lolmacrogame

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 gtest_lite Namespace Reference

gtest_lite: a keretrendszer függvényinek és objektumainak névtere

Classes

- struct Test

Functions

- bool almostEQ (double a, double b)

5.1.1 Detailed Description

gtest_lite: a keretrendszer függvényinek és objektumainak névtere

5.1.2 Function Documentation

5.1.2.1 almostEQ()

```
bool gtest_lite::almostEQ (
            double a,
            double b )  [inline]
```

Segédfüggvény valós számok összehasonlításához Nem bombabiztos, de nekünk most jó lesz Elméleti hátér: `http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm`

5.2 IOParser Namespace Reference

Classes

- class File

    a file holder that closes the file

Functions

- std::vector< std::string > split_string (const std::string &str, char delimiter)
- Champion ∗ create_champ (const std::string &line)
- Item create_item (const std::string &line)

5.2.1 Function Documentation

5.2.1.1 create_champ()

```
Champion * IOParser::create_champ (
             const std::string & line )
```

5.2.1.2 create_item()

```
Item IOParser::create_item (
             const std::string & line )
```

5.2.1.3 split_string()

```
std::vector< std::string > IOParser::split_string (
             const std::string & str,
             char delimiter )
```

5.3 Menu Namespace Reference

Classes

- class MenuState

    the general menustate class, used as a base for simple menus
- class MainState
- class ModeSelectionState
- class MenuButton

5.4 Resources Namespace Reference

Classes

- class Holder

  the class which holdes the resources for the application

Enumerations

- enum class Type { FONT }

  the types of resources there are

5.4.1 Enumeration Type Documentation

5.4.1.1 Type

```
enum Resources::Type  [strong]
```

the types of resources there are

Enumerator

FONT	

5.5 UI Namespace Reference

Classes

- class GridElement

  the base class for grid elements
- class Button

  the button class which implements a shape with some text on it, with an onclick method
- class TextBox

  the textbox element, which is a rectangle where text you can input text into
- class Grid

  the grid holds multiple grid elements, and places them in a given way
- class NamedBox

  the named box, which is a grid element that holds a shape and a text inside of it

Chapter 6

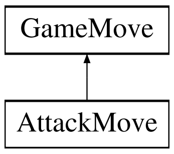
Class Documentation

6.1 AttackMove Class Reference

the class that implements the attack move

`#include <gamemoves.hpp>`

Inheritance diagram for AttackMove:



Public Member Functions

- AttackMove ()
- void finish (Cell ∗cell_) override

  finishes the gamemove, by giving it the cell to use
- std::string get_state_info () const override

  gets this gamemoves state information
- void do_move (Champion ∗champ, std::shared_ptr< Map > map) override

  does the move with the champ on the map

Additional Inherited Members

6.1.1 Detailed Description

the class that implements the attack move

6.1.2 Constructor & Destructor Documentation

6.1.2.1 AttackMove()

```
AttackMove::AttackMove ( )  [inline]
```

6.1.3 Member Function Documentation

6.1.3.1 do_move()

```
void AttackMove::do_move (
            Champion * champ,
            std::shared_ptr< Map > map )  [override], [virtual]
```

does the move with the champ on the map

Parameters

champ	the champ whose move it is
map	the map to do the moves on

Implements GameMove.

6.1.3.2 finish()

```
void AttackMove::finish (
            Cell * cell_ )  [override], [virtual]
```

finishes the gamemove, by giving it the cell to use

Reimplemented from GameMove.

6.1.3.3 get_state_info()

```
std::string AttackMove::get_state_info ( ) const  [override], [virtual]
```

gets this gamemoves state information

Returns

Reimplemented from GameMove.

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

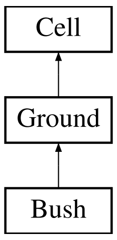
- include/gamemoves.hpp
- src/gamemoves.cpp

6.2 Bush Class Reference

calculates vision differently than the ground object

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

Inheritance diagram for Bush:



Public Member Functions

- Bush ()

6.2.1 Detailed Description

calculates vision differently than the ground object

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Bush()

```
Bush::Bush ( )
```

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

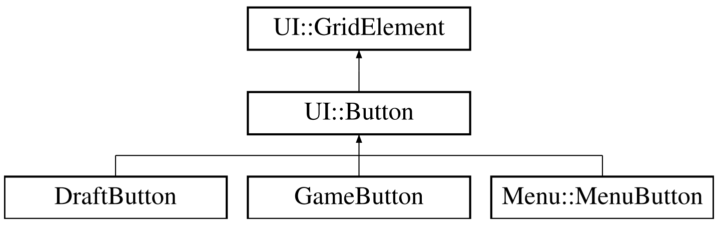
- include/map.hpp
- src/map.cpp

6.3 UI::Button Class Reference

the button class which implements a shape with some text on it, with an onclick method

```
#include <UIcomponents.hpp>
```

Inheritance diagram for UI::Button:



Public Member Functions

- Button ()=default
- Button (const sf::String &text, std::function< void()> onclick=[ ]() { std::cout<< "onclick not implemented yet"<< std::endl;}, sf::Vector2f pos={0, 0})

  constructs a button with the given params
- bool contains (int x, int y) const override

  checks if the given coordinates are inside the grid element
- void draw (sf::RenderWindow &window) override

  tells the gridelement to draw itself to the window
- sf::Vector2f get_size () override

  get's the size of the grid element
- void set_position (sf::Vector2f pos) override

  set's the position of the grid element relative to the window
- void update_text_position ()

  updates the texts position relative to the shape
- sf::FloatRect get_global_bounds () const

  gets the global bounds of the buttons shape
- void onclick_here (const sf::Event &event)

  the method to call to check if the button was clicked, if so then it calls his onclick

Protected Attributes

- sf::RectangleShape shape
- sf::Text text
- std::function< void()> onclick

6.3.1 Detailed Description

the button class which implements a shape with some text on it, with an onclick method

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Button() [1/2]

```
UI::Button::Button ( )  [default]
```

6.3.2.2 Button() [2/2]

```
Button::Button (
            const sf::String & text,
            std::function< void()> onclick = []() { std::cout << "onclick not implemented yet" << std↩
::endl;},
            sf::Vector2f pos = {0, 0} )  [explicit]
```

constructs a button with the given params

Parameters

text	the text on the button
onclick	the method to be called when the button gets clicked on
pos	the position relative to the window of the button

6.3.3 Member Function Documentation

6.3.3.1 contains()

```
bool Button::contains (
            int x,
            int y ) const  [override], [virtual]
```

checks if the given coordinates are inside the grid element

Parameters

x	x coordinate
y	y coordinate

Returns

true if they're inside, false if not

Implements UI::GridElement.

6.3.3.2 draw()

```
void Button::draw (
            sf::RenderWindow & window )  [override], [virtual]
```

tells the gridelement to draw itself to the window

Parameters

window	

Implements UI::GridElement.

6.3.3.3 get_global_bounds()

```
sf::FloatRect UI::Button::get_global_bounds ( ) const  [inline]
```

gets the global bounds of the buttons shape

Returns

the global bounds

6.3.3.4 get_size()

```
sf::Vector2f UI::Button::get_size ( )  [inline], [override], [virtual]
```

get's the size of the grid element

Returns

the size

Implements UI::GridElement.

6.3.3.5 onclick_here()

```
void Button::onclick_here (
             const sf::Event & event )
```

the method to call to check if the button was clicked, if so then it calls his onclick

Parameters

event	the event

6.3.3.6 set_position()

```
void Button::set_position (
             sf::Vector2f pos )  [override], [virtual]
```

set's the position of the grid element relative to the window

Parameters

pos	

Implements UI::GridElement.

6.3.3.7 update_text_position()

```
void Button::update_text_position ( )
```

updates the texts position relative to the shape

6.3.4 Member Data Documentation

6.3.4.1 onclick

```
std::function<void()> UI::Button::onclick  [protected]
```

6.3.4.2 shape

```
sf::RectangleShape UI::Button::shape  [protected]
```

6.3.4.3 text

```
sf::Text UI::Button::text  [protected]
```

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

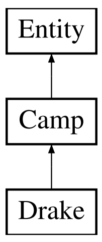
- include/UIcomponents.hpp
- src/UIcomponents.cpp

6.4 Camp Class Reference

a common class for camps which are not able to move (baron nashor, drakes and jungle camps) because of how the game works, every camp can give an effect to the champion(s) that slain it

```
#include <gameobjects.hpp>
```

Inheritance diagram for Camp:



Public Member Functions

- Camp (double hp_=100, double dmg_=15)

  constructs a camp with the given stats
- void set_effect (Effect e)

  set's the effect given by slaying this camp
- Effect get_buff_given () const override

  set's the basic stats for this camp
- void respawn () override

  if the entity isn't alive then should try to revive them, but generally this feature is not enabled, only the entities who want to use it should implement it

Additional Inherited Members

6.4.1 Detailed Description

a common class for camps which are not able to move (baron nashor, drakes and jungle camps) because of how the game works, every camp can give an effect to the champion(s) that slain it

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Camp()

```
Camp::Camp (
             double hp_ = 100,
             double dmg_ = 15 )
```

constructs a camp with the given stats

Parameters

hp↩_	
dmg↩_	

6.4.3 Member Function Documentation

6.4.3.1 get_buff_given()

```
Effect Camp::get_buff_given ( ) const  [inline], [override], [virtual]
```

set's the basic stats for this camp

Parameters

hp	
dmg	

Reimplemented from Entity.

6.4.3.2 respawn()

```
void Camp::respawn ( )  [override], [virtual]
```

if the entity isn't alive then should try to revive them, but generally this feature is not enabled, only the entities who want to use it should implement it

Reimplemented from Entity.

Reimplemented in Drake.

6.4.3.3 set_effect()

```
void Camp::set_effect (
             Effect e )  [inline]
```

set's the effect given by slaying this camp

Parameters

e	the effect to save

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

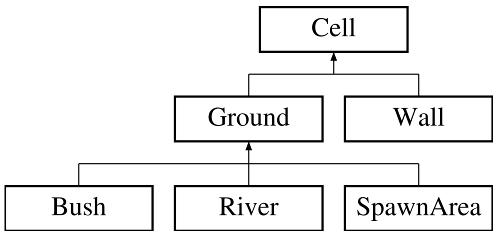
- include/gameobjects.hpp
- src/gameobjects.cpp

6.5 Cell Class Reference

the base class for a cell on the map

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

Inheritance diagram for Cell:



Public Member Functions

- Cell ()

  cell constructor
- virtual ∼Cell ()

  deletes the owned entities from the map
- virtual bool should_update_vision_around (Side side_)

  updates the vision of the cell, should be called after every move and round ends
- virtual bool can_buy_items () const

  returns true if entities can buy items on this cell
- void set_shop (bool shop_)

  set's this cell property to a shop cell
- virtual void set_selected ()

  sets the current cell as selected
- virtual bool is_selected () const

  returns true if the current cell is selected
- virtual bool can_move_here () const

  true if entities are able to move here
- virtual bool can_ward_here () const

  true if champions are able to put wards on this spot
- virtual bool can_attack_entity (Side enemy_side_) const

  returns if there are entities to attack on this cell
- virtual void add_entity (Entity ∗entity)

  adds an entity to its entity list
- virtual bool remove_entity (Entity ∗entity)

  removes the entity from the entity list
- void set_color (sf::Color color)

  sets the cell's color
- bool contains (const int x, const int y)

  checks if the given coordinates are inside the cell
- Entity ∗ get_entity_clicked (const int x, const int y)

  get's the entity clicked by the given coordinates
- virtual void set_highlighted ()

  sets the current cell to a highlighted color, to indicate it's clickable
- virtual void reset_selection_color ()

  only resets cell color to the default one, if it isn't a vision cell
- virtual void reset_vision_color ()

  sets back cell to having vision, but doesn't change it if it's selected
- virtual void set_vision (bool has_vision_)

  sets the property vision to the given argument
- void set_position (sf::Vector2f pos_)

  sets the current cells position

- virtual void draw (sf::RenderWindow &window)

  draws the cell and its entities to the screen
- void update_shape (sf::Vector2f map_position, sf::Vector2f cell_size, float margin=2)

  updates the shapes properties
- sf::Vector2f get_index () const

  gets the current index, this is where the cell is on the map
- void update_entities_shape (sf::Vector2f mappos)

  updates the entities positions on the given map position
- Entity ∗ get_first_entity ()

  gets the first entity on the cell
- Entity ∗ get_attackable_entity (Side side_)

  gets an entity that is attackable and is on the other side than given in params @params side_ the side which the entity asking is on, so it gives an entity of the opposing team
- void unselect ()

  unselects the current cell
- void do_attack (Map ∗map)

  tries to do attack on each one of its entities
- void update (Map ∗map)

  tells its entities to update themselves

6.5.1 Detailed Description

the base class for a cell on the map

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Cell()

```
Cell::Cell ( )  [inline]
```

cell constructor

6.5.2.2 ∼Cell()

```
Cell::∼Cell ( )  [virtual]
```

deletes the owned entities from the map

6.5.3 Member Function Documentation

6.5.3.1 add_entity()

```
void Cell::add_entity (
            Entity ∗ entity )  [virtual]
```

adds an entity to its entity list

Parameters

entity	the entity to add

6.5.3.2 can_attack_entity()

```
bool Cell::can_attack_entity (
            Side enemy_side_ ) const  [virtual]
```

returns if there are entities to attack on this cell

Parameters

*enemy_↩ side_*	returns true only if there are entities that aren't from this side

6.5.3.3 can_buy_items()

```
virtual bool Cell::can_buy_items ( ) const  [inline], [virtual]
```

returns true if entities can buy items on this cell

6.5.3.4 can_move_here()

```
virtual bool Cell::can_move_here ( ) const  [inline], [virtual]
```

true if entities are able to move here

Reimplemented in Wall.

6.5.3.5 can_ward_here()

```
virtual bool Cell::can_ward_here ( ) const  [inline], [virtual]
```

true if champions are able to put wards on this spot

Reimplemented in Wall.

6.5.3.6 contains()

```
bool Cell::contains (
            const int x,
            const int y )
```

checks if the given coordinates are inside the cell

Parameters

x	coordinate
y	coordinate

6.5.3.7 do_attack()

```
void Cell::do_attack (
             Map * map )
```

tries to do attack on each one of its entities

Parameters

map	

6.5.3.8 draw()

```
void Cell::draw (
             sf::RenderWindow & window )  [virtual]
```

draws the cell and its entities to the screen

Parameters

window	window to draw to

6.5.3.9 get_attackable_entity()

```
Entity * Cell::get_attackable_entity (
             Side side_ )
```

gets an entity that is attackable and is on the other side than given in params @params side_ the side which the entity asking is on, so it gives an entity of the opposing team

6.5.3.10 get_entity_clicked()

```
Entity * Cell::get_entity_clicked (
             const int x,
             const int y )
```

get's the entity clicked by the given coordinates

Parameters

x	coordinate
y	coordinate

6.5.3.11 get_first_entity()

```
Entity* Cell::get_first_entity ( )  [inline]
```

gets the first entity on the cell

6.5.3.12 get_index()

```
sf::Vector2f Cell::get_index ( ) const  [inline]
```

gets the current index, this is where the cell is on the map

6.5.3.13 is_selected()

```
virtual bool Cell::is_selected ( ) const  [inline], [virtual]
```

returns true if the current cell is selected

6.5.3.14 remove_entity()

```
bool Cell::remove_entity (
             Entity * entity )  [virtual]
```

removes the entity from the entity list

Parameters

entity	the entity to remove

Returns

true if the entity was found and removed

6.5.3.15 reset_selection_color()

```
void Cell::reset_selection_color ( )  [virtual]
```

only resets cell color to the default one, if it isn't a vision cell

6.5.3.16 reset_vision_color()

```
void Cell::reset_vision_color ( )  [virtual]
```

sets back cell to having vision, but doesn't change it if it's selected

6.5.3.17 set_color()

```
void Cell::set_color (
             sf::Color color )
```

sets the cell's color

Parameters

color	the color to set it to

6.5.3.18 set_highlighted()

```
void Cell::set_highlighted ( )  [virtual]
```

sets the current cell to a highlighted color, to indicate it's clickable

6.5.3.19 set_position()

```
void Cell::set_position (
             sf::Vector2f pos_ )
```

sets the current cells position

Parameters

pos	the position to change to

6.5.3.20 set_selected()

```
void Cell::set_selected ( )  [virtual]
```

sets the current cell as selected

6.5.3.21 set_shop()

```
void Cell::set_shop (
             bool shop_ )  [inline]
```

set's this cell property to a shop cell

Parameters

shop	true if this is a cell where entities can shop

6.5.3.22 set_vision()

```
void Cell::set_vision (
             bool has_vision_ )  [virtual]
```

sets the property vision to the given argument

Parameters

vision	true if this cell has vision

6.5.3.23 should_update_vision_around()

```
bool Cell::should_update_vision_around (
             Side side_ )  [virtual]
```

updates the vision of the cell, should be called after every move and round ends

Parameters

*side_*	which side should have vision

6.5.3.24 unselect()

```
void Cell::unselect ( )
```

unselects the current cell

6.5.3.25 update()

```
void Cell::update (
             Map * map )
```

tells its entities to update themselves

Parameters

map	the map

6.5.3.26 update_entities_shape()

```
void Cell::update_entities_shape (
             sf::Vector2f mappos )
```

updates the entities positions on the given map position

Parameters

mappos	where it should put the shapes

6.5.3.27 update_shape()

```
void Cell::update_shape (
             sf::Vector2f map_position,
             sf::Vector2f cell_size,
             float margin = 2 )
```

updates the shapes properties

Parameters

map_position	the map position where this shape should be
cell_size	the size of this cell
margin	the margin to leave between it's neighbours

