Priv_Draw ()

  Draws the frame to the screen.
- void Priv_DrawBCG ()

  Draws the background to the window Assumes that the window has been properly cleared.
- void Priv_DrawMap ()

  Draws the map to the window. Assumes that the window has been properly cleared.
- void Priv_InitializeText (sf::Text &text, int32_t x, int32_t y)

  Initializes some text at set location.
- void Priv_ClearSpeedHighlights ()

  Clears highlights from the game speed buttons.
- void Priv_ChangeCircle (sf::CircleShape &circle, u_int32_t range)

  Sets a circle at desired position Used to draw ranges of towers.

Private Attributes

- bool m_gameOver
- bool m_buildPhase
- bool m_drawRange
- bool m_drawUpgradeRange
- uint32_t m_roundNum
- uint32_t m_gameSpeed
- int32_t m_frameInTick
- Game m_gameLogic
- std::vector< sf::Sprite > m_mapTileSprites
- std::map< int32_t, Button ∗ > m_buttons
- sf::RectangleShape m_selectedShape
- sf::Text m_scoreText
- sf::Text m_healthText
- sf::Text m_moneyText
- sf::Text m_roundNumText
- sf::CircleShape m_rangeCircle
- sf::CircleShape m_upgradeRange
- sf::CircleShape m_projectile
- int32_t m_selX
- int32_t m_selY

Additional Inherited Members

13.10.1 Detailed Description

GameState class runs and draws the game part of the software.

13.10.2 Constructor & Destructor Documentation

13.10.2.1 GameState()

```
GameState::GameState (
             GUI & gui,
             sf::RenderWindow & window,
             Difficulty difficulty,
             const std::string & filename )
```

Construct a new Game State object.

Parameters

gui	A ref to the GUI
window	A ref to the window
difficulty	The game difficulty
filename	The name of the map file

13.10.2.2 ∼GameState()

```
GameState::∼GameState ( )
```

Destroy the Game State object Frees the buttons.

13.10.3 Member Function Documentation

13.10.3.1 Priv_ChangeCircle()

```
void GameState::Priv_ChangeCircle (
             sf::CircleShape & circle,
             u_int32_t range )  [private]
```

Sets a circle at desired position Used to draw ranges of towers.

Parameters

circle	The cirle used in drawing
range	The range to draw

13.10.3.2 Priv_ClearSpeedHighlights()

```
void GameState::Priv_ClearSpeedHighlights ( )  [private]
```

Clears highlights from the game speed buttons.

13.10.3.3 Priv_Draw()

```
void GameState::Priv_Draw ( )  [private]
```

Draws the frame to the screen.

13.10.3.4 Priv_DrawBCG()

```
void GameState::Priv_DrawBCG ( )  [private]
```

Draws the background to the window Assumes that the window has been properly cleared.

13.10.3.5 Priv_DrawMap()

```
void GameState::Priv_DrawMap ( )  [private]
```

Draws the map to the window. Assumes that the window has been properly cleared.

13.10.3.6 Priv_InitializeText()

```
void GameState::Priv_InitializeText (
             sf::Text & text,
             int32_t x,
             int32_t y )  [private]
```

Initializes some text at set location.

Parameters

text	The text to initialize
x	x-coordinate
y	y-coordinate

13.10.3.7 Priv_PollEvents()

```
void GameState::Priv_PollEvents ( )  [private]
```

Polls the events that have happened in GUI.

13.10.3.8 Run()

```
void GameState::Run ( )  [virtual]
```

Run and draw the game state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Alternates between the building/upgrading phase and running the waves.) Exits the loop and calls GUIs MoveToEndState(int score) when the game is over.

Implements State.

13.10.4 Member Data Documentation

13.10.4.1 m_buildPhase

```
bool GameState::m_buildPhase  [private]
```

13.10.4.2 m_buttons

```
std::map<int32_t, Button*> GameState::m_buttons  [private]
```

13.10.4.3 m_drawRange

```
bool GameState::m_drawRange  [private]
```

13.10.4.4 m_drawUpgradeRange

```
bool GameState::m_drawUpgradeRange  [private]
```

13.10.4.5 m_frameInTick

```
int32_t GameState::m_frameInTick  [private]
```

13.10.4.6 m_gameLogic

```
Game GameState::m_gameLogic  [private]
```

13.10.4.7 m_gameOver

```
bool GameState::m_gameOver  [private]
```

13.10.4.8 m_gameSpeed

```
uint32_t GameState::m_gameSpeed  [private]
```

13.10.4.9 m_healthText

```
sf::Text GameState::m_healthText  [private]
```

13.10.4.10 m_mapTileSprites

```
std::vector<sf::Sprite> GameState::m_mapTileSprites  [private]
```

13.10.4.11 m_moneyText

```
sf::Text GameState::m_moneyText  [private]
```

13.10.4.12 m_projectile

```
sf::CircleShape GameState::m_projectile  [private]
```

13.10.4.13 m_rangeCircle

```
sf::CircleShape GameState::m_rangeCircle  [private]
```

13.10.4.14 m_roundNum

```
uint32_t GameState::m_roundNum  [private]
```

13.10.4.15 m_roundNumText

```
sf::Text GameState::m_roundNumText  [private]
```

13.10.4.16 m_scoreText

```
sf::Text GameState::m_scoreText  [private]
```

13.10.4.17 m_selectedShape

```
sf::RectangleShape GameState::m_selectedShape  [private]
```

13.10.4.18 m_selX

```
int32_t GameState::m_selX  [private]
```

13.10.4.19 m_selY

```
int32_t GameState::m_selY  [private]
```

13.10.4.20 m_upgradeRange

```
sf::CircleShape GameState::m_upgradeRange  [private]
```

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

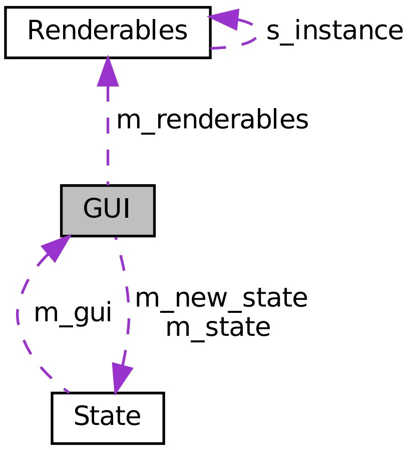
- src/states/gamestate.hpp
- src/states/gamestate.cpp

13.11 GUI Class Reference

A class to add elements to the Graphical User Interface.

```
#include <gui.hpp>
```

Collaboration diagram for GUI:



Public Member Functions

- GUI ()

  Construct a new GUI object This will create a Renderables object, which stores the sprites of everything.
- ∼GUI ()

  Destroy the GUI object Frees the dynamically allocated member variables.
- void init ()

  Initializes the different variables and makes the window ready.
- bool running ()

  Tells if the GUI is open.
- void update ()

  Updates GUI each frame.

- Button ∗ CreateButton (std::string text, int32_t x, int32_t y)

    Is used to create button objects.
- TowerButton ∗ CreateTowerButton (TowerType type, int32_t x, int32_t y)

    Is used to create towerbutton objects.
- const sf::Font & GetFont () const

    Get the Font used here.
- void ChangeState (State ∗state)

    Changes the state present in the GUI.

Private Member Functions

- void Priv_DeleteState ()

    Private deletes the state in the GUI, if present.

Private Attributes

- Renderables ∗ m_renderables
- sf::RenderWindow ∗ m_window
- sf::VideoMode m_videoMode
- sf::Event m_event
- std::vector< sf::Sprite > enemies
- float x_velo = 3.f
- float y_velo = 4.f
- sf::Vector2f start
- sf::Font m_font
- State ∗ m_state
- State ∗ m_new_state

13.11.1 Detailed Description

A class to add elements to the Graphical User Interface.

13.11.2 Constructor & Destructor Documentation

13.11.2.1 GUI()

```
GUI::GUI ( )
```

Construct a new GUI object This will create a Renderables object, which stores the sprites of everything.

13.11.2.2 ∼GUI()

```
GUI::∼GUI ( )
```

Destroy the GUI object Frees the dynamically allocated member variables.

13.11.3 Member Function Documentation

13.11.3.1 ChangeState()

```
void GUI::ChangeState (
            State * state )
```

Changes the state present in the GUI.

Parameters

state	The new state

13.11.3.2 CreateButton()

```
Button * GUI::CreateButton (
            std::string text,
            int32_t x,
            int32_t y )
```

Is used to create button objects.

Parameters

text	What is shown on the button
x	The x-coordinate of the upper left corner
y	The y-coordinate of the upper left corner

Returns

Button∗

13.11.3.3 CreateTowerButton()

```
TowerButton * GUI::CreateTowerButton (
            TowerType type,
            int32_t x,
            int32_t y )
```

Is used to create towerbutton objects.

Parameters

text	What is shown on the button
x	The x-coordinate of the upper left corner
y	The y-coordinate of the upper left corner

Returns

Button∗

13.11.3.4 GetFont()

```
const sf::Font & GUI::GetFont ( ) const
```

Get the Font used here.

Returns

const sf::Font&

13.11.3.5 init()

```
void GUI::init ( )
```

Initializes the different variables and makes the window ready.

13.11.3.6 Priv_DeleteState()

```
void GUI::Priv_DeleteState ( )  [private]
```

Private deletes the state in the GUI, if present.

13.11.3.7 running()

```
bool GUI::running ( )
```

Tells if the GUI is open.

Returns

bool

13.11.3.8 update()

```
void GUI::update ( )
```

Updates GUI each frame.

13.11.4 Member Data Documentation

13.11.4.1 enemies

```
std::vector<sf::Sprite> GUI::enemies  [private]
```

13.11.4.2 m_event

```
sf::Event GUI::m_event  [private]
```

13.11.4.3 m_font

```
sf::Font GUI::m_font  [private]
```

13.11.4.4 m_new_state

```
State* GUI::m_new_state  [private]
```

13.11.4.5 m_renderables

```
Renderables* GUI::m_renderables  [private]
```

13.11.4.6 m_state

```
State* GUI::m_state  [private]
```

13.11.4.7 m_videoMode

```
sf::VideoMode GUI::m_videoMode  [private]
```

13.11.4.8 m_window

```
sf::RenderWindow* GUI::m_window  [private]
```

13.11.4.9 start

```
sf::Vector2f GUI::start  [private]
```

13.11.4.10 x_velo

```
float GUI::x_velo = 3.f  [private]
```

13.11.4.11 y_velo

```
float GUI::y_velo = 4.f  [private]
```

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

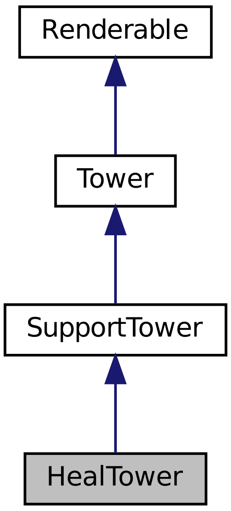
- src/gui.hpp
- src/gui.cpp

13.12 HealTower Class Reference

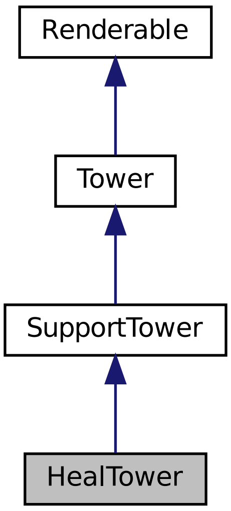
A tower which heals attacking towers.

```
#include <support_towers.hpp>
```

Inheritance diagram for HealTower:



Collaboration diagram for HealTower:



Public Member Functions

- HealTower (uint32_t range, const std::pair< int32_t, int32_t > &coords, uint32_t healStrength, const std↩ ::string &name, const std::vector< sf::Sprite > &sprites)

  Construct a new healing tower.
- ∼HealTower ()=default

  Default destructor.
- void Act (std::list< AttackingTower ∗ > &towers)

  Goes through the attacking towers and heals them if in range and not in full health already.

Private Attributes

- uint32_t m_healStrength

Additional Inherited Members

13.12.1 Detailed Description

A tower which heals attacking towers.

13.12.2 Constructor & Destructor Documentation

13.12.2.1 HealTower()

```
HealTower::HealTower (
            uint32_t range,
            const std::pair< int32_t, int32_t > & coords,
            uint32_t healStrength,
            const std::string & name,
            const std::vector< sf::Sprite > & sprites )
```

Construct a new healing tower.

Parameters

range	The basic range
coords	The coordinates of the tower
healStrength	The amount this tower can heal each tower during each tick
name	The name of the tower
sprites	Support towers cannot be upgraded, so the collection only has one sprite for them

13.12.2.2 ∼HealTower()

```
HealTower::∼HealTower ( )  [default]
```

Default destructor.

13.12.3 Member Function Documentation

13.12.3.1 Act()

```
void HealTower::Act (
             std::list< AttackingTower * > & towers )  [virtual]
```

Goes through the attacking towers and heals them if in range and not in full health already.

Parameters

towers	The towers on the game board

Implements SupportTower.

13.12.4 Member Data Documentation

13.12.4.1 m_healStrength

```
uint32_t HealTower::m_healStrength  [private]
```

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

- src/support_towers.hpp
- src/support_towers.cpp

13.13 Highscores Class Reference

A class used to handle the high score savings at the end of game.

```
#include <highscores.hpp>
```

Public Member Functions

- Highscores (const std::string &filename="highscores.txt")

    Construct a new Highscores object. Loads and sorts saved high scores.
- ∼Highscores ()=default

    Default destructor.
- std::vector< std::string > GetTop10 ()

    Get the top 10 of high scores.
- std::string GetTop10asString ()

    Get the top 10 of high scores as string.
- bool AddScore (const std::string &name, uint32_t score, Difficulty difficulty)

    Saves new score to highscores. Can be only done once.

Private Member Functions

- void Priv_LoadHighscores ()

    Private. Helper function to load the scores.
- void Priv_SortHighscores ()

    Private. Helper function to sort the scores.

Private Attributes

- std::vector< std::tuple< std::string, uint32_t, Difficulty > > m_highscores
- std::string m_filename
- bool m_saved

13.13.1 Detailed Description

A class used to handle the high score savings at the end of game.

13.13.2 Constructor & Destructor Documentation

13.13.2.1 Highscores()

```
Highscores::Highscores (
            const std::string & filename = "highscores.txt" )
```

Construct a new Highscores object. Loads and sorts saved high scores.

Parameters

filename	name of the file containing high scores.

13.13.2.2 ∼Highscores()

```
Highscores::∼Highscores ( )  [default]
```

Default destructor.

13.13.3 Member Function Documentation

13.13.3.1 AddScore()

```
bool Highscores::AddScore (
             const std::string & name,
             uint32_t score,
             Difficulty difficulty )
```

Saves new score to highscores. Can be only done once.

Parameters

name	name of the player, cannot contain ':'
score	score of the player
difficulty	difficulty of the game

Returns

bool if adding score was successful

13.13.3.2 GetTop10()

```
std::vector< std::string > Highscores::GetTop10 ( )
```

Get the top 10 of high scores.

Returns

std::vector<std::string> A vector of strings in format "<ranking>. <name>: <score> (<difficulty>)"

13.13.3.3 GetTop10asString()

```
std::string Highscores::GetTop10asString ( )
```

Get the top 10 of high scores as string.

Returns

std::string with all scores in format "<ranking>. <name>: <score> (<difficulty>)" seperated by newline

13.13.3.4 Priv_LoadHighscores()

```
void Highscores::Priv_LoadHighscores ( ) [private]
```

Private. Helper function to load the scores.

13.13.3.5 Priv_SortHighscores()

```
void Highscores::Priv_SortHighscores ( ) [private]
```

Private. Helper function to sort the scores.

13.13.4 Member Data Documentation

13.13.4.1 m_filename

```
std::string Highscores::m_filename [private]
```

13.13.4.2 m_highscores

```
std::vector<std::tuple<std::string, uint32_t, Difficulty> > Highscores::m_highscores [private]
```

13.13.4.3 m_saved

```
bool Highscores::m_saved [private]
```

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

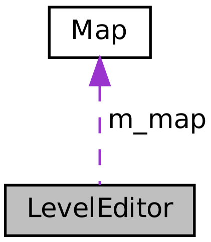
- src/highscores.hpp
- src/highscores.cpp

13.14 LevelEditor Class Reference

The logical core of the level editor state.

```
#include <level_editor.hpp>
```

Collaboration diagram for LevelEditor:



Public Member Functions

- LevelEditor (int32_t width, int32_t height, std::string &mapPath)

  Construct a new Level Editor object.
- ∼LevelEditor ()=default

  Default destructor.
- bool Edit (std::pair< int32_t, int32_t > coordinate, TileType tile)

  Edit a tile at the coordinates.
- bool Save ()

  Saves the map to file May throw an exception.
- const Map & GetMap () const

  Get the Map object.
- bool Validate ()

  Checks if the map is valid.

Private Attributes

- Map m_map
- int32_t m_width
- int32_t m_height
- std::string m_mapPath

13.14.1 Detailed Description

The logical core of the level editor state.

13.14.2 Constructor & Destructor Documentation

13.14.2.1 LevelEditor()

```
LevelEditor::LevelEditor (
            int32_t width,
            int32_t height,
            std::string & mapPath )
```

Construct a new Level Editor object.

Parameters

width	The width of the map
height	The height of the map
mapPath	Where the map file is located

13.14.2.2 ∼LevelEditor()

```
LevelEditor::∼LevelEditor ( )  [default]
```

Default destructor.

13.14.3 Member Function Documentation

13.14.3.1 Edit()

```
bool LevelEditor::Edit (
            std::pair< int32_t, int32_t > coordinate,
            TileType tile )
```

Edit a tile at the coordinates.

Parameters

coordinate	The location of the edit
tile	What to change the tile into

Returns

bool Whether the editing was successfull

13.14.3.2 GetMap()

```
const Map & LevelEditor::GetMap ( ) const
```

Get the Map object.

