

Tower Defense 2

Generated by Doxygen 1.8.17

1 Tower Defense 2	1
1.1 Group members	1
1.2 Overview	1
1.2.1 Features of the project	1
1.2.1.1 Minimum Requirements	1
1.2.1.2 Additional Features Implemented	2
1.3 Building instructions	2
1.4 Project Structure	2
1.5 Using the Software	2
1.6 Work log	3
2 Contents	5
3 LIBS directory	7
3.1 List of External Libs	7
4 Meeting Notes	9
4.1 Week 1 Meeting 1 07.11.2022 18:00	9
4.1.1 Summary of the meeting	9
4.1.2 Project status	10
4.1.3 Next meeting	10
4.1.4 TODOs for the next meeting	10
4.2 Week 1 Meeting 2 09.11.2022 17:00	10
4.2.1 Summary of works	10
4.2.2 Challenges	10
4.2.3 Actions	11
4.2.4 Project status	11
4.3 Week 2 Meeting 18.11.2022 15:00	11
4.3.1 Summary of works	11
4.3.2 Challenges	11
4.3.3 Actions	12
4.3.4 Project status	12
4.4 Week 3 Meeting 1 25.11.2022 15:00	12
4.4.1 Summary of works	12
4.4.2 Actions	12
4.4.3 Project status	12
4.5 Week 4 Meeting 1 30.11.2022 15:00	13
4.5.1 Summary of works	13
4.5.2 Actions	13
4.5.3 Project status	13
4.6 Week 5 Meeting 1 07.12.2022 11:00	13
4.6.1 Summary of works	14
4.6.2 Actions	14

4.6.3 Project status	14
5 Contents	15
6 Source content	17
6.1 Files:	17
7 Test files	19
7.1 Unit Tests	19
7.1.1 Test of EnemyFactory	19
7.1.2 Test of The general logic and interactions between towers and enemies	19
8 Namespace Index	21
8.1 Namespace List	21
9 Hierarchical Index	23
9.1 Class Hierarchy	23
10 Class Index	25
10.1 Class List	25
11 File Index	27
11.1 File List	27
12 Namespace Documentation	29
12.1 UtilFunctions Namespace Reference	29
12.1.1 Detailed Description	29
12.1.2 Function Documentation	29
12.1.2.1 distance()	29
13 Class Documentation	31
13.1 Assignment Class Reference	31
13.1.1 Detailed Description	33
13.1.2 Constructor & Destructor Documentation	33
13.1.2.1 Assignment()	33
13.1.2.2 ~Assignment()	33
13.1.3 Member Function Documentation	33
13.1.3.1 Advance()	33
13.1.3.2 CrLeft()	34
13.1.3.3 GetCredits()	34
13.1.3.4 IsAlive()	34
13.1.3.5 MovedLastTick()	34
13.1.3.6 TakeDmg()	34
13.1.4 Friends And Related Function Documentation	35
13.1.4.1 operator<<	35
13.1.5 Member Data Documentation	35

13.1.5.1 m_curCr	35
13.1.5.2 m_maxCr	35
13.1.5.3 m_movedLastTick	35
13.1.5.4 m_timeRemainder	36
13.1.5.5 m_timeToMove	36
13.2 AttackingTower Class Reference	36
13.2.1 Detailed Description	38
13.2.2 Constructor & Destructor Documentation	38
13.2.2.1 AttackingTower()	39
13.2.2.2 ~AttackingTower()	39
13.2.3 Member Function Documentation	39
13.2.3.1 ApplyBuff()	39
13.2.3.2 Attack()	40
13.2.3.3 Bachelor()	40
13.2.3.4 Doctor()	40
13.2.3.5 Freshman()	41
13.2.3.6 GetUpgradeCost()	41
13.2.3.7 Heal()	41
13.2.3.8 IsFunctional()	42
13.2.3.9 IsUpgradeable()	42
13.2.3.10 Master()	42
13.2.3.11 Priv_UpdateRange()	43
13.2.3.12 Teekkari()	43
13.2.3.13 Upgrade()	43
13.2.4 Friends And Related Function Documentation	44
13.2.4.1 operator<<	44
13.2.5 Member Data Documentation	44
13.2.5.1 m_basePower	44
13.2.5.2 m_buffs	44
13.2.5.3 m_health	44
13.2.5.4 m_inRangeInd	44
13.2.5.5 m_level	44
13.2.5.6 m_map	45
13.2.5.7 m_maxHealth	45
13.2.5.8 m_upgCost	45
13.3 BuffTower Class Reference	45
13.3.1 Detailed Description	46
13.3.2 Constructor & Destructor Documentation	46
13.3.2.1 BuffTower()	47
13.3.2.2 ~BuffTower()	47
13.3.3 Member Function Documentation	47
13.3.3.1 Act()	47

13.3.4 Member Data Documentation	47
13.3.4.1 m_buffStrength	48
13.4 Button Class Reference	48
13.4.1 Detailed Description	49
13.4.2 Constructor & Destructor Documentation	49
13.4.2.1 Button()	49
13.4.2.2 ~Button()	49
13.4.3 Member Function Documentation	49
13.4.3.1 addHighlight()	50
13.4.3.2 changeText()	50
13.4.3.3 disableButton()	50
13.4.3.4 drawButton()	50
13.4.3.5 enableButton()	50
13.4.3.6 getGlobalBounds()	51
13.4.3.7 removeHighlight()	51
13.4.4 Member Data Documentation	51
13.4.4.1 m_button	51
13.4.4.2 m_font	51
13.4.4.3 m_text	51
13.5 Degree Class Reference	52
13.5.1 Detailed Description	53
13.5.2 Constructor & Destructor Documentation	53
13.5.2.1 Degree()	53
13.5.2.2 ~Degree()	53
13.5.3 Member Function Documentation	54
13.5.3.1 TakeDmg()	54
13.5.4 Member Data Documentation	54
13.5.4.1 m_descendants	54
13.5.4.2 m_ef	54
13.6 EditorState Class Reference	55
13.6.1 Detailed Description	56
13.6.2 Constructor & Destructor Documentation	56
13.6.2.1 EditorState()	56
13.6.2.2 ~EditorState()	57
13.6.3 Member Function Documentation	57
13.6.3.1 Priv_Draw()	57
13.6.3.2 Priv_PollEvents()	57
13.6.3.3 Run()	57
13.6.4 Member Data Documentation	57
13.6.4.1 m_buttons	57
13.6.4.2 m_drawSelectedShape	58
13.6.4.3 m_editor	58

13.6.4.4 m_instructions	58
13.6.4.5 m_mapPath	58
13.6.4.6 m_mapTileSprites	58
13.6.4.7 m_selectedButton	58
13.6.4.8 m_selectedShape	58
13.6.4.9 m_selX	58
13.6.4.10 m_selY	59
13.6.4.11 m_unvalidated	59
13.6.4.12 m_validated	59
13.7 EndState Class Reference	59
13.7.1 Detailed Description	60
13.7.2 Constructor & Destructor Documentation	60
13.7.2.1 EndState()	60
13.7.2.2 ~EndState()	61
13.7.3 Member Function Documentation	61
13.7.3.1 Priv_Draw()	61
13.7.3.2 Priv_PollEvents()	61
13.7.3.3 Run()	61
13.7.4 Member Data Documentation	62
13.7.4.1 m_buttons	62
13.7.4.2 m_difficulty	62
13.7.4.3 m_font	62
13.7.4.4 m_highscores	62
13.7.4.5 m_input	62
13.7.4.6 m_score	62
13.7.4.7 m_text_highscores	62
13.7.4.8 m_text_name	63
13.7.4.9 m_text_score	63
13.8 EnemyFactory Class Reference	63
13.8.1 Detailed Description	64
13.8.2 Constructor & Destructor Documentation	64
13.8.2.1 EnemyFactory() [1/2]	64
13.8.2.2 EnemyFactory() [2/2]	65
13.8.2.3 ~EnemyFactory()	65
13.8.3 Member Function Documentation	65
13.8.3.1 CreateEnemy()	65
13.8.3.2 EnemiesLeft()	65
13.8.3.3 GetDifficulty()	66
13.8.3.4 NextRoundInit()	66
13.8.3.5 NextTick()	66
13.8.3.6 operator=()	66
13.8.3.7 Priv_Free()	67

13.8.3.8 Priv_NextNum()	67
13.8.4 Friends And Related Function Documentation	67
13.8.4.1 operator<<	67
13.8.5 Member Data Documentation	67
13.8.5.1 m_batchSizeDeltas	67
13.8.5.2 m_batchSizes	67
13.8.5.3 m_diff	68
13.8.5.4 m_nums	68
13.8.5.5 m_round	68
13.8.5.6 m_roundEnemies	68
13.9 Game Class Reference	68
13.9.1 Detailed Description	70
13.9.2 Constructor & Destructor Documentation	70
13.9.2.1 Game() [1/2]	70
13.9.2.2 ~Game()	70
13.9.2.3 Game() [2/2]	71
13.9.3 Member Function Documentation	71
13.9.3.1 AddTower()	71
13.9.3.2 CreateTower()	71
13.9.3.3 DestroyTower()	72
13.9.3.4 EnemyTurn()	72
13.9.3.5 GetAttackingTowers()	72
13.9.3.6 GetAttacks()	72
13.9.3.7 GetDifficulty()	73
13.9.3.8 GetEnemies()	73
13.9.3.9 GetHealth()	73
13.9.3.10 GetMap()	73
13.9.3.11 GetMoney()	74
13.9.3.12 GetScore()	74
13.9.3.13 GetSupportTowers()	74
13.9.3.14 GetTower()	74
13.9.3.15 IsActionPossible()	75
13.9.3.16 operator=()	75
13.9.3.17 RoundIsFinished()	75
13.9.3.18 StartNextRound()	76
13.9.3.19 TowerTurn()	76
13.9.3.20 UpgradeTower()	76
13.9.4 Friends And Related Function Documentation	76
13.9.4.1 operator<<	76
13.9.5 Member Data Documentation	77
13.9.5.1 m_attackingTowers	77
13.9.5.2 m_enemies	77

13.9.5.3 m_enemyFactory	77
13.9.5.4 m_map	77
13.9.5.5 m_money	77
13.9.5.6 m_playerHealth	77
13.9.5.7 m_score	77
13.9.5.8 m_supportingTowers	78
13.9.5.9 m_tickAttacks	78
13.10 GameState Class Reference	78
13.10.1 Detailed Description	80
13.10.2 Constructor & Destructor Documentation	80
13.10.2.1 GameState()	80
13.10.2.2 ~GameState()	80
13.10.3 Member Function Documentation	80
13.10.3.1 Priv_ChangeCircle()	80
13.10.3.2 Priv_ClearSpeedHighlights()	81
13.10.3.3 Priv_Draw()	81
13.10.3.4 Priv_DrawBCG()	81
13.10.3.5 Priv_DrawMap()	81
13.10.3.6 Priv_InitializeText()	81
13.10.3.7 Priv_PollEvents()	82
13.10.3.8 Run()	82
13.10.4 Member Data Documentation	82
13.10.4.1 m_buildPhase	82
13.10.4.2 m_buttons	82
13.10.4.3 m_drawRange	82
13.10.4.4 m_drawUpgradeRange	83
13.10.4.5 m_frameInTick	83
13.10.4.6 m_gameLogic	83
13.10.4.7 m_gameOver	83
13.10.4.8 m_gameSpeed	83
13.10.4.9 m_healthText	83
13.10.4.10 m_mapTileSprites	83
13.10.4.11 m_moneyText	83
13.10.4.12 m_projectile	84
13.10.4.13 m_rangeCircle	84
13.10.4.14 m_roundNum	84
13.10.4.15 m_roundNumText	84
13.10.4.16 m_scoreText	84
13.10.4.17 m_selectedShape	84
13.10.4.18 m_selX	84
13.10.4.19 m_selY	84
13.10.4.20 m_upgradeRange	85

13.11 GUI Class Reference	85
13.11.1 Detailed Description	86
13.11.2 Constructor & Destructor Documentation	86
13.11.2.1 GUI()	86
13.11.2.2 ~GUI()	87
13.11.3 Member Function Documentation	87
13.11.3.1 ChangeState()	87
13.11.3.2 CreateButton()	87
13.11.3.3 CreateTowerButton()	87
13.11.3.4 GetFont()	88
13.11.3.5 init()	88
13.11.3.6 Priv_DeleteState()	88
13.11.3.7 running()	89
13.11.3.8 update()	89
13.11.4 Member Data Documentation	89
13.11.4.1 enemies	89
13.11.4.2 m_event	89
13.11.4.3 m_font	89
13.11.4.4 m_new_state	89
13.11.4.5 m_renderables	90
13.11.4.6 m_state	90
13.11.4.7 m_videoMode	90
13.11.4.8 m_window	90
13.11.4.9 start	90
13.11.4.10 x_velo	90
13.11.4.11 y_velo	90
13.12 HealTower Class Reference	91
13.12.1 Detailed Description	92
13.12.2 Constructor & Destructor Documentation	92
13.12.2.1 HealTower()	92
13.12.2.2 ~HealTower()	93
13.12.3 Member Function Documentation	93
13.12.3.1 Act()	93
13.12.4 Member Data Documentation	93
13.12.4.1 m_healStrength	93
13.13 Highscores Class Reference	93
13.13.1 Detailed Description	94
13.13.2 Constructor & Destructor Documentation	94
13.13.2.1 Highscores()	94
13.13.2.2 ~Highscores()	95
13.13.3 Member Function Documentation	95
13.13.3.1 AddScore()	95