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

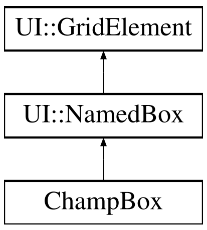
- include/map.hpp
- src/map.cpp

6.6 ChampBox Class Reference

a champion box implementation, which holds a champ

```
#include <draft.hpp>
```

Inheritance diagram for ChampBox:



Public Member Functions

- ChampBox (const std::string &label, [[maybe_unused]]sf::RectangleShape frame, Resources::Holder &holder, Champion ∗champ)
- Champion ∗ get_champ () const

    gets the champion which is held in this box

Additional Inherited Members

6.6.1 Detailed Description

a champion box implementation, which holds a champ

6.6.2 Constructor & Destructor Documentation

6.6.2.1 ChampBox()

```
ChampBox::ChampBox (
            const std::string & label,
            [[maybe_unused] ]sf::RectangleShape frame,
            Resources::Holder & holder,
            Champion ∗ champ )  [inline]
```

6.6.3 Member Function Documentation

6.6.3.1 get_champ()

`Champion* ChampBox::get_champ ( ) const [inline]`

gets the champion which is held in this box

Returns

the champion

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

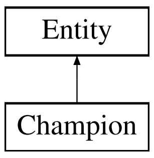
- include/draft.hpp

6.7 Champion Class Reference

class for describing champions, they're a type of entities that the players can manipulate with gamemoves

`#include <gameobjects.hpp>`

Inheritance diagram for Champion:



Public Member Functions

- Champion ()

  constructs a basic champion with all the necessary values
- Champion (const std::string &name_, double damage_, double dmg_per_level_, double hp_, double hp_↩
per_level_)

  constructing a champion with all its necessary properties
- ∼Champion () override

  destructor for champion class, frees the heap allocated properties
- void fight (Entity ∗other)

  this champion fights another entity, calculates who won, then decreases both entities base_hp gives the required assets to the appropriate entites (such as gold, xp, cs)
- void add_item (Item ∗item)

  adds an item to the champions item list if the required criteria is met
- std::vector< std::string > get_stats () const override

  gets this champion's new statistics, but also calls the parent for it's statistics

- void set_icon (char c)

    set's the icon for this champion
- double get_total_dmg () const override

    returns the total damage, by adding buffs/items to the base dmg
- void set_side (Side side_) override

    set's which team(side) is the entity on
- std::string get_name () const

    gets the champions name
- void set_font (Resources::Holder &holder)

    set's the font face for the text
- void draw (sf::RenderWindow &window) override

    draws the champion to the window
- int getmovepoints () const

    gets the currently available movepoints
- void add_gamemove (GameMove ∗move)

    add's a gamemove to the champions gamemove list
- double get_max_hp () const override

    returns the base base_hp, works the same way as get_base_dmg
- bool is_gamemove_complete () const

    check's if the latest gamemove is complete, this means if it can be used up, or it needs some more data
- Cell ∗ get_simulation_cell () override

    returns the cell on which the champion advances to with gamemoves during a round
- sf::Vector2f last_gamemove_index () const

    gets the map position where the last gamemove will act
- sf::Vector2f current_gamemove_index () const

    gets the map position where the current gamemove will act
- void finish_gamemove (Cell ∗cell)

    finishes the last gamemove's setup, by giving it the selected cell
- void remove_last_gamemove ()

    removes the last gamemove from this champion
- void despawn_wards (std::shared_ptr< Map > map)

    despawns the champions wards from the map
- void update_shape_pos (sf::Vector2f pos) override

    update the champion's shape position, to match where it should be on the map
- void do_move (std::shared_ptr< Map > map)

    does one move from the gamemoves, starting from the first one
- void set_simulation (bool sim)

    set's the champions state to the given param, need to know which state it's in while drawing it to the screen
- void round_end (const std::shared_ptr< Map > &map)

    after a round ends (e.g. both players finished their turns) the champion ends it, reset's the necessary variables, prepares for the upcoming round
- void add_xp (int xp)

    adds xp to the champion, and also checks if the champion leveled up with this xp
- bool can_fight_back () const override

    describes if the champion can fight back another entities Used if this champ has a chance to win, when it's in execute range of the other entity
- void place_ward (const std::shared_ptr< Map > &map, Cell ∗c)

    places a ward on the map, if the given prerequisites are true
- bool gives_vision () const override

    describes if this entity gives vision
- void clear_gamemoves ()

clears the gamemoves list and also deletes each from the heap, set's the current_gamemove to nullptr

- void move (std::shared_ptr< Map > &map)

    moves the player on the map
- void killed_other (Entity ∗other) override

    if this entity killed another one
- void set_spawn_point (Cell ∗spawn_point_)

    set's the spawnpoint for the champion
- std::string get_current_gamemove_state_info () const

    gets the current gamemoves state information

Additional Inherited Members

6.7.1 Detailed Description

class for describing champions, they're a type of entities that the players can manipulate with gamemoves

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Champion() [1/2]

```
Champion::Champion ( )
```

constructs a basic champion with all the necessary values

6.7.2.2 Champion() [2/2]

```
Champion::Champion (
            const std::string & name_,
            double damage_,
            double dmg_per_level_,
            double hp_,
            double hp_per_level_ )
```

constructing a champion with all its necessary properties

Parameters

*name_*	
*damage_*	
*dmg_per_↩ level_*	
*hp_*	
hp_per_level↩ _	

6.7.2.3 ∼Champion()

```
Champion::∼Champion ( )  [override]
```

destructor for champion class, frees the heap allocated properties

6.7.3 Member Function Documentation

6.7.3.1 add_gamemove()

```
void Champion::add_gamemove (
             GameMove * move )
```

add's a gamemove to the champions gamemove list

Parameters

move	the gamemove to add

6.7.3.2 add_item()

```
void Champion::add_item (
             Item * item )
```

adds an item to the champions item list if the required criteria is met

Parameters

item	the one to add to the list

6.7.3.3 add_xp()

```
void Champion::add_xp (
             int xp )
```

adds xp to the champion, and also checks if the champion leveled up with this xp

6.7.3.4 can_fight_back()

```
bool Champion::can_fight_back ( ) const  [inline], [override], [virtual]
```

describes if the champion can fight back another entities Used if this champ has a chance to win, when it's in execute range of the other entity

Reimplemented from Entity.

6.7.3.5 clear_gamemoves()

```
void Champion::clear_gamemoves ( )
```

clears the gamemoves list and also deletes each from the heap, set's the current_gamemove to nullptr

6.7.3.6 current_gamemove_index()

```
sf::Vector2f Champion::current_gamemove_index ( ) const
```

gets the map position where the current gamemove will act

6.7.3.7 despawn_wards()

```
void Champion::despawn_wards (
             std::shared_ptr< Map > map )
```

despawns the champions wards from the map

Parameters

map	

6.7.3.8 do_move()

```
void Champion::do_move (
             std::shared_ptr< Map > map )
```

does one move from the gamemoves, starting from the first one

Parameters

map	the map let's it communicate with other entities surrounding it

6.7.3.9 draw()

```
void Champion::draw (
             sf::RenderWindow & window )  [override], [virtual]
```

draws the champion to the window

Parameters

window	the window to draw to

Reimplemented from Entity.

6.7.3.10 fight()

```
void Champion::fight (
             Entity * other )
```

this champion fights another entity, calculates who won, then decreases both entities base_hp gives the required assets to the appropriate entites (such as gold, xp, cs)

Parameters

other	the entity to fight

6.7.3.11 finish_gamemove()

```
void Champion::finish_gamemove (
             Cell * cell )
```

finishes the last gamemove's setup, by giving it the selected cell

Parameters

cell	the cell that was selected by the user

6.7.3.12 get_current_gamemove_state_info()

```
std::string Champion::get_current_gamemove_state_info ( ) const
```

gets the current gamemoves state information

Returns

the state info

6.7.3.13 get_max_hp()

```
double Champion::get_max_hp ( ) const  [override], [virtual]
```

returns the base base_hp, works the same way as get_base_dmg

Reimplemented from Entity.

6.7.3.14 get_name()

```
std::string Champion::get_name ( ) const  [inline]
```

gets the champions name

6.7.3.15 get_simulation_cell()

```
Cell * Champion::get_simulation_cell ( )  [override], [virtual]
```

returns the cell on which the champion advances to with gamemoves during a round

Reimplemented from Entity.

6.7.3.16 get_stats()

```
std::vector< std::string > Champion::get_stats ( ) const  [override], [virtual]
```

gets this champion's new statistics, but also calls the parent for it's statistics

Reimplemented from Entity.

6.7.3.17 get_total_dmg()

```
double Champion::get_total_dmg ( ) const  [override], [virtual]
```

returns the total damage, by adding buffs/items to the base dmg

Reimplemented from Entity.

6.7.3.18 getmovepoints()

```
int Champion::getmovepoints ( ) const  [inline]
```

gets the currently available movepoints

6.7.3.19 gives_vision()

```
bool Champion::gives_vision ( ) const  [inline], [override], [virtual]
```

describes if this entity gives vision

Reimplemented from Entity.

6.7.3.20 is_gamemove_complete()

```
bool Champion::is_gamemove_complete ( ) const
```

check's if the latest gamemove is complete, this means if it can be used up, or it needs some more data

Returns

true if the gamemove is complete, false if not

6.7.3.21 killed_other()

```
void Champion::killed_other (
             Entity * other )  [override], [virtual]
```

if this entity killed another one

Parameters

other	the other entity that was killed

Reimplemented from Entity.

6.7.3.22 last_gamemove_index()

```
sf::Vector2f Champion::last_gamemove_index ( ) const
```

gets the map position where the last gamemove will act

6.7.3.23 move()

```
void Champion::move (
             std::shared_ptr< Map > & map )
```

moves the player on the map

Parameters

map	the map to move on

6.7.3.24 place_ward()

```
void Champion::place_ward (
             const std::shared_ptr< Map > & map,
             Cell * c )
```

places a ward on the map, if the given prerequisites are true

Parameters

map	the map to place the ward on
cell	the cell on the map where the ward should be placed

6.7.3.25 remove_last_gamemove()

```
void Champion::remove_last_gamemove ( )
```

removes the last gamemove from this champion

6.7.3.26 round_end()

```
void Champion::round_end (
            const std::shared_ptr< Map > & map )
```

after a round ends (e.g. both players finished their turns) the champion ends it, reset's the necessary variables, prepares for the upcoming round

Parameters

map	the map is there if there are entities to remove

6.7.3.27 set_font()

```
void Champion::set_font (
            Resources::Holder & holder )
```

set's the font face for the text

Parameters

holder	the object that let's you retrieve the font face

6.7.3.28 set_icon()

```
void Champion::set_icon (
            char c )  [inline]
```

set's the icon for this champion

Parameters

c	the char to use

6.7.3.29 set_side()

```
void Champion::set_side (
            Side side_ )  [override], [virtual]
```

set's which team(side) is the entity on

Reimplemented from Entity.

6.7.3.30 set_simulation()

```
void Champion::set_simulation (
            bool sim )  [inline]
```

set's the champions state to the given param, need to know which state it's in while drawing it to the screen

Parameters

sim	true if its simulation, false if not

6.7.3.31 set_spawn_point()

```
void Champion::set_spawn_point (
            Cell * spawn_point_ )
```

set's the spawnpoint for the champion

Parameters

*spawn_↩ point_*	

6.7.3.32 update_shape_pos()

```
void Champion::update_shape_pos (
            sf::Vector2f pos )  [override], [virtual]
```

update the champion's shape position, to match where it should be on the map

Parameters

pos	the position to change to

Reimplemented from Entity.

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

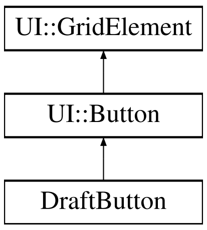
- include/gameobjects.hpp
- src/gameobjects.cpp

6.8 DraftButton Class Reference

class that specializes Button, to a draftbutton with the correct design

```
#include <draft.hpp>
```

Inheritance diagram for DraftButton:



Public Member Functions

- DraftButton (Resources::Holder &h, const sf::String &str, [[maybe_unused]] std::function< void()> onclick=[ ]() { std::cout<< "not impl"<< std::endl;})

Additional Inherited Members

6.8.1 Detailed Description

class that specializes Button, to a draftbutton with the correct design

6.8.2 Constructor & Destructor Documentation

6.8.2.1 DraftButton()

```
DraftButton::DraftButton (
            Resources::Holder & h,
            const sf::String & str,
            [[maybe_unused] ] std::function< void()> onclick = []() { std::cout << "not impl" << std↩
::endl; } )
```

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

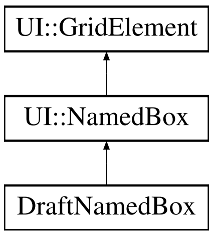
- include/draft.hpp
- src/draft.cpp

6.9 DraftNamedBox Class Reference

class that specializes NamedBox, to a NamedBox with the correct design

```
#include <draft.hpp>
```

Inheritance diagram for DraftNamedBox:



Public Member Functions

- DraftNamedBox (Resources::Holder &holder, sf::Vector2f size={100, 30})
  constructs the draftnamedbox with the correct design

Additional Inherited Members

6.9.1 Detailed Description

class that specializes NamedBox, to a NamedBox with the correct design

6.9.2 Constructor & Destructor Documentation

6.9.2.1 DraftNamedBox()

```
DraftNamedBox::DraftNamedBox (
            Resources::Holder & holder,
            sf::Vector2f size = {100,30} )  [explicit]
```

constructs the draftnamedbox with the correct design

Parameters

holder	the object that can get the font face for the component
size	the size of the namedbox

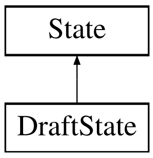
The documentation for this class was generated from the following files:

- include/draft.hpp
- src/draft.cpp

6.10 DraftState Class Reference

```
#include <draft.hpp>
```

Inheritance diagram for DraftState:



Public Member Functions

- DraftState (StateManager &state_manager, Settings &settings, sf::RenderWindow &window)
- ∼DraftState () override
- void handle_events (sf::Event &event) override

  handles the given event
- void update () override

  updates the states
- void draw (sf::RenderWindow &window) override

  draws the state's contents to the given window
- void lock_in (StateManager &state_manager, sf::RenderWindow &window, Settings &settings)

  locks in the currently selected champion to the correct draftstate
- void dont_ban ()

  sets the currently selected champbox to an empty champion, which means the player didn't ban

Additional Inherited Members

6.10.1 Constructor & Destructor Documentation

6.10.1.1 DraftState()

```
DraftState::DraftState (
            StateManager & state_manager,
            Settings & settings,
            sf::RenderWindow & window )
```

6.10.1.2 ∼DraftState()

```
DraftState::∼DraftState ( )  [override]
```

6.10.2 Member Function Documentation

6.10.2.1 dont_ban()

```
void DraftState::dont_ban ( )
```

sets the currently selected champbox to an empty champion, which means the player didn't ban

6.10.2.2 draw()

```
void DraftState::draw (
            sf::RenderWindow & window )  [override], [virtual]
```

draws the state's contents to the given window

Parameters

window	the window to draw to

Implements State.

6.10.2.3 handle_events()

```
void DraftState::handle_events (
            sf::Event & event )  [override], [virtual]
```

handles the given event

Parameters

event	the event to be handled

Implements State.

6.10.2.4 lock_in()

```
void DraftState::lock_in (
             StateManager & state_manager,
             sf::RenderWindow & window,
             Settings & settings )
```

locks in the currently selected champion to the correct draftstate

Parameters

state_manager	the statemanager of the application
window	the window of the program
settings	the currently used gamesettings

6.10.2.5 update()

```
void DraftState::update ( )  [override], [virtual]
```

updates the states

Implements State.

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

- include/draft.hpp
- src/draft.cpp

6.11 DraftTurn Class Reference

class used to store one draft turn

```
#include <draft.hpp>
```

Public Member Functions

- DraftTurn (std::vector< Champion * > &champs, bool ban_phase_=false)

    constructor, initializes it's champs and if its ban_phase or not
- void do_turn (Champion *champ)

    does one turn
- bool is_ban_phase () const

    returns the ban_phase variable, true if this turn is ban_phase

6.11.1 Detailed Description

class used to store one draft turn

6.11.2 Constructor & Destructor Documentation

6.11.2.1 DraftTurn()

```
DraftTurn::DraftTurn (
             std::vector< Champion * > & champs,
             bool ban_phase_ = false )  [inline], [explicit]
```

constructor, initializes it's champs and if its ban_phase or not

Parameters

champs	the champs vector which should be used for it

6.11.3 Member Function Documentation

6.11.3.1 do_turn()

```
void DraftTurn::do_turn (
             Champion * champ )
```

does one turn

6.11.3.2 is_ban_phase()

```
bool DraftTurn::is_ban_phase ( ) const  [inline]
```

returns the ban_phase variable, true if this turn is ban_phase

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

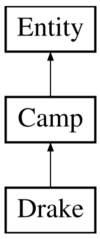
- include/draft.hpp
- src/draft.cpp

6.12 Drake Class Reference

the class that describes dragons, there are different types of dragons, with different effects (currently only two)

```
#include <gameobjects.hpp>
```

Inheritance diagram for Drake:



Public Member Functions

- Drake ()

  set's up the dragons attributes (base_hp,dmg) and decides which type it should be
- void decide_which_type ()

  decides the type of effect that should be given by slaying this dragon
- void respawn () override

  if the entity isn't alive then should try to revive them, but generally this feature is not enabled, only the entities who want to use it should implement it

Additional Inherited Members

6.12.1 Detailed Description

the class that describes dragons, there are different types of dragons, with different effects (currently only two)

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Drake()

```
Drake::Drake ( )
```

set's up the dragons attributes (base_hp,dmg) and decides which type it should be

6.12.3 Member Function Documentation

6.12.3.1 decide_which_type()

```
void Drake::decide_which_type ( )
```

decides the type of effect that should be given by slaying this dragon

6.12.3.2 respawn()

```
void Drake::respawn ( )  [override], [virtual]
```

if the entity isn't alive then should try to revive them, but generally this feature is not enabled, only the entities who want to use it should implement it

Reimplemented from Camp.

