Cats vs Rats TD

Generated by Doxygen 1.9.1

Chapter 1

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 2

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.

2.1 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 inlcude files of the dependent library should also be placed in this folder.

Chapter 3

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

3.1 Meeting 09.11.2022 19::30

Participants:

1. Henrik Turtinen
2. Antti Chen

3.1.1 Summary of works

1. Henrik and Antti started Planning the project. See "Project Status".

3.1.2 Challenges

1. ...

3.1.3 Actions

1. All: Next team meeting will be held tomorrow 10.11.2022. The goal is to complete project plan.

   Please reflect these action decisions in your git commit messages so that your group members and advisor can follow the progress.

3.1.4 Project status

Writing project plan has been started. It includes a sketch of the Game ui, and the overall idea of the Game. The Game features rodents that attack and cats that protect their base. There will be different types of cats, maps, enemies etc.

Aalto GitLab repository has been made and shared for all group members.

3.1.4.1 TODOs

1. All: Finish Project plan for tower defense, most important parts are to choose graphics library (SFML, SDL, Qt), define scope of the work and how work is shared.

3.2 Meeting 10.11.2022 14::00

Participants:

1. Kasperi Kaivola
2. Henrik Turtinen
3. Otso Luukkanen

3.2.1 Summary of works

1. Henrik Turtinen

   Started the project plan. It includes sketch of the Game UI and the basic idea: cats against rodents. Created GitLab repo to version.aalto.fi
2. Antti Chen

   Project plan: same as above

3.2.2 Challenges

1. ...

3.2.3 Actions

1. Otso: Reserve team meeting with course assistant next week. Use Telegram poll to see what dates are available.
2. All: No coding before next meeting.

Please reflect these action decisions in your git commit messages so that your group members and advisor can follow the progress.

3.2.4 Project status

Project plan has been completed, see /plan directory. Meeting with course assistant and review of the project plan.

Framework: SFML

3.2.4.1 TODOs

1. Member 1: Write an action.
2. ...

3.3 Meeting 16.11.2022 12::00

Participants:

1. Kasperi Kaivola
2. Henrik Turtinen
3. Otso Luukkanen
4. Antti Chen
5. Mark Heidmets (course assistant)

3.3.1 Summary of works

No work done between this and the last meeting.

3.3.2 Challenges

3.3.3 Actions

1. All: Implement game base class, that handles things like starting the actual game, updating the game window (fps), rendering the window and processing game inputs.
2. All: Read through book SFML Game Development, especially parts that talk about game main class.
3. All: take a look at code snippet that project advisor / course assistant shared in our groups Team group. It has an fps lock implementation.

   Please reflect these action decisions in your git commit messages so that your group members and advisor can follow the progress.

3.3.4 Project status

Project plan was ok according to the advisor.

3.3.4.1 TODOs

1. Member 1: Write an action.
2. ...

3.4 Meeting 22.11.2022 18:00

Participants:

1. Henrik Turtinen
2. Antti Chen
3. Kasperi Kaivola
4. Otso Luukkanen

3.4.1 Summary of works

1. Henrik and Antti
   Started to work on resources class
2. Kasperi and Otso
   Started to work on creating the game world

3.4.2 Challenges

3.4.3 Actions

3.4.4 Project status

The project is running late on schedule

3.4.4.1 TODOs

1. All: continue working on the project

3.5 Meeting 04.12.2022 21:00

Participants:

1. Henrik Turtinen
2. Antti Chen
3. Kasperi Kaivola
4. Otso Luukkanen

3.5.1 Summary of works

Overall it has been a really busy week and we haven't had time to work on the project a lot.

1. Henrik and Antti
   Continued working on resources class
2. Kasperi and Otso
   Continued working on creating the game world

3.5.2 Challenges

3.5.3 Actions

3.5.4 Project status

The project is running late on schedule

3.5.4.1 TODOs

1. All: continue working on the project during the coming week. Other deadlines are behind and we have plenty of time to work on the project.

3.6 Meeting 06.12.2022 13:30

Participants:

1. Henrik Turtinen
2. Antti Chen
3. Kasperi Kaivola
4. Otso Luukkanen

3.6.1 Summary of works

1. Henrik and Antti Didn't have time to do anything.
2. Kasperi and Otso Didn't have time to do anything.

3.6.2 Challenges

1. We have one week left and most of the game to be made. W

3.6.3 Actions

We've committed to use most of our time to make the project.

3.6.4 Project status

The project is running late on schedule but we believe that we have enough time

3.6.4.1 TODOs

1. Antti and Henrik: Start to try to make the enemies move
2. Otso and Kasperi: Start to create UI.

3.7 Meeting 08.12.2022 16:30

Participants:

1. Henrik Turtinen
2. Antti Chen
3. Kasperi Kaivola
4. Otso Luukkanen

3.7.1 Summary of works

1. Henrik and Antti Got the enemies to move and created wave controller
2. Kasperi and Otso Made main menu and level selection menu Edited map rendering

3.7.2 Challenges

3.7.3 Actions

3.7.4 Project status

We're on track to have the project ready on time.

3.7.4.1 TODOs

1. Antti and Henrik: Create towers
2. Otso and Kasperi: Continue with the UI. Make sounds work.

3.8 Meeting 10.12.2022 21:00

Participants:

1. Henrik Turtinen
2. Antti Chen
3. Kasperi Kaivola
4. Otso Luukkanen

3.8.1 Summary of works

1. Henrik and Antti Got towers and projectiles to work.
2. Kasperi and Otso UI is quite good now, the game has different scenes and buttons. The UI also now shows relevant stats.

3.8.2 Challenges

3.8.3 Actions

3.8.4 Project status

We're pretty happy with the status of the project. We still need to do some documentation.

3.8.4.1 TODOs

1. Henrik: Finish documentation
2. Otso: Create readMe files
3. Kasperi: Create upgradeable towers
4. Antti: Clean up the code

Chapter 4

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 5

Tower Defense

This is the template for the projects. Please copy the project description here. You can use Markdown language to render it as formatted **HTML** file.

5.1 Group

- Henrik Turtinen, 787323
- Antti Chen, 780757
- Kasperi Kaivola, 792415
- Otso Luukkanen, 792305

5.2 Repository organization

Your project implementation should follow the skelaton organization in this repository. See readme.md files in each folder.

5.3 Project Implementation

You must use git repository for the work on the project, making frequent enough commits so that the project group (and course staff) can follow the progress.

The completed project work will be demonstrated to the group's advisor at a demo session. The final demonstrations are arranged on week 50. After the final demonstrations project group evaluates another project, and self-evaluates own project. In addition, project members will give a confidential individual assessment of each group member

The course staff should be able to easily compile the project work using makefile and related instructions provided in the git repository. The final output should be in the **master branch** of the git repository.

5.4 Working practices

Each project group is assigned an advisor from the project teaching personnel. There will be a dedicated Teams channel for each project topic to facilitate discussion between the groups in the same topic and the advisor.

The group should meet weekly. The weekly meeting does not need to be long if there are no special issues to discuss, and can be taken remotely as voice/video chat on the group Teams channel (or Zoom or other similar tool), preferably at a regular weekly time. In the meeting the group updates:

- What each member has done during the week
- Are there challenges or problems? Discuss the possible solutions
- Plan for the next week for everyone
- Deviations and changes to the project plan, if any
- After the meetings, the meeting notes will be committed to the project repository in the `Meeting-notes.md` file.
  - The commits within the week should have some commit messages referring to the meeting notes so that the project advisor can follow the progress.
  - **The meeting notes should be in English.**

  Everyone may not be able to participate to all meetings, but at least a couple of members should be present in each meeting. Regular absence from meetings will affect in individual evaluation.

5.5 Source code documentation

See doc folder, readMe files in subfolders and comments on code files

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.

This folder has folders for the games source and media files

Entity folder includes code files related to creating entitites, like towers and enemies

Game folder includes source files that make the game work, like files relating to the game world. This folder also includes the different scenes.

Media folder includes all of the textures, sounds etc. we used.

Resource folder includes the implementation of a resource holder.

SceneItem folder

UI folder includes source files related to our games user interface like the game status menu.

Chapter 7

Cats vs Rats UI library

This subfolder contains classes that are used in Game scene.

Class file names follow class names: MyClass -> MyClass.cpp, MyClass.hpp

Side menu uses the following classes:

- Button TODO
- GameStatusMenu
  - Element that shows wave number, enemies, HP and money in real time
- Grid
  - Holds Sector elements and allows to handle map in tiles when using mouse
- Price TODO
- Sector
  - Used in Grid class, allow to handle map in tiles when using mouse
- SelectTowerButton TODO
- TowerMenu
  - This class allows to build and sell towers in Game, uses mouse and keyboard inputs
- WavePause
  - Button that stops the game
- WaveStart
  - Button that starts the next wave

Chapter 8

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.

8.1 Unit Tests

8.1.1 Test of MyClass

Involved Classes:

Test File:

Results:

Chapter 9

Namespace Index

9.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 10

Hierarchical Index

10.1 Class Hierarchy

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

Chapter 11

Class Index

11.1 Class List

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

Chapter 12

File Index

12.1 File List

Here is a list of all files with brief descriptions:

Chapter 13

Namespace Documentation

13.1 Fonts Namespace Reference

Enumerations

- enum ID { GameTitleFont }

13.1.1 Enumeration Type Documentation

13.1.1.1 ID

```
enum Fonts::ID
```

Enumerator

GameTitleFont	

Definition at line 44 of file Resource.hpp.

```
44 { GameTitleFont };
```

13.2 Maps Namespace Reference

Enumerations

- enum ID { Map }

13.2.1 Enumeration Type Documentation

13.2.1.1 ID

```
enum Maps::ID
```

Enumerator

Map	

Definition at line 40 of file Resource.hpp.

```
40 { Map };
```

13.3 Scenes Namespace Reference

Enumerations

- enum class ID { GameTitle , LevelSelect , MapScene , GameEnd }

13.3.1 Enumeration Type Documentation

13.3.1.1 ID

```
enum Scenes::ID  [strong]
```

Enumerator

GameTitle	
LevelSelect	Title menu of the game, has name and list of creators.
MapScene	Main menu for selecting map.
GameEnd	Game and its GUI. For Game Over Scene

Definition at line 17 of file Scene.hpp.

```
17              {
18   GameTitle, /// Title menu of the game, has name and list of creators
19   LevelSelect, /// Main menu for selecting map
20   MapScene, /// Game and its GUI
21   GameEnd /// For Game Over Scene
22 };
```

13.4 SoundBuffers Namespace Reference

Enumerations

- enum ID {
  EnemyDeath , Explosion , GunCat , BombCatMeow ,
  FreezeCatMeow }

13.4.1 Enumeration Type Documentation

13.4.1.1 ID

```
enum SoundBuffers::ID
```

Enumerator

EnemyDeath	
Explosion	
GunCat	
BombCatMeow	
FreezeCatMeow	

Definition at line 47 of file Resource.hpp.

```
47 { EnemyDeath, Explosion, GunCat, BombCatMeow, FreezeCatMeow };
```

13.5 Textures Namespace Reference

Enumerations

- enum ID {
  PathTile , GrassTile , HouseTile , GunCat ,
  UpgradedGunCat , FreezeCat , UpgradedFzeezeCat , BombCat ,
  UpgradedBombCat , FatRat , FastRat , BasicRat ,
  Bullet , Bomb , Snowflake , PlayButton ,
  Explosion }

13.5.1 Enumeration Type Documentation

13.5.1.1 ID

```
enum Textures::ID
```

Enumerator

PathTile	
GrassTile	
HouseTile	
GunCat	
UpgradedGunCat	
FreezeCat	
UpgradedFzeezeCat	

Enumerator

BombCat	
UpgradedBombCat	
FatRat	
FastRat	
BasicRat	
Bullet	
Bomb	
Snowflake	
PlayButton	
Explosion	

Definition at line 18 of file Resource.hpp.

```
           {
    PathTile,
    GrassTile,
    HouseTile,
    GunCat,
    UpgradedGunCat,
    FreezeCat,
    UpgradedFzeezeCat,
    BombCat,
    UpgradedBombCat,
    FatRat,
    FastRat,
    BasicRat,
    Bullet,
    Bomb,
    Snowflake,
    PlayButton,
    Explosion
 };
```

Chapter 14

Class Documentation

14.1 Block Class Reference

```
#include <Block.hpp>
```

14.1.1 Detailed Description

Definition at line 8 of file Block.hpp.

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

- src/game/Block.hpp

14.2 Button Class Reference

```
#include <Button.hpp>
```

Inheritance diagram for Button:

Collaboration diagram for Button:

Public Member Functions

- Button (const sf::Vector2f &position)
- virtual ∼Button ()
- bool handleInput (const sf::Event &event, World &world)
- void update (World &world, const sf::Vector2f &mousePosition)
- virtual void draw (sf::RenderTarget &target, sf::RenderStates states) const override
- bool hasBeenClicked () const

Protected Member Functions

- bool mouseHovers () const
- virtual void onClick (World &world)

Private Attributes

- sf::CircleShape sprite
- sf::Sprite icon
- bool mouseIn
- bool clicked

14.2.1 Detailed Description

Definition at line 11 of file Button.hpp.

14.2.2 Constructor & Destructor Documentation

14.2.2.1 Button()

```
Button::Button (
              const sf::Vector2f & position )
```

Definition at line 9 of file Button.cpp.

```
9                                           :
10     sprite(),
11     mouseIn(false),
12     clicked(false) {
13   sprite.setPosition(position);
14   sprite.setRadius(buttonSize);
15   sprite.setOrigin(buttonSize, buttonSize);
16   sprite.setFillColor(buttonColor);
17   sprite.setOutlineColor(outlineColor);
18   sprite.setOutlineThickness(outlineThickness);
19   icon.setPosition(position);
20 }
```

14.2.2.2 ∼Button()

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

Definition at line 14 of file Button.hpp.

```
14 {};
```

14.2.3 Member Function Documentation

14.2.3.1 draw()

```
void Button::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override], [virtual]
```

Reimplemented in SelectTowerButton< T >.

Definition at line 35 of file Button.cpp.

```
                                                                     {
  target.draw(sprite, states);
  target.draw(icon, states);
}
```

14.2.3.2 handleInput()

```
bool Button::handleInput (
            const sf::Event & event,
            World & world )
```

Definition at line 22 of file Button.cpp.

```
                                                             {
  if (event.type == sf::Event::MouseButtonPressed
      && event.mouseButton.button == sf::Mouse::Button::Left) {
    onClick(world);
    return true;
  }
  return false;
}
```

14.2.3.3 hasBeenClicked()

```
bool Button::hasBeenClicked ( ) const  [inline]
```

Definition at line 18 of file Button.hpp.

```
{ return clicked; };
```

14.2.3.4 mouseHovers()

```
bool Button::mouseHovers ( ) const  [inline], [protected]
```

Definition at line 20 of file Button.hpp.

```
{ return mouseIn; }
```

14.2.3.5 onClick()

```
void Button::onClick (
            World & world )  [protected], [virtual]
```

Reimplemented in SelectTowerButton< T >.

Definition at line 40 of file Button.cpp.

```
                              {
  clicked = true;
}
```

14.2.3.6 update()

```
void Button::update (
            World & world,
            const sf::Vector2f & mousePosition )
```

Definition at line 31 of file Button.cpp.

```
                                                                   {
  mouseIn = sprite.getGlobalBounds().contains(mousePosition);
}
```

14.2.4 Member Data Documentation

14.2.4.1 clicked

```
bool Button::clicked  [private]
```

Definition at line 26 of file Button.hpp.

14.2.4.2 icon

```
sf::Sprite Button::icon  [private]
```

Definition at line 24 of file Button.hpp.

14.2.4.3 mouseIn

```
bool Button::mouseIn  [private]
```

Definition at line 25 of file Button.hpp.

14.2.4.4 sprite

```
sf::CircleShape Button::sprite  [private]
```

Definition at line 23 of file Button.hpp.

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

- src/ui/Button.hpp
- src/ui/Button.cpp

14.3 Contains Class Reference

14.3.1 Detailed Description

of Scenes

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

- src/game/Scene.hpp

14.4 Enemy Class Reference

A class for ingame enemies. Derived from Entity class.

```
#include <Enemy.hpp>
```

Inheritance diagram for Enemy:

Collaboration diagram for Enemy:

Public Member Functions

- Enemy (TextureHolder &textures, int textureID, std::map< int, std::pair< int, int >> &pathMarkers, int hit↩ Points, float speed)
- void update (sf::Time deltaTime, World &world)
- EnemyPtr clone (TextureHolder &textures, int textureID, std::map< int, std::pair< int, int >> &pathMarkers, int hitPoints, float speed) const

  Copies an enemy.
- void takeDamage (int damageAmount)

  Makes this enemy instance take damage.
- void slowDown ()

  Slows the enemy down (caused by FreezeCat tower).
- TextureHolder & getTextureHolder () const

  returns a reference to the TextureHolder that contains the texture for this enemy.
- std::map< int, std::pair< int, int > > & getPathMarkers () const

  Gets a reference to this Enemy's pathmarkers list.
- int getHitPoints () const

returns hitpoints for this enemy (int)
- int getValue () const

  Get value for this enemy (how much money will killing it give).
- float getSpeed () const

  Get speed for this enemy.
- bool isAtFinish () const
- bool isAlive () const

  Is this enemy alive (hitpoints>0)?
- bool ifShouldRemove () const

  Should this enemy be removed from the enemies list?
- bool isNotFrozen () const

  Is this enemy not frozen (slowed down)?

Static Public Member Functions

- static EnemyPtr make (TextureHolder &textures, int textureID, std::map< int, std::pair< int, int >> &path↩ Markers, int hitPoints, float speed)

  Makes a new enemy. Returns a pointer to it.

Private Attributes

- std::map< int, std::pair< int, int > > & pathMarkers_
- TextureHolder & textures_
- float speed_
- float slowdownFactor_ = 1.0
- int slowdownCounter_
- sf::Time slowDownTime = sf::seconds(.1f)
- sf::Time slowedDownAt_ = sf::seconds(0.f)
- int nextMarker
- int hitPoints_
- int value_
- bool atFinishTile
- bool isFrozen

Additional Inherited Members

14.4.1 Detailed Description

A class for ingame enemies. Derived from Entity class.

Enemies have a set amount of hitpoints and speed, a set texture and a vector of points they follow.

Definition at line 21 of file Enemy.hpp.

14.4.2 Constructor & Destructor Documentation

14.4.2.1 Enemy()

```
Enemy::Enemy (
            TextureHolder & textures,
            int textureID,
            std::map< int, std::pair< int, int >> & pathMarkers,
            int hitPoints,
            float speed )
```

Definition at line 10 of file Enemy.cpp.

```
                          :
    Entity(textureID, textures),
    textures_(textures),
    pathMarkers_(pathMarkers),
    atFinishTile(false),
    hitPoints_(hitPoints),
    value_(hitPoints),
    speed_(speed),
    slowdownCounter_(0),
    isFrozen(false)
  {
  int tileX = pathMarkers_.at(1).first;
  int tileY = pathMarkers_.at(1).second;

  float coordX = tileX * 64.f;
  float coordY = tileY * 64.f;

  entitySprite.setPosition(coordX, coordY);
  nextMarker = 2;
  //std::cout « "x: " « entitySprite.getPosition().x « " y: " « entitySprite.getPosition().y « std::endl;
}
```

14.4.3 Member Function Documentation

14.4.3.1 clone()

```
EnemyPtr Enemy::clone (
            TextureHolder & textures,
            int textureID,
            std::map< int, std::pair< int, int >> & pathMarkers,
            int hitPoints,
            float speed ) const
```

Copies an enemy.

Parameters

textures	A pointer to a TextureHolder class instance.
textureID	Texture ID to find correct texture inside textures
pathMarkers	A list of coordinate points the enemy will follow.
hitPoints	Amount of hitpoints for the enemy as an integer.
speed	Enemy speed as a float value.

Returns

A shared pointer to the copied enemy.

Definition at line 100 of file Enemy.cpp.

```
                                                        {
  return std::make_shared<Enemy> (textures, textureID, pathMarkers, hitPoints, speed);
}
```

14.4.3.2 getHitPoints()

`int Enemy::getHitPoints ( ) const [inline]`

returns hitpoints for this enemy (int)

Returns

Hitpoints as int

Definition at line 76 of file Enemy.hpp.

```
{ return hitPoints_; }
```

14.4.3.3 getPathMarkers()

`std::map<int, std::pair<int, int> >& Enemy::getPathMarkers ( ) const [inline]`

Gets a reference to this Enemy's pathmarkers list.

Returns

Reference to the pathmarkers list.

Definition at line 71 of file Enemy.hpp.

```
{ return pathMarkers_; }
```

14.4.3.4 getSpeed()

`float Enemy::getSpeed ( ) const [inline]`

Get speed for this enemy.

Returns

Speed as float.

Definition at line 86 of file Enemy.hpp.

```
{ return speed_; }
```

14.4.3.5 getTextureHolder()

```
TextureHolder& Enemy::getTextureHolder ( ) const  [inline]
```

returns a reference to the TextureHolder that contains the texture for this enemy.

Returns

Definition at line 66 of file Enemy.hpp.

```
66 { return textures_; }
```

14.4.3.6 getValue()

```
int Enemy::getValue ( ) const  [inline]
```

Get value for this enemy (how much money will killing it give).

Returns

Value as int.

Definition at line 81 of file Enemy.hpp.

```
81 { return value_; }
```

14.4.3.7 ifShouldRemove()

```
bool Enemy::ifShouldRemove ( ) const  [inline]
```

Should this enemy be removed from the enemies list?

Returns

true if enemy should be removed, otherwise false.

Definition at line 101 of file Enemy.hpp.

```
101 { return !isAlive() || atFinishTile; }
```

14.4.3.8 isAlive()

```
bool Enemy::isAlive ( ) const  [inline]
```

Is this enemy alive (hitpoints>0)?

Returns

true if enemy is alive, otherwise false.

Definition at line 96 of file Enemy.hpp.

```
96 { return hitPoints_ > 0; }
```

14.4.3.9 isAtFinish()

```
bool Enemy::isAtFinish ( ) const  [inline]
```

Is this enemy at the finish line (house)?

Returns

true if enemy has reached the house, otherwise false.

Definition at line 91 of file Enemy.hpp.

```
91 { return atFinishTile; }
```

14.4.3.10 isNotFrozen()

```
bool Enemy::isNotFrozen ( ) const  [inline]
```

Is this enemy not frozen (slowed down)?

Returns

false if enemy is frozen, otherwise true.

Definition at line 106 of file Enemy.hpp.

```
106 { return !isFrozen; }
```

14.4.3.11 make()

```
static EnemyPtr Enemy::make (
             TextureHolder & textures,
             int textureID,
             std::map< int, std::pair< int, int >> & pathMarkers,
             int hitPoints,
             float speed )  [inline], [static]
```

Makes a new enemy. Returns a pointer to it.

Parameters

textures	A pointer to a TextureHolder class instance.
textureID	Texture ID to find correct texture inside textures
pathMarkers	A list of coordinate points the enemy will follow.
hitPoints	Amount of hitpoints for the enemy as an integer.
speed	Enemy speed as a float value.

Returns

A shared pointer to the created enemy.

Definition at line 39 of file Enemy.hpp.

```
                                        {
        return std::make_unique<Enemy>(textures, textureID, pathMarkers, hitPoints, speed);
    }
```

14.4.3.12 slowDown()

```
void Enemy::slowDown ( )
```

Slows the enemy down (caused by FreezeCat tower).

Definition at line 31 of file Enemy.cpp.

```
                     {
    slowdownCounter_ = 0;
    slowdownFactor_ = 0.5f;
    isFrozen = true;
}
```

14.4.3.13 takeDamage()

```
void Enemy::takeDamage (
              int damageAmount )  [inline]
```

Makes this enemy instance take damage.