13.13.3.2 GetTop10()	95
13.13.3.3 GetTop10asString()	96
13.13.3.4 Priv_LoadHighscores()	96
13.13.3.5 Priv_SortHighscores()	96
13.13.4 Member Data Documentation	96
13.13.4.1 m_filename	96
13.13.4.2 m_highscores	96
13.13.4.3 m_saved	97
13.14 LevelEditor Class Reference	97
13.14.1 Detailed Description	98
13.14.2 Constructor & Destructor Documentation	98
13.14.2.1 LevelEditor()	98
13.14.2.2 ~LevelEditor()	98
13.14.3 Member Function Documentation	98
13.14.3.1 Edit()	98
13.14.3.2 GetMap()	99
13.14.3.3 Save()	99
13.14.3.4 Validate()	99
13.14.4 Member Data Documentation	99
13.14.4.1 m_height	100
13.14.4.2 m_map	100
13.14.4.3 m_mapPath	100
13.14.4.4 m_width	100
13.15 Map Class Reference	100
13.15.1 Detailed Description	101
13.15.2 Constructor & Destructor Documentation	101
13.15.2.1 Map()	101
13.15.2.2 ~Map()	102
13.15.3 Member Function Documentation	102
13.15.3.1 BuildPath()	102
13.15.3.2 Edit()	102
13.15.3.3 GetEnd()	103
13.15.3.4 GetGrid()	103
13.15.3.5 GetNeighbors()	103
13.15.3.6 GetPath()	104
13.15.3.7 GetPos()	104
13.15.3.8 GetStart()	104
13.15.3.9 InitializeMap()	104
13.15.3.10 TestTilePos()	105
13.15.3.11 ValidateMap()	105
13.15.4 Member Data Documentation	105
13.15.4.1 m_end	106

13.15.4.2 m_grid	106
13.15.4.3 m_height	106
13.15.4.4 m_path	106
13.15.4.5 m_start	106
13.15.4.6 m_width	106
13.16 MenuState Class Reference	107
13.16.1 Detailed Description	108
13.16.2 Constructor & Destructor Documentation	108
13.16.2.1 MenuState()	108
13.16.2.2 ~MenuState()	108
13.16.3 Member Function Documentation	108
13.16.3.1 Draw()	109
13.16.3.2 PollEvents()	109
13.16.3.3 Run()	109
13.16.3.4 RunLevelEditor()	109
13.16.4 Member Data Documentation	109
13.16.4.1 m_buttons	109
13.16.4.2 m_difficulty	109
13.16.4.3 m_editing	110
13.16.4.4 m_height	110
13.16.4.5 m_selectedMap	110
13.16.4.6 m_texts	110
13.16.4.7 m_width	110
13.17 Renderable Class Reference	111
13.17.1 Detailed Description	112
13.17.2 Constructor & Destructor Documentation	112
13.17.2.1 ~Renderable()	112
13.17.2.2 Renderable()	112
13.17.3 Member Function Documentation	112
13.17.3.1 EntityName()	112
13.17.3.2 GetSprite()	113
13.17.3.3 SetSprite()	113
13.17.4 Member Data Documentation	113
13.17.4.1 m_entityName	113
13.17.4.2 m_sprite	113
13.18 Renderables Class Reference	114
13.18.1 Detailed Description	116
13.18.2 Constructor & Destructor Documentation	116
13.18.2.1 Renderables()	116
13.18.3 Member Function Documentation	116
13.18.3.1 getAttackSound()	116
13.18.3.2 getBachelor1Sprite()	117

13.18.3.3	getBachelor2Sprite()	117
13.18.3.4	getBachelor3Sprite()	117
13.18.3.5	getBachelorsThesisSprite()	117
13.18.3.6	getBackgroundSprite()	117
13.18.3.7	getBscSprite()	117
13.18.3.8	getCalculatorSprite()	117
13.18.3.9	getCoffeeTableSprite()	117
13.18.3.10	getDoctor1Sprite()	118
13.18.3.11	getDoctor2Sprite()	118
13.18.3.12	getDoctor3Sprite()	118
13.18.3.13	getDoctoralThesisSprite()	118
13.18.3.14	getDscSprite()	118
13.18.3.15	getEndtileSprite()	118
13.18.3.16	getEssaySprite()	118
13.18.3.17	getFreshman1Sprite()	118
13.18.3.18	getFreshman2Sprite()	119
13.18.3.19	getFreshman3Sprite()	119
13.18.3.20	getHomeworkSprite()	119
13.18.3.21	getMaster1Sprite()	119
13.18.3.22	getMaster2Sprite()	119
13.18.3.23	getMaster3Sprite()	119
13.18.3.24	getMastersThesisSprite()	119
13.18.3.25	getMenuBackgroundSprite()	119
13.18.3.26	getMscSprite()	120
13.18.3.27	getPathtileSprite()	120
13.18.3.28	getProjectSprite()	120
13.18.3.29	getSelectSound()	120
13.18.3.30	getStarttileSprite()	120
13.18.3.31	getTeekkari1Sprite()	120
13.18.3.32	getTeekkari2Sprite()	120
13.18.3.33	getTeekkari3Sprite()	120
13.18.3.34	getTowertileSprite()	121
13.18.4	Member Data Documentation	121
13.18.4.1	attack	121
13.18.4.2	attack_sound	121
13.18.4.3	bachelor_1	121
13.18.4.4	bachelor_1_sprite	121
13.18.4.5	bachelor_2	121
13.18.4.6	bachelor_2_sprite	121
13.18.4.7	bachelor_3	122
13.18.4.8	bachelor_3_sprite	122
13.18.4.9	bachelors_thesis	122

13.18.4.10 bachelors_thesis_sprite	122
13.18.4.11 background	122
13.18.4.12 background_sprite	122
13.18.4.13 bsc	122
13.18.4.14 bsc_sprite	122
13.18.4.15 calculator	123
13.18.4.16 calculator_sprite	123
13.18.4.17 coffee_table	123
13.18.4.18 coffee_table_sprite	123
13.18.4.19 doctor_1	123
13.18.4.20 doctor_1_sprite	123
13.18.4.21 doctor_2	123
13.18.4.22 doctor_2_sprite	123
13.18.4.23 doctor_3	124
13.18.4.24 doctor_3_sprite	124
13.18.4.25 doctoral_thesis	124
13.18.4.26 doctoral_thesis_sprite	124
13.18.4.27 dsc	124
13.18.4.28 dsc_sprite	124
13.18.4.29 endTile	124
13.18.4.30 endTile_sprite	124
13.18.4.31 essay	125
13.18.4.32 essay_sprite	125
13.18.4.33 freshman_1	125
13.18.4.34 freshman_1_sprite	125
13.18.4.35 freshman_2	125
13.18.4.36 freshman_2_sprite	125
13.18.4.37 freshman_3	125
13.18.4.38 freshman_3_sprite	125
13.18.4.39 homework	126
13.18.4.40 homework_sprite	126
13.18.4.41 master_1	126
13.18.4.42 master_1_sprite	126
13.18.4.43 master_2	126
13.18.4.44 master_2_sprite	126
13.18.4.45 master_3	126
13.18.4.46 master_3_sprite	126
13.18.4.47 masters_thesis	127
13.18.4.48 masters_thesis_sprite	127
13.18.4.49 menu_background	127
13.18.4.50 menu_background_sprite	127
13.18.4.51 msc	127

13.18.4.52 msc_sprite	127
13.18.4.53 pathTile	127
13.18.4.54 pathTile_sprite	127
13.18.4.55 project	128
13.18.4.56 project_sprite	128
13.18.4.57 s_instance	128
13.18.4.58 select	128
13.18.4.59 select_sound	128
13.18.4.60 startTile	128
13.18.4.61 startTile_sprite	128
13.18.4.62 teekkari_1	129
13.18.4.63 teekkari_1_sprite	129
13.18.4.64 teekkari_2	129
13.18.4.65 teekkari_2_sprite	129
13.18.4.66 teekkari_3	129
13.18.4.67 teekkari_3_sprite	129
13.18.4.68 towerTile	129
13.18.4.69 towerTile_sprite	130
13.19 State Class Reference	130
13.19.1 Detailed Description	131
13.19.2 Constructor & Destructor Documentation	131
13.19.2.1 State()	131
13.19.2.2 ~State()	131
13.19.3 Member Function Documentation	132
13.19.3.1 Run()	132
13.19.4 Member Data Documentation	132
13.19.4.1 m_event	132
13.19.4.2 m_gui	132
13.19.4.3 m_window	132
13.20 SupportTower Class Reference	133
13.20.1 Detailed Description	134
13.20.2 Constructor & Destructor Documentation	134
13.20.2.1 SupportTower()	134
13.20.2.2 ~SupportTower()	135
13.20.3 Member Function Documentation	135
13.20.3.1 Act()	135
13.20.3.2 Calculator()	135
13.20.3.3 CoffeeTable()	136
13.20.3.4 IsUpgradeable()	136
13.20.4 Friends And Related Function Documentation	136
13.20.4.1 operator<<	136
13.21 Tower Class Reference	137

13.21.1 Detailed Description	138
13.21.2 Constructor & Destructor Documentation	138
13.21.2.1 ~Tower()	138
13.21.2.2 Tower()	139
13.21.3 Member Function Documentation	140
13.21.3.1 GetCoords()	140
13.21.3.2 GetRange()	140
13.21.3.3 IsUpgradeable()	140
13.21.4 Member Data Documentation	141
13.21.4.1 m_allSprites	141
13.21.4.2 m_coords	141
13.21.4.3 m_range	141
13.21.4.4 towerHealths	141
13.21.4.5 towerPowers	141
13.21.4.6 towerPrices	142
13.21.4.7 towerRanges	142
13.22 TowerButton Class Reference	142
13.22.1 Detailed Description	143
13.22.2 Constructor & Destructor Documentation	143
13.22.2.1 TowerButton()	143
13.22.2.2 ~TowerButton()	144
13.22.3 Member Function Documentation	144
13.22.3.1 disableButton()	144
13.22.3.2 drawButton()	144
13.22.3.3 enableButton()	144
13.22.4 Member Data Documentation	145
13.22.4.1 m_name	145
13.22.4.2 m_sprite	145
14 File Documentation	147
14.1 build/CMakeCache.txt File Reference	147
14.2 build/CMakeFiles/3.16.3/CompilerIdC/CMakeCCompilerId.c File Reference	147
14.2.1 Macro Definition Documentation	147
14.2.1.1 ARCHITECTURE_ID	148
14.2.1.2 C_DIALECT	148
14.2.1.3 COMPILER_ID	148
14.2.1.4 DEC	148
14.2.1.5 HEX	148
14.2.1.6 PLATFORM_ID	149
14.2.1.7 STRINGIFY	149
14.2.1.8 STRINGIFY_HELPER	149
14.2.2 Function Documentation	149

14.2.2.1 main()	149
14.2.3 Variable Documentation	149
14.2.3.1 info_arch	149
14.2.3.2 info_compiler	149
14.2.3.3 info_language_dialect_default	150
14.2.3.4 info_platform	150
14.3 build/CMakeFiles/3.16.3/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference	150
14.3.1 Macro Definition Documentation	150
14.3.1.1 ARCHITECTURE_ID	150
14.3.1.2 COMPILER_ID	151
14.3.1.3 CXX_STD	151
14.3.1.4 DEC	151
14.3.1.5 HEX	151
14.3.1.6 PLATFORM_ID	151
14.3.1.7 STRINGIFY	152
14.3.1.8 STRINGIFY_HELPER	152
14.3.2 Function Documentation	152
14.3.2.1 main()	152
14.3.3 Variable Documentation	152
14.3.3.1 info_arch	152
14.3.3.2 info_compiler	152
14.3.3.3 info_language_dialect_default	152
14.3.3.4 info_platform	153
14.4 build/CMakeFiles/TargetDirectories.txt File Reference	153
14.5 build/CMakeFiles/tower-defense.dir/link.txt File Reference	153
14.6 CMakeLists.txt File Reference	153
14.7 doc/readme.md File Reference	153
14.8 libs/readme.md File Reference	153
14.9 plan/readme.md File Reference	153
14.10 src/readme.md File Reference	153
14.11 tests/readme.md File Reference	153
14.12 highscores.txt File Reference	153
14.13 Meeting-notes.md File Reference	153
14.14 README.md File Reference	153
14.15 src/assignment.cpp File Reference	153
14.15.1 Function Documentation	154
14.15.1.1 operator<<()	154
14.16 src/assignment.hpp File Reference	154
14.16.1 Enumeration Type Documentation	155
14.16.1.1 Enemy	155
14.17 src/attacking_tower.cpp File Reference	155
14.17.1 Function Documentation	156

14.17.1.1 operator<<()	156
14.18 src/attacking_tower.hpp File Reference	156
14.19 src/button.cpp File Reference	157
14.19.1 Macro Definition Documentation	158
14.19.1.1 TILE_SIZE	158
14.20 src/button.hpp File Reference	158
14.21 src/degree.cpp File Reference	159
14.22 src/degree.hpp File Reference	159
14.23 src/enemy_factory.cpp File Reference	160
14.23.1 Macro Definition Documentation	161
14.23.1.1 DETAILED_DEBUG_PRINT	161
14.23.2 Function Documentation	161
14.23.2.1 operator<<()	161
14.24 src/enemy_factory.hpp File Reference	161
14.24.1 Enumeration Type Documentation	162
14.24.1.1 Difficulty	162
14.25 src/game.cpp File Reference	162
14.25.1 Function Documentation	163
14.25.1.1 operator<<()	163
14.26 src/game.hpp File Reference	163
14.26.1 Enumeration Type Documentation	164
14.26.1.1 Action	164
14.27 src/gui.cpp File Reference	165
14.28 src/gui.hpp File Reference	165
14.29 src/highscores.cpp File Reference	166
14.30 src/highscores.hpp File Reference	167
14.31 src/level_editor.cpp File Reference	168
14.31.1 Macro Definition Documentation	169
14.31.1.1 PRINT_EDITOR_ERRORS	169
14.32 src/level_editor.hpp File Reference	170
14.33 src/main.cpp File Reference	170
14.33.1 Function Documentation	171
14.33.1.1 main()	171
14.34 src/map.cpp File Reference	171
14.35 src/map.hpp File Reference	172
14.35.1 Enumeration Type Documentation	172
14.35.1.1 TileType	172
14.36 src/maps/map1.txt File Reference	173
14.37 src/maps/map2.txt File Reference	173
14.38 src/maps/map3.txt File Reference	173
14.39 src/renderable.cpp File Reference	173
14.40 src/renderable.hpp File Reference	173