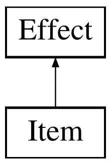
The documentation for this class was generated from the following files:

- include/gameobjects.hpp
- src/gameobjects.cpp

6.13 Effect Class Reference

```
#include <gameobjects.hpp>
```

Inheritance diagram for Effect:



Public Member Functions

- Effect (int dmg=0, int hp=0, bool expires=false, int cooldown=0)

  contructor which set's bonus_dmg and bonus_hp to their respective values
- double get_bonus_dmg () const

  get's the bonus_dmg
- double get_bonus_hp () const

  get's the bonus_hp
- void set_bonus_dmg (double bonus_dmg_)

  set's bonus_dmg
- void set_bonus_hp (double bonus_hp_)

  set's bonus_hp
- bool not_zero () const
- bool update_expire ()

  decreases the cooldown and checks if the effect has expired

6.13.1 Constructor & Destructor Documentation

6.13.1.1 Effect()

```
Effect::Effect (
            int dmg = 0,
            int hp = 0,
            bool expires = false,
            int cooldown = 0 )  [inline], [explicit]
```

contructor which set's bonus_dmg and bonus_hp to their respective values

Parameters

dmg	the new damage to use
hp	the new base_hp to use
expires	if the effect expires after some time
cooldown	the time it takes to expire

6.13.2 Member Function Documentation

6.13.2.1 get_bonus_dmg()

```
double Effect::get_bonus_dmg ( ) const  [inline]
```

get's the bonus_dmg

6.13.2.2 get_bonus_hp()

```
double Effect::get_bonus_hp ( ) const  [inline]
```

get's the bonus_hp

6.13.2.3 not_zero()

```
bool Effect::not_zero ( ) const  [inline]
```

checks if the two properties are zero, or not

Returns

true if both of them aren't zero

6.13.2.4 set_bonus_dmg()

```
void Effect::set_bonus_dmg (
             double bonus_dmg_ )  [inline]
```

set's bonus_dmg

6.13.2.5 set_bonus_hp()

```
void Effect::set_bonus_hp (
             double bonus_hp_ )  [inline]
```

set's bonus_hp

6.13.2.6 update_expire()

```
bool Effect::update_expire ( )
```

decreases the cooldown and checks if the effect has expired

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

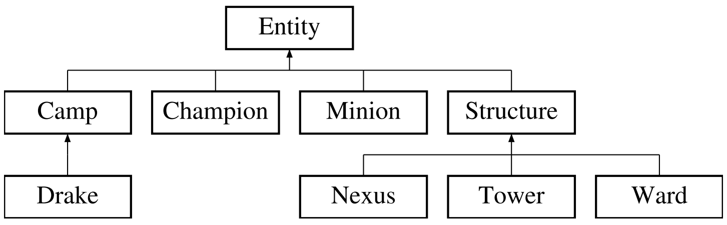
- include/gameobjects.hpp
- src/gameobjects.cpp

6.14 Entity Class Reference

The class that describes an entity.

```
#include <gameobjects.hpp>
```

Inheritance diagram for Entity:



Public Member Functions

- virtual ∼Entity ()=default

  entity's default destructor
- Entity (std::string name="")

  entity's constructor
- virtual void draw (sf::RenderWindow &window)

  the method that draws the entity to the window
- virtual double get_max_hp () const

  returns the base base_hp, works the same way as get_base_dmg
- virtual double get_total_dmg () const

  returns the total damage, by adding buffs/items to the base dmg
- int get_xp_given () const

  returns the amount of experience given to the entity that kills this entity
- double get_current_hp () const
- void refill_hp ()

  refill the base_hp of the champion
- virtual Effect get_buff_given () const

  gets the buff given to the other enemy, if this one gets killed
- int get_gold_given () const

  returns the amount of gold given to the entity that kills this entity
- virtual void set_side (Side side_)

  set's which team(side) is the entity on
- Side get_side () const

  gets the current team the entity is on
- void set_xp_given (int xp_given_)

  set's the amount of xp this entity could give
- virtual std::vector< std::string > get_stats () const

  gets this entity's statistics that could describe it
- bool is_alive () const

  checks if the entity is alive currently
- virtual bool should_focus () const

  returns true, if this entity should be focused by other entities when trying to pick a fight
- virtual bool gives_creep_score () const

  checks if this entity increases the creep score of the other entity when killed
- void remove_hp (double dmg)

  removes the given damage from the entity's total base_hp, and checks if the entity died by this damage
- void check_death ()

  checks if the entity died
- virtual void respawn ()

  if the entity isn't alive then should try to revive them, but generally this feature is not enabled, only the entities who want to use it should implement it
- virtual bool clicked (int x, int y)

  checks if the entity's shape was clicked on
- Cell ∗ get_real_cell ()

  returns the cell on which the entity is at every start of the round
- virtual Cell ∗ get_simulation_cell ()

  returns the cell on which the entity advances to with gamemoves during a round
- void set_cell (Cell ∗c)

  set's this entity's cell, it should be a valid cell on the map
- virtual void update_shape_pos (sf::Vector2f pos)

update's the shape's position so it appears on it's cell
- virtual bool gives_vision () const

    return true if this entity should give vision around him
- void set_color (sf::Color color_)

    set's this entity's shape fillcolor to the given color
- void set_name (std::string name_)

    set's the entitiy's name
- virtual bool can_fight_back () const

    returns true, if this is an entity that can fight back
- virtual void killed_other (Entity ∗entity)

    if this entity killed another, then this method should be called
- virtual void attack (Map ∗map)

    does an attack on its surrounding area

Static Public Member Functions

- static std::string to_ui_int_format (double num)

    changes the given num to the format which should be used on the ui

Protected Attributes

- std::string name
- bool alive = true
- double base_hp = 10
- double current_hp = 10
- double damage = 10
- int respawn_counter = 0
- int respawn_timer = 8
- int xp_given = 10
- int gold_given = 30
- Cell ∗ cell = nullptr
- Side side = Side::BLUE
- sf::Color color
- sf::RectangleShape shape

6.14.1 Detailed Description

The class that describes an entity.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 ∼Entity()

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

entity's default destructor

6.14.2.2 Entity()

```
Entity::Entity (
            std::string name = "" )  [explicit]
```

entity's constructor

Parameters

name	the name of the entity

6.14.3 Member Function Documentation

6.14.3.1 attack()

```
void Entity::attack (
            Map * map )  [virtual]
```

does an attack on its surrounding area

Parameters

map	the map to attack on

Reimplemented in Tower.

6.14.3.2 can_fight_back()

```
virtual bool Entity::can_fight_back ( ) const  [inline], [virtual]
```

returns true, if this is an entity that can fight back

Reimplemented in Champion.

6.14.3.3 check_death()

```
void Entity::check_death ( )
```

checks if the entity died

6.14.3.4 clicked()

```
bool Entity::clicked (
             int x,
             int y )  [virtual]
```

checks if the entity's shape was clicked on

6.14.3.5 draw()

```
void Entity::draw (
             sf::RenderWindow & window )  [virtual]
```

the method that draws the entity to the window

Parameters

window	the window to draw to

Reimplemented in Champion.

6.14.3.6 get_buff_given()

```
virtual Effect Entity::get_buff_given ( ) const  [inline], [virtual]
```

gets the buff given to the other enemy, if this one gets killed

Returns

the buff

Reimplemented in Camp.

6.14.3.7 get_current_hp()

```
double Entity::get_current_hp ( ) const  [inline]
```

gets the current hp of this entity

Returns

the hp

6.14.3.8 get_gold_given()

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

returns the amount of gold given to the entity that kills this entity

6.14.3.9 get_max_hp()

```
virtual double Entity::get_max_hp ( ) const  [inline], [virtual]
```

returns the base base_hp, works the same way as get_base_dmg

Reimplemented in Champion.

6.14.3.10 get_real_cell()

```
Cell* Entity::get_real_cell ( )  [inline]
```

returns the cell on which the entity is at every start of the round

6.14.3.11 get_side()

```
Side Entity::get_side ( ) const  [inline]
```

gets the current team the entity is on

6.14.3.12 get_simulation_cell()

```
virtual Cell* Entity::get_simulation_cell ( )  [inline], [virtual]
```

returns the cell on which the entity advances to with gamemoves during a round

Reimplemented in Champion.

6.14.3.13 get_stats()

```
std::vector< std::string > Entity::get_stats ( ) const  [virtual]
```

gets this entity's statistics that could describe it

Reimplemented in Champion, and Ward.

6.14.3.14 get_total_dmg()

```
virtual double Entity::get_total_dmg ( ) const  [inline], [virtual]
```

returns the total damage, by adding buffs/items to the base dmg

Reimplemented in Champion.

6.14.3.15 get_xp_given()

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

returns the amount of experience given to the entity that kills this entity

6.14.3.16 gives_creep_score()

```
virtual bool Entity::gives_creep_score ( ) const  [inline], [virtual]
```

checks if this entity increases the creep score of the other entity when killed

Reimplemented in Minion.

6.14.3.17 gives_vision()

```
virtual bool Entity::gives_vision ( ) const  [inline], [virtual]
```

return true if this entity should give vision around him

Reimplemented in Minion, and Champion.

6.14.3.18 is_alive()

```
bool Entity::is_alive ( ) const  [inline]
```

checks if the entity is alive currently

6.14.3.19 killed_other()

```
void Entity::killed_other (
             Entity * entity )  [virtual]
```

if this entity killed another, then this method should be called

Parameters

entity	

Reimplemented in Champion.

6.14.3.20 refill_hp()

```
void Entity::refill_hp ( )  [inline]
```

refill the base_hp of the champion

6.14.3.21 remove_hp()

```
void Entity::remove_hp (
             double dmg )
```

removes the given damage from the entity's total base_hp, and checks if the entity died by this damage

Parameters

dmg	the amount of damage dealt to this entity

6.14.3.22 respawn()

```
virtual void Entity::respawn ( )  [inline], [virtual]
```

if the entity isn't alive then should try to revive them, but generally this feature is not enabled, only the entities who want to use it should implement it

Reimplemented in Drake, and Camp.

6.14.3.23 set_cell()

```
void Entity::set_cell (
             Cell * c )  [inline]
```

set's this entity's cell, it should be a valid cell on the map

6.14.3.24 set_color()

```
void Entity::set_color (
             sf::Color color_ )  [inline]
```

set's this entity's shape fillcolor to the given color

Parameters

color	the color to use

6.14.3.25 set_name()

```
void Entity::set_name (
             std::string name_ )  [inline]
```

set's the entitiy's name

Parameters

name	the name to use instead

6.14.3.26 set_side()

```
virtual void Entity::set_side (
             Side side_ )  [inline], [virtual]
```

set's which team(side) is the entity on

Reimplemented in Champion.

6.14.3.27 set_xp_given()

```
void Entity::set_xp_given (
             int xp_given_ )  [inline]
```

set's the amount of xp this entity could give

6.14.3.28 should_focus()

```
virtual bool Entity::should_focus ( ) const  [inline], [virtual]
```

returns true, if this entity should be focused by other entities when trying to pick a fight

Reimplemented in Minion.

6.14.3.29 to_ui_int_format()

```
std::string Entity::to_ui_int_format (
             double num )  [static]
```

changes the given num to the format which should be used on the ui

Parameters

num	the number to convert

Returns

the string which should be drawn to the screen

6.14.3.30 update_shape_pos()

```
void Entity::update_shape_pos (
             sf::Vector2f pos )  [virtual]
```

update's the shape's position so it appears on it's cell

Reimplemented in Champion.

6.14.4 Member Data Documentation

6.14.4.1 alive

```
bool Entity::alive = true  [protected]
```

6.14.4.2 base_hp

```
double Entity::base_hp = 10  [protected]
```

6.14.4.3 cell

```
Cell* Entity::cell = nullptr  [protected]
```

6.14.4.4 color

```
sf::Color Entity::color  [protected]
```

6.14.4.5 current_hp

```
double Entity::current_hp = 10  [protected]
```

6.14.4.6 damage

```
double Entity::damage = 10  [protected]
```

6.14.4.7 gold_given

```
int Entity::gold_given = 30  [protected]
```

6.14.4.8 name

```
std::string Entity::name  [protected]
```

6.14.4.9 respawn_counter

```
int Entity::respawn_counter = 0  [protected]
```

6.14.4.10 respawn_timer

```
int Entity::respawn_timer = 8  [protected]
```

6.14.4.11 shape

```
sf::RectangleShape Entity::shape  [protected]
```

6.14.4.12 side

```
Side Entity::side = Side::BLUE  [protected]
```

6.14.4.13 xp_given

```
int Entity::xp_given = 10  [protected]
```

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

- include/gameobjects.hpp
- src/gameobjects.cpp

6.15 IOParser::File Class Reference

a file holder that closes the file

`#include <ioparser.h>`

Public Member Functions

- File (const std::filesystem::path &path)

  opens the file at the given path, throws error if the path is wrong
- ∼File ()
- std::fstream & getfile ()

  gets the opened file

6.15.1 Detailed Description

a file holder that closes the file

6.15.2 Constructor & Destructor Documentation

6.15.2.1 File()

```
IOParser::File::File (
             const std::filesystem::path & path )  [explicit]
```

opens the file at the given path, throws error if the path is wrong

Parameters

path	the filepath

6.15.2.2 ∼File()

```
IOParser::File::~File ( )  [inline]
```

6.15.3 Member Function Documentation

6.15.3.1 getfile()

```
std::fstream& IOParser::File::getfile ( )  [inline]
```

gets the opened file

Returns

the file

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

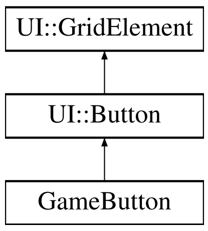
- include/ioparser.h
- src/ioparser.cpp

6.16 GameButton Class Reference

a button that has a specific style used for game buttons

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

Inheritance diagram for GameButton:



Public Member Functions

- GameButton (Resources::Holder &holder, const sf::String &str, std::function< void()> onclick=[ ]() { std↩ ::cout<< "not impl"<< std::endl;}, sf::Vector2f pos={0, 0})

  constructs a gamebutton

Additional Inherited Members

6.16.1 Detailed Description

a button that has a specific style used for game buttons

6.16.2 Constructor & Destructor Documentation

6.16.2.1 GameButton()

```
GameButton::GameButton (
            Resources::Holder & holder,
            const sf::String & str,
            std::function< void()> onclick = []() { std::cout << "not impl" << std::endl; },
            sf::Vector2f pos = {0,0} )
```

constructs a gamebutton

Parameters

holder	the resources holder
str	the title of the button
onclick	the onclick that has to be called if the gamebutton gets clicked
pos	the position of the gamebutton on the window

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

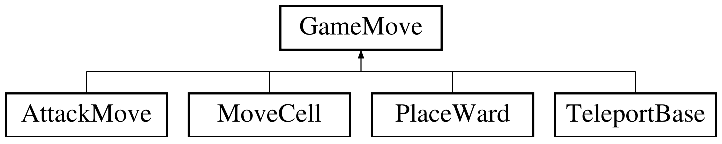
- include/game.hpp
- src/game.cpp

6.17 GameMove Class Reference

abstract class that is the base for all gamemoves

`#include <gamemoves.hpp>`

Inheritance diagram for GameMove:



Public Member Functions

- GameMove ()

  constructor that sets the gamemoves cell to a nullptr
- virtual ∼GameMove ()=default

  the default destructor, doesn't need to free the cell, as its not his responsibility
- bool is_complete () const

  checks if the gamemove is complete, returns true if it is
- virtual Cell ∗ position_cell () const

  gets the current position cell of the gamemove
- virtual void finish (Cell ∗cell_)

  finishes the gamemove, by giving it the cell to use
- int get_movepoints () const
- void set_movepoints (int points_)
- virtual void do_move (Champion ∗champ, std::shared_ptr< Map > map)=0

  does the move with the champ on the map
- virtual bool check_gamemove_addable (Player ∗current_player, Champion ∗selected_champ)

  checks if the gamemove is addable or not to the selected champion
- virtual bool changes_pos () const

  checks if this move changes entities position
- virtual std::string get_state_info () const

  gets this gamemoves state information
- std::string get_formatted_info (const std::string &name) const

  returns a specialized standard formatted output string

Protected Member Functions

- Cell ∗ get_cell () const

  gets the cell of this gamemove
- void set_cell (Cell ∗cell_)

  sets the cell given in params

6.17.1 Detailed Description

abstract class that is the base for all gamemoves

6.17.2 Constructor & Destructor Documentation

6.17.2.1 GameMove()

```
GameMove::GameMove ( )  [inline]
```

constructor that sets the gamemoves cell to a nullptr

6.17.2.2 ∼GameMove()

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

the default destructor, doesn't need to free the cell, as its not his responsibility

6.17.3 Member Function Documentation

6.17.3.1 changes_pos()

```
virtual bool GameMove::changes_pos ( ) const  [inline], [virtual]
```

checks if this move changes entities position

Returns

true if this gamemove changes the entities position

Reimplemented in TeleportBase, and MoveCell.

6.17.3.2 check_gamemove_addable()

```
bool GameMove::check_gamemove_addable (
            Player * current_player,
            Champion * selected_champ )  [virtual]
```

checks if the gamemove is addable or not to the selected champion

Parameters

current_player	the currently selected player
selected_champ	the currently selected champion

6.17.3.3 do_move()

```
virtual void GameMove::do_move (
            Champion * champ,
            std::shared_ptr< Map > map ) [pure virtual]
```

does the move with the champ on the map

Parameters

champ	the champ whose move it is
map	the map to do the moves on

Implemented in TeleportBase, PlaceWard, AttackMove, and MoveCell.