Returns

const Map&

13.14.3.3 Save()

```
bool LevelEditor::Save ( )
```

Saves the map to file May throw an exception.

Returns

bool Whether the saving was successfull

13.14.3.4 Validate()

```
bool LevelEditor::Validate ( )
```

Checks if the map is valid.

Returns

bool

13.14.4 Member Data Documentation

13.14.4.1 m_height

```
int32_t LevelEditor::m_height [private]
```

13.14.4.2 m_map

```
Map LevelEditor::m_map [private]
```

13.14.4.3 m_mapPath

```
std::string LevelEditor::m_mapPath [private]
```

13.14.4.4 m_width

```
int32_t LevelEditor::m_width [private]
```

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

- src/level_editor.hpp
- src/level_editor.cpp

13.15 Map Class Reference

Map class that represents the grid map system behind each unique level.

```
#include <map.hpp>
```

Public Member Functions

- Map (int32_t width, int32_t height)

    Construct a new Map object.
- ∼Map ()=default

    Default destructor.
- bool InitializeMap (const std::string &filename)

    Initialize a map from a text file to m_grid.
- const std::pair< int32_t, int32_t > GetStart () const

    Get the start coordinates.
- const std::pair< int32_t, int32_t > GetEnd () const

    Get the end coordinates.
- TileType GetPos (std::pair< int32_t, int32_t > coordinate) const

    Get the tile at position (x, y)
- const std::map< std::pair< int32_t, int32_t >, TileType > & GetGrid () const

    Get the Grid object.
- const std::vector< std::pair< int32_t, int32_t > > & GetPath () const

    Get the Path object.
- bool Edit (std::pair< int32_t, int32_t > coordinate, TileType tile)

    Change a tile on the map to another.
- bool ValidateMap ()

    Validate the current m_grid. (Called by InitializeMap).
- bool BuildPath ()

    Build the path of the current map in m_path.
- std::vector< std::pair< int32_t, int32_t > > GetNeighbors (int32_t x, int32_t y)

    Get the vertical and horizontal neighbors of (x, y).
- bool TestTilePos (std::pair< int32_t, int32_t > coordinate, TileType tile)

    Tests if the position is valid for the tile.

Private Attributes

- int32_t m_width
- int32_t m_height
- std::map< std::pair< int32_t, int32_t >, TileType > m_grid
- std::pair< int32_t, int32_t > m_start
- std::pair< int32_t, int32_t > m_end
- std::vector< std::pair< int32_t, int32_t > > m_path

13.15.1 Detailed Description

Map class that represents the grid map system behind each unique level.

13.15.2 Constructor & Destructor Documentation

13.15.2.1 Map()

```
Map::Map (
             int32_t width,
             int32_t height )
```

Construct a new Map object.

Parameters

width	Limit of x-coordinate of the map.
height	Limit of y-coordinate of the map.

13.15.2.2 ∼Map()

```
Map::∼Map ( )  [default]
```

Default destructor.

13.15.3 Member Function Documentation

13.15.3.1 BuildPath()

```
bool Map::BuildPath ( )
```

Build the path of the current map in m_path.

Returns

true if path is built successfully

false if path cannot be built.

13.15.3.2 Edit()

```
bool Map::Edit (
            std::pair< int32_t, int32_t > coordinate,
            TileType tile )
```

Change a tile on the map to another.

Parameters

coordinate	Where to perform the change
tile	What to change the tile into

Returns

true if editing was successful

false if editing was not successful.

13.15.3.3 GetEnd()

```
const std::pair< int, int > Map::GetEnd ( ) const
```

Get the end coordinates.

Returns

std::pair<int32_t, int32_t>

13.15.3.4 GetGrid()

```
const std::map< std::pair< int, int >, TileType > & Map::GetGrid ( ) const
```

Get the Grid object.

Returns

const std::map<std::pair<int32_t, int32_t>, TileType>&

13.15.3.5 GetNeighbors()

```
std::vector< std::pair< int, int > > Map::GetNeighbors (
             int32_t x,
             int32_t y )
```

Get the vertical and horizontal neighbors of (x, y).

Parameters

x	x-coordinate
y	y-coordinate

Returns

std::vector<std::pair<int32_t, int32_t>>

13.15.3.6 GetPath()

```
const std::vector< std::pair< int, int > > & Map::GetPath ( ) const
```

Get the Path object.

Returns

const std::vector<std::pair<int32_t, int32_t>>&

13.15.3.7 GetPos()

```
TileType Map::GetPos (
            std::pair< int32_t, int32_t > coordinate ) const
```

Get the tile at position (x, y)

Parameters

coordinate	The position

Returns

TilyType

13.15.3.8 GetStart()

```
const std::pair< int, int > Map::GetStart ( ) const
```

Get the start coordinates.

Returns

std::pair<int32_t, int32_t>

13.15.3.9 InitializeMap()

```
bool Map::InitializeMap (
            const std::string & filename )
```

Initialize a map from a text file to m_grid.

Parameters

filename	The file to be used

Returns

bool Whether the initialization was successfull

13.15.3.10 TestTilePos()

```
bool Map::TestTilePos (
             std::pair< int32_t, int32_t > coordinate,
             TileType tile )
```

Tests if the position is valid for the tile.

Parameters

coordinate	The position
tile	The type of the tile

Returns

true if position is valid

false if position is invalid

13.15.3.11 ValidateMap()

```
bool Map::ValidateMap ( )
```

Validate the current m_grid. (Called by InitializeMap).

Returns

true if map is valid

false if map is not valid

13.15.4 Member Data Documentation

13.15.4.1 m_end

```
std::pair<int32_t, int32_t> Map::m_end  [private]
```

13.15.4.2 m_grid

```
std::map<std::pair<int32_t, int32_t>, TileType> Map::m_grid  [private]
```

13.15.4.3 m_height

```
int32_t Map::m_height  [private]
```

13.15.4.4 m_path

```
std::vector<std::pair<int32_t, int32_t> > Map::m_path  [private]
```

13.15.4.5 m_start

```
std::pair<int32_t, int32_t> Map::m_start  [private]
```

13.15.4.6 m_width

```
int32_t Map::m_width  [private]
```

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

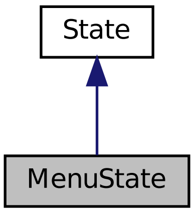
- src/map.hpp
- src/map.cpp

13.16 MenuState Class Reference

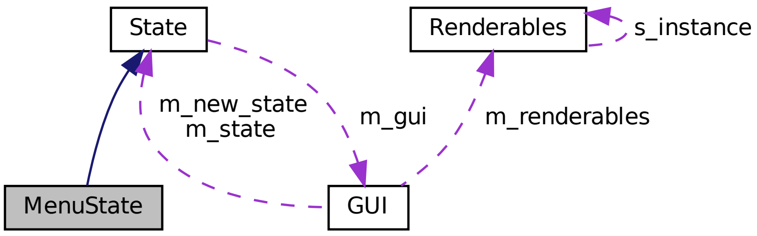
A state run by GUI corresponding to the main menu state.

`#include <menustate.hpp>`

Inheritance diagram for MenuState:



Collaboration diagram for MenuState:



Public Member Functions

- MenuState (GUI &gui, sf::RenderWindow &window)

  Construct a new Menu State object.
- ∼MenuState ()

  Destroy the Menu State object Frees the buttons.
- void Run ()

  Run and draw the Menu state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Has options to select difficulty and map file. Has buttons to game editor and to play.) Exits the loop and calls GUIs MoveToGameState(int score) when the user presses "Play".

Private Member Functions

- void RunLevelEditor (int width, int height, const std::string &map)
- void PollEvents ()
- void Draw ()

Private Attributes

- Difficulty m_difficulty
- std::string m_selectedMap
- int m_width = 30
- int m_height = 20
- std::map< int, Button * > m_buttons
- std::vector< sf::Text > m_texts
- bool m_editing

Additional Inherited Members

13.16.1 Detailed Description

A state run by GUI corresponding to the main menu state.

13.16.2 Constructor & Destructor Documentation

13.16.2.1 MenuState()

```
MenuState::MenuState (
            GUI & gui,
            sf::RenderWindow & window )
```

Construct a new Menu State object.

Parameters

gui	A ref to the GUI
window	A ref to the window

13.16.2.2 ∼MenuState()

```
MenuState::∼MenuState ( )
```

Destroy the Menu State object Frees the buttons.

13.16.3 Member Function Documentation

13.16.3.1 Draw()

```
void MenuState::Draw ( )  [private]
```

13.16.3.2 PollEvents()

```
void MenuState::PollEvents ( )  [private]
```

13.16.3.3 Run()

```
void MenuState::Run ( )  [virtual]
```

Run and draw the Menu state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window. Has options to select difficulty and map file. Has buttons to game editor and to play.) Exits the loop and calls GUIs MoveToGameState(int score) when the user presses "Play".

Implements State.

13.16.3.4 RunLevelEditor()

```
void MenuState::RunLevelEditor (
             int width,
             int height,
             const std::string & map )  [private]
```

13.16.4 Member Data Documentation

13.16.4.1 m_buttons

```
std::map<int, Button*> MenuState::m_buttons  [private]
```

13.16.4.2 m_difficulty

```
Difficulty MenuState::m_difficulty  [private]
```

13.16.4.3 m_editing

```
bool MenuState::m_editing  [private]
```

13.16.4.4 m_height

```
int MenuState::m_height = 20  [private]
```

13.16.4.5 m_selectedMap

```
std::string MenuState::m_selectedMap  [private]
```

13.16.4.6 m_texts

```
std::vector<sf::Text> MenuState::m_texts  [private]
```

13.16.4.7 m_width

```
int MenuState::m_width = 30  [private]
```

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

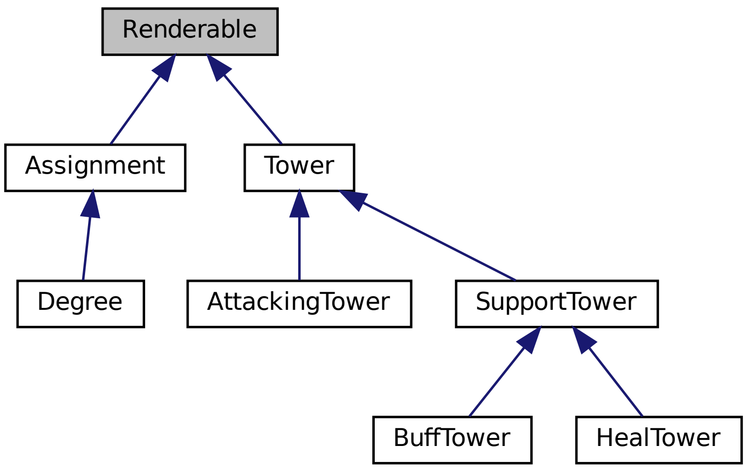
- src/states/menustate.hpp
- src/states/menustate.cpp

13.17 Renderable Class Reference

A class which encapsulates the renderable objects (enemies and towers) Is not supposed to be directly instanciated, so the constructor is protected.

```
#include <renderable.hpp>
```

Inheritance diagram for Renderable:



Public Member Functions

- virtual ~Renderable ()

  A virtual destructor.
- const std::string & EntityName () const

  Get the name of the entity Mainly for debugging.
- sf::Sprite & GetSprite ()

  Get the Sprite of this entity For rendering purposes.

Protected Member Functions

- Renderable (const std::string &entityName, const sf::Sprite &sprite)

  Construct a new Renderable object.
- void SetSprite (const sf::Sprite &newSprite)

  Set the Sprite object Used by the towers when they are upgraded.

Protected Attributes

- std::string m_entityName
- sf::Sprite m_sprite

13.17.1 Detailed Description

A class which encapsulates the renderable objects (enemies and towers) Is not supposed to be directly instanciated, so the constructor is protected.

13.17.2 Constructor & Destructor Documentation

13.17.2.1 ∼Renderable()

```
virtual Renderable::∼Renderable ( )  [inline], [virtual]
```

A virtual destructor.

13.17.2.2 Renderable()

```
Renderable::Renderable (
            const std::string & entityName,
            const sf::Sprite & sprite )  [protected]
```

Construct a new Renderable object.

Parameters

entityName	The name
sprite	The sprite used by the entity

13.17.3 Member Function Documentation

13.17.3.1 EntityName()

```
const std::string & Renderable::EntityName ( ) const
```

Get the name of the entity Mainly for debugging.

Returns

const std::string&

13.17.3.2 GetSprite()

```
sf::Sprite & Renderable::GetSprite ( )
```

Get the Sprite of this entity For rendering purposes.

Returns

sf::Sprite&

13.17.3.3 SetSprite()

```
void Renderable::SetSprite (
             const sf::Sprite & newSprite )  [protected]
```

Set the Sprite object Used by the towers when they are upgraded.

Parameters

newSprite	A ref to the new sprite

13.17.4 Member Data Documentation

13.17.4.1 m_entityName

```
std::string Renderable::m_entityName  [protected]
```

13.17.4.2 m_sprite

```
sf::Sprite Renderable::m_sprite  [protected]
```

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

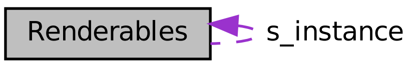
- src/renderable.hpp
- src/renderable.cpp

13.18 Renderables Class Reference

A class which handles the textures for different sprites One instance of this class must be constructed somewhere in the code before any static getters are accessed!

```
#include <renderables.hpp>
```

Collaboration diagram for Renderables:



Public Member Functions

- Renderables ()

Static Public Member Functions

- static sf::Sprite & getBachelor1Sprite ()
- static sf::Sprite & getBachelor2Sprite ()
- static sf::Sprite & getBachelor3Sprite ()
- static sf::Sprite & getBachelorsThesisSprite ()
- static sf::Sprite & getBackgroundSprite ()
- static sf::Sprite & getBscSprite ()
- static sf::Sprite & getCalculatorSprite ()
- static sf::Sprite & getCoffeeTableSprite ()
- static sf::Sprite & getDoctor1Sprite ()
- static sf::Sprite & getDoctor2Sprite ()
- static sf::Sprite & getDoctor3Sprite ()
- static sf::Sprite & getDoctoralThesisSprite ()
- static sf::Sprite & getDscSprite ()
- static sf::Sprite & getEndtileSprite ()
- static sf::Sprite & getEssaySprite ()
- static sf::Sprite & getFreshman1Sprite ()
- static sf::Sprite & getFreshman2Sprite ()
- static sf::Sprite & getFreshman3Sprite ()
- static sf::Sprite & getHomeworkSprite ()
- static sf::Sprite & getMaster1Sprite ()
- static sf::Sprite & getMaster2Sprite ()
- static sf::Sprite & getMaster3Sprite ()
- static sf::Sprite & getMastersThesisSprite ()
- static sf::Sprite & getMenuBackgroundSprite ()
- static sf::Sprite & getMscSprite ()
- static sf::Sprite & getPathtileSprite ()
- static sf::Sprite & getProjectSprite ()
- static sf::Sprite & getStarttileSprite ()
- static sf::Sprite & getTeekkari1Sprite ()
- static sf::Sprite & getTeekkari2Sprite ()
- static sf::Sprite & getTeekkari3Sprite ()
- static sf::Sprite & getTowertileSprite ()
- static sf::Sound & getAttackSound ()
- static sf::Sound & getSelectSound ()

Private Attributes

- sf::Texture bachelor_1
- sf::Texture bachelor_2
- sf::Texture bachelor_3
- sf::Texture bachelors_thesis
- sf::Texture background
- sf::Texture bsc
- sf::Texture calculator
- sf::Texture coffee_table
- sf::Texture doctor_1
- sf::Texture doctor_2
- sf::Texture doctor_3
- sf::Texture doctoral_thesis
- sf::Texture dsc
- sf::Texture endTile
- sf::Texture essay
- sf::Texture freshman_1
- sf::Texture freshman_2
- sf::Texture freshman_3
- sf::Texture homework
- sf::Texture master_1
- sf::Texture master_2
- sf::Texture master_3
- sf::Texture masters_thesis
- sf::Texture menu_background
- sf::Texture msc
- sf::Texture pathTile
- sf::Texture project
- sf::Texture startTile
- sf::Texture teekkari_1
- sf::Texture teekkari_2
- sf::Texture teekkari_3
- sf::Texture towerTile
- sf::Sprite bachelor_1_sprite
- sf::Sprite bachelor_2_sprite
- sf::Sprite bachelor_3_sprite
- sf::Sprite bachelors_thesis_sprite
- sf::Sprite background_sprite
- sf::Sprite bsc_sprite
- sf::Sprite calculator_sprite
- sf::Sprite coffee_table_sprite
- sf::Sprite doctor_1_sprite
- sf::Sprite doctor_2_sprite
- sf::Sprite doctor_3_sprite
- sf::Sprite doctoral_thesis_sprite
- sf::Sprite dsc_sprite
- sf::Sprite endTile_sprite
- sf::Sprite essay_sprite
- sf::Sprite freshman_1_sprite
- sf::Sprite freshman_2_sprite
- sf::Sprite freshman_3_sprite
- sf::Sprite homework_sprite
- sf::Sprite master_1_sprite
- sf::Sprite master_2_sprite

- sf::Sprite master_3_sprite
- sf::Sprite masters_thesis_sprite
- sf::Sprite menu_background_sprite
- sf::Sprite msc_sprite
- sf::Sprite pathTile_sprite
- sf::Sprite project_sprite
- sf::Sprite startTile_sprite
- sf::Sprite teekkari_1_sprite
- sf::Sprite teekkari_2_sprite
- sf::Sprite teekkari_3_sprite
- sf::Sprite towerTile_sprite
- sf::SoundBuffer attack
- sf::SoundBuffer select
- sf::Sound attack_sound
- sf::Sound select_sound

Static Private Attributes

- static Renderables ∗ s_instance = nullptr

    A static pointer to an instance of this class Is initialized when an instance is constructed somewhere in the program.

