

lolmacrogame

Generated by Doxygen 1.9.1

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	9
4.1 File List	9
5 Namespace Documentation	11
5.1 gtest_lite Namespace Reference	11
5.1.1 Detailed Description	11
5.1.2 Function Documentation	11
5.1.2.1 almostEQ()	11
5.2 IOParser Namespace Reference	12
5.2.1 Function Documentation	12
5.2.1.1 create_champ()	12
5.2.1.2 create_item()	12
5.2.1.3 split_string()	12
5.3 Menu Namespace Reference	12
5.4 Resources Namespace Reference	13
5.4.1 Enumeration Type Documentation	13
5.4.1.1 Type	13
5.5 UI Namespace Reference	13
6 Class Documentation	15
6.1 AttackMove Class Reference	15
6.1.1 Detailed Description	15
6.1.2 Constructor & Destructor Documentation	15
6.1.2.1 AttackMove()	16
6.1.3 Member Function Documentation	16
6.1.3.1 do_move()	16
6.1.3.2 finish()	16
6.1.3.3 get_state_info()	16
6.2 Bush Class Reference	17
6.2.1 Detailed Description	17
6.2.2 Constructor & Destructor Documentation	17
6.2.2.1 Bush()	17
6.3 UI::Button Class Reference	17
6.3.1 Detailed Description	18
6.3.2 Constructor & Destructor Documentation	18

6.3.2.1 Button() [1/2]	18
6.3.2.2 Button() [2/2]	18
6.3.3 Member Function Documentation	19
6.3.3.1 contains()	19
6.3.3.2 draw()	19
6.3.3.3 get_global_bounds()	20
6.3.3.4 get_size()	20
6.3.3.5 onclick_here()	20
6.3.3.6 set_position()	20
6.3.3.7 update_text_position()	21
6.3.4 Member Data Documentation	21
6.3.4.1 onclick	21
6.3.4.2 shape	21
6.3.4.3 text	21
6.4 Camp Class Reference	21
6.4.1 Detailed Description	22
6.4.2 Constructor & Destructor Documentation	22
6.4.2.1 Camp()	22
6.4.3 Member Function Documentation	22
6.4.3.1 get_buff_given()	22
6.4.3.2 respawn()	23
6.4.3.3 set_effect()	23
6.5 Cell Class Reference	23
6.5.1 Detailed Description	25
6.5.2 Constructor & Destructor Documentation	25
6.5.2.1 Cell()	25
6.5.2.2 ~Cell()	25
6.5.3 Member Function Documentation	25
6.5.3.1 add_entity()	25
6.5.3.2 can_attack_entity()	26
6.5.3.3 can_buy_items()	26
6.5.3.4 can_move_here()	26
6.5.3.5 can_ward_here()	26
6.5.3.6 contains()	26
6.5.3.7 do_attack()	27
6.5.3.8 draw()	27
6.5.3.9 get_attackable_entity()	27
6.5.3.10 get_entity_clicked()	27
6.5.3.11 get_first_entity()	28
6.5.3.12 get_index()	28
6.5.3.13 is_selected()	28
6.5.3.14 remove_entity()	28

6.5.3.15 reset_selection_color()	29
6.5.3.16 reset_vision_color()	29
6.5.3.17 set_color()	29
6.5.3.18 set_highlighted()	29
6.5.3.19 set_position()	29
6.5.3.20 set_selected()	30
6.5.3.21 set_shop()	30
6.5.3.22 set_vision()	30
6.5.3.23 should_update_vision_around()	30
6.5.3.24 unselect()	31
6.5.3.25 update()	31
6.5.3.26 update_entities_shape()	31
6.5.3.27 update_shape()	31
6.6 ChampBox Class Reference	32
6.6.1 Detailed Description	32
6.6.2 Constructor & Destructor Documentation	32
6.6.2.1 ChampBox()	32
6.6.3 Member Function Documentation	33
6.6.3.1 get_champ()	33
6.7 Champion Class Reference	33
6.7.1 Detailed Description	35
6.7.2 Constructor & Destructor Documentation	35
6.7.2.1 Champion() [1/2]	35
6.7.2.2 Champion() [2/2]	35
6.7.2.3 ~Champion()	36
6.7.3 Member Function Documentation	36
6.7.3.1 add_gamemove()	36
6.7.3.2 add_item()	36
6.7.3.3 add_xp()	36
6.7.3.4 can_fight_back()	37
6.7.3.5 clear_gamemoves()	37
6.7.3.6 current_gamemove_index()	37
6.7.3.7 despawn_wards()	37
6.7.3.8 do_move()	37
6.7.3.9 draw()	38
6.7.3.10 fight()	38
6.7.3.11 finish_gamemove()	38
6.7.3.12 get_current_gamemove_state_info()	39
6.7.3.13 get_max_hp()	39
6.7.3.14 get_name()	39
6.7.3.15 get_simulation_cell()	39
6.7.3.16 get_stats()	39

6.7.3.17	get_total_dmg()	40
6.7.3.18	getmovepoints()	40
6.7.3.19	gives_vision()	40
6.7.3.20	is_gamemove_complete()	40
6.7.3.21	killed_other()	40
6.7.3.22	last_gamemove_index()	41
6.7.3.23	move()	41
6.7.3.24	place_ward()	41
6.7.3.25	remove_last_gamemove()	41
6.7.3.26	round_end()	42
6.7.3.27	set_font()	42
6.7.3.28	set_icon()	42
6.7.3.29	set_side()	42
6.7.3.30	set_simulation()	43
6.7.3.31	set_spawn_point()	43
6.7.3.32	update_shape_pos()	43
6.8	DraftButton Class Reference	44
6.8.1	Detailed Description	44
6.8.2	Constructor & Destructor Documentation	44
6.8.2.1	DraftButton()	44
6.9	DraftNamedBox Class Reference	45
6.9.1	Detailed Description	45
6.9.2	Constructor & Destructor Documentation	45
6.9.2.1	DraftNamedBox()	45
6.10	DraftState Class Reference	46
6.10.1	Constructor & Destructor Documentation	46
6.10.1.1	DraftState()	46
6.10.1.2	~DraftState()	47
6.10.2	Member Function Documentation	47
6.10.2.1	dont_ban()	47
6.10.2.2	draw()	47
6.10.2.3	handle_events()	47
6.10.2.4	lock_in()	48
6.10.2.5	update()	48
6.11	DraftTurn Class Reference	48
6.11.1	Detailed Description	48
6.11.2	Constructor & Destructor Documentation	49
6.11.2.1	DraftTurn()	49
6.11.3	Member Function Documentation	49
6.11.3.1	do_turn()	49
6.11.3.2	is_ban_phase()	49
6.12	Drake Class Reference	50

6.12.1 Detailed Description	50
6.12.2 Constructor & Destructor Documentation	50
6.12.2.1 Drake()	50
6.12.3 Member Function Documentation	50
6.12.3.1 decide_which_type()	51
6.12.3.2 respawn()	51
6.13 Effect Class Reference	51
6.13.1 Constructor & Destructor Documentation	52
6.13.1.1 Effect()	52
6.13.2 Member Function Documentation	52
6.13.2.1 get_bonus_dmg()	52
6.13.2.2 get_bonus_hp()	52
6.13.2.3 not_zero()	52
6.13.2.4 set_bonus_dmg()	53
6.13.2.5 set_bonus_hp()	53
6.13.2.6 update_expire()	53
6.14 Entity Class Reference	53
6.14.1 Detailed Description	55
6.14.2 Constructor & Destructor Documentation	55
6.14.2.1 ~Entity()	55
6.14.2.2 Entity()	56
6.14.3 Member Function Documentation	56
6.14.3.1 attack()	56
6.14.3.2 can_fight_back()	56
6.14.3.3 check_death()	56
6.14.3.4 clicked()	57
6.14.3.5 draw()	57
6.14.3.6 get_buff_given()	57
6.14.3.7 get_current_hp()	57
6.14.3.8 get_gold_given()	58
6.14.3.9 get_max_hp()	58
6.14.3.10 get_real_cell()	58
6.14.3.11 get_side()	58
6.14.3.12 get_simulation_cell()	58
6.14.3.13 get_stats()	58
6.14.3.14 get_total_dmg()	59
6.14.3.15 get_xp_given()	59
6.14.3.16 gives_creep_score()	59
6.14.3.17 gives_vision()	59
6.14.3.18 is_alive()	59
6.14.3.19 killed_other()	59
6.14.3.20 refill_hp()	60

6.14.3.21 remove_hp()	60
6.14.3.22 respawn()	60
6.14.3.23 set_cell()	60
6.14.3.24 set_color()	61
6.14.3.25 set_name()	61
6.14.3.26 set_side()	61
6.14.3.27 set_xp_given()	61
6.14.3.28 should_focus()	62
6.14.3.29 to_ui_int_format()	62
6.14.3.30 update_shape_pos()	62
6.14.4 Member Data Documentation	62
6.14.4.1 alive	62
6.14.4.2 base_hp	63
6.14.4.3 cell	63
6.14.4.4 color	63
6.14.4.5 current_hp	63
6.14.4.6 damage	63
6.14.4.7 gold_given	63
6.14.4.8 name	63
6.14.4.9 respawn_counter	63
6.14.4.10 respawn_timer	64
6.14.4.11 shape	64
6.14.4.12 side	64
6.14.4.13 xp_given	64
6.15 IOParser::File Class Reference	64
6.15.1 Detailed Description	64
6.15.2 Constructor & Destructor Documentation	64
6.15.2.1 File()	64
6.15.2.2 ~File()	65
6.15.3 Member Function Documentation	65
6.15.3.1 getfile()	65
6.16 GameButton Class Reference	65
6.16.1 Detailed Description	66
6.16.2 Constructor & Destructor Documentation	66
6.16.2.1 GameButton()	66
6.17 GameMove Class Reference	66
6.17.1 Detailed Description	67
6.17.2 Constructor & Destructor Documentation	67
6.17.2.1 GameMove()	68
6.17.2.2 ~GameMove()	68
6.17.3 Member Function Documentation	68
6.17.3.1 changes_pos()	68