14.41 src/renderables.cpp File Reference	174
14.42 src/renderables.hpp File Reference	175
14.43 src/states/editorstate.cpp File Reference	175
14.43.1 Macro Definition Documentation	176
14.43.1.1 TILE_SIZE	176
14.44 src/states/editorstate.hpp File Reference	176
14.45 src/states/endstate.cpp File Reference	177
14.46 src/states/endstate.hpp File Reference	178
14.47 src/states/gamestate.cpp File Reference	179
14.47.1 Macro Definition Documentation	179
14.47.1.1 ANIMATION_LENGTH	179
14.47.1.2 PROJECTILE_RADIUS	179
14.47.1.3 TILE_SIZE	180
14.48 src/states/gamestate.hpp File Reference	180
14.49 src/states/menustate.cpp File Reference	181
14.50 src/states/menustate.hpp File Reference	181
14.51 src/states/state.hpp File Reference	182
14.52 src/support_towers.cpp File Reference	183
14.52.1 Function Documentation	183
14.52.1.1 operator<<()	183
14.53 src/support_towers.hpp File Reference	184
14.54 src/tower.cpp File Reference	185
14.55 src/tower.hpp File Reference	185
14.55.1 Macro Definition Documentation	186
14.55.1.1 TILE_SIZE	186
14.55.2 Enumeration Type Documentation	186
14.55.2.1 TowerType	186
14.56 src/utils.cpp File Reference	187
14.57 src/utils.hpp File Reference	187
14.58 tests/enemy_factory_test.cpp File Reference	188
14.58.1 Function Documentation	189
14.58.1.1 main()	189
14.59 tests/highscores_test.cpp File Reference	189
14.59.1 Function Documentation	190
14.59.1.1 main()	190
14.60 tests/level_editor_test.cpp File Reference	190
14.60.1 Detailed Description	191
14.60.2 Function Documentation	191
14.60.2.1 main()	191
14.61 tests/map_test.cpp File Reference	192
14.61.1 Detailed Description	192
14.61.2 Function Documentation	192

14.61.2.1 main()	192
14.62 tests/testmap1.txt File Reference	193
14.63 tests/testmap2.txt File Reference	193
14.64 tests/testmapWrong.txt File Reference	193
14.65 tests/text_based_test.cpp File Reference	193
14.65.1 Detailed Description	193
14.65.2 Function Documentation	193
14.65.2.1 main()	193
Index	195

Chapter 1

Tower Defense 2

1.1 Group members

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

1.2 Overview

Goal: create a [Tower](#) Defense game, using C++

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

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

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

1.2.1 Features of the project

1.2.1.1 Minimum Requirements

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

1.2.1.2 Additional Features Implemented

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

1.3 Building instructions

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

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

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

```
sudo apt-get install libsFML-dev
```

1.4 Project Structure

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

1.5 Using the Software

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

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

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

The game launches on this screen.

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

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

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

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

Attack towers will deal damage to enemies.

Buff towers will enhance the damage of Attack towers.

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

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

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

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

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

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

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

1.6 Work log

Division of work among group members

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

Chapter 2

Contents

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

The document should contain the following parts:

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

Chapter 3

LIBS directory

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

3.1 List of External Libs

1. `Project1`
2. `Project2`

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

Chapter 4

Meeting Notes

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

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

4.1 Week 1 Meeting 1 07.11.2022 18:00

Participants:

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

4.1.1 Summary of the meeting

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

4.1.2 Project status

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

4.1.3 Next meeting

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

4.1.4 TODOs for the next meeting

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

4.2 Week 1 Meeting 2 09.11.2022 17:00

Participants:

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

4.2.1 Summary of works

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

4.2.2 Challenges

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

4.2.3 Actions

This week:

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

Before next meeting:

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

4.2.4 Project status

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

4.3 Week 2 Meeting 18.11.2022 15:00

Participants:

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

4.3.1 Summary of works

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

4.3.2 Challenges

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

4.3.3 Actions

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

4.3.4 Project status

[Map](#) and enemies are roughly complete. Goal of the next week: Get the wave part of the game running

4.4 Week 3 Meeting 1 25.11.2022 15:00

Participants:

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

4.4.1 Summary of works

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

4.4.2 Actions

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

4.4.3 Project status

[GUI](#) still a work in progress. Basic structures for core and classes done. Communication between classes is partly done.

Next focusing on piecing the puzzle together and adding the rest of the features to complete the game.

4.5 Week 4 Meeting 1 30.11.2022 15:00

Participants:

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

4.5.1 Summary of works

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

4.5.2 Actions

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

4.5.3 Project status

Backend is close to complete, [GUI](#) and the game loops have been designed and need to be implemented.

4.6 Week 5 Meeting 1 07.12.2022 11:00

Participants:

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

4.6.1 Summary of works

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

4.6.2 Actions

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

4.6.3 Project status

Let's finish this before the deadline!

Chapter 5

Contents

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

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

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

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

Chapter 6

Source content

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

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

6.1 Files:

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

Chapter 7

Test files

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

It might be a good idea to also take some notes about the tests since you are required to report these in the final report.

7.1 Unit Tests

7.1.1 Test of EnemyFactory

Involved Classes: [EnemyFactory](#), [Assignment](#), [Degree\(TODO\)](#), [Renderable](#)

Test File: [enemy_factory_test.cpp](#)

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

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

Involved Classes: [Game](#), [Map](#), all tower and enemy related classes

Test File: [text_based_test.cpp](#)

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

Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all namespaces with brief descriptions:

UtilFunctions	
A namespace containing some utility functions needed in the project	29

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

Button	48
TowerButton	142
EnemyFactory	63
Game	68
GUI	85
Highscores	93
LevelEditor	97
Map	100
Renderable	111
Assignment	31
Degree	52
Tower	137
AttackingTower	36
SupportTower	133
BuffTower	45
HealTower	91
Renderables	114
State	130
EditorState	55
EndState	59
GameState	78
MenuState	107

Chapter 10

Class Index

10.1 Class List

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

Assignment	Generic Enemy class, these just die when they are killed	31
AttackingTower	A class for the offensive towers	36
BuffTower	A tower which buffs attacking towers making the do more damage	45
Button	A class for the buttons in the game	48
Degree	A more advanced type of enemy Splits into multiple other enemies upon death The descendants are stored in the m_descendants collection as pairs where the first one is the type and the second one is the amount	52
EditorState	A gamestate run by GUI corresponding to the level editor	55
EndState	A gamestate run by GUI corresponding to the game over screen	59
EnemyFactory	A class which handles the logic about what enemies come and how much of during each round	63
Game	A class which holds the logic of the game	68
GameState	GameState class runs and draws the game part of the software	78
GUI	A class to add elements to the Graphical User Interface	85
HealTower	A tower which heals attacking towers	91
Highscores	A class used to handle the high score savings at the end of game	93
LevelEditor	The logical core of the level editor state	97
Map	Map class that represents the grid map system behind each unique level	100
MenuState	A state run by GUI corresponding to the main menu state	107

Renderable	A class which encapsulates the renderable objects (enemies and towers) Is not supposed to be directly instanciated, so the constructor is protected	111
Renderables	A class which handles the textures for different sprites One instance of this class must be constructed somewhere in the code before any static getters are accessed!	114
State	Abstract State class represents the classes that run and draw the different software states . . .	130
SupportTower	A virtual base class for the supporting towers Cannot be directly instanciated	133
Tower	An abstract base-class for the towers Sub-classes will be Attacking towers and supporting towers. Cannot be directly instanciated	137
TowerButton	A subclass of buttons, for building the towers	142

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

build/CMakeFiles/3.16.3/CompilerIdC/CMakeCCompilerId.c	147
build/CMakeFiles/3.16.3/CompilerIdCXX/CMakeCXXCompilerId.cpp	150
src/assignment.cpp	153
src/assignment.hpp	154
src/attacking_tower.cpp	155
src/attacking_tower.hpp	156
src/button.cpp	157
src/button.hpp	158
src/degree.cpp	159
src/degree.hpp	159
src/enemy_factory.cpp	160
src/enemy_factory.hpp	161
src/game.cpp	162
src/game.hpp	163
src/gui.cpp	165
src/gui.hpp	165
src/highscores.cpp	166
src/highscores.hpp	167
src/level_editor.cpp	168
src/level_editor.hpp	170
src/main.cpp	170
src/map.cpp	171
src/map.hpp	172
src/renderable.cpp	173
src/renderable.hpp	173
src/renderables.cpp	174
src/renderables.hpp	175
src/support_towers.cpp	183
src/support_towers.hpp	184
src/tower.cpp	185
src/tower.hpp	185
src/utils.cpp	187
src/utils.hpp	187
src/states/editorstate.cpp	175
src/states/editorstate.hpp	176

src/states/endstate.cpp	177
src/states/endstate.hpp	178
src/states/gamestate.cpp	179
src/states/gamestate.hpp	180
src/states/menustate.cpp	181
src/states/menustate.hpp	181
src/states/state.hpp	182
tests/enemy_factory_test.cpp	188
tests/highscores_test.cpp	189
tests/level_editor_test.cpp	
Used to test the LevelEditor class	190
tests/map_test.cpp	
Used to test the Map class	192
tests/text_based_test.cpp	
Used to test the game running with text output to terminal	193

Chapter 12

Namespace Documentation

12.1 UtilFunctions Namespace Reference

A namespace containing some utility functions needed in the project.

Functions

- float [distance](#) (const std::pair< int32_t, int32_t > &c1, const std::pair< int32_t, int32_t > &c2)
Calculates the Euclidean distance between two coordinate pairs.

12.1.1 Detailed Description

A namespace containing some utility functions needed in the project.

12.1.2 Function Documentation

12.1.2.1 distance()

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

Calculates the Euclidean distance between two coordinate pairs.

Parameters

<i>c1</i>	coordinate 1
<i>c2</i>	coordinate 2

Returns

float

Chapter 13

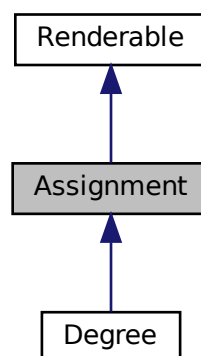
Class Documentation

13.1 Assignment Class Reference

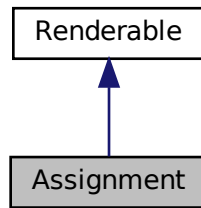
Generic Enemy class, these just die when they are killed.

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

Inheritance diagram for Assignment:



Collaboration diagram for Assignment:



Public Member Functions

- [Assignment](#) (uint32_t cr, uint32_t timeToMove, const std::string &name, const sf::Sprite &sprite)
Construct a new [Assignment](#) object.
- virtual [~Assignment](#) ()=default
A virtual destructor.
- bool [Advance](#) ()
Tells if the enemy must be moved forward Returns true if the enemy moves to the next place.
- bool [MovedLastTick](#) () const
Tells if the enemy moved during the last tick For animation purposes.
- bool [IsAlive](#) () const
Tells whether this enemy is still alive or not.
- uint32_t [CrLeft](#) () const
Tells the amount of health the enemy has left.
- uint32_t [GetCredits](#) () const
Totalt amount of credits the enemy is worth.
- virtual uint32_t [TakeDmg](#) (uint32_t dmg, std::list< [Assignment](#) * > &location)
Makes the enemy take damage Splitting enemies hurt the attacking tower when they die. The return value tells the amount of this damage. The method is overridden in subclass [Degree](#).

Protected Attributes

- uint32_t [m_maxCr](#)
- uint32_t [m_curCr](#)
- uint32_t [m_timeToMove](#)
- uint32_t [m_timeRemainder](#)
- bool [m_movedLastTick](#)

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Assignment](#) &as)
Overload for stream operator << for debugging.

Additional Inherited Members

13.1.1 Detailed Description

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

13.1.2 Constructor & Destructor Documentation

13.1.2.1 Assignment()

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

Construct a new [Assignment](#) object.

Parameters

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

13.1.2.2 ~Assignment()

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

A virtual destructor.

13.1.3 Member Function Documentation

13.1.3.1 Advance()

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

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

Returns

bool

13.1.3.2 CrLeft()

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

Tells the amount of health the enemy has left.

Returns

uint32_t

13.1.3.3 GetCredits()

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

Totalt amount of credits the enemy is worth.

Returns

uint32_t

13.1.3.4 IsAlive()

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

Tells whether this enemy is still alive or not.

Returns

bool

13.1.3.5 MovedLastTick()

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

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

Returns

bool

13.1.3.6 TakeDmg()

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

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

Parameters

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

Returns

uint32_t

Reimplemented in [Degree](#).

13.1.4 Friends And Related Function Documentation

13.1.4.1 operator<<

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

Overload for stream operator << for debugging.