13.18.1 Detailed Description

A class which handles the textures for different sprites One instance of this class must be constructed somewhere in the code before any static getters are accessed!

13.18.2 Constructor & Destructor Documentation

13.18.2.1 Renderables()

```
Renderables::Renderables ( )
```

13.18.3 Member Function Documentation

13.18.3.1 getAttackSound()

```
sf::Sound & Renderables::getAttackSound ( )  [static]
```

13.18.3.2 getBachelor1Sprite()

```
sf::Sprite & Renderables::getBachelor1Sprite ( )  [static]
```

13.18.3.3 getBachelor2Sprite()

```
sf::Sprite & Renderables::getBachelor2Sprite ( )  [static]
```

13.18.3.4 getBachelor3Sprite()

```
sf::Sprite & Renderables::getBachelor3Sprite ( )  [static]
```

13.18.3.5 getBachelorsThesisSprite()

```
sf::Sprite & Renderables::getBachelorsThesisSprite ( )  [static]
```

13.18.3.6 getBackgroundSprite()

```
sf::Sprite & Renderables::getBackgroundSprite ( )  [static]
```

13.18.3.7 getBscSprite()

```
sf::Sprite & Renderables::getBscSprite ( )  [static]
```

13.18.3.8 getCalculatorSprite()

```
sf::Sprite & Renderables::getCalculatorSprite ( )  [static]
```

13.18.3.9 getCoffeeTableSprite()

```
sf::Sprite & Renderables::getCoffeeTableSprite ( )  [static]
```

13.18.3.10 getDoctor1Sprite()

```
sf::Sprite & Renderables::getDoctor1Sprite ( )  [static]
```

13.18.3.11 getDoctor2Sprite()

```
sf::Sprite & Renderables::getDoctor2Sprite ( )  [static]
```

13.18.3.12 getDoctor3Sprite()

```
sf::Sprite & Renderables::getDoctor3Sprite ( )  [static]
```

13.18.3.13 getDoctoralThesisSprite()

```
sf::Sprite & Renderables::getDoctoralThesisSprite ( )  [static]
```

13.18.3.14 getDscSprite()

```
sf::Sprite & Renderables::getDscSprite ( )  [static]
```

13.18.3.15 getEndtileSprite()

```
sf::Sprite & Renderables::getEndtileSprite ( )  [static]
```

13.18.3.16 getEssaySprite()

```
sf::Sprite & Renderables::getEssaySprite ( )  [static]
```

13.18.3.17 getFreshman1Sprite()

```
sf::Sprite & Renderables::getFreshman1Sprite ( )  [static]
```

13.18.3.18 getFreshman2Sprite()

```
sf::Sprite & Renderables::getFreshman2Sprite ( )  [static]
```

13.18.3.19 getFreshman3Sprite()

```
sf::Sprite & Renderables::getFreshman3Sprite ( )  [static]
```

13.18.3.20 getHomeworkSprite()

```
sf::Sprite & Renderables::getHomeworkSprite ( )  [static]
```

13.18.3.21 getMaster1Sprite()

```
sf::Sprite & Renderables::getMaster1Sprite ( )  [static]
```

13.18.3.22 getMaster2Sprite()

```
sf::Sprite & Renderables::getMaster2Sprite ( )  [static]
```

13.18.3.23 getMaster3Sprite()

```
sf::Sprite & Renderables::getMaster3Sprite ( )  [static]
```

13.18.3.24 getMastersThesisSprite()

```
sf::Sprite & Renderables::getMastersThesisSprite ( )  [static]
```

13.18.3.25 getMenuBackgroundSprite()

```
sf::Sprite & Renderables::getMenuBackgroundSprite ( )  [static]
```

13.18.3.26 getMscSprite()

```
sf::Sprite & Renderables::getMscSprite ( )  [static]
```

13.18.3.27 getPathtileSprite()

```
sf::Sprite & Renderables::getPathtileSprite ( )  [static]
```

13.18.3.28 getProjectSprite()

```
sf::Sprite & Renderables::getProjectSprite ( )  [static]
```

13.18.3.29 getSelectSound()

```
sf::Sound & Renderables::getSelectSound ( )  [static]
```

13.18.3.30 getStarttileSprite()

```
sf::Sprite & Renderables::getStarttileSprite ( )  [static]
```

13.18.3.31 getTeekkari1Sprite()

```
sf::Sprite & Renderables::getTeekkari1Sprite ( )  [static]
```

13.18.3.32 getTeekkari2Sprite()

```
sf::Sprite & Renderables::getTeekkari2Sprite ( )  [static]
```

13.18.3.33 getTeekkari3Sprite()

```
sf::Sprite & Renderables::getTeekkari3Sprite ( )  [static]
```

13.18.3.34 getTowertileSprite()

```
sf::Sprite & Renderables::getTowertileSprite ( )  [static]
```

13.18.4 Member Data Documentation

13.18.4.1 attack

```
sf::SoundBuffer Renderables::attack  [private]
```

13.18.4.2 attack_sound

```
sf::Sound Renderables::attack_sound  [private]
```

13.18.4.3 bachelor_1

```
sf::Texture Renderables::bachelor_1  [private]
```

13.18.4.4 bachelor_1_sprite

```
sf::Sprite Renderables::bachelor_1_sprite  [private]
```

13.18.4.5 bachelor_2

```
sf::Texture Renderables::bachelor_2  [private]
```

13.18.4.6 bachelor_2_sprite

```
sf::Sprite Renderables::bachelor_2_sprite  [private]
```

13.18.4.7 bachelor_3

```
sf::Texture Renderables::bachelor_3  [private]
```

13.18.4.8 bachelor_3_sprite

```
sf::Sprite Renderables::bachelor_3_sprite  [private]
```

13.18.4.9 bachelors_thesis

```
sf::Texture Renderables::bachelors_thesis  [private]
```

13.18.4.10 bachelors_thesis_sprite

```
sf::Sprite Renderables::bachelors_thesis_sprite  [private]
```

13.18.4.11 background

```
sf::Texture Renderables::background  [private]
```

13.18.4.12 background_sprite

```
sf::Sprite Renderables::background_sprite  [private]
```

13.18.4.13 bsc

```
sf::Texture Renderables::bsc  [private]
```

13.18.4.14 bsc_sprite

```
sf::Sprite Renderables::bsc_sprite  [private]
```

13.18.4.15 calculator

```
sf::Texture Renderables::calculator  [private]
```

13.18.4.16 calculator_sprite

```
sf::Sprite Renderables::calculator_sprite  [private]
```

13.18.4.17 coffee_table

```
sf::Texture Renderables::coffee_table  [private]
```

13.18.4.18 coffee_table_sprite

```
sf::Sprite Renderables::coffee_table_sprite  [private]
```

13.18.4.19 doctor_1

```
sf::Texture Renderables::doctor_1  [private]
```

13.18.4.20 doctor_1_sprite

```
sf::Sprite Renderables::doctor_1_sprite  [private]
```

13.18.4.21 doctor_2

```
sf::Texture Renderables::doctor_2  [private]
```

13.18.4.22 doctor_2_sprite

```
sf::Sprite Renderables::doctor_2_sprite  [private]
```

13.18.4.23 doctor_3

```
sf::Texture Renderables::doctor_3  [private]
```

13.18.4.24 doctor_3_sprite

```
sf::Sprite Renderables::doctor_3_sprite  [private]
```

13.18.4.25 doctoral_thesis

```
sf::Texture Renderables::doctoral_thesis  [private]
```

13.18.4.26 doctoral_thesis_sprite

```
sf::Sprite Renderables::doctoral_thesis_sprite  [private]
```

13.18.4.27 dsc

```
sf::Texture Renderables::dsc  [private]
```

13.18.4.28 dsc_sprite

```
sf::Sprite Renderables::dsc_sprite  [private]
```

13.18.4.29 endTile

```
sf::Texture Renderables::endTile  [private]
```

13.18.4.30 endTile_sprite

```
sf::Sprite Renderables::endTile_sprite  [private]
```

13.18.4.31 essay

```
sf::Texture Renderables::essay  [private]
```

13.18.4.32 essay_sprite

```
sf::Sprite Renderables::essay_sprite  [private]
```

13.18.4.33 freshman_1

```
sf::Texture Renderables::freshman_1  [private]
```

13.18.4.34 freshman_1_sprite

```
sf::Sprite Renderables::freshman_1_sprite  [private]
```

13.18.4.35 freshman_2

```
sf::Texture Renderables::freshman_2  [private]
```

13.18.4.36 freshman_2_sprite

```
sf::Sprite Renderables::freshman_2_sprite  [private]
```

13.18.4.37 freshman_3

```
sf::Texture Renderables::freshman_3  [private]
```

13.18.4.38 freshman_3_sprite

```
sf::Sprite Renderables::freshman_3_sprite  [private]
```

13.18.4.39 homework

```
sf::Texture Renderables::homework  [private]
```

13.18.4.40 homework_sprite

```
sf::Sprite Renderables::homework_sprite  [private]
```

13.18.4.41 master_1

```
sf::Texture Renderables::master_1  [private]
```

13.18.4.42 master_1_sprite

```
sf::Sprite Renderables::master_1_sprite  [private]
```

13.18.4.43 master_2

```
sf::Texture Renderables::master_2  [private]
```

13.18.4.44 master_2_sprite

```
sf::Sprite Renderables::master_2_sprite  [private]
```

13.18.4.45 master_3

```
sf::Texture Renderables::master_3  [private]
```

13.18.4.46 master_3_sprite

```
sf::Sprite Renderables::master_3_sprite  [private]
```

13.18.4.47 masters_thesis

```
sf::Texture Renderables::masters_thesis  [private]
```

13.18.4.48 masters_thesis_sprite

```
sf::Sprite Renderables::masters_thesis_sprite  [private]
```

13.18.4.49 menu_background

```
sf::Texture Renderables::menu_background  [private]
```

13.18.4.50 menu_background_sprite

```
sf::Sprite Renderables::menu_background_sprite  [private]
```

13.18.4.51 msc

```
sf::Texture Renderables::msc  [private]
```

13.18.4.52 msc_sprite

```
sf::Sprite Renderables::msc_sprite  [private]
```

13.18.4.53 pathTile

```
sf::Texture Renderables::pathTile  [private]
```

13.18.4.54 pathTile_sprite

```
sf::Sprite Renderables::pathTile_sprite  [private]
```

13.18.4.55 project

```
sf::Texture Renderables::project  [private]
```

13.18.4.56 project_sprite

```
sf::Sprite Renderables::project_sprite  [private]
```

13.18.4.57 s_instance

```
Renderables * Renderables::s_instance = nullptr  [static], [private]
```

A static pointer to an instance of this class Is initialized when an instance is constructed somewhere in the program.

13.18.4.58 select

```
sf::SoundBuffer Renderables::select  [private]
```

13.18.4.59 select_sound

```
sf::Sound Renderables::select_sound  [private]
```

13.18.4.60 startTile

```
sf::Texture Renderables::startTile  [private]
```

13.18.4.61 startTile_sprite

```
sf::Sprite Renderables::startTile_sprite  [private]
```

13.18.4.62 teekkari_1

```
sf::Texture Renderables::teekkari_1  [private]
```

13.18.4.63 teekkari_1_sprite

```
sf::Sprite Renderables::teekkari_1_sprite  [private]
```

13.18.4.64 teekkari_2

```
sf::Texture Renderables::teekkari_2  [private]
```

13.18.4.65 teekkari_2_sprite

```
sf::Sprite Renderables::teekkari_2_sprite  [private]
```

13.18.4.66 teekkari_3

```
sf::Texture Renderables::teekkari_3  [private]
```

13.18.4.67 teekkari_3_sprite

```
sf::Sprite Renderables::teekkari_3_sprite  [private]
```

13.18.4.68 towerTile

```
sf::Texture Renderables::towerTile  [private]
```

13.18.4.69 towerTile_sprite

```
sf::Sprite Renderables::towerTile_sprite  [private]
```

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

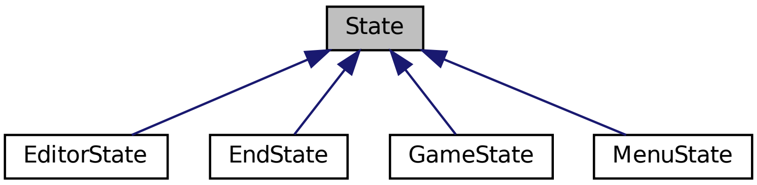
- src/renderables.hpp
- src/renderables.cpp

13.19 State Class Reference

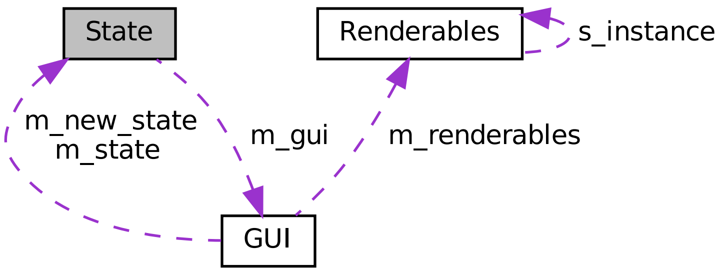
Abstract State class represents the classes that run and draw the different software states.

```
#include <state.hpp>
```

Inheritance diagram for State:



Collaboration diagram for State:



Public Member Functions

- State (GUI &gui, sf::RenderWindow &window)

  Construct a new State object.
- virtual ∼State ()=default

  Virtual destructor.
- virtual void Run ()=0

  Run the state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window.) A pure virtual function.

Protected Attributes

- GUI & m_gui
- sf::RenderWindow & m_window
- sf::Event m_event

13.19.1 Detailed Description

Abstract State class represents the classes that run and draw the different software states.

13.19.2 Constructor & Destructor Documentation

13.19.2.1 State()

```
State::State (
             GUI & gui,
             sf::RenderWindow & window )  [inline]
```

Construct a new State object.

Parameters

gui	A ref to the gui
window	A ref to the window

13.19.2.2 ∼State()

```
virtual State::∼State ( )  [virtual], [default]
```

Virtual destructor.

13.19.3 Member Function Documentation

13.19.3.1 Run()

```
virtual void State::Run ( )  [pure virtual]
```

Run the state. (Hosts the loop that Polls GUI events, calls the necessary backend methods and draws the window.) A pure virtual function.

Implemented in EndState, EditorState, MenuState, and GameState.

13.19.4 Member Data Documentation

13.19.4.1 m_event

```
sf::Event State::m_event  [protected]
```

13.19.4.2 m_gui

```
GUI& State::m_gui  [protected]
```

13.19.4.3 m_window

```
sf::RenderWindow& State::m_window  [protected]
```

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

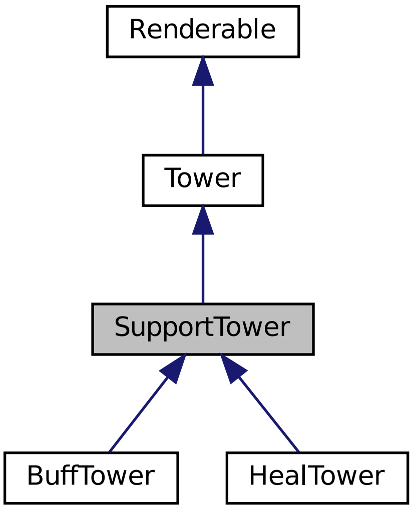
- src/states/state.hpp

13.20 SupportTower Class Reference

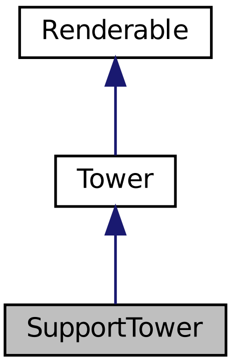
A virtual base class for the supporting towers Cannot be directly instanciated.

```
#include <support_towers.hpp>
```

Inheritance diagram for SupportTower:



Collaboration diagram for SupportTower:



Public Member Functions

- virtual ∼SupportTower ()

  Virtual destructor.
- bool IsUpgradeable (uint32_t money) const

  No SupportTower can be upgraded.
- virtual void Act (std::list< AttackingTower ∗ > &towers)=0

  Used by the supporting towers to buff/heal the attacking towers The towers go through the attacking towers present on the game board and apply buffs/heal those that are within range.

Static Public Member Functions

- static SupportTower ∗ Calculator (const std::pair< int32_t, int32_t > &coords)

  Creates a BuffTower called calculator.
- static SupportTower ∗ CoffeeTable (const std::pair< int32_t, int32_t > &coords)

  Creates a HealTower called coffee_table.

Protected Member Functions

- SupportTower (uint32_t range, const std::pair< int32_t, int32_t > &coords, const std::string &name, const std::vector< sf::Sprite > &sprites)

  Construct a new Support Tower object.

Friends

- std::ostream & operator<< (std::ostream &os, const SupportTower &st)

  Overload to stream output operator.

Additional Inherited Members

13.20.1 Detailed Description

A virtual base class for the supporting towers Cannot be directly instanciated.

13.20.2 Constructor & Destructor Documentation

13.20.2.1 SupportTower()

```
SupportTower::SupportTower (
             uint32_t range,
             const std::pair< int32_t, int32_t > & coords,
             const std::string & name,
             const std::vector< sf::Sprite > & sprites )  [protected]
```

Construct a new Support Tower object.

Parameters

range	The range of the tower
coords	The coordinates of the tower
name	The name of this tower
sprites	Support towers cannot be upgraded, so the collection only has one sprite for them

13.20.2.2 ∼SupportTower()

```
virtual SupportTower::∼SupportTower ( )  [inline], [virtual]
```

Virtual destructor.

13.20.3 Member Function Documentation

13.20.3.1 Act()

```
virtual void SupportTower::Act (
             std::list< AttackingTower * > & towers )  [pure virtual]
```

Used by the supporting towers to buff/heal the attacking towers The towers go through the attacking towers present on the game board and apply buffs/heal those that are within range.

Parameters

towers	The towers present on the game board

Implemented in HealTower, and BuffTower.