6.17.3.2 check_gamemove_addable()	68
6.17.3.3 do_move()	69
6.17.3.4 finish()	69
6.17.3.5 get_cell()	69
6.17.3.6 get_formatted_info()	69
6.17.3.7 get_movepoints()	70
6.17.3.8 get_state_info()	70
6.17.3.9 is_complete()	70
6.17.3.10 position_cell()	70
6.17.3.11 set_cell()	70
6.17.3.12 set_movepoints()	71
6.18 GameState Class Reference	71
6.18.1 Detailed Description	72
6.18.2 Constructor & Destructor Documentation	72
6.18.2.1 GameState()	72
6.18.2.2 ~GameState()	73
6.18.3 Member Function Documentation	73
6.18.3.1 after_gamemove()	73
6.18.3.2 draw()	73
6.18.3.3 end_turn()	73
6.18.3.4 handle_events()	74
6.18.3.5 is_gamemove_finisher()	74
6.18.3.6 next_player()	74
6.18.3.7 onclick_attack()	74
6.18.3.8 onclick_base()	75
6.18.3.9 onclick_item()	75
6.18.3.10 onclick_movecell()	75
6.18.3.11 onclick_reset_gamemove()	75
6.18.3.12 onclick_ward()	75
6.18.3.13 show_cell_info()	75
6.18.3.14 show_stats()	76
6.18.3.15 update()	76
6.18.4 Member Data Documentation	76
6.18.4.1 create_simulation	76
6.19 UI::Grid Class Reference	76
6.19.1 Detailed Description	77
6.19.2 Constructor & Destructor Documentation	77
6.19.2.1 Grid()	77
6.19.3 Member Function Documentation	77
6.19.3.1 contains()	77
6.19.3.2 get_global_bounds()	78
6.19.3.3 set_elements()	78

6.19.3.4 set_elements_pos()	78
6.20 UI::GridElement Class Reference	79
6.20.1 Detailed Description	79
6.20.2 Constructor & Destructor Documentation	79
6.20.2.1 ~GridElement()	79
6.20.3 Member Function Documentation	79
6.20.3.1 contains()	79
6.20.3.2 draw()	80
6.20.3.3 get_size()	80
6.20.3.4 set_position()	80
6.21 Ground Class Reference	81
6.21.1 Detailed Description	81
6.21.2 Constructor & Destructor Documentation	81
6.21.2.1 Ground()	81
6.22 Resources::Holder Class Reference	82
6.22.1 Detailed Description	82
6.22.2 Member Function Documentation	82
6.22.2.1 get()	82
6.22.2.2 load()	82
6.23 Item Class Reference	83
6.23.1 Detailed Description	83
6.23.2 Constructor & Destructor Documentation	83
6.23.2.1 Item()	83
6.23.3 Member Function Documentation	83
6.23.3.1 get_gold_value()	84
6.23.3.2 get_name()	84
6.24 ItemBox Class Reference	84
6.24.1 Detailed Description	84
6.24.2 Constructor & Destructor Documentation	85
6.24.2.1 ItemBox()	85
6.24.3 Member Function Documentation	85
6.24.3.1 get_item()	85
6.25 Menu::MainState Class Reference	85
6.25.1 Constructor & Destructor Documentation	86
6.25.1.1 MainState()	86
6.25.1.2 ~MainState()	86
6.25.2 Member Function Documentation	86
6.25.2.1 draw()	86
6.25.2.2 handle_events()	87
6.26 Map Class Reference	87
6.26.1 Detailed Description	88
6.26.2 Constructor & Destructor Documentation	88

6.26.2.1 Map()	88
6.26.2.2 ~Map()	89
6.26.3 Member Function Documentation	89
6.26.3.1 check_game_end()	89
6.26.3.2 de_spawn()	89
6.26.3.3 disable_vision()	89
6.26.3.4 do_attack()	90
6.26.3.5 draw()	90
6.26.3.6 get_cell_grid_size()	90
6.26.3.7 get_clicked_cell()	90
6.26.3.8 get_selected_champ()	90
6.26.3.9 getcell()	91
6.26.3.10 getnearbycells()	91
6.26.3.11 in_bounds()	91
6.26.3.12 in_bounds_col()	91
6.26.3.13 in_bounds_row()	92
6.26.3.14 move()	92
6.26.3.15 reset_cell_selections()	92
6.26.3.16 reset_cell_vision()	92
6.26.3.17 select_accessible_cells()	93
6.26.3.18 select_attackable_entities()	93
6.26.3.19 select_wardable_cells()	93
6.26.3.20 set_selected_nearby_cells()	93
6.26.3.21 spawn()	94
6.26.3.22 update()	94
6.26.3.23 update_vision()	94
6.26.3.24 update_vision_side()	94
6.27 Menu::MenuButton Class Reference	95
6.27.1 Constructor & Destructor Documentation	95
6.27.1.1 MenuButton()	95
6.28 Menu::MenuState Class Reference	96
6.28.1 Detailed Description	96
6.28.2 Constructor & Destructor Documentation	96
6.28.2.1 MenuState()	97
6.28.2.2 ~MenuState()	97
6.28.3 Member Function Documentation	97
6.28.3.1 draw()	97
6.28.3.2 handle_events()	97
6.28.3.3 update()	98
6.28.4 Member Data Documentation	98
6.28.4.1 buttons	98
6.28.4.2 resources_holder	98

6.28.4.3 setting	98
6.29 Minion Class Reference	98
6.29.1 Detailed Description	99
6.29.2 Constructor & Destructor Documentation	99
6.29.2.1 Minion()	99
6.29.3 Member Function Documentation	99
6.29.3.1 do_move()	100
6.29.3.2 get_next_direction_pos_index()	100
6.29.3.3 gives_creep_score()	100
6.29.3.4 gives_vision()	100
6.29.3.5 should_focus()	100
6.30 MinionWave Class Reference	101
6.30.1 Detailed Description	101
6.30.2 Constructor & Destructor Documentation	101
6.30.2.1 MinionWave()	101
6.30.3 Member Function Documentation	101
6.30.3.1 do_move()	101
6.30.3.2 round_end()	102
6.30.3.3 spawn()	102
6.31 Menu::ModeSelectionMode Class Reference	102
6.31.1 Constructor & Destructor Documentation	103
6.31.1.1 ModeSelectionMode()	103
6.32 MoveCell Class Reference	103
6.32.1 Member Function Documentation	104
6.32.1.1 changes_pos()	104
6.32.1.2 do_move()	104
6.32.1.3 get_state_info()	104
6.33 UI::NamedBox Class Reference	105
6.33.1 Detailed Description	105
6.33.2 Constructor & Destructor Documentation	106
6.33.2.1 NamedBox() [1/2]	106
6.33.2.2 NamedBox() [2/2]	106
6.33.3 Member Function Documentation	106
6.33.3.1 contains()	106
6.33.3.2 draw()	107
6.33.3.3 get_global_bounds()	107
6.33.3.4 get_size()	107
6.33.3.5 set_char_size()	107
6.33.3.6 set_label()	108
6.33.3.7 set_label_color()	108
6.33.3.8 set_position()	108
6.33.4 Member Data Documentation	109

6.33.4.1 frame	109
6.33.4.2 label	109
6.34 Nexus Class Reference	109
6.34.1 Detailed Description	109
6.34.2 Constructor & Destructor Documentation	110
6.34.2.1 Nexus()	110
6.35 PlaceWard Class Reference	110
6.35.1 Detailed Description	110
6.35.2 Constructor & Destructor Documentation	110
6.35.2.1 PlaceWard()	111
6.35.3 Member Function Documentation	111
6.35.3.1 do_move()	111
6.35.3.2 get_state_info()	111
6.36 Player Class Reference	111
6.36.1 Detailed Description	113
6.36.2 Constructor & Destructor Documentation	113
6.36.2.1 Player()	113
6.36.2.2 ~Player()	113
6.36.3 Member Function Documentation	113
6.36.3.1 check_round_end()	113
6.36.3.2 clear_gamemoves()	113
6.36.3.3 despawn_from_map()	114
6.36.3.4 did_start()	115
6.36.3.5 do_moves()	115
6.36.3.6 get_gamemoves_state()	115
6.36.3.7 get_selected_champs()	115
6.36.3.8 get_side()	116
6.36.3.9 get_spawn_point()	116
6.36.3.10 is_gamemove_active()	116
6.36.3.11 is_his_champ()	116
6.36.3.12 round_end()	116
6.36.3.13 set_champ_icons()	117
6.36.3.14 set_font()	117
6.36.3.15 set_side()	117
6.36.3.16 set_simulation()	117
6.36.3.17 set_spawn_point()	118
6.36.3.18 set_starter()	118
6.36.3.19 spawn_champs()	118
6.36.3.20 spawn_minions()	119
6.36.3.21 update_champ_positions()	119
6.37 River Class Reference	119
6.37.1 Detailed Description	120

6.37.2 Constructor & Destructor Documentation	120
6.37.2.1 River()	120
6.38 Settings Class Reference	120
6.38.1 Detailed Description	120
6.38.2 Constructor & Destructor Documentation	121
6.38.2.1 Settings()	121
6.38.3 Member Function Documentation	121
6.38.3.1 get_champs_filepath()	121
6.38.3.2 get_items_filepath()	121
6.38.3.3 get_output_prefix()	122
6.38.3.4 set_champs_filepath()	122
6.38.3.5 set_gamemode()	122
6.38.3.6 set_items_filepath()	122
6.38.3.7 set_output_prefix()	123
6.39 SimulationState Class Reference	123
6.39.1 Detailed Description	124
6.39.2 Constructor & Destructor Documentation	124
6.39.2.1 SimulationState()	124
6.39.2.2 ~SimulationState()	124
6.39.3 Member Function Documentation	124
6.39.3.1 draw()	124
6.39.3.2 handle_events()	125
6.39.3.3 update()	125
6.40 SpawnArea Class Reference	125
6.40.1 Detailed Description	126
6.40.2 Constructor & Destructor Documentation	126
6.40.2.1 SpawnArea()	126
6.41 State Class Reference	126
6.41.1 Detailed Description	127
6.41.2 Constructor & Destructor Documentation	127
6.41.2.1 ~State()	127
6.41.2.2 State()	127
6.41.3 Member Function Documentation	127
6.41.3.1 draw()	127
6.41.3.2 handle_events()	128
6.41.3.3 update()	128
6.41.4 Member Data Documentation	128
6.41.4.1 state_manager	128
6.42 StateManager Class Reference	128
6.42.1 Detailed Description	129
6.42.2 Constructor & Destructor Documentation	129
6.42.2.1 StateManager()	129