Parameters

damageAmount	Damage to take.

Definition at line 57 of file Enemy.hpp.

```
{ hitPoints_ -= damageAmount; }
```

14.4.3.14 update()

```
void Enemy::update (
            sf::Time deltaTime,
            World & world ) [virtual]
```

Parameters

deltaTime	time since last update in Game loop
world	World class the enemy belongs to

Implements Entity.

Definition at line 36 of file Enemy.cpp.

```
                                               {

  int finishTileX = pathMarkers_.at(pathMarkers_.size()).first;
  int finishTileY = pathMarkers_.at(pathMarkers_.size()).second;
  float finishCoordX = finishTileX * 64.f;
  float finishCoordY = finishTileY * 64.f;

  if (!(abs(finishCoordX - entitySprite.getPosition().x) <= 4.f && abs(finishCoordY -
    entitySprite.getPosition().y) <= 4.f)) {

    //Check if has reached next marker
    int nextTileX = pathMarkers_.at(nextMarker).first;
    int nextTileY = pathMarkers_.at(nextMarker).second;
    float nextCoordX = nextTileX * 64.f;
    float nextCoordY = nextTileY * 64.f;

    if (pathMarkers_.size() != nextMarker) {
      if (abs(nextCoordX - entitySprite.getPosition().x) <= 4.f && abs(nextCoordY -
    entitySprite.getPosition().y) <= 4.f) {
        entitySprite.setPosition(nextCoordX, nextCoordY);
        nextMarker++;
      }
    }

    //Recalculate next marker coordinates in case of nextmarker++
    nextTileX = pathMarkers_.at(nextMarker).first;
    nextTileY = pathMarkers_.at(nextMarker).second;
    nextCoordX = nextTileX * 64.f;
    nextCoordY = nextTileY * 64.f;
    //float movement = round(speed_ * slowdownFactor_);
    float movement = speed_ * slowdownFactor_;

    //Move either along x- or y-axis
    if (entitySprite.getPosition().x == nextCoordX) {
      if (entitySprite.getPosition().y < nextCoordY) {
        entitySprite.move(0.0f, movement);
      } else {
        entitySprite.move(0.0f, -movement);
      }
    } else if (entitySprite.getPosition().y == nextCoordY) {
      if (entitySprite.getPosition().x < nextCoordX) {
        entitySprite.move(movement, 0.0f);
      } else {
        entitySprite.move(-movement, 0.0f);
      }
    }

    // std::cout « "X: " « entitySprite.getPosition().x « ",Y: " « entitySprite.getPosition().y « "\n";
    /*
    std::cout « "getPos x: " « entitySprite.getPosition().x « " y: " « entitySprite.getPosition().y «
    std::ends;
    std::cout « "   nextCr x: " « nextCoordX « " y: " « nextCoordY « std::ends;
    std::cout « "   nextMarker: " « nextMarker « std::ends;
    std::cout « "   finCr x: " « finishCoordX « " y: " « finishCoordY « std::endl;
    */

    slowdownCounter_++;
    // handle slowdown
    if (slowdownFactor_ < 1.0f && slowdownCounter_ >= 100) {
      slowdownFactor_ = 1.0;
      isFrozen = false;
    }
  } else {
    atFinishTile = true;
  }
}
```

14.4.4 Member Data Documentation

14.4.4.1 atFinishTile

```
bool Enemy::atFinishTile  [private]
```

Definition at line 119 of file Enemy.hpp.

14.4.4.2 hitPoints_

```
int Enemy::hitPoints_  [private]
```

Definition at line 117 of file Enemy.hpp.

14.4.4.3 isFrozen

```
bool Enemy::isFrozen  [private]
```

Definition at line 120 of file Enemy.hpp.

14.4.4.4 nextMarker

```
int Enemy::nextMarker  [private]
```

Definition at line 116 of file Enemy.hpp.

14.4.4.5 pathMarkers_

```
std::map<int, std::pair<int, int> >& Enemy::pathMarkers_  [private]
```

Definition at line 109 of file Enemy.hpp.

14.4.4.6 slowdownCounter_

```
int Enemy::slowdownCounter_  [private]
```

Definition at line 113 of file Enemy.hpp.

14.4.4.7 slowdownFactor_

```
float Enemy::slowdownFactor_ = 1.0  [private]
```

Definition at line 112 of file Enemy.hpp.

14.4.4.8 slowDownTime

```
sf::Time Enemy::slowDownTime = sf::seconds(.1f)  [private]
```

Definition at line 114 of file Enemy.hpp.

14.4.4.9 slowedDownAt_

```
sf::Time Enemy::slowedDownAt_ = sf::seconds(0.f)  [private]
```

Definition at line 115 of file Enemy.hpp.

14.4.4.10 speed_

```
float Enemy::speed_  [private]
```

Definition at line 111 of file Enemy.hpp.

14.4.4.11 textures_

```
TextureHolder& Enemy::textures_  [private]
```

Definition at line 110 of file Enemy.hpp.

14.4.4.12 value_

```
int Enemy::value_  [private]
```

Definition at line 118 of file Enemy.hpp.

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

- src/entity/Enemy.hpp
- src/entity/Enemy.cpp

14.5 Entity Class Reference

Visible entity on the map.

```
#include <Entity.hpp>
```

Inheritance diagram for Entity:

Collaboration diagram for Entity:

Public Member Functions

- Entity (int textureID, const TextureHolder &textures)
- void draw (sf::RenderTarget &target, sf::RenderStates states) const

  draw this entity, structure according to SFML manual
- virtual void update (sf::Time deltaTime, World &world)=0

  Virtual function to update this Entity, subclasses will implement.
- sf::Vector2f getPosition () const

  Get sprite position.
- int getTextureID () const

  Get textureID used for this Entity.

Protected Attributes

- sf::Sprite entitySprite

Private Attributes

- sf::Vector2f mVelocity
- int textureID_

14.5.1 Detailed Description

Visible entity on the map.

Definition at line 22 of file Entity.hpp.

14.5.2 Constructor & Destructor Documentation

14.5.2.1 Entity()

```
Entity::Entity (
                int textureID,
                const TextureHolder & textures )
```

Parameters

textureID	Texture number to be used for the Entity
textures	Reference to a TextureHolder class instance that holds the texture

Definition at line 8 of file Entity.cpp.

```
8                                                                : entitySprite() {
9    textureID_ = textureID;
10    entitySprite.setTexture(textures.get(Textures::ID(textureID_)));
11 }
```

14.5.3 Member Function Documentation

14.5.3.1 draw()

```
void Entity::draw (
             sf::RenderTarget & target,
             sf::RenderStates states ) const
```

draw this entity, structure according to SFML manual

Parameters

target	where to draw this Entity
states	RenderStates class to use for this drawable

Definition at line 13 of file Entity.cpp.

```
13                                                                  {
14    target.draw(entitySprite, states);
15 }
```

14.5.3.2 getPosition()

```
sf::Vector2f Entity::getPosition ( ) const  [inline]
```

Get sprite position.

Returns

SFML 2D vector, the position

Definition at line 46 of file Entity.hpp.

```
46 { return entitySprite.getPosition(); }
```

14.5.3.3 getTextureID()

```
int Entity::getTextureID ( ) const  [inline]
```

Get textureID used for this Entity.

Returns

integer, the textureID (index in TextureHolder list)

Definition at line 51 of file Entity.hpp.

```
51 { return textureID_; }
```

14.5.3.4 update()

```
virtual void Entity::update (
            sf::Time deltaTime,
            World & world )  [pure virtual]
```

Virtual function to update this Entity, subclasses will implement.

Parameters

deltaTime	Time since last frame
world	Reference to a world class where to update

Implemented in Tower, Projectile, and Enemy.

14.5.4 Member Data Documentation

14.5.4.1 entitySprite

```
sf::Sprite Entity::entitySprite  [protected]
```

Definition at line 54 of file Entity.hpp.

14.5.4.2 mVelocity

```
sf::Vector2f Entity::mVelocity  [private]
```

Definition at line 57 of file Entity.hpp.

14.5.4.3 textureID_

```
int Entity::textureID_  [private]
```

Definition at line 58 of file Entity.hpp.

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

- src/entity/Entity.hpp
- src/entity/Entity.cpp

14.6 Game Class Reference

Class where the game, setting are set and loading and rendering is called from.

```
#include <Game.hpp>
```

Collaboration diagram for Game:

Public Member Functions

- Game ()
- void load ()

    load all resources to Resource class instances
- void render ()

    call's current Scene class instance's draw function to display everything on screen
- void processEvents ()

    takes player input from window and forwards is to current Scene class
- void handleSceneChange ()

    Changes the current Scene class instance (for example when loading a level)
- void Update (sf::Time deltaTime)

    Updates everything on screen that should move etc.
- void run ()

    Run the game.

Public Attributes

- TextureHolder textures
- FontHolder fonts
- SoundBufferHolder sounds

Private Attributes

- sf::VideoMode videomode
- sf::RenderWindow window
- std::unique_ptr< Scene > scene
- sf::Clock clock
- sf::RenderStates renderStates

14.6.1 Detailed Description

Class where the game, setting are set and loading and rendering is called from.

Definition at line 13 of file Game.hpp.

14.6.2 Constructor & Destructor Documentation

14.6.2.1 Game()

```
Game::Game ( )
```

Definition at line 16 of file Game.cpp.

```
              : videomode(1280, 1024),
                  window(videomode, "Cats vs Rats TD"),
                  textures(),
                  renderStates(sf::RenderStates::Default) {
     load();
     scene = std::make_unique<GameTitle>(textures, fonts);
     //scene = std::make_unique<GameEnd>(textures, fonts);
 }
```

14.6.3 Member Function Documentation

14.6.3.1 handleSceneChange()

```
void Game::handleSceneChange ( )
```

Changes the current Scene class instance (for example when loading a level)

Definition at line 98 of file Game.cpp.

```
                                {
     if (scene->requestedScene().scene != scene->sceneType()) {
        //std::cout « "Requested: " « scene->requestedScene().scene « " ,sceneType: " «scene->sceneType «
       "\n";
        auto request = scene->requestedScene();
        switch (request.scene) {
          case Scenes::ID::GameTitle:scene = std::make_unique<GameTitle>(textures, fonts);
            break;
          case Scenes::ID::MapScene:scene = std::make_unique<MapScene>(textures, fonts, sounds,
       request.number);
            break;
          case Scenes::ID::LevelSelect:scene = std::make_unique<LevelSelect>(textures, fonts);
            break;
          case Scenes::ID::GameEnd:scene = std::make_unique<GameEnd>(textures, fonts);
            break;
          default:scene = std::make_unique<GameTitle>(textures, fonts);
            break;
        }
     }
   }
```

14.6.3.2 load()

```
void Game::load ( )
```

load all resources to Resource class instances

Definition at line 25 of file Game.cpp.

```
                {
  //load textures and sounds at start of game, return false if failed
  //set renderStates
  renderStates.transform.scale(1, 1);
  textures.load(Textures::PathTile, "src/media/textures/pathtile.png");
  textures.load(Textures::GrassTile, "src/media/textures/grasstile.png");
  textures.load(Textures::HouseTile, "src/media/textures/house.png");
  textures.load(Textures::FatRat, "src/media/textures/fatrat.png");
  textures.load(Textures::BasicRat, "src/media/textures/basicrat.png");
  textures.load(Textures::FastRat, "src/media/textures/fastrat.png");
  textures.load(Textures::GunCat, "src/media/textures/guncat.png");
  textures.load(Textures::UpgradedGunCat, "src/media/textures/level2guncat.png");
  textures.load(Textures::FreezeCat, "src/media/textures/freezecat.png");
  textures.load(Textures::UpgradedFzeezeCat, "src/media/textures/level2freezecat.png");
  textures.load(Textures::BombCat, "src/media/textures/bombcat.png");
  textures.load(Textures::UpgradedBombCat, "src/media/textures/level2bombcat.png");
  textures.load(Textures::Bullet, "src/media/textures/bullet.png");
  textures.load(Textures::Snowflake, "src/media/textures/snowflake.png");
  textures.load(Textures::Bomb, "src/media/textures/bomb.png");
  textures.load(Textures::Explosion, "src/media/textures/explosion_128.png");
  textures.load(Textures::PlayButton, "src/media/textures/playbutton.png");
  //fonts.load(Fonts::GameTitleFont, "src/media/fonts/SnackerComic_PerosnalUseOnly.ttf");
  fonts.load(Fonts::GameTitleFont, "src/media/fonts/BalonkuRegular-la1w.otf");
  sounds.load(SoundBuffers::EnemyDeath, "src/media/sounds/44429_468340-lq.wav");
  sounds.load(SoundBuffers::Explosion, "src/media/sounds/explosion_sound.mp3");
  sounds.load(SoundBuffers::GunCat, "src/media/sounds/gun_sound.mp3");
  sounds.load(SoundBuffers::BombCatMeow, "src/media/sounds/sadmeow_speedup.mp3");
  sounds.load(SoundBuffers::FreezeCatMeow, "src/media/sounds/freezecat.mp3");
}
```

14.6.3.3 processEvents()

```
void Game::processEvents ( )
```

takes player input from window and forwards is to current Scene class

Definition at line 65 of file Game.cpp.

```
                         {
  //handle input from player
  sf::Event event;
  while (window.pollEvent(event)) {
    scene->handleInput(event);
    if (event.type == sf::Event::Closed)
      window.close();
  }
}
```

14.6.3.4 render()

```
void Game::render ( )
```

call's current Scene class instance's draw function to display everything on screen

Definition at line 55 of file Game.cpp.

```
                  {
  //render everything to the screen
  //sf::CircleShape shape(50);
  //shape.setFillColor(sf::Color(150, 50, 250));
  //window.draw(shape);
  window.clear(sf::Color(250, 250, 250)); //????????????????
  window.draw(*scene, renderStates);
  window.display();
}
```

14.6.3.5 run()

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

Run the game.

Definition at line 80 of file Game.cpp.

```
                {
  // Counts time between frames
  sf::Time TimePerFrame = sf::seconds(0.0167); // 1/60=0.0167
  sf::Time deltaTime = sf::Time::Zero;
  // Initialize time to zero
  while (window.isOpen()) {
    handleSceneChange();
    processEvents();
    deltaTime += clock.restart();
    while (deltaTime > TimePerFrame) {
      deltaTime -= TimePerFrame;
      processEvents();
      Update(deltaTime);
    }
    window.clear(sf::Color::White);
    render();
  }
}
```

14.6.3.6 Update()

```
void Game::Update (
              sf::Time deltaTime )
```

Updates everything on screen that should move etc.

Parameters

deltaTime	time since last frame

Definition at line 75 of file Game.cpp.

```
                                  {
  //update elements of the game, positions etc.
  scene->update(deltaTime);
}
```

14.6.4 Member Data Documentation

14.6.4.1 clock

```
sf::Clock Game::clock  [private]
```

Definition at line 57 of file Game.hpp.

14.6.4.2 fonts

```
FontHolder Game::fonts
```

Instance of class Resource, which stores fonts

Definition at line 48 of file Game.hpp.

14.6.4.3 renderStates

```
sf::RenderStates Game::renderStates  [private]
```

Definition at line 58 of file Game.hpp.

14.6.4.4 scene

```
std::unique_ptr<Scene> Game::scene  [private]
```

Definition at line 56 of file Game.hpp.

14.6.4.5 sounds

```
SoundBufferHolder Game::sounds
```

Instance of class Resource, which stores sounds

Definition at line 52 of file Game.hpp.

14.6.4.6 textures

```
TextureHolder Game::textures
```

Instance of class Resource, which stores textures

Definition at line 44 of file Game.hpp.

14.6.4.7 videomode

```
sf::VideoMode Game::videomode  [private]
```

Definition at line 54 of file Game.hpp.

14.6.4.8 window

```
sf::RenderWindow Game::window  [private]
```

Definition at line 55 of file Game.hpp.

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

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

14.7 GameCommandsMenu Class Reference

Shows the player how to play the game (at the right side of the screen in a MapScene)

```
#include <GameCommandsMenu.hpp>
```

Inheritance diagram for GameCommandsMenu:

Collaboration diagram for GameCommandsMenu:

Public Member Functions

- GameCommandsMenu (const sf::Vector2f &textPosition, const sf::Vector2f &towerPosition, FontHolder &fonts, World &world)

    Constructor.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

Private Attributes

- sf::Text menuString_
- sf::Text towerIndexText_
- std::vector< sf::Sprite > towerSprites_

14.7.1 Detailed Description

Shows the player how to play the game (at the right side of the screen in a MapScene)

Definition at line 17 of file GameCommandsMenu.hpp.

14.7.2 Constructor & Destructor Documentation

14.7.2.1 GameCommandsMenu()

```
GameCommandsMenu::GameCommandsMenu (
            const sf::Vector2f & textPosition,
            const sf::Vector2f & towerPosition,
            FontHolder & fonts,
            World & world )
```

Constructor.

Parameters

textPosition	Coordinates for the guide text
towerPosition	Coordinates for the Tower sprites position
fonts	Reference to a FontHolder to get fonts for text
world	Reference to the current Game World

Definition at line 7 of file GameCommandsMenu.cpp.

```
                                                     {
    menuString_.setFont(fonts.get(Fonts::GameTitleFont));
    menuString_.setString("Controls:\n"
                          "Left Click:\n"
                          "   Select location\n"
                          "Right Click:\n"
                          "   sell\n"
                          "Number 1, 2, 3:\n"
                          "   Buy tower type\n"
                          "Number 4:\n"
                          "   Upgrade tower\n"
                          "    (Price:500)");
    menuString_.setCharacterSize(24);
    menuString_.setFillColor(sf::Color::Black);
    menuString_.setPosition(textPosition);
    menuString_.setLineSpacing(0.8f);

    towerIndexText_.setFont(fonts.get(Fonts::GameTitleFont));
    towerIndexText_.setString("1             GunCat\n"
                              "              Price:300\n\n"
                              "2             IceCat\n"
                              "              Price:400\n\n"
                              "3             BombCat\n"
                              "              Price:500");
    towerIndexText_.setCharacterSize(24);
    towerIndexText_.setFillColor(sf::Color::Black);
    towerIndexText_.setPosition(towerPosition);
    towerIndexText_.move(-10.f, 2.f);
    towerIndexText_.setLineSpacing(0.8f);
    // load and set textures
    sf::Sprite tower1;
    tower1.setTexture(world.getTextures().get(Textures::GunCat));
    sf::Sprite tower2;
    tower2.setTexture(world.getTextures().get(Textures::FreezeCat));
    sf::Sprite tower3;
    tower3.setTexture(world.getTextures().get(Textures::BombCat));

    towerSprites_.push_back(tower1);
    towerSprites_.push_back(tower2);
    towerSprites_.push_back(tower3);
    int moveAmount = 0;

    for (auto &tower : towerSprites_) {
      tower.setPosition(towerPosition);
      tower.move(0, tileSize * float(moveAmount));
      moveAmount += 1;
    }

}
```

14.7.3 Member Function Documentation

14.7.3.1 draw()

```
void GameCommandsMenu::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

Definition at line 59 of file GameCommandsMenu.cpp.

```
                                                                          {
  target.draw(menuString_);
  for (auto tower : towerSprites_) {
    target.draw(tower);
  }
  target.draw(towerIndexText_);
}
```

14.7.4 Member Data Documentation

14.7.4.1 menuString_

```
sf::Text GameCommandsMenu::menuString_  [private]
```

Definition at line 32 of file GameCommandsMenu.hpp.

14.7.4.2 towerIndexText_

```
sf::Text GameCommandsMenu::towerIndexText_  [private]
```

Definition at line 33 of file GameCommandsMenu.hpp.

14.7.4.3 towerSprites_

```
std::vector<sf::Sprite> GameCommandsMenu::towerSprites_  [private]
```

Definition at line 34 of file GameCommandsMenu.hpp.

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

- src/ui/GameCommandsMenu.hpp
- src/ui/GameCommandsMenu.cpp

14.8 GameEnd Class Reference

A class that inherits Scene, is shown when the player loses the game.

```
#include <GameEnd.hpp>
```

Inheritance diagram for GameEnd:

Collaboration diagram for GameEnd:

Public Member Functions

- GameEnd (TextureHolder &textures, FontHolder &fonts)

    Constructor for the class.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

    draw the scene to target
- void handleInput (const sf::Event &event) override

    handle input for current scene
- void handlePlayerInput (sf::Keyboard::Key key, bool isPressed)

    Takes input from handleInput function.
- sceneRequest requestedScene () override

    Next scene from this one.
- void update (sf::Time dt) override

    Update scene function.
- Scenes::ID sceneType () override

    Return current scene's type.

Public Attributes

- std::vector< sf::Text > options

    Options to choose from in this scene.
- std::size_t optionIndex

    What option is currently selected.

Private Attributes

- int width = 16
- int height = 16
- TextureHolder & textures_
- FontHolder & fonts_
- sf::Text textGameOver_
- sf::Text textInstruction_
- sceneRequest nextScene_

14.8.1 Detailed Description

A class that inherits Scene, is shown when the player loses the game.

Definition at line 16 of file GameEnd.hpp.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 GameEnd()

```
GameEnd::GameEnd (
            TextureHolder & textures,
            FontHolder & fonts )  [explicit]
```

Constructor for the class.

Parameters

textures	Reference to a TextureHolder instance
fonts	Reference to a FontHolder instance

Definition at line 8 of file GameEnd.cpp.

```
    : textures_(textures), fonts_(fonts) {
  // help text
  textGameOver_.setFont(fonts_.get(Fonts::GameTitleFont));
  textGameOver_.setString("Game Over!");
  textGameOver_.setCharacterSize(50);
  textGameOver_.setFillColor(sf::Color::Black);
  textGameOver_.setPosition(tileSize * float(width) / 3.f, tileSize * float(height) / 4.1f);
  textInstruction_.setFont(fonts_.get(Fonts::GameTitleFont));
  textInstruction_.setString("press Enter to go back to main menu");
  textInstruction_.setCharacterSize(24);
  textInstruction_.setFillColor(sf::Color::Black);
  textInstruction_.setPosition(tileSize * float(width) / 3.f, tileSize * float(height) / 3.4f);

  optionIndex = 0;
  //option for enter and going back to main menu
  sf::Text backToMenuOption;
  backToMenuOption.setFont(fonts_.get(Fonts::GameTitleFont));
  backToMenuOption.setString("Back To Menu");
  backToMenuOption.setPosition(tileSize * float(width) / 2.f, tileSize * float(height) / 3.f);
  backToMenuOption.setFillColor(sf::Color::Red); // optionIndex = 0;
  backToMenuOption.setCharacterSize(60);
  options.push_back(backToMenuOption);

  // nextScene struct for requestedScene
  nextScene_.scene = Scenes::ID::GameEnd;
  nextScene_.number = 1;
}
```

14.8.3 Member Function Documentation

14.8.3.1 draw()

```
void GameEnd::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

draw the scene to target

Parameters

target	sf::RenderTarget to draw the scene to
states	sf::RenderStates object for drawing

Definition at line 39 of file GameEnd.cpp.

```
                                                                        {
  for (const auto &text : options) {
    target.draw(text);
  }
  target.draw(textGameOver_);
  target.draw(textInstruction_);
}
```

14.8.3.2 handleInput()

```
void GameEnd::handleInput (
              const sf::Event & event )  [override], [virtual]
```

handle input for current scene

Parameters

event	sf::Event (keypress etc.)

Implements Scene.

Definition at line 46 of file GameEnd.cpp.

```
                                                 {
  switch (event.type) {
    case sf::Event::KeyPressed:handlePlayerInput(event.key.code, true);
      break;
    case sf::Event::KeyReleased:handlePlayerInput(event.key.code, false);
      break;
    default:break;
  }
}
```

14.8.3.3 handlePlayerInput()

```
void GameEnd::handlePlayerInput (
              sf::Keyboard::Key key,
              bool isPressed )
```

Takes input from handleInput function.