13.20.3.2 Calculator()

```
SupportTower * SupportTower::Calculator (
             const std::pair< int32_t, int32_t > & coords )  [static]
```

Creates a BuffTower called calculator.

Parameters

coords	Where to create the tower

Returns

SupportTower∗

13.20.3.3 CoffeeTable()

```
SupportTower * SupportTower::CoffeeTable (
            const std::pair< int32_t, int32_t > & coords )  [static]
```

Creates a HealTower called coffee_table.

Parameters

coords	Where to create the tower

Returns

SupportTower∗

13.20.3.4 IsUpgradeable()

```
bool SupportTower::IsUpgradeable (
            uint32_t money ) const  [virtual]
```

No SupportTower can be upgraded.

Returns

false

Implements Tower.

13.20.4 Friends And Related Function Documentation

13.20.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const SupportTower & st )  [friend]
```

Overload to stream output operator.

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

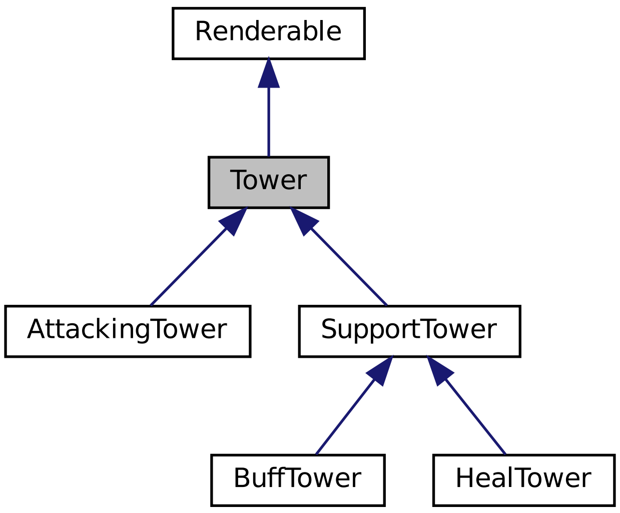
- src/support_towers.hpp
- src/support_towers.cpp

13.21 Tower Class Reference

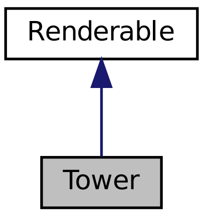
An abstract base-class for the towers Sub-classes will be Attacking towers and supporting towers. Cannot be directly instanciated.

```
#include <tower.hpp>
```

Inheritance diagram for Tower:



Collaboration diagram for Tower:



Public Member Functions

- virtual ~Tower ()

  Virtual destructor.

- const std::pair< int32_t, int32_t > & GetCoords () const

  Get the coordinates of the tower.
- virtual bool IsUpgradeable (uint32_t money) const =0

  Tells if the tower can be upgraded with set amount of money Supporting towers can never be upgraded.
- uint32_t GetRange () const

  Get the range of the tower.

Static Public Attributes

- static const std::map< TowerType, uint32_t > towerPrices

  The prices of the different kinds of towers.
- static const std::map< TowerType, uint32_t > towerRanges

  The base ranges of the towers.
- static const std::map< TowerType, uint32_t > towerPowers

  The base powers of the attacking towers.
- static const std::map< TowerType, uint32_t > towerHealths

  The healths of the attacking towers.

Protected Member Functions

- Tower (uint32_t range, const std::pair< int32_t, int32_t > &coords, const std::string &name, const std↩ ::vector< sf::Sprite > &sprites)

  Plain towers are not meant to be constructed, only the subclassess are.

Protected Attributes

- uint32_t m_range
- std::pair< int32_t, int32_t > m_coords
- std::vector< sf::Sprite > m_allSprites

13.21.1 Detailed Description

An abstract base-class for the towers Sub-classes will be Attacking towers and supporting towers. Cannot be directly instanciated.

13.21.2 Constructor & Destructor Documentation

13.21.2.1 ∼Tower()

```
virtual Tower::∼Tower ( )  [inline], [virtual]
```

Virtual destructor.

13.21.2.2 Tower()

```
Tower::Tower (
            uint32_t range,
            const std::pair< int32_t, int32_t > & coords,
            const std::string & name,
            const std::vector< sf::Sprite > & sprites )  [protected]
```

Plain towers are not meant to be constructed, only the subclassess are.

Parameters

range	The basic range, all towers have this
coords	The coordinates of the tower
name	The name of the tower
sprites	Sprites for the different levels of the tower

13.21.3 Member Function Documentation

13.21.3.1 GetCoords()

```
const std::pair< int32_t, int32_t > & Tower::GetCoords ( ) const
```

Get the coordinates of the tower.

Returns

const std::pair<int32_t, int32_t>&

13.21.3.2 GetRange()

```
uint32_t Tower::GetRange ( ) const
```

Get the range of the tower.

Returns

uint32_t

13.21.3.3 IsUpgradeable()

```
virtual bool Tower::IsUpgradeable (
            uint32_t money ) const  [pure virtual]
```

Tells if the tower can be upgraded with set amount of money Supporting towers can never be upgraded.

Parameters

money	The amount of money the player has

Returns

bool

Implemented in AttackingTower, and SupportTower.

13.21.4 Member Data Documentation

13.21.4.1 m_allSprites

```
std::vector<sf::Sprite> Tower::m_allSprites  [protected]
```

13.21.4.2 m_coords

```
std::pair<int32_t, int32_t> Tower::m_coords  [protected]
```

13.21.4.3 m_range

```
uint32_t Tower::m_range  [protected]
```

13.21.4.4 towerHealths

```
const std::map<TowerType, uint32_t> Tower::towerHealths  [inline], [static]
```

Initial value:

```
= {
      {TowerType::Freshman, 10}, {TowerType::Teekkari, 20},
      {TowerType::Bachelor, 30}, {TowerType::Master, 40},
      {TowerType::Doctor, 50},
  }
```

The healths of the attacking towers.

13.21.4.5 towerPowers

```
const std::map<TowerType, uint32_t> Tower::towerPowers  [inline], [static]
```

Initial value:

```
= {
      {TowerType::Freshman, 1},  {TowerType::Teekkari, 5},
      {TowerType::Bachelor, 20}, {TowerType::Master, 50},
      {TowerType::Doctor, 100},
  }
```

The base powers of the attacking towers.

13.21.4.6 towerPrices

```
const std::map<TowerType, uint32_t> Tower::towerPrices  [inline], [static]
```

Initial value:

```
= {
      {TowerType::Freshman, 50},     {TowerType::Teekkari, 300},
      {TowerType::Bachelor, 1000},   {TowerType::Master, 3000},
      {TowerType::Doctor, 5000},     {TowerType::Calculator, 1000},
      {TowerType::CoffeeTable, 1000}}
```

The prices of the different kinds of towers.

13.21.4.7 towerRanges

```
const std::map<TowerType, uint32_t> Tower::towerRanges  [inline], [static]
```

Initial value:

```
= {
      {TowerType::Freshman, 2},   {TowerType::Teekkari, 3},
      {TowerType::Bachelor, 5},   {TowerType::Master, 7},
      {TowerType::Doctor, 9},     {TowerType::Calculator, 5},
      {TowerType::CoffeeTable, 5}}
```

The base ranges of the towers.

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

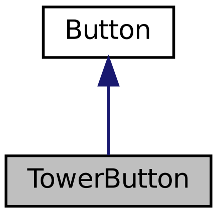
- src/tower.hpp
- src/tower.cpp

13.22 TowerButton Class Reference

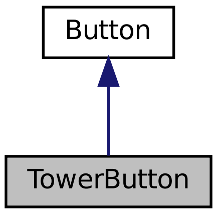
A subclass of buttons, for building the towers.

```
#include <button.hpp>
```

Inheritance diagram for TowerButton:



Collaboration diagram for TowerButton:



Public Member Functions

- TowerButton (TowerType tower, int32_t x, int32_t y, sf::Font &font)

    Construct a new Tower Button object.
- ∼TowerButton ()=default

    Default destructor.
- void drawButton (sf::RenderWindow &window)

    Draws the button on the window.
- void disableButton ()

    Makes the button grayed out.
- void enableButton ()

    Restores the original look of the button.

Private Attributes

- sf::Sprite m_sprite
- std::string m_name

Additional Inherited Members

13.22.1 Detailed Description

A subclass of buttons, for building the towers.

13.22.2 Constructor & Destructor Documentation

13.22.2.1 TowerButton()

```
TowerButton::TowerButton (
            TowerType tower,
            int32_t x,
            int32_t y,
            sf::Font & font )
```

Construct a new Tower Button object.

Parameters

tower	The type of tower this is supposed to represent
x	The x-coordinate of the upper left corner
y	The y-coordinate of the upper left corner
font	The font used by the text

13.22.2.2 ∼TowerButton()

```
TowerButton::∼TowerButton ( )  [default]
```

Default destructor.

13.22.3 Member Function Documentation

13.22.3.1 disableButton()

```
void TowerButton::disableButton ( )  [virtual]
```

Makes the button grayed out.

Reimplemented from Button.

13.22.3.2 drawButton()

```
void TowerButton::drawButton (
             sf::RenderWindow & window )  [virtual]
```

Draws the button on the window.

Parameters

window	A ref to the window where to draw

Reimplemented from Button.

13.22.3.3 enableButton()

```
void TowerButton::enableButton ( )  [virtual]
```

Restores the original look of the button.

Reimplemented from Button.

13.22.4 Member Data Documentation

13.22.4.1 m_name

```
std::string TowerButton::m_name  [private]
```

13.22.4.2 m_sprite

```
sf::Sprite TowerButton::m_sprite  [private]
```

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

- src/button.hpp
- src/button.cpp

Chapter 14

File Documentation

14.1 build/CMakeCache.txt File Reference

14.2 build/CMakeFiles/3.16.3/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_DIALECT

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" "" "]"
- char const ∗ info_platform = "INFO" ":" "platform[" "]"
- char const ∗ info_arch = "INFO" ":" "arch[" "]"
- const char ∗ info_language_dialect_default

14.2.1 Macro Definition Documentation

14.2.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

14.2.1.2 C_DIALECT

```
#define C_DIALECT
```

14.2.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

14.2.1.4 DEC

```
#define DEC(
                n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

14.2.1.5 HEX

```
#define HEX(
                n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)      & 0xF))
```

14.2.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

14.2.1.7 STRINGIFY

```
#define STRINGIFY(
             X ) STRINGIFY_HELPER(X)
```

14.2.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
             X ) #X
```

14.2.2 Function Documentation

14.2.2.1 main()

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

14.2.3 Variable Documentation

14.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" "]"
```

14.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" "" "]"
```

14.2.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
=
  "INFO" ":" "dialect_default["    "]"
```

14.2.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" "]"
```

14.3 build/CMakeFiles/3.16.3/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" "" "]"
- char const ∗ info_platform = "INFO" ":" "platform[" "]"
- char const ∗ info_arch = "INFO" ":" "arch[" "]"
- const char ∗ info_language_dialect_default

14.3.1 Macro Definition Documentation

14.3.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

14.3.1.2 COMPILER_ID

```
#define COMPILER_ID ""
```

14.3.1.3 CXX_STD

```
#define CXX_STD __cplusplus
```

14.3.1.4 DEC

```
#define DEC(
              n )
```

Value:

```
('0' + (((n) / 10000000)%10)), \
('0' + (((n) / 1000000)%10)),  \
('0' + (((n) / 100000)%10)),   \
('0' + (((n) / 10000)%10)),    \
('0' + (((n) / 1000)%10)),     \
('0' + (((n) / 100)%10)),      \
('0' + (((n) / 10)%10)),       \
('0' +  ((n) % 10))
```

14.3.1.5 HEX

```
#define HEX(
              n )
```

Value:

```
('0' + ((n)»28 & 0xF)), \
('0' + ((n)»24 & 0xF)), \
('0' + ((n)»20 & 0xF)), \
('0' + ((n)»16 & 0xF)), \
('0' + ((n)»12 & 0xF)), \
('0' + ((n)»8  & 0xF)), \
('0' + ((n)»4  & 0xF)), \
('0' + ((n)     & 0xF))
```

14.3.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

14.3.1.7 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

14.3.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

14.3.2 Function Documentation

14.3.2.1 main()

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

14.3.3 Variable Documentation

14.3.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" "]"
```

14.3.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" "" "]"
```

14.3.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
= "INFO" ":" "dialect_default["
  "98"
"]"
```

14.3.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" "]"
```

14.4 build/CMakeFiles/TargetDirectories.txt File Reference

14.5 build/CMakeFiles/tower-defense.dir/link.txt File Reference

14.6 CMakeLists.txt File Reference

14.7 doc/readme.md File Reference

14.8 libs/readme.md File Reference

14.9 plan/readme.md File Reference

14.10 src/readme.md File Reference

14.11 tests/readme.md File Reference

14.12 highscores.txt File Reference

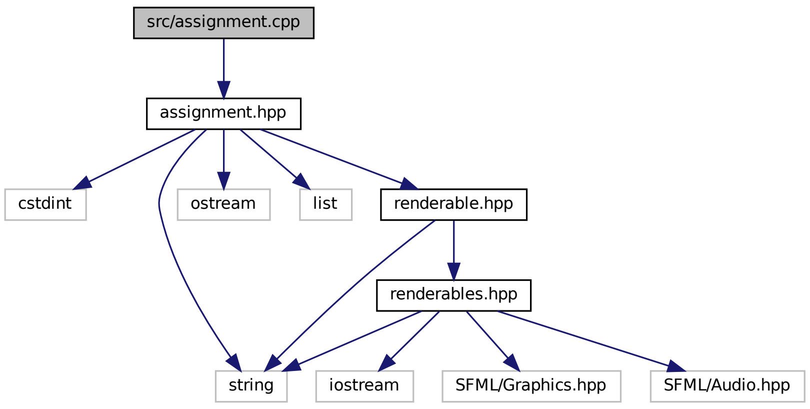
14.13 Meeting-notes.md File Reference

14.14 README.md File Reference

14.15 src/assignment.cpp File Reference

```
#include "assignment.hpp"
```

Include dependency graph for assignment.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const Assignment &as)

14.15.1 Function Documentation

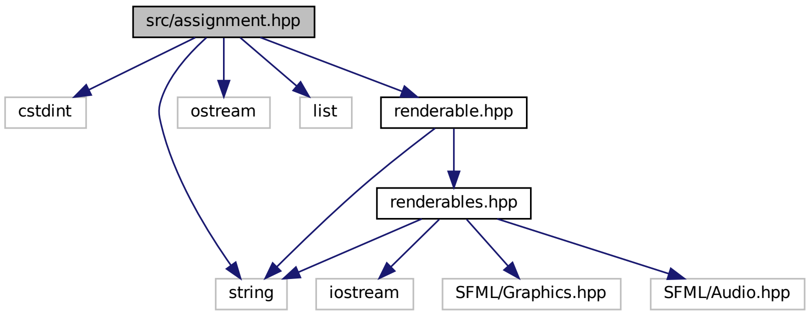
14.15.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            const Assignment & as )
```

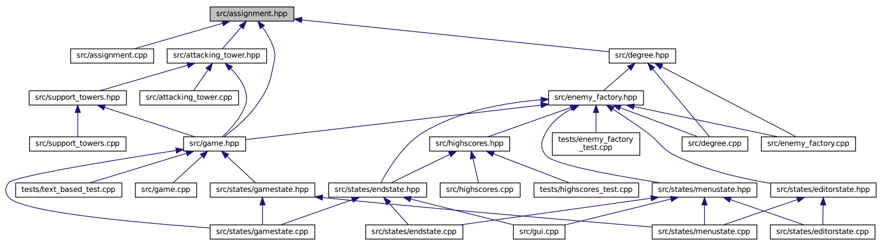
14.16 src/assignment.hpp File Reference

```
#include <cstdint>
#include <string>
#include <ostream>
#include <list>
#include "renderable.hpp"
```

Include dependency graph for assignment.hpp:



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



Classes

- class Assignment

    Generic Enemy class, these just die when they are killed.

Enumerations

- enum Enemy {
  Homework, Essay, Project, B_Thesis,
  M_Thesis, D_Thesis, BSc, MSc,
  DSc }

    An enumeration for the different enemy types.

14.16.1 Enumeration Type Documentation

14.16.1.1 Enemy

```
enum Enemy
```

An enumeration for the different enemy types.