6.42.3 Member Function Documentation	129
6.42.3.1 change_state()	129
6.42.3.2 draw()	130
6.42.3.3 exit()	130
6.42.3.4 get_size()	130
6.42.3.5 handle_events()	130
6.42.3.6 has_state()	132
6.42.3.7 pop_state()	132
6.42.3.8 push_state()	132
6.42.3.9 update()	132
6.42.3.10 update_state()	133
6.43 Structure Class Reference	133
6.43.1 Detailed Description	133
6.44 TeamCol Class Reference	133
6.44.1 Detailed Description	134
6.44.2 Constructor & Destructor Documentation	134
6.44.2.1 TeamCol()	134
6.44.3 Member Function Documentation	134
6.44.3.1 champs_size()	134
6.44.3.2 draw_to_window()	135
6.44.3.3 get_champs()	135
6.44.3.4 operator[]()	135
6.44.3.5 set_position()	135
6.45 TeleportBase Class Reference	136
6.45.1 Detailed Description	136
6.45.2 Constructor & Destructor Documentation	136
6.45.2.1 TeleportBase()	136
6.45.3 Member Function Documentation	136
6.45.3.1 changes_pos()	137
6.45.3.2 do_move()	137
6.45.3.3 get_state_info()	137
6.46 gtest_lite::Test Struct Reference	138
6.46.1 Detailed Description	138
6.46.2 Constructor & Destructor Documentation	138
6.46.2.1 Test()	138
6.46.2.2 ~Test()	139
6.46.3 Member Function Documentation	139
6.46.3.1 begin()	139
6.46.3.2 end()	139
6.46.3.3 expect()	139
6.46.3.4 fail()	139
6.46.3.5 tstatus()	140

6.46.4 Member Data Documentation	140
6.46.4.1 failed	140
6.46.4.2 name	140
6.46.4.3 null	140
6.46.4.4 status	140
6.46.4.5 sum	141
6.46.4.6 tmp	141
6.47 UI::TextBox Class Reference	141
6.47.1 Detailed Description	142
6.47.2 Constructor & Destructor Documentation	142
6.47.2.1 TextBox()	142
6.47.3 Member Function Documentation	142
6.47.3.1 add_char()	142
6.47.3.2 contains()	142
6.47.3.3 draw()	143
6.47.3.4 get_global_bounds()	143
6.47.3.5 get_is_selected()	144
6.47.3.6 get_size()	144
6.47.3.7 get_text()	144
6.47.3.8 remove_char()	144
6.47.3.9 set_position()	144
6.47.3.10 set_selected()	145
6.48 Tower Class Reference	145
6.48.1 Detailed Description	146
6.48.2 Constructor & Destructor Documentation	146
6.48.2.1 Tower()	146
6.48.3 Member Function Documentation	146
6.48.3.1 attack()	146
6.49 Wall Class Reference	146
6.49.1 Detailed Description	147
6.49.2 Constructor & Destructor Documentation	147
6.49.2.1 Wall()	147
6.49.3 Member Function Documentation	147
6.49.3.1 can_move_here()	147
6.49.3.2 can_ward_here()	148
6.50 Ward Class Reference	148
6.50.1 Detailed Description	148
6.50.2 Constructor & Destructor Documentation	148
6.50.2.1 Ward()	149
6.50.3 Member Function Documentation	149
6.50.3.1 do_move()	149
6.50.3.2 get_stats()	149

7 File Documentation	151
7.1 include/draft.hpp File Reference	151
7.2 include/game.hpp File Reference	151
7.3 include/gamemoves.hpp File Reference	152
7.4 include/gameobjects.hpp File Reference	152
7.4.1 Enumeration Type Documentation	153
7.4.1.1 Side	153
7.5 include/ioparser.h File Reference	154
7.6 include/map.hpp File Reference	154
7.7 include/menu.hpp File Reference	155
7.8 include/resources.hpp File Reference	155
7.9 include/simulation.hpp File Reference	156
7.10 include/statemanagement.hpp File Reference	156
7.10.1 Enumeration Type Documentation	157
7.10.1.1 GameMode	157
7.11 include/UIcomponents.hpp File Reference	157
7.12 src/draft.cpp File Reference	158
7.12.1 Function Documentation	158
7.12.1.1 onclick_back()	158
7.13 src/game.cpp File Reference	158
7.14 src/gamemoves.cpp File Reference	158
7.15 src/gameobjects.cpp File Reference	158
7.16 src/ioparser.cpp File Reference	158
7.17 src/main.cpp File Reference	159
7.17.1 Function Documentation	159
7.17.1.1 main()	159
7.18 src/map.cpp File Reference	159
7.19 src/menu.cpp File Reference	159
7.20 src/resources.cpp File Reference	160
7.21 src/simulation.cpp File Reference	160
7.22 src/statemanagement.cpp File Reference	160
7.23 src/UIcomponents.cpp File Reference	160
7.24 test/gtest_lite.h File Reference	160
7.24.1 Detailed Description	162
7.24.2 Macro Definition Documentation	162
7.24.2.1 CREATE_Has_	162
7.24.2.2 END	162
7.24.2.3 EXPECT	162
7.24.2.4 Nem célszerű közvetlenül használni, vagy módosítani	163
7.24.2.5 EXPECT_DOUBLE_EQ	163
7.24.2.6 EXPECT_EQ	163
7.24.2.7 EXPECT_FALSE	163

7.24.2.8 EXPECT_NE	163
7.24.2.9 EXPECT_NO_THROW	163
7.24.2.10 EXPECT_STREQ	163
7.24.2.11 EXPECT_STRNE	163
7.24.2.12 EXPECT_THROW	164
7.24.2.13 EXPECT_THROW_THROW	164
7.24.2.14 EXPECT_TRUE	164
7.24.2.15 EXPECTCMP	164
7.24.2.16 EXPECTEXP	164
7.24.2.17 EXPECTNE	165
7.24.2.18 EXPECTTHROW	165
7.24.2.19 FAIL	165
7.24.2.20 GTEND	165
7.24.2.21 GTINIT	165
7.24.2.22 SUCCEED	165
7.24.2.23 TEST	165
7.24.3 Function Documentation	166
7.24.3.1 hasMember()	166
7.25 test/main_test.cpp File Reference	166
7.25.1 Function Documentation	166
7.25.1.1 champexiststest()	166
7.25.1.2 main()	166
Index	167

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

gtest_lite	Gtest_lite: a keretrendszer függvényinek és objektumainak névtére	11
IOParser	12
Menu	12
Resources	13
UI	13

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Cell	23
Ground	81
Bush	17
River	119
SpawnArea	125
Wall	146
DraftTurn	48
Effect	51
Item	83
Entity	53
Camp	21
Drake	50
Champion	33
Minion	98
Structure	133
Nexus	109
Tower	145
Ward	148
IOParser::File	64
GameMove	66
AttackMove	15
MoveCell	103
PlaceWard	110
TeleportBase	136
UI::Grid	76
UI::GridElement	79
UI::Button	17
DraftButton	44
GameButton	65
Menu::MenuButton	95
UI::NamedBox	105
ChampBox	32
DraftNamedBox	45

ItemBox	84
UI::TextBox	141
Resources::Holder	82
Map	87
MinionWave	101
Player	111
Settings	120
State	126
DraftState	46
GameState	71
Menu::MenuState	96
Menu::MainState	85
Menu::ModeSelectionState	102
SimulationState	123
StateManager	128
TeamCol	133
gtest_lite::Test	138

Chapter 3

Class Index

3.1 Class List

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

AttackMove	Class that implements the attack move	15
Bush	Calculates vision differently than the ground object	17
UI::Button	Button class which implements a shape with some text on it, with an onclick method	17
Camp	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	21
Cell	Base class for a cell on the map	23
ChampBox	Champion box implementation, which holds a champ	32
Champion	Class for describing champions, they're a type of entities that the players can manipulate with gamemoves	33
DraftButton	Class that specializes Button, to a draftbutton with the correct design	44
DraftNamedBox	Class that specializes NamedBox, to a NamedBox with the correct design	45
DraftState	46
DraftTurn	Class used to store one draft turn	48
Drake	Class that describes dragons, there are different types of dragons, with different effects (currently only two)	50
Effect	51
Entity	The class that describes an entity	53
IOParser::File	File holder that closes the file	64
GameButton	Button that has a specific style used for game buttons	65
GameMove	Abstract class that is the base for all gamemoves	66

GameState	
State that is responsible for navigating through a game	71
UI::Grid	
Grid holds multiple grid elements, and places them in a given way	76
UI::GridElement	
Base class for grid elements	79
Ground	
Basic cell type, that can be moved on by the player	81
Resources::Holder	
Class which holds the resources for the application	82
Item	
Class that describes items, which are primarily used to give bonuses to champions (could be used on entities too if needed)	83
ItemBox	
Specialized namedbox class that holds an item	84
Menu::MainState	85
Map	
Class that describes the map	87
Menu::MenuButton	95
Menu::MenuState	
General menustate class, used as a base for simple menus	96
Minion	
Class for minions, which are a type of monsters that go through the lanes, attacking anything that's in front of them	98
MinionWave	
Holds a wave of minions, and commands them	101
Menu::ModeSelectionMode	102
MoveCell	103
UI::NamedBox	
Named box, which is a grid element that holds a shape and a text inside of it	105
Nexus	
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	109
PlaceWard	
Class that implements the ward placing mechanism	110
Player	
Class that holds everything a player has	111
River	
Only difference from ground is that it has another color	119
Settings	
Settings class, which holds the applications settings that could be needed at any state	120
SimulationState	
State that implements the simulation	123
SpawnArea	
Spawn area, where champions spawn	125
State	
Abstract State class which is used to handle one state	126
StateManager	
Class that handles the state management for this application	128
Structure	
Common parent class for structures, it shouldn't have a move (as in map movements) functions, it's position doesn't change	133
TeamCol	133
TeleportBase	
Class that implements the teleport to the base gamemove	136
gtest_lite::Test	138
UI::TextBox	
Textbox element, which is a rectangle where text you can input text into	141