Parameters

key	keyboard key that was pressed
isPressed	Is the key pressed currently?

Definition at line 57 of file GameEnd.cpp.

```
                                                                 {
  if (isPressed) {
    if (key == sf::Keyboard::Enter) {
      nextScene_.scene = Scenes::ID::LevelSelect;
      nextScene_.number = optionIndex + 1;
    }
  }
}
```

14.8.3.4 requestedScene()

`sceneRequest GameEnd::requestedScene ( )  [override], [virtual]`

Next scene from this one.

Returns

the request

Implements Scene.

Definition at line 36 of file GameEnd.cpp.

```
                                      {
  return nextScene_;
}
```

14.8.3.5 sceneType()

`Scenes::ID GameEnd::sceneType ( )  [inline], [override], [virtual]`

Return current scene's type.

Returns

current scene's type, GameEnd

Implements Scene.

Definition at line 55 of file GameEnd.hpp.

```
{ return Scenes::ID::GameEnd; }
```

14.8.3.6 update()

```
void GameEnd::update (
              sf::Time dt )  [override], [virtual]
```

Update scene function.

Parameters

dt	deltatime, time since last frame update

Implements Scene.

Definition at line 55 of file GameEnd.cpp.

```
55 {}
```

14.8.4 Member Data Documentation

14.8.4.1 fonts_

```
FontHolder& GameEnd::fonts_  [private]
```

Definition at line 68 of file GameEnd.hpp.

14.8.4.2 height

```
int GameEnd::height = 16  [private]
```

Definition at line 66 of file GameEnd.hpp.

14.8.4.3 nextScene_

```
sceneRequest GameEnd::nextScene_  [private]
```

Definition at line 71 of file GameEnd.hpp.

14.8.4.4 optionIndex

```
std::size_t GameEnd::optionIndex
```

What option is currently selected.

Definition at line 63 of file GameEnd.hpp.

14.8.4.5 options

```
std::vector<sf::Text> GameEnd::options
```

Options to choose from in this scene.

Definition at line 59 of file GameEnd.hpp.

14.8.4.6 textGameOver_

```
sf::Text GameEnd::textGameOver_  [private]
```

Definition at line 69 of file GameEnd.hpp.

14.8.4.7 textInstruction_

```
sf::Text GameEnd::textInstruction_  [private]
```

Definition at line 70 of file GameEnd.hpp.

14.8.4.8 textures_

```
TextureHolder& GameEnd::textures_  [private]
```

Definition at line 67 of file GameEnd.hpp.

14.8.4.9 width

```
int GameEnd::width = 16  [private]
```

Definition at line 65 of file GameEnd.hpp.

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

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

14.9 GameStatusMenu Class Reference

Shows the player information from the game world: hp, money, enemies left and wave number.

`#include <GameStatusMenu.hpp>`

Inheritance diagram for GameStatusMenu:

Collaboration diagram for GameStatusMenu:

Public Member Functions

- GameStatusMenu (const sf::Vector2f &position, FontHolder &fonts)
- void update (World &world, int enemiesNotSpawned, int waveNumber)

  Update The status menu numbers.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  draw the status menu to screen

Private Attributes

- sf::Text menuString_

14.9.1 Detailed Description

Shows the player information from the game world: hp, money, enemies left and wave number.

Definition at line 13 of file GameStatusMenu.hpp.

14.9.2 Constructor & Destructor Documentation

14.9.2.1 GameStatusMenu()

```
GameStatusMenu::GameStatusMenu (
            const sf::Vector2f & position,
            FontHolder & fonts )  [explicit]
```

Definition at line 7 of file GameStatusMenu.cpp.

```
                                                                    {
  menuString_.setFont(fonts.get(Fonts::GameTitleFont));
  menuString_.setString("GameStatusMenu");
   menuString_.setCharacterSize(24);
   menuString_.setFillColor(sf::Color::Black);
   menuString_.setPosition(position);
 }
```

14.9.3 Member Function Documentation

14.9.3.1 draw()

```
void GameStatusMenu::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

draw the status menu to screen

Parameters

target	sf::RenderTarget to draw the statusmenu to
states	sf::RenderStates object for drawing

Definition at line 23 of file GameStatusMenu.cpp.

```
{
  target.draw(menuString_);
}
```

14.9.3.2 update()

```
void GameStatusMenu::update (
            World & world,
            int enemiesNotSpawned,
            int waveNumber )
```

Update The status menu numbers.

Parameters

world	
enemiesNotSpawned	enemies left in current wave that haven't spawned yet
waveNumber	current wave number

Definition at line 14 of file GameStatusMenu.cpp.

```
{
  std::stringstream ss;
  ss « "Wave Number: " « waveNumber « "\n"
     « "Enemies Left: " « (world.getEnemies().size() + enemiesNotSpawned) « "\n"
     « "Hit Points: " « world.getHP() « "\n"
     « "Money: " « world.getMoney() « "\n";
  menuString_.setString(ss.str());

}
```

14.9.4 Member Data Documentation

14.9.4.1 menuString_

```
sf::Text GameStatusMenu::menuString_ [private]
```

Definition at line 30 of file GameStatusMenu.hpp.

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

- src/ui/GameStatusMenu.hpp
- src/ui/GameStatusMenu.cpp

14.10 GameTitle Class Reference

Welcome screen for the game.

```
#include <GameTitle.hpp>
```

Inheritance diagram for GameTitle:

Collaboration diagram for GameTitle:

Public Member Functions

- GameTitle (TextureHolder &textures, FontHolder &fonts)

  Constructor for the GameTitle.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  draw the scene to target
- void handleInput (const sf::Event &event) override

  handle input for current scene
- sceneRequest requestedScene () override

  Next scene from this one.
- void update (sf::Time dt) override

  Update scene function.
- Scenes::ID sceneType () override

  Return current scene's type.

Private Attributes

- int width = 20
- int height = 16
- TextureHolder & textures_
- FontHolder & fonts_
- sf::Time textEffectTime_
- sf::Text text_
- sf::Text text2_
- bool showText_
- sf::Sprite grassSprite_
- sf::Sprite pathSprite
- sf::RenderTexture mapTex_
- sf::Sprite mapSprite_
- sceneRequest nextScene_

Additional Inherited Members

14.10.1 Detailed Description

Welcome screen for the game.

Definition at line 14 of file GameTitle.hpp.

14.10.2 Constructor & Destructor Documentation

14.10.2.1 GameTitle()

```
GameTitle::GameTitle (
            TextureHolder & textures,
            FontHolder & fonts )  [explicit]
```

Constructor for the GameTitle.

Parameters

textures	Reference to a Resource class holding textures
fonts	Reference to a Resource class holding fonts

Definition at line 10 of file GameTitle.cpp.

```
                                                                     : textures_(textures), fonts_(fonts) {
    showText_ = true;
    // load text resources
    text_.setFont(fonts_.get(Fonts::GameTitleFont));
    //textGameOver_.setFont(font);
    text_.setString("                  Cats vs Rats TD\n"
                    "by: Antti, Henrik, Kasperi and Otso");
    text_.setCharacterSize(43);
    text_.setFillColor(sf::Color::Black);
    text_.setPosition(tileSize * float(width) / 6.f, tileSize * float(height) / 3.f);
    text2_.setFont(fonts_.get(Fonts::GameTitleFont));
    //textGameOver_.setFont(font);
    text2_.setString("Press Enter");
    text2_.setCharacterSize(43);
    text2_.setFillColor(sf::Color::Red);
    text2_.setPosition(tileSize * float(width) / 2.8f, tileSize * float(height) / 2.f);
    // load texture resources
    grassSprite_.setTexture(textures_.get(Textures::GrassTile));

    // create holder for map sprite that consists of multiple
    // small tiles
    mapTex_.clear();
    mapTex_.create(1280, 1024);
    for (int i = 0; i < height; ++i) {
      for (int j = 0; j < width; ++j) {
        grassSprite_.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
        mapTex_.draw(grassSprite_);
      }
    }
    mapSprite_.setTexture(mapTex_.getTexture());
    mapSprite_.setOrigin({0, mapSprite_.getLocalBounds().height});
    mapSprite_.setScale({1, -1});
    mapSprite_.setColor(sf::Color(255, 255, 255, 64));

    //scene request
    nextScene_.scene = Scenes::ID::GameTitle;
    nextScene_.number = 1;
}
```

14.10.3 Member Function Documentation

14.10.3.1 draw()

```
void GameTitle::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

draw the scene to target

Parameters

target	sf::RenderTarget to draw the scene to
states	sf::RenderStates object for drawing

Definition at line 49 of file GameTitle.cpp.

```
                                                                          {
  // draw background
  target.draw(mapSprite_);
  //if (showText_) {
  //  target.draw(text_);
  //}
  target.draw(text_);
  target.draw(text2_);

}
```

14.10.3.2 handleInput()

```
void GameTitle::handleInput (
              const sf::Event & event )  [override], [virtual]
```

handle input for current scene

Parameters

event	sf::Event (keypress etc.)

Implements Scene.

Definition at line 59 of file GameTitle.cpp.

```
                                                {
  if (event.type == sf::Event::EventType::KeyPressed) {
    if (event.key.code == sf::Keyboard::Enter) {
      nextScene_.scene = Scenes::ID::LevelSelect;
    }
  }
  //std::cout « "input";
  //requestSceneChange(SceneChangeRequest { nextScene_ });
}
```

14.10.3.3 requestedScene()

```
sceneRequest GameTitle::requestedScene ( )  [override], [virtual]
```

Next scene from this one.

Returns

the request

Implements Scene.

Definition at line 76 of file GameTitle.cpp.

```
                                       {
  return nextScene_;
}
```

14.10.3.4 sceneType()

```
Scenes::ID GameTitle::sceneType ( )  [inline], [override], [virtual]
```

Return current scene's type.

Returns

current scene's type, GameTitle

Implements Scene.

Definition at line 47 of file GameTitle.hpp.

```
{ return Scenes::ID::GameTitle; }
```

14.10.3.5 update()

```
void GameTitle::update (
            sf::Time dt )  [override], [virtual]
```

Update scene function.

Parameters

dt	deltatime, time since last frame update

Implements Scene.

Definition at line 68 of file GameTitle.cpp.

```
                                 {
  textEffectTime_ += dt;
  if (textEffectTime_ >= sf::seconds(0.1)) {
    showText_ = !showText_;
    text_.setFillColor(showText_ ? sf::Color::Black : sf::Color::Red);
    textEffectTime_ = sf::Time::Zero;
  }
}
```

14.10.4 Member Data Documentation

14.10.4.1 fonts_

```
FontHolder& GameTitle::fonts_  [private]
```

Definition at line 52 of file GameTitle.hpp.

14.10.4.2 grassSprite_

```
sf::Sprite GameTitle::grassSprite_  [private]
```

Definition at line 57 of file GameTitle.hpp.

14.10.4.3 height

```
int GameTitle::height = 16  [private]
```

Definition at line 50 of file GameTitle.hpp.

14.10.4.4 mapSprite_

```
sf::Sprite GameTitle::mapSprite_  [private]
```

Definition at line 60 of file GameTitle.hpp.

14.10.4.5 mapTex_

```
sf::RenderTexture GameTitle::mapTex_  [private]
```

Definition at line 59 of file GameTitle.hpp.

14.10.4.6 nextScene_

```
sceneRequest GameTitle::nextScene_  [private]
```

Definition at line 61 of file GameTitle.hpp.

14.10.4.7 pathSprite

```
sf::Sprite GameTitle::pathSprite  [private]
```

Definition at line 58 of file GameTitle.hpp.

14.10.4.8 showText_

```
bool GameTitle::showText_  [private]
```

Definition at line 56 of file GameTitle.hpp.

14.10.4.9 text2_

```
sf::Text GameTitle::text2_  [private]
```

Definition at line 55 of file GameTitle.hpp.

14.10.4.10 text_

```
sf::Text GameTitle::text_  [private]
```

Definition at line 54 of file GameTitle.hpp.

14.10.4.11 textEffectTime_

```
sf::Time GameTitle::textEffectTime_  [private]
```

Definition at line 53 of file GameTitle.hpp.

14.10.4.12 textures_

```
TextureHolder& GameTitle::textures_  [private]
```

Definition at line 51 of file GameTitle.hpp.

14.10.4.13 width

```
int GameTitle::width = 20  [private]
```

Definition at line 49 of file GameTitle.hpp.

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

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

14.11 Grid Class Reference

Grid is used to place the towers according to the visual tiles.

```
#include <Grid.hpp>
```

Inheritance diagram for Grid:

Collaboration diagram for Grid:

Public Member Functions

- Grid ()
- virtual void draw (sf::RenderTarget &target, sf::RenderStates states) const override
- Sector $*$ getSelectedSector ()

    Get selected Sector. Used in TowerMenu class to change where to place a tower.

Private Attributes

- int width
- int height
- Sector $*$ selectedSector
- std::vector$<$ std::vector$<$ int $>$ $>$ blocks

14.11.1 Detailed Description

Grid is used to place the towers according to the visual tiles.

Definition at line 10 of file Grid.hpp.

14.11.2 Constructor & Destructor Documentation

14.11.2.1 Grid()

```
Grid::Grid ( )
```

Definition at line 5 of file Grid.cpp.

```
            :
     width(16),
     height(16),
     blocks() {
 };
```

14.11.3 Member Function Documentation

14.11.3.1 draw()

```
void Grid::draw (
             sf::RenderTarget & target,
             sf::RenderStates states ) const  [override], [virtual]
```

Definition at line 11 of file Grid.cpp.

```
                                                              {
}
```

14.11.3.2 getSelectedSector()

```
Sector * Grid::getSelectedSector ( )
```

Get selected Sector. Used in TowerMenu class to change where to place a tower.

Returns

Definition at line 13 of file Grid.cpp.

```
                               {
  return selectedSector;
}
```

14.11.4 Member Data Documentation

14.11.4.1 blocks

```
std::vector<std::vector<int> > Grid::blocks  [private]
```

Definition at line 23 of file Grid.hpp.

14.11.4.2 height

```
int Grid::height  [private]
```

Definition at line 21 of file Grid.hpp.

14.11.4.3 selectedSector

```
Sector* Grid::selectedSector [private]
```

Definition at line 22 of file Grid.hpp.

14.11.4.4 width

```
int Grid::width [private]
```

Definition at line 20 of file Grid.hpp.

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

- src/ui/Grid.hpp
- src/ui/Grid.cpp

14.12 LevelSelect Class Reference

A class for level selection menu, inherits Scene class.

```
#include <LevelSelect.hpp>
```

Inheritance diagram for LevelSelect:

Collaboration diagram for LevelSelect:

Public Member Functions

- LevelSelect (TextureHolder &textures, FontHolder &fonts)

  Constructor for the class.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  draw the scene to target
- void handleInput (const sf::Event &event) override

  handle input for current scene
- void handlePlayerInput (sf::Keyboard::Key key, bool isPressed)

  Takes input from handleInput function.
- sceneRequest requestedScene () override

  Next scene from this one.
- void update (sf::Time dt) override

  Update scene function.
- void updateOptionText ()

  Update, which option is selected (and so should be colored red)
- Scenes::ID sceneType () override

  Return current scene's type.

Public Attributes

- std::vector< sf::Text > options

  Options to choose from in this scene, so maps.
- std::size_t optionIndex

  What option is currently selected.

Private Attributes

- int width = 20
- int height = 16
- TextureHolder & textures_
- FontHolder & fonts_
- sf::Time textEffectTime_
- sf::Text text_
- bool showText_
- sf::Sprite grassSprite_
- sf::Sprite pathSprite
- sf::RenderTexture mapTex_
- sf::Sprite mapSprite_
- sceneRequest nextScene_

14.12.1 Detailed Description

A class for level selection menu, inherits Scene class.

Definition at line 18 of file LevelSelect.hpp.

14.12.2 Constructor & Destructor Documentation

14.12.2.1 LevelSelect()

```
LevelSelect::LevelSelect (
               TextureHolder & textures,
               FontHolder & fonts )   [explicit]
```

Constructor for the class.

Parameters

textures	Reference to a TextureHolder instance
fonts	Reference to a FontHolder instance

Definition at line 7 of file LevelSelect.cpp.

```
      : textures_(textures), fonts_(fonts) {
    std::vector<std::string> maps;
    for (int i = 0; i < 5; i++) {
```

```
    std::string mapfile = "src/media/maps/map";
    mapfile.append(std::to_string(i + 1));
    mapfile.append(".txt");
    std::ifstream infile(mapfile);
    std::string line; //line to store a line of the text file
    std::getline(infile, line); //first line, map name
    maps.push_back(line);
    std::getline(infile, line); //second line, difficulty
    maps[i] = maps[i] + " (" + line + ")";
    infile.close();
  }
  showText_ = true;
  optionIndex = 0;
  // load text resources
  text_.setFont(fonts_.get(Fonts::GameTitleFont));
  //textGameOver_.setFont(font);
  text_.setString("Use Up and Down keys to navigate,\n"
                  "choose Map and press Enter");
  text_.setCharacterSize(30);
  text_.setFillColor(sf::Color::Black);
  text_.setPosition(tileSize * float(width) / 3.8f, tileSize * float(height) / 5.f);
  // load texture resources
  grassSprite_.setTexture(textures_.get(Textures::GrassTile));

  // create holder for map sprite that consists of multiple
  // small tiles
  mapTex_.clear();
  mapTex_.create(1280, 1024);
  for (int i = 0; i < height; ++i) {
    for (int j = 0; j < width; ++j) {
      grassSprite_.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
      mapTex_.draw(grassSprite_);
    }
  }
  mapSprite_.setTexture(mapTex_.getTexture());
  mapSprite_.setOrigin({0, mapSprite_.getLocalBounds().height});
  mapSprite_.setScale({1, -1});
  mapSprite_.setColor(sf::Color(255, 255, 255, 128)); // opacity 50%

  // initialize menu options
  sf::Text mapOption1;
  mapOption1.setFont(fonts_.get(Fonts::GameTitleFont));
  mapOption1.setString(maps[0]);
  mapOption1.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f);
  mapOption1.setFillColor(sf::Color::Red); // optionIndex = 0;
  mapOption1.setCharacterSize(60);
  sf::Text mapOption2;
  mapOption2.setFont(fonts_.get(Fonts::GameTitleFont));
  mapOption2.setString(maps[1]);
  mapOption2.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 100);
  mapOption2.setFillColor(sf::Color::Black);
  mapOption2.setCharacterSize(60);
  sf::Text mapOption3;
  mapOption3.setFont(fonts_.get(Fonts::GameTitleFont));
  mapOption3.setString(maps[2]);
  mapOption3.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 200);
  mapOption3.setFillColor(sf::Color::Black);
  mapOption3.setCharacterSize(60);
  sf::Text mapOption4;
  mapOption4.setFont(fonts_.get(Fonts::GameTitleFont));
  mapOption4.setString(maps[3]);
  mapOption4.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 300);
  mapOption4.setFillColor(sf::Color::Black);
  mapOption4.setCharacterSize(60);
  sf::Text mapOption5;
  mapOption5.setFont(fonts_.get(Fonts::GameTitleFont));
  mapOption5.setString(maps[4]);
  mapOption5.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 400);
  mapOption5.setFillColor(sf::Color::Black);
  mapOption5.setCharacterSize(60);

  options.push_back(mapOption1);
  options.push_back(mapOption2);
  options.push_back(mapOption3);
  options.push_back(mapOption4);
  options.push_back(mapOption5);


  //sf::Text
  // next scene:
//  sceneType.scene = Scenes::ID::LevelSelect;
//  sceneType.number = 0;
  nextScene_.scene = Scenes::ID::LevelSelect;
  nextScene_.number = 1;
}
```

14.12.3 Member Function Documentation

14.12.3.1 draw()

```
void LevelSelect::draw (
             sf::RenderTarget & target,
             sf::RenderStates states ) const  [override]
```

draw the scene to target

Parameters

target	sf::RenderTarget to draw the scene to
states	sf::RenderStates object for drawing

Definition at line 99 of file LevelSelect.cpp.

```
                                                                             {
  target.draw(mapSprite_);
  for (const auto &text : options) {
    target.draw(text);
  }
  target.draw(text_);
}
```

14.12.3.2 handleInput()

```
void LevelSelect::handleInput (
             const sf::Event & event )  [override], [virtual]
```

handle input for current scene

Parameters

event	sf::Event (keypress etc.)

Implements Scene.

Definition at line 106 of file LevelSelect.cpp.

```
                                                   {
  switch (event.type) {
    case sf::Event::KeyPressed:handlePlayerInput(event.key.code, true);
      break;
    case sf::Event::KeyReleased:handlePlayerInput(event.key.code, false);
      break;
    default:break;
      //case sf::Event::MouseButtonPressed:handlePlayerInput(event.mouseButton, true);
  }
}
```

14.12.3.3 handlePlayerInput()

```
void LevelSelect::handlePlayerInput (
              sf::Keyboard::Key key,
              bool isPressed )
```

Takes input from handleInput function.

Parameters

key	keyboard key that was pressed
isPressed	Is the key pressed currently?

Definition at line 134 of file LevelSelect.cpp.

```
                                                                  {
  //std::cout « "input: " « key « ", isPressed: " « isPressed « "\n";
  if (isPressed) {
    //std::cout « "START: index is: " « optionIndex « ", key is " « key « "\n";
    if (key == sf::Keyboard::Up) {
      if (optionIndex > 0)
        optionIndex--;
      else
        optionIndex = options.size() - 1;
      updateOptionText();
    } else if (key == sf::Keyboard::Down) {
      if (optionIndex < options.size() - 1)
        optionIndex++;
      else
        optionIndex = 0;
      updateOptionText();
    } else if (key == sf::Keyboard::Enter) {
      nextScene_.scene = Scenes::ID::MapScene;
      nextScene_.number = optionIndex + 1;
      //updateOptionText();
    }
    //std::cout « "END: index is: " « optionIndex « "key is" « key « "\n";
  }

//   switch (event.type) {
//     case sf::Event::MouseButtonPressed:
//       //handlePlayerInput(event.mouseButton, false);
//       break;
//     case sf::Event::MouseButtonReleased:
//       //handlePlayerInput(event.mouseButton, false);
//       break;
  //case sf::Event::MouseMoved:handlePlayerInput(event.)
//     case sf::Event::Closed:
//       window.close();
//       break;
//   }
}
```

14.12.3.4 requestedScene()

```
sceneRequest LevelSelect::requestedScene ( ) [override], [virtual]
```

Next scene from this one.

Returns

the request

Implements Scene.

Definition at line 96 of file LevelSelect.cpp.

```
                                              {
  return nextScene_;
}
```

14.12.3.5 sceneType()

```
Scenes::ID LevelSelect::sceneType ( )  [inline], [override], [virtual]
```

Return current scene's type.

Returns

current scene's type, GameEnd

Implements Scene.

Definition at line 61 of file LevelSelect.hpp.

```
61 { return Scenes::ID::LevelSelect; }
```

14.12.3.6 update()

```
void LevelSelect::update (
              sf::Time dt )  [override], [virtual]
```

Update scene function.

Parameters

dt	deltatime, time since last frame update

Implements Scene.

Definition at line 116 of file LevelSelect.cpp.

```
116                                   {
117 //  textEffectTime_ += dt;
118 //  if (textEffectTime_ >= sf::seconds(0.5f))
119 //  {
120 //    showText_ = !showText_;
121 //    textEffectTime_ = sf::Time::Zero;
122 //  }
123 }
```

14.12.3.7 updateOptionText()

```
void LevelSelect::updateOptionText ( )
```

Update, which option is selected (and so should be colored red)

Definition at line 124 of file LevelSelect.cpp.

```
124                                    {
125   if (options.empty())
126     return;
127   // Black all texts
128   for (size_t i = 0; i < options.size(); i++) {
129     options[i].setFillColor(sf::Color::Black);
130   }
131   // Red the selected text
132   options[optionIndex].setFillColor(sf::Color::Red);
133 }
```

14.12.4 Member Data Documentation

14.12.4.1 fonts_

```
FontHolder& LevelSelect::fonts_  [private]
```

Definition at line 74 of file LevelSelect.hpp.