6.17.3.4 finish()

```
virtual void GameMove::finish (
            Cell * cell_ ) [inline], [virtual]
```

finishes the gamemove, by giving it the cell to use

Reimplemented in AttackMove.

6.17.3.5 get_cell()

```
Cell* GameMove::get_cell ( ) const [inline], [protected]
```

gets the cell of this gamemove

Returns

the cell

6.17.3.6 get_formatted_info()

```
std::string GameMove::get_formatted_info (
            const std::string & name ) const
```

returns a specialized standard formatted output string

Parameters

name	the name of the derived gamemoe

Returns

6.17.3.7 get_movepoints()

```
int GameMove::get_movepoints ( ) const  [inline]
```

the amount of points needed to perform this action

Returns

the points

6.17.3.8 get_state_info()

```
std::string GameMove::get_state_info ( ) const  [virtual]
```

gets this gamemoves state information

Returns

Reimplemented in TeleportBase, PlaceWard, AttackMove, and MoveCell.

6.17.3.9 is_complete()

```
bool GameMove::is_complete ( ) const  [inline]
```

checks if the gamemove is complete, returns true if it is

6.17.3.10 position_cell()

```
virtual Cell* GameMove::position_cell ( ) const  [inline], [virtual]
```

gets the current position cell of the gamemove

6.17.3.11 set_cell()

```
void GameMove::set_cell (
            Cell * cell_ )  [inline], [protected]
```

sets the cell given in params

Parameters

cell←_	the new cell

6.17.3.12 set_movepoints()

```
void GameMove::set_movepoints (
             int points_ )  [inline]
```

sets the points

Parameters

points←_	the new points value

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

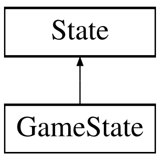
- include/gamemoves.hpp
- src/gamemoves.cpp

6.18 GameState Class Reference

the state that is responsible for navigating through a game

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

Inheritance diagram for GameState:



Public Member Functions

- GameState (StateManager &state_manager, std::vector< Champion ∗ > p1champs, std::vector< Champion ∗ > p2champs, Settings &settings, sf::RenderWindow &window)

    constructs the gamestate
- ∼GameState () override
- void handle_events (sf::Event &e) override

    handles the given event
- void update () override

updates the states
- void draw (sf::RenderWindow &window) override

    draws the state's contents to the given window
- void onclick_movecell ()

    handles onclick of the movecell gamemove button
- void onclick_attack ()

    handles onclick of the attack gamemove button
- void onclick_base ()

    handles onclick of the base gamemove button
- void onclick_ward ()

    handles onclick of the ward gamemove button
- void onclick_item (Item ∗selected_item)

    handles onclick of the item box, if the champ can buy the item, then it does
- void onclick_reset_gamemove ()

    removes the last gamemove of the selected champion, if he has an unfinished one
- void after_gamemove ()

    after a gamemove which does move the player, this should be called to update the surroundings and the player
- bool is_gamemove_finisher (Cell ∗clicked_cell)

    checks if clicking on the given cell at the current state of the game finishes the last move of a champion
- void end_turn ()

    ends the current player's turn
- void show_cell_info (sf::Vector2f index)

    show's cell information at the given index
- void show_stats (std::vector< std::string > &stats)

    shows statistics on the window
- void next_player ()

    sets the current_player to the next one

Public Attributes

- std::function< void()> create_simulation

    should be called if a simulation state is needed

Additional Inherited Members

6.18.1 Detailed Description

the state that is responsible for navigating through a game

6.18.2 Constructor & Destructor Documentation

6.18.2.1 GameState()

```
GameState::GameState (
            StateManager & state_manager,
            std::vector< Champion * > p1champs,
            std::vector< Champion * > p2champs,
            Settings & settings,
            sf::RenderWindow & window )
```

constructs the gamestate

Parameters

state_manager	the state manager of the application
p1champs	the champions of the first player
p2champs	the champions of the second player
settings	the application settings
window	a reference to the window

6.18.2.2 ∼GameState()

```
GameState::∼GameState ( )  [override]
```

6.18.3 Member Function Documentation

6.18.3.1 after_gamemove()

```
void GameState::after_gamemove ( )
```

after a gamemove which does move the player, this should be called to update the surroundings and the player

6.18.3.2 draw()

```
void GameState::draw (
             sf::RenderWindow & window )  [override], [virtual]
```

draws the state's contents to the given window

Parameters

window	the window to draw to

Implements State.

6.18.3.3 end_turn()

```
void GameState::end_turn ( )
```

ends the current player's turn

6.18.3.4 handle_events()

```
void GameState::handle_events (
            sf::Event & event )  [override], [virtual]
```

handles the given event

Parameters

event	the event to be handled

Implements State.

6.18.3.5 is_gamemove_finisher()

```
bool GameState::is_gamemove_finisher (
            Cell * clicked_cell )
```

checks if clicking on the given cell at the current state of the game finishes the last move of a champion

Parameters

clicked_cell	the cell clicked on

Returns

true if this action should finish a gamemove of a champ

6.18.3.6 next_player()

```
void GameState::next_player ( )
```

sets the current_player to the next one

6.18.3.7 onclick_attack()

```
void GameState::onclick_attack ( )
```

handles onclick of the attack gamemove button

6.18.3.8 onclick_base()

```
void GameState::onclick_base ( )
```

handles onclick of the base gamemove button

6.18.3.9 onclick_item()

```
void GameState::onclick_item (
             Item * selected_item )
```

handles onclick of the item box, if the champ can buy the item, then it does

6.18.3.10 onclick_movecell()

```
void GameState::onclick_movecell ( )
```

handles onclick of the movecell gamemove button

6.18.3.11 onclick_reset_gamemove()

```
void GameState::onclick_reset_gamemove ( )
```

removes the last gamemove of the selected champion, if he has an unfinished one

6.18.3.12 onclick_ward()

```
void GameState::onclick_ward ( )
```

handles onclick of the ward gamemove button

6.18.3.13 show_cell_info()

```
void GameState::show_cell_info (
             sf::Vector2f index )
```

show's cell information at the given index

Parameters

index	

6.18.3.14 show_stats()

```
void GameState::show_stats (
            std::vector< std::string > & stats )
```

shows statistics on the window

Parameters

stats	the statistics to show

6.18.3.15 update()

```
void GameState::update ( ) [override], [virtual]
```

updates the states

Implements State.

6.18.4 Member Data Documentation

6.18.4.1 create_simulation

```
std::function<void()> GameState::create_simulation
```

should be called if a simulation state is needed

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

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

6.19 UI::Grid Class Reference

the grid holds multiple grid elements, and places them in a given way

```
#include <UIcomponents.hpp>
```

Public Member Functions

- Grid (sf::Vector2f start_pos, sf::Vector2f margin_, sf::Vector2f direction_={1, 0})

  constructs a grid with the given params
- void set_elements (std::vector< GridElement ∗ > elements_)

  set's this grid's elements
- void set_elements_pos ()

  set's the elements position by calculating it with the its properties
- bool contains (int x, int y) const

  checks if the given coordinates are inside the grid
- sf::FloatRect get_global_bounds () const

  gets the global bounds of the rectangle

6.19.1 Detailed Description

the grid holds multiple grid elements, and places them in a given way

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Grid()

```
Grid::Grid (
             sf::Vector2f start_pos,
             sf::Vector2f margin_,
             sf::Vector2f direction_ = {1, 0} )
```

constructs a grid with the given params

Parameters

start_pos	the start position of the grid relative to the window
margin↩_	the margin between grid elements
direction↩_	the direction of the grid, it is the vector the grid goes towards while placing elements

6.19.3 Member Function Documentation

6.19.3.1 contains()

```
bool Grid::contains (
             int x,
             int y ) const
```

checks if the given coordinates are inside the grid

Parameters

x	x coordinate
y	y coordinate

Returns

true if they're inside, false if not

6.19.3.2 get_global_bounds()

```
sf::FloatRect Grid::get_global_bounds ( ) const
```

gets the global bounds of the rectangle

Returns

the rectangle

6.19.3.3 set_elements()

```
void Grid::set_elements (
             std::vector< GridElement * > elements_ )
```

set's this grid's elements

Parameters

elements↩ _	

6.19.3.4 set_elements_pos()

```
void Grid::set_elements_pos ( )
```

set's the elements position by calculating it with the its properties

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

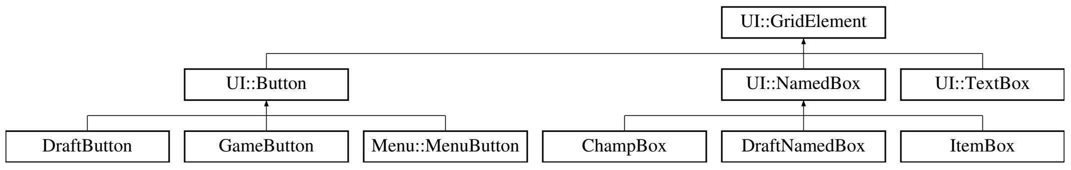
- include/UIcomponents.hpp
- src/UIcomponents.cpp

6.20 UI::GridElement Class Reference

the base class for grid elements

`#include <UIcomponents.hpp>`

Inheritance diagram for UI::GridElement:



Public Member Functions

- virtual ~GridElement ()=default
- virtual void draw (sf::RenderWindow &window)=0

  tells the gridelement to draw itself to the window
- virtual void set_position (sf::Vector2f pos)=0

  set's the position of the grid element relative to the window
- virtual bool contains (int x, int y) const =0

  checks if the given coordinates are inside the grid element
- virtual sf::Vector2f get_size ()=0

  get's the size of the grid element

6.20.1 Detailed Description

the base class for grid elements

6.20.2 Constructor & Destructor Documentation

6.20.2.1 ~GridElement()

```
virtual UI::GridElement::~GridElement ( )  [virtual], [default]
```

6.20.3 Member Function Documentation

6.20.3.1 contains()

```
virtual bool UI::GridElement::contains (
                int x,
                int y ) const  [pure virtual]
```

checks if the given coordinates are inside the grid element

Parameters

x	x coordinate
y	y coordinate

Returns

true if they're inside, false if not

Implemented in UI::NamedBox, UI::TextBox, and UI::Button.

6.20.3.2 draw()

```
virtual void UI::GridElement::draw (
             sf::RenderWindow & window )  [pure virtual]
```

tells the gridelement to draw itself to the window

Parameters

window	

Implemented in UI::NamedBox, UI::TextBox, and UI::Button.

6.20.3.3 get_size()

```
virtual sf::Vector2f UI::GridElement::get_size ( )  [pure virtual]
```

get's the size of the grid element

Returns

the size

Implemented in UI::NamedBox, UI::TextBox, and UI::Button.

6.20.3.4 set_position()

```
virtual void UI::GridElement::set_position (
             sf::Vector2f pos )  [pure virtual]
```

set's the position of the grid element relative to the window

Parameters

pos	

Implemented in UI::NamedBox, UI::TextBox, and UI::Button.

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

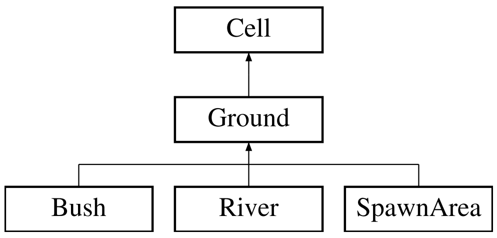
- include/UIcomponents.hpp

6.21 Ground Class Reference

the basic cell type, that can be moved on by the player

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

Inheritance diagram for Ground:



Public Member Functions

- Ground ()

6.21.1 Detailed Description

the basic cell type, that can be moved on by the player

6.21.2 Constructor & Destructor Documentation

6.21.2.1 Ground()

```
Ground::Ground ( )
```

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

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

6.22 Resources::Holder Class Reference

the class which holdes the resources for the application

```
#include <resources.hpp>
```

Public Member Functions

- void load (Type type, const sf::String &filename)

  loads the given resources
- sf::Font & get (Type type)

6.22.1 Detailed Description

the class which holdes the resources for the application

6.22.2 Member Function Documentation

6.22.2.1 get()

```
sf::Font & Holder::get (
            Type type )
```

6.22.2.2 load()

```
void Holder::load (
            Type type,
            const sf::String & filename )
```

loads the given resources

Parameters

type	the type of the resource
filename	the path to the resource

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

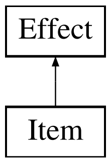
- include/resources.hpp
- src/resources.cpp

6.23 Item Class Reference

the class that describes items, which are primarily used to give bonuses to champions (could be used on entities too if needed)

```
#include <gameobjects.hpp>
```

Inheritance diagram for Item:



Public Member Functions

- Item (std::string name_, int gold_, double bonus_dmg_, double bonus_hp_)
- int get_gold_value () const

  gets the amount of gold needed to purchase this item from the shop
- std::string get_name () const

  gets this items name

6.23.1 Detailed Description

the class that describes items, which are primarily used to give bonuses to champions (could be used on entities too if needed)

6.23.2 Constructor & Destructor Documentation

6.23.2.1 Item()

```
Item::Item (
            std::string name_,
            int gold_,
            double bonus_dmg_,
            double bonus_hp_ )
```

6.23.3 Member Function Documentation

6.23.3.1 get_gold_value()

```
int Item::get_gold_value ( ) const  [inline]
```

gets the amount of gold needed to purchase this item from the shop

6.23.3.2 get_name()

```
std::string Item::get_name ( ) const  [inline]
```

gets this items name

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

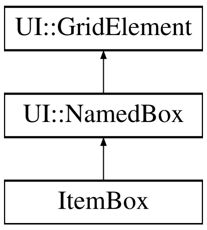
- include/gameobjects.hpp
- src/gameobjects.cpp

6.24 ItemBox Class Reference

a specialized namedbox class that holds an item

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

Inheritance diagram for ItemBox:



Public Member Functions

- ItemBox (const std::string &label, sf::RectangleShape frame, Resources::Holder &holder, Item ∗item)
    constructs an itembox
- Item ∗ get_item () const
    gets the held item

Additional Inherited Members

6.24.1 Detailed Description

a specialized namedbox class that holds an item

6.24.2 Constructor & Destructor Documentation

6.24.2.1 ItemBox()

```
ItemBox::ItemBox (
            const std::string & label,
            sf::RectangleShape frame,
            Resources::Holder & holder,
            Item * item )
```

constructs an itembox

Parameters

label	the label on the box
frame	the frame to put the box inside
holder	the resource holder
item	the item to put inside

6.24.3 Member Function Documentation

6.24.3.1 get_item()

```
Item* ItemBox::get_item ( ) const  [inline]
```

gets the held item

Returns

the item

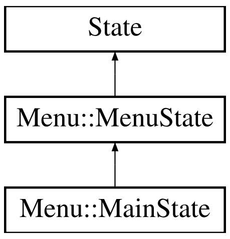
The documentation for this class was generated from the following files:

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

6.25 Menu::MainState Class Reference

```
#include <menu.hpp>
```

Inheritance diagram for Menu::MainState:



Public Member Functions

- MainState (StateManager &s, sf::RenderWindow &window, Settings setting)
- ∼MainState () override
- void handle_events (sf::Event &event) override

  handles the given event
- void draw (sf::RenderWindow &window) override

  draws the state's contents to the given window

Additional Inherited Members

6.25.1 Constructor & Destructor Documentation

6.25.1.1 MainState()

```
Menu::MainState::MainState (
             StateManager & s,
             sf::RenderWindow & window,
             Settings setting )
```

6.25.1.2 ∼MainState()

```
Menu::MainState::∼MainState ( )  [override]
```

6.25.2 Member Function Documentation

6.25.2.1 draw()

```
void Menu::MainState::draw (
             sf::RenderWindow & window )  [override], [virtual]
```

draws the state's contents to the given window

Parameters

window	the window to draw to

Reimplemented from Menu::MenuState.

6.25.2.2 handle_events()

```
void Menu::MainState::handle_events (
            sf::Event & event )  [override], [virtual]
```

handles the given event

Parameters

event	the event to be handled

Reimplemented from Menu::MenuState.