Tower	Class for a tower, which damages other entities that come near it	145
Wall	Can't be moved on to by entities	146
Ward	The ward is a type of structure (as it cannot move), that gives vision, but expires after a given time intervall	148

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/draft.hpp	151
include/game.hpp	151
include/gamemoves.hpp	152
include/gameobjects.hpp	152
include/ioparser.h	154
include/map.hpp	154
include/menu.hpp	155
include/resources.hpp	155
include/simulation.hpp	156
include/statemanagement.hpp	156
include/UIcomponents.hpp	157
src/draft.cpp	158
src/game.cpp	158
src/gamemoves.cpp	158
src/gameobjects.cpp	158
src/ioparser.cpp	158
src/main.cpp	159
src/map.cpp	159
src/menu.cpp	159
src/resources.cpp	160
src/simulation.cpp	160
src/statemanagement.cpp	160
src/UIcomponents.cpp	160
test/gtest_lite.h	160
test/main_test.cpp	166

Chapter 5

Namespace Documentation

5.1 gtest_lite Namespace Reference

[gtest_lite](#): a keretrendszer függvényinek és objektumainak névtére

Classes

- struct [Test](#)

Functions

- bool [almostEQ](#) (double a, double b)

5.1.1 Detailed Description

[gtest_lite](#): a keretrendszer függvényinek és objektumainak névtére

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 holds 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

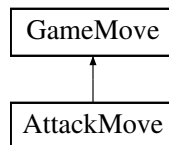
Class Documentation

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

<i>champ</i>	the champ whose move it is
<i>map</i>	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:

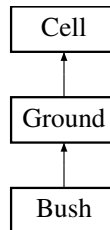
- [include/gamemoves.hpp](#)
- [src/gamemoves.cpp](#)

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:

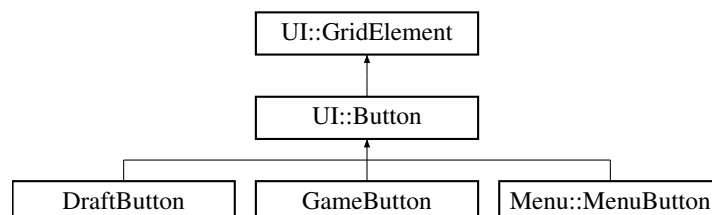
- include/[map.hpp](#)
- src/[map.cpp](#)

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

<i>text</i>	the text on the button
<i>onclick</i>	the method to be called when the button gets clicked on
<i>pos</i>	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

<i>x</i>	x coordinate
<i>y</i>	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

<i>window</i>	
---------------	--

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

<i>event</i>	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

<i>pos</i>	
------------	--

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:

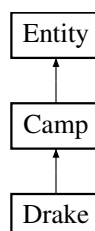
- [include/UIcomponents.hpp](#)
- [src/UIcomponents.cpp](#)

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

<i>hp</i> ↔	
—	
<i>dmg</i> ↔	
—	

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

<i>hp</i>	
<i>dmg</i>	

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

<i>e</i>	the effect to save
----------	--------------------

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

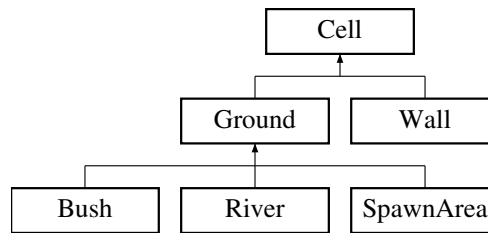
- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

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

<i>entity</i>	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

<i>enemy_↔ side_</i>	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

<i>x</i>	coordinate
<i>y</i>	coordinate

6.5.3.7 do_attack()

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

tries to do attack on each one of its entities

Parameters

<i>map</i>	
------------	--

6.5.3.8 draw()

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

draws the cell and its entities to the screen

Parameters

<i>window</i>	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

<i>x</i>	coordinate
<i>y</i>	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

<i>entity</i>	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

<i>color</i>	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

<i>pos</i>	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

<i>shop</i>	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

<i>vision</i>	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

<i>side_</i>	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

<i>map</i>	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

<i>mappos</i>	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

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

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

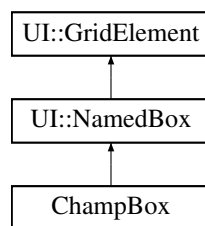
- [include/map.hpp](#)
- [src/map.cpp](#)

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:

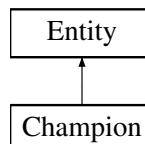
- include/draft.hpp

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

<i>name_</i>	
<i>damage_</i>	
<i>dmg_per_level_</i>	
<i>hp_</i>	
<i>hp_per_level_</i>	
—	

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

<i>move</i>	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

<i>item</i>	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

<i>map</i>	
------------	--

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

<i>map</i>	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

<i>window</i>	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

<i>other</i>	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

<i>cell</i>	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

<i>other</i>	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

<i>map</i>	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

<i>map</i>	the map to place the ward on
<i>cell</i>	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

<i>map</i>	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

<i>holder</i>	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

<i>c</i>	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

<i>sim</i>	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

<i>spawn_↔ point_</i>	
---------------------------	--

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

<i>pos</i>	the position to change to
------------	---------------------------

Reimplemented from [Entity](#).

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

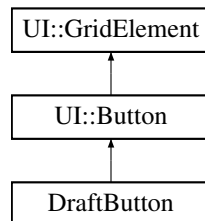
- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

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:

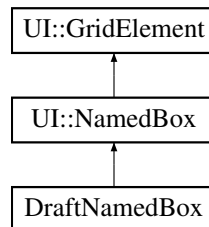
- include/[draft.hpp](#)
- src/[draft.cpp](#)

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

<i>holder</i>	the object that can get the font face for the component
<i>size</i>	the size of the namedbox

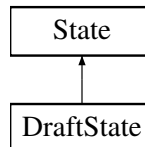
The documentation for this class was generated from the following files:

- [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

<i>window</i>	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

<i>event</i>	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

<i>state_manager</i>	the statemanager of the application
<i>window</i>	the window of the program
<i>settings</i>	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

<i>champs</i>	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:

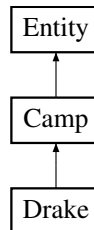
- [include/draft.hpp](#)
- [src/draft.cpp](#)

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](#).

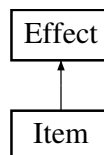
The documentation for this class was generated from the following files:

- [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

<i>dmg</i>	the new damage to use
<i>hp</i>	the new base_hp to use
<i>expires</i>	if the effect expires after some time
<i>cooldown</i>	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:

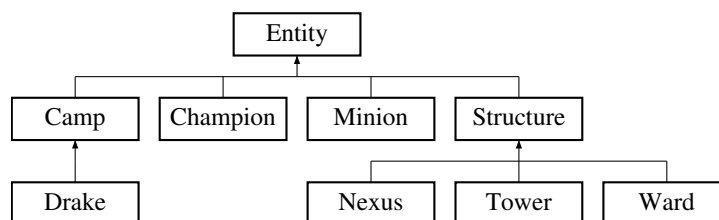
- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

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 entity'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

<i>name</i>	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

<i>map</i>	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

<i>window</i>	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

<i>entity</i>	
---------------	--

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

<i>dmg</i>	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

<i>color</i>	the color to use
--------------	------------------

6.14.3.25 set_name()

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

set's the entity's name

Parameters

<i>name</i>	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

<i>num</i>	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

<i>path</i>	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:

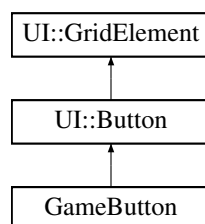
- [include/ioparser.h](#)
- [src/ioparser.cpp](#)

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

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

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

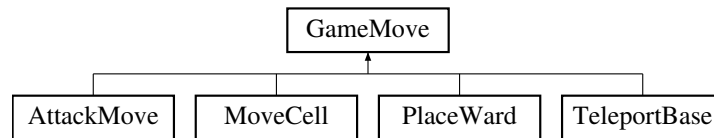
- include/[game.hpp](#)
- src/[game.cpp](#)

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

<i>current_player</i>	the currently selected player
<i>selected_champ</i>	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

<i>champ</i>	the champ whose move it is
<i>map</i>	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

<i>name</i>	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

<i>cell</i> ↔	the new cell
—	

6.17.3.12 set_movepoints()

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

sets the points

Parameters

<i>points</i> ↔	the new points value
—	

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

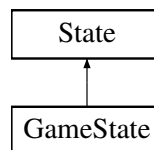
- [include/gamemoves.hpp](#)
- [src/gamemoves.cpp](#)

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

<i>state_manager</i>	the state manager of the application
<i>p1champs</i>	the champions of the first player
<i>p2champs</i>	the champions of the second player
<i>settings</i>	the application settings
<i>window</i>	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

<i>window</i>	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

<i>event</i>	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

<i>clicked_cell</i>	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

<i>index</i>	
--------------	--

6.18.3.14 show_stats()

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

shows statistics on the window

Parameters

<i>stats</i>	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

<i>start_pos</i>	the start position of the grid relative to the window
<i>margin_↔</i> —	the margin between grid elements
<i>direction_↔</i> —	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

<i>x</i>	x coordinate
<i>y</i>	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

<i>elements</i> ↔	
—	

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:

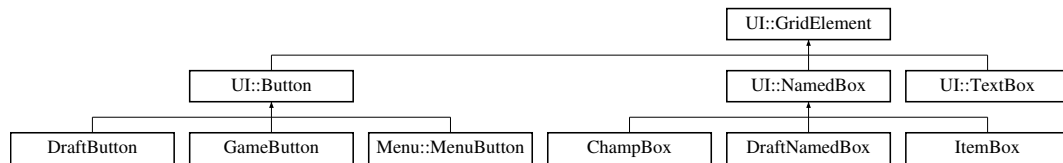
- [include/Uicomponents.hpp](#)
- [src/Uicomponents.cpp](#)

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

<i>x</i>	x coordinate
<i>y</i>	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

<i>window</i>	
---------------	--

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

<i>pos</i>	
------------	--

Implemented in [UI::NamedBox](#), [UI::TextBox](#), and [UI::Button](#).