14.12.4.2 grassSprite_

```
sf::Sprite LevelSelect::grassSprite_  [private]
```

Definition at line 78 of file LevelSelect.hpp.

14.12.4.3 height

```
int LevelSelect::height = 16  [private]
```

Definition at line 72 of file LevelSelect.hpp.

14.12.4.4 mapSprite_

```
sf::Sprite LevelSelect::mapSprite_  [private]
```

Definition at line 81 of file LevelSelect.hpp.

14.12.4.5 mapTex_

```
sf::RenderTexture LevelSelect::mapTex_  [private]
```

Definition at line 80 of file LevelSelect.hpp.

14.12.4.6 nextScene_

```
sceneRequest LevelSelect::nextScene_  [private]
```

Definition at line 82 of file LevelSelect.hpp.

14.12.4.7 optionIndex

```
std::size_t LevelSelect::optionIndex
```

What option is currently selected.

Definition at line 69 of file LevelSelect.hpp.

14.12.4.8 options

```
std::vector<sf::Text> LevelSelect::options
```

Options to choose from in this scene, so maps.

Definition at line 65 of file LevelSelect.hpp.

14.12.4.9 pathSprite

```
sf::Sprite LevelSelect::pathSprite  [private]
```

Definition at line 79 of file LevelSelect.hpp.

14.12.4.10 showText_

```
bool LevelSelect::showText_  [private]
```

Definition at line 77 of file LevelSelect.hpp.

14.12.4.11 text_

```
sf::Text LevelSelect::text_  [private]
```

Definition at line 76 of file LevelSelect.hpp.

14.12.4.12 textEffectTime_

```
sf::Time LevelSelect::textEffectTime_  [private]
```

Definition at line 75 of file LevelSelect.hpp.

14.12.4.13 textures_

```
TextureHolder& LevelSelect::textures_  [private]
```

Definition at line 73 of file LevelSelect.hpp.

14.12.4.14 width

```
int LevelSelect::width = 20  [private]
```

Definition at line 71 of file LevelSelect.hpp.

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

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

14.13 Map Class Reference

```
#include <Map.hpp>
```

Private Attributes

- std::string name
- std::string diff

14.13.1 Detailed Description

Definition at line 8 of file Map.hpp.

14.13.2 Member Data Documentation

14.13.2.1 diff

```
std::string Map::diff  [private]
```

Definition at line 12 of file Map.hpp.

14.13.2.2 name

```
std::string Map::name  [private]
```

Definition at line 11 of file Map.hpp.

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

- src/game/Map.hpp

14.14 MapGrid Class Reference

The game map consists of blocks, this class handles reading maps from file and rendering the grid.

```
#include <MapGrid.hpp>
```

Inheritance diagram for MapGrid:

Collaboration diagram for MapGrid:

Public Member Functions

- MapGrid (TextureHolder &textureholder, int mapNum, std::map< int, std::pair< int, int >> &pathMarkers)

  Constructor for the class.
- int getBlockAt (int x, int y) const

  Find out if a block is road or not.
- void setBlockRow (std::vector< int > &row)

  set a row of map blocks. Used by the constructor when reading a map
- virtual void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  draw the scene to target
- int getWidth () const

  Get map width.
- int getHeight () const

  Get map height.

Private Attributes

- int width = 16
- int height = 16
- std::string diff
- std::string name
- std::vector< std::vector< int > > map
- std::map< int, std::pair< int, int > > & pathMarkers_
- TextureHolder & textures_
- sf::RenderTexture mapTex_
- sf::Sprite mapSprite_
- int mapNum_

14.14.1 Detailed Description

The game map consists of blocks, this class handles reading maps from file and rendering the grid.

Definition at line 17 of file MapGrid.hpp.

14.14.2 Constructor & Destructor Documentation

14.14.2.1 MapGrid()

```
MapGrid::MapGrid (
            TextureHolder & textureholder,
            int mapNum,
            std::map< int, std::pair< int, int >> & pathMarkers )
```

Constructor for the class.

Parameters

textureholder	Reference to Resource class instance that holds textures for the map
mapNum	Which map to load from file (file=map(mapNum).txt)
pathMarkers	Map that will hold the enemy pathMarkers for this map

Definition at line 15 of file MapGrid.cpp.

```
                                                                                                                      :
      textures_(
      textureholder), mapNum_(mapNum), pathMarkers_(pathMarkers) {
    std::string mapfile = "src/media/maps/map";
    mapfile.append(std::to_string(mapNum));
    mapfile.append(".txt");
    std::ifstream infile(mapfile);
    std::string line; //line to store a line of the text file
    std::getline(infile, line); //first line, map name
    this->name = line;
    std::getline(infile, line); //second line, difficulty
    this->diff = line;
    std::getline(infile, line); //third line, map width
    this->width = std::stoi(line);
    std::getline(infile, line); //fourth line, map height
    this->height = std::stoi(line);
```

```
  //read the map
  for (int i = 0; i < height; ++i) {
    std::getline(infile, line);
    std::vector<int> row;
    int offset = 0;
    for (int j = 0; j < line.length() + 1; ++j) {
      if (line[j] == '+') {
        row.emplace_back(0);
      } else if (line[j] == '(') {
        row.emplace_back(1);
        std::string it;
        j++;
        while (line[j] != ')') {
          it += line[j];
          j++;
          offset++;
        }
        offset++;
        int key = std::stoi(it);
        pathMarkers_.insert(std::make_pair(key, std::make_pair(j - offset, i)));
      } else if (line[j] == '-') {
        row.emplace_back(1);
      }
    }
    setBlockRow(row);
  }
  infile.close();
  /*
  for (int i = 0; i < pathMarkers.size(); i++) {
    std::cout « i + 1 « ": " « pathMarkers.at(i + 1).first « "," « pathMarkers.at(i + 1).second «
    std::endl;
  }
  */

  mapTex_.clear();
  mapTex_.create(1280, 1024);
  sf::Sprite grassSprite;
  grassSprite.setTexture(textures_.get(Textures::GrassTile));
  sf::Sprite pathSprite;
  pathSprite.setTexture(textures_.get(Textures::PathTile));
  for (int i = 0; i < height; ++i) {
    for (int j = 0; j < width; ++j) {
      if (getBlockAt(j, i) == 0) {
        grassSprite.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
        mapTex_.draw(grassSprite);
      } else if (getBlockAt(j, i) == 1) {
        pathSprite.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
        mapTex_.draw(pathSprite);
      }
    }
  }
  mapSprite_.setTexture(mapTex_.getTexture());
  mapSprite_.setOrigin({0, mapSprite_.getLocalBounds().height});
  mapSprite_.setScale({1, -1});
}
```

14.14.3 Member Function Documentation

14.14.3.1 draw()

```
void MapGrid::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override], [virtual]
```

draw the scene to target

Parameters

target	sf::RenderTarget to draw the scene to
states	sf::RenderStates object for drawing

Definition at line 12 of file MapGrid.cpp.

```
{
  target.draw(mapSprite_);
}
```

14.14.3.2 getBlockAt()

```
int MapGrid::getBlockAt (
            int x,
            int y ) const  [inline]
```

Find out if a block is road or not.

Parameters

x	x-coordinate of the block (in blocks)
y	y-coordinate of the block (in blocks)

Returns

0=grass,1=road

Definition at line 32 of file MapGrid.hpp.

```
{ return map[y][x]; }
```

14.14.3.3 getHeight()

```
int MapGrid::getHeight ( ) const  [inline]
```

Get map height.

Returns

map height

Definition at line 53 of file MapGrid.hpp.

```
{ return height; }
```

14.14.3.4 getWidth()

```
int MapGrid::getWidth ( ) const  [inline]
```

Get map width.

Returns

map width

Definition at line 48 of file MapGrid.hpp.

```
{ return width; }
```

14.14.3.5 setBlockRow()

```
void MapGrid::setBlockRow (
             std::vector< int > & row )  [inline]
```

set a row of map blocks. Used by the constructor when reading a map

Parameters

row	row number to change

Definition at line 37 of file MapGrid.hpp.

```
37 { map.emplace_back(row); }
```

14.14.4 Member Data Documentation

14.14.4.1 diff

```
std::string MapGrid::diff  [private]
```

Definition at line 57 of file MapGrid.hpp.

14.14.4.2 height

```
int MapGrid::height = 16  [private]
```

Definition at line 56 of file MapGrid.hpp.

14.14.4.3 map

```
std::vector<std::vector<int> > MapGrid::map  [private]
```

Definition at line 59 of file MapGrid.hpp.

14.14.4.4 mapNum_

```
int MapGrid::mapNum_  [private]
```

Definition at line 64 of file MapGrid.hpp.

14.14.4.5 mapSprite_

```
sf::Sprite MapGrid::mapSprite_  [private]
```

Definition at line 63 of file MapGrid.hpp.

14.14.4.6 mapTex_

```
sf::RenderTexture MapGrid::mapTex_  [private]
```

Definition at line 62 of file MapGrid.hpp.

14.14.4.7 name

```
std::string MapGrid::name  [private]
```

Definition at line 58 of file MapGrid.hpp.

14.14.4.8 pathMarkers_

```
std::map<int, std::pair<int, int> >& MapGrid::pathMarkers_  [private]
```

Definition at line 60 of file MapGrid.hpp.

14.14.4.9 textures_

```
TextureHolder& MapGrid::textures_  [private]
```

Definition at line 61 of file MapGrid.hpp.

14.14.4.10 width

```
int MapGrid::width = 16  [private]
```

Definition at line 55 of file MapGrid.hpp.

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

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

14.15 MapScene Class Reference

A class used when ingame. Inherits from Scene class. The main scene of the game.

```
#include <MapScene.hpp>
```

Inheritance diagram for MapScene:

Collaboration diagram for MapScene:

Public Member Functions

- MapScene (TextureHolder &textures, FontHolder &fonts, SoundBufferHolder &soundBuffers, int mapNum)

  Constructor for the class.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  draw the scene to target
- void handleInput (const sf::Event &event) override

  handle input for current scene
- void update (sf::Time delta) override

  Update scene function.
- sceneRequest requestedScene () override

  Next scene from this one (GameOverScene)
- Scenes::ID sceneType () override

  Return current scene's type.

Public Attributes

- int mapNum_

  Current Map number.
- std::map< int, std::pair< int, int > > pathMarkers

  A map that contains enemy pathMarkers for current level (coordinates enemies go through)
- TowerMenu towerMenu_

  The towermenu class that is rendered with maps.

Private Attributes

- World world_
- TextureHolder & textures_
- SoundBufferHolder & soundBuffers_
- FontHolder & fonts_
- WaveController waveController_
- sf::Text fpsText
- int width = 16
- int height = 16
- GameStatusMenu statusMenu_
- GameCommandsMenu commandsMenu_
- WaveStart waveStart_
- WavePause wavePause_
- request nextScene_

14.15.1 Detailed Description

A class used when ingame. Inherits from Scene class. The main scene of the game.

Definition at line 19 of file MapScene.hpp.

14.15.2 Constructor & Destructor Documentation

14.15.2.1 MapScene()

```
MapScene::MapScene (
              TextureHolder & textures,
              FontHolder & fonts,
              SoundBufferHolder & soundBuffers,
              int mapNum )
```

Constructor for the class.

Parameters

textures	Reference to a TextureHolder instance
fonts	Reference to a FontHolder instance
soundBuffers	Reference to a SoundBufferHolder instance
mapNum	Map number

Definition at line 9 of file MapScene.cpp.

```
    : textures_(textures), soundBuffers_(soundBuffers), world_(textures, soundBuffers, mapNum,
    pathMarkers),
      mapNum_(mapNum), waveController_(textures, pathMarkers), fonts_(fonts), towerMenu_(world_),
      statusMenu_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 60.f), fonts),
      commandsMenu_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 2.2f),
                    sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 1.35f),
                    fonts, world_),
      waveStart_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 5.f), fonts),
      wavePause_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 2.9f), fonts) {
  //world_ = World(textures_, soundBuffers, mapNum, pathMarkers);
  fpsText.setFont(fonts_.get(Fonts::GameTitleFont));
  fpsText.setString("updates / second");
  fpsText.setCharacterSize(30);
  fpsText.setFillColor(sf::Color::Black);
  fpsText.setPosition(tileSize * float(width) / 30.f, tileSize * float(height) / 60.f);
  // initialize scene request struct
  nextScene_.scene = Scenes::ID::MapScene;
  nextScene_.number = 1;
}
```

14.15.3 Member Function Documentation

14.15.3.1 draw()

```
void MapScene::draw (
              sf::RenderTarget & target,
              sf::RenderStates states ) const [override]
```

draw the scene to target

Parameters

target	sf::RenderTarget to draw the scene to
states	sf::RenderStates object for drawing

Definition at line 29 of file MapScene.cpp.

```
                                                                        {
  world_.draw(target, states);
  //target.draw(fpsText);
  towerMenu_.draw(target, states);
  statusMenu_.draw(target, states);
  commandsMenu_.draw(target, states);
  waveStart_.draw(target, states);
  wavePause_.draw(target, states);
}
```

14.15.3.2 handleInput()

```
void MapScene::handleInput (
              const sf::Event & event )  [override], [virtual]
```

handle input for current scene

Parameters

event	sf::Event (keypress etc.)

Implements Scene.

Definition at line 38 of file MapScene.cpp.

```
                                                {
  //towerMenu_.handleInput(event, world_);
  towerMenu_.handleInput(event, world_);
  waveStart_.handleInput(event, world_);
  wavePause_.handleInput(event, world_);
}
```

14.15.3.3 requestedScene()

```
sceneRequest MapScene::requestedScene ( )  [override], [virtual]
```

Next scene from this one (GameOverScene)

Returns

the request

Implements Scene.

Definition at line 56 of file MapScene.cpp.

```
                                          {
  return nextScene_;
}
```

14.15.3.4 sceneType()

```
Scenes::ID MapScene::sceneType ( )  [inline], [override], [virtual]
```

Return current scene's type.

Returns

current scene's type, MapScene

Implements Scene.

Definition at line 54 of file MapScene.hpp.

```
{ return Scenes::ID::MapScene; }
```

14.15.3.5 update()

```
void MapScene::update (
            sf::Time delta )  [override], [virtual]
```

Update scene function.

Parameters

delta	deltatime, time since last frame update

Implements Scene.

Definition at line 44 of file MapScene.cpp.

```
                                {
  world_.update(deltaTime);
  if (!world_.paused) waveController_.update(deltaTime, world_);
  double fps = 1.0 / deltaTime.asSeconds();
  fpsText.setString(std::to_string(fps));
  statusMenu_.update(world_, waveController_.getWaveEnemiesLeft(), waveController_.getWaveNumber());
  waveStart_.update(world_, waveController_.getWaveEnemiesLeft());
  // check if game is over
  if (world_.getHP() <= 0) {
    nextScene_.scene = Scenes::ID::GameEnd;
  }
}
```

14.15.4 Member Data Documentation

14.15.4.1 commandsMenu_

```
GameCommandsMenu MapScene::commandsMenu_  [private]
```

Definition at line 77 of file MapScene.hpp.

14.15.4.2 fonts_

```
FontHolder& MapScene::fonts_  [private]
```

Definition at line 71 of file MapScene.hpp.

14.15.4.3 fpsText

```
sf::Text MapScene::fpsText  [private]
```

Definition at line 73 of file MapScene.hpp.

14.15.4.4 height

```
int MapScene::height = 16  [private]
```

Definition at line 75 of file MapScene.hpp.

14.15.4.5 mapNum_

```
int MapScene::mapNum_
```

Current Map number.

Definition at line 58 of file MapScene.hpp.

14.15.4.6 nextScene_

```
request MapScene::nextScene_  [private]
```

Definition at line 80 of file MapScene.hpp.

14.15.4.7 pathMarkers

```
std::map<int, std::pair<int, int> > MapScene::pathMarkers
```

A map that contains enemy pathMarkers for current level (coordinates enemies go through)

Definition at line 62 of file MapScene.hpp.

14.15.4.8 soundBuffers_

`SoundBufferHolder& MapScene::soundBuffers_ [private]`

Definition at line 70 of file MapScene.hpp.

14.15.4.9 statusMenu_

`GameStatusMenu MapScene::statusMenu_ [private]`

Definition at line 76 of file MapScene.hpp.

14.15.4.10 textures_

`TextureHolder& MapScene::textures_ [private]`

Definition at line 69 of file MapScene.hpp.

14.15.4.11 towerMenu_

`TowerMenu MapScene::towerMenu_`

The towermenu class that is rendered with maps.

Definition at line 66 of file MapScene.hpp.

14.15.4.12 waveController_

`WaveController MapScene::waveController_ [private]`

Definition at line 72 of file MapScene.hpp.

14.15.4.13 wavePause_

`WavePause MapScene::wavePause_ [private]`

Definition at line 79 of file MapScene.hpp.

14.15.4.14 waveStart_

```
WaveStart MapScene::waveStart_  [private]
```

Definition at line 78 of file MapScene.hpp.

14.15.4.15 width

```
int MapScene::width = 16  [private]
```

Definition at line 74 of file MapScene.hpp.

14.15.4.16 world_

```
World MapScene::world_  [private]
```

Definition at line 68 of file MapScene.hpp.

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

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

14.16 Price Class Reference

```
#include <Price.hpp>
```

Inheritance diagram for Price:

Collaboration diagram for Price:

Public Member Functions

- Price (int value, const sf::Vector2f &position)
- virtual void draw (sf::RenderTarget &target, sf::RenderStates states) const override

Static Private Member Functions

- static std::string valueToString (int value)

Private Attributes

- sf::Text price

14.16.1 Detailed Description

Definition at line 5 of file Price.hpp.

14.16.2 Constructor & Destructor Documentation

14.16.2.1 Price()

```
Price::Price (
            int value,
            const sf::Vector2f & position )
```

Definition at line 6 of file Price.cpp.

```
                                                     {
    price.setFillColor(PriceColor);
    price.setCharacterSize(fontSize);
    price.setString(valueToString(value));
     price.setPosition(position);
     //have to set font
 }
```

14.16.3 Member Function Documentation

14.16.3.1 draw()

```
void Price::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override], [virtual]
```

Definition at line 14 of file Price.cpp.

```
                                                                          {
 }
```

14.16.3.2 valueToString()

```
std::string Price::valueToString (
            int value )  [static], [private]
```

Definition at line 17 of file Price.cpp.

```
                                      {
     return ((value >= 0) ? "+$" : "-$") + std::to_string(abs(value));
 }
```

14.16.4 Member Data Documentation

14.16.4.1 price

```
sf::Text Price::price  [private]
```

Definition at line 10 of file Price.hpp.

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

- src/ui/Price.hpp
- src/ui/Price.cpp

14.17 Projectile Class Reference

Extends Entity class, Bombs, Bullets etc.

```
#include <Projectile.hpp>
```

Inheritance diagram for Projectile:

Collaboration diagram for Projectile:

Public Member Functions

- Projectile (TextureHolder &textures, int textureID, ProjectileType type, Sector start, const std::pair< float, std::shared_ptr< Enemy >> &closestEnemy, const std::map< int, std::pair< float, std::shared_ptr< Enemy >>> &closestEnemies, std::map< int, std::pair< float, std::shared_ptr< Enemy >>> neighbourEnemies, float range, int damage, float velocity)
- void update (sf::Time deltaTime, World &world)

  update projectile in World
- bool ifShouldRemove () const

  Should the projectile be removed, has it hit something?

Static Public Member Functions

- static ProjectilePtr make (TextureHolder &textures, int textureID, ProjectileType type, Sector start, const std::pair< float, std::shared_ptr< Enemy >> &closestEnemy, const std::map< int, std::pair< float, std↩ ::shared_ptr< Enemy >>> &closestEnemies, const std::map< int, std::pair< float, std::shared_ptr< Enemy >>> &neighbourEnemies, float range, int damage, float velocity)

  make new projectile

Public Attributes

- ProjectileType type_

  type of this projectile

Private Attributes

- Sector start_
- Sector finish_
- std::pair< float, std::shared_ptr< Enemy > > closestEnemy_
- std::map< int, std::pair< float, std::shared_ptr< Enemy > > > closestEnemies_
- std::map< int, std::pair< float, std::shared_ptr< Enemy > > > neighbourEnemies_
- TextureHolder & textures_
- float range_
- int damage_
- float velocity_
- float distanceCovered
- float distance
- bool atFinish

Additional Inherited Members

14.17.1 Detailed Description

Extends Entity class, Bombs, Bullets etc.

Definition at line 31 of file Projectile.hpp.

14.17.2 Constructor & Destructor Documentation

14.17.2.1 Projectile()

```
Projectile::Projectile (
            TextureHolder & textures,
            int textureID,
            ProjectileType type,
            Sector start,
            const std::pair< float, std::shared_ptr< Enemy >> & closestEnemy,
            const std::map< int, std::pair< float, std::shared_ptr< Enemy >>> & closest↩
Enemies,
            std::map< int, std::pair< float, std::shared_ptr< Enemy >>> neighbourEnemies,
            float range,
            int damage,
            float velocity )
```

Parameters

textures	Reference to a TextureHolder class instance
textureID	an integer, index of wanted texture in the TextureHolder
type	Is the projectile a bullet or a bomb, etc.
start	Start location as a Sector type
closestEnemy	The closest enemy to The projectile, target
closestEnemies	A map of closest enemies, needed for splash damage
neighbourEnemies	
range	Splash damage range
damage	How much damage does the projectile do
velocity	How fast the projectile moves

Definition at line 5 of file Projectile.cpp.

```

                             :
   Entity(textureID, textures),
   textures_(textures),
   type_(type),
   start_(start),
    closestEnemy_(closestEnemy),
    neighbourEnemies_(std::move(neighbourEnemies)),
    range_(range),
    damage_(damage),
    velocity_(velocity),
    distanceCovered(0.f),
    distance(closestEnemy.first),
    atFinish(false)
{

  for (auto const& [key, value] : closestEnemies) {
    std::shared_ptr pTemp (value.second);
    std::pair <int, std::pair<float, std::shared_ptr<Enemy>> p = {key, {value.first, pTemp}};
    closestEnemies_.insert(p);
  }
  for (auto const& [key, value] : neighbourEnemies_) {
    std::shared_ptr pTemp (value.second);
    std::pair <int, std::pair<float, std::shared_ptr<Enemy>> p = {key, {value.first, pTemp}};
    neighbourEnemies_.insert(p);
  }

  if (type == ProjectileType::Bomb) {

    entitySprite.setPosition((float)start_.x , (float)start_.y);
  }
  else {
    entitySprite.setPosition((float)start_.x + 32.f , (float)start_.y + 32.f);
  }
}
```

14.17.3 Member Function Documentation

14.17.3.1 ifShouldRemove()

```
bool Projectile::ifShouldRemove ( ) const  [inline]
```

Should the projectile be removed, has it hit something?