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

- include/menu.hpp
- src/menu.cpp

6.26 Map Class Reference

the class that describes the map

`#include <map.hpp>`

Public Member Functions

- Map (sf::Vector2f pos)

  set's the map's default properties, and its position on the window
- ∼Map ()

  frees every cell on the map
- void draw (sf::RenderWindow &window)

  tells every cell to draw its contents to the screen
- void spawn (Entity ∗entity, sf::Vector2f pos)

  spawns an entity on the given position to the map !the entity which gets spawned onto the map will be freed up when the map destructs! despawn the given entity if you don't want this to happen
- void de_spawn (Entity ∗entity)

  despawns an entity on the given position to the map
- void update ()

  tell's every one of its cells to update its contents
- Cell ∗ get_clicked_cell (const int x, const int y)

  returns the cell clicked on
- std::vector< Cell ∗ > getnearbycells (sf::Vector2f pos, int distance=1)

  gets the nearby cell pointers into a vector
- sf::Vector2u get_cell_grid_size () const

  gets the amount of cells that are on the map
- Champion ∗ get_selected_champ ()

  gets the currently selected champion
- Cell ∗ getcell (sf::Vector2f pos)

  gets cell at the given position @pos the position

- template<typename P >
  void set_selected_nearby_cells (Champion ∗champ, P pred)

  sets the selected nearby cells which satisfy the predicate function
- void select_accessible_cells (Champion ∗champ)

  selects accessible cells where the champion could move
- void select_attackable_entities (Champion ∗champ)

  select attackable entites, which could be attacked by the champ
- void select_wardable_cells (Champion ∗c)

  select cells where the champion could ward
- bool in_bounds_row (int p)

  checks if the given number is a good index inside the map
- bool in_bounds_col (int p)

  checks if the given number is a good index inside the map
- bool in_bounds (sf::Vector2f)

  checks if the given index is inside the map or in other words valid
- void move (Entity ∗entity, sf::Vector2f from, sf::Vector2f to)

  moves the entity from one cell to another
- void reset_cell_selections ()

  resets every cells selection
- void update_vision ()

  updates the vision of the map
- void update_vision_side (Side side_)

  updates the vision for the appropriate side
- bool check_game_end ()

  checks if the game has concluded
- void disable_vision ()

  disables vision on the map
- void reset_cell_vision ()

  resets all the cells vision to having vision, but doesnt change fields that are selected
- void do_attack ()

  tells every one of its entities to try an attack

6.26.1 Detailed Description

the class that describes the map

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Map()

```
Map::Map (
             sf::Vector2f pos )
```

set's the map's default properties, and its position on the window

Parameters

pos	the position on the window

6.26.2.2 ∼Map()

```
Map::∼Map ( )
```

frees every cell on the map

6.26.3 Member Function Documentation

6.26.3.1 check_game_end()

```
bool Map::check_game_end ( )
```

checks if the game has concluded

Returns

true if game ended

6.26.3.2 de_spawn()

```
void Map::de_spawn (
             Entity * entity )
```

despawns an entity on the given position to the map

Parameters

entity	the entity to de_spawn

6.26.3.3 disable_vision()

```
void Map::disable_vision ( )
```

disables vision on the map

6.26.3.4 do_attack()

```
void Map::do_attack ( )
```

tells every one of its entities to try an attack

6.26.3.5 draw()

```
void Map::draw (
             sf::RenderWindow & window )
```

tells every cell to draw its contents to the screen

Parameters

window	where it draws its contents

6.26.3.6 get_cell_grid_size()

```
sf::Vector2u Map::get_cell_grid_size ( ) const  [inline]
```

gets the amount of cells that are on the map

6.26.3.7 get_clicked_cell()

```
Cell * Map::get_clicked_cell (
             const int x,
             const int y )
```

returns the cell clicked on

Parameters

x	coordinate
y	coordinate

6.26.3.8 get_selected_champ()

```
Champion* Map::get_selected_champ ( )
```

gets the currently selected champion

6.26.3.9 getcell()

```
Cell* Map::getcell (
             sf::Vector2f pos )  [inline]
```

gets cell at the given position @pos the position

6.26.3.10 getnearbycells()

```
std::vector< Cell * > Map::getnearbycells (
             sf::Vector2f pos,
             int distance = 1 )
```

gets the nearby cell pointers into a vector

Parameters

pos	this is the middle cell
distance	this is how far away a cell should be to cound it as nearby

6.26.3.11 in_bounds()

```
bool Map::in_bounds (
             sf::Vector2f index )
```

checks if the given index is inside the map or in other words valid

Returns

true if its inside, false otherwise

6.26.3.12 in_bounds_col()

```
bool Map::in_bounds_col (
             int p )  [inline]
```

checks if the given number is a good index inside the map

Parameters

p	the y coordinate

6.26.3.13 in_bounds_row()

```
bool Map::in_bounds_row (
             int p )  [inline]
```

checks if the given number is a good index inside the map

Parameters

p	the x coordinate

6.26.3.14 move()

```
void Map::move (
             Entity * entity,
             sf::Vector2f from,
             sf::Vector2f to )
```

moves the entity from one cell to another

Parameters

entity	the entity to move
from	the index where to move it from
to	the index where to move it into

6.26.3.15 reset_cell_selections()

```
void Map::reset_cell_selections ( )
```

resets every cells selection

6.26.3.16 reset_cell_vision()

```
void Map::reset_cell_vision ( )
```

resets all the cells vision to having vision, but doesnt change fields that are selected

6.26.3.17 select_accessible_cells()

```
void Map::select_accessible_cells (
             Champion * champ )
```

selects accessible cells where the champion could move

Parameters

champ	the champion to use

6.26.3.18 select_attackable_entities()

```
void Map::select_attackable_entities (
             Champion * champ )
```

select attackable entites, which could be attacked by the champ

Parameters

champ	the champion to use

6.26.3.19 select_wardable_cells()

```
void Map::select_wardable_cells (
             Champion * c )
```

select cells where the champion could ward

Parameters

champ	the champion to use

6.26.3.20 set_selected_nearby_cells()

```
template<typename P >
void Map::set_selected_nearby_cells (
             Champion * champ,
             P pred )
```

sets the selected nearby cells which satisfy the predicate function

Parameters

champ,the	champion which is in the middle
pred	the predicate which should be satisfied by the cell

6.26.3.21 spawn()

```
void Map::spawn (
             Entity * entity,
             sf::Vector2f pos )
```

spawns an entity on the given position to the map !the entity which gets spawned onto the map will be freed up when the map destructs! despawn the given entity if you don't want this to happen

Parameters

entity	the entity to spawn
pos	the position where to spawn it

6.26.3.22 update()

```
void Map::update ( )
```

tell's every one of its cells to update its contents

6.26.3.23 update_vision()

```
void Map::update_vision ( )
```

updates the vision of the map

6.26.3.24 update_vision_side()

```
void Map::update_vision_side (
             Side side_ )  [inline]
```

updates the vision for the appropriate side

Parameters

side↩ _	the side which has vision of the map

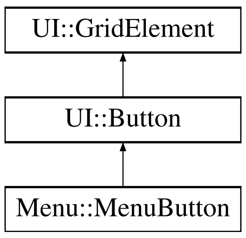
The documentation for this class was generated from the following files:

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

6.27 Menu::MenuButton Class Reference

```
#include <menu.hpp>
```

Inheritance diagram for Menu::MenuButton:



Public Member Functions

- MenuButton (Resources::Holder &h, const sf::String &str, std::function< void()> onclick=[ ]() { std::cout<< "not impl"<< std::endl;})

Additional Inherited Members

6.27.1 Constructor & Destructor Documentation

6.27.1.1 MenuButton()

```
Menu::MenuButton::MenuButton (
            Resources::Holder & h,
            const sf::String & str,
            std::function< void()> onclick = []() { std::cout << "not impl" << std::endl; }
)
```

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

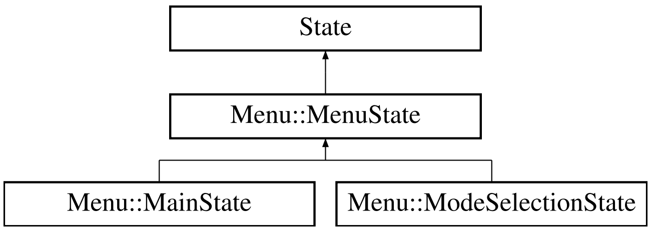
- include/menu.hpp
- src/menu.cpp

6.28 Menu::MenuState Class Reference

the general menustate class, used as a base for simple menus

```
#include <menu.hpp>
```

Inheritance diagram for Menu::MenuState:



Public Member Functions

- MenuState (StateManager &state_manager, Settings setting)
- ∼MenuState () override
- void handle_events (sf::Event &event) override

  handles the given event
- void update () override

  updates the states
- void draw (sf::RenderWindow &window) override

  draws the state's contents to the given window

Protected Attributes

- std::vector< UI::Button ∗ > buttons
- Resources::Holder resources_holder
- Settings setting

Additional Inherited Members

6.28.1 Detailed Description

the general menustate class, used as a base for simple menus

6.28.2 Constructor & Destructor Documentation

6.28.2.1 MenuState()

```
Menu::MenuState::MenuState (
            StateManager & state_manager,
            Settings setting )  [inline], [explicit]
```

6.28.2.2 ∼MenuState()

```
Menu::MenuState::∼MenuState ( )  [override]
```

6.28.3 Member Function Documentation

6.28.3.1 draw()

```
void Menu::MenuState::draw (
            sf::RenderWindow & window )  [override], [virtual]
```

draws the state's contents to the given window

Parameters

window	the window to draw to

Implements State.

Reimplemented in Menu::MainState.

6.28.3.2 handle_events()

```
void Menu::MenuState::handle_events (
            sf::Event & event )  [override], [virtual]
```

handles the given event

Parameters

event	the event to be handled

Implements State.

Reimplemented in Menu::MainState.

6.28.3.3 update()

```
void Menu::MenuState::update ( )  [override], [virtual]
```

updates the states

Implements State.

6.28.4 Member Data Documentation

6.28.4.1 buttons

```
std::vector<UI::Button *> Menu::MenuState::buttons  [protected]
```

6.28.4.2 resources_holder

```
Resources::Holder Menu::MenuState::resources_holder  [protected]
```

6.28.4.3 setting

```
Settings Menu::MenuState::setting  [protected]
```

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

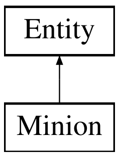
- include/menu.hpp
- src/menu.cpp

6.29 Minion Class Reference

class for minions, which are a type of monsters that go through the lanes, attacking anything that's in front of them

```
#include <gameobjects.hpp>
```

Inheritance diagram for Minion:



Public Member Functions

- Minion (Side side_, std::vector< sf::Vector2f > directions_, Cell *spawn_point)

    set's up the minions attributes (base_hp,dmg)
- bool gives_vision () const override

    describes if the minion gives vision or not
- bool should_focus () const override

    set's if minions should be focused they should be focused by towers, so if a minion is under tower, then it should attack the minion not other entities
- bool gives_creep_score () const override

    describes if the minion increases creep score of the other entity that killed them or not
- void do_move (const std::shared_ptr< Map > &map)

    does one move first it checks if it can attack anything on the next cell it should go to, if not then moves to that cell
- sf::Vector2f get_next_direction_pos_index ()

    get's the next

Additional Inherited Members

6.29.1 Detailed Description

class for minions, which are a type of monsters that go through the lanes, attacking anything that's in front of them

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Minion()

```
Minion::Minion (
            Side side_,
            std::vector< sf::Vector2f > directions_,
            Cell * spawn_point )
```

set's up the minions attributes (base_hp,dmg)

Parameters

*side_*	the team the minion is on
directions←_	the vector of map positions, where the minion should go (top/mid/bot)
spawn_point	the spawn_point of the minion on the minimap

6.29.3 Member Function Documentation

6.29.3.1 do_move()

```
void Minion::do_move (
            const std::shared_ptr< Map > & map )
```

does one move first it checks if it can attack anything on the next cell it should go to, if not then moves to that cell

Parameters

map	the map where it should move on

6.29.3.2 get_next_direction_pos_index()

```
sf::Vector2f Minion::get_next_direction_pos_index ( )
```

get's the next

6.29.3.3 gives_creep_score()

```
bool Minion::gives_creep_score ( ) const  [inline], [override], [virtual]
```

describes if the minion increases creep score of the other entity that killed them or not

Reimplemented from Entity.

6.29.3.4 gives_vision()

```
bool Minion::gives_vision ( ) const  [inline], [override], [virtual]
```

describes if the minion gives vision or not

Reimplemented from Entity.

6.29.3.5 should_focus()

```
bool Minion::should_focus ( ) const  [inline], [override], [virtual]
```

set's if minions should be focused they should be focused by towers, so if a minion is under tower, then it should attack the minion not other entities

Reimplemented from Entity.

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

- include/gameobjects.hpp
- src/gameobjects.cpp

6.30 MinionWave Class Reference

holds a wave of minions, and commands them

```
#include <gameobjects.hpp>
```

Public Member Functions

- MinionWave ()

  initializes the size of minions in a minion wave
- void spawn (sf::Vector2f startpoint, const std::vector< sf::Vector2f > &directions_, const std::shared_ptr< Map > &map, Side side_)

  spawns minions
- void round_end ()

  after the round ended, prepares for the next round
- void do_move (const std::shared_ptr< Map > &map)

  does one move with the minions

6.30.1 Detailed Description

holds a wave of minions, and commands them

6.30.2 Constructor & Destructor Documentation

6.30.2.1 MinionWave()

```
MinionWave::MinionWave ( )  [inline]
```

initializes the size of minions in a minion wave

6.30.3 Member Function Documentation

6.30.3.1 do_move()

```
void MinionWave::do_move (
            const std::shared_ptr< Map > & map )
```

does one move with the minions

Parameters

map	the map it moves on

6.30.3.2 round_end()

```
void MinionWave::round_end ( )
```

after the round ended, prepares for the next round

6.30.3.3 spawn()

```
void MinionWave::spawn (
            sf::Vector2f startpoint,
            const std::vector< sf::Vector2f > & directions_,
            const std::shared_ptr< Map > & map,
            Side side_ )
```

spawns minions

Parameters

startpoint	the point where the minions first appear
directions↩_	the directions where the minions should try to move in
map	the map where they move
*side_*	the side on which the minions are

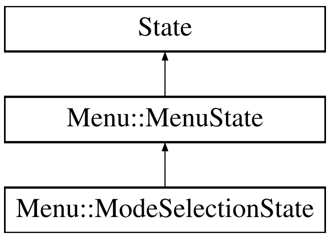
The documentation for this class was generated from the following files:

- include/gameobjects.hpp
- src/gameobjects.cpp

6.31 Menu::ModeSelectionState Class Reference

```
#include <menu.hpp>
```

Inheritance diagram for Menu::ModeSelectionState:



Public Member Functions

- ModeSelectionState (StateManager &state_manager, sf::RenderWindow &window, Settings setting)

Additional Inherited Members

6.31.1 Constructor & Destructor Documentation

6.31.1.1 ModeSelectionState()

```
Menu::ModeSelectionState::ModeSelectionState (
            StateManager & state_manager,
            sf::RenderWindow & window,
            Settings setting )
```

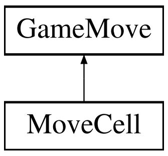
The documentation for this class was generated from the following files:

- include/menu.hpp
- src/menu.cpp

6.32 MoveCell Class Reference

```
#include <gamemoves.hpp>
```

Inheritance diagram for MoveCell:



Public Member Functions

- void do_move (Champion ∗champ, std::shared_ptr< Map > map) override

  does the move with the champ on the map
- std::string get_state_info () const override

  gets this gamemoves state information
- bool changes_pos () const override

  checks if this move changes entities position

Additional Inherited Members

6.32.1 Member Function Documentation

6.32.1.1 changes_pos()

```
bool MoveCell::changes_pos ( ) const  [inline], [override], [virtual]
```

checks if this move changes entities position

Returns

true if this gamemove changes the entities position

Reimplemented from GameMove.

6.32.1.2 do_move()

```
void MoveCell::do_move (
            Champion * champ,
            std::shared_ptr< Map > map )  [override], [virtual]
```

does the move with the champ on the map

Parameters

champ	the champ whose move it is
map	the map to do the moves on

Implements GameMove.

6.32.1.3 get_state_info()

```
std::string MoveCell::get_state_info ( ) const  [override], [virtual]
```

gets this gamemoves state information

Returns

Reimplemented from GameMove.

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

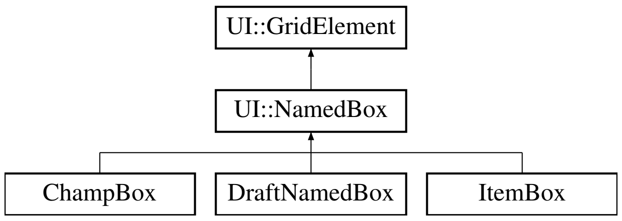
- include/gamemoves.hpp
- src/gamemoves.cpp

6.33 UI::NamedBox Class Reference

the named box, which is a grid element that holds a shape and a text inside of it

```
#include <UIcomponents.hpp>
```

Inheritance diagram for UI::NamedBox:



Public Member Functions

- NamedBox (Resources::Holder &holder)

  constructs a named box
- NamedBox (const std::string &label, sf::RectangleShape frame, Resources::Holder &holder)

  constructs a named box with the given parameters
- void set_position (sf::Vector2f pos) override

  set's the position of the grid element relative to the window
- bool contains (int x, int y) const override

  checks if the given coordinates are inside the grid element
- sf::Vector2f get_size () override

  get's the size of the grid element
- void draw (sf::RenderWindow &window) override

  tells the gridelement to draw itself to the window
- void set_label (const std::string &label_text)

  set's the new text of the named box
- void set_char_size (unsigned size)

  set's the character size of the named box
- void set_label_color (const sf::Color &c)

  set's the label color
- sf::FloatRect get_global_bounds () const

  gets the global bounds of the shape

Protected Attributes

- sf::RectangleShape frame
- sf::Text label

6.33.1 Detailed Description

the named box, which is a grid element that holds a shape and a text inside of it

6.33.2 Constructor & Destructor Documentation

6.33.2.1 NamedBox() [1/2]

```
NamedBox::NamedBox (
             Resources::Holder & holder )  [explicit]
```

constructs a named box

Parameters

holder	the resources holder

6.33.2.2 NamedBox() [2/2]

```
NamedBox::NamedBox (
             const std::string & label,
             sf::RectangleShape frame,
             Resources::Holder & holder )
```

constructs a named box with the given parameters

Parameters

label	the text inside the shape
frame	the shape
holder	the resources holder

6.33.3 Member Function Documentation

6.33.3.1 contains()

```
bool NamedBox::contains (
             int x,
             int y ) const  [override], [virtual]
```

checks if the given coordinates are inside the grid element

Parameters

x	x coordinate
y	y coordinate

Returns

true if they're inside, false if not

Implements UI::GridElement.

6.33.3.2 draw()

```
void NamedBox::draw (
             sf::RenderWindow & window )  [override], [virtual]
```

tells the gridelement to draw itself to the window

Parameters

window	

Implements UI::GridElement.