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

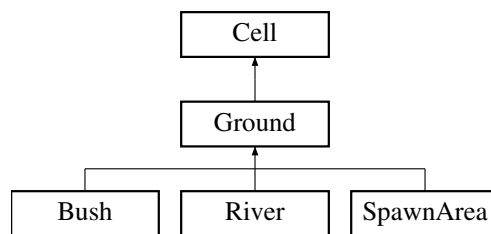
- [include/UIcomponents.hpp](#)

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

<i>type</i>	the type of the resource
<i>filename</i>	the path to the resource

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

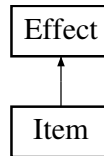
- include/[resources.hpp](#)
- src/[resources.cpp](#)

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:

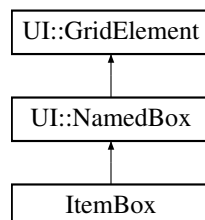
- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

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

<i>label</i>	the label on the box
<i>frame</i>	the frame to put the box inside
<i>holder</i>	the resource holder
<i>item</i>	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

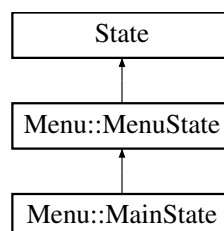
The documentation for this class was generated from the following files:

- [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

<i>window</i>	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

<i>event</i>	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

<i>pos</i>	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

<i>entity</i>	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

<i>window</i>	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

<i>x</i>	coordinate
<i>y</i>	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

<i>pos</i>	this is the middle cell
<i>distance</i>	this is how far away a cell should be to count 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

<i>p</i>	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

<i>p</i>	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

<i>entity</i>	the entity to move
<i>from</i>	the index where to move it from
<i>to</i>	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

<i>champ</i>	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

<i>champ</i>	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

<i>champ</i>	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

<i>champ,the</i>	champion which is in the middle
<i>pred</i>	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

<i>entity</i>	the entity to spawn
<i>pos</i>	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

<i>side</i> ↔	the side which has vision of the map
—	

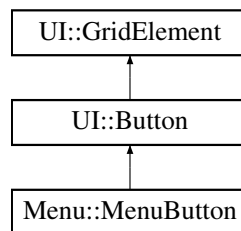
The documentation for this class was generated from the following files:

- [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:

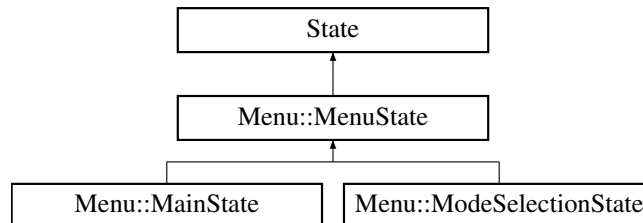
- [include/menu.hpp](#)
- [src/menu.cpp](#)

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

<i>window</i>	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

<i>event</i>	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:

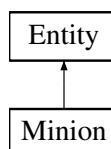
- [include/menu.hpp](#)
- [src/menu.cpp](#)

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

<i>side_</i>	the team the minion is on
<i>directions_</i>	the vector of map positions, where the minion should go (top/mid/bot)
<i>spawn_point</i>	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

<i>map</i>	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

<i>map</i>	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

<i>startpoint</i>	the point where the minions first appear
<i>directions_</i>	the directions where the minions should try to move in
<i>map</i>	the map where they move
<i>side_</i>	the side on which the minions are

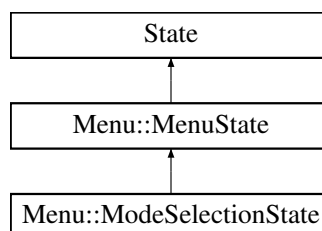
The documentation for this class was generated from the following files:

- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

6.31 Menu::ModeSelectionMode Class Reference

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

Inheritance diagram for Menu::ModeSelectionMode:



Public Member Functions

- [ModeSelectionMode](#) ([StateManager](#) &[state_manager](#), [sf::RenderWindow](#) &[window](#), [Settings](#) [setting](#))

Additional Inherited Members

6.31.1 Constructor & Destructor Documentation

6.31.1.1 ModeSelectionMode()

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

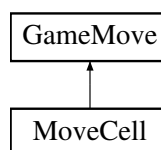
The documentation for this class was generated from the following files:

- 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

<i>champ</i>	the champ whose move it is
<i>map</i>	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:

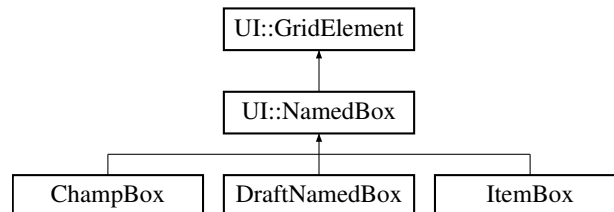
- [include/gamemoves.hpp](#)
- [src/gamemoves.cpp](#)

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

<i>holder</i>	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

<i>label</i>	the text inside the shape
<i>frame</i>	the shape
<i>holder</i>	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

<i>x</i>	x coordinate
<i>y</i>	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

<i>window</i>	
---------------	--

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

<i>size</i>	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

<i>label_text</i>	
-------------------	--

6.33.3.7 set_label_color()

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

set's the label color

Parameters

<i>c</i>	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

<i>pos</i>	
------------	--

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:

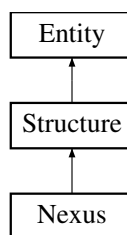
- [include/Uicomponents.hpp](#)
- [src/Uicomponents.cpp](#)

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:

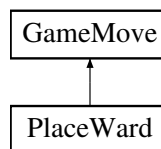
- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

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

<i>champ</i>	the champ whose move it is
<i>map</i>	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

<i>champs</i>	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

<i>map</i>	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

<i>map</i>	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

<i>c</i>	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

<i>map</i>	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

<i>icons</i>	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

<i>holder</i>	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

<i>side</i>	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

<i>sim</i>	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

<i>point</i>	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

<i>starter</i>	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

<i>map</i>	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

<i>map</i>	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

<i>map</i>	the map is needed to determine where a given cell is
------------	--

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

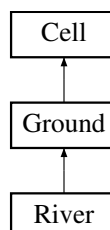
- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

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

<i>champs_filepath</i>	the filepath to the champions textfile
<i>items_filepath</i>	the filepath to the items textfile
<i>output_prefix</i>	the prefix of the output textfile
<i>mode</i>	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

<i>path</i>	the new path to use
-------------	---------------------

6.38.3.5 set_gamemode()

```
void Settings::set_gamemode (
    GameMode mode_ ) [inline]
```

sets the new gamemode

Parameters

<i>mode</i>	the new gamemode
-------------	------------------

6.38.3.6 set_items_filepath()

```
void Settings::set_items_filepath (
    std::string path ) [inline]
```

sets the champions filepath

Parameters

<i>path</i>	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

<i>path</i>	the new prefix to use
-------------	-----------------------

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

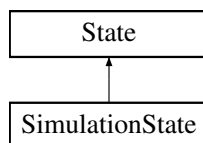
- include/statemanagement.hpp
- src/statemanagement.cpp

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

<i>players</i>	the players of the game
<i>map</i>	the map where they play
<i>window</i>	the window
<i>settings</i>	the current settings of the game
<i>state_manager</i>	the state_manager
<i>callback_</i>	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

<i>window</i>	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

<i>event</i>	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:

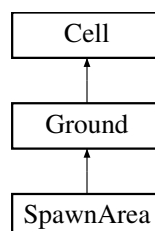
- include/[simulation.hpp](#)
- src/[simulation.cpp](#)

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:

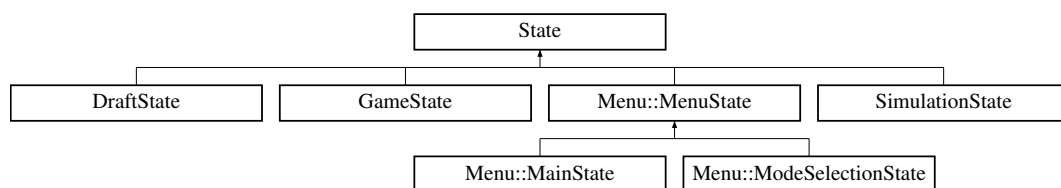
- [include/map.hpp](#)
- [src/map.cpp](#)

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

<i>window</i>	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

<i>event</i>	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

<i>state</i>	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

<i>window</i>	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

<i>window</i>	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

<i>window</i>	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

<i>state</i>	
--------------	--

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:

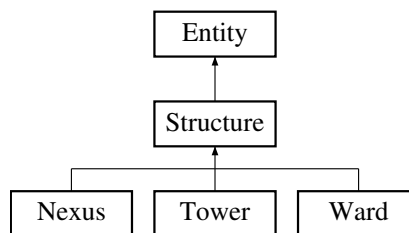
- include/[statemanagement.hpp](#)
- src/[statemanagement.cpp](#)

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

<i>holder</i>	the object that can get the font face for the component
<i>size</i>	the size of the teamcol
<i>margin</i>	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

<i>index</i>	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:

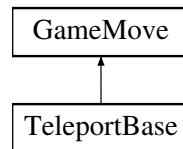
- include/draft.hpp
- src/draft.cpp

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

<i>champ</i>	the champ whose move it is
<i>map</i>	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:

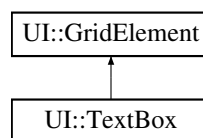
- test/[gtest_lite.h](#)

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

<i>label</i>	the name of the textbox, this is placed outside the rectangle, showing what the textbox is for
<i>holder</i>	the resources holder
<i>pos</i>	the position relative to the window
<i>text_default</i>	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

<i>c</i>	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

<i>x</i>	x coordinate
<i>y</i>	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

<i>window</i>	
---------------	--

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

<i>pos</i>	
------------	--

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

<i>s</i>	true if the textbox got selected, false otherwise
----------	---

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

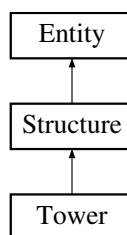
- [include/UIcomponents.hpp](#)
- [src/UIcomponents.cpp](#)

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

<i>map</i>	the map where it searches for entities
------------	--

Reimplemented from [Entity](#).