Returns

Whether the projectile should be removed, true is should

Definition at line 72 of file Projectile.hpp.

```
{ return atFinish; }
```

14.17.3.2 make()

```
ProjectilePtr Projectile::make (
            TextureHolder & textures,
            int textureID,
            ProjectileType type,
            Sector start,
            const std::pair< float, std::shared_ptr< Enemy >> & closestEnemy,
            const std::map< int, std::pair< float, std::shared_ptr< Enemy >>> & closest↩
Enemies,
            const std::map< int, std::pair< float, std::shared_ptr< Enemy >>> & neighbour↩
Enemies,
            float range,
            int damage,
            float velocity )  [static]
```

make new projectile

Parameters

textures	Reference to a TextureHolder class instance
textureID	an integer, index of wanted texture in the TextureHolder
type	Is the projectile a bullet or a bomb, etc.
start	Start location as a Sector type
closestEnemy	The closest enemy to The projectile, target
closestEnemies	A map of closest enemies, needed for splash damage
neighbourEnemies	
range	Splash damage range
damage	How much damage does the projectile do
velocity	How fast the projectile moves

Returns

unique_ptr to the Projectile

Definition at line 137 of file Projectile.cpp.

```
                                             {
  std::map<int, std::pair<float, std::shared_ptr<Enemy>>> closestEnemiesTemp;
  for (auto const& [key, value] : neighbourEnemies) {
    std::shared_ptr pTemp (value.second);
    std::pair <int, std::pair<float, std::shared_ptr<Enemy>>> p = {key, {value.first, pTemp}};
    closestEnemiesTemp.insert(p);
  }
  std::map<int, std::pair<float, std::shared_ptr<Enemy>>> neighbourEnemiesTemp;
  for (auto const& [key, value] : neighbourEnemies) {
    std::shared_ptr pTemp (value.second);
    std::pair <int, std::pair<float, std::shared_ptr<Enemy>>> p = {key, {value.first, pTemp}};
    neighbourEnemiesTemp.insert(p);
  }
  return std::make_unique<Projectile>(textures, textureID, type, start, closestEnemy,
   closestEnemiesTemp, neighbourEnemiesTemp, range, damage, velocity);
}
```

14.17.3.3 update()

```
void Projectile::update (
            sf::Time deltaTime,
            World & world )  [virtual]
```

update projectile in World

Parameters

deltaTime	time since last frame
world	reference to World class where the Projectile is

Implements Entity.

Definition at line 40 of file Projectile.cpp.

```
                                                         {
    //closestEnemies_= getClosestEnemies(world);
    if(closestEnemy_.second==nullptr) {return;}
    float enemyX = closestEnemy_.second->getPosition().x;
    float enemyY = closestEnemy_.second->getPosition().y;
    //std::shared_ptr<Enemy> freezeEnemy=nullptr;
    std::shared_ptr<Enemy> freezeEnemy (closestEnemy_.second);
    if (type_ == ProjectileType::FreezeGun) {
      for (int i = 0; i < closestEnemies_.size(); i++) {
        if (closestEnemies_[i].second!=nullptr && closestEnemies_[i].first <= range_) {
          if (closestEnemies_[i].second->isNotFrozen()) {
            enemyX = closestEnemies_[i].second->getPosition().x;
            enemyY = closestEnemies_[i].second->getPosition().y;
            freezeEnemy = closestEnemies_[i].second;
            break;
          }
        }
      }
    }

    auto towerX = float(start_.x);
    auto towerY = float(start_.y);
    float distanceX = abs(enemyX - towerX);
    float distanceY = abs(enemyY - towerY);

    float factorX = distanceX / distance;
    float factorY = distanceY / distance;

    float velocityX = velocity_ * factorX;
    float velocityY = velocity_ * factorY;

    if (enemyX < towerX) {
      velocityX = -velocityX;
    }
    if (enemyY < towerY) {
      velocityY = -velocityY;
    }

    //std::cout « "DistanceCovered: " « distanceCovered « "   Distance: " « distance «std::endl;
    //std::cout « entitySprite.getPosition().x « "," « entitySprite.getPosition().y « std::endl;
    //std::cout « velocityX « "," « velocityY « std::endl;

    if (type_ == ProjectileType::Bullet) {
      if (distanceCovered <= distance) {
        entitySprite.move(velocityX, velocityY);
        distanceCovered += velocity_;
      }
      else {
        if (closestEnemy_.second != nullptr) {
          closestEnemy_.second->takeDamage(damage_);
        }
        atFinish = true;
      }
    }
    else if (type_ == ProjectileType::FreezeGun) {
      if (distanceCovered <= distance) {
        entitySprite.move(velocityX, velocityY);
        distanceCovered += velocity_;
      }
      else {
         if (freezeEnemy != nullptr) {
           freezeEnemy->slowDown();
           freezeEnemy->takeDamage(damage_);
         }
         atFinish = true;
       }
     }
     else if (type_ == ProjectileType::Bomb) {
       //Explosion texture
       if (distance-distanceCovered <= 30.f) {
         entitySprite.setTexture(textures_.get(Textures::Explosion));
       }
       //Check if at finish
       if (distanceCovered <= distance) {
```

```
      entitySprite.move(velocityX, velocityY);
      distanceCovered += velocity_;
    }
    else {
      //Closest enemy takes damage
      if (closestEnemy_.second != nullptr) {
        closestEnemy_.second->takeDamage(damage_);
      }

      //Other enemies inside range take damage as well
      for (auto enemy : neighbourEnemies_) {
        //std::cout « enemy.second.first « std::endl;
        if (enemy.second.first <= range_) {
          if (enemy.second.second != nullptr) {
            enemy.second.second->takeDamage(damage_);
          }
        }
      }
      atFinish = true;
    }
  }
  //std::cout « "DistanceCovered: " « distanceCovered « "   Distance: " « distance «std::endl;
}
```

14.17.4 Member Data Documentation

14.17.4.1 atFinish

`bool Projectile::atFinish [private]`

Definition at line 90 of file Projectile.hpp.

14.17.4.2 closestEnemies_

`std::map<int, std::pair<float, std::shared_ptr<Enemy> > > Projectile::closestEnemies_↩ [private]`

Definition at line 82 of file Projectile.hpp.

14.17.4.3 closestEnemy_

`std::pair<float, std::shared_ptr<Enemy> > Projectile::closestEnemy_ [private]`

Definition at line 81 of file Projectile.hpp.

14.17.4.4 damage_

`int Projectile::damage_ [private]`

Definition at line 86 of file Projectile.hpp.

14.17.4.5 distance

```
float Projectile::distance  [private]
```

Definition at line 89 of file Projectile.hpp.

14.17.4.6 distanceCovered

```
float Projectile::distanceCovered  [private]
```

Definition at line 88 of file Projectile.hpp.

14.17.4.7 finish_

```
Sector Projectile::finish_  [private]
```

Definition at line 80 of file Projectile.hpp.

14.17.4.8 neighbourEnemies_

```
std::map<int, std::pair<float, std::shared_ptr<Enemy> > > Projectile::neighbourEnemies_↩
[private]
```

Definition at line 83 of file Projectile.hpp.

14.17.4.9 range_

```
float Projectile::range_  [private]
```

Definition at line 85 of file Projectile.hpp.

14.17.4.10 start_

```
Sector Projectile::start_  [private]
```

Definition at line 79 of file Projectile.hpp.

14.17.4.11 textures_

```
TextureHolder& Projectile::textures_ [private]
```

Definition at line 84 of file Projectile.hpp.

14.17.4.12 type_

```
ProjectileType Projectile::type_
```

type of this projectile

Definition at line 76 of file Projectile.hpp.

14.17.4.13 velocity_

```
float Projectile::velocity_ [private]
```

Definition at line 87 of file Projectile.hpp.

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

- src/entity/Projectile.hpp
- src/entity/Projectile.cpp

14.18 ProjectileType Class Reference

Contains identification for different projectiles.

```
#include <Projectile.hpp>
```

14.18.1 Detailed Description

Contains identification for different projectiles.

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

- src/entity/Projectile.hpp

14.19 request Struct Reference

```
#include <Scene.hpp>
```

Public Attributes

- Scenes::ID scene
- size_t number

14.19.1 Detailed Description

Definition at line 25 of file Scene.hpp.

14.19.2 Member Data Documentation

14.19.2.1 number

```
size_t request::number
```

Definition at line 27 of file Scene.hpp.

14.19.2.2 scene

```
Scenes::ID request::scene
```

Definition at line 26 of file Scene.hpp.

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

- src/game/Scene.hpp

14.20 ResourceHolder< Resource, Identifier > Class Template Reference

```
#include <Resource.hpp>
```

Public Member Functions

- void load (Identifier id, const std::string &filename)

  A template function for a Resource.
- Resource & get (Identifier id)
- const Resource & get (Identifier id) const

Private Attributes

- std::map< Identifier, std::unique_ptr< Resource > > mResourceMap

14.20.1 Detailed Description

template<typename Resource, typename Identifier>
class ResourceHolder< Resource, Identifier >

Template Parameters

Resource	The resource, could be a texture, or font, etc.
Identifier	Easily readable name for the texture

Definition at line 55 of file Resource.hpp.

14.20.2 Member Function Documentation

14.20.2.1 get() `[1/2]`

```
template<typename Resource , typename Identifier >
Resource & ResourceHolder< Resource, Identifier >::get (
            Identifier id )
```

Parameters

id	Identifier for wanted resource

Returns

Reference to wanted resource

Definition at line 92 of file Resource.hpp.

```
                                                                        {
    auto found = mResourceMap.find(id);
    assert(found != mResourceMap.end());
    return *found->second;
}
```

14.20.2.2 get() `[2/2]`

```
template<typename Resource , typename Identifier >
const Resource & ResourceHolder< Resource, Identifier >::get (
            Identifier id ) const
```

Definition at line 98 of file Resource.hpp.

```
                                                                                  {
    auto found = mResourceMap.find(id);
    assert(found != mResourceMap.end());
    return *found->second;
}
```

14.20.2.3 load()

```
template<typename Resource , typename Identifier >
void ResourceHolder< Resource, Identifier >::load (
            Identifier id,
            const std::string & filename )
```

A template function for a Resource.

Parameters

id	Identifier for this texture
filename	File to load to this identifier

Template Parameters

Resource	
Identifier	Identifier for resources

Parameters

id	Identifier name, used when accessing a Resource
filename	File to load from

Definition at line 82 of file Resource.hpp.

```
                                                                           {
  std::unique_ptr<Resource> resource(new Resource());
  if (!resource->loadFromFile(filename))
    throw std::runtime_error("ResourceHolder::load - Failed to load " + filename);
  auto inserted = mResourceMap.insert(
      std::make_pair(id, std::move(resource)));
  assert(inserted.second);
}
```

14.20.3 Member Data Documentation

14.20.3.1 mResourceMap

```
template<typename Resource , typename Identifier >
std::map<Identifier, std::unique_ptr<Resource> > ResourceHolder< Resource, Identifier >::m↩
ResourceMap [private]
```

Definition at line 72 of file Resource.hpp.

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

- src/resource/Resource.hpp

14.21 Scene Class Reference

This is a class for different UI "pages" of the game such as main menu or the game itself.

```
#include <Scene.hpp>
```

Inheritance diagram for Scene:

Collaboration diagram for Scene:

Public Member Functions

- virtual void handleInput (const sf::Event &event)=0
- virtual void update (sf::Time deltaTime)=0

  updates everything is a scene including UI elements and entities
- virtual sceneRequest requestedScene ()=0

  Should the current scene be changed.
- virtual Scenes::ID sceneType ()=0

  Returns the ID the scene, does not change.

Public Attributes

- TextureHolder textures

14.21.1 Detailed Description

This is a class for different UI "pages" of the game such as main menu or the game itself.

These scenes do not have many relationships in the code. Scene object is called directly from the main Game loop. All scenes handle input, draw GUI and allow to change to next scene.

Definition at line 37 of file Scene.hpp.

14.21.2 Member Function Documentation

14.21.2.1 handleInput()

```
virtual void Scene::handleInput (
             const sf::Event & event )  [pure virtual]
```

Implemented in MapScene, LevelSelect, GameTitle, and GameEnd.

14.21.2.2 requestedScene()

```
virtual sceneRequest Scene::requestedScene ( )  [pure virtual]
```

Should the current scene be changed.

Returns

Returns the next scene ID (GUI page type) or the current scene ID if no need to change

Implemented in MapScene, LevelSelect, GameTitle, and GameEnd.

14.21.2.3 sceneType()

```
virtual Scenes::ID Scene::sceneType ( ) [pure virtual]
```

Returns the ID the scene, does not change.

Returns

Does not change after scene creation

Implemented in MapScene, LevelSelect, GameTitle, and GameEnd.

14.21.2.4 update()

```
virtual void Scene::update (
            sf::Time deltaTime ) [pure virtual]
```

updates everything is a scene including UI elements and entities

<

Parameters

deltaTime	time since last update in Game loop

Implemented in LevelSelect, GameTitle, GameEnd, and MapScene.

14.21.3 Member Data Documentation

14.21.3.1 textures

```
TextureHolder Scene::textures
```

Definition at line 55 of file Scene.hpp.

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

- src/game/Scene.hpp

14.22 SceneItem Class Reference

```
#include <SceneItem.hpp>
```

Inheritance diagram for SceneItem:

Collaboration diagram for SceneItem:

Public Types

- typedef std::unique_ptr< SceneItem > Ptr

Public Member Functions

- SceneItem ()

Private Member Functions

- virtual void draw (sf::RenderTarget &target, sf::RenderStates states) const

14.22.1 Detailed Description

Definition at line 10 of file SceneItem.hpp.

14.22.2 Member Typedef Documentation

14.22.2.1 Ptr

```
typedef std::unique_ptr<SceneItem> SceneItem::Ptr
```

Definition at line 12 of file SceneItem.hpp.

14.22.3 Constructor & Destructor Documentation

14.22.3.1 SceneItem()

```
SceneItem::SceneItem ( )
```

14.22.4 Member Function Documentation

14.22.4.1 draw()

```
void SceneItem::draw (
             sf::RenderTarget & target,
             sf::RenderStates states ) const  [private], [virtual]
```

Definition at line 7 of file SceneItem.cpp.

```
                                                    {
  // see book p.60
   states.transform *= getTransform();
}
```

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

- src/sceneItem/SceneItem.hpp
- src/sceneItem/SceneItem.cpp

14.23 Sector Struct Reference

A Sector is a 64x64 pixel block in the game map The Sector class is used to align towers properly.

`#include <Sector.hpp>`

Public Member Functions

- sf::Vector2f upperLeftPoint () const
- bool operator== (const Sector &rhs) const

Static Public Member Functions

- template<typename T >
  static Sector fromCoords (T xCoord, T yCoord)

Public Attributes

- int x
- int y

Static Public Attributes

- static const int Size = 64
- static const sf::Vector2f DiagVector = sf::Vector2f(Sector::Size, Sector::Size)
- static const float scale = 1.f

14.23.1 Detailed Description

A Sector is a 64x64 pixel block in the game map The Sector class is used to align towers properly.

Definition at line 12 of file Sector.hpp.

14.23.2 Member Function Documentation

14.23.2.1 fromCoords()

```
template<typename T >
Sector Sector::fromCoords (
            T xCoord,
            T yCoord )  [inline], [static]
```

Definition at line 30 of file Sector.hpp.

```
                                                       {
   int xCoordMod = xCoord % 64;
   int yCoordMod = yCoord % 64;
   return Sector{static_cast<int>(xCoord - xCoordMod), static_cast<int>(yCoord - yCoordMod)};
 }
```

14.23.2.2 operator==()

```
bool Sector::operator== (
            const Sector & rhs ) const
```

Definition at line 15 of file Sector.cpp.

```
                                                  {
   return x == rhs.x && y == rhs.y;
 }
```

14.23.2.3 upperLeftPoint()

```
sf::Vector2f Sector::upperLeftPoint ( ) const
```

Definition at line 11 of file Sector.cpp.

```
                                           {
   return sf::Vector2f(static_cast<float>(x), static_cast<float>(y));
 }
```

14.23.3 Member Data Documentation

14.23.3.1 DiagVector

```
const sf::Vector2f Sector::DiagVector = sf::Vector2f(Sector::Size, Sector::Size)  [static]
```

Definition at line 21 of file Sector.hpp.

14.23.3.2 scale

```
const float Sector::scale = 1.f  [static]
```

Definition at line 22 of file Sector.hpp.

14.23.3.3 Size

```
const int Sector::Size = 64  [static]
```

Definition at line 20 of file Sector.hpp.

14.23.3.4 x

```
int Sector::x
```

Definition at line 13 of file Sector.hpp.

14.23.3.5 y

```
int Sector::y
```

Definition at line 14 of file Sector.hpp.

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

- src/ui/Sector.hpp
- src/ui/Sector.cpp

14.24 SelectTowerButton< T > Class Template Reference

```
#include <SelectTowerButton.hpp>
```

Inheritance diagram for SelectTowerButton< T >:

Collaboration diagram for SelectTowerButton< T >:

Public Member Functions

- SelectTowerButton (const sf::Vector2f &position, const Sector &towerSector)
- virtual void draw (sf::RenderTarget &target, sf::RenderStates states) const override

Protected Member Functions

- virtual void onClick (World &world) override

Protected Attributes

- Sector towerSector

14.24.1 Detailed Description

template<typename T>
class SelectTowerButton< T >

Definition at line 11 of file SelectTowerButton.hpp.

14.24.2 Constructor & Destructor Documentation

14.24.2.1 SelectTowerButton()

```
template<typename T >
SelectTowerButton< T >::SelectTowerButton (
             const sf::Vector2f & position,
             const Sector & towerSector )
```

Definition at line 22 of file SelectTowerButton.hpp.

```
                                                                                   :
      Button(position, T::TEXTURE_ID),
      towerSector(towerSector) {
 }
```

14.24.3 Member Function Documentation

14.24.3.1 draw()

```
template<typename T >
void SelectTowerButton< T >::draw (
             sf::RenderTarget & target,
             sf::RenderStates states ) const  [override], [virtual]
```

Reimplemented from Button.

Definition at line 28 of file SelectTowerButton.hpp.

```
                                                                           {

    Button::draw(target, states);
 }
```

14.24.3.2 onClick()

```
template<typename T >
void SelectTowerButton< T >::onClick (
            World & world ) [override], [protected], [virtual]
```

Reimplemented from Button.

Definition at line 34 of file SelectTowerButton.hpp.

```
                                                     {
  Button::onClick(world);
}
```

14.24.4 Member Data Documentation

14.24.4.1 towerSector

```
template<typename T >
Sector SelectTowerButton< T >::towerSector [protected]
```

Definition at line 17 of file SelectTowerButton.hpp.

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

- src/ui/SelectTowerButton.hpp

14.25 Tower Class Reference

Tower to display on the map and shoot, extends Entity class.

`#include <Tower.hpp>`

Inheritance diagram for Tower:

Collaboration diagram for Tower:

Public Member Functions

- Tower (TextureHolder &textures, SoundBufferHolder &sounds, int textureID, Sector sector, int damage, float range, sf::Time shootDelay, int price, TowerType towerType)
- void update (sf::Time deltaTime, World &world)

  Update the projectile after a frame change.
- bool upgrade ()

  Function to call when this tower is upgraded.
- float getRange () const

  Get this tower's range as a float value.
- int getPrice () const

  Get this tower's price.
- const Sector & getSector () const

  Get a reference to the Sector this tower is in.

Public Attributes

- TowerType type

    This tower's type.

Private Member Functions

- bool inRange (const sf::Vector2f enemyPos, float range)
- std::pair< float, std::shared_ptr< Enemy > > getClosestEnemy (World &world)
- std::map< int, std::pair< float, std::shared_ptr< Enemy > > > getClosestEnemies (World &world)
- std::map< int, std::pair< float, std::shared_ptr< Enemy > > > getNeighbourEnemies (std::pair< float, std↩
::shared_ptr< Enemy >> ClosestEnemy, World &world)

Private Attributes

- bool upgradeable
- Sector sector
- TextureHolder & textures_
- SoundBufferHolder & sounds_
- std::map< int, std::pair< int, int > > testii
- float bombRange
- int damage
- float range
- int price
- float velocity_
- sf::Sound soundBullet_
- sf::Sound soundBomb_
- sf::Sound soundSnow_
- sf::Time shootDelay
- sf::Time timeSinceFiring

Additional Inherited Members

14.25.1 Detailed Description

Tower to display on the map and shoot, extends Entity class.

Definition at line 33 of file Tower.hpp.

14.25.2 Constructor & Destructor Documentation

14.25.2.1 Tower()

```
Tower::Tower (
            TextureHolder & textures,
            SoundBufferHolder & sounds,
            int textureID,
            Sector sector,
            int damage,
            float range,
            sf::Time shootDelay,
            int price,
            TowerType towerType )
```

Definition at line 9 of file Tower.cpp.

```
                                      :
  Entity(textureID, textures),
  textures_(textures),
  sounds_(sounds),
  sector(sector),
  damage(damage),
  range(range),
  shootDelay(shootDelay),
  timeSinceFiring(sf::Time::Zero),
  price(price),
  type(towerType),
  upgradeable(true),
  bombRange(150.f),
  velocity_(10.f)
{
  entitySprite.setPosition(sector.x, sector.y);
  soundBullet_.setBuffer(sounds.get(SoundBuffers::GunCat));
  soundBullet_.setVolume(50.f);
  soundBomb_.setBuffer(sounds.get(SoundBuffers::BombCatMeow));
  soundSnow_.setBuffer(sounds.get(SoundBuffers::FreezeCatMeow));
  if (towerType == TowerType::FreezeCat) {
    upgradeable = false;
  }
}
```

14.25.3 Member Function Documentation

14.25.3.1 getClosestEnemies()

```
std::map< int, std::pair< float, std::shared_ptr< Enemy > > > Tower::getClosestEnemies (
            World & world ) [private]
```

Definition at line 102 of file Tower.cpp.

```
                                                                                    {
  std::map<float, std::shared_ptr<Enemy» enemyDistances;

  float towerX = entitySprite.getPosition().x;
  float towerY = entitySprite.getPosition().y;
  for (const auto& enemy : world.getEnemies()) {
    float enemyX = enemy->getPosition().x;
    float enemyY = enemy->getPosition().y;
    float distanceX = abs(enemyX - towerX);
    float distanceY = abs(enemyY - towerY);
    float distance = sqrt(distanceX * distanceX + distanceY * distanceY);
    enemyDistances.insert(std::make_pair(distance, std::shared_ptr<Enemy>(enemy)));
  }
  std::map<int, std::pair<float, std::shared_ptr<Enemy»> closestEnemies;
  int counter = 0;
  for (std::map<float, std::shared_ptr<Enemy»::iterator enemy = enemyDistances.begin(); enemy !=
   enemyDistances.end(); enemy++) {
    // Sort into result
    closestEnemies[counter] = std::make_pair(enemy->first, std::shared_ptr<Enemy>(enemy->second));
    counter++;
  }

  return closestEnemies;
}
```

14.25.3.2 getClosestEnemy()

```
std::pair< float, std::shared_ptr< Enemy > > Tower::getClosestEnemy (
            World & world )  [private]
```

Definition at line 76 of file Tower.cpp.

```
                                                                          {
  std::shared_ptr<Enemy> closestEnemy = nullptr;
  float closestDistance = 0.0;

  float towerX = entitySprite.getPosition().x;
  float towerY = entitySprite.getPosition().y;

  for (auto&& enemy : world.getEnemies()) {

    float enemyX = enemy->getPosition().x;
    float enemyY = enemy->getPosition().y;

    float distanceX = abs(enemyX - towerX);
    float distanceY = abs(enemyY - towerY);

    float distance = sqrt(distanceX * distanceX + distanceY * distanceY);

    if (closestDistance == 0.0 || distance < closestDistance) {
      closestEnemy = enemy;
      closestDistance = distance;
    }
  }
  //std::cout « closestDistance « std::endl;
  return std::make_pair(closestDistance, closestEnemy);
}
```

14.25.3.3 getNeighbourEnemies()

```
std::map< int, std::pair< float, std::shared_ptr< Enemy > > > Tower::getNeighbourEnemies (
            std::pair< float, std::shared_ptr< Enemy >> ClosestEnemy,
            World & world )  [private]
```

Definition at line 126 of file Tower.cpp.

```
                                          {
  std::map<float, std::shared_ptr<Enemy» enemyDistances;

  float closestEnemyX = closestEnemy.second->getPosition().x;
  float closestEnemyY = closestEnemy.second->getPosition().y;
  for (auto& enemy : world.getEnemies()) {
    float enemyX = enemy->getPosition().x;
    float enemyY = enemy->getPosition().y;
    float distanceX = abs(enemyX - closestEnemyX);
    float distanceY = abs(enemyY - closestEnemyY);
    float distance = sqrt(distanceX * distanceX + distanceY * distanceY);
    if (enemy != closestEnemy.second) {
      enemyDistances.insert(std::make_pair(distance, std::shared_ptr<Enemy>(enemy)));
    }
  }
  std::map<int, std::pair<float, std::shared_ptr<Enemy»> neighbourEnemies;
  int counter = 0;
  for (std::map<float, std::shared_ptr<Enemy»::iterator enemy = enemyDistances.begin(); enemy !=
   enemyDistances.end(); enemy++) {
    // Sort into result
    neighbourEnemies[counter] = std::make_pair(enemy->first, std::shared_ptr<Enemy>(enemy->second));
    counter++;
  }

  return neighbourEnemies;
}
```

14.25.3.4 getPrice()

```
int Tower::getPrice ( ) const  [inline]
```

Get this tower's price.