13.1.5 Member Data Documentation

13.1.5.1 m_curCr

```
uint32_t Assignment::m\_curCr [protected]
```

13.1.5.2 m_maxCr

```
uint32_t Assignment::m\_maxCr [protected]
```

13.1.5.3 m_movedLastTick

```
bool Assignment::m\_movedLastTick [protected]
```

13.1.5.4 m_timeRemainder

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

13.1.5.5 m_timeToMove

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

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

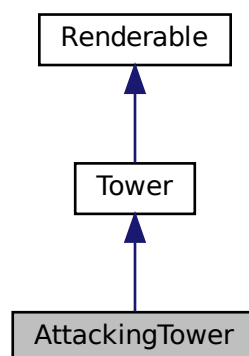
- [src/assignment.hpp](#)
- [src/assignment.cpp](#)

13.2 AttackingTower Class Reference

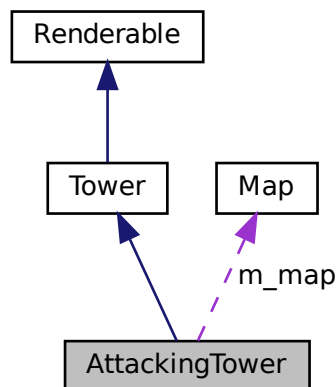
A class for the offensive towers.

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

Inheritance diagram for AttackingTower:



Collaboration diagram for AttackingTower:



Public Member Functions

- [AttackingTower](#) (uint32_t power, uint32_t range, uint32_t health, uint32_t upgCost, const std::pair< int32_t, int32_t > &coords, const [Map](#) &map, const std::string &name, const std::vector< sf::Sprite > &sprites)
Construct a new Attacking Tower object.
- [~AttackingTower](#) ()=default
Default destructor.
- void [Attack](#) (std::vector< std::list< [Assignment](#) * >> &enemies, std::list< std::pair< std::pair< int32_t, int32_t >, int32_t >, std::pair< int32_t, int32_t >>> &attackCollection)
Performs an attack against one enemy The tower goes through the possible targetable locations in m_inRangeInd, starting from the one closest to end and when it finds a living enemy in one of the locations, it attacks that. After performing an attack, it clears the buffs It also adds the attacks which it performs to a collection given as a reference.
- void [ApplyBuff](#) (float b)
Used by the buffing towers to apply a buff In case multiple buffing towers buff a single tower, the buffs stack additively, not multiplicatively.
- bool [IsFunctional](#) () const
Tells whether the tower is functional or destroyed.
- void [Heal](#) (uint32_t h)
Used by the healing towers to heal other towers Will not heal over the maximum health.
- bool [IsUpgradeable](#) (uint32_t money) const
Tells whether or not this tower can be upgraded with the current amount of money Also checks that this tower is not already max level.
- uint32_t [GetUpgradeCost](#) () const
Tells the upgrade cost.
- uint32_t [Upgrade](#) ()
Can be used to upgrade the tower to the next level. Assumes that tower is upgradeable.

Static Public Member Functions

- static [AttackingTower](#) * [Freshman](#) (const std::pair< int32_t, int32_t > &coords, const [Map](#) &map)
Static function to create a specific tower.
- static [AttackingTower](#) * [Teekkari](#) (const std::pair< int32_t, int32_t > &coords, const [Map](#) &map)
Static function to create a specific tower.
- static [AttackingTower](#) * [Bachelor](#) (const std::pair< int32_t, int32_t > &coords, const [Map](#) &map)
Static function to create a specific tower.
- static [AttackingTower](#) * [Master](#) (const std::pair< int32_t, int32_t > &coords, const [Map](#) &map)
Static function to create a specific tower.
- static [AttackingTower](#) * [Doctor](#) (const std::pair< int32_t, int32_t > &coords, const [Map](#) &map)
Static function to create a specific tower.

Private Member Functions

- void [Priv_UpdateRange](#) (uint32_t newRange)
Private. Used to update the m_inRangeInd.

Private Attributes

- uint32_t [m_basePower](#)
- uint32_t [m_maxHealth](#)
- uint32_t [m_health](#)
- uint32_t [m_upgCost](#)
- uint32_t [m_level](#)
- float [m_buffs](#)
- std::vector< uint32_t > [m_inRangeInd](#)
- const [Map](#) & [m_map](#)

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [AttackingTower](#) &at)
Overload for the stream output operator.

Additional Inherited Members

13.2.1 Detailed Description

A class for the offensive towers.

13.2.2 Constructor & Destructor Documentation

13.2.2.1 AttackingTower()

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

Construct a new Attacking Tower object.

Parameters

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

13.2.2.2 ~AttackingTower()

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

Default destructor.

13.2.3 Member Function Documentation

13.2.3.1 ApplyBuff()

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

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

Parameters

<i>b</i>	The buff amount as a decimal number (e.g. 20% buff is 0.2f)
----------	---

13.2.3.2 Attack()

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

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

Parameters

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

13.2.3.3 Bachelor()

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

Static function to create a specific tower.

Parameters

<i>coords</i>	Where the tower is placed
<i>map</i>	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.4 Doctor()

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

Static function to create a specific tower.

Parameters

<i>coords</i>	Where the tower is placed
<i>map</i>	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.5 Freshman()

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

Static function to create a specific tower.

Parameters

<i>coords</i>	Where the tower is placed
<i>map</i>	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.6 GetUpgradeCost()

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

Tells the upgrade cost.

Returns

uint32_t

13.2.3.7 Heal()

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

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

Parameters

<i>h</i>	The heal amount
----------	-----------------

13.2.3.8 IsFunctional()

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

Tells whether the tower is functional or destroyed.

Returns

bool

13.2.3.9 IsUpgradeable()

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

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

Returns

bool

Implements [Tower](#).

13.2.3.10 Master()

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

Static function to create a specific tower.

Parameters

<i>coords</i>	Where the tower is placed
<i>map</i>	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.11 Priv_UpdateRange()

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

Private. Used to update the m_inRangeInd.

Parameters

<i>newRange</i>	The new range of the tower
-----------------	----------------------------

13.2.3.12 Teekkari()

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

Static function to create a specific tower.

Parameters

<i>coords</i>	Where the tower is placed
<i>map</i>	A const ref to the map the tower is placed on

Returns

AttackingTower*

13.2.3.13 Upgrade()

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

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

Returns

uint32_t The cost of the upgrade

13.2.4 Friends And Related Function Documentation

13.2.4.1 operator<<

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

Overload for the stream output operator.

13.2.5 Member Data Documentation

13.2.5.1 m_basePower

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

13.2.5.2 m_buffs

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

13.2.5.3 m_health

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

13.2.5.4 m_inRangeInd

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

13.2.5.5 m_level

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


13.2.5.6 m_map

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

13.2.5.7 m_maxHealth

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

13.2.5.8 m_upgCost

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

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

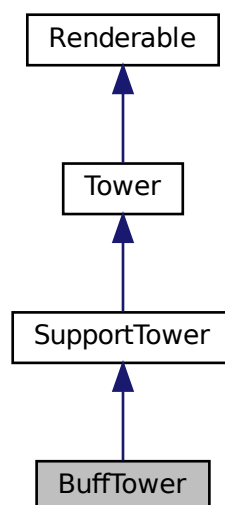
- [src/attacking_tower.hpp](#)
- [src/attacking_tower.cpp](#)

13.3 BuffTower Class Reference

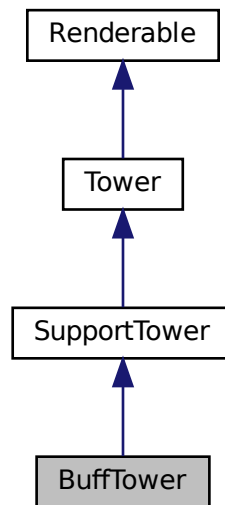
A tower which buffs attacking towers making the do more damage.

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

Inheritance diagram for BuffTower:



Collaboration diagram for BuffTower:



Public Member Functions

- [BuffTower](#) (uint32_t range, const std::pair< int32_t, int32_t > &coords, float buffStrength, const std::string &name, const std::vector< sf::Sprite > &sprites)
Construct a new Buff [Tower](#) object.
- [~BuffTower](#) ()=default
Default destructor.
- void [Act](#) (std::list< [AttackingTower](#) * > &towers)
Goes trough the attacking towers in the game and applies buff to them.

Private Attributes

- float [m_buffStrength](#)

Additional Inherited Members

13.3.1 Detailed Description

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

13.3.2 Constructor & Destructor Documentation

13.3.2.1 BuffTower()

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

Construct a new Buff [Tower](#) object.

Parameters

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

13.3.2.2 ~BuffTower()

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

Default destructor.

13.3.3 Member Function Documentation

13.3.3.1 Act()

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

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

Parameters

<i>towers</i>	The towers on the game board
---------------	------------------------------

Implements [SupportTower](#).

13.3.4 Member Data Documentation

13.3.4.1 m_buffStrength

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

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

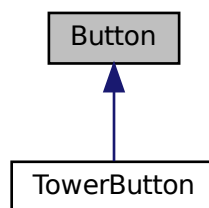
- [src/support_towers.hpp](#)
- [src/support_towers.cpp](#)

13.4 Button Class Reference

A class for the buttons in the game.

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

Inheritance diagram for Button:



Public Member Functions

- [Button](#) (std::string text, int32_t x, int32_t y, sf::Font &font)
Construct a new [Button](#) object.
- virtual [~Button](#) ()=default
A virtual destructor.
- void [addHighlight](#) ()
Adds a highlighted border for the button.
- void [removeHighlight](#) ()
Removes the highlight.
- virtual void [disableButton](#) ()
Makes the button grayed out.
- virtual void [enableButton](#) ()
Restores the original look of the button.
- virtual void [changeText](#) (std::string text)
Changes the text shown in the button.
- virtual void [drawButton](#) (sf::RenderWindow &window)
Draws the button on the window.
- sf::FloatRect [getGlobalBounds](#) ()
Get the bounds of this button.

Protected Attributes

- sf::RectangleShape [m_button](#)
- sf::Text [m_text](#)
- sf::Font & [m_font](#)

13.4.1 Detailed Description

A class for the buttons in the game.

13.4.2 Constructor & Destructor Documentation

13.4.2.1 Button()

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

Construct a new [Button](#) object.

Parameters

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

13.4.2.2 ~Button()

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

A virtual destructor.

13.4.3 Member Function Documentation

13.4.3.1 addHighlight()

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

Adds a highlighted border for the button.

13.4.3.2 changeText()

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

Changes the text shown in the button.

Parameters

<i>text</i>	The new text
-------------	--------------

13.4.3.3 disableButton()

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

Makes the button grayed out.

Reimplemented in [TowerButton](#).

13.4.3.4 drawButton()

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

Draws the button on the window.

Parameters

<i>window</i>	A ref to the window where to draw
---------------	-----------------------------------

Reimplemented in [TowerButton](#).

13.4.3.5 enableButton()

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

Restores the original look of the button.

Reimplemented in [TowerButton](#).

13.4.3.6 getGlobalBounds()

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

Get the bounds of this button.

Returns

sf::FloatRect

13.4.3.7 removeHighlight()

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

Removes the highlight.

13.4.4 Member Data Documentation

13.4.4.1 m_button

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

13.4.4.2 m_font

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

13.4.4.3 m_text

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

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

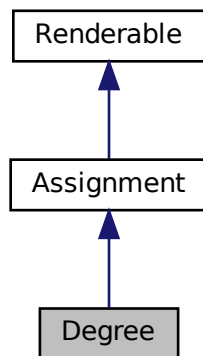
- [src/button.hpp](#)
- [src/button.cpp](#)

13.5 Degree Class Reference

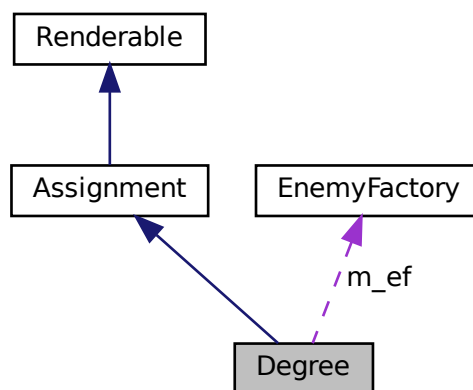
A more advanced type of enemy Splits into multiple other enemies upon death The descendants are stored in the `m_descendants` collection as pairs where the first one is the type and the second one is the amount.

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

Inheritance diagram for Degree:



Collaboration diagram for Degree:



Public Member Functions

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

Construct a new [Degree](#) object.

- [~Degree](#) ()=default
Default destructor.
- uint32_t [TakeDmg](#) (uint32_t dmg, std::list< [Assignment](#) * > &location)
Makes the enemy take damage.

Private Attributes

- const std::list< std::pair< [Enemy](#), uint32_t > > [m_descendants](#)
- const [EnemyFactory](#) & [m_ef](#)

Additional Inherited Members

13.5.1 Detailed Description

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

13.5.2 Constructor & Destructor Documentation

13.5.2.1 Degree()

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

Construct a new [Degree](#) object.

Parameters

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

13.5.2.2 ~Degree()

```
Degree::~Degree ( ) [default]
```

Default destructor.

13.5.3 Member Function Documentation

13.5.3.1 TakeDmg()

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

Makes the enemy take damage.

Parameters

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

Returns

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

Reimplemented from [Assignment](#).