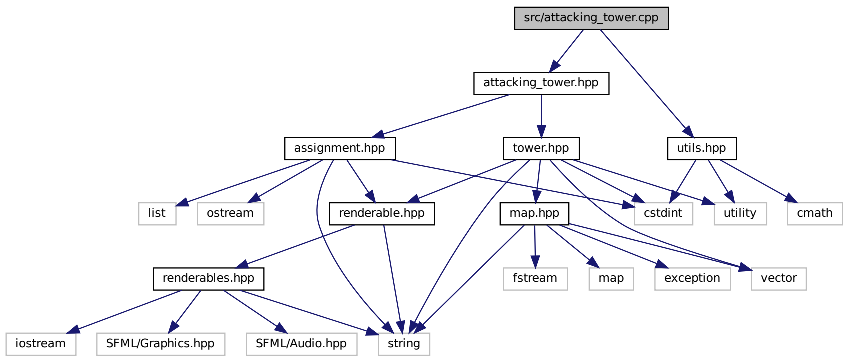
Enumerator

Homework	
Essay	
Project	
B_Thesis	
M_Thesis	
D_Thesis	
BSc	
MSc	
DSc	

14.17 src/attacking_tower.cpp File Reference

```
#include "attacking_tower.hpp"
#include "utils.hpp"
```

Include dependency graph for attacking_tower.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const AttackingTower &at)

14.17.1 Function Documentation

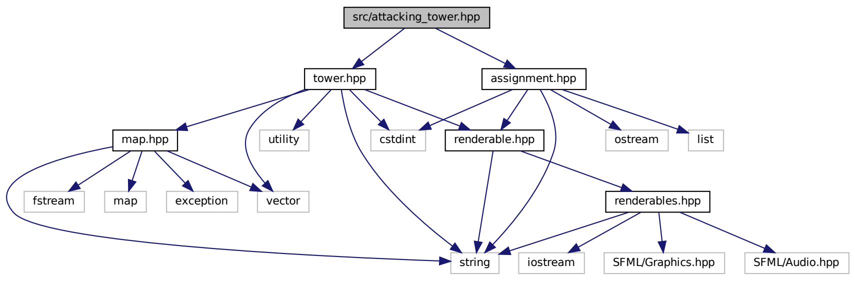
14.17.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            const AttackingTower & at )
```

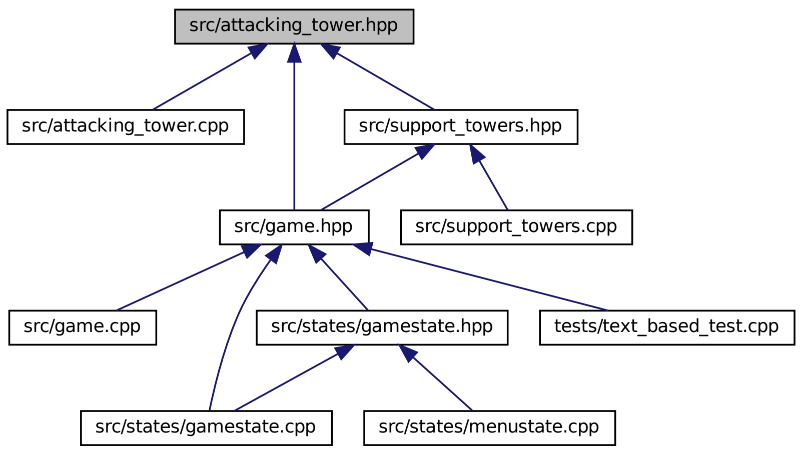
14.18 src/attacking_tower.hpp File Reference

```
#include "tower.hpp"
#include "assignment.hpp"
```

Include dependency graph for attacking_tower.hpp:



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



Classes

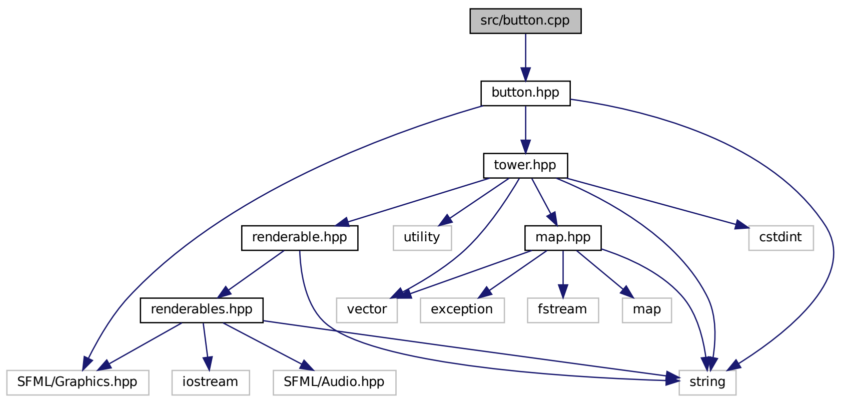
- class AttackingTower

  A class for the offensive towers.

14.19 src/button.cpp File Reference

```
#include "button.hpp"
```

Include dependency graph for button.cpp:



Macros

- #define TILE_SIZE 30

14.19.1 Macro Definition Documentation

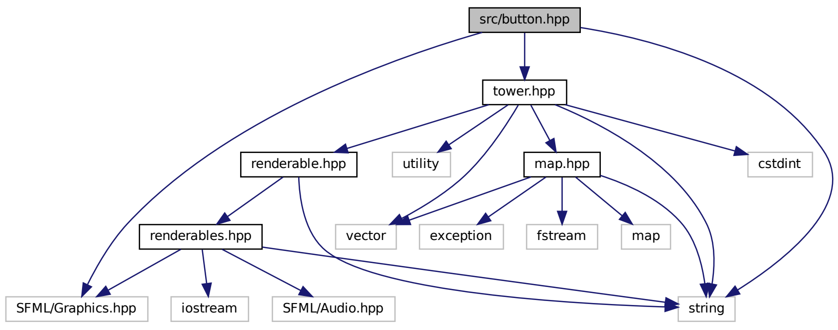
14.19.1.1 TILE_SIZE

```
#define TILE_SIZE 30
```

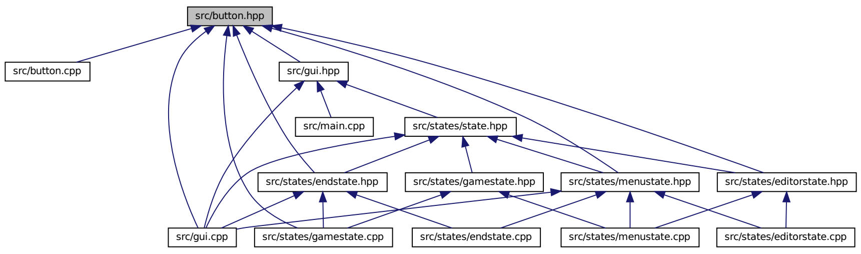
14.20 src/button.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <string>
#include "tower.hpp"
```

Include dependency graph for button.hpp:



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



Classes

- class Button

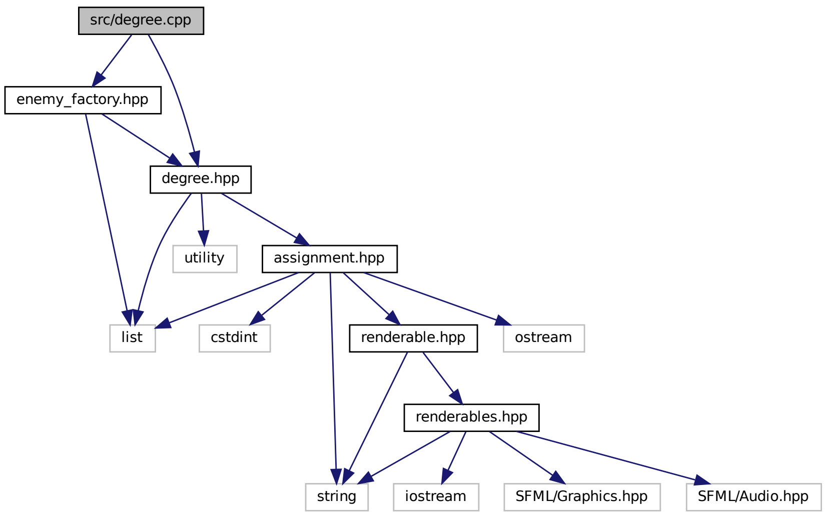
  A class for the buttons in the game.
- class TowerButton

  A subclass of buttons, for building the towers.

14.21 src/degree.cpp File Reference

```
#include "degree.hpp"
#include "enemy_factory.hpp"
```

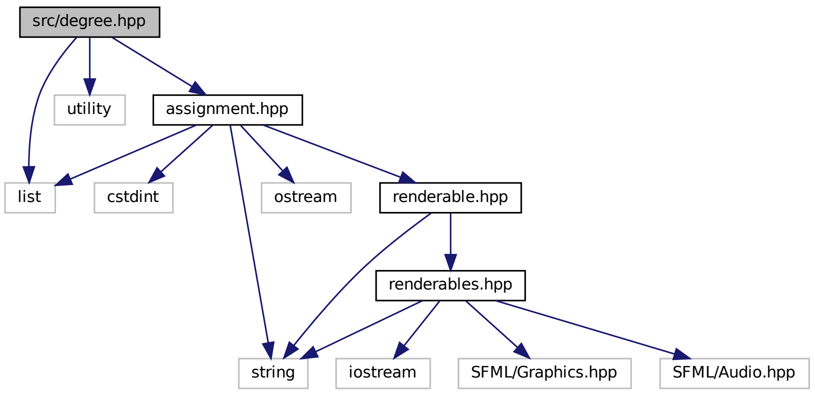
Include dependency graph for degree.cpp:



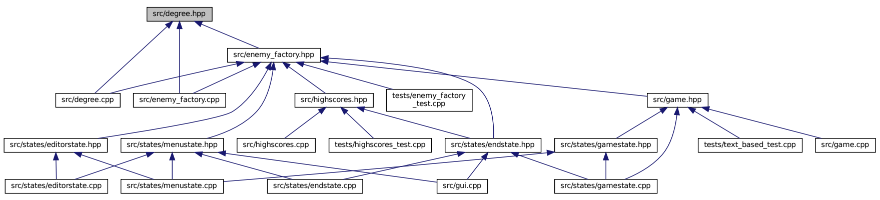
14.22 src/degree.hpp File Reference

```
#include <list>
#include <utility>
#include "assignment.hpp"
```

Include dependency graph for degree.hpp:



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



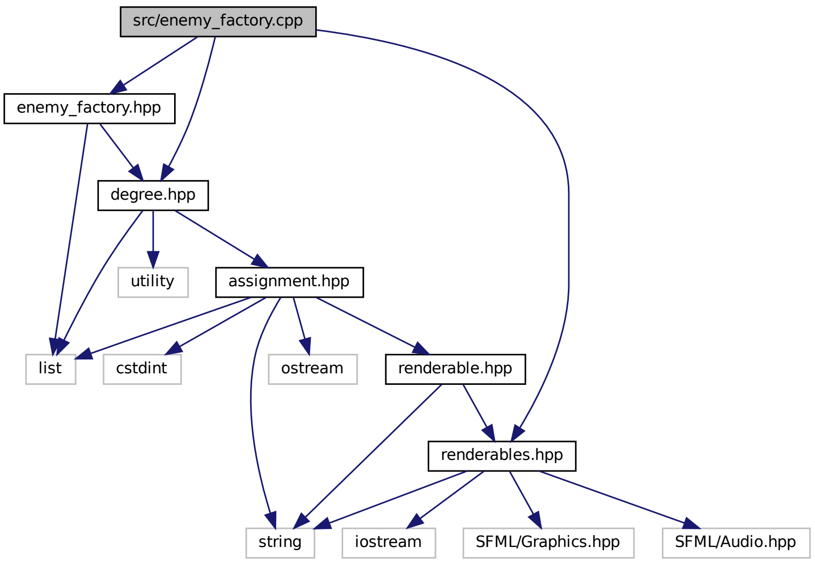
Classes

- class Degree

  A more advanced type of enemy Splits into multiple other enemies upon death The descendants are stored in the m_descendants collection as pairs where the first one is the type and the second one is the amount.

14.23 src/enemy_factory.cpp File Reference

```
#include "enemy_factory.hpp"
#include "degree.hpp"
#include "renderables.hpp"
```
Include dependency graph for enemy_factory.cpp:



Macros

- #define DETAILED_DEBUG_PRINT 0

Functions

- std::ostream & operator<< (std::ostream &os, const EnemyFactory &ef)

14.23.1 Macro Definition Documentation

14.23.1.1 DETAILED_DEBUG_PRINT

```
#define DETAILED_DEBUG_PRINT 0
```

14.23.2 Function Documentation

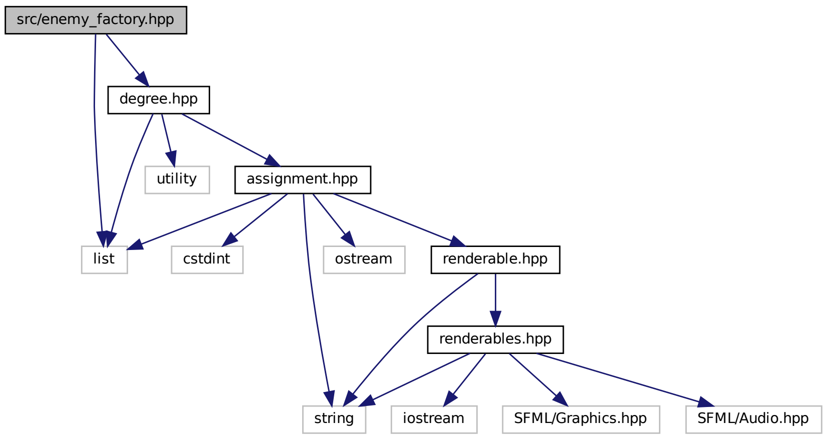
14.23.2.1 operator<<()

```
std::ostream& operator<< (
             std::ostream & os,
             const EnemyFactory & ef )
```

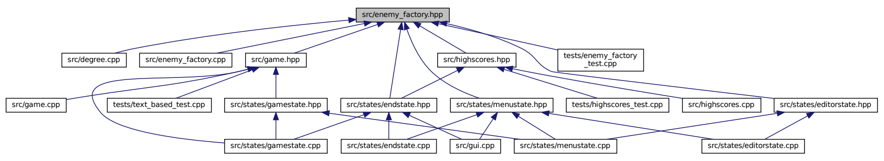
14.24 src/enemy_factory.hpp File Reference

```
#include <list>
#include "degree.hpp"
```

Include dependency graph for enemy_factory.hpp:



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



Classes

- class EnemyFactory

    A class which handles the logic about what enemies come and how much of during each round.

Enumerations

- enum Difficulty { Easy, Medium, Hard }

    An enumeration for the game difficulties.

14.24.1 Enumeration Type Documentation

14.24.1.1 Difficulty

```
enum Difficulty
```

An enumeration for the game difficulties.

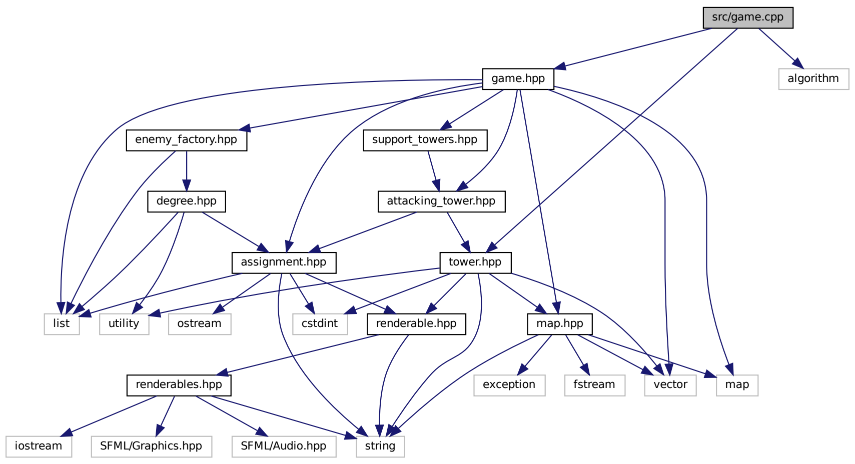
Enumerator

Easy	
Medium	
Hard	

14.25 src/game.cpp File Reference

```
#include "game.hpp"
#include <algorithm>
#include "tower.hpp"
```

Include dependency graph for game.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const Game &game)

14.25.1 Function Documentation

14.25.1.1 operator<<()

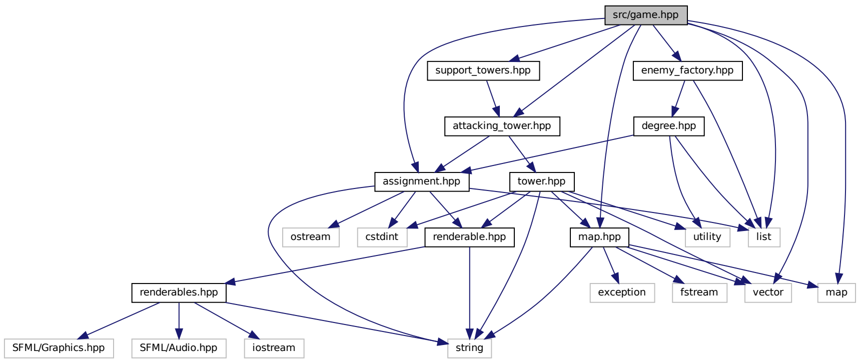
```
std::ostream& operator<< (
            std::ostream & os,
            const Game & game )
```

14.26 src/game.hpp File Reference

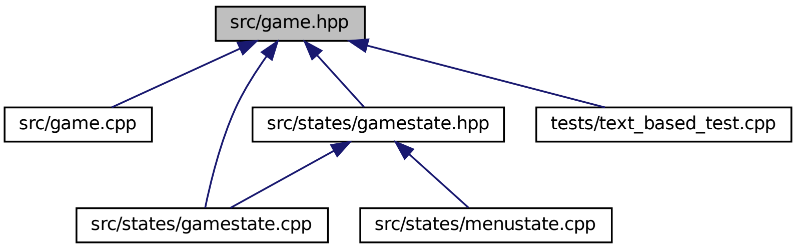
```
#include <list>
#include <map>
#include <vector>
#include "assignment.hpp"
#include "attacking_tower.hpp"
#include "enemy_factory.hpp"
#include "map.hpp"
```

```
#include "support_towers.hpp"
```

Include dependency graph for game.hpp:



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



Classes

- class Game

  A class which holds the logic of the game.

Enumerations

- enum Action {
  BuyFreshman, BuyTeekkari, BuyBachelor, BuyMaster,
  BuyDoctor, BuyCalculator, BuyCoffeeTable, UpgradeTower,
  DestroyTower }

  An enumeration for different actions on grid cells Used to test the possibility of actions.

14.26.1 Enumeration Type Documentation

14.26.1.1 Action

```
enum Action
```

An enumeration for different actions on grid cells Used to test the possibility of actions.