Returns

Price as int

Definition at line 64 of file Tower.hpp.

```
64 { return price; }
```

14.25.3.5 getRange()

```
float Tower::getRange ( ) const  [inline]
```

Get this tower's range as a float value.

Returns

Radius of the range as float

Definition at line 59 of file Tower.hpp.

```
59 { return range; };
```

14.25.3.6 getSector()

```
const Sector& Tower::getSector ( ) const  [inline]
```

Get a reference to the Sector this tower is in.

Returns

Reference to location Sector

Definition at line 69 of file Tower.hpp.

```
69 { return sector; }
```

14.25.3.7 inRange()

```
bool Tower::inRange (
              const sf::Vector2f enemyPos,
              float range )  [private]
```

Definition at line 152 of file Tower.cpp.

```
                                                              {
  float enemyX = enemyPos.x;
  float enemyY = enemyPos.y;
  float towerX = entitySprite.getPosition().x;
  float towerY = entitySprite.getPosition().y;
  float distanceX = abs(enemyX - towerX);
  float distanceY = abs(enemyY - towerY);
  float distance = sqrt(distanceX * distanceX + distanceY * distanceY);
  if (distance <= range) {
    return true;
  }
  else
    return false;
}
```

14.25.3.8 update()

```
void Tower::update (
              sf::Time deltaTime,
              World & world )  [virtual]
```

Update the projectile after a frame change.

Parameters

deltaTime	Time since last frame
world	Reference to World class where the projectile is displayed

Implements Entity.

Definition at line 42 of file Tower.cpp.

```
                                                   {
  timeSinceFiring += frameDelay;
  //std::cout « "timeSinceFiring: " « timeSinceFiring.asSeconds() « "   deltaTime: " «
    deltaTime.asSeconds() « "   shootDelay: " « shootDelay.asSeconds() « std::endl;

  if (timeSinceFiring < shootDelay) {
    return;
  }

  if (timeSinceFiring >= shootDelay) {
    std::pair<float, std::shared_ptr<Enemy» closestEnemy = getClosestEnemy(world);
    if (closestEnemy.second != nullptr && inRange(closestEnemy.second->getPosition(), range)) {
      if (type == TowerType::GunCat) {
        // Shoot Bullet
        world.addProjectile(Projectile::make(textures_, Textures::Bullet, ProjectileType::Bullet, sector,
    closestEnemy, getClosestEnemies(world), getNeighbourEnemies(closestEnemy, world), 1.f, damage,
    velocity_));
        soundBullet_.play();
        timeSinceFiring = sf::Time::Zero;
      }
      else if (type == TowerType::FreezeCat) {
        // Shoot Freeze
        world.addProjectile(Projectile::make(textures_, Textures::Snowflake, ProjectileType::FreezeGun,
    sector, closestEnemy, getClosestEnemies(world), getNeighbourEnemies(closestEnemy, world), range,
    damage, velocity_));
        soundSnow_.play();
        timeSinceFiring = sf::Time::Zero;
      }
      else if (type == TowerType::BombCat) {
        // Shoot Bomb
        world.addProjectile(Projectile::make(textures_, Textures::Bomb, ProjectileType::Bomb, sector,
    closestEnemy, getClosestEnemies(world), getNeighbourEnemies(closestEnemy, world), bombRange, damage,
    velocity_));
        soundBomb_.play();
        timeSinceFiring = sf::Time::Zero;
      }
    }
    timeSinceFiring = sf::Time::Zero;
  }
}
```

14.25.3.9 upgrade()

```
bool Tower::upgrade ( )
```

Function to call when this tower is upgraded.

Returns

Was the tower able to be upgraded (is there an upgrade available)

Definition at line 167 of file Tower.cpp.

```
                   {
  if (type == TowerType::GunCat) {
    damage = damage * 2;
    velocity_ = velocity_ * 1.5;
    shootDelay = sf::seconds(1);
    entitySprite.setTexture(textures_.get(Textures::UpgradedGunCat));
    upgradeable = false;
    return true;
  }
  else if (type == TowerType::FreezeCat) {
    range = 350.f;
    velocity_ = velocity_ * 1.5;
    shootDelay = sf::seconds(1);
    entitySprite.setTexture(textures_.get(Textures::UpgradedFzeezeCat));
    upgradeable = false;
    return true;
  }
  else if (type == TowerType::BombCat) {
    bombRange = 200.f;
    shootDelay = sf::seconds(2);
    entitySprite.setTexture(textures_.get(Textures::UpgradedBombCat));
    upgradeable = false;
    return true;
  }
  else
    return false;
}
```

14.25.4 Member Data Documentation

14.25.4.1 bombRange

```
float Tower::bombRange  [private]
```

Definition at line 86 of file Tower.hpp.

14.25.4.2 damage

```
int Tower::damage  [private]
```

Definition at line 87 of file Tower.hpp.

14.25.4.3 price

```
int Tower::price  [private]
```

Definition at line 89 of file Tower.hpp.

14.25.4.4 range

```
float Tower::range  [private]
```

Definition at line 88 of file Tower.hpp.

14.25.4.5 sector

```
Sector Tower::sector  [private]
```

Definition at line 82 of file Tower.hpp.

14.25.4.6 shootDelay

```
sf::Time Tower::shootDelay  [private]
```

Definition at line 94 of file Tower.hpp.

14.25.4.7 soundBomb_

```
sf::Sound Tower::soundBomb_  [private]
```

Definition at line 92 of file Tower.hpp.

14.25.4.8 soundBullet_

```
sf::Sound Tower::soundBullet_  [private]
```

Definition at line 91 of file Tower.hpp.

14.25.4.9 sounds_

```
SoundBufferHolder& Tower::sounds_  [private]
```

Definition at line 84 of file Tower.hpp.

14.25.4.10 soundSnow_

```
sf::Sound Tower::soundSnow_  [private]
```

Definition at line 93 of file Tower.hpp.

14.25.4.11 testii

```
std::map<int, std::pair<int, int> > Tower::testii  [private]
```

Definition at line 85 of file Tower.hpp.

14.25.4.12 textures_

```
TextureHolder& Tower::textures_  [private]
```

Definition at line 83 of file Tower.hpp.

14.25.4.13 timeSinceFiring

```
sf::Time Tower::timeSinceFiring  [private]
```

Definition at line 95 of file Tower.hpp.

14.25.4.14 type

```
TowerType Tower::type
```

This tower's type.

Definition at line 73 of file Tower.hpp.

14.25.4.15 upgradeable

```
bool Tower::upgradeable  [private]
```

Definition at line 76 of file Tower.hpp.

14.25.4.16 velocity_

```
float Tower::velocity_  [private]
```

Definition at line 90 of file Tower.hpp.

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

- src/entity/Tower.hpp
- src/entity/Tower.cpp

14.26 TowerMenu Class Reference

TowerMenu class allows the player to buy and upgrade towers. Controls: Left click to select a Sector. Right click to sell a tower at selected Sector. 1 to buy tower 1 (GunCat) 2 to buy tower 2 (FreezeCat) 3 to buy tower 3 (BombCat) 4 to upgrade tower in selected Sector.

```
#include <TowerMenu.hpp>
```

Inheritance diagram for TowerMenu:

Collaboration diagram for TowerMenu:

Public Member Functions

- TowerMenu (World &world)
- virtual ~TowerMenu ()
- void handleInput (const sf::Event &event, World &world)

  handle player input
- void update (World &world, const sf::Vector2f &mousePosition)
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override

  Draw a square to show which Sector is selected.

Private Attributes

- Sector selectedSector
- sf::RectangleShape hoverIndicator
- bool showSectorIndicator
- sf::RectangleShape sectorIndicator
- sf::CircleShape rangeIndicator

14.26.1 Detailed Description

TowerMenu class allows the player to buy and upgrade towers. Controls: Left click to select a Sector. Right click to sell a tower at selected Sector. 1 to buy tower 1 (GunCat) 2 to buy tower 2 (FreezeCat) 3 to buy tower 3 (BombCat) 4 to upgrade tower in selected Sector.

Definition at line 16 of file TowerMenu.hpp.

14.26.2 Constructor & Destructor Documentation

14.26.2.1 TowerMenu()

```
TowerMenu::TowerMenu (
            World & world )
```

Definition at line 9 of file TowerMenu.cpp.

```
                                   :
       showSectorIndicator(false),
       sectorIndicator(),
       rangeIndicator() {
     sectorIndicator.setSize(Sector::DiagVector);
     sectorIndicator.setFillColor(sf::Color::Transparent);
     sectorIndicator.setOutlineColor(hoverColor);
     sectorIndicator.setOutlineThickness(3.f);
     rangeIndicator.setRadius(0);
     rangeIndicator.setFillColor(rangeIndicatorColor);
     rangeIndicator.setOutlineColor(hoverColor);
     rangeIndicator.setOutlineThickness(3.f);
 }
```

14.26.2.2 ∼TowerMenu()

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

Definition at line 19 of file TowerMenu.hpp.

```
{}
```

14.26.3 Member Function Documentation

14.26.3.1 draw()

```
void TowerMenu::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

Draw a square to show which Sector is selected.

Parameters

target	sf::RenderTarget to draw the scene to
states	sf::RenderStates object for drawing

Definition at line 81 of file TowerMenu.cpp.

```
                                                                            {
  target.draw(sectorIndicator, states);
  target.draw(rangeIndicator, states);
}
```

14.26.3.2 handleInput()

```
void TowerMenu::handleInput (
             const sf::Event & event,
             World & world )
```

handle player input

Parameters

event	Player did something
world	Current World Controls: Left click to select a Sector. Right click to sell a tower at selected Sector. 1 to buy tower 1 (GunCat) 2 to buy tower 2 (FreezeCat) 3 to buy tower 3 (BombCat) 4 to upgrade tower in selected Sector

Definition at line 23 of file TowerMenu.cpp.

```
                                                                  {
  if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button == sf::Mouse::Button::Left)
    {
    showSectorIndicator = true;
    Sector sector = Sector::fromCoords(event.mouseButton.x, event.mouseButton.y);
    if ((sector.x / Sector::Size) <= world.getMapGrid().getHeight() - 1
        && (sector.y / Sector::Size) <= world.getMapGrid().getWidth() - 1
        && world.getMapGrid().getBlockAt(sector.x / Sector::Size, sector.y / Sector::Size) == 0) {
      auto t = world.getTowerAt(sector);
      float selectedTowerRange;
      if (t != nullptr) selectedTowerRange = t->getRange(); else selectedTowerRange = 0;
      sectorIndicator.setPosition(sector.upperLeftPoint());
      rangeIndicator.setPosition(sector.upperLeftPoint().x - selectedTowerRange + ((float) Sector::Size /
    2),
                                    sector.upperLeftPoint().y - selectedTowerRange + ((float) Sector::Size /
    2));
      rangeIndicator.setRadius(selectedTowerRange);
      selectedSector = sector;
    }
  } else if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
    sf::Mouse::Button::Right) {
    Sector sector = Sector::fromCoords(event.mouseButton.x, event.mouseButton.y);
    if (world.getTowerAt(sector) != nullptr)
      world.removeTower(const_cast<Tower *>(world.getTowerAt(sector)),
                         (int) (0.8 * world.getTowerAt(sector)->getPrice()));
  } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num1) {
    if (world.getTowerAt(selectedSector) == nullptr) {
      TowerPtr
          t = std::make_unique<Tower>(world.getTextures(), world.getSounds(), Textures::GunCat,
                                      selectedSector, 2, 250.f, sf::seconds(2), 300, TowerType::GunCat);
      world.addTower(std::move(t), t->getPrice());
    }
  } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num2
      && world.getMapGrid().getBlockAt(selectedSector.x / 64, selectedSector.y / 64) == 0) {
    if (world.getTowerAt(selectedSector) == nullptr) {
      TowerPtr
          t = std::make_unique<Tower>(world.getTextures(), world.getSounds(), Textures::FreezeCat,
                                      selectedSector, 1, 250.f, sf::seconds(2), 400,
    TowerType::FreezeCat);
```

```
      world.addTower(std::move(t), t->getPrice());
    }
  } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::P) {
    world.paused = !world.paused;
  } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num3
      && world.getMapGrid().getBlockAt(selectedSector.x / 64, selectedSector.y / 64) == 0) {
    if (world.getTowerAt(selectedSector) == nullptr) {
      TowerPtr
          t = std::make_unique<Tower>(world.getTextures(), world.getSounds(), Textures::BombCat,
                                      selectedSector, 1, 250.f, sf::seconds(3), 500, TowerType::BombCat);
      world.addTower(std::move(t), t->getPrice());
    }
  } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num4
      && world.getMapGrid().getBlockAt(selectedSector.x / 64, selectedSector.y / 64) == 0) {
    if (world.getTowerAt(selectedSector) != nullptr) {
      world.upgradeTower(world.getTowerAt(selectedSector), 500);
    }
  }
}
```

14.26.3.3 update()

```
void TowerMenu::update (
            World & world,
            const sf::Vector2f & mousePosition )
```

Definition at line 77 of file TowerMenu.cpp.

```
                                                                              {

}
```

14.26.4 Member Data Documentation

14.26.4.1 hoverIndicator

```
sf::RectangleShape TowerMenu::hoverIndicator  [private]
```

Definition at line 41 of file TowerMenu.hpp.

14.26.4.2 rangeIndicator

```
sf::CircleShape TowerMenu::rangeIndicator  [private]
```

Definition at line 44 of file TowerMenu.hpp.

14.26.4.3 sectorIndicator

```
sf::RectangleShape TowerMenu::sectorIndicator  [private]
```

Definition at line 43 of file TowerMenu.hpp.

14.26.4.4 selectedSector

```
Sector TowerMenu::selectedSector [private]
```

Definition at line 40 of file TowerMenu.hpp.

14.26.4.5 showSectorIndicator

```
bool TowerMenu::showSectorIndicator [private]
```

Definition at line 42 of file TowerMenu.hpp.

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

- src/ui/TowerMenu.hpp
- src/ui/TowerMenu.cpp

14.27 TowerType Class Reference

contains possible Tower types

```
#include <Tower.hpp>
```

14.27.1 Detailed Description

contains possible Tower types

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

- src/entity/Tower.hpp

14.28 Wave Class Reference

A class for a single wave of enemies. A wave can be started by a player with a button.

```
#include <Wave.hpp>
```

Public Member Functions

- Wave (TextureHolder &textureholder, int textureID, int count, sf::Time spacing, std::map< int, std::pair< int, int >> &pathMarkers, int hitPoints, float speed)

  Constructor for a Wave.
- EnemyPtr ifNextEnemy (sf::Time deltaTime)

  if next enemy should spawn this function returns it
- bool isEmpty () const

  is the current wave done
- int getEnemiesLeft ()

  return amount of enemies left during this wave (shown in UI etc.)

Static Public Attributes

- static const sf::Time SPACING_HUGE = sf::milliseconds(1400)

    Large time difference for spawning enemies.
- static const sf::Time SPACING_WIDE = sf::milliseconds(1000)
- static const sf::Time SPACING_MEDIUM = sf::milliseconds(400)
- static const sf::Time SPACING_NARROW = sf::milliseconds(200)

    narrow time difference for spawning enemies

Private Attributes

- int enemiesLeft
- EnemyPtr enemyTemplate
- sf::Time spacing_
- sf::Time elapsed_

14.28.1 Detailed Description

A class for a single wave of enemies. A wave can be started by a player with a button.

Definition at line 16 of file Wave.hpp.

14.28.2 Constructor & Destructor Documentation

14.28.2.1 Wave()

```
Wave::Wave (
             TextureHolder & textureholder,
             int textureID,
             int count,
             sf::Time spacing,
             std::map< int, std::pair< int, int >> & pathMarkers,
             int hitPoints,
             float speed )
```

Constructor for a Wave.

Parameters

textureholder	Reference to TextureHolder object for textures
textureID	What texture to use for this wave of enemies (same for all)
count	How many enemies to spawn
spacing	How close to each other should enemies be
pathMarkers	Path for enemies
hitPoints	Enemy hitpoints (same for all)
speed	How fast enemies should be (same for all)

Definition at line 12 of file Wave.cpp.

```
                         :
      enemyTemplate(Enemy::make(textureholder, textureID, pathMarkers, hitPoints, speed)),
      enemiesLeft(count),
      elapsed_(sf::Time::Zero),
      spacing_(spacing) {
  }
```

14.28.3 Member Function Documentation

14.28.3.1 getEnemiesLeft()

```
int Wave::getEnemiesLeft ( )  [inline]
```

return amount of enemies left during this wave (shown in UI etc.)

Returns

integer, how many enemies left

Definition at line 60 of file Wave.hpp.

```
{ return enemiesLeft; }
```

14.28.3.2 ifNextEnemy()

```
EnemyPtr Wave::ifNextEnemy (
              sf::Time deltaTime )
```

if next enemy should spawn this function returns it

Parameters

deltaTime	time since last frame

Returns

Pointer to next incoming enemy, if shouldn't spawn yet return nullptr

Definition at line 25 of file Wave.cpp.

```
                                               {
    elapsed_ += deltaTime;
    if (enemiesLeft && elapsed_ >= deltaTime) {
      elapsed_ -= spacing_;
      enemiesLeft--;
      return enemyTemplate->clone(enemyTemplate->getTextureHolder(),
                                   enemyTemplate->getTextureID(),
                                   enemyTemplate->getPathMarkers(),
                                   enemyTemplate->getHitPoints(),
                                   enemyTemplate->getSpeed());
    }
    return nullptr;
  }
```

14.28.3.3 isEmpty()

```
bool Wave::isEmpty ( ) const  [inline]
```

is the current wave done

Returns

If there are no incoming enemies left, return true, else false

Definition at line 55 of file Wave.hpp.

```
55 { return enemiesLeft <= 0; }
```

14.28.4 Member Data Documentation

14.28.4.1 elapsed_

```
sf::Time Wave::elapsed_  [private]
```

Definition at line 66 of file Wave.hpp.

14.28.4.2 enemiesLeft

```
int Wave::enemiesLeft  [private]
```

Definition at line 63 of file Wave.hpp.

14.28.4.3 enemyTemplate

```
EnemyPtr Wave::enemyTemplate  [private]
```

Definition at line 64 of file Wave.hpp.

14.28.4.4 spacing_

```
sf::Time Wave::spacing_  [private]
```

Definition at line 65 of file Wave.hpp.

14.28.4.5 SPACING_HUGE

```
const sf::Time Wave::SPACING_HUGE = sf::milliseconds(1400)  [static]
```

Large time difference for spawning enemies.

Definition at line 21 of file Wave.hpp.

14.28.4.6 SPACING_MEDIUM

```
const sf::Time Wave::SPACING_MEDIUM = sf::milliseconds(400)  [static]
```

Definition at line 23 of file Wave.hpp.

14.28.4.7 SPACING_NARROW

```
const sf::Time Wave::SPACING_NARROW = sf::milliseconds(200)  [static]
```

narrow time difference for spawning enemies

Definition at line 27 of file Wave.hpp.

14.28.4.8 SPACING_WIDE

```
const sf::Time Wave::SPACING_WIDE = sf::milliseconds(1000)  [static]
```

Definition at line 22 of file Wave.hpp.

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

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

14.29 WaveController Class Reference

WaveController controls the current wave and makes a new one when the player is ready.

```
#include <WaveController.hpp>
```

Collaboration diagram for WaveController:

Public Member Functions

- WaveController (TextureHolder &textureholder, std::map< int, std::pair< int, int >> &pathMarkers)

  Constructor for WaveController.
- void update (sf::Time deltaTime, World &world)

  Checks wave status If player has ordered a new wave, starts it, or if enemies still left in current wave, spawns the next one.
- WavePtr makeNewWave (int waveNumber)

  Make a new wave to the world.
- int getWaveNumber () const

  Get current wave number.
- int getWaveEnemiesLeft ()

  Get how many enemies there are left in the current wave.

Private Attributes

- TextureHolder & textureholder_
- std::map< int, std::pair< int, int > > & pathMarkers_
- int waveNumber
- WavePtr wave

14.29.1 Detailed Description

WaveController controls the current wave and makes a new one when the player is ready.

Definition at line 14 of file WaveController.hpp.

14.29.2 Constructor & Destructor Documentation

14.29.2.1 WaveController()

```
WaveController::WaveController (
            TextureHolder & textureholder,
            std::map< int, std::pair< int, int >> & pathMarkers )  [inline]
```

Constructor for WaveController.

Parameters

textureholder	Reference to a TextureHolder instance, get enemy textures from here
pathMarkers	map that contains enemy path

Definition at line 21 of file WaveController.hpp.

```
                                                                                :
      waveNumber(0), wave(), textureholder_(textureholder), pathMarkers_(pathMarkers) {};
```

14.29.3 Member Function Documentation

14.29.3.1 getWaveEnemiesLeft()

```
int WaveController::getWaveEnemiesLeft ( )  [inline]
```

Get how many enemies there are left in the current wave.

Returns

Amount of enemies as an integer

Definition at line 45 of file WaveController.hpp.

```
45 { return wave->getEnemiesLeft(); }
```

14.29.3.2 getWaveNumber()

```
int WaveController::getWaveNumber ( ) const  [inline]
```

Get current wave number.

Returns

current wave number

Definition at line 40 of file WaveController.hpp.

```
40 { return waveNumber; }
```

14.29.3.3 makeNewWave()

```
WavePtr WaveController::makeNewWave (
             int waveNumber )
```

Make a new wave to the world.

Parameters

waveNumber	goes to a switch case, determines how hard the wave will be

Returns

Pointer to the new Wave object

Definition at line 29 of file WaveController.cpp.