6.33.3.3 get_global_bounds()

```
sf::FloatRect UI::NamedBox::get_global_bounds ( ) const  [inline]
```

gets the global bounds of the shape

Returns

the rectangle

6.33.3.4 get_size()

```
sf::Vector2f NamedBox::get_size ( )  [override], [virtual]
```

get's the size of the grid element

Returns

the size

Implements UI::GridElement.

6.33.3.5 set_char_size()

```
void NamedBox::set_char_size (
             unsigned size )
```

set's the character size of the named box

Parameters

size	the new size

6.33.3.6 set_label()

```
void NamedBox::set_label (
             const std::string & label_text )
```

set's the new text of the named box

Parameters

label_text	

6.33.3.7 set_label_color()

```
void NamedBox::set_label_color (
             const sf::Color & c )
```

set's the label color

Parameters

c	the color to use

6.33.3.8 set_position()

```
void NamedBox::set_position (
             sf::Vector2f pos )  [override], [virtual]
```

set's the position of the grid element relative to the window

Parameters

pos	

Implements UI::GridElement.

6.33.4 Member Data Documentation

6.33.4.1 frame

```
sf::RectangleShape UI::NamedBox::frame  [protected]
```

6.33.4.2 label

```
sf::Text UI::NamedBox::label  [protected]
```

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

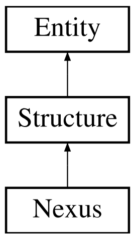
- include/UIcomponents.hpp
- src/UIcomponents.cpp

6.34 Nexus Class Reference

the class for the nexus, which doesn't do damage to entities, but if it dies, the game is over and the team who destroyed it wins

```
#include <gameobjects.hpp>
```

Inheritance diagram for Nexus:



Public Member Functions

- Nexus ()

    set's up the nexuses attributes (base_hp,dmg)

Additional Inherited Members

6.34.1 Detailed Description

the class for the nexus, which doesn't do damage to entities, but if it dies, the game is over and the team who destroyed it wins

6.34.2 Constructor & Destructor Documentation

6.34.2.1 Nexus()

```
Nexus::Nexus ( )
```

set's up the nexuses attributes (base_hp,dmg)

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

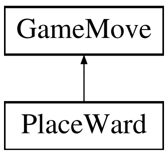
- include/gameobjects.hpp
- src/gameobjects.cpp

6.35 PlaceWard Class Reference

the class that implements the ward placing mechanism

```
#include <gamemoves.hpp>
```

Inheritance diagram for PlaceWard:



Public Member Functions

- PlaceWard ()
- std::string get_state_info () const override

  gets this gamemoves state information
- void do_move (Champion ∗champ, std::shared_ptr< Map > map) override

  does the move with the champ on the map

Additional Inherited Members

6.35.1 Detailed Description

the class that implements the ward placing mechanism

6.35.2 Constructor & Destructor Documentation

6.35.2.1 PlaceWard()

```
PlaceWard::PlaceWard ( )  [inline]
```

6.35.3 Member Function Documentation

6.35.3.1 do_move()

```
void PlaceWard::do_move (
            Champion * champ,
            std::shared_ptr< Map > map )  [override], [virtual]
```

does the move with the champ on the map

Parameters

champ	the champ whose move it is
map	the map to do the moves on

Implements GameMove.

6.35.3.2 get_state_info()

```
std::string PlaceWard::get_state_info ( ) const  [override], [virtual]
```

gets this gamemoves state information

Returns

Reimplemented from GameMove.

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

- include/gamemoves.hpp
- src/gamemoves.cpp

6.36 Player Class Reference

the class that holds everything a player has

```
#include <gameobjects.hpp>
```

Public Member Functions

- Player (std::vector< Champion ∗ > champs)

  constructor for player class, set's it's properties to the appropriate values
- ∼Player ()

  frees heap allocated properties, which are the minion waves and champions
- void spawn_champs (const std::shared_ptr< Map > &map)

  spawns the champions on the map, informs the map about it, and set's the champs cell
- void set_spawn_point (Cell ∗spawn_point_)

  set's the champions spawn point, their base
- void set_champ_icons (const std::string &icons)

  set's the champ icons, in order with each character of the string
- void set_font (Resources::Holder &holder)

  set's all the champion's icons font face
- bool is_gamemove_active () const

  check's if there's a gamemove currently active on any of the champion
- void do_moves (const std::shared_ptr< Map > &map)

  does the moves on each of the champions
- bool is_his_champ (Champion ∗c)

  returns true, if the given champ is his
- bool did_start () const

  returns true, if this player started the game
- void set_starter (bool starter_)

  set's if this player started the game, so it had the first turn
- bool check_round_end ()

  check's if the round ended, returns true if it did goes through the champions, and checks if they have movepoints left or not
- void spawn_minions (const std::shared_ptr< Map > &map)

  spawns minions on to the map
- void round_end (std::shared_ptr< Map > &map)

  ends the round, calls the champions round end methods
- void set_simulation (bool sim)

  set's the simulation state for the player
- void update_champ_positions (const std::shared_ptr< Map > &map)

  update all of its champion position to the appropriate positions on the map
- void set_side (Side side)

  set's the current players side
- Side get_side () const

  returns the players team side
- Champion ∗ get_selected_champs (sf::Vector2f index)

  gets the selected champions on the given map index
- Cell ∗ get_spawn_point () const

  set's the spawn point where the champions should spawn
- void clear_gamemoves ()

  clears all the gamemoves from the champions
- void despawn_from_map (std::shared_ptr< Map > &map)

  despawns the player's entites from the map
- std::string get_gamemoves_state () const

6.36.1 Detailed Description

the class that holds everything a player has

6.36.2 Constructor & Destructor Documentation

6.36.2.1 Player()

```
Player::Player (
            std::vector< Champion * > champs )  [explicit]
```

constructor for player class, set's it's properties to the appropriate values

Parameters

champs	the champions owned by this player

6.36.2.2 ∼Player()

```
Player::∼Player ( )
```

frees heap allocated properties, which are the minion waves and champions

6.36.3 Member Function Documentation

6.36.3.1 check_round_end()

```
bool Player::check_round_end ( )
```

check's if the round ended, returns true if it did goes through the champions, and checks if they have movepoints left or not

6.36.3.2 clear_gamemoves()

```
void Player::clear_gamemoves ( )
```

clears all the gamemoves from the champions

6.36.3.3 despawn_from_map()

```
void Player::despawn_from_map (
             std::shared_ptr< Map > & map )
```

despawns the player's entites from the map

Parameters

map	the map where they should be removed from

6.36.3.4 did_start()

```
bool Player::did_start ( ) const  [inline]
```

returns true, if this player started the game

6.36.3.5 do_moves()

```
void Player::do_moves (
             const std::shared_ptr< Map > & map )
```

does the moves on each of the champions

Parameters

map	the map to do the gamemoves on

6.36.3.6 get_gamemoves_state()

```
std::string Player::get_gamemoves_state ( ) const
```

returns the current gamemoves state informations

Returns

a block of text describing the gamemove state

6.36.3.7 get_selected_champs()

```
Champion * Player::get_selected_champs (
             sf::Vector2f index )
```

gets the selected champions on the given map index

6.36.3.8 get_side()

```
Side Player::get_side ( ) const  [inline]
```

returns the players team side

6.36.3.9 get_spawn_point()

```
Cell* Player::get_spawn_point ( ) const  [inline]
```

set's the spawn point where the champions should spawn

6.36.3.10 is_gamemove_active()

```
bool Player::is_gamemove_active ( ) const
```

check's if there's a gamemove currently active on any of the champion

6.36.3.11 is_his_champ()

```
bool Player::is_his_champ (
             Champion * c )
```

returns true, if the given champ is his

Parameters

c	the champ to check

6.36.3.12 round_end()

```
void Player::round_end (
             std::shared_ptr< Map > & map )
```

ends the round, calls the champions round end methods

Parameters

map	checks if minions should spawn, so it spawns them to the map

6.36.3.13 set_champ_icons()

```
void Player::set_champ_icons (
             const std::string & icons )
```

set's the champ icons, in order with each character of the string

Parameters

icons	the string to get the characters from

6.36.3.14 set_font()

```
void Player::set_font (
             Resources::Holder & holder )
```

set's all the champion's icons font face

Parameters

holder	the object that can be used to retrieve the font face

6.36.3.15 set_side()

```
void Player::set_side (
             Side side )
```

set's the current players side

Parameters

side	the team where the player is on

6.36.3.16 set_simulation()

```
void Player::set_simulation (
             bool sim )
```

set's the simulation state for the player

Parameters

sim	if it's simulation then sim is true

6.36.3.17 set_spawn_point()

```
void Player::set_spawn_point (
            Cell * spawn_point_ )
```

set's the champions spawn point, their base

Parameters

point	the spawnpoint

6.36.3.18 set_starter()

```
void Player::set_starter (
            bool starter_ )  [inline]
```

set's if this player started the game, so it had the first turn

Parameters

starter	true if this player was the starter, otherwise false

6.36.3.19 spawn_champs()

```
void Player::spawn_champs (
            const std::shared_ptr< Map > & map )
```

spawns the champions on the map, informs the map about it, and set's the champs cell

Parameters

map	the map where they spawn

6.36.3.20 spawn_minions()

```
void Player::spawn_minions (
            const std::shared_ptr< Map > & map )
```

spawns minions on to the map

Parameters

map	the map to spawn to

6.36.3.21 update_champ_positions()

```
void Player::update_champ_positions (
            const std::shared_ptr< Map > & map )
```

update all of its champion position to the appropriate positions on the map

Parameters

map	the map is needed to determine where a given cell is

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

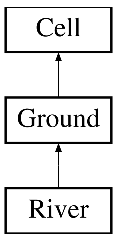
- include/gameobjects.hpp
- src/gameobjects.cpp

6.37 River Class Reference

the only difference from ground is that it has another color

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

Inheritance diagram for River:



Public Member Functions

- River ()

6.37.1 Detailed Description

the only difference from ground is that it has another color

6.37.2 Constructor & Destructor Documentation

6.37.2.1 River()

```
River::River ( )
```

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

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

6.38 Settings Class Reference

the settings class, which holds the applications settings that could be needed at any state

```
#include <statemanagement.hpp>
```

Public Member Functions

- Settings (std::string champs_filepath, std::string items_filepath, std::string output_prefix, GameMode mode)

  constructs a simple settings object
- std::string get_champs_filepath () const
- std::string get_items_filepath () const
- std::string get_output_prefix () const
- void set_champs_filepath (std::string path)

  sets the champions filepath
- void set_items_filepath (std::string path)

  sets the champions filepath
- void set_output_prefix (std::string prefix)

  sets the output prefix
- void set_gamemode (GameMode mode_)

  sets the new gamemode

6.38.1 Detailed Description

the settings class, which holds the applications settings that could be needed at any state

6.38.2 Constructor & Destructor Documentation

6.38.2.1 Settings()

```
Settings::Settings (
            std::string champs_filepath,
            std::string items_filepath,
            std::string output_prefix,
            GameMode mode )
```

constructs a simple settings object

Parameters

champs_filepath	the filepath to the champions textfile
items_filepath	the filepath to the items textfile
output_prefix	the prefix of the output textfile
mode	the gamemode of the applicatino

6.38.3 Member Function Documentation

6.38.3.1 get_champs_filepath()

```
std::string Settings::get_champs_filepath ( ) const  [inline]
```

Returns

the champions filepath

6.38.3.2 get_items_filepath()

```
std::string Settings::get_items_filepath ( ) const  [inline]
```

Returns

the items filepath

6.38.3.3 get_output_prefix()

```
std::string Settings::get_output_prefix ( ) const  [inline]
```

Returns

the output prefix

6.38.3.4 set_champs_filepath()

```
void Settings::set_champs_filepath (
             std::string path )  [inline]
```

sets the champions filepath

Parameters

path	the new path to use

6.38.3.5 set_gamemode()

```
void Settings::set_gamemode (
             GameMode mode_ )  [inline]
```

sets the new gamemode

Parameters

mode	the new gamemode

6.38.3.6 set_items_filepath()

```
void Settings::set_items_filepath (
             std::string path )  [inline]
```

sets the champions filepath

Parameters

path	the new path to use

6.38.3.7 set_output_prefix()

```
void Settings::set_output_prefix (
             std::string prefix )  [inline]
```

sets the output prefix

Parameters

path	the new prefix to use

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

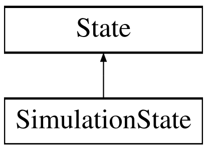
- include/statemanagement.hpp
- src/statemanagement.cpp

6.39 SimulationState Class Reference

the state that implements the simulation

```
#include <simulation.hpp>
```

Inheritance diagram for SimulationState:



Public Member Functions

- SimulationState (std::vector< Player * > &players, std::shared_ptr< Map > &map, sf::RenderWindow &window, Settings &settings, StateManager &state_manager, std::ofstream &output_file_, std::function< void()> callback_=[ ](){})

  constructs the simulation state with the given params
- ∼SimulationState () override
- void handle_events (sf::Event &event) override

  handles the given event
- void update () override

  updates the states
- void draw (sf::RenderWindow &window) override

  draws the state's contents to the given window

Additional Inherited Members

6.39.1 Detailed Description

the state that implements the simulation

6.39.2 Constructor & Destructor Documentation

6.39.2.1 SimulationState()

```
SimulationState::SimulationState (
             std::vector< Player * > & players,
             std::shared_ptr< Map > & map,
             sf::RenderWindow & window,
             Settings & settings,
             StateManager & state_manager,
             std::ofstream & output_file_,
             std::function< void()> callback_ = [](){} )
```

constructs the simulation state with the given params

Parameters

players	the players of the game
map	the map where they play
window	the window
settings	the current settings of the game
state_manager	the state_manager
*callback_*	the callback function to be called, when this state ends

6.39.2.2 ∼SimulationState()

```
SimulationState::∼SimulationState ( )  [override]
```

6.39.3 Member Function Documentation

6.39.3.1 draw()

```
void SimulationState::draw (
             sf::RenderWindow & window )  [override], [virtual]
```

draws the state's contents to the given window

Parameters

window	the window to draw to

Implements State.

6.39.3.2 handle_events()

```
void SimulationState::handle_events (
            sf::Event & event )  [override], [virtual]
```

handles the given event

Parameters

event	the event to be handled

Implements State.

6.39.3.3 update()

```
void SimulationState::update ( )  [override], [virtual]
```

updates the states

Implements State.

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

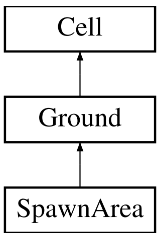
- include/simulation.hpp
- src/simulation.cpp

6.40 SpawnArea Class Reference

spawn area, where champions spawn

`#include <map.hpp>`

Inheritance diagram for SpawnArea:



Public Member Functions

- SpawnArea ()

6.40.1 Detailed Description

spawn area, where champions spawn

6.40.2 Constructor & Destructor Documentation

6.40.2.1 SpawnArea()

```
SpawnArea::SpawnArea ( )
```

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

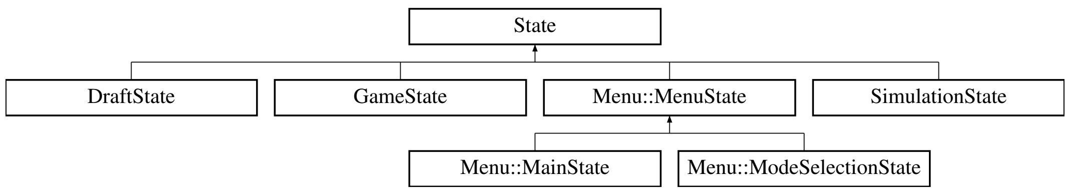
- include/map.hpp
- src/map.cpp

6.41 State Class Reference

the abstract State class which is used to handle one state

```
#include <statemanagement.hpp>
```

Inheritance diagram for State:



Public Member Functions

- virtual ~State ()=default
- virtual void handle_events (sf::Event &event)=0

  handles the given event
- virtual void update ()=0

  updates the states
- virtual void draw (sf::RenderWindow &window)=0

  draws the state's contents to the given window

Protected Member Functions

- State (StateManager &state_manager)

Protected Attributes

- StateManager & state_manager

6.41.1 Detailed Description

the abstract State class which is used to handle one state

6.41.2 Constructor & Destructor Documentation

6.41.2.1 ∼State()

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

6.41.2.2 State()

```
State::State (
             StateManager & state_manager )  [inline], [explicit], [protected]
```

6.41.3 Member Function Documentation

6.41.3.1 draw()

```
virtual void State::draw (
             sf::RenderWindow & window )  [pure virtual]
```

draws the state's contents to the given window

Parameters

window	the window to draw to

Implemented in SimulationState, Menu::MainState, Menu::MenuState, GameState, and DraftState.

6.41.3.2 handle_events()

```
virtual void State::handle_events (
            sf::Event & event )  [pure virtual]
```

handles the given event

Parameters

event	the event to be handled

Implemented in SimulationState, Menu::MainState, Menu::MenuState, DraftState, and GameState.

6.41.3.3 update()

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

updates the states

Implemented in SimulationState, Menu::MenuState, GameState, and DraftState.