13.5.4 Member Data Documentation

13.5.4.1 m_descendants

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

13.5.4.2 m_ef

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

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

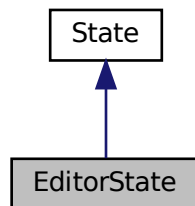
- [src/degree.hpp](#)
- [src/degree.cpp](#)

13.6 EditorState Class Reference

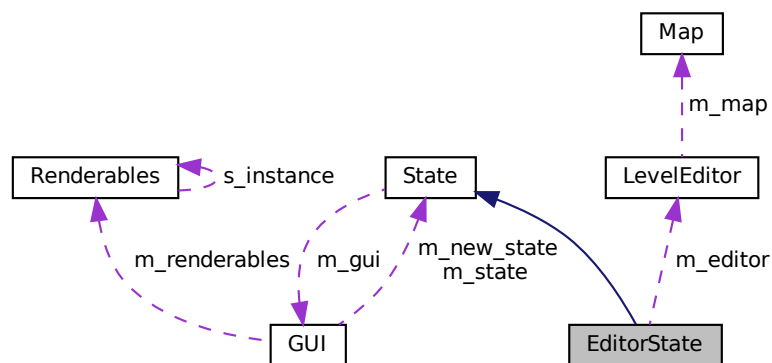
A gamestate run by [GUI](#) corresponding to the level editor.

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

Inheritance diagram for EditorState:



Collaboration diagram for EditorState:



Public Member Functions

- [EditorState](#) ([GUI](#) &gui, sf::RenderWindow &window, std::string &mapPath)
Construct a new Menu [State](#) object.
- [~EditorState](#) ()
Destroy the Menu [State](#) object Frees the buttons.
- void [Run](#) ()
Run and draw the Menu state. (Hosts the loop that Polls [GUI](#) events, calls the necessary backend methods and draws the window. Has options to select difficulty and map file. Has buttons to game editor and to play.) Exits the loop and calls GUIs `MoveToGameState(int score)` when the user presses "Play".

Private Member Functions

- void [Priv_PollEvents](#) ()
Polls the events that have happened in [GUI](#).
- void [Priv_Draw](#) ()
Draws the frame to the screen.

Private Attributes

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

Additional Inherited Members

13.6.1 Detailed Description

A gamestate run by [GUI](#) corresponding to the level editor.

13.6.2 Constructor & Destructor Documentation

13.6.2.1 EditorState()

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

Construct a new Menu [State](#) object.

Parameters

<i>gui</i>	A ref to the GUI used
<i>window</i>	A ref to the window used
<i>mapPath</i>	The map file which is edited

13.6.2.2 ~EditorState()

```
EditorState::~~EditorState ( )
```

Destroy the Menu [State](#) object Frees the buttons.

13.6.3 Member Function Documentation

13.6.3.1 Priv_Draw()

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

Draws the frame to the screen.

13.6.3.2 Priv_PollEvents()

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

Polls the events that have happened in [GUI](#).

13.6.3.3 Run()

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

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

Implements [State](#).

13.6.4 Member Data Documentation

13.6.4.1 m_buttons

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

13.6.4.2 m_drawSelectedShape

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

13.6.4.3 m_editor

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

13.6.4.4 m_instructions

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

13.6.4.5 m_mapPath

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

13.6.4.6 m_mapTileSprites

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

13.6.4.7 m_selectedButton

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

13.6.4.8 m_selectedShape

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

13.6.4.9 m_selX

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

13.6.4.10 m_selY

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

13.6.4.11 m_unvalidated

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

13.6.4.12 m_validated

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

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

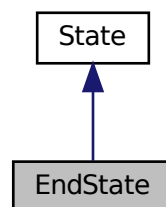
- [src/states/editorstate.hpp](#)
- [src/states/editorstate.cpp](#)

13.7 EndState Class Reference

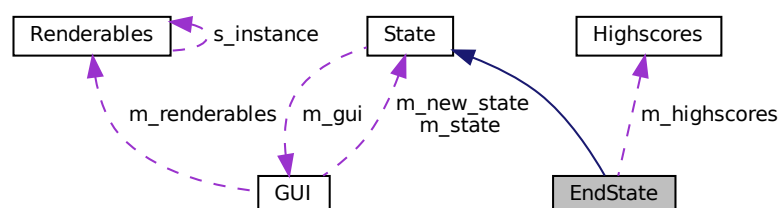
A gamestate run by [GUI](#) corresponding to the game over screen.

```
#include <endstate.hpp>
```

Inheritance diagram for EndState:



Collaboration diagram for EndState:



Public Member Functions

- [EndState](#) ([GUI](#) &gui, [sf::RenderWindow](#) &>window, [uint32_t](#) score, [Difficulty](#) difficulty)
Construct a new End [State](#) object.
- [~EndState](#) ()
Destroy the End [State](#) object Frees the buttons.
- void [Run](#) ()
Run and draw the [Game](#) Over state of the software. (Hosts the loop that Polls [GUI](#) events, calls the necessary backend methods and draws the window. Allows saving the score, displays top scores and has a button to main menu.) Exits the loop and calls GUIs MoveToMenuState(int score) when the user presses "Main menu".

Private Member Functions

- void [Priv_PollEvents](#) ()
Polls the events that have happened in [GUI](#).
- void [Priv_Draw](#) ()
Draws the frame to the screen.

Private Attributes

- [uint32_t](#) [m_score](#)
- [Difficulty](#) [m_difficulty](#)
- [std::map](#)< [int32_t](#), [Button](#) * > [m_buttons](#)
- [Highscores](#) [m_highscores](#)
- [std::string](#) [m_input](#)
- [sf::Text](#) [m_text_score](#)
- [sf::Text](#) [m_text_name](#)
- [sf::Text](#) [m_text_highscores](#)
- [sf::Font](#) [m_font](#)

Additional Inherited Members

13.7.1 Detailed Description

A gamestate run by [GUI](#) corresponding to the game over screen.

13.7.2 Constructor & Destructor Documentation

13.7.2.1 EndState()

```
EndState::EndState (
    GUI & gui,
    sf::RenderWindow & window,
    uint32\_t score,
    Difficulty difficulty )
```

Construct a new End [State](#) object.

Parameters

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

13.7.2.2 ~EndState()

```
EndState::~~EndState ( )
```

Destroy the End [State](#) object Frees the buttons.

13.7.3 Member Function Documentation

13.7.3.1 Priv_Draw()

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

Draws the frame to the screen.

13.7.3.2 Priv_PollEvents()

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

Polls the events that have happened in [GUI](#).

13.7.3.3 Run()

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

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

Implements [State](#).

13.7.4 Member Data Documentation

13.7.4.1 m_buttons

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

13.7.4.2 m_difficulty

```
Difficulty EndState::m_difficulty [private]
```

13.7.4.3 m_font

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

13.7.4.4 m_highscores

```
Highscores EndState::m_highscores [private]
```

13.7.4.5 m_input

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

13.7.4.6 m_score

```
uint32_t EndState::m_score [private]
```

13.7.4.7 m_text_highscores

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

13.7.4.8 m_text_name

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

13.7.4.9 m_text_score

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

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

- [src/states/endstate.hpp](#)
- [src/states/endstate.cpp](#)

13.8 EnemyFactory Class Reference

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

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

Public Member Functions

- [EnemyFactory](#) (Difficulty diff)
Constructs a new Enemy Factory object.
- [EnemyFactory](#) (const [EnemyFactory](#) &other)=delete
Delete copy constructor.
- [EnemyFactory](#) & operator= (const [EnemyFactory](#) &other)=delete
Delete assignment operator.
- [~EnemyFactory](#) ()
Destroys the Enemy Factory object, if there were some enemies left in the storage, frees them.
- uint32_t [NextRoundInit](#) ()
Initializes the enemies of the next round After calling this, the enemies which come each tick can be obtained using [NextTick\(\)](#) Also frees the previously allocated enemies if the previous round for some reason did not finish.
- std::list< [Assignment](#) * > [NextTick](#) ()
Used to get the enemies which appear on the next game tick Removes them from the objects own collection, so the responsibility is transferred to the object which calls the function The max amount of enemies during the tick for round n is approximately $\text{base-2-log}(a_n) + 1$ where a_n is the element of the sequence used to determine the types present.
- bool [EnemiesLeft](#) () const
Tells if there are still enemies left which have not been handed to the [Game](#) logic unit.
- [Assignment](#) * [CreateEnemy](#) ([Enemy](#) e) const
Creates an Enemy object Can be used either by this class itself or by the game core to spawn the additional enemies at some location.
- [Difficulty](#) [GetDifficulty](#) () const
Tells the set difficulty.

Private Member Functions

- `uint32_t Priv_NextNum ()`
Private. Uses `m_nums` to calculate the next one in the sequence.
- `void Priv_Free ()`
Private Used to free the enemies which are left.

Private Attributes

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

Friends

- `std::ostream & operator<< (std::ostream &os, const EnemyFactory &ef)`
An overload for the stream operator for debugging purposes.

13.8.1 Detailed Description

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

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

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

13.8.2 Constructor & Destructor Documentation

13.8.2.1 EnemyFactory() [1/2]

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

Constructs a new Enemy Factory object.

Parameters

<i>diff</i>	The difficulty of the game, scales the HP of the enemies
-------------	--

13.8.2.2 EnemyFactory() [2/2]

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

Delete copy constructor.

13.8.2.3 ~EnemyFactory()

```
EnemyFactory::~EnemyFactory ( )
```

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

13.8.3 Member Function Documentation**13.8.3.1 CreateEnemy()**

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

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

Returns

Assignment* A dynamically allocated enemy

13.8.3.2 EnemiesLeft()

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

Tells if there are still enemies left which have not been handed to the [Game](#) logic unit.

Returns

bool

13.8.3.3 GetDifficulty()

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

Tells the set difficulty.

Returns

Difficulty

13.8.3.4 NextRoundInit()

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

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

Returns

uint32_t The round number which is starting

13.8.3.5 NextTick()

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

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

Returns

std::list<Assignment*>

13.8.3.6 operator=()

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

Delete assignment operator.

13.8.3.7 Priv_Free()

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

Private Used to free the enemies which are left.

13.8.3.8 Priv_NextNum()

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

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

Returns

uint32_t The next number in the sequence

13.8.4 Friends And Related Function Documentation

13.8.4.1 operator<<

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

An overload for the stream operator for debugging purposes.

13.8.5 Member Data Documentation

13.8.5.1 m_batchSizeDeltas

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

13.8.5.2 m_batchSizes

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

13.8.5.3 m_diff

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

13.8.5.4 m_nums

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

13.8.5.5 m_round

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

13.8.5.6 m_roundEnemies

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

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

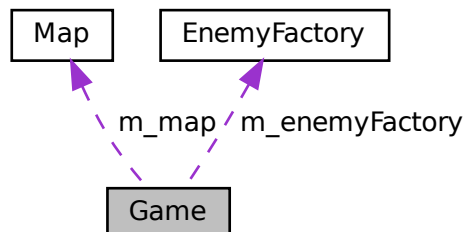
- [src/enemy_factory.hpp](#)
- [src/enemy_factory.cpp](#)

13.9 Game Class Reference

A class which holds the logic of the game.

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

Collaboration diagram for Game:



Public Member Functions

- [Game](#) (uint32_t mapWidth, uint32_t mapHeight, const std::string &filename, [Difficulty](#) difficulty)
Construct a new [Game](#) object.
- [~Game](#) ()
Destroy the [Game](#) object In case of a game over / quit the enemies and towers present must be freed.
- [Game](#) (const [Game](#) &other)=delete
Delete copy constructor.
- [Game](#) & operator= (const [Game](#) &other)=delete
Delete assignment operator.
- uint32_t [StartNextRound](#) ()
Used to start the next round Calls the enemyfactory to initialize its own state such that the round can start.
- bool [EnemyTurn](#) ()
Advances the enemies.
- bool [TowerTurn](#) ()
Makes the towers attack the enemies If enemies died during the attacking, returns true (for sound effects)
- bool [RoundIsFinished](#) ()
Used to check if the round is still ongoing. SHOULD be called only AFTER [StartNextRound](#)()!
- std::vector< std::list< [Assignment](#) * > > & [GetEnemies](#) ()
Get the Enemies in the game now Method for getting enemies and their coordinates for [GUI](#) to draw.
- const std::list< std::pair< std::pair< int32_t, int32_t >, std::pair< int32_t, int32_t > > > & [GetAttacks](#) () const
Gives information about which attacks happened during the turn.
- const [Map](#) & [GetMap](#) () const
Gets a const ref version of the map used.
- bool [AddTower](#) ([Tower](#) *t)
For adding a tower to the [Game](#). Only for the [text_based_test.cpp](#).
- bool [IsActionPossible](#) (const std::pair< int32_t, int32_t > &coords, [Action](#) a) const
Used by [GUI](#) states to check what can be done.
- void [CreateTower](#) (const std::pair< int32_t, int32_t > &coords, [TowerType](#) t)
*Create a [Tower](#) object in the game Uses the coordinates and enumeration to place a tower on the playing field
Reduces the players money.*
- void [UpgradeTower](#) (const std::pair< int32_t, int32_t > &coords)
Upgrades tower and reduces player's money.
- void [DestroyTower](#) (const std::pair< int32_t, int32_t > &coords)
Destroys tower.
- const std::list< [AttackingTower](#) * > & [GetAttackingTowers](#) () const
Get a ref to the Attacking Towers for drawing them.
- const std::list< [SupportTower](#) * > & [GetSupportTowers](#) () const
Get a ref to the Support Towers for drawing them.
- const [Tower](#) * [GetTower](#) (const std::pair< int32_t, int32_t > &coords) const
Get a pointer to [Tower](#) in a cell The function will return nullptr if no tower is at the desired location.
- uint32_t [GetScore](#) () const
Calculates player's score. Total money earned is multiplied by 100 and then divided by the length of the path. When the path is shorter there is shorter time to defeat enemies so player get more points.
- uint32_t [GetMoney](#) () const
Tells the player's current money.
- uint32_t [GetHealth](#) () const
Tells the player's current health.
- [Difficulty](#) [GetDifficulty](#) () const
Tells the difficulty of the current game.

Private Attributes

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

Friends

- std::ostream & [operator](#)<< (std::ostream &os, const [Game](#) &game)
Overload for the stream output operator.