```
29                                                     {
30   //Wave format: textureholder, textureID, amount of enemies, spacing of enemies, path markers to follow,
       hitpoints of enemies
31
32   switch (waveNumber) {
33     case 1:
34       return std::make_unique<Wave>(textureholder_,
35                                     Textures::BasicRat,
36                                     10,
37                                     Wave::SPACING_HUGE,
38                                     pathMarkers_,
39                                     4,
40                                     2.f);
41     case 2:
42       return std::make_unique<Wave>(textureholder_,
43                                     Textures::BasicRat,
44                                     20,
45                                     Wave::SPACING_HUGE,
46                                     pathMarkers_,
47                                     4,
48                                     2.f);
49     case 3:
50       return std::make_unique<Wave>(textureholder_,
51                                     Textures::BasicRat,
52                                     40,
53                                     Wave::SPACING_WIDE,
54                                     pathMarkers_,
55                                     4,
56                                     2.f);
57     case 4:
58       return std::make_unique<Wave>(textureholder_,
59                                     Textures::BasicRat,
60                                     40,
61                                     Wave::SPACING_MEDIUM,
62                                     pathMarkers_,
63                                     4,
64                                     2.f);
65     case 5:
66       return std::make_unique<Wave>(textureholder_,
67                                     Textures::FastRat,
68                                     30,
69                                     Wave::SPACING_WIDE,
70                                     pathMarkers_,
71                                     2,
72                                     4.f);
73     case 6:
74       return std::make_unique<Wave>(textureholder_,
75                                     Textures::FastRat,
76                                     50,
77                                     Wave::SPACING_MEDIUM,
78                                     pathMarkers_,
79                                     2,
80                                     4.f);
81     case 7:
82       return std::make_unique<Wave>(textureholder_,
83                                     Textures::FastRat,
84                                     100,
85                                     Wave::SPACING_MEDIUM,
86                                     pathMarkers_,
87                                     2,
88                                     4.f);
89     case 8:
90       return std::make_unique<Wave>(textureholder_,
91                                     Textures::FastRat,
92                                     150,
93                                     Wave::SPACING_NARROW,
94                                     pathMarkers_,
95                                     2,
96                                     4.f);
97     case 9:
98       return std::make_unique<Wave>(textureholder_,
99                                     Textures::BasicRat,
100                                     100,
101                                     Wave::SPACING_MEDIUM,
102                                     pathMarkers_,
103                                     4,
104                                     2.f);
105     case 10:
106       return std::make_unique<Wave>(textureholder_,
107                                     Textures::FatRat,
108                                     10,
109                                     Wave::SPACING_HUGE,
110                                     pathMarkers_,
111                                     30,
112                                     1.f);
113     case 11:
      return std::make_unique<Wave>(texturehodler_,
                                    Textures::FatRat,
                                    20,
                                    Wave::SPACING_WIDE,
                                    pathMarkers_,
                                    40,
                                    1.f);
    case 12:
      return std::make_unique<Wave>(textureholder_,
                                    Textures::FatRat,
                                    40,
                                    Wave::SPACING_WIDE,
                                    pathMarkers_,
                                    50,
                                    1.f);
    case 20:
      return std::make_unique<Wave>(textureholder_,
                                    Textures::FatRat,
                                    5,
                                    Wave::SPACING_HUGE,
                                    pathMarkers_,
                                    10000,
                                    4.f);
    default:
      if (waveNumber % 2)
        return std::make_unique<Wave>(textureholder_,
                                      Textures::FatRat,
                                      50,
                                      Wave::SPACING_WIDE,
                                      pathMarkers_,
                                      waveNumber * 10,
                                      1.f);
      else
        return std::make_unique<Wave>(textureholder_,
                                      Textures::FastRat,
                                      waveNumber * 20,
                                      Wave::SPACING_NARROW,
                                      pathMarkers_,
                                      waveNumber,
                                      4.f);
  }

}
```

14.29.3.4 update()

```
void WaveController::update (
              sf::Time deltaTime,
              World & world )
```

Checks wave status If player has ordered a new wave, starts it, or if enemies still left in current wave, spawns the next one.

Parameters

deltaTime	time since last update
world	world to spawn enemies in

Definition at line 10 of file WaveController.cpp.

```
10                                                             {
11   //Add an Enemy to World::enemies
12   if (!wave || (wave->isEmpty() && world.getEnemies().empty())) {
13     if (world.isReadyForNextWave) {
14       world.isReadyForNextWave = false;
15       wave = makeNewWave(++waveNumber);
16     } else {
17       wave = std::make_unique<Wave>(textureholder_, Textures::BasicRat, 0, Wave::SPACING_HUGE,
     pathMarkers_, 2, 1.0f);
18     }
19
20   }
```

```
  auto next = wave->ifNextEnemy(delayTime);

  //if not nullptr
  if (next && !world.paused) {
    world.getEnemies().push_back(std::move(next));
  }
}
```

14.29.4 Member Data Documentation

14.29.4.1 pathMarkers_

`std::map<int, std::pair<int, int> >& WaveController::pathMarkers_ [private]`

Definition at line 49 of file WaveController.hpp.

14.29.4.2 textureholder_

`TextureHolder& WaveController::textureholder_ [private]`

Definition at line 48 of file WaveController.hpp.

14.29.4.3 wave

`WavePtr WaveController::wave [private]`

Definition at line 51 of file WaveController.hpp.

14.29.4.4 waveNumber

`int WaveController::waveNumber [private]`

Definition at line 50 of file WaveController.hpp.

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

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

14.30 WavePause Class Reference

A class for pause button to pause a wave of enemies.

```
#include <WavePause.hpp>
```

Inheritance diagram for WavePause:

Collaboration diagram for WavePause:

Public Member Functions

- WavePause (const sf::Vector2f &position, FontHolder &fonts)

    The constructor for the WavePause button.
- void update (World &world, int enemiesNotSpawned)
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override
- void handleInput (const sf::Event &event, World &world)

Private Attributes

- sf::Text menuString_
- sf::CircleShape triangle_ = sf::CircleShape(40, 3)
- sf::RectangleShape square_ = sf::RectangleShape(sf::Vector2f(100, 100))
- sf::Vector2< float > mousePosition_ = sf::Vector2f(0.f, 0.f)

14.30.1 Detailed Description

A class for pause button to pause a wave of enemies.

Definition at line 13 of file WavePause.hpp.

14.30.2 Constructor & Destructor Documentation

14.30.2.1 WavePause()

```
WavePause::WavePause (
            const sf::Vector2f & position,
            FontHolder & fonts )  [explicit]
```

The constructor for the WavePause button.

Parameters

position	Button coordinates
fonts	Reference to a FontHolder

Definition at line 6 of file WavePause.cpp.

```
                                                           :
   mousePosition_(sf::Vector2f(0.f, 0.f)) {
  //mousePosition_ = sf::V
  menuString_.setFont(fonts.get(Fonts::GameTitleFont));
  menuString_.setString("Pause Game");
  menuString_.setCharacterSize(24);
  menuString_.setFillColor(sf::Color::Black);
  menuString_.setPosition(position);
  menuString_.move(0.f, -40.f);

  triangle_.setPosition(position);
  triangle_.setOutlineThickness(5);
  triangle_.setFillColor(sf::Color::Green);
  triangle_.setOutlineColor(sf::Color::Black);
  triangle_.setRotation(90.f);
  triangle_.move(80.f, 12.f);
  square_.setPosition(position);
  square_.setOutlineThickness(3);
  square_.setOutlineColor(sf::Color::Black);
}
```

14.30.3 Member Function Documentation

14.30.3.1 draw()

```
void WavePause::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [override]
```

Definition at line 30 of file WavePause.cpp.

```
                                                                    {
  target.draw(menuString_);
  target.draw(square_);
  target.draw(triangle_);
}
```

14.30.3.2 handleInput()

```
void WavePause::handleInput (
            const sf::Event & event,
            World & world )
```

Definition at line 35 of file WavePause.cpp.

```
                                                          {
  // left click
  if (event.type == sf::Event::MouseButtonPressed
      && event.mouseButton.button == sf::Mouse::Button::Left) {
    // inside square bounds
    if (square_.getGlobalBounds().contains(
        sf::Vector2f(event.mouseButton.x, event.mouseButton.y))) {
      world.paused = !world.paused;
      triangle_.setFillColor(world.paused ? sf::Color(200, 200, 200) : sf::Color::Green);
      menuString_.setString(world.paused ? "Paused" : "Pause Game");
    }
  }
}
```

14.30.3.3 update()

```
void WavePause::update (
            World & world,
            int enemiesNotSpawned )
```

Definition at line 26 of file WavePause.cpp.

```
                                                {
  //std::cout « "Is ready: " « world.isReadyForNextWave « "\n";
  triangle_.setFillColor(sf::Color::Green);
}
```

14.30.4 Member Data Documentation

14.30.4.1 menuString_

```
sf::Text WavePause::menuString_  [private]
```

Definition at line 25 of file WavePause.hpp.

14.30.4.2 mousePosition_

```
sf::Vector2<float> WavePause::mousePosition_ = sf::Vector2f(0.f, 0.f)  [private]
```

Definition at line 28 of file WavePause.hpp.

14.30.4.3 square_

```
sf::RectangleShape WavePause::square_ = sf::RectangleShape(sf::Vector2f(100, 100))  [private]
```

Definition at line 27 of file WavePause.hpp.

14.30.4.4 triangle_

```
sf::CircleShape WavePause::triangle_ = sf::CircleShape(40, 3)  [private]
```

Definition at line 26 of file WavePause.hpp.

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

- src/ui/WavePause.hpp
- src/ui/WavePause.cpp

14.31 WaveStart Class Reference

A class for the button to start next wave.

```
#include <WaveStart.hpp>
```

Inheritance diagram for WaveStart:

Collaboration diagram for WaveStart:

Public Member Functions

- WaveStart (const sf::Vector2f &position, FontHolder &fonts)

  The constructor for the WaveStart button.
- void update (World &world, int enemiesNotSpawned)
- void draw (sf::RenderTarget &target, sf::RenderStates states) const override
- void handleInput (const sf::Event &event, World &world)

Private Attributes

- sf::Text menuString_
- sf::CircleShape triangle_ = sf::CircleShape(40, 3)
- sf::RectangleShape square_ = sf::RectangleShape(sf::Vector2f(100, 100))
- sf::Vector2< float > mousePosition_ = sf::Vector2f(0.f, 0.f)

14.31.1 Detailed Description

A class for the button to start next wave.

Definition at line 13 of file WaveStart.hpp.

14.31.2 Constructor & Destructor Documentation

14.31.2.1 WaveStart()

```
WaveStart::WaveStart (
            const sf::Vector2f & position,
            FontHolder & fonts )  [explicit]
```

The constructor for the WaveStart button.

Parameters

position	Button coordinates
fonts	Reference to a FontHolder

Definition at line 6 of file WaveStart.cpp.

```
                                                                  :
   mousePosition_(sf::Vector2f(0.f, 0.f)) {
  //mousePosition_ = sf::V
  menuString_.setFont(fonts.get(Fonts::GameTitleFont));
  menuString_.setString("Start Wave");
  menuString_.setCharacterSize(24);
  menuString_.setFillColor(sf::Color::Black);
  menuString_.setPosition(position);
  menuString_.move(0.f, -40.f);

  triangle_.setPosition(position);
  triangle_.setOutlineThickness(5);
  triangle_.setFillColor(sf::Color::Green);
  triangle_.setOutlineColor(sf::Color::Black);
  triangle_.setRotation(90.f);
  triangle_.move(80.f, 12.f);
  square_.setPosition(position);
  square_.setOutlineThickness(3);
  square_.setOutlineColor(sf::Color::Black);
}
```

14.31.3 Member Function Documentation

14.31.3.1 draw()

```
void WaveStart::draw (
              sf::RenderTarget & target,
              sf::RenderStates states ) const  [override]
```

Definition at line 36 of file WaveStart.cpp.

```
                                                                          {
  target.draw(menuString_);
  target.draw(square_);
  target.draw(triangle_);
}
```

14.31.3.2 handleInput()

```
void WaveStart::handleInput (
              const sf::Event & event,
              World & world )
```

Definition at line 41 of file WaveStart.cpp.

```
                                                              {
  // left click
  if (event.type == sf::Event::MouseButtonPressed
      && event.mouseButton.button == sf::Mouse::Button::Left) {
    // inside square bounds
    if (square_.getGlobalBounds().contains(
        sf::Vector2f(event.mouseButton.x, event.mouseButton.y))) {
      world.isReadyForNextWave = true;
      menuString_.setString("Wave incoming...");
    }
  }
}
```

14.31.3.3 update()

```
void WaveStart::update (
            World & world,
            int enemiesNotSpawned )
```

Definition at line 26 of file WaveStart.cpp.

```
                                                          {
  //std::cout « "Is ready: " « world.isReadyForNextWave « "\n";
  if ((world.getEnemies().size() + enemiesNotSpawned) > 0) {
    world.isReadyForNextWave = false;
    triangle_.setFillColor(sf::Color(200, 200, 200));
  } else {
    triangle_.setFillColor(sf::Color::Green);
    menuString_.setString("Start wave");
  }
}
```

14.31.4 Member Data Documentation

14.31.4.1 menuString_

```
sf::Text WaveStart::menuString_  [private]
```

Definition at line 25 of file WaveStart.hpp.

14.31.4.2 mousePosition_

```
sf::Vector2<float> WaveStart::mousePosition_ = sf::Vector2f(0.f, 0.f)  [private]
```

Definition at line 28 of file WaveStart.hpp.

14.31.4.3 square_

```
sf::RectangleShape WaveStart::square_ = sf::RectangleShape(sf::Vector2f(100, 100))  [private]
```

Definition at line 27 of file WaveStart.hpp.

14.31.4.4 triangle_

```
sf::CircleShape WaveStart::triangle_ = sf::CircleShape(40, 3)  [private]
```

Definition at line 26 of file WaveStart.hpp.

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

- src/ui/WaveStart.hpp
- src/ui/WaveStart.cpp

14.32 World Class Reference

The world class houses all of the things in a game level. Towers, enemies, map, map grid, etc. All those elements are used here to run and update the game.

```
#include <World.hpp>
```

Inheritance diagram for World:

Collaboration diagram for World:

Public Member Functions

- World (TextureHolder &textureholder, SoundBufferHolder &soundBufferHolder, int mapNum, std::map< int, std::pair< int, int >> &pathMarkers)

    Constructor for world.
- void update (sf::Time delta)

    Update all enemies, towers and projectiles.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const

    call other classes' draw functions. Map, towers, enemies, house, and projectiles.
- void addProjectile (ProjectilePtr &&projectile)

    Add a projectile to the world (a tower shot one).
- bool addTower (TowerPtr &&tower, int price)

    Add a tower to the world (when player buys one)
- bool upgradeTower (Tower ∗tower, int price)

    Upgrade a tower.
- void removeTower (Tower ∗tower, int price)

    Find a tower and remove it.
- Enemies & getEnemies ()

    Get the enemies currently on the map.
- int getMoney () const

    Get the amount of money the player has.
- int getHP () const

    Get the player hp.
- const MapGrid & getMapGrid () const

    Get the current map's MapGrid.
- Tower ∗ getTowerAt (const Sector &target) const

    Find a tower at given coordinates.
- TextureHolder & getTextures ()

    Get world TextureHolder.
- SoundBufferHolder & getSounds ()

    Get world SoundBufferHolder.

Public Attributes

- bool isReadyForNextWave = false

    Is the player ready for next wave (pressed the button)?
- bool paused = false

    Is the game paused? If so, dont move anything.

Private Member Functions

- template<typename T >
  void update (const std::vector< std::unique_ptr< T >> &vec, sf::Time delta)
- template<typename T >
  void update (const std::vector< std::shared_ptr< T >> &vec, sf::Time delta)
- template<typename T >
  void draw (const std::vector< std::unique_ptr< T >> &vec, sf::RenderTarget &target, const sf::RenderStates &states) const
- template<typename T >
  void draw (const std::vector< std::shared_ptr< T >> &vec, sf::RenderTarget &target, const sf::RenderStates &states) const
- template<typename T >
  void clean (std::vector< std::unique_ptr< T >> &vec)
- template<typename T >
  void clean (std::vector< std::shared_ptr< T >> &vec)

Private Attributes

- TextureHolder & textures
- sf::Sprite endHouse
- SoundBufferHolder & sounds
- sf::Sound deathSound_
- sf::Sound explosionSound_
- MapGrid grid_
- int money_
- int hp_
- int mapNum_
- Enemies enemies
- Towers towers
- Projectiles projectiles
- ProjectilePtr projectile
- Grid towerGrid_

14.32.1 Detailed Description

The world class houses all of the things in a game level. Towers, enemies, map, map grid, etc. All those elements are used here to run and update the game.

Definition at line 26 of file World.hpp.

14.32.2 Constructor & Destructor Documentation

14.32.2.1 World()

```
World::World (
            TextureHolder & textureholder,
            SoundBufferHolder & soundBufferHolder,
            int mapNum,
            std::map< int, std::pair< int, int >> & pathMarkers )
```

Constructor for world.

Parameters

textureholder	Reference to a TextureHolder instance, mostly passed along to other classes
soundBufferHolder	Reference to a SoundBufferHolder instance, mostly passed along to other classes
mapNum	which map is used here (int)
pathMarkers	map that contains enemy path

Definition at line 12 of file World.cpp.

```
                                                          :
  grid_(textureholder, mapNum, pathMarkers),
  money_(startMoney),
  hp_(startHP),
  textures(textureholder),
  sounds(soundBufferHolder),
  mapNum_(mapNum),
  towerGrid_() {
deathSound_.setBuffer(soundBufferHolder.get(SoundBuffers::EnemyDeath));
explosionSound_.setBuffer(soundBufferHolder.get(SoundBuffers::Explosion));
endHouse.setTexture(textures.get(Textures::HouseTile));
// x coords
float x = float(pathMarkers.rbegin()->second.first) * tileSize;
// y coords
float y = float(pathMarkers.rbegin()->second.second) * tileSize;
endHouse.setPosition(x, y);
endHouse.setScale(2.f, 2.f);
endHouse.move(-55.f, -64.f);
}
```

14.32.3 Member Function Documentation

14.32.3.1 addProjectile()

```
void World::addProjectile (
            ProjectilePtr && projectile )
```

Add a projectile to the world (a tower shot one).

Parameters

projectile	ProjectilePtr, pointer to the projectile to add

Returns

Definition at line 79 of file World.cpp.

```
                                                          {
projectiles.push_back(std::move(projectile));
}
```

14.32.3.2 addTower()

```
bool World::addTower (
            TowerPtr && tower,
            int price )
```

Add a tower to the world (when player buys one)

Parameters

tower	A TowerPtr, pointer to the tower.
price	The amount of money to deduct from player's balance

Definition at line 83 of file World.cpp.

```
                                                  {
  if (money_ >= price) {
    money_ -= price;
    //std::cout « "Money left: " « money_ « std::endl;
    //std::cout « "HP left: " « hp_ « std::endl;
    towers.push_back(std::move(tower));
    return true;
  } else
    return false;
}
```

14.32.3.3 clean() [1/2]

```
template<typename T >
void World::clean (
            std::vector< std::shared_ptr< T >> & vec )  [inline], [private]
```

Definition at line 205 of file World.hpp.

```
                                                          {
  for (auto elem = vec.begin(); elem != vec.end();) {
    if ((*elem)->ifShouldRemove()) {
      elem = vec.erase(elem);
    } else {
      ++elem;
    }
  }
}
```

14.32.3.4 clean() [2/2]

```
template<typename T >
void World::clean (
            std::vector< std::unique_ptr< T >> & vec )  [inline], [private]
```

Definition at line 192 of file World.hpp.

```
                                                          {
  if (!vec.empty()) {
    vec.erase(std::remove_if(vec.begin(),
                             vec.end(),
                             [](const std::unique_ptr<T> &entity) {
                               if (entity != nullptr) {
                                 return entity->ifShouldRemove();
                               } else return false;
                             }), vec.end());
  }
}
```

14.32.3.5 draw() [1/3]

```
template<typename T >
void World::draw (
            const std::vector< std::shared_ptr< T >> & vec,
            sf::RenderTarget & target,
            const sf::RenderStates & states ) const  [inline], [private]
```

Definition at line 183 of file World.hpp.

```
                                                                 {
  for (auto &&entity : vec) {
    target.draw(*entity, states);
  }
}
```

14.32.3.6 draw() [2/3]

```
template<typename T >
void World::draw (
            const std::vector< std::unique_ptr< T >> & vec,
            sf::RenderTarget & target,
            const sf::RenderStates & states ) const  [inline], [private]
```

Definition at line 175 of file World.hpp.

```
                                                                 {
  for (auto &&entity : vec) {
    target.draw(*entity, states);
  }
}
```

14.32.3.7 draw() [3/3]

```
void World::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const
```

call other classes' draw functions. Map, towers, enemies, house, and projectiles.

Parameters

target	sf::RenderTarget to draw the scene to
states	sf::RenderStates object for drawing

Definition at line 71 of file World.cpp.

```
                                                                      {
  target.draw(grid_); //draw map
  draw(towers, target, states); //draw towers
  draw(enemies, target, states); // draw enemies
  target.draw(endHouse);
  draw(projectiles, target, states); // draw projectiles
}
```

14.32.3.8 getEnemies()

```
Enemies& World::getEnemies ( ) [inline]
```

Get the enemies currently on the map.

Returns

A vector of enemies.

Definition at line 81 of file World.hpp.

```
81 { return enemies; }
```

14.32.3.9 getHP()

```
int World::getHP ( ) const [inline]
```

Get the player hp.

Returns

integer, hp

Definition at line 91 of file World.hpp.

```
91 { return hp_; }
```

14.32.3.10 getMapGrid()

```
const MapGrid& World::getMapGrid ( ) const [inline]
```

Get the current map's MapGrid.

Returns

the map's MapGrid class instance

Definition at line 96 of file World.hpp.

```
96 { return grid_; }
```

14.32.3.11 getMoney()

```
int World::getMoney ( ) const  [inline]
```

Get the amount of money the player has.

Returns

integer, amount of money

Definition at line 86 of file World.hpp.

```
86 { return money_; }
```

14.32.3.12 getSounds()

```
SoundBufferHolder& World::getSounds ( )  [inline]
```

Get world SoundBufferHolder.

Returns

reference to a SoundBufferHolder

Definition at line 112 of file World.hpp.

```
112 { return sounds; }
```

14.32.3.13 getTextures()

```
TextureHolder& World::getTextures ( )  [inline]
```

Get world TextureHolder.

Returns

reference to a TextureHolder

Definition at line 107 of file World.hpp.

```
107 { return textures; }
```

14.32.3.14 getTowerAt()

```
Tower * World::getTowerAt (
             const Sector & target ) const
```

Find a tower at given coordinates.

Parameters

target	Target coordinates (using Sector class)

Returns

Pointer to the tower

Definition at line 107 of file World.cpp.

```
                                                       {
  auto found = std::find_if(
      towers.cbegin(),
      towers.cend(),
      [&target](const TowerPtr &tower) { return tower->getSector() == target; }
  );
  return found == towers.cend() ? nullptr : &**found;
}
```

14.32.3.15 removeTower()

```
void World::removeTower (
              Tower * tower,
              int price )
```

Find a tower and remove it.

Parameters

tower	A pointer to a tower that should be removed
price	how much money is returned to the player

Definition at line 94 of file World.cpp.

```
                                                 {
  towers.erase(
      std::find_if(
          towers.begin(),
          towers.end(),
          [tower](const TowerPtr &other) { return other.get() == tower; }
      )
  );
  money_ += price;
  //std::cout « "Money left: " « money_ « std::endl;
  //std::cout « "HP left: " « hp_ « std::endl;
}
```

14.32.3.16 update() [1/3]

```
template<typename T >
void World::update (
              const std::vector< std::shared_ptr< T >> & vec,
              sf::Time delta )  [inline], [private]
```

Definition at line 166 of file World.hpp.

```
                                                                                  {
```

```
  for (auto &&entity : vec) {
    if (entity != nullptr) {
      entity->update(deltaTime, *this);
    }
  }
}
```

14.32.3.17 update() [2/3]

```
template<typename T >
void World::update (
            const std::vector< std::unique_ptr< T >> & vec,
            sf::Time delta )  [inline], [private]
```

Definition at line 157 of file World.hpp.

```
                                                                                {
  for (auto &&entity : vec) {
    if (entity != nullptr) {
      entity->update(deltaTime, *this);
    }
  }
}
```

14.32.3.18 update() [3/3]

```
void World::update (
            sf::Time delta )
```

Update all enemies, towers and projectiles.

Parameters

delta	time since last frame Updates positions of everything. If game is paused, stops enemies, towers and projectiles. Checks for dead enemies and those that have reached the cats' house.

Definition at line 35 of file World.cpp.