6.41.4 Member Data Documentation

6.41.4.1 state_manager

```
StateManager& State::state_manager  [protected]
```

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

- include/statemanagement.hpp

6.42 StateManager Class Reference

the class that handles the state management for this application

```
#include <statemanagement.hpp>
```

Public Member Functions

- StateManager ()

  default constructor for the state manager
- void change_state (std::unique_ptr< State > state)

  buffers the given state, so at the end of the main loop it gets changed to this That means the current state will get removed, and replaced by this one
- void push_state (std::unique_ptr< State > state)

  buffers the given state, so at the end of the main loop it gets on top of the other states That means the already existing states stay intact
- void pop_state ()

  buffers a pop state event until the main loop has ended This means the upmost state gets removed and the one under it becomes the current state
- void update_state ()

  updates the internal states variable according to the buffer and the last commanded action
- void handle_events (sf::RenderWindow &window) const

  handles the events of the application, and calls the appropriate method of the current state
- void update ()

  calls the current state's update method
- void draw (sf::RenderWindow &window)

  tells the current state to draw its contents to the window
- bool has_state () const
- void exit ()

  tells all its states to exit

Static Public Member Functions

- static sf::Vector2f get_size (sf::RenderWindow &window)

  gets the size of the window in sf::Vector2f type

6.42.1 Detailed Description

the class that handles the state management for this application

6.42.2 Constructor & Destructor Documentation

6.42.2.1 StateManager()

```
StateManager::StateManager ( )  [inline]
```

default constructor for the state manager

6.42.3 Member Function Documentation

6.42.3.1 change_state()

```
void StateManager::change_state (
             std::unique_ptr< State > state )
```

buffers the given state, so at the end of the main loop it gets changed to this That means the current state will get removed, and replaced by this one

Parameters

state	the state to change to

6.42.3.2 draw()

```
void StateManager::draw (
            sf::RenderWindow & window )
```

tells the current state to draw its contents to the window

Parameters

window	the window to draw to

6.42.3.3 exit()

```
void StateManager::exit ( )
```

tells all its states to exit

6.42.3.4 get_size()

```
sf::Vector2f StateManager::get_size (
            sf::RenderWindow & window )  [static]
```

gets the size of the window in sf::Vector2f type

Parameters

window	the window whose size will be calculated

Returns

the size of the window

6.42.3.5 handle_events()

```
void StateManager::handle_events (
            sf::RenderWindow & window ) const
```

handles the events of the application, and calls the appropriate method of the current state

Parameters

window	the window that gives the events

6.42.3.6 has_state()

```
bool StateManager::has_state ( ) const  [inline]
```

checks if there are states

Returns

true if there are, false if not

6.42.3.7 pop_state()

```
void StateManager::pop_state ( )
```

buffers a pop state event until the main loop has ended This means the upmost state gets removed and the one under it becomes the current state

6.42.3.8 push_state()

```
void StateManager::push_state (
             std::unique_ptr< State > state )
```

buffers the given state, so at the end of the main loop it gets on top of the other states That means the already existing states stay intact

Parameters

state	

6.42.3.9 update()

```
void StateManager::update ( )
```

calls the current state's update method

6.42.3.10 update_state()

```
void StateManager::update_state ( )
```

updates the internal states variable according to the buffer and the last commanded action

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

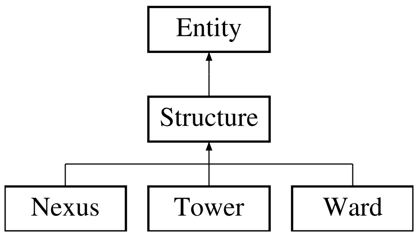
- include/statemanagement.hpp
- src/statemanagement.cpp

6.43 Structure Class Reference

common parent class for structures, it shouldn't have a move (as in map movements) functions, it's position doesn't change

```
#include <gameobjects.hpp>
```

Inheritance diagram for Structure:



Additional Inherited Members

6.43.1 Detailed Description

common parent class for structures, it shouldn't have a move (as in map movements) functions, it's position doesn't change

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

- include/gameobjects.hpp

6.44 TeamCol Class Reference

```
#include <draft.hpp>
```

Public Member Functions

- TeamCol (Resources::Holder &holder, sf::Vector2f start_pos_, sf::Vector2f size={100, 30}, float margin=10)

  constructs the draftnamedbox with the correct design
- void set_position ()

  sets the position of the team column
- void draw_to_window (sf::RenderWindow &window)

  draws the teamcol to the window
- size_t champs_size () const

  gets the champions list size
- Champion $*$ operator[ ] (size_t index)

  gets the champion at the given index
- std::vector$<$ Champion $*$ $>$ & get_champs ()

  returns the champ list

6.44.1 Detailed Description

class that holds a column of champions

6.44.2 Constructor & Destructor Documentation

6.44.2.1 TeamCol()

```
TeamCol::TeamCol (
            Resources::Holder & holder,
            sf::Vector2f start_pos_,
            sf::Vector2f size = {100,30},
            float margin = 10 )
```

constructs the draftnamedbox with the correct design

Parameters

holder	the object that can get the font face for the component
size	the size of the teamcol
margin	the margin between the elements

6.44.3 Member Function Documentation

6.44.3.1 champs_size()

```
size_t TeamCol::champs_size ( ) const  [inline]
```

gets the champions list size

6.44.3.2 draw_to_window()

```
void TeamCol::draw_to_window (
             sf::RenderWindow & window )
```

draws the teamcol to the window

6.44.3.3 get_champs()

```
std::vector<Champion *>& TeamCol::get_champs ( )  [inline]
```

returns the champ list

6.44.3.4 operator[]()

```
Champion* TeamCol::operator[] (
             size_t index )  [inline]
```

gets the champion at the given index

Parameters

index	the champ at this index

6.44.3.5 set_position()

```
void TeamCol::set_position ( )
```

sets the position of the team column

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

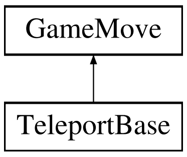
- include/draft.hpp
- src/draft.cpp

6.45 TeleportBase Class Reference

the class that implements the teleport to the base gamemove

```
#include <gamemoves.hpp>
```

Inheritance diagram for TeleportBase:



Public Member Functions

- TeleportBase ()
- std::string get_state_info () const override

    gets this gamemoves state information
- bool changes_pos () const override

    checks if this move changes entities position
- void do_move (Champion ∗champ, std::shared_ptr< Map > map) override

    does the move with the champ on the map

Additional Inherited Members

6.45.1 Detailed Description

the class that implements the teleport to the base gamemove

6.45.2 Constructor & Destructor Documentation

6.45.2.1 TeleportBase()

```
TeleportBase::TeleportBase ( )  [inline]
```

6.45.3 Member Function Documentation

6.45.3.1 changes_pos()

```
bool TeleportBase::changes_pos ( ) const  [inline], [override], [virtual]
```

checks if this move changes entities position

Returns

true if this gamemove changes the entities position

Reimplemented from GameMove.

6.45.3.2 do_move()

```
void TeleportBase::do_move (
            Champion * champ,
            std::shared_ptr< Map > map )  [override], [virtual]
```

does the move with the champ on the map

Parameters

champ	the champ whose move it is
map	the map to do the moves on

Implements GameMove.

6.45.3.3 get_state_info()

```
std::string TeleportBase::get_state_info ( ) const  [override], [virtual]
```

gets this gamemoves state information

Returns

Reimplemented from GameMove.

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

- include/gamemoves.hpp
- src/gamemoves.cpp

6.46 gtest_lite::Test Struct Reference

```
#include <gtest_lite.h>
```

Public Member Functions

- Test ()
- void begin (const char ∗n)

  Teszt kezdete.
- void end ()

  Teszt vége.
- bool fail ()
- std::ostream & expect (bool st, const char ∗file, int line, const char ∗expr)

  Eredményt adminisztráló tagfüggvény True a jó eset.
- std::ostream & tstatus (bool st, const char ∗file, int line)

  Eredményt adminisztráló tagfüggvény True a jó eset, mindig ír.
- ∼Test ()

  Destruktor.

Public Attributes

- int sum

  tesztek számlálója
- int failed

  hibás tesztek
- bool status

  éppen futó teszt státusza.
- bool tmp

  temp a kivételkezeléshez;
- std::string name

  éppen futó teszt neve.
- std::fstream null

  nyelő, ha nem kell kiírni semmit

6.46.1 Detailed Description

Tesztek állapotát tároló osztály. Egyetlen egy statikus példány keletkezik, aminek a destruktora a futás végén hívódik meg.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 Test()

```
gtest_lite::Test::Test ( )  [inline]
```

6.46.2.2 ∼Test()

```
gtest_lite::Test::∼Test ( ) [inline]
```

Destruktor.

6.46.3 Member Function Documentation

6.46.3.1 begin()

```
void gtest_lite::Test::begin (
            const char * n ) [inline]
```

Teszt kezdete.

6.46.3.2 end()

```
void gtest_lite::Test::end ( ) [inline]
```

Teszt vége.

6.46.3.3 expect()

```
std::ostream& gtest_lite::Test::expect (
            bool st,
            const char * file,
            int line,
            const char * expr ) [inline]
```

Eredményt adminisztráló tagfüggvény True a jó eset.

6.46.3.4 fail()

```
bool gtest_lite::Test::fail ( ) [inline]
```

6.46.3.5 tstatus()

```
std::ostream& gtest_lite::Test::tstatus (
             bool st,
             const char * file,
             int line )  [inline]
```

Eredményt adminisztráló tagfüggvény True a jó eset, mindig ír.

6.46.4 Member Data Documentation

6.46.4.1 failed

```
int gtest_lite::Test::failed
```

hibás tesztek

6.46.4.2 name

```
std::string gtest_lite::Test::name
```

éppen futó teszt neve.

6.46.4.3 null

```
std::fstream gtest_lite::Test::null
```

nyelő, ha nem kell kiírni semmit

6.46.4.4 status

```
bool gtest_lite::Test::status
```

éppen futó teszt státusza.

6.46.4.5 sum

`int gtest_lite::Test::sum`

tesztek számlálója

6.46.4.6 tmp

`bool gtest_lite::Test::tmp`

temp a kivételkezeléshez;

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

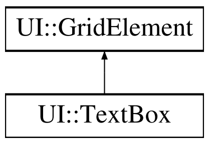
- test/gtest_lite.h

6.47 UI::TextBox Class Reference

the textbox element, which is a rectangle where text you can input text into

`#include <UIcomponents.hpp>`

Inheritance diagram for UI::TextBox:



Public Member Functions

- TextBox (const std::string &label, Resources::Holder &holder, sf::Vector2f pos={0, 0}, const std::string &text↩_default="")

    constructs a textbox with the given params
- void draw (sf::RenderWindow &window) override

    tells the gridelement to draw itself to the window
- void set_position (sf::Vector2f pos) override

    set's the position of the grid element relative to the window
- sf::Vector2f get_size () override

    get's the size of the grid element
- bool contains (int x, int y) const override

    checks if the given coordinates are inside the grid element
- sf::FloatRect get_global_bounds () const

    gets the global bounds of the shape
- void set_selected (bool s)

    set's the textbox selected, that means it takes in input text
- bool get_is_selected () const

    checks if the textbox is selected
- void add_char (char c)

    adds the given character to the rectangles text
- void remove_char ()

    removes the last character from the text
- std::string get_text () const

    gets the textbox's inside text

6.47.1 Detailed Description

the textbox element, which is a rectangle where text you can input text into

6.47.2 Constructor & Destructor Documentation

6.47.2.1 TextBox()

```
TextBox::TextBox (
            const std::string & label,
            Resources::Holder & holder,
            sf::Vector2f pos = {0, 0},
            const std::string & text_default = "" )
```

constructs a textbox with the given params

Parameters

label	the name of the textbox, this is placed outside the rectangle, showing what the textbox is for
holder	the resources holder
pos	the position relative to the window
text_default	the default text inside the rectangle

6.47.3 Member Function Documentation

6.47.3.1 add_char()

```
void TextBox::add_char (
            char c )
```

adds the given character to the rectangles text

Parameters

c	the char to add

6.47.3.2 contains()

```
bool TextBox::contains (
            int x,
            int y ) const  [override], [virtual]
```

checks if the given coordinates are inside the grid element

Parameters

x	x coordinate
y	y coordinate

Returns

true if they're inside, false if not

Implements UI::GridElement.

6.47.3.3 draw()

```
void TextBox::draw (
            sf::RenderWindow & window )  [override], [virtual]
```

tells the gridelement to draw itself to the window

Parameters

window	

Implements UI::GridElement.

6.47.3.4 get_global_bounds()

```
sf::FloatRect UI::TextBox::get_global_bounds ( ) const  [inline]
```

gets the global bounds of the shape

Returns

the rectangle

6.47.3.5 get_is_selected()

```
bool UI::TextBox::get_is_selected ( ) const  [inline]
```

checks if the textbox is selected

Returns

true if it is selected, false otherwise

6.47.3.6 get_size()

```
sf::Vector2f UI::TextBox::get_size ( )  [inline], [override], [virtual]
```

get's the size of the grid element

Returns

the size

Implements UI::GridElement.

6.47.3.7 get_text()

```
std::string UI::TextBox::get_text ( ) const  [inline]
```

gets the textbox's inside text

Returns

the text

6.47.3.8 remove_char()

```
void TextBox::remove_char ( )
```

removes the last character from the text

6.47.3.9 set_position()

```
void UI::TextBox::set_position (
             sf::Vector2f pos )  [inline], [override], [virtual]
```

set's the position of the grid element relative to the window

Parameters

pos	

Implements UI::GridElement.

6.47.3.10 set_selected()

```
void UI::TextBox::set_selected (
            bool s )  [inline]
```

set's the textbox selected, that means it takes in input text

Parameters

s	true if the textbox got selected, false otherwise

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

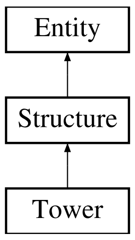
- include/UIcomponents.hpp
- src/UIcomponents.cpp

6.48 Tower Class Reference

the class for a tower, which damages other entities that come near it

```
#include <gameobjects.hpp>
```

Inheritance diagram for Tower:



Public Member Functions

- Tower ()

  set's up the towers attributes (base_hp,dmg)
- void attack (Map *map) override

  checks if it can attack anyone in it's range, but first checks if there are entities that should be focused

Additional Inherited Members

6.48.1 Detailed Description

the class for a tower, which damages other entities that come near it

6.48.2 Constructor & Destructor Documentation

6.48.2.1 Tower()

```
Tower::Tower ( )
```

set's up the towers attributes (base_hp,dmg)

6.48.3 Member Function Documentation

6.48.3.1 attack()

```
void Tower::attack (
            Map * map )  [override], [virtual]
```

checks if it can attack anyone in it's range, but first checks if there are entities that should be focused

Parameters

map	the map where it searches for entities

Reimplemented from Entity.

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

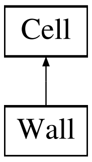
- include/gameobjects.hpp
- src/gameobjects.cpp

6.49 Wall Class Reference

can't be moved on to by entities

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

Inheritance diagram for Wall:



Public Member Functions

- Wall ()
- bool can_ward_here () const override

    true if champions are able to put wards on this spot
- bool can_move_here () const override

    true if entities are able to move here

6.49.1 Detailed Description

can't be moved on to by entities

6.49.2 Constructor & Destructor Documentation

6.49.2.1 Wall()

```
Wall::Wall ( )
```

6.49.3 Member Function Documentation

6.49.3.1 can_move_here()

```
bool Wall::can_move_here ( ) const  [inline], [override], [virtual]
```

true if entities are able to move here

Reimplemented from Cell.

6.49.3.2 can_ward_here()

```
bool Wall::can_ward_here ( ) const  [inline], [override], [virtual]
```

true if champions are able to put wards on this spot

Reimplemented from Cell.

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

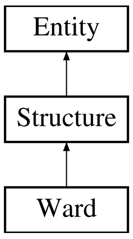
- include/map.hpp
- src/map.cpp

6.50 Ward Class Reference

The ward is a type of structure (as it cannot move), that gives vision, but expires after a given time intervall.

```
#include <gameobjects.hpp>
```

Inheritance diagram for Ward:



Public Member Functions

- Ward ()

    default contsructor initializes the ward's cooldown and it's color
- void do_move ()

    does a gamemove with the ward, which means check if it expired yet
- std::vector< std::string > get_stats () const override

    gets the stats of this ward

Additional Inherited Members

6.50.1 Detailed Description

The ward is a type of structure (as it cannot move), that gives vision, but expires after a given time intervall.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 Ward()

```
Ward::Ward ( )
```

default contsructor initializes the ward's cooldown and it's color

6.50.3 Member Function Documentation

6.50.3.1 do_move()

```
void Ward::do_move ( )
```

does a gamemove with the ward, which means check if it expired yet

6.50.3.2 get_stats()

```
std::vector< std::string > Ward::get_stats ( ) const  [override], [virtual]
```

gets the stats of this ward

Returns

Reimplemented from Entity.