13.9.1 Detailed Description

A class which holds the logic of the game.

13.9.2 Constructor & Destructor Documentation

13.9.2.1 [Game\(\)](#) [1/2]

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

Construct a new [Game](#) object.

Parameters

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

13.9.2.2 [~Game\(\)](#)

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

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

13.9.2.3 Game() [2/2]

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

Delete copy constructor.

13.9.3 Member Function Documentation

13.9.3.1 AddTower()

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

For adding a tower to the [Game](#). Only for the [text_based_test.cpp](#).

Parameters

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

Returns

bool Whether the adding was successfull

13.9.3.2 CreateTower()

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

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

Parameters

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

13.9.3.3 DestroyTower()

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

Destroys tower.

Parameters

<i>coords</i>	position of tower to destroy
---------------	------------------------------

13.9.3.4 EnemyTurn()

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

Advances the enemies.

Returns

true: No game over
false: [Game](#) over

13.9.3.5 GetAttackingTowers()

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

Get a ref to the Attacking Towers for drawing them.

Returns

const std::list<AttackingTower*>&

13.9.3.6 GetAttacks()

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

Gives information about which attacks happened during the turn.

Returns

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

13.9.3.7 GetDifficulty()

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

Tells the difficulty of the current game.

Returns

Difficulty

13.9.3.8 GetEnemies()

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

Get the Enemies in the game now Method for getting enemies and their coordinates for [GUI](#) to draw.

Returns

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

13.9.3.9 GetHealth()

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

Tells the player's current health.

Returns

uint32_t

13.9.3.10 GetMap()

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

Gets a const ref version of the map used.

Returns

const [Map](#)&

13.9.3.11 GetMoney()

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

Tells the player's current money.

Returns

uint32_t

13.9.3.12 GetScore()

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

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

Returns

uint32_t player's score

13.9.3.13 GetSupportTowers()

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

Get a ref to the Support Towers for drawing them.

Returns

const std::list<SupportTower*>&

13.9.3.14 GetTower()

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

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

Parameters

<i>coords</i>	The xy-coordinates where we want to look
---------------	--

Returns

const Tower*

13.9.3.15 IsActionPossible()

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

Used by [GUI](#) states to check what can be done.

Parameters

<i>coords</i>	The grid coordinates
<i>a</i>	Enumeration telling the desired action

Returns

bool

13.9.3.16 operator=()

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

Delete assignment operator.

13.9.3.17 RoundIsFinished()

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

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

Returns

bool

13.9.3.18 StartNextRound()

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

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

Returns

uint32_t The number of the round starting

13.9.3.19 TowerTurn()

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

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

Returns

bool

13.9.3.20 UpgradeTower()

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

Upgrades tower and reduces player's money.

Parameters

<i>coords</i>	position of tower to upgrade
---------------	------------------------------

13.9.4 Friends And Related Function Documentation

13.9.4.1 operator<<

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

Overload for the stream output operator.

13.9.5 Member Data Documentation

13.9.5.1 m_attakingTowers

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

13.9.5.2 m_enemies

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

13.9.5.3 m_enemyFactory

```
EnemyFactory Game::m_enemyFactory [private]
```

13.9.5.4 m_map

```
Map Game::m_map [private]
```

13.9.5.5 m_money

```
uint32_t Game::m_money [private]
```

13.9.5.6 m_playerHealth

```
uint32_t Game::m_playerHealth [private]
```

13.9.5.7 m_score

```
uint32_t Game::m_score [private]
```

13.9.5.8 m_supportingTowers

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

13.9.5.9 m_tickAttacks

```
std::list<std::pair<std::pair<int32_t, int32_t>, std::pair<int32_t, int32_t> > > Game::m_tickAttacks [private]
```

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

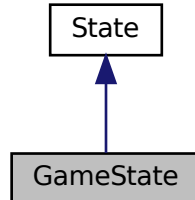
- [src/game.hpp](#)
- [src/game.cpp](#)

13.10 GameState Class Reference

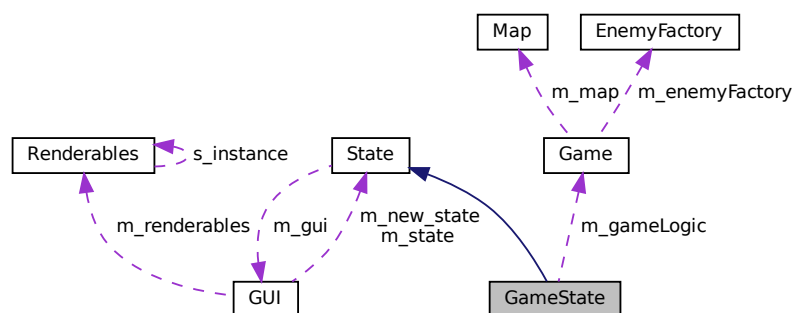
[GameState](#) class runs and draws the game part of the software.

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

Inheritance diagram for GameState:



Collaboration diagram for GameState:



Public Member Functions

- [GameState](#) ([GUI](#) &gui, [sf::RenderWindow](#) &>window, [Difficulty](#) difficulty, const std::string &filename)
Construct a new [Game State](#) object.
- [~GameState](#) ()
Destroy the [Game State](#) object Frees the buttons.
- void [Run](#) ()
Run and draw the game state. (Hosts the loop that Polls [GUI](#) events, calls the necessary backend methods and draws the window. Alternates between the building/upgrading phase and running the waves.) Exits the loop and calls GUIs [MoveToEndState](#)(int score) when the game is over.

Private Member Functions

- void [Priv_PollEvents](#) ()
Polls the events that have happened in [GUI](#).
- void [Priv_Draw](#) ()
Draws the frame to the screen.
- void [Priv_DrawBCG](#) ()
Draws the background to the window Assumes that the window has been properly cleared.
- void [Priv_DrawMap](#) ()
Draws the map to the window. Assumes that the window has been properly cleared.
- void [Priv_InitializeText](#) ([sf::Text](#) &text, int32_t x, int32_t y)
Initializes some text at set location.
- void [Priv_ClearSpeedHighlights](#) ()
Clears highlights from the game speed buttons.
- void [Priv_ChangeCircle](#) ([sf::CircleShape](#) &circle, unsigned int32_t range)
Sets a circle at desired position Used to draw ranges of towers.

Private Attributes

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

Additional Inherited Members

13.10.1 Detailed Description

[GameState](#) class runs and draws the game part of the software.

13.10.2 Constructor & Destructor Documentation

13.10.2.1 GameState()

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

Construct a new [Game State](#) object.

Parameters

<i>gui</i>	A ref to the GUI
<i>window</i>	A ref to the window
<i>difficulty</i>	The game difficulty
<i>filename</i>	The name of the map file

13.10.2.2 ~GameState()

```
GameState::~GameState ( )
```

Destroy the [Game State](#) object Frees the buttons.

13.10.3 Member Function Documentation

13.10.3.1 Priv_ChangeCircle()

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

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

Parameters

<i>circle</i>	The cirle used in drawing
<i>range</i>	The range to draw

13.10.3.2 Priv_ClearSpeedHighlights()

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

Clears highlights from the game speed buttons.

13.10.3.3 Priv_Draw()

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

Draws the frame to the screen.

13.10.3.4 Priv_DrawBCG()

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

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

13.10.3.5 Priv_DrawMap()

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

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

13.10.3.6 Priv_InitializeText()

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

Initializes some text at set location.

Parameters

<i>text</i>	The text to initialize
<i>x</i>	x-coordinate
<i>y</i>	y-coordinate

13.10.3.7 Priv_PollEvents()

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

Polls the events that have happened in [GUI](#).

13.10.3.8 Run()

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

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

Implements [State](#).

13.10.4 Member Data Documentation**13.10.4.1 m_buildPhase**

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

13.10.4.2 m_buttons

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

13.10.4.3 m_drawRange

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

13.10.4.4 m_drawUpgradeRange

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

13.10.4.5 m_frameInTick

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

13.10.4.6 m_gameLogic

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

13.10.4.7 m_gameOver

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

13.10.4.8 m_gameSpeed

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

13.10.4.9 m_healthText

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

13.10.4.10 m_mapTileSprites

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

13.10.4.11 m_moneyText

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

13.10.4.12 m_projectile

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

13.10.4.13 m_rangeCircle

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

13.10.4.14 m_roundNum

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

13.10.4.15 m_roundNumText

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

13.10.4.16 m_scoreText

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

13.10.4.17 m_selectedShape

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

13.10.4.18 m_selX

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

13.10.4.19 m_selY

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


13.10.4.20 m_upgradeRange

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

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

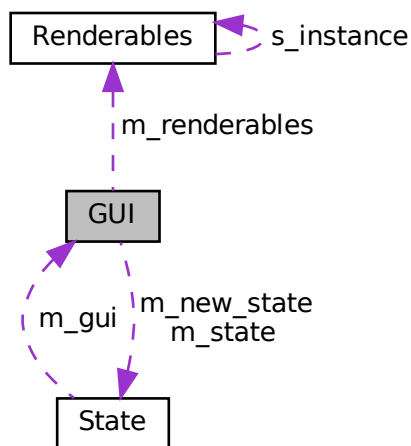
- [src/states/gamestate.hpp](#)
- [src/states/gamestate.cpp](#)

13.11 GUI Class Reference

A class to add elements to the Graphical User Interface.

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

Collaboration diagram for GUI:



Public Member Functions

- [GUI \(\)](#)
Construct a new [GUI](#) object This will create a [Renderables](#) object, which stores the sprites of everything.
- [~GUI \(\)](#)
Destroy the [GUI](#) object Frees the dynamically allocated member variables.
- void [init \(\)](#)
Initializes the different variables and makes the window ready.
- bool [running \(\)](#)
Tells if the [GUI](#) is open.
- void [update \(\)](#)
Updates [GUI](#) each frame.

- `Button * CreateButton (std::string text, int32_t x, int32_t y)`
Is used to create button objects.
- `TowerButton * CreateTowerButton (TowerType type, int32_t x, int32_t y)`
Is used to create towerbutton objects.
- `const sf::Font & GetFont () const`
Get the Font used here.
- `void ChangeState (State *state)`
Changes the state present in the GUI.

Private Member Functions

- `void Priv_DeleteState ()`
Private deletes the state in the GUI, if present.

Private Attributes

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

13.11.1 Detailed Description

A class to add elements to the Graphical User Interface.

13.11.2 Constructor & Destructor Documentation

13.11.2.1 GUI()

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

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

13.11.2.2 ~GUI()

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

Destroy the [GUI](#) object Frees the dynamically allocated member variables.

13.11.3 Member Function Documentation

13.11.3.1 ChangeState()

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

Changes the state present in the [GUI](#).

Parameters

<i>state</i>	The new state
--------------	---------------

13.11.3.2 CreateButton()

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

Is used to create button objects.

Parameters

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

Returns

Button*

13.11.3.3 CreateTowerButton()

```
TowerButton * GUI::CreateTowerButton (
    TowerType type,
```

```
int32_t x,  
int32_t y )
```

Is used to create towerbutton objects.

Parameters

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

Returns

Button*

13.11.3.4 GetFont()

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

Get the Font used here.

Returns

const sf::Font&

13.11.3.5 init()

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

Initializes the different variables and makes the window ready.

13.11.3.6 Priv_DeleteState()

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

Private deletes the state in the [GUI](#), if present.

13.11.3.7 running()

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

Tells if the [GUI](#) is open.

Returns

bool

13.11.3.8 update()

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

Updates [GUI](#) each frame.

13.11.4 Member Data Documentation

13.11.4.1 enemies

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

13.11.4.2 m_event

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

13.11.4.3 m_font

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

13.11.4.4 m_new_state

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

13.11.4.5 m_renderables

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

13.11.4.6 m_state

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

13.11.4.7 m_videoMode

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

13.11.4.8 m_window

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

13.11.4.9 start

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

13.11.4.10 x_velo

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

13.11.4.11 y_velo

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

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

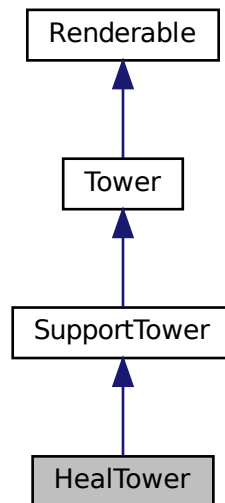
- [src/gui.hpp](#)
- [src/gui.cpp](#)

13.12 HealTower Class Reference

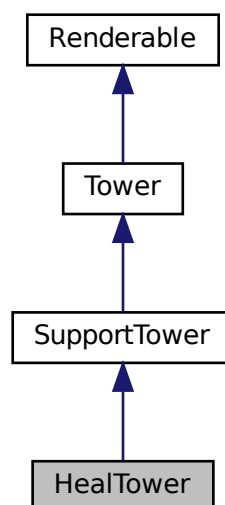
A tower which heals attacking towers.

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

Inheritance diagram for HealTower:



Collaboration diagram for HealTower:



Public Member Functions

- [HealTower](#) (uint32_t range, const std::pair< int32_t, int32_t > &coords, uint32_t healStrength, const std::string &name, const std::vector< sf::Sprite > &sprites)
Construct a new healing tower.
- [~HealTower](#) ()=default
Default destructor.
- void [Act](#) (std::list< [AttackingTower](#) * > &towers)
Goes through the attacking towers and heals them if in range and not in full health already.

Private Attributes

- uint32_t [m_healStrength](#)

Additional Inherited Members

13.12.1 Detailed Description

A tower which heals attacking towers.

13.12.2 Constructor & Destructor Documentation

13.12.2.1 HealTower()

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

Construct a new healing tower.

Parameters

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

13.12.2.2 ~HealTower()

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

Default destructor.