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

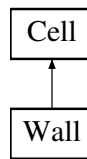
- [include/gameobjects.hpp](#)
- [src/gameobjects.cpp](#)

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:

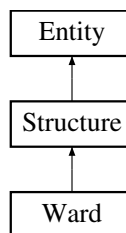
- [include/map.hpp](#)
- [src/map.cpp](#)

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 interval.

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

Inheritance diagram for Ward:



Public Member Functions

- [Ward \(\)](#)
default constructor 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 interval.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 Ward()

```
Ward::Ward ( )
```

default constructor initializes the ward's cooldown and its 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 interval.*
- 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 holds 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 névtére

Macros

- `#define TEST(C, N) { gtest_lite::test.begin(#C"."#N);`
- `#define END gtest_lite::test.end(); }`
Tesztet 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 *) eltérést 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.l. 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(); }
```

Tesztet 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 ) EXPECTTNE((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 kivételt.", "kivételt 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 *) eltéréset 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 tesztek könnyen átemelhetők a gtest keretrendszerbe.

Parameters

<i>C</i>	- teszteset neve (csak a gtest kompatibilitás miatt van külön neve az eseteknek)
<i>N</i>	- 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

- ~Cell
 - Cell, [25](#)
- ~Champion
 - Champion, [36](#)
- ~DraftState
 - DraftState, [46](#)
- ~Entity
 - Entity, [55](#)
- ~File
 - IOParser::File, [65](#)
- ~GameMove
 - GameMove, [68](#)
- ~GameState
 - GameState, [73](#)
- ~GridElement
 - UI::GridElement, [79](#)
- ~MainState
 - Menu::MainState, [86](#)
- ~Map
 - Map, [89](#)
- ~MenuState
 - Menu::MenuState, [97](#)
- ~Player
 - Player, [113](#)
- ~SimulationState
 - SimulationState, [124](#)
- ~State
 - State, [127](#)
- ~Test
 - gtest_lite::Test, [138](#)
- add_char
 - UI::TextBox, [142](#)
- add_entity
 - Cell, [25](#)
- add_gamemove
 - Champion, [36](#)
- add_item
 - Champion, [36](#)
- add_xp
 - Champion, [36](#)
- after_gamemove
 - GameState, [73](#)
- alive
 - Entity, [62](#)
- almostEQ
 - gtest_lite, [11](#)
- attack
 - Entity, [56](#)
 - Tower, [146](#)
- AttackMove, [15](#)
 - AttackMove, [15](#)
 - do_move, [16](#)
 - finish, [16](#)
 - get_state_info, [16](#)
- base_hp
 - Entity, [62](#)
- begin
 - gtest_lite::Test, [139](#)
- BLUE
 - gameobjects.hpp, [154](#)
- Bush, [17](#)
 - Bush, [17](#)
- Button
 - UI::Button, [18](#)
- buttons
 - Menu::MenuState, [98](#)
- Camp, [21](#)
 - Camp, [22](#)
 - get_buff_given, [22](#)
 - respawn, [23](#)
 - set_effect, [23](#)
- can_attack_entity
 - Cell, [26](#)
- can_buy_items
 - Cell, [26](#)
- can_fight_back
 - Champion, [36](#)
 - Entity, [56](#)
- can_move_here
 - Cell, [26](#)
 - Wall, [147](#)
- can_ward_here
 - Cell, [26](#)
 - Wall, [147](#)
- Cell, [23](#)
 - ~Cell, [25](#)
 - add_entity, [25](#)
 - can_attack_entity, [26](#)
 - can_buy_items, [26](#)
 - can_move_here, [26](#)
 - can_ward_here, [26](#)
 - Cell, [25](#)
 - contains, [26](#)
 - do_attack, [27](#)
 - draw, [27](#)
 - get_attackable_entity, [27](#)
 - get_entity_clicked, [27](#)

- get_first_entity, 28
- get_index, 28
- is_selected, 28
- remove_entity, 28
- reset_selection_color, 28
- reset_vision_color, 29
- set_color, 29
- set_highlighted, 29
- set_position, 29
- set_selected, 30
- set_shop, 30
- set_vision, 30
- should_update_vision_around, 30
- unselect, 31
- update, 31
- update_entities_shape, 31
- update_shape, 31
- cell
 - Entity, 63
- ChampBox, 32
 - ChampBox, 32
 - get_champ, 33
- champexiststest
 - main_test.cpp, 166
- Champion, 33
 - ~Champion, 36
 - add_gamemove, 36
 - add_item, 36
 - add_xp, 36
 - can_fight_back, 36
 - Champion, 35
 - clear_gamemoves, 37
 - current_gamemove_index, 37
 - despawn_wards, 37
 - do_move, 37
 - draw, 38
 - fight, 38
 - finish_gamemove, 38
 - get_current_gamemove_state_info, 38
 - get_max_hp, 39
 - get_name, 39
 - get_simulation_cell, 39
 - get_stats, 39
 - get_total_dmg, 39
 - getmovepoints, 40
 - gives_vision, 40
 - is_gamemove_complete, 40
 - killed_other, 40
 - last_gamemove_index, 41
 - move, 41
 - place_ward, 41
 - remove_last_gamemove, 41
 - round_end, 42
 - set_font, 42
 - set_icon, 42
 - set_side, 42
 - set_simulation, 43
 - set_spawn_point, 43
 - update_shape_pos, 43
- champs_size
 - TeamCol, 134
- change_state
 - StateManager, 129
- changes_pos
 - GameMove, 68
 - MoveCell, 104
 - TeleportBase, 136
- check_death
 - Entity, 56
- check_game_end
 - Map, 89
- check_gamemove_addable
 - GameMove, 68
- check_round_end
 - Player, 113
- clear_gamemoves
 - Champion, 37
 - Player, 113
- clicked
 - Entity, 56
- color
 - Entity, 63
- contains
 - Cell, 26
 - UI::Button, 19
 - UI::Grid, 77
 - UI::GridElement, 79
 - UI::NamedBox, 106
 - UI::TextBox, 142
- create_champ
 - IOParser, 12
- CREATE_Has_
 - gtest_lite.h, 162
- create_item
 - IOParser, 12
- create_simulation
 - GameState, 76
- current_gamemove_index
 - Champion, 37
- current_hp
 - Entity, 63
- damage
 - Entity, 63
- de_spawn
 - Map, 89
- decide_which_type
 - Drake, 50
- despawn_from_map
 - Player, 113
- despawn_wards
 - Champion, 37
- did_start
 - Player, 115
- disable_vision
 - Map, 89
- do_attack

- Cell, [27](#)
- Map, [89](#)
- do_move
 - AttackMove, [16](#)
 - Champion, [37](#)
 - GameMove, [68](#)
 - Minion, [99](#)
 - MinionWave, [101](#)
 - MoveCell, [104](#)
 - PlaceWard, [111](#)
 - TeleportBase, [137](#)
 - Ward, [149](#)
- do_moves
 - Player, [115](#)
- do_turn
 - DraftTurn, [49](#)
- dont_ban
 - DraftState, [47](#)
- draft.cpp
 - onclick_back, [158](#)
- DraftButton, [44](#)
 - DraftButton, [44](#)
- DraftNamedBox, [45](#)
 - DraftNamedBox, [45](#)
- DraftState, [46](#)
 - ~DraftState, [46](#)
 - dont_ban, [47](#)
 - DraftState, [46](#)
 - draw, [47](#)
 - handle_events, [47](#)
 - lock_in, [47](#)
 - update, [48](#)
- DraftTurn, [48](#)
 - do_turn, [49](#)
 - DraftTurn, [49](#)
 - is_ban_phase, [49](#)
- Drake, [50](#)
 - decide_which_type, [50](#)
 - Drake, [50](#)
 - respawn, [51](#)
- draw
 - Cell, [27](#)
 - Champion, [38](#)
 - DraftState, [47](#)
 - Entity, [57](#)
 - GameState, [73](#)
 - Map, [90](#)
 - Menu::MainState, [86](#)
 - Menu::MenuState, [97](#)
 - SimulationState, [124](#)
 - State, [127](#)
 - StateManager, [130](#)
 - UI::Button, [19](#)
 - UI::GridElement, [80](#)
 - UI::NamedBox, [107](#)
 - UI::TextBox, [143](#)
- draw_to_window
 - TeamCol, [135](#)
- Effect, [51](#)
 - Effect, [52](#)
 - get_bonus_dmg, [52](#)
 - get_bonus_hp, [52](#)
 - not_zero, [52](#)
 - set_bonus_dmg, [52](#)
 - set_bonus_hp, [53](#)
 - update_expire, [53](#)
- END
 - gtest_lite.h, [162](#)
- end
 - gtest_lite::Test, [139](#)
- end_turn
 - GameState, [73](#)
- Entity, [53](#)
 - ~Entity, [55](#)
 - alive, [62](#)
 - attack, [56](#)
 - base_hp, [62](#)
 - can_fight_back, [56](#)
 - cell, [63](#)
 - check_death, [56](#)
 - clicked, [56](#)
 - color, [63](#)
 - current_hp, [63](#)
 - damage, [63](#)
 - draw, [57](#)
 - Entity, [55](#)
 - get_buff_given, [57](#)
 - get_current_hp, [57](#)
 - get_gold_given, [57](#)
 - get_max_hp, [58](#)
 - get_real_cell, [58](#)
 - get_side, [58](#)
 - get_simulation_cell, [58](#)
 - get_stats, [58](#)
 - get_total_dmg, [58](#)