```
                                    {
  if (!paused) {
    //Update all enemies
    update(enemies, deltaTime);
    //Update all towers
    update(towers, deltaTime);
    //Update all projectiles
    update(projectiles, deltaTime);
  }
  //Check which enemies are dead or at finish -> changes money amount
  for (auto &&enemy : enemies) {
    if (!enemy->isAlive()) {
      deathSound_.play();
      this->money_ += enemy->getValue() * 5;
    }
    if (enemy->isAtFinish()) {
      this->hp_ -= enemy->getValue();
    }
  }

  //Clean dead enemies from enemies vector
  clean(enemies);

  //Clean dead projectiles from projectiles vector
  for (auto &&projectile : projectiles) {
    if (projectile->type_ == ProjectileType::Bomb) {
      if (projectile->ifShouldRemove()) {
        explosionSound_.play();
      }
    }
  }
  clean(projectiles);

  // std::cout « "Enemies: " « enemies.size() « "   Towers: " « towers.size() « "   Projectiles: " «
    projectiles.size() « std::endl;
}
```

14.32.3.19 upgradeTower()

```
bool World::upgradeTower (
            Tower * tower,
            int price )
```

Upgrade a tower.

Parameters

tower	Tower to upgrade, calls its function for upgrading
price	How much the upgrade costs

Returns

Whether the upgrade succeeded (true if there was an upgrade available and the player had enough money)

Definition at line 115 of file World.cpp.

```
                                                    {
  if (money_ >= price) {
    if (tower->upgrade()) money_ -= price;
    return true;
  } else return false;
}
```

14.32.4 Member Data Documentation

14.32.4.1 deathSound_

```
sf::Sound World::deathSound_  [private]
```

Definition at line 125 of file World.hpp.

14.32.4.2 endHouse

```
sf::Sprite World::endHouse  [private]
```

Definition at line 123 of file World.hpp.

14.32.4.3 enemies

```
Enemies World::enemies  [private]
```

Definition at line 131 of file World.hpp.

14.32.4.4 explosionSound_

```
sf::Sound World::explosionSound_  [private]
```

Definition at line 126 of file World.hpp.

14.32.4.5 grid_

```
MapGrid World::grid_  [private]
```

Definition at line 127 of file World.hpp.

14.32.4.6 hp_

```
int World::hp_  [private]
```

Definition at line 129 of file World.hpp.

14.32.4.7 isReadyForNextWave

```
bool World::isReadyForNextWave = false
```

Is the player ready for next wave (pressed the button)?

Definition at line 116 of file World.hpp.

14.32.4.8 mapNum_

```
int World::mapNum_  [private]
```

Definition at line 130 of file World.hpp.

14.32.4.9 money_

```
int World::money_ [private]
```

Definition at line 128 of file World.hpp.

14.32.4.10 paused

```
bool World::paused = false
```

Is the game paused? If so, dont move anything.

Definition at line 120 of file World.hpp.

14.32.4.11 projectile

```
ProjectilePtr World::projectile [private]
```

Definition at line 134 of file World.hpp.

14.32.4.12 projectiles

```
Projectiles World::projectiles [private]
```

Definition at line 133 of file World.hpp.

14.32.4.13 sounds

```
SoundBufferHolder& World::sounds [private]
```

Definition at line 124 of file World.hpp.

14.32.4.14 textures

```
TextureHolder& World::textures [private]
```

Definition at line 122 of file World.hpp.

14.32.4.15 towerGrid_

```
Grid World::towerGrid_  [private]
```

Definition at line 135 of file World.hpp.

14.32.4.16 towers

```
Towers World::towers  [private]
```

Definition at line 132 of file World.hpp.

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

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

Chapter 15

File Documentation

15.1 doc/readme.md File Reference

15.2 libs/readme.md File Reference

15.3 plan/readme.md File Reference

15.4 src/readme.md File Reference

15.5 src/ui/readme.md File Reference

15.6 tests/readme.md File Reference

15.7 Meeting-notes.md File Reference

15.8 README.md File Reference

15.9 src/entity/Enemy.cpp File Reference

```
#include <cmath>
#include <memory>
#include "Enemy.hpp"
#include "SFML/System/Time.hpp"
```

Include dependency graph for Enemy.cpp:

15.10 src/entity/Enemy.hpp File Reference

```
#include "Entity.hpp"
#include "SFML/System/Time.hpp"
#include <iostream>
```

Include dependency graph for Enemy.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Enemy

  A class for ingame enemies. Derived from Entity class.

Typedefs

- using EnemyPtr = std::shared_ptr< Enemy >
- using Enemies = std::vector< EnemyPtr >

15.10.1 Typedef Documentation

15.10.1.1 Enemies

```
using Enemies = std::vector<EnemyPtr>
```

Definition at line 14 of file Enemy.hpp.

15.10.1.2 EnemyPtr

```
using EnemyPtr = std::shared_ptr<Enemy>
```

Definition at line 13 of file Enemy.hpp.

15.11 src/entity/Entity.cpp File Reference

```
#include "Entity.hpp"
#include <SFML/Graphics/RenderTarget.hpp>
```

Include dependency graph for Entity.cpp:

15.12 src/entity/Entity.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include <SFML/Graphics.hpp>
#include "SFML/Graphics/Sprite.hpp"
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include "resource/Resource.hpp"
#include <vector>
```

Include dependency graph for Entity.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Entity

  Visible entity on the map.

15.13 src/entity/Projectile.cpp File Reference

```
#include "Projectile.hpp"
#include <utility>
```

Include dependency graph for Projectile.cpp:

15.14 src/entity/Projectile.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include "ui/Sector.hpp"
#include "Entity.hpp"
#include "Enemy.hpp"
#include <vector>
#include <memory>
```

Include dependency graph for Projectile.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Projectile

  Extends Entity class, Bombs, Bullets etc.

Typedefs

- using ProjectilePtr = std::unique_ptr< Projectile >
- using Projectiles = std::vector< ProjectilePtr >

Enumerations

- enum class ProjectileType { Bullet , FreezeGun , Bomb }

15.14.1 Typedef Documentation

15.14.1.1 ProjectilePtr

```
using ProjectilePtr = std::unique_ptr<Projectile>
```

Definition at line 25 of file Projectile.hpp.

15.14.1.2 Projectiles

```
using Projectiles = std::vector<ProjectilePtr>
```

Definition at line 26 of file Projectile.hpp.

15.14.2 Enumeration Type Documentation

15.14.2.1 ProjectileType

```
enum ProjectileType  [strong]
```

Enumerator

Bullet	
FreezeGun	
Bomb	

Definition at line 18 of file Projectile.hpp.

```
                            {
    Bullet,
    FreezeGun,
    Bomb
};
```

15.15 src/entity/Rat.cpp File Reference

15.16 src/entity/Rat.hpp File Reference

15.17 src/entity/Tower.cpp File Reference

```
#include "Tower.hpp"
#include <SFML/Graphics/RenderTarget.hpp>
```

```
#include "game/World.hpp"
#include "SFML/System/Time.hpp"
#include "SFML/Audio/Sound.hpp"
```
Include dependency graph for Tower.cpp:

Variables

- sf::Time frameDelay = sf::seconds(0.0167)

15.17.1 Variable Documentation

15.17.1.1 frameDelay

```
sf::Time frameDelay = sf::seconds(0.0167)
```

Definition at line 7 of file Tower.cpp.

15.18 src/entity/Tower.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include "ui/Sector.hpp"
#include "Entity.hpp"
#include "Enemy.hpp"
#include "Projectile.hpp"
#include <vector>
#include <memory>
#include "SFML/Audio/Sound.hpp"
```
Include dependency graph for Tower.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Tower

  Tower to display on the map and shoot, extends Entity class.

Typedefs

- using TowerPtr = std::unique_ptr< Tower >
- using Towers = std::vector< TowerPtr >

Enumerations

- enum class TowerType { GunCat , FreezeCat , BombCat }

15.18.1 Typedef Documentation

15.18.1.1 TowerPtr

```
using TowerPtr = std::unique_ptr<Tower>
```

Definition at line 27 of file Tower.hpp.

15.18.1.2 Towers

```
using Towers = std::vector<TowerPtr>
```

Definition at line 28 of file Tower.hpp.

15.18.2 Enumeration Type Documentation

15.18.2.1 TowerType

```
enum TowerType  [strong]
```

Enumerator

GunCat	
FreezeCat	
BombCat	

Definition at line 20 of file Tower.hpp.

```
                    {
    GunCat,
    FreezeCat,
    BombCat
};
```

15.19 src/game/Block.cpp File Reference

```
#include "Block.hpp"
```

Include dependency graph for Block.cpp:

15.20 src/game/Block.hpp File Reference

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

Classes

- class Block

15.21 src/game/Game.cpp File Reference

```
#include "Game.hpp"
#include <SFML/Graphics.hpp>
#include "MapGrid.hpp"
#include "World.hpp"
#include "MapScene.hpp"
#include "LevelSelect.hpp"
#include "GameTitle.hpp"
#include "GameEnd.hpp"
#include "resource/Resource.hpp"
#include <iostream>
```

Include dependency graph for Game.cpp:

15.22 src/game/Game.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "Scene.hpp"
#include "resource/Resource.hpp"
```

Include dependency graph for Game.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Game

  Class where the game, setting are set and loading and rendering is called from.

15.23 src/game/GameEnd.cpp File Reference

```
#include <iostream>
#include "GameEnd.hpp"
```

Include dependency graph for GameEnd.cpp:

15.24 src/game/GameEnd.hpp File Reference

```
#include "Scene.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/Sprite.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
```

Include dependency graph for GameEnd.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class GameEnd

  A class that inherits Scene, is shown when the player loses the game.

15.25 src/game/GameTitle.cpp File Reference

```
#include <iostream>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include "GameTitle.hpp"
```

Include dependency graph for GameTitle.cpp:

15.26 src/game/GameTitle.hpp File Reference

```
#include "Scene.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
```

Include dependency graph for GameTitle.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class GameTitle

  Welcome screen for the game.

15.27 src/game/LevelSelect.cpp File Reference

```
#include <iostream>
#include "LevelSelect.hpp"
```

Include dependency graph for LevelSelect.cpp:

15.28 src/game/LevelSelect.hpp File Reference

```
#include "Scene.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/Sprite.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
#include <fstream>
#include <iostream>
```

Include dependency graph for LevelSelect.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class LevelSelect

  A class for level selection menu, inherits Scene class.

15.29 src/game/Map.cpp File Reference

```
#include "Map.hpp"
```

Include dependency graph for Map.cpp:

15.30 src/game/Map.hpp File Reference

```
#include <string>
```

Include dependency graph for Map.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Map

15.31 src/game/MapGrid.cpp File Reference

```
#include "MapGrid.hpp"
#include "SFML/Graphics/Sprite.hpp"
#include "SFML/Graphics/Texture.hpp"
#include "SFML/Graphics/CircleShape.hpp"
#include <fstream>
#include <iostream>
```

Include dependency graph for MapGrid.cpp:

15.32 src/game/MapGrid.hpp File Reference

```
#include <vector>
#include "SFML/Graphics/RenderTarget.hpp"
#include "SFML/Graphics/Drawable.hpp"
#include "resource/Resource.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
#include "SFML/Graphics/Sprite.hpp"
```

Include dependency graph for MapGrid.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class MapGrid

  The game map consists of blocks, this class handles reading maps from file and rendering the grid.

15.33 src/game/MapScene.cpp File Reference

```
#include "MapScene.hpp"
#include <utility>
```

Include dependency graph for MapScene.cpp:

15.34 src/game/MapScene.hpp File Reference

```
#include "Scene.hpp"
#include "World.hpp"
#include "WaveController.hpp"
#include "ui/TowerMenu.hpp"
#include "ui/GameStatusMenu.hpp"
#include "ui/WaveStart.hpp"
#include "ui/WavePause.hpp"
#include "ui/GameCommandsMenu.hpp"
```

Include dependency graph for MapScene.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class MapScene

  A class used when ingame. Inherits from Scene class. The main scene of the game.

15.35 src/game/Scene.cpp File Reference

```
#include "Scene.hpp"
```

Include dependency graph for Scene.cpp:

15.36 src/game/Scene.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include <SFML/Window/Event.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include "resource/Resource.hpp"
```

Include dependency graph for Scene.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct request
- class Scene

  This is a class for different UI "pages" of the game such as main menu or the game itself.

Namespaces

- Scenes

Typedefs

- typedef request sceneRequest
- using ScenePtr = std::unique_ptr< Scene >

Enumerations

- enum class Scenes::ID { Scenes::GameTitle , Scenes::LevelSelect , Scenes::MapScene , Scenes::GameEnd }

15.36.1 Typedef Documentation

15.36.1.1 ScenePtr

```
using ScenePtr = std::unique_ptr<Scene>
```

Definition at line 59 of file Scene.hpp.

15.36.1.2 sceneRequest

```
typedef request sceneRequest
```

Definition at line 29 of file Scene.hpp.

15.37 src/game/Wave.cpp File Reference

```
#include "Wave.hpp"
```
Include dependency graph for Wave.cpp:

15.38 src/game/Wave.hpp File Reference

```
#include "entity/Enemy.hpp"
#include <SFML/System/Time.hpp>
#include <iostream>
```
Include dependency graph for Wave.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Wave

    A class for a single wave of enemies. A wave can be started by a player with a button.

Typedefs

- using WavePtr = std::unique_ptr< Wave >

15.38.1 Typedef Documentation

15.38.1.1 WavePtr

```
using WavePtr = std::unique_ptr<Wave>
```

Definition at line 11 of file Wave.hpp.

15.39 src/game/WaveController.cpp File Reference

```
#include "WaveController.hpp"
#include <iostream>
```

Include dependency graph for WaveController.cpp:

Variables

- sf::Time delayTime = sf::seconds(0.0167)

15.39.1 Variable Documentation

15.39.1.1 delayTime

```
sf::Time delayTime = sf::seconds(0.0167)
```

Definition at line 8 of file WaveController.cpp.

15.40 src/game/WaveController.hpp File Reference

```
#include "World.hpp"
#include "Wave.hpp"
```

Include dependency graph for WaveController.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class WaveController

  WaveController controls the current wave and makes a new one when the player is ready.

15.41 src/game/World.cpp File Reference

```
#include "World.hpp"
#include "SFML/Graphics/RectangleShape.hpp"
#include "SFML/Graphics/CircleShape.hpp"
```

Include dependency graph for World.cpp:

15.42 src/game/World.hpp File Reference

```
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/System/Time.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <algorithm>
#include "ui/Sector.hpp"
#include "MapGrid.hpp"
#include "resource/Resource.hpp"
#include "entity/Enemy.hpp"
#include "entity/Tower.hpp"
#include "entity/Projectile.hpp"
#include "ui/Grid.hpp"
#include "SFML/Audio/Sound.hpp"
```

Include dependency graph for World.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class World

  The world class houses all of the things in a game level. Towers, enemies, map, map grid, etc. All those elements are used here to run and update the game.

15.43 src/main.cpp File Reference

```
#include <iostream>
#include "game/Game.hpp"
```

Include dependency graph for main.cpp:

Functions

- int main ()

15.43.1 Function Documentation

15.43.1.1 main()

```
int main ( )
```

Definition at line 4 of file main.cpp.

```
            {
  Game newGame; //create game object
  newGame.run(); //run the game
  return 0;
}
```

15.44 src/resource/Resource.cpp File Reference

```
#include "Resource.hpp"
```

Include dependency graph for Resource.cpp:

15.45 src/resource/Resource.hpp File Reference

```
#include "string"
#include <SFML/System/Time.hpp>
#include "map"
#include <stdexcept>
#include <memory>
#include <cassert>
#include "SFML/Graphics/Texture.hpp"
#include "SFML/Graphics/Font.hpp"
#include "SFML/Audio/SoundBuffer.hpp"
```

Include dependency graph for Resource.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ResourceHolder< Resource, Identifier >

Namespaces

- Textures
- Maps
- Fonts
- SoundBuffers

Typedefs

- typedef ResourceHolder< sf::Texture, Textures::ID > TextureHolder
- typedef ResourceHolder< std::string, Maps::ID > MapHolder
- typedef ResourceHolder< sf::Font, Fonts::ID > FontHolder
- typedef ResourceHolder< sf::SoundBuffer, SoundBuffers::ID > SoundBufferHolder

Enumerations

- enum Textures::ID {
  Textures::PathTile , Textures::GrassTile , Textures::HouseTile , Textures::GunCat ,
  Textures::UpgradedGunCat , Textures::FreezeCat , Textures::UpgradedFzeezeCat , Textures::BombCat ,
  Textures::UpgradedBombCat , Textures::FatRat , Textures::FastRat , Textures::BasicRat ,
  Textures::Bullet , Textures::Bomb , Textures::Snowflake , Textures::PlayButton ,
  Textures::Explosion }
- enum Maps::ID { Maps::Map }
- enum Fonts::ID { Fonts::GameTitleFont }
- enum SoundBuffers::ID {
  SoundBuffers::EnemyDeath , SoundBuffers::Explosion , SoundBuffers::GunCat , SoundBuffers::BombCatMeow
  ,
  SoundBuffers::FreezeCatMeow }

15.45.1 Typedef Documentation

15.45.1.1 FontHolder

```
typedef ResourceHolder<sf::Font, Fonts::ID> FontHolder
```

Definition at line 106 of file Resource.hpp.

15.45.1.2 MapHolder

```
typedef ResourceHolder<std::string, Maps::ID> MapHolder
```

Definition at line 105 of file Resource.hpp.

15.45.1.3 SoundBufferHolder

```
typedef ResourceHolder<sf::SoundBuffer, SoundBuffers::ID> SoundBufferHolder
```

Definition at line 107 of file Resource.hpp.

15.45.1.4 TextureHolder

```
typedef ResourceHolder<sf::Texture, Textures::ID> TextureHolder
```

Definition at line 104 of file Resource.hpp.

15.46 src/sceneItem/SceneItem.cpp File Reference

```
#include "SceneItem.hpp"
```
Include dependency graph for SceneItem.cpp:

15.47 src/sceneItem/SceneItem.hpp File Reference

```
#include <SFML/Graphics.hpp>
```
Include dependency graph for SceneItem.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class SceneItem

15.48 src/ui/Button.cpp File Reference

```
#include "Button.hpp"
#include <SFML/Graphics/RenderTarget.hpp>
```
Include dependency graph for Button.cpp:

15.49 src/ui/Button.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Window/Event.hpp>
#include <vector>
#include <memory>
#include <SFML/Graphics/CircleShape.hpp>
```
Include dependency graph for Button.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Button

Typedefs

- using ButtonPtr = std::unique_ptr< Button >
- using Buttons = std::vector< ButtonPtr >

15.49.1 Typedef Documentation

15.49.1.1 ButtonPtr

```
using ButtonPtr = std::unique_ptr<Button>
```

Definition at line 29 of file Button.hpp.

15.49.1.2 Buttons

```
using Buttons = std::vector<ButtonPtr>
```

Definition at line 30 of file Button.hpp.

15.50 src/ui/GameCommandsMenu.cpp File Reference

```
#include "GameCommandsMenu.hpp"
```

Include dependency graph for GameCommandsMenu.cpp:

15.51 src/ui/GameCommandsMenu.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "resource/Resource.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/Texture.hpp"
#include <SFML/Graphics/RenderStates.hpp>
#include "game/World.hpp"
```

Include dependency graph for GameCommandsMenu.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class GameCommandsMenu

  Shows the player how to play the game (at the right side of the screen in a MapScene)

15.52 src/ui/GameStatusMenu.cpp File Reference

```
#include "GameStatusMenu.hpp"
#include <sstream>
```

Include dependency graph for GameStatusMenu.cpp:

15.53 src/ui/GameStatusMenu.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "game/World.hpp"
```

Include dependency graph for GameStatusMenu.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class GameStatusMenu

  Shows the player information from the game world: hp, money, enemies left and wave number.

15.54 src/ui/Grid.cpp File Reference

```
#include "Grid.hpp"
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
```
Include dependency graph for Grid.cpp:

15.55 src/ui/Grid.hpp File Reference

```
#include "Sector.hpp"
#include <vector>
#include <SFML/Graphics/Drawable.hpp>
#include "entity/Tower.hpp"
```
Include dependency graph for Grid.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Grid

  Grid is used to place the towers according to the visual tiles.

15.56 src/ui/Price.cpp File Reference

```
#include "Price.hpp"
```
Include dependency graph for Price.cpp:

15.57 src/ui/Price.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Text.hpp>
```
Include dependency graph for Price.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Price

15.58 src/ui/Sector.cpp File Reference

```
#include "Sector.hpp"
```
Include dependency graph for Sector.cpp:

Functions

- Sector operator+ (const Sector &lhs, const Sector &rhs)

15.58.1 Function Documentation

15.58.1.1 operator+()

```
Sector operator+ (
            const Sector & lhs,
            const Sector & rhs )
```

Definition at line 7 of file Sector.cpp.

```
{
  return Sector{lhs.x + rhs.x, lhs.y + rhs.y};
}
```

15.59 src/ui/Sector.hpp File Reference

```
#include <SFML/System/Vector2.hpp>
#include <SFML/Graphics/Rect.hpp>
#include <SFML/System/Time.hpp>
#include <vector>
#include <cmath>
```

Include dependency graph for Sector.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct Sector

  A Sector is a 64x64 pixel block in the game map The Sector class is used to align towers properly.

Typedefs

- using Path = std::vector< Sector >

Functions

- Sector operator+ (const Sector &lhs, const Sector &rhs)

15.59.1 Typedef Documentation

15.59.1.1 Path

```
using Path = std::vector<Sector>
```

Definition at line 27 of file Sector.hpp.

15.59.2 Function Documentation

15.59.2.1 operator+()

```
Sector operator+ (
            const Sector & lhs,
            const Sector & rhs )
```

Definition at line 7 of file Sector.cpp.

```
                                                              {
  return Sector{lhs.x + rhs.x, lhs.y + rhs.y};
}
```

15.60 src/ui/SelectTowerButton.hpp File Reference

```
#include "Button.hpp"
#include <SFML/Graphics/CircleShape.hpp>
#include <SFML/Graphics/Text.hpp>
#include <memory>
#include "game/World.hpp"
#include "Sector.hpp"
```

Include dependency graph for SelectTowerButton.hpp:

Classes

- class SelectTowerButton< T >

15.61 src/ui/TowerMenu.cpp File Reference

```
#include "TowerMenu.hpp"
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Window/Event.hpp>
#include "game/World.hpp"
```

Include dependency graph for TowerMenu.cpp:

15.62 src/ui/TowerMenu.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "Sector.hpp"
#include <SFML/Graphics/RectangleShape.hpp>
#include "entity/Tower.hpp"
```

Include dependency graph for TowerMenu.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class TowerMenu

  TowerMenu class allows the player to buy and upgrade towers. Controls: Left click to select a Sector. Right click to sell a tower at selected Sector. 1 to buy tower 1 (GunCat) 2 to buy tower 2 (FreezeCat) 3 to buy tower 3 (BombCat) 4 to upgrade tower in selected Sector.

15.63 src/ui/WavePause.cpp File Reference

```
#include "WavePause.hpp"
```

Include dependency graph for WavePause.cpp:

15.64 src/ui/WavePause.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "game/World.hpp"
```

Include dependency graph for WavePause.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class WavePause

  A class for pause button to pause a wave of enemies.

15.65 src/ui/WaveStart.cpp File Reference

```
#include "WaveStart.hpp"
```

Include dependency graph for WaveStart.cpp:

15.66 src/ui/WaveStart.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "game/World.hpp"
```

Include dependency graph for WaveStart.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class WaveStart

  A class for the button to start next wave.

Index