13.12.3 Member Function Documentation**13.12.3.1 Act()**

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

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

Parameters

<i>towers</i>	The towers on the game board
---------------	------------------------------

Implements [SupportTower](#).

13.12.4 Member Data Documentation**13.12.4.1 m_healStrength**

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

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

- [src/support_towers.hpp](#)
- [src/support_towers.cpp](#)

13.13 Highscores Class Reference

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

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

Public Member Functions

- [Highscores](#) (const std::string &filename="highscores.txt")
Construct a new [Highscores](#) object. Loads and sorts saved high scores.
- [~Highscores](#) ()=default
Default destructor.
- std::vector< std::string > [GetTop10](#) ()
Get the top 10 of high scores.
- std::string [GetTop10asString](#) ()
Get the top 10 of high scores as string.
- bool [AddScore](#) (const std::string &name, uint32_t score, [Difficulty](#) difficulty)
Saves new score to highscores. Can be only done once.

Private Member Functions

- void [Priv_LoadHighscores](#) ()
Private. Helper function to load the scores.
- void [Priv_SortHighscores](#) ()
Private. Helper function to sort the scores.

Private Attributes

- std::vector< std::tuple< std::string, uint32_t, [Difficulty](#) > > [m_highscores](#)
- std::string [m_filename](#)
- bool [m_saved](#)

13.13.1 Detailed Description

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

13.13.2 Constructor & Destructor Documentation

13.13.2.1 Highscores()

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

Construct a new [Highscores](#) object. Loads and sorts saved high scores.

Parameters

<i>filename</i>	name of the file containing high scores.
-----------------	--

13.13.2.2 ~Highscores()

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

Default destructor.

13.13.3 Member Function Documentation

13.13.3.1 AddScore()

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

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

Parameters

<i>name</i>	name of the player, cannot contain '.'
<i>score</i>	score of the player
<i>difficulty</i>	difficulty of the game

Returns

bool if adding score was successful

13.13.3.2 GetTop10()

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

Get the top 10 of high scores.

Returns

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

13.13.3.3 GetTop10asString()

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

Get the top 10 of high scores as string.

Returns

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

13.13.3.4 Priv_LoadHighscores()

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

Private. Helper function to load the scores.

13.13.3.5 Priv_SortHighscores()

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

Private. Helper function to sort the scores.

13.13.4 Member Data Documentation

13.13.4.1 m_filename

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

13.13.4.2 m_highscores

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

13.13.4.3 m_saved

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

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

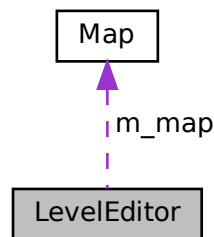
- [src/highscores.hpp](#)
- [src/highscores.cpp](#)

13.14 LevelEditor Class Reference

The logical core of the level editor state.

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

Collaboration diagram for LevelEditor:



Public Member Functions

- [LevelEditor](#) (int32_t width, int32_t height, std::string &mapPath)
Construct a new Level Editor object.
- [~LevelEditor](#) ()=default
Default destructor.
- bool [Edit](#) (std::pair< int32_t, int32_t > coordinate, [TileType](#) tile)
Edit a tile at the coordinates.
- bool [Save](#) ()
Saves the map to file May throw an exception.
- const [Map](#) & [GetMap](#) () const
Get the Map object.
- bool [Validate](#) ()
Checks if the map is valid.

Private Attributes

- [Map m_map](#)
- [int32_t m_width](#)
- [int32_t m_height](#)
- [std::string m_mapPath](#)

13.14.1 Detailed Description

The logical core of the level editor state.

13.14.2 Constructor & Destructor Documentation

13.14.2.1 LevelEditor()

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

Construct a new Level Editor object.

Parameters

<i>width</i>	The width of the map
<i>height</i>	The height of the map
<i>mapPath</i>	Where the map file is located

13.14.2.2 ~LevelEditor()

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

Default destructor.

13.14.3 Member Function Documentation

13.14.3.1 Edit()

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

Edit a tile at the coordinates.

Parameters

<i>coordinate</i>	The location of the edit
<i>tile</i>	What to change the tile into

Returns

bool Whether the editing was successfull

13.14.3.2 GetMap()

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

Get the [Map](#) object.

Returns

const [Map](#)&

13.14.3.3 Save()

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

Saves the map to file May throw an exception.

Returns

bool Whether the saving was successfull

13.14.3.4 Validate()

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

Checks if the map is valid.

Returns

bool

13.14.4 Member Data Documentation

13.14.4.1 m_height

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

13.14.4.2 m_map

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

13.14.4.3 m_mapPath

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

13.14.4.4 m_width

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

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

- [src/level_editor.hpp](#)
- [src/level_editor.cpp](#)

13.15 Map Class Reference

[Map](#) class that represents the grid map system behind each unique level.

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


Public Member Functions

- [Map](#) (int32_t width, int32_t height)
Construct a new [Map](#) object.
- [~Map](#) ()=default
Default destructor.
- bool [InitializeMap](#) (const std::string &filename)
Initialize a map from a text file to m_grid.
- const std::pair< int32_t, int32_t > [GetStart](#) () const
Get the start coordinates.
- const std::pair< int32_t, int32_t > [GetEnd](#) () const
Get the end coordinates.
- [TileType](#) [GetPos](#) (std::pair< int32_t, int32_t > coordinate) const
Get the tile at position (x, y)
- const std::map< std::pair< int32_t, int32_t >, [TileType](#) > & [GetGrid](#) () const
Get the Grid object.
- const std::vector< std::pair< int32_t, int32_t > > & [GetPath](#) () const
Get the Path object.
- bool [Edit](#) (std::pair< int32_t, int32_t > coordinate, [TileType](#) tile)
Change a tile on the map to another.
- bool [ValidateMap](#) ()
Validate the current m_grid. (Called by InitializeMap).
- bool [BuildPath](#) ()
Build the path of the current map in m_path.
- std::vector< std::pair< int32_t, int32_t > > [GetNeighbors](#) (int32_t x, int32_t y)
Get the vertical and horizontal neighbors of (x, y).
- bool [TestTilePos](#) (std::pair< int32_t, int32_t > coordinate, [TileType](#) tile)
Tests if the position is valid for the tile.

Private Attributes

- int32_t [m_width](#)
- int32_t [m_height](#)
- std::map< std::pair< int32_t, int32_t >, [TileType](#) > [m_grid](#)
- std::pair< int32_t, int32_t > [m_start](#)
- std::pair< int32_t, int32_t > [m_end](#)
- std::vector< std::pair< int32_t, int32_t > > [m_path](#)

13.15.1 Detailed Description

[Map](#) class that represents the grid map system behind each unique level.

13.15.2 Constructor & Destructor Documentation

13.15.2.1 Map()

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

Construct a new [Map](#) object.

Parameters

<i>width</i>	Limit of x-coordinate of the map.
<i>height</i>	Limit of y-coordinate of the map.

13.15.2.2 ~Map()

```
Map::~~Map ( ) [default]
```

Default destructor.

13.15.3 Member Function Documentation**13.15.3.1 BuildPath()**

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

Build the path of the current map in m_path.

Returns

true if path is built successfully

false if path cannot be built.

13.15.3.2 Edit()

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

Change a tile on the map to another.

Parameters

<i>coordinate</i>	Where to perform the change
<i>tile</i>	What to change the tile into

Returns

true if editing was successful
false if editing was not successful.

13.15.3.3 GetEnd()

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

Get the end coordinates.

Returns

`std::pair<int32_t, int32_t>`

13.15.3.4 GetGrid()

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

Get the Grid object.

Returns

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

13.15.3.5 GetNeighbors()

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

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

Parameters

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

Returns

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

13.15.3.6 GetPath()

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

Get the Path object.

Returns

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

13.15.3.7 GetPos()

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

Get the tile at position (x, y)

Parameters

<i>coordinate</i>	The position
-------------------	--------------

Returns

TilyType

13.15.3.8 GetStart()

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

Get the start coordinates.

Returns

```
std::pair<int32_t, int32_t>
```

13.15.3.9 InitializeMap()

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

Initialize a map from a text file to m_grid.

Parameters

<i>filename</i>	The file to be used
-----------------	---------------------

Returns

bool Whether the initialization was successfull

13.15.3.10 TestTilePos()

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

Tests if the position is valid for the tile.

Parameters

<i>coordinate</i>	The position
<i>tile</i>	The type of the tile

Returns

true if position is valid

false if position is invalid

13.15.3.11 ValidateMap()

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

Validate the current m_grid. (Called by InitializeMap).

Returns

true if map is valid

false if map is not valid

13.15.4 Member Data Documentation

13.15.4.1 m_end

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

13.15.4.2 m_grid

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

13.15.4.3 m_height

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

13.15.4.4 m_path

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

13.15.4.5 m_start

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

13.15.4.6 m_width

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

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

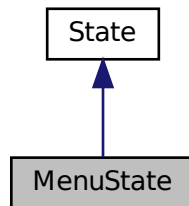
- [src/map.hpp](#)
- [src/map.cpp](#)

13.16 MenuState Class Reference

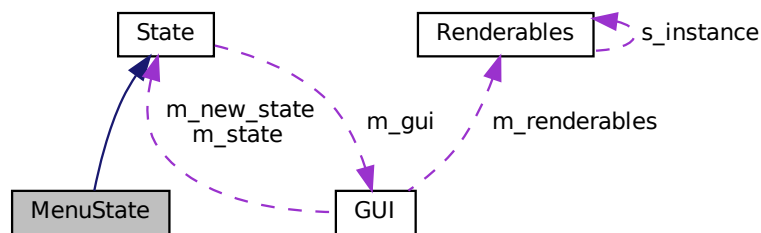
A state run by [GUI](#) corresponding to the main menu state.

```
#include <menustate.hpp>
```

Inheritance diagram for MenuState:



Collaboration diagram for MenuState:



Public Member Functions

- [MenuState](#) ([GUI](#) &gui, sf::RenderWindow &window)
Construct a new Menu [State](#) object.
- [~MenuState](#) ()
Destroy the Menu [State](#) object Frees the buttons.
- void [Run](#) ()
Run and draw the Menu state. (Hosts the loop that Polls [GUI](#) events, calls the necessary backend methods and draws the window. Has options to select difficulty and map file. Has buttons to game editor and to play.) Exits the loop and calls GUIs MoveToGameState(int score) when the user presses "Play".

Private Member Functions

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

Private Attributes

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

Additional Inherited Members

13.16.1 Detailed Description

A state run by [GUI](#) corresponding to the main menu state.

13.16.2 Constructor & Destructor Documentation

13.16.2.1 MenuState()

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

Construct a new Menu [State](#) object.

Parameters

<i>gui</i>	A ref to the GUI
<i>window</i>	A ref to the window

13.16.2.2 ~MenuState()

```
MenuState::~MenuState ( )
```

Destroy the Menu [State](#) object Frees the buttons.

13.16.3 Member Function Documentation

13.16.3.1 Draw()

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

13.16.3.2 PollEvents()

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

13.16.3.3 Run()

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

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

Implements [State](#).

13.16.3.4 RunLevelEditor()

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

13.16.4 Member Data Documentation

13.16.4.1 m_buttons

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

13.16.4.2 m_difficulty

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

13.16.4.3 m_editing

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

13.16.4.4 m_height

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

13.16.4.5 m_selectedMap

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

13.16.4.6 m_texts

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

13.16.4.7 m_width

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

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

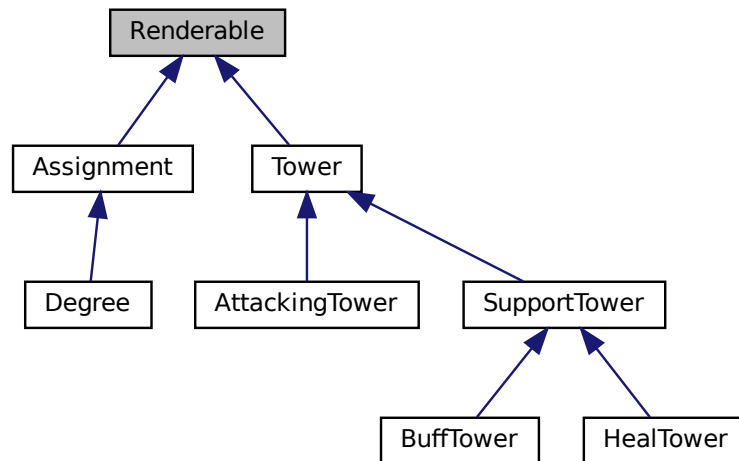
- [src/states/menustate.hpp](#)
- [src/states/menustate.cpp](#)

13.17 Renderable Class Reference

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

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

Inheritance diagram for Renderable:



Public Member Functions

- virtual `~Renderable()`
A virtual destructor.
- const std::string & `EntityName()` const
Get the name of the entity Mainly for debugging.
- sf::Sprite & `GetSprite()`
Get the Sprite of this entity For rendering purposes.

Protected Member Functions

- `Renderable` (const std::string &entityName, const sf::Sprite &sprite)
Construct a new `Renderable` object.
- void `SetSprite` (const sf::Sprite &newSprite)
Set the Sprite object Used by the towers when they are upgraded.

Protected Attributes

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

13.17.1 Detailed Description

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

13.17.2 Constructor & Destructor Documentation

13.17.2.1 ~Renderable()

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

A virtual destructor.

13.17.2.2 Renderable()

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

Construct a new [Renderable](#) object.

Parameters

<i>entityName</i>	The name
<i>sprite</i>	The sprite used by the entity

13.17.3 Member Function Documentation

13.17.3.1 EntityName()

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

Get the name of the entity Mainly for debugging.