 - get_xp_given, [59](#)
 - gives_creep_score, [59](#)
 - gives_vision, [59](#)
 - gold_given, [63](#)
 - is_alive, [59](#)
 - killed_other, [59](#)
 - name, [63](#)
 - refill_hp, [60](#)
 - remove_hp, [60](#)
 - respawn, [60](#)
 - respawn_counter, [63](#)
 - respawn_timer, [63](#)
 - set_cell, [60](#)
 - set_color, [60](#)
 - set_name, [61](#)
 - set_side, [61](#)
 - set_xp_given, [61](#)
 - shape, [64](#)
 - should_focus, [61](#)
 - side, [64](#)
 - to_ui_int_format, [62](#)

- update_shape_pos, 62
 - xp_given, 64
- exit
 - StateManager, 130
- EXPECT
 - gtest_lite.h, 162
- expect
 - gtest_lite::Test, 139
- EXPECT_DOUBLE_EQ
 - gtest_lite.h, 163
- EXPECT_EQ
 - gtest_lite.h, 163
- EXPECT_FALSE
 - gtest_lite.h, 163
- EXPECT_NE
 - gtest_lite.h, 163
- EXPECT_NO_THROW
 - gtest_lite.h, 163
- EXPECT_STREQ
 - gtest_lite.h, 163
- EXPECT_STRNE
 - gtest_lite.h, 163
- EXPECT_THROW
 - gtest_lite.h, 164
- EXPECT_THROW_THROW
 - gtest_lite.h, 164
- EXPECT_TRUE
 - gtest_lite.h, 164
- EXPECTCMP
 - gtest_lite.h, 164
- EXPECTEXP
 - gtest_lite.h, 164
- EXPECTNE
 - gtest_lite.h, 164
- EXPECTTHROW
 - gtest_lite.h, 165
- FAIL
 - gtest_lite.h, 165
- fail
 - gtest_lite::Test, 139
- failed
 - gtest_lite::Test, 140
- fight
 - Champion, 38
- File
 - IOParser::File, 64
- finish
 - AttackMove, 16
 - GameMove, 69
- finish_gamemove
 - Champion, 38
- FONT
 - Resources, 13
- frame
 - UI::NamedBox, 109
- GameButton, 65
 - GameButton, 66
- GameMode
 - statemanagement.hpp, 157
- GameMove, 66
 - ~GameMove, 68
 - changes_pos, 68
 - check_gamemove_addable, 68
 - do_move, 68
 - finish, 69
 - GameMove, 67
 - get_cell, 69
 - get_formatted_info, 69
 - get_movepoints, 70
 - get_state_info, 70
 - is_complete, 70
 - position_cell, 70
 - set_cell, 70
 - set_movepoints, 71
- gameobjects.hpp
 - BLUE, 154
 - NEUTRAL, 154
 - RED, 154
 - Side, 153
- GameState, 71
 - ~GameState, 73
 - after_gamemove, 73
 - create_simulation, 76
 - draw, 73
 - end_turn, 73
 - GameState, 72
 - handle_events, 73
 - is_gamemove_finisher, 74
 - next_player, 74
 - onclick_attack, 74
 - onclick_base, 74
 - onclick_item, 75
 - onclick_movecell, 75
 - onclick_reset_gamemove, 75
 - onclick_ward, 75
 - show_cell_info, 75
 - show_stats, 76
 - update, 76
- get
 - Resources::Holder, 82
- get_attackable_entity
 - Cell, 27
- get_bonus_dmg
 - Effect, 52
- get_bonus_hp
 - Effect, 52
- get_buff_given
 - Camp, 22
 - Entity, 57
- get_cell
 - GameMove, 69
- get_cell_grid_size
 - Map, 90
- get_champ
 - ChampBox, 33

- get_champs
 - TeamCol, 135
- get_champs_filepath
 - Settings, 121
- get_clicked_cell
 - Map, 90
- get_current_gamemove_state_info
 - Champion, 38
- get_current_hp
 - Entity, 57
- get_entity_clicked
 - Cell, 27
- get_first_entity
 - Cell, 28
- get_formatted_info
 - GameMove, 69
- get_gamemoves_state
 - Player, 115
- get_global_bounds
 - UI::Button, 19
 - UI::Grid, 78
 - UI::NamedBox, 107
 - UI::TextBox, 143
- get_gold_given
 - Entity, 57
- get_gold_value
 - Item, 83
- get_index
 - Cell, 28
- get_is_selected
 - UI::TextBox, 143
- get_item
 - ItemBox, 85
- get_items_filepath
 - Settings, 121
- get_max_hp
 - Champion, 39
 - Entity, 58
- get_movepoints
 - GameMove, 70
- get_name
 - Champion, 39
 - Item, 84
- get_next_direction_pos_index
 - Minion, 100
- get_output_prefix
 - Settings, 121
- get_real_cell
 - Entity, 58
- get_selected_champ
 - Map, 90
- get_selected_champs
 - Player, 115
- get_side
 - Entity, 58
 - Player, 115
- get_simulation_cell
 - Champion, 39
 - Entity, 58
- get_size
 - StateManager, 130
 - UI::Button, 20
 - UI::GridElement, 80
 - UI::NamedBox, 107
 - UI::TextBox, 144
- get_spawn_point
 - Player, 116
- get_state_info
 - AttackMove, 16
 - GameMove, 70
 - MoveCell, 104
 - PlaceWard, 111
 - TeleportBase, 137
- get_stats
 - Champion, 39
 - Entity, 58
 - Ward, 149
- get_text
 - UI::TextBox, 144
- get_total_dmg
 - Champion, 39
 - Entity, 58
- get_xp_given
 - Entity, 59
- getcell
 - Map, 91
- getfile
 - IOParser::File, 65
- getmovepoints
 - Champion, 40
- getnearbycells
 - Map, 91
- gives_creep_score
 - Entity, 59
 - Minion, 100
- gives_vision
 - Champion, 40
 - Entity, 59
 - Minion, 100
- gold_given
 - Entity, 63
- Grid
 - UI::Grid, 77
- Ground, 81
 - Ground, 81
- GTEND
 - gtest_lite.h, 165
- gtest_lite, 11
 - almostEQ, 11
- gtest_lite.h
 - CREATE_Has_, 162
 - END, 162
 - EXPECT, 162
 - EXPECT_DOUBLE_EQ, 163
 - EXPECT_EQ, 163
 - EXPECT_FALSE, 163

- EXPECT_NE, 163
- EXPECT_NO_THROW, 163
- EXPECT_STREQ, 163
- EXPECT_STRNE, 163
- EXPECT_THROW, 164
- EXPECT_THROW_THROW, 164
- EXPECT_TRUE, 164
- EXPECTCMP, 164
- EXPECTEXP, 164
- EXPECTNE, 164
- EXPECTTHROW, 165
- FAIL, 165
- GTEND, 165
- GTINIT, 165
- hasMember, 166
- SUCCEED, 165
- TEST, 165
- gtest_lite::Test, 138
 - ~Test, 138
 - begin, 139
 - end, 139
 - expect, 139
 - fail, 139
 - failed, 140
 - name, 140
 - null, 140
 - status, 140
 - sum, 140
 - Test, 138
 - tmp, 141
 - tstatus, 139
- GTINIT
 - gtest_lite.h, 165
- handle_events
 - DraftState, 47
 - GameState, 73
 - Menu::MainState, 86
 - Menu::MenuState, 97
 - SimulationState, 125
 - State, 128
 - StateManager, 130
- has_state
 - StateManager, 132
- hasMember
 - gtest_lite.h, 166
- in_bounds
 - Map, 91
- in_bounds_col
 - Map, 91
- in_bounds_row
 - Map, 92
- include/draft.hpp, 151
- include/game.hpp, 151
- include/gamemoves.hpp, 152
- include/gameobjects.hpp, 152
- include/ioparser.h, 154
- include/map.hpp, 154
- include/menu.hpp, 155
- include/resources.hpp, 155
- include/simulation.hpp, 156
- include/statemanagement.hpp, 156
- include/UIcomponents.hpp, 157
- IOParser, 12
 - create_champ, 12
 - create_item, 12
 - split_string, 12
- IOParser::File, 64
 - ~File, 65
 - File, 64
 - getfile, 65
- is_alive
 - Entity, 59
- is_ban_phase
 - DraftTurn, 49
- is_complete
 - GameMove, 70
- is_gamemove_active
 - Player, 116
- is_gamemove_complete
 - Champion, 40
- is_gamemove_finisher
 - GameState, 74
- is_his_champ
 - Player, 116
- is_selected
 - Cell, 28
- Item, 83
 - get_gold_value, 83
 - get_name, 84
 - Item, 83
- ItemBox, 84
 - get_item, 85
 - ItemBox, 85
- killed_other
 - Champion, 40
 - Entity, 59
- label
 - UI::NamedBox, 109
- last_gamemove_index
 - Champion, 41
- load
 - Resources::Holder, 82
- lock_in
 - DraftState, 47
- main
 - main.cpp, 159
 - main_test.cpp, 166
- main.cpp
 - main, 159
- main_test.cpp
 - champexiststest, 166
 - main, 166
- MainState

- Menu::MainState, 86
- Map, 87
 - ~Map, 89
 - check_game_end, 89
 - de_spawn, 89
 - disable_vision, 89
 - do_attack, 89
 - draw, 90
 - get_cell_grid_size, 90
 - get_clicked_cell, 90
 - get_selected_champ, 90
 - getcell, 91
 - getnearbycells, 91
 - in_bounds, 91
 - in_bounds_col, 91
 - in_bounds_row, 92
 - Map, 88
 - move, 92
 - reset_cell_selections, 92
 - reset_cell_vision, 92
 - select_accessible_cells, 92
 - select_attackable_entities, 93
 - select_wardable_cells, 93
 - set_selected_nearby_cells, 93
 - spawn, 94
 - update, 94
 - update_vision, 94
 - update_vision_side, 94
- Menu, 12
- Menu::MainState, 85
 - ~MainState, 86
 - draw, 86
 - handle_events, 86
 - MainState, 86
- Menu::MenuButton, 95
 - MenuButton, 95
- Menu::MenuState, 96
 - ~MenuState, 97
 - buttons, 98
 - draw, 97
 - handle_events, 97
 - MenuState, 96
 - resources_holder, 98
 - setting, 98
 - update, 97
- Menu::ModeSelectionState, 102
 - ModeSelectionState, 103
- MenuButton
 - Menu::MenuButton, 95
- MenuState
 - Menu::MenuState, 96
- Minion, 98
 - do_move, 99
 - get_next_direction_pos_index, 100
 - gives_creep_score, 100
 - gives_vision, 100
 - Minion, 99
 - should_focus, 100
- MinionWave, 101
 - do_move, 101
 - MinionWave, 101
 - round_end, 102
 - spawn, 102
- ModeSelectionState