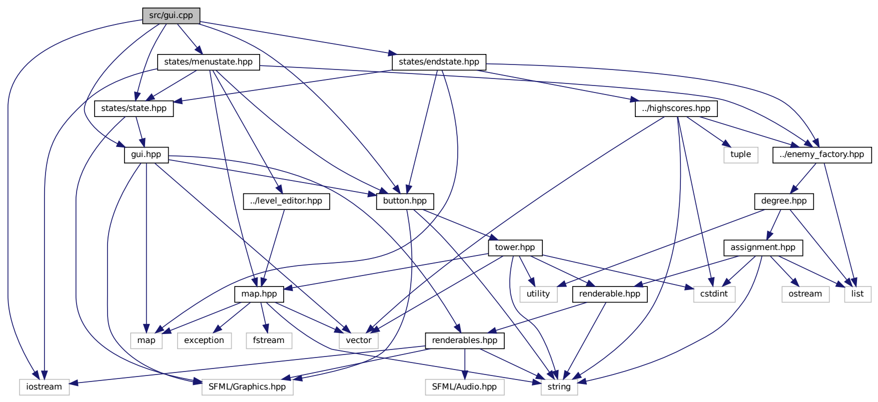
Enumerator

BuyFreshman	
BuyTeekkari	
BuyBachelor	
BuyMaster	
BuyDoctor	
BuyCalculator	
BuyCoffeeTable	
UpgradeTower	
DestroyTower	

14.27 src/gui.cpp File Reference

```
#include "gui.hpp"
#include <iostream>
#include "button.hpp"
#include "states/state.hpp"
#include "states/menustate.hpp"
#include "states/endstate.hpp"
```

Include dependency graph for gui.cpp:

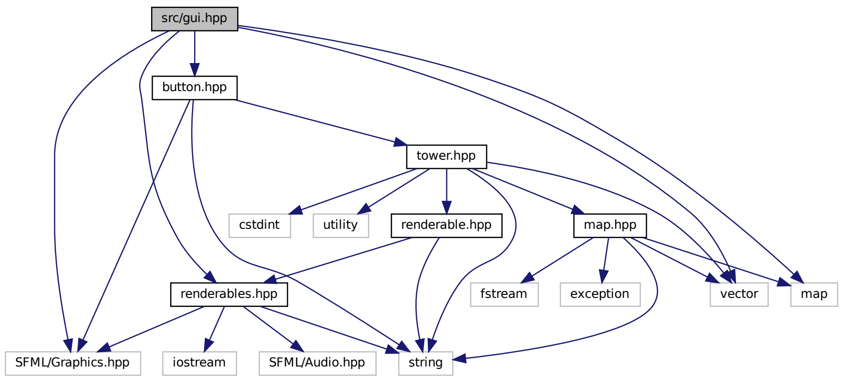


14.28 src/gui.hpp File Reference

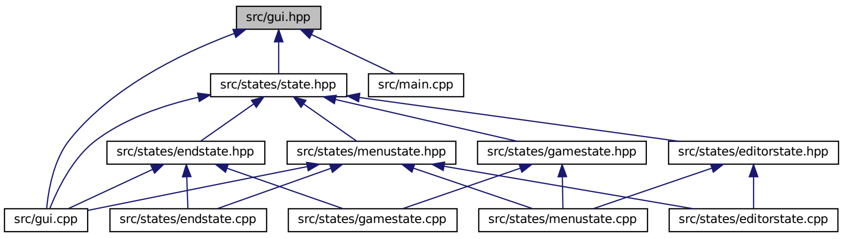
```
#include <SFML/Graphics.hpp>
#include <map>
#include <vector>
#include "button.hpp"
```

```
#include "renderables.hpp"
```

Include dependency graph for gui.hpp:



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



Classes

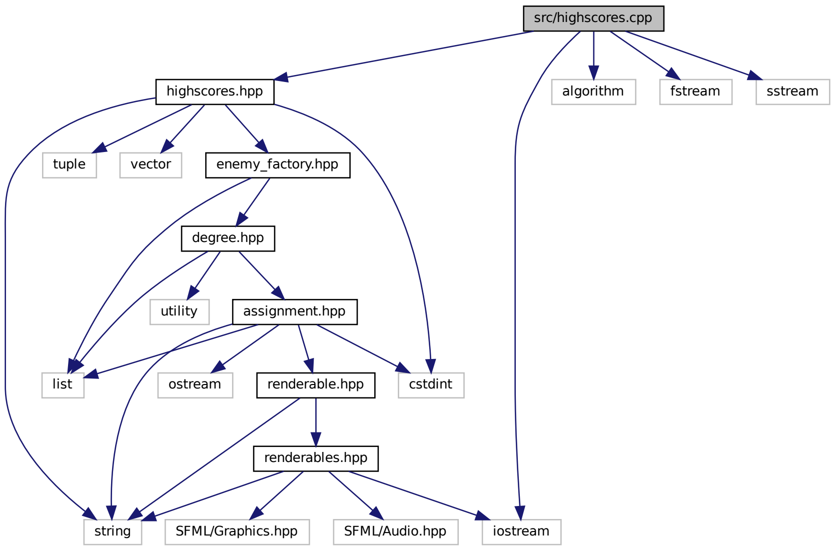
- class GUI

  A class to add elements to the Graphical User Interface.

14.29 src/highscores.cpp File Reference

```
#include "highscores.hpp"
#include <algorithm>
#include <fstream>
#include <iostream>
```

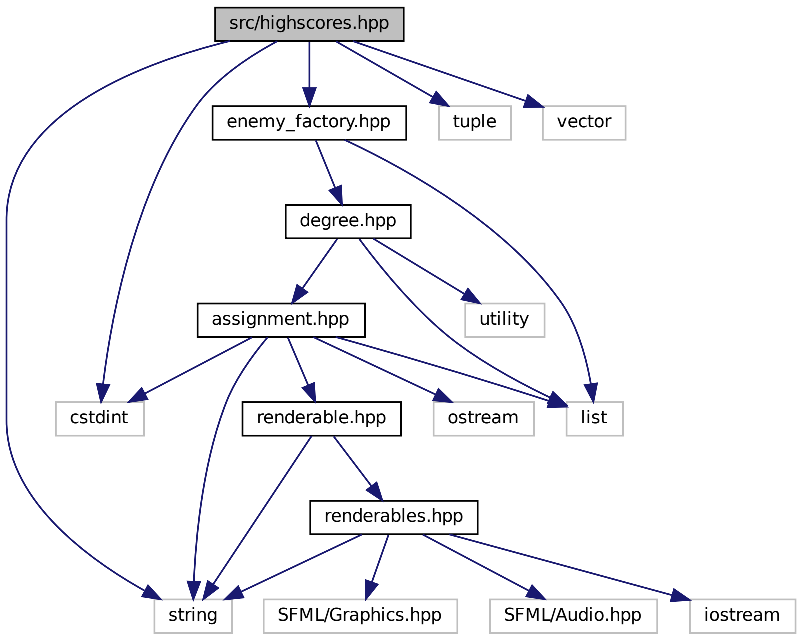
```
#include <sstream>
```
Include dependency graph for highscores.cpp:



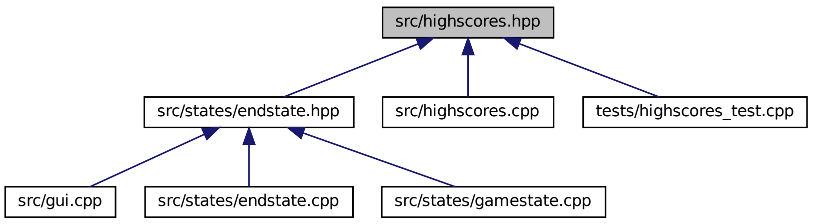
14.30 src/highscores.hpp File Reference

```
#include <cstdint>
#include <string>
#include <tuple>
#include <vector>
#include "enemy_factory.hpp"
```

Include dependency graph for highscores.hpp:



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



Classes

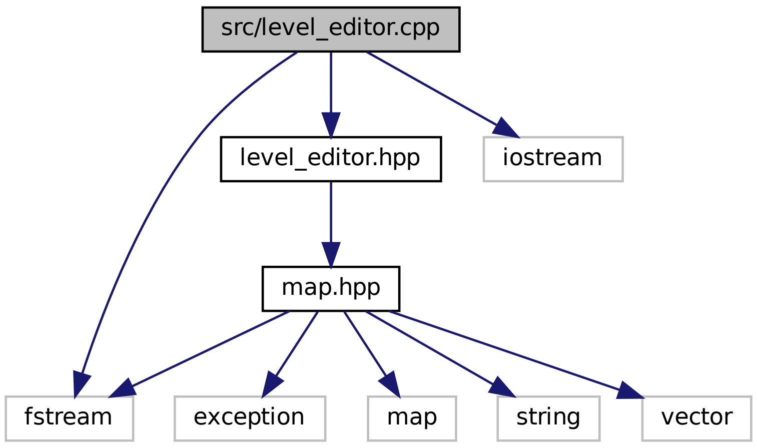
- class Highscores

    A class used to handle the high score savings at the end of game.

14.31 src/level_editor.cpp File Reference

```
#include "level_editor.hpp"
#include <fstream>
```

```
#include <iostream>
```
Include dependency graph for level_editor.cpp:



Macros

- #define PRINT_EDITOR_ERRORS false

14.31.1 Macro Definition Documentation

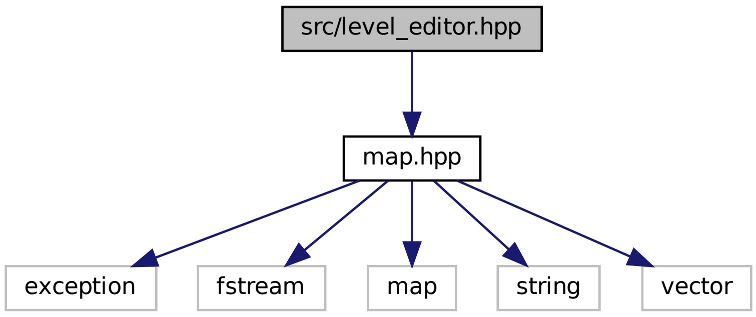
14.31.1.1 PRINT_EDITOR_ERRORS

```
#define PRINT_EDITOR_ERRORS false
```

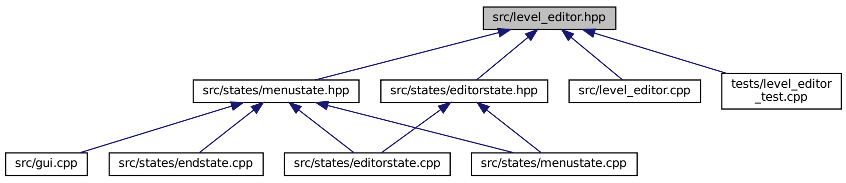
14.32 src/level_editor.hpp File Reference

```
#include "map.hpp"
```

Include dependency graph for level_editor.hpp:



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



Classes

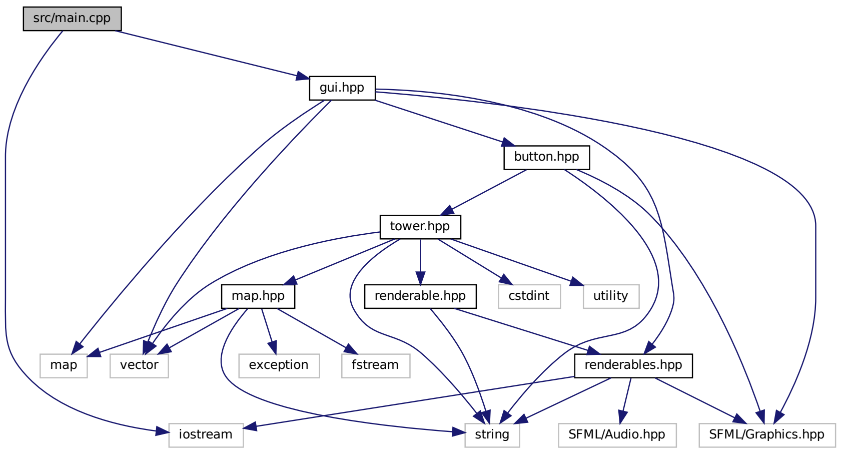
- class LevelEditor

  The logical core of the level editor state.

14.33 src/main.cpp File Reference

```
#include <iostream>
#include "gui.hpp"
```

Include dependency graph for main.cpp:



Functions

- int main ()

14.33.1 Function Documentation

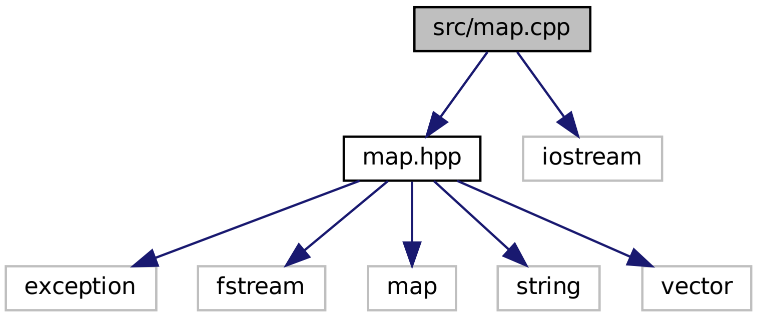
14.33.1.1 main()

```
int main ( )
```

14.34 src/map.cpp File Reference

```
#include "map.hpp"
#include <iostream>
```

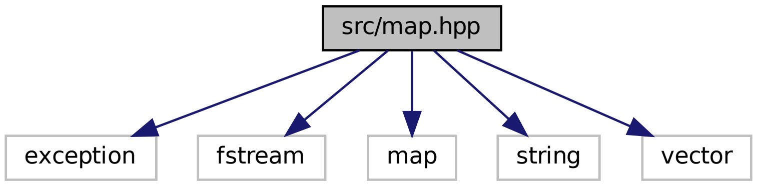
Include dependency graph for map.cpp:



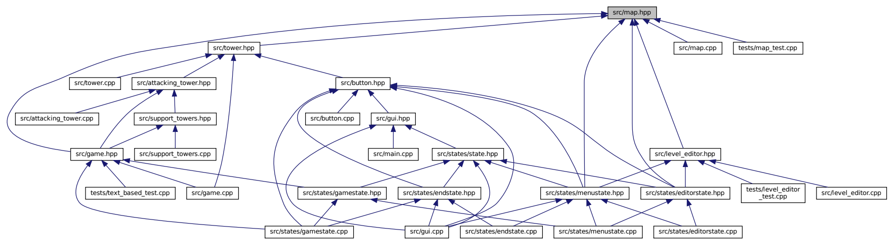
14.35 src/map.hpp File Reference

```
#include <exception>
#include <fstream>
#include <map>
#include <string>
#include <vector>
```

Include dependency graph for map.hpp:



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



Classes

- class Map

    Map class that represents the grid map system behind each unique level.

Enumerations

- enum TileType { towerTile, startTile, pathTile, endTile }

    Grid tile enumeration representing different kinds of map positions.

14.35.1 Enumeration Type Documentation

14.35.1.1 TileType

```
enum TileType
```

Grid tile enumeration representing different kinds of map positions.

Enumerator

towerTile	
startTile	
pathTile	
endTile	

14.36 src/maps/map1.txt File Reference

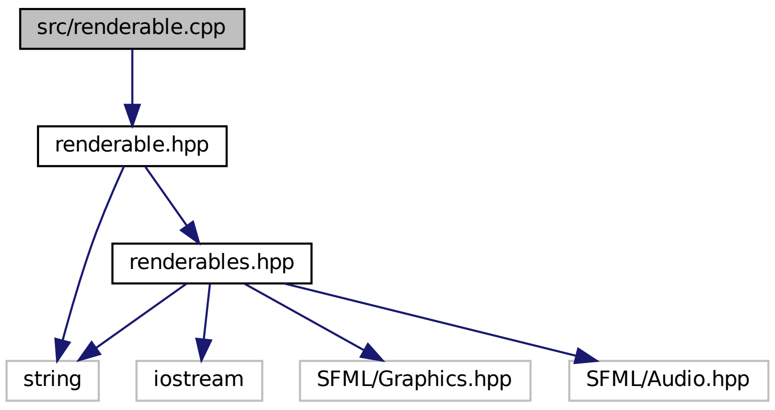
14.37 src/maps/map2.txt File Reference

14.38 src/maps/map3.txt File Reference

14.39 src/renderable.cpp File Reference

```
#include "renderable.hpp"
```

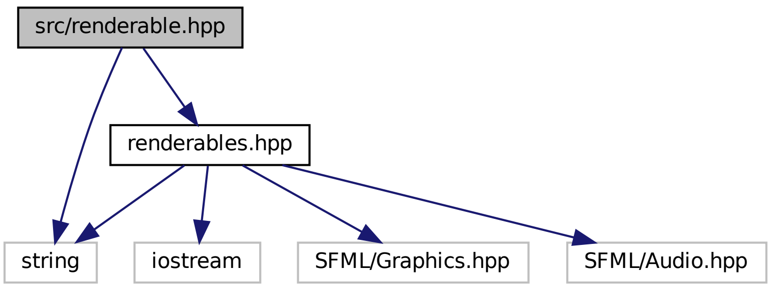
Include dependency graph for renderable.cpp:



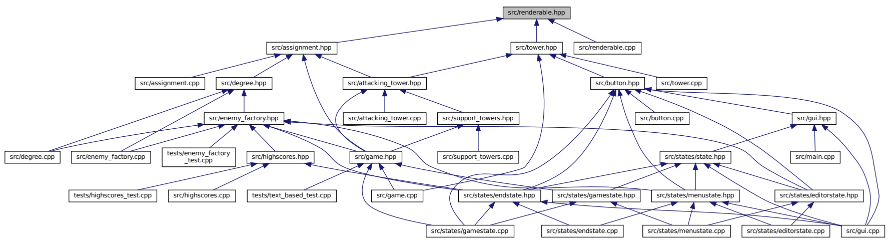
14.40 src/renderable.hpp File Reference

```
#include <string>
#include "renderables.hpp"
```

Include dependency graph for renderable.hpp:



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



Classes

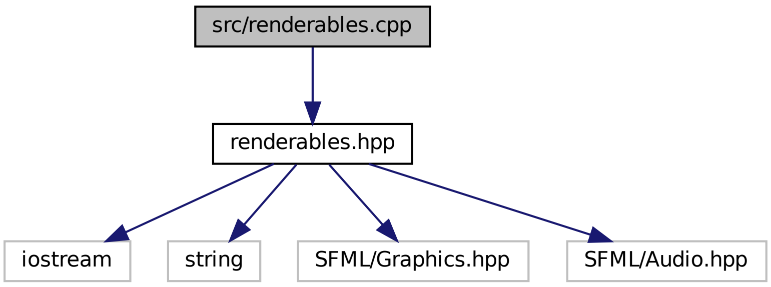
- class Renderable

    A class which encapsulates the renderable objects (enemies and towers) Is not supposed to be directly instanciated, so the constructor is protected.