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

- include/gameobjects.hpp
- src/gameobjects.cpp

Chapter 7

File Documentation

7.1 include/draft.hpp File Reference

```
#include "UIcomponents.hpp"
#include "game.hpp"
#include "gameobjects.hpp"
#include "ioparser.h"
#include "resources.hpp"
#include "statemanagement.hpp"
#include <fstream>
#include <vector>
```

Classes

- class DraftTurn

  class used to store one draft turn
- class DraftNamedBox

  class that specializes NamedBox, to a NamedBox with the correct design
- class TeamCol
- class DraftButton

  class that specializes Button, to a draftbutton with the correct design
- class ChampBox

  a champion box implementation, which holds a champ
- class DraftState

7.2 include/game.hpp File Reference

```
#include "UIcomponents.hpp"
#include "gameobjects.hpp"
#include "ioparser.h"
#include "map.hpp"
#include "resources.hpp"
#include "simulation.hpp"
#include "statemanagement.hpp"
```

```
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <utility>
#include <vector>
```

Classes

- class ItemBox

  a specialized namedbox class that holds an item
- class GameButton

  a button that has a specific style used for game buttons
- class GameState

  the state that is responsible for navigating through a game

7.3 include/gamemoves.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <memory>
```

Classes

- class GameMove

  abstract class that is the base for all gamemoves
- class MoveCell
- class AttackMove

  the class that implements the attack move
- class PlaceWard

  the class that implements the ward placing mechanism
- class TeleportBase

  the class that implements the teleport to the base gamemove

7.4 include/gameobjects.hpp File Reference

```
#include "gamemoves.hpp"
#include "map.hpp"
#include "resources.hpp"
#include <SFML/Graphics.hpp>
#include <iostream>
#include <sstream>
#include <list>
#include <memory>
#include <utility>
```

Classes

- class Effect
- class Entity

  The class that describes an entity.
- class Item

  the class that describes items, which are primarily used to give bonuses to champions (could be used on entities too if needed)
- class Structure

  common parent class for structures, it shouldn't have a move (as in map movements) functions, it's position doesn't change
- class Ward

  The ward is a type of structure (as it cannot move), that gives vision, but expires after a given time intervall.
- class Champion

  class for describing champions, they're a type of entities that the players can manipulate with gamemoves
- class Tower

  the class for a tower, which damages other entities that come near it
- class Nexus

  the class for the nexus, which doesn't do damage to entities, but if it dies, the game is over and the team who destroyed it wins
- class Camp

  a common class for camps which are not able to move (baron nashor, drakes and jungle camps) because of how the game works, every camp can give an effect to the champion(s) that slain it
- class Drake

  the class that describes dragons, there are different types of dragons, with different effects (currently only two)
- class Minion

  class for minions, which are a type of monsters that go through the lanes, attacking anything that's in front of them
- class MinionWave

  holds a wave of minions, and commands them
- class Player

  the class that holds everything a player has

Enumerations

- enum class Side { BLUE , RED , NEUTRAL }

  the enum that holds which team the entity is on

7.4.1 Enumeration Type Documentation

7.4.1.1 Side

```
enum Side  [strong]
```

the enum that holds which team the entity is on

Enumerator

BLUE	
RED	
NEUTRAL	

7.5 include/ioparser.h File Reference

```
#include <vector>
#include <string>
#include <sstream>
#include <fstream>
#include "gameobjects.hpp"
```

Classes

- class IOParser::File

  a file holder that closes the file

Namespaces

- IOParser

Functions

- std::vector< std::string > IOParser::split_string (const std::string &str, char delimiter)
- Champion ∗ IOParser::create_champ (const std::string &line)
- Item IOParser::create_item (const std::string &line)

7.6 include/map.hpp File Reference

```
#include "gameobjects.hpp"
#include <vector>
#include <filesystem>
#include <memory>
#include <fstream>
#include <array>
```

Classes

- class Cell

  the base class for a cell on the map
- class Ground

  the basic cell type, that can be moved on by the player
- class River

  the only difference from ground is that it has another color
- class Wall

  can't be moved on to by entities
- class Bush

  calculates vision differently than the ground object
- class SpawnArea

  spawn area, where champions spawn
- class Map

  the class that describes the map

7.7 include/menu.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Window/Mouse.hpp>
#include <utility>
#include "UIcomponents.hpp"
#include "draft.hpp"
#include "resources.hpp"
```

Classes

- class Menu::MenuState

  the general menustate class, used as a base for simple menus
- class Menu::MainState
- class Menu::ModeSelectionState
- class Menu::MenuButton

Namespaces

- Menu

7.8 include/resources.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <map>
#include <memory>
```

Classes

- class Resources::Holder

  the class which holdes the resources for the application

Namespaces

- Resources

Enumerations

- enum class Resources::Type { Resources::FONT }

  the types of resources there are

7.9 include/simulation.hpp File Reference

```
#include "UIcomponents.hpp"
#include "gameobjects.hpp"
#include "map.hpp"
#include "resources.hpp"
#include "statemanagement.hpp"
#include "ioparser.h"
#include <vector>
```

Classes

- class SimulationState

  the state that implements the simulation

7.10 include/statemanagement.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <memory>
#include <stack>
#include <utility>
```

Classes

- class StateManager

  the class that handles the state management for this application
- class State

  the abstract State class which is used to handle one state
- class Settings

  the settings class, which holds the applications settings that could be needed at any state

Enumerations

- enum class GameMode { TWO_PLAYER }

    the gamemode of the game

7.10.1 Enumeration Type Documentation

7.10.1.1 GameMode

```
enum GameMode  [strong]
```

the gamemode of the game

Enumerator

TWO_PLAYER	

7.11 include/UIcomponents.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <functional>
#include <iostream>
#include <vector>
#include "resources.hpp"
#include "statemanagement.hpp"
```

Classes

- class UI::GridElement

    the base class for grid elements
- class UI::Button

    the button class which implements a shape with some text on it, with an onclick method
- class UI::TextBox

    the textbox element, which is a rectangle where text you can input text into
- class UI::Grid

    the grid holds multiple grid elements, and places them in a given way
- class UI::NamedBox

    the named box, which is a grid element that holds a shape and a text inside of it

Namespaces

- UI

7.12 src/draft.cpp File Reference

```
#include "../include/draft.hpp"
```

Functions

- void onclick_back (StateManager &s)

7.12.1 Function Documentation

7.12.1.1 onclick_back()

```
void onclick_back (
            StateManager & s )
```

7.13 src/game.cpp File Reference

```
#include "../include/game.hpp"
```

7.14 src/gamemoves.cpp File Reference

```
#include "../include/gamemoves.hpp"
#include "../include/map.hpp"
```

7.15 src/gameobjects.cpp File Reference

```
#include "../include/gameobjects.hpp"
#include "SFML/Graphics/Color.hpp"
#include <cstdlib>
#include <utility>
```

7.16 src/ioparser.cpp File Reference

```
#include "../include/ioparser.h"
```

Namespaces

- IOParser

Functions

- std::vector< std::string > IOParser::split_string (const std::string &str, char delimiter)
- Champion ∗ IOParser::create_champ (const std::string &line)
- Item IOParser::create_item (const std::string &line)

7.17 src/main.cpp File Reference

```
#include <SFML/Graphics.hpp>
#include "../include/menu.hpp"
```

Functions

- int main ()

7.17.1 Function Documentation

7.17.1.1 main()

```
int main ( )
```

7.18 src/map.cpp File Reference

```
#include "../include/map.hpp"
```

7.19 src/menu.cpp File Reference

```
#include <utility>
#include "../include/menu.hpp"
```

Namespaces

- Menu

7.20 src/resources.cpp File Reference

```
#include "../include/resources.hpp"
```

7.21 src/simulation.cpp File Reference

```
#include <utility>
#include "../include/simulation.hpp"
```

7.22 src/statemanagement.cpp File Reference

```
#include <utility>
#include "../include/statemanagement.hpp"
```

7.23 src/UIcomponents.cpp File Reference

```
#include <utility>
#include "../include/UIcomponents.hpp"
```

7.24 test/gtest_lite.h File Reference

```
#include <iostream>
#include <cassert>
#include <cmath>
#include <cstring>
#include <limits>
#include <string>
#include <fstream>
```

Classes

- struct gtest_lite::Test

Namespaces

- gtest_lite

  gtest_lite: a keretrendszer függvényinek és objektumainak névtere

Macros

- #define TEST(C, N) { gtest_lite::test.begin(#C"."#N);
- #define END gtest_lite::test.end(); }

  Teszteset vége.
- #define SUCCEED() gtest_lite::test.tstatus(true, __FILE__, __LINE__)

  Sikeres teszt makrója.
- #define FAIL() gtest_lite::test.tstatus(false, __FILE__, __LINE__)

  Sikertelen teszt makrója.
- #define EXPECT_EQ(expected, actual) EXPECTCMP((expected) == (actual), expected, actual)

  Azonosságot elváró makró
- #define EXPECT_NE(expected, actual) EXPECTNE((expected) != (actual), expected, actual)

  Eltérést elváró makró
- #define EXPECT_TRUE(actual) EXPECTCMP(actual, "true", actual)

  Igaz értéket elváró makró
- #define EXPECT_FALSE(actual) EXPECTCMP(!(actual), "false", actual)

  Hamis értéket elváró makró
- #define EXPECT_DOUBLE_EQ(expected, actual) EXPECTCMP(gtest_lite::almostEQ(expected, actual), expected, actual)

  Valós számok azonosságát elváró makró
- #define EXPECT_STREQ(expected, actual)

  C stringek (const char ∗) azonosságát tesztelő makró
- #define EXPECT_STRNE(expected, actual)

  C stringek (const char ∗) eltéréset tesztelő makró
- #define EXPECT_THROW(statement, exception_type)

  Kivételt várunk.
- #define EXPECT_THROW_THROW(statement, exception_type)

  Kivételt várunk és továbbdobjuk – ilyen nincs a gtest-ben.
- #define EXPECT_NO_THROW(statement)

  Nem várunk kivételt.
- #define CREATE_Has_(X)
- #define EXPECT(expr, msg) gtest_lite::test.expect(expr, __FILE__, __LINE__, #msg)

  EXPECT: makró, hogy könnyen lecserélhető legyen.
- #define EXPECTEXP(expr, exp, act)

  EXPECTEXP: általános kifejezés kiértékelése.
- #define EXPECTCMP(expr, exp, act)

  EXPECTCMP: összehasonlítás.
- #define EXPECTNE(expr, exp, act)

  EXPECTNE: összehasonlítás.
- #define EXPECTTHROW(statement, exp, act)

  EXPECTTHROW: kivételkezelés.
- #define GTINIT(IS)
- #define GTEND(os)

Functions

- void hasMember (...)

  Segédfüggvény egy publikus adattag, vagy tagfüggvény létezésének tesztelésére fordítási időben.
- bool gtest_lite::almostEQ (double a, double b)

7.24.1 Detailed Description

Google gtest keretrendszerhez hasonló rendszer. Sz.I. 2015., 2016., 2017. (_Has_X)

A tesztelés legalapvetőbb funkcióit támogató függvények és makrók. Nem szálbiztos megvalósítás. Szabadon felhasználható, bővíthető.

Használati példa: Teszteljük az f(x)=2∗x függvényt: int f(int x) { return 2∗x; }

int main() { TEST(TeszEsetNeve, TesztNeve) EXPECT_EQ(0, f(0)); EXPECT_EQ(4, f(2)) << "A függvény hibás eredményt adott" << std::endl; ... END ...

A működés részleteinek megértése szorgalmi feladat.

7.24.2 Macro Definition Documentation

7.24.2.1 CREATE_Has_

```
#define CREATE_Has_(
              X )
```

Value:
```
template<typename T> struct Has_##X {  \
    struct Fallback { int X; };         \
    struct Derived : T, Fallback {};    \
    template<typename C, C> struct ChT; \
    template<typename D> static char (&f(ChT<int Fallback::*, &D::X>*))[1]; \
    template<typename D> static char (&f(...))[2]; \
    static bool const member = sizeof(f<Derived>(0)) == 2; \
};
```

Segédmakró egy adattag, vagy tagfüggvény létezésének tesztelésére futási időben Ötlet: https↩ ://cpptalk.wordpress.com/2009/09/12/substitution-failure-is-not-an-error-2

7.24.2.2 END

```
#define END gtest_lite::test.end(); }
```

Teszteset vége.

7.24.2.3 EXPECT

```
#define EXPECT(
              expr,
              msg ) gtest_lite::test.expect(expr, __FILE__, __LINE__, #msg)
```

EXPECT: makró, hogy könnyen lecserélhető legyen.
Belső megvalósításhoz tartozó makrók, és osztályok.

7.24.2.4 Nem célszerű közvetlenül használni, vagy módosítani

7.24.2.5 EXPECT_DOUBLE_EQ

```
#define EXPECT_DOUBLE_EQ(
                expected,
                actual ) EXPECTCMP(gtest_lite::almostEQ(expected, actual), expected, actual)
```

Valós számok azonosságát elváró makró

7.24.2.6 EXPECT_EQ

```
#define EXPECT_EQ(
                expected,
                actual ) EXPECTCMP((expected) == (actual), expected, actual)
```

Azonosságot elváró makró

7.24.2.7 EXPECT_FALSE

```
#define EXPECT_FALSE(
                actual ) EXPECTCMP(!(actual), "false", actual)
```

Hamis értéket elváró makró

7.24.2.8 EXPECT_NE

```
#define EXPECT_NE(
                expected,
                actual ) EXPECTNE((expected) != (actual), expected, actual)
```

Eltérést elváró makró

7.24.2.9 EXPECT_NO_THROW

```
#define EXPECT_NO_THROW(
                statement )
```

Value:

```
    try { gtest_lite::test.tmp = true; statement; } \
    catch (...) { gtest_lite::test.tmp = false; }\
    EXPECTTHROW(statement, "nem dob kivetelt.", "kivetelt dobott.")
```

Nem várunk kivételt.

7.24.2.10 EXPECT_STREQ

```
#define EXPECT_STREQ(
                expected,
                actual )
```

Value:

```
    ((actual != NULL) ? \
    EXPECTCMP(strcmp(expected, actual) == 0, expected, actual) : \
    EXPECT(false, "STR_EQ NULL pointert kapott!"))
```

C stringek (const char ∗) azonosságát tesztelő makró

7.24.2.11 EXPECT_STRNE

```
#define EXPECT_STRNE(
                expected,
                actual )
```

Value:

```
    ((actual != NULL) ? \
    EXPECTNE(strcmp(expected, actual) != 0, expected, actual) : \
    EXPECT(false, "STR_EQ NULL pointert kapott!"))
```

C stringek (const char ∗) eltérését tesztelő makró

7.24.2.12 EXPECT_THROW

```
#define EXPECT_THROW(
                statement,
                exception_type )
```

Value:

```
    try { gtest_lite::test.tmp = false; statement; } \
    catch (exception_type) { gtest_lite::test.tmp = true; } \
    catch (...) { } \
    EXPECTTHROW(statement, "kivetelt dob.", "nem dobott '"#exception_type"' kivetelt.")
```

Kivételt várunk.

7.24.2.13 EXPECT_THROW_THROW

```
#define EXPECT_THROW_THROW(
                statement,
                exception_type )
```

Value:

```
    try { gtest_lite::test.tmp = false; statement; } \
    catch (exception_type) { gtest_lite::test.tmp = true; throw; } \
    EXPECTTHROW(statement, "kivetelt dob.", "nem dobott '"#exception_type"' kivetelt.")
```

Kivételt várunk és továbbdobjuk – ilyen nincs a gtest-ben.

7.24.2.14 EXPECT_TRUE

```
#define EXPECT_TRUE(
                actual ) EXPECTCMP(actual, "true", actual)
```

Igaz értéket elváró makró

7.24.2.15 EXPECTCMP

```
#define EXPECTCMP(
                expr,
                exp,
                act )
```

Value:

```
    gtest_lite::test.expect(expr, __FILE__, __LINE__, #act) \
    « "**A(z) '"#act « "'kifejezes\n**   erteke: " « std::boolalpha « (act) \
    « "\n**   elvart: " « (exp) « std::endl
```

EXPECTCMP: összehasonlítás.

7.24.2.16 EXPECTEXP

```
#define EXPECTEXP(
                expr,
                exp,
                act )
```

Value:

```
    gtest_lite::test.expect(expr, __FILE__, __LINE__, #expr) \
    « "**A(z) '"#act « "'kifejezes\n**   erteke: " « std::boolalpha « (act) \
    « "\n**   elvart: " « (exp) « std::endl
```

EXPECTEXP: általános kifejezés kiértékelése.

7.24.2.17 EXPECTNE

```
#define EXPECTNE(
                expr,
                exp,
                act )
```

Value:

```
    gtest_lite::test.expect(expr, __FILE__, __LINE__, #act) \
    « "**A(z) '"#act « "'kifejezes\n**   erteke: " « std::boolalpha « (act) \
    « "\n**   elvart, hogy nem: " « (exp) « std::endl
```

EXPECTNE: összehasonlítás.

7.24.2.18 EXPECTTHROW

```
#define EXPECTTHROW(
                statement,
                exp,
                act )
```

Value:

```
    gtest_lite::test.expect(gtest_lite::test.tmp, __FILE__, __LINE__, #statement) \
    « "**Az utasitas " « (act) \
    « "\n**Azt vartuk, hogy " « (exp) « std::endl
```

EXPECTTHROW: kivételkezelés.

7.24.2.19 FAIL

```
#define FAIL( ) gtest_lite::test.tstatus(false, __FILE__, __LINE__)
```

Sikertelen teszt makrója.

7.24.2.20 GTEND

```
#define GTEND(
                os )
```

7.24.2.21 GTINIT

```
#define GTINIT(
                IS )
```

7.24.2.22 SUCCEED

```
#define SUCCEED( ) gtest_lite::test.tstatus(true, __FILE__, __LINE__)
```

Sikeres teszt makrója.

7.24.2.23 TEST

```
#define TEST(
                C,
                N ) { gtest_lite::test.begin(#C"."#N);
```

Teszt kezdete. A makró paraméterezése hasonlít a gtest paraméterezéséhez. Így az itt elkészített testek könnyen átemelhetők a gtest keretrendszerbe.

Parameters

C	- tezteset neve (csak a gtest kompatibilitás miatt van külön neve az eseteknek)
N	- teszt neve

7.24.3 Function Documentation

7.24.3.1 hasMember()

```
void hasMember (
              ... )
```

Segédfüggvény egy publikus adattag, vagy tagfüggvény létezésének tesztelésére fordítási időben.

7.25 test/main_test.cpp File Reference

```
#include <iostream>
#include "gtest_lite.h"
#include "../include/gameobjects.hpp"
#include "../include/ioparser.h"
#include "../include/menu.hpp"
#include <vector>
```

Functions

- bool champexiststest (std::vector< Champion * > &champs, const std::string &name)
- int main ()

7.25.1 Function Documentation

7.25.1.1 champexiststest()

```
bool champexiststest (
            std::vector< Champion * > & champs,
            const std::string & name )
```

7.25.1.2 main()

```
int main ( )
```

Index