 - Menu::ModeSelectionState, 103
- move
 - Champion, 41
 - Map, 92
- MoveCell, 103
 - changes_pos, 104
 - do_move, 104
 - get_state_info, 104
- name
 - Entity, 63
 - gtest_lite::Test, 140
- NamedBox
 - UI::NamedBox, 106
- NEUTRAL
 - gameobjects.hpp, 154
- next_player
 - GameState, 74
- Nexus, 109
 - Nexus, 110
- not_zero
 - Effect, 52
- null
 - gtest_lite::Test, 140
- onclick
 - UI::Button, 21
- onclick_attack
 - GameState, 74
- onclick_back
 - draft.cpp, 158
- onclick_base
 - GameState, 74
- onclick_here
 - UI::Button, 20
- onclick_item
 - GameState, 75
- onclick_movecell
 - GameState, 75
- onclick_reset_gamemove
 - GameState, 75
- onclick_ward
 - GameState, 75
- operator[]
 - TeamCol, 135
- place_ward
 - Champion, 41
- PlaceWard, 110
 - do_move, 111
 - get_state_info, 111
 - PlaceWard, 110
- Player, 111

- ~Player, 113
- check_round_end, 113
- clear_gamemoves, 113
- despawn_from_map, 113
- did_start, 115
- do_moves, 115
- get_gamemoves_state, 115
- get_selected_champs, 115
- get_side, 115
- get_spawn_point, 116
- is_gamemove_active, 116
- is_his_champ, 116
- Player, 113
- round_end, 116
- set_champ_icons, 117
- set_font, 117
- set_side, 117
- set_simulation, 117
- set_spawn_point, 118
- set_starter, 118
- spawn_champs, 118
- spawn_minions, 118
- update_champ_positions, 119
- pop_state
 - StateManager, 132
- position_cell
 - GameMove, 70
- push_state
 - StateManager, 132
- RED
 - gameobjects.hpp, 154
- refill_hp
 - Entity, 60
- remove_char
 - UI::TextBox, 144
- remove_entity
 - Cell, 28
- remove_hp
 - Entity, 60
- remove_last_gamemove
 - Champion, 41
- reset_cell_selections
 - Map, 92
- reset_cell_vision
 - Map, 92
- reset_selection_color
 - Cell, 28
- reset_vision_color
 - Cell, 29
- Resources, 13
 - FONT, 13
 - Type, 13
- Resources::Holder, 82
 - get, 82
 - load, 82
- resources_holder
 - Menu::MenuState, 98
- respawn
 - Camp, 23
 - Drake, 51
 - Entity, 60
 - respawn_counter
 - Entity, 63
 - respawn_timer
 - Entity, 63
 - River, 119
 - River, 120
 - round_end
 - Champion, 42
 - MinionWave, 102
 - Player, 116
 - select_accessible_cells
 - Map, 92
 - select_attackable_entities
 - Map, 93
 - select_wardable_cells
 - Map, 93
 - set_bonus_dmg
 - Effect, 52
 - set_bonus_hp
 - Effect, 53
 - set_cell
 - Entity, 60
 - GameMove, 70
 - set_champ_icons
 - Player, 117
 - set_champs_filepath
 - Settings, 122
 - set_char_size
 - UI::NamedBox, 107
 - set_color
 - Cell, 29
 - Entity, 60
 - set_effect
 - Camp, 23
 - set_elements
 - UI::Grid, 78
 - set_elements_pos
 - UI::Grid, 78
 - set_font
 - Champion, 42
 - Player, 117
 - set_gamemode
 - Settings, 122
 - set_highlighted
 - Cell, 29
 - set_icon
 - Champion, 42
 - set_items_filepath
 - Settings, 122
 - set_label
 - UI::NamedBox, 108
 - set_label_color
 - UI::NamedBox, 108
 - set_movepoints
 - GameMove, 71

- set_name
 - Entity, 61
- set_output_prefix
 - Settings, 123
- set_position
 - Cell, 29
 - TeamCol, 135
 - UI::Button, 20
 - UI::GridElement, 80
 - UI::NamedBox, 108
 - UI::TextBox, 144
- set_selected
 - Cell, 30
 - UI::TextBox, 145
- set_selected_nearby_cells
 - Map, 93
- set_shop
 - Cell, 30
- set_side
 - Champion, 42
 - Entity, 61
 - Player, 117
- set_simulation
 - Champion, 43
 - Player, 117
- set_spawn_point
 - Champion, 43
 - Player, 118
- set_starter
 - Player, 118
- set_vision
 - Cell, 30
- set_xp_given
 - Entity, 61
- setting
 - Menu::MenuState, 98
- Settings, 120
 - get_champs_filepath, 121
 - get_items_filepath, 121
 - get_output_prefix, 121
 - set_champs_filepath, 122
 - set_gamemode, 122
 - set_items_filepath, 122
 - set_output_prefix, 123
 - Settings, 121
- shape
 - Entity, 64
 - UI::Button, 21
- should_focus
 - Entity, 61
 - Minion, 100
- should_update_vision_around
 - Cell, 30
- show_cell_info
 - GameState, 75
- show_stats
 - GameState, 76
- Side
 - gameobjects.hpp, 153
- side
 - Entity, 64
- SimulationState, 123
 - ~SimulationState, 124
 - draw, 124
 - handle_events, 125
 - SimulationState, 124
 - update, 125
- spawn
 - Map, 94
 - MinionWave, 102
- spawn_champs
 - Player, 118
- spawn_minions
 - Player, 118
- SpawnArea, 125
 - SpawnArea, 126
- split_string
 - IOParser, 12
- src/draft.cpp, 158
- src/game.cpp, 158
- src/gamemoves.cpp, 158
- src/gameobjects.cpp, 158
- src/ioparser.cpp, 158
- src/main.cpp, 159
- src/map.cpp, 159
- src/menu.cpp, 159
- src/resources.cpp, 160
- src/simulation.cpp, 160
- src/statemanagement.cpp, 160
- src/UIcomponents.cpp, 160
- State, 126
 - ~State, 127
 - draw, 127
 - handle_events, 128
 - State, 127
 - state_manager, 128
 - update, 128
- state_manager
 - State, 128
- statemanagement.hpp
 - GameMode, 157
 - TWO_PLAYER, 157
- StateManager, 128
 - change_state, 129
 - draw, 130
 - exit, 130
 - get_size, 130
 - handle_events, 130
 - has_state, 132
 - pop_state, 132
 - push_state, 132
 - StateManager, 129
 - update, 132
 - update_state, 132
- status
 - gtest_lite::Test, 140

- Structure, [133](#)
- SUCCEED
 - gtest_lite.h, [165](#)
- sum
 - gtest_lite::Test, [140](#)
- TeamCol, [133](#)
 - champs_size, [134](#)
 - draw_to_window, [135](#)
 - get_champs, [135](#)
 - operator[], [135](#)
 - set_position, [135](#)
 - TeamCol, [134](#)
- TeleportBase, [136](#)
 - changes_pos, [136](#)
 - do_move, [137](#)
 - get_state_info, [137](#)
 - TeleportBase, [136](#)
- TEST
 - gtest_lite.h, [165](#)
- Test
 - gtest_lite::Test, [138](#)
- test/gtest_lite.h, [160](#)
- test/main_test.cpp, [166](#)
- text
 - UI::Button, [21](#)
- TextBox
 - UI::TextBox, [142](#)
- tmp
 - gtest_lite::Test, [141](#)
- to_ui_int_format
 - Entity, [62](#)
- Tower, [145](#)
 - attack, [146](#)
 - Tower, [146](#)
- tstatus
 - gtest_lite::Test, [139](#)
- TWO_PLAYER
 - statemanagement.hpp, [157](#)
- Type
 - Resources, [13](#)
- UI, [13](#)
- UI::Button, [17](#)
 - Button, [18](#)
 - contains, [19](#)
 - draw, [19](#)
 - get_global_bounds, [19](#)
 - get_size, [20](#)
 - onclick, [21](#)
 - onclick_here, [20](#)
 - set_position, [20](#)
 - shape, [21](#)
 - text, [21](#)
 - update_text_position, [21](#)
- UI::Grid, [76](#)
 - contains, [77](#)
 - get_global_bounds, [78](#)
 - Grid, [77](#)
 - set_elements, [78](#)
 - set_elements_pos, [78](#)
- UI::GridElement, [79](#)
 - ~GridElement, [79](#)
 - contains, [79](#)
 - draw, [80](#)
 - get_size, [80](#)
 - set_position, [80](#)
- UI::NamedBox, [105](#)
 - contains, [106](#)
 - draw, [107](#)
 - frame, [109](#)
 - get_global_bounds, [107](#)
 - get_size, [107](#)
 - label, [109](#)
 - NamedBox, [106](#)
 - set_char_size, [107](#)
 - set_label, [108](#)
 - set_label_color, [108](#)
 - set_position, [108](#)
- UI::TextBox, [141](#)
 - add_char, [142](#)
 - contains, [142](#)
 - draw, [143](#)
 - get_global_bounds, [143](#)
 - get_is_selected, [143](#)
 - get_size, [144](#)
 - get_text, [144](#)
 - remove_char, [144](#)
 - set_position, [144](#)
 - set_selected, [145](#)
 - TextBox, [142](#)
- unselect
 - Cell, [31](#)
- update
 - Cell, [31](#)
 - DraftState, [48](#)
 - GameState, [76](#)
 - Map, [94](#)
 - Menu::MenuState, [97](#)
 - SimulationState, [125](#)
 - State, [128](#)
 - StateManager, [132](#)
- update_champ_positions
 - Player, [119](#)
- update_entities_shape
 - Cell, [31](#)
- update_expire
 - Effect, [53](#)
- update_shape
 - Cell, [31](#)
- update_shape_pos
 - Champion, [43](#)
 - Entity, [62](#)
- update_state
 - StateManager, [132](#)
- update_text_position
 - UI::Button, [21](#)

update_vision
 Map, [94](#)
update_vision_side
 Map, [94](#)

Wall, [146](#)
 can_move_here, [147](#)
 can_ward_here, [147](#)
 Wall, [147](#)
Ward, [148](#)
 do_move, [149](#)
 get_stats, [149](#)
 Ward, [148](#)

xp_given
 Entity, [64](#)