14.41 src/renderables.cpp File Reference

```
#include "renderables.hpp"
```

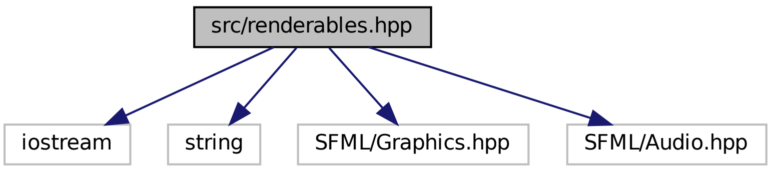
Include dependency graph for renderables.cpp:



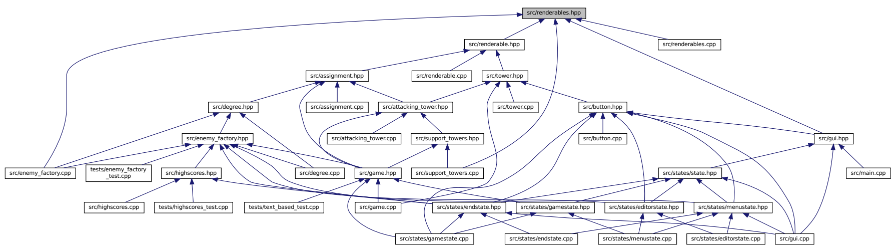
14.42 src/renderables.hpp File Reference

```
#include <iostream>
#include <string>
#include "SFML/Graphics.hpp"
#include "SFML/Audio.hpp"
```

Include dependency graph for renderables.hpp:



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



Classes

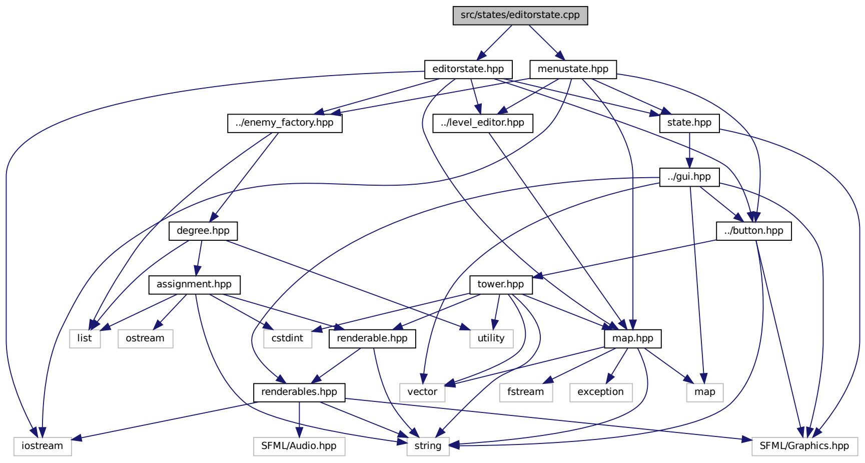
- class Renderables

  A class which handles the textures for different sprites One instance of this class must be constructed somewhere in the code before any static getters are accessed!

14.43 src/states/editorstate.cpp File Reference

```
#include "editorstate.hpp"
#include "menustate.hpp"
```

Include dependency graph for editorstate.cpp:



Macros

- #define TILE_SIZE 30

14.43.1 Macro Definition Documentation

14.43.1.1 TILE_SIZE

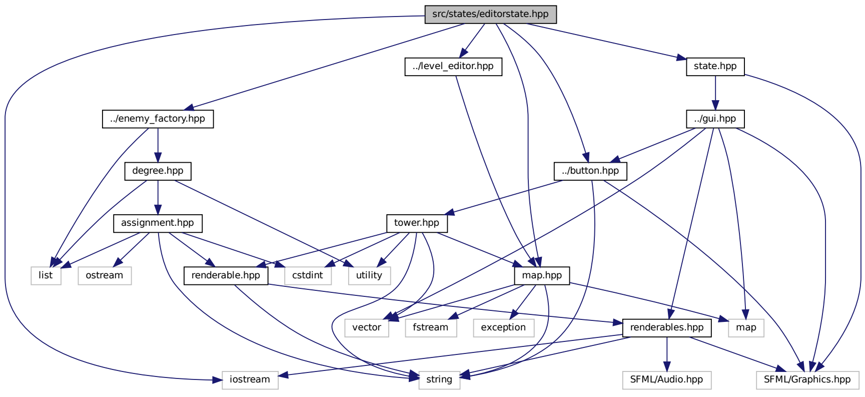
```
#define TILE_SIZE 30
```

14.44 src/states/editorstate.hpp File Reference

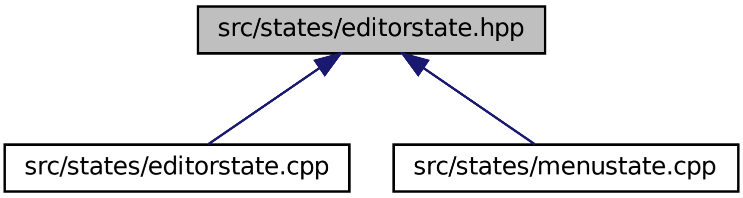
```
#include <iostream>
#include "../button.hpp"
#include "../enemy_factory.hpp"
#include "../level_editor.hpp"
#include "../map.hpp"
```

```
#include "state.hpp"
```

Include dependency graph for editorstate.hpp:



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



Classes

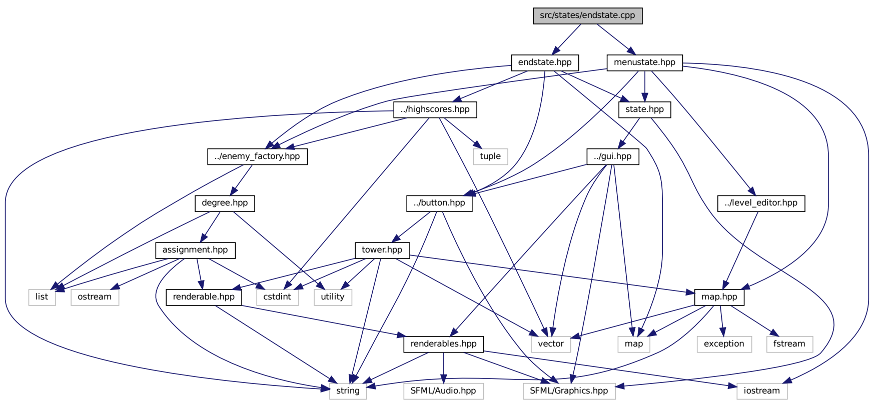
- class EditorState

  A gamestate run by GUI corresponding to the level editor.

14.45 src/states/endstate.cpp File Reference

```
#include "endstate.hpp"
#include "menustate.hpp"
```

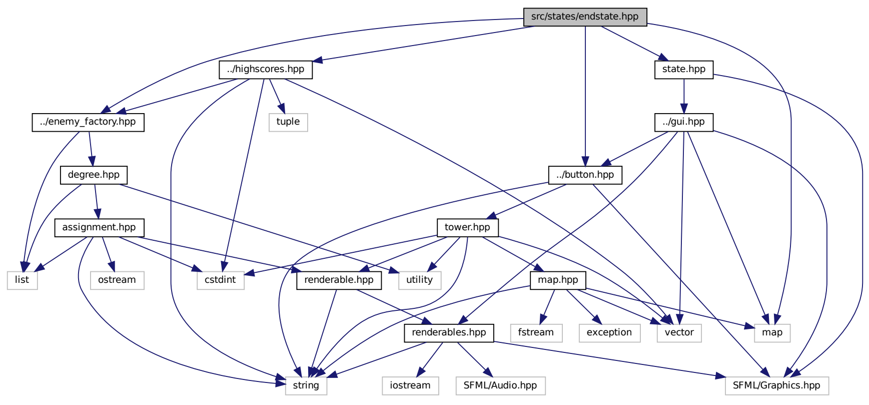
Include dependency graph for endstate.cpp:



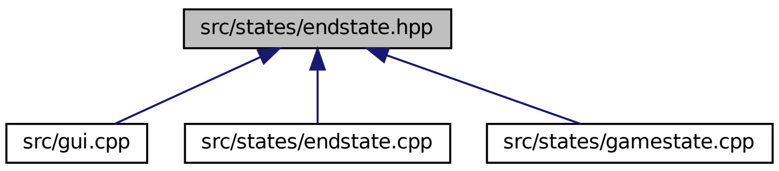
14.46 src/states/endstate.hpp File Reference

```
#include <map>
#include "../button.hpp"
#include "../enemy_factory.hpp"
#include "../highscores.hpp"
#include "state.hpp"
```

Include dependency graph for endstate.hpp:



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



Classes

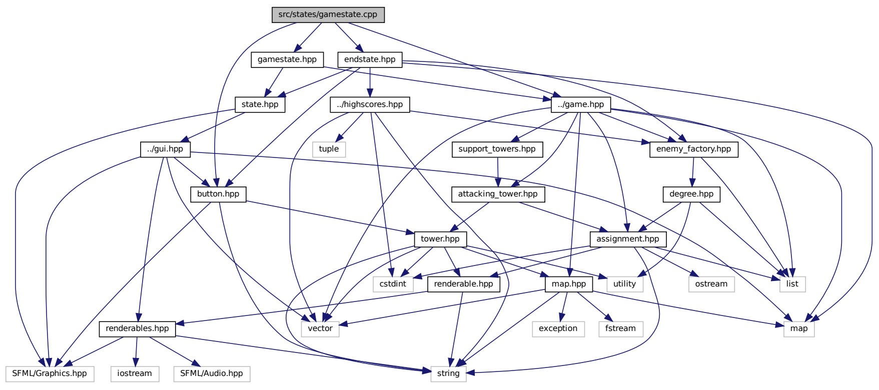
- class EndState

  A gamestate run by GUI corresponding to the game over screen.

14.47 src/states/gamestate.cpp File Reference

```
#include "gamestate.hpp"
#include "button.hpp"
#include "endstate.hpp"
#include "game.hpp"
```

Include dependency graph for gamestate.cpp:



Macros

- #define ANIMATION_LENGTH 20
- #define TILE_SIZE 30
- #define PROJECTILE_RADIUS 5

14.47.1 Macro Definition Documentation

14.47.1.1 ANIMATION_LENGTH

```
#define ANIMATION_LENGTH 20
```

14.47.1.2 PROJECTILE_RADIUS

```
#define PROJECTILE_RADIUS 5
```

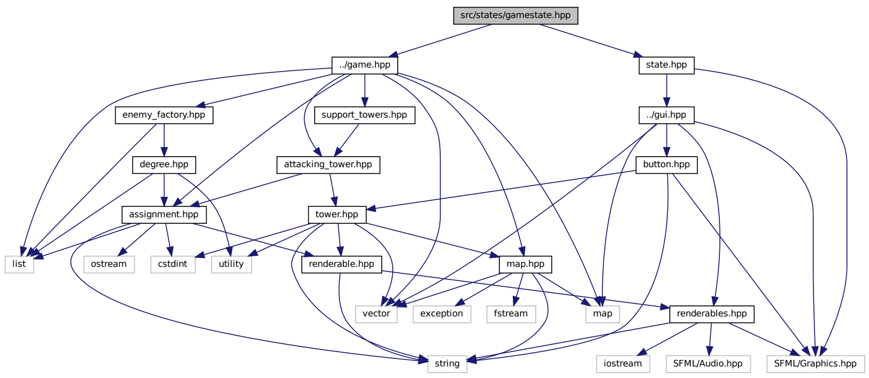
14.47.1.3 TILE_SIZE

```
#define TILE_SIZE 30
```

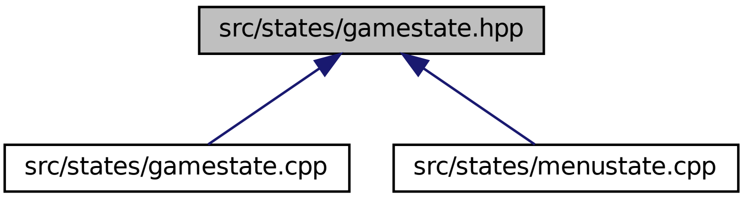
14.48 src/states/gamestate.hpp File Reference

```
#include "../game.hpp"
#include "state.hpp"
```

Include dependency graph for gamestate.hpp:



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



Classes

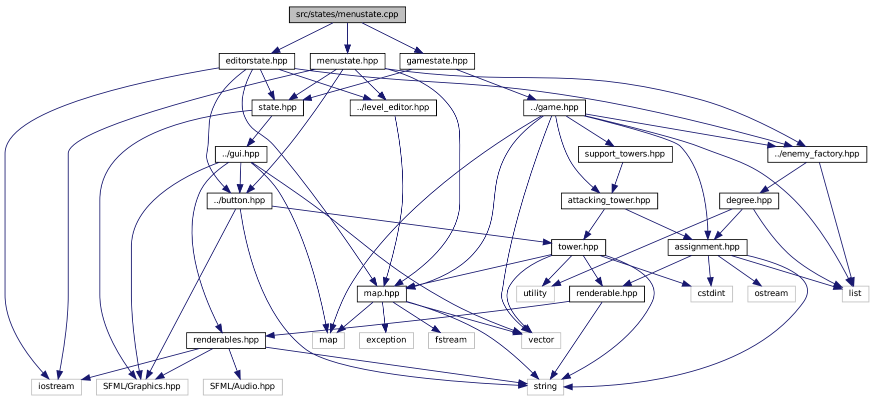
- class GameState

  GameState class runs and draws the game part of the software.

14.49 src/states/menustate.cpp File Reference

```
#include "menustate.hpp"
#include "editorstate.hpp"
#include "gamestate.hpp"
```

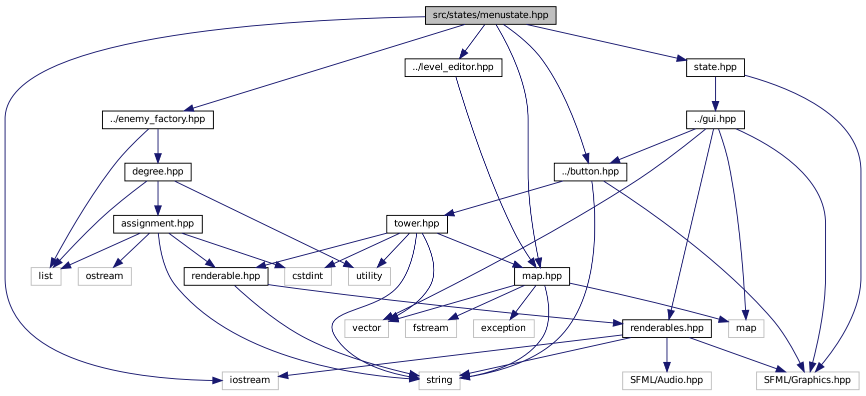
Include dependency graph for menustate.cpp:



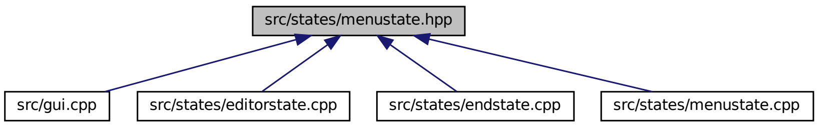
14.50 src/states/menustate.hpp File Reference

```
#include <iostream>
#include "../button.hpp"
#include "../enemy_factory.hpp"
#include "../level_editor.hpp"
#include "../map.hpp"
#include "state.hpp"
```

Include dependency graph for menustate.hpp:



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



Classes

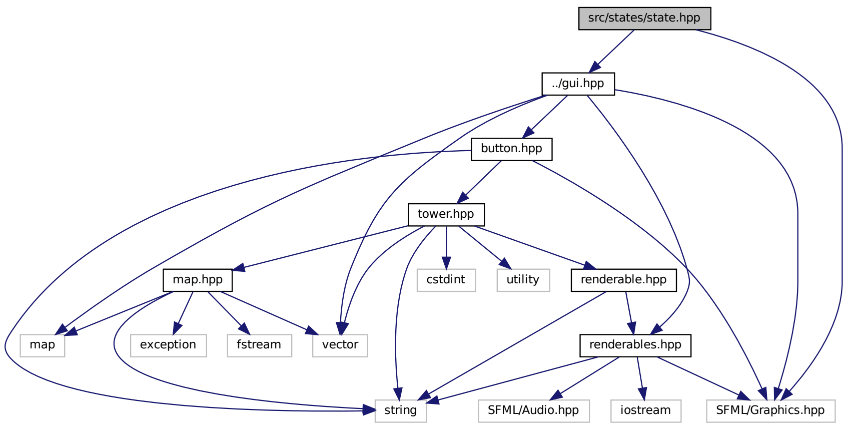
- class MenuState

  A state run by GUI corresponding to the main menu state.

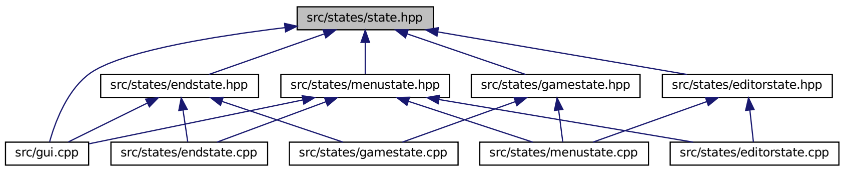
14.51 src/states/state.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "../gui.hpp"
```

Include dependency graph for state.hpp:



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



Classes

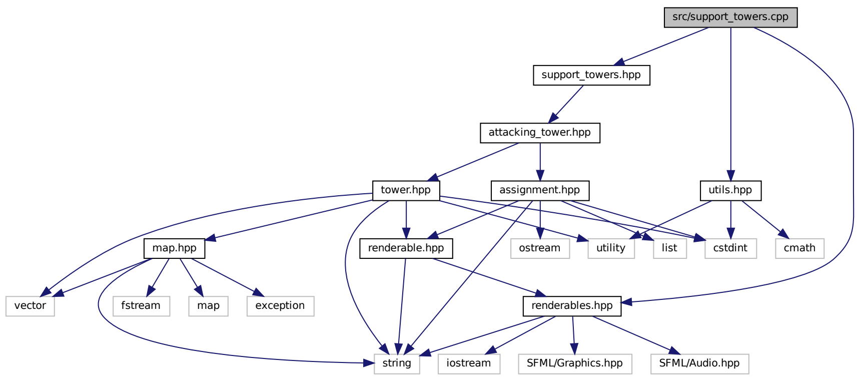
- class State

  Abstract State class represents the classes that run and draw the different software states.

14.52 src/support_towers.cpp File Reference

```
#include "support_towers.hpp"
#include "utils.hpp"
#include "renderables.hpp"
```

Include dependency graph for support_towers.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const SupportTower &st)

14.52.1 Function Documentation

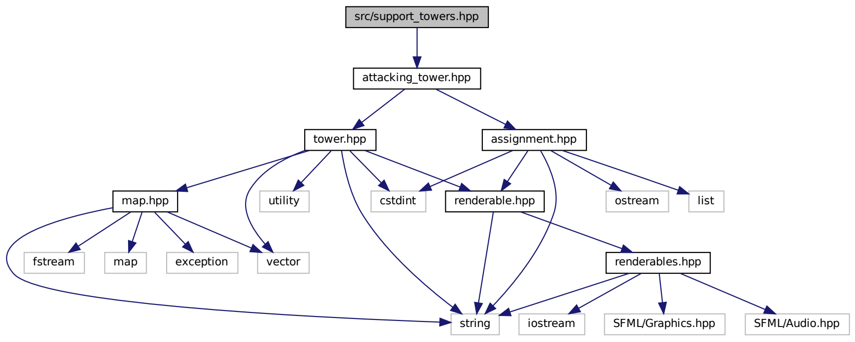
14.52.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            const SupportTower & st )
```

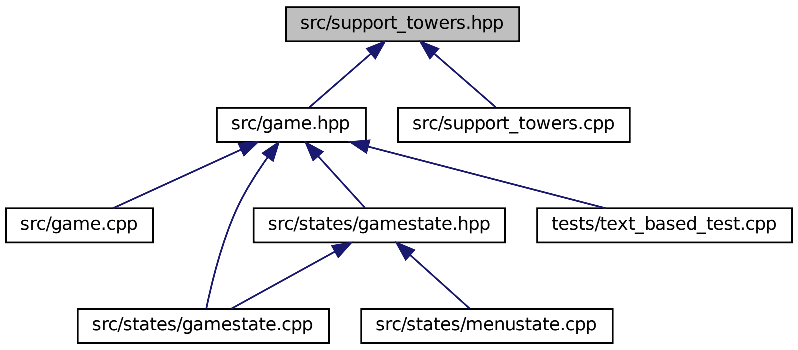
14.53 src/support_towers.hpp File Reference

```
#include "attacking_tower.hpp"
```

Include dependency graph for support_towers.hpp:



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



Classes

- class SupportTower

  A virtual base class for the supporting towers Cannot be directly instanciated.
- class BuffTower

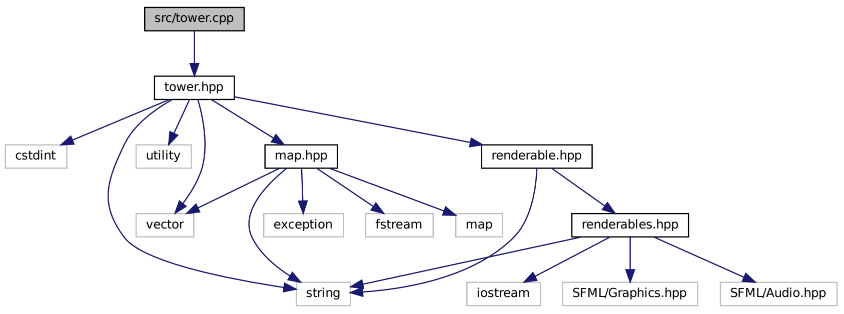
  A tower which buffs attacking towers making the do more damage.
- class HealTower

  A tower which heals attacking towers.

14.54 src/tower.cpp File Reference

```
#include "tower.hpp"
```

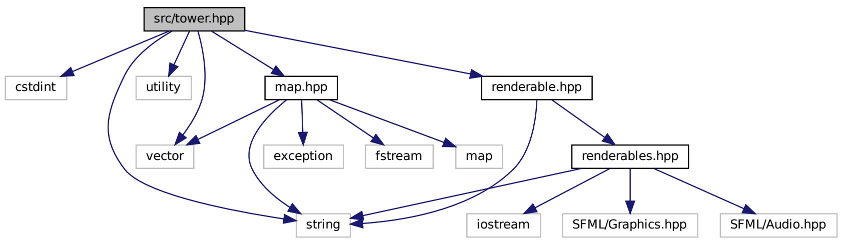
Include dependency graph for tower.cpp:



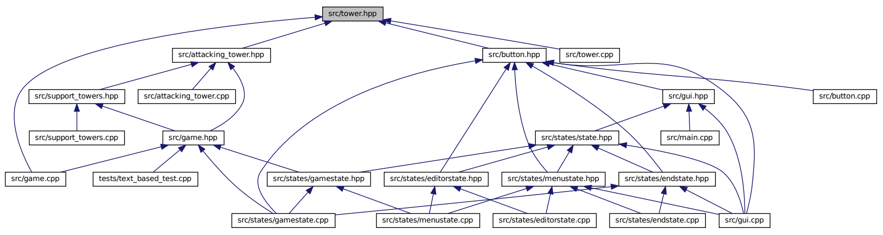
14.55 src/tower.hpp File Reference

```
#include <cstdint>
#include <string>
#include <utility>
#include <vector>
#include "map.hpp"
#include "renderable.hpp"
```

Include dependency graph for tower.hpp:



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



Classes

- class Tower

  An abstract base-class for the towers Sub-classes will be Attacking towers and supporting towers. Cannot be directly instanciated.

Macros

- #define TILE_SIZE 30

Enumerations

- enum TowerType {
  Freshman, Teekkari, Bachelor, Master,
  Doctor, Calculator, CoffeeTable }

  An enumeration for the different towertypes.

14.55.1 Macro Definition Documentation

14.55.1.1 TILE_SIZE

```
#define TILE_SIZE 30
```

14.55.2 Enumeration Type Documentation

14.55.2.1 TowerType

```
enum TowerType
```

An enumeration for the different towertypes.

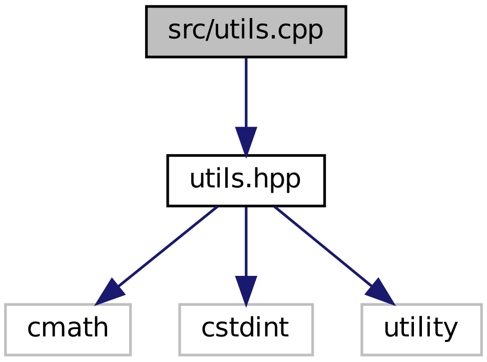
Enumerator

Freshman	
Teekkari	
Bachelor	
Master	
Doctor	
Calculator	
CoffeeTable	

14.56 src/utils.cpp File Reference

```
#include "utils.hpp"
```

Include dependency graph for utils.cpp:



Namespaces

- UtilFunctions

    A namespace containing some utility functions needed in the project.

Functions

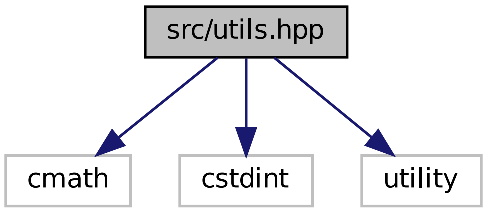
- float UtilFunctions::distance (const std::pair< int32_t, int32_t > &c1, const std::pair< int32_t, int32_t > &c2)

    Calculates the Euclidean distance between two coordinate pairds.

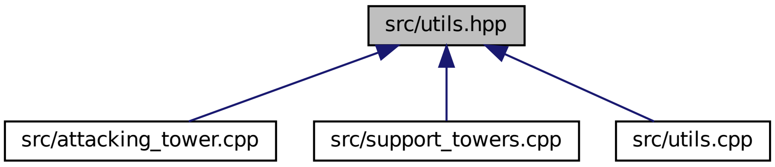
14.57 src/utils.hpp File Reference

```
#include <cmath>
#include <cstdint>
#include <utility>
```

Include dependency graph for utils.hpp:



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



Namespaces

- UtilFunctions

  A namespace containing some utility functions needed in the project.

Functions

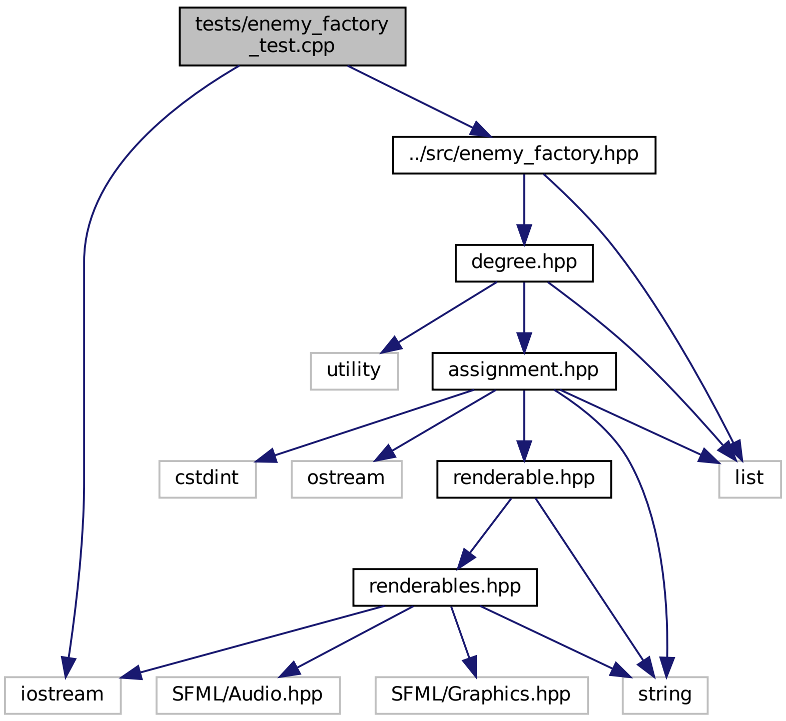
- float UtilFunctions::distance (const std::pair< int32_t, int32_t > &c1, const std::pair< int32_t, int32_t > &c2)

  Calculates the Euclidean distance between two coordinate pairds.

14.58 tests/enemy_factory_test.cpp File Reference

```
#include <iostream>
#include "../src/enemy_factory.hpp"
```

Include dependency graph for enemy_factory_test.cpp:



Functions

- int main ()

14.58.1 Function Documentation

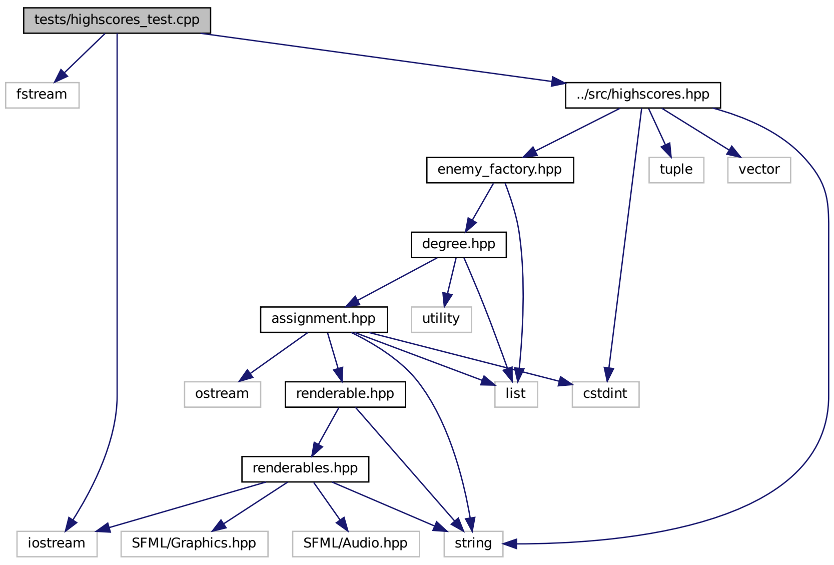
14.58.1.1 main()

```
int main ( )
```

14.59 tests/highscores_test.cpp File Reference

```
#include <fstream>
#include <iostream>
```

```
#include "../src/highscores.hpp"
```
Include dependency graph for highscores_test.cpp:



Functions

- int main ()

14.59.1 Function Documentation

14.59.1.1 main()

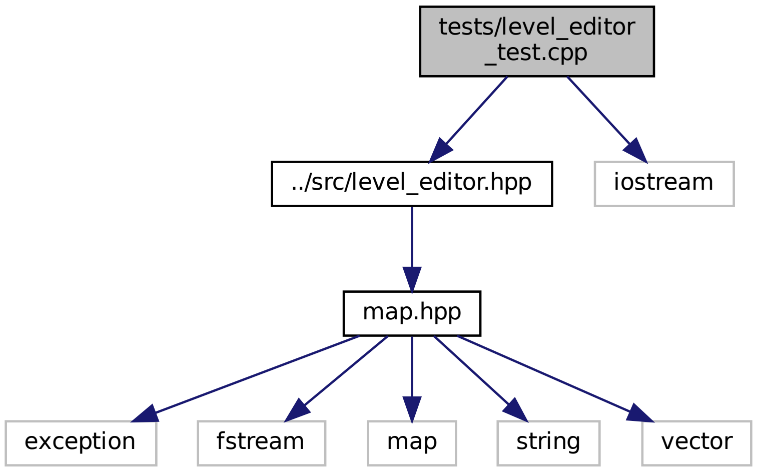
```
int main ( )
```

14.60 tests/level_editor_test.cpp File Reference

Used to test the LevelEditor class.

```
#include "../src/level_editor.hpp"
#include <iostream>
```

Include dependency graph for level_editor_test.cpp:



Functions

- int main (void)

14.60.1 Detailed Description

Used to test the LevelEditor class.

14.60.2 Function Documentation

14.60.2.1 main()

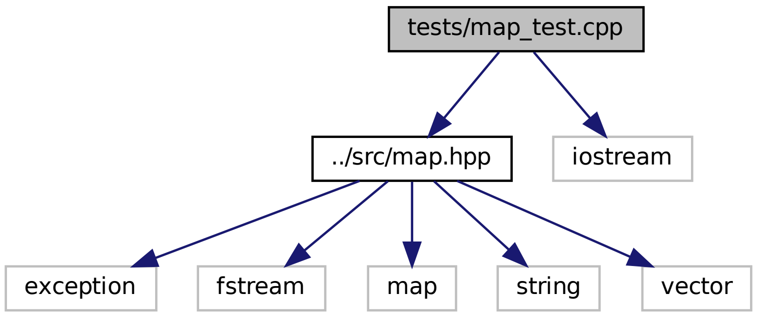
```
int main (
            void )
```

14.61 tests/map_test.cpp File Reference

Used to test the Map class.

```
#include "../src/map.hpp"
#include <iostream>
```

Include dependency graph for map_test.cpp:



Functions

- int main ()

14.61.1 Detailed Description

Used to test the Map class.

14.61.2 Function Documentation

14.61.2.1 main()

```
int main ( )
```

14.62 tests/testmap1.txt File Reference

14.63 tests/testmap2.txt File Reference

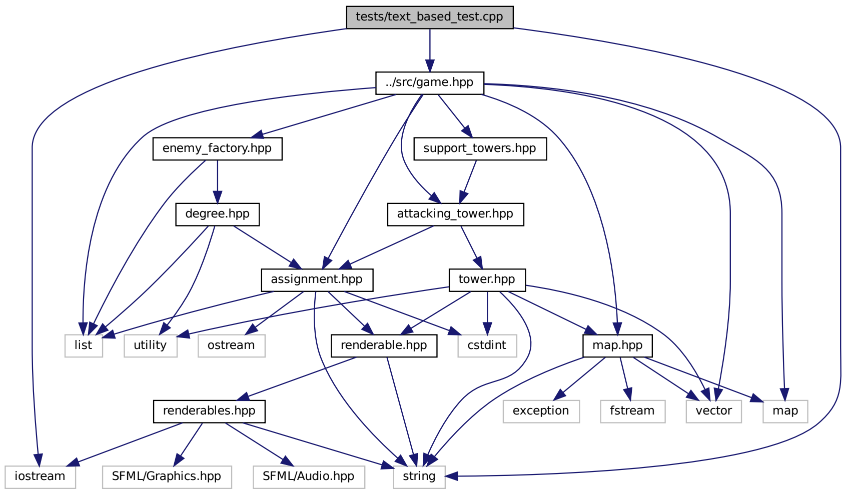
14.64 tests/testmapWrong.txt File Reference

14.65 tests/text_based_test.cpp File Reference

Used to test the game running with text output to terminal.

```
#include <iostream>
#include <string>
#include "../src/game.hpp"
```

Include dependency graph for text_based_test.cpp:



Functions

- int main (void)

14.65.1 Detailed Description

Used to test the game running with text output to terminal.

14.65.2 Function Documentation

14.65.2.1 main()

```
int main (
            void  )
```

Index