Returns

const std::string&

13.17.3.2 GetSprite()

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

Get the Sprite of this entity For rendering purposes.

Returns

sf::Sprite&

13.17.3.3 SetSprite()

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

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

Parameters

<i>newSprite</i>	A ref to the new sprite
------------------	-------------------------

13.17.4 Member Data Documentation

13.17.4.1 m_entityName

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

13.17.4.2 m_sprite

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

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

- [src/renderable.hpp](#)
- [src/renderable.cpp](#)

13.18 Renderables Class Reference

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

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

Collaboration diagram for Renderables:



Public Member Functions

- [Renderables](#) ()

Static Public Member Functions

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

Private Attributes

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

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

Static Private Attributes

- static [Renderables](#) * [s_instance](#) = nullptr

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

13.18.1 Detailed Description

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

13.18.2 Constructor & Destructor Documentation

13.18.2.1 Renderables()

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

13.18.3 Member Function Documentation

13.18.3.1 getAttackSound()

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


13.18.3.2 getBachelor1Sprite()

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

13.18.3.3 getBachelor2Sprite()

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

13.18.3.4 getBachelor3Sprite()

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

13.18.3.5 getBachelorsThesisSprite()

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

13.18.3.6 getBackgroundSprite()

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

13.18.3.7 getBscSprite()

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

13.18.3.8 getCalculatorSprite()

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

13.18.3.9 getCoffeeTableSprite()

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

13.18.3.10 getDoctor1Sprite()

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

13.18.3.11 getDoctor2Sprite()

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

13.18.3.12 getDoctor3Sprite()

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

13.18.3.13 getDoctoralThesisSprite()

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

13.18.3.14 getDscSprite()

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

13.18.3.15 getEndtileSprite()

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

13.18.3.16 getEssaySprite()

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

13.18.3.17 getFreshman1Sprite()

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

13.18.3.18 getFreshman2Sprite()

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

13.18.3.19 getFreshman3Sprite()

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

13.18.3.20 getHomeworkSprite()

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

13.18.3.21 getMaster1Sprite()

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

13.18.3.22 getMaster2Sprite()

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

13.18.3.23 getMaster3Sprite()

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

13.18.3.24 getMastersThesisSprite()

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

13.18.3.25 getMenuBackgroundSprite()

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

13.18.3.26 getMscSprite()

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

13.18.3.27 getPathtileSprite()

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

13.18.3.28 getProjectSprite()

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

13.18.3.29 getSelectSound()

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

13.18.3.30 getStarttileSprite()

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

13.18.3.31 getTeekkari1Sprite()

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

13.18.3.32 getTeekkari2Sprite()

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

13.18.3.33 getTeekkari3Sprite()

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

13.18.3.34 getTowertileSprite()

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

13.18.4 Member Data Documentation

13.18.4.1 attack

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

13.18.4.2 attack_sound

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

13.18.4.3 bachelor_1

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

13.18.4.4 bachelor_1_sprite

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

13.18.4.5 bachelor_2

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

13.18.4.6 bachelor_2_sprite

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

13.18.4.7 bachelor_3

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

13.18.4.8 bachelor_3_sprite

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

13.18.4.9 bachelors_thesis

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

13.18.4.10 bachelors_thesis_sprite

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

13.18.4.11 background

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

13.18.4.12 background_sprite

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

13.18.4.13 bsc

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

13.18.4.14 bsc_sprite

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

13.18.4.15 calculator

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

13.18.4.16 calculator_sprite

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

13.18.4.17 coffee_table

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

13.18.4.18 coffee_table_sprite

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

13.18.4.19 doctor_1

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

13.18.4.20 doctor_1_sprite

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

13.18.4.21 doctor_2

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

13.18.4.22 doctor_2_sprite

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

13.18.4.23 doctor_3

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

13.18.4.24 doctor_3_sprite

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

13.18.4.25 doctoral_thesis

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

13.18.4.26 doctoral_thesis_sprite

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

13.18.4.27 dsc

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

13.18.4.28 dsc_sprite

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

13.18.4.29 endTile

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

13.18.4.30 endTile_sprite

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


13.18.4.31 essay

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

13.18.4.32 essay_sprite

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

13.18.4.33 freshman_1

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

13.18.4.34 freshman_1_sprite

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

13.18.4.35 freshman_2

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

13.18.4.36 freshman_2_sprite

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

13.18.4.37 freshman_3

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

13.18.4.38 freshman_3_sprite

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

13.18.4.39 homework

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

13.18.4.40 homework_sprite

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

13.18.4.41 master_1

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

13.18.4.42 master_1_sprite

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

13.18.4.43 master_2

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

13.18.4.44 master_2_sprite

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

13.18.4.45 master_3

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

13.18.4.46 master_3_sprite

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

13.18.4.47 masters_thesis

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

13.18.4.48 masters_thesis_sprite

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

13.18.4.49 menu_background

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

13.18.4.50 menu_background_sprite

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

13.18.4.51 msc

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

13.18.4.52 msc_sprite

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

13.18.4.53 pathTile

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

13.18.4.54 pathTile_sprite

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

13.18.4.55 project

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

13.18.4.56 project_sprite

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

13.18.4.57 s_instance

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

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

13.18.4.58 select

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

13.18.4.59 select_sound

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

13.18.4.60 startTile

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

13.18.4.61 startTile_sprite

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

13.18.4.62 teekkari_1

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

13.18.4.63 teekkari_1_sprite

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

13.18.4.64 teekkari_2

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

13.18.4.65 teekkari_2_sprite

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

13.18.4.66 teekkari_3

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

13.18.4.67 teekkari_3_sprite

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

13.18.4.68 towerTile

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

13.18.4.69 towerTile_sprite

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

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

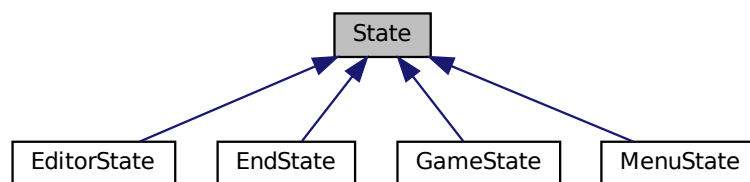
- [src/renderables.hpp](#)
- [src/renderables.cpp](#)

13.19 State Class Reference

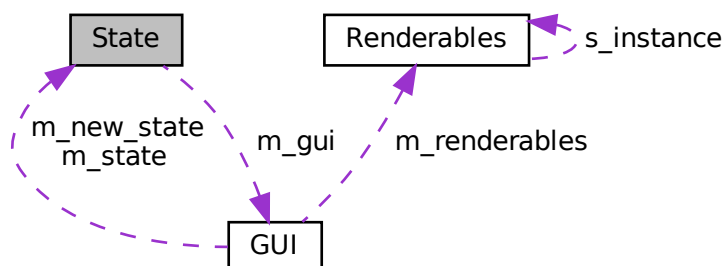
Abstract [State](#) class represents the classes that run and draw the different software states.

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

Inheritance diagram for State:



Collaboration diagram for State:



Public Member Functions

- [State](#) ([GUI](#) &gui, [sf::RenderWindow](#) &>window)
Construct a new [State](#) object.
- virtual [~State](#) ()=default
Virtual destructor.
- virtual void [Run](#) ()=0
*Run the state. (Hosts the loop that Polls [GUI](#) events, calls the necessary backend methods and draws the window.)
A pure virtual function.*

Protected Attributes

- [GUI](#) & [m_gui](#)
- [sf::RenderWindow](#) & [m_window](#)
- [sf::Event](#) [m_event](#)

13.19.1 Detailed Description

Abstract [State](#) class represents the classes that run and draw the different software states.

13.19.2 Constructor & Destructor Documentation

13.19.2.1 State()

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

Construct a new [State](#) object.

Parameters

<i>gui</i>	A ref to the gui
<i>window</i>	A ref to the window

13.19.2.2 ~State()

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

Virtual destructor.

13.19.3 Member Function Documentation

13.19.3.1 Run()

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

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

Implemented in [EndState](#), [EditorState](#), [MenuState](#), and [GameState](#).

13.19.4 Member Data Documentation

13.19.4.1 m_event

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

13.19.4.2 m_gui

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

13.19.4.3 m_window

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

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

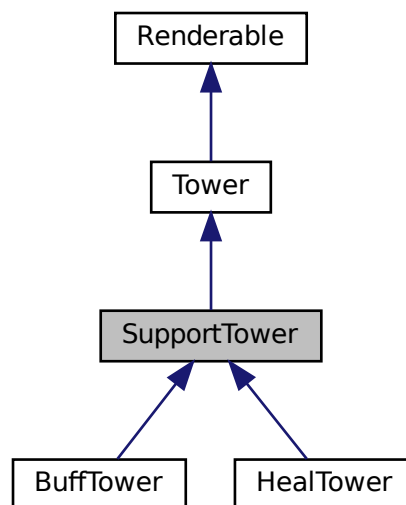
- [src/states/state.hpp](#)

13.20 SupportTower Class Reference

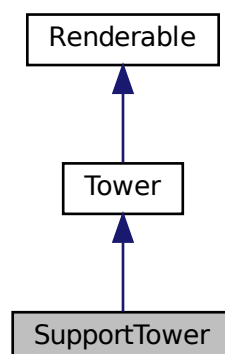
A virtual base class for the supporting towers Cannot be directly instantiated.

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

Inheritance diagram for SupportTower:



Collaboration diagram for SupportTower:



Public Member Functions

- virtual `~SupportTower()`
Virtual destructor.
- bool `IsUpgradeable` (uint32_t money) const
No `SupportTower` can be upgraded.
- virtual void `Act` (std::list< `AttackingTower` * > &towers)=0
Used by the supporting towers to buff/heal the attacking towers The towers go through the attacking towers present on the game board and apply buffs/heal those that are within range.

Static Public Member Functions

- static `SupportTower` * `Calculator` (const std::pair< int32_t, int32_t > &coords)
Creates a `BuffTower` called calculator.
- static `SupportTower` * `CoffeeTable` (const std::pair< int32_t, int32_t > &coords)
Creates a `HealTower` called coffee_table.

Protected Member Functions

- `SupportTower` (uint32_t range, const std::pair< int32_t, int32_t > &coords, const std::string &name, const std::vector< sf::Sprite > &sprites)
Construct a new Support `Tower` object.

Friends

- std::ostream & `operator<<` (std::ostream &os, const `SupportTower` &st)
Overload to stream output operator.

Additional Inherited Members

13.20.1 Detailed Description

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

13.20.2 Constructor & Destructor Documentation

13.20.2.1 SupportTower()

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

Construct a new Support `Tower` object.

Parameters

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

13.20.2.2 ~SupportTower()

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

Virtual destructor.

13.20.3 Member Function Documentation

13.20.3.1 Act()

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

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

Parameters

<i>towers</i>	The towers present on the game board
---------------	--------------------------------------

Implemented in [HealTower](#), and [BuffTower](#).

13.20.3.2 Calculator()

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

Creates a [BuffTower](#) called calculator.

Parameters

<i>coords</i>	Where to create the tower
---------------	---------------------------

Returns

SupportTower*

13.20.3.3 CoffeeTable()

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

Creates a [HealTower](#) called coffee_table.

Parameters

<i>coords</i>	Where to create the tower
---------------	---------------------------

Returns

SupportTower*

13.20.3.4 IsUpgradeable()

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

No [SupportTower](#) can be upgraded.

Returns

false

Implements [Tower](#).

13.20.4 Friends And Related Function Documentation**13.20.4.1 operator<<**

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

Overload to stream output operator.

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

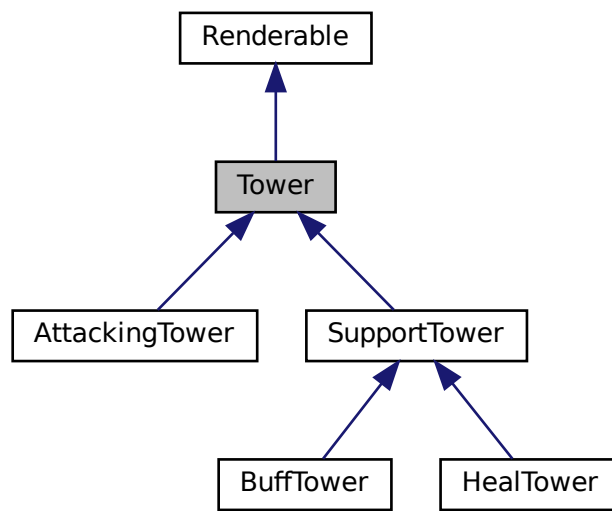
- [src/support_towers.hpp](#)
- [src/support_towers.cpp](#)

13.21 Tower Class Reference

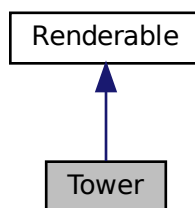
An abstract base-class for the towers. Sub-classes will be Attacking towers and supporting towers. Cannot be directly instantiated.

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

Inheritance diagram for Tower:



Collaboration diagram for Tower:



Public Member Functions

- virtual `~Tower()`
Virtual destructor.

- `const std::pair< int32_t, int32_t > & GetCoords () const`
Get the coordinates of the tower.
- `virtual bool IsUpgradeable (uint32_t money) const =0`
Tells if the tower can be upgraded with set amount of money Supporting towers can never be upgraded.
- `uint32_t GetRange () const`
Get the range of the tower.

Static Public Attributes

- `static const std::map< TowerType, uint32_t > towerPrices`
The prices of the different kinds of towers.
- `static const std::map< TowerType, uint32_t > towerRanges`
The base ranges of the towers.
- `static const std::map< TowerType, uint32_t > towerPowers`
The base powers of the attacking towers.
- `static const std::map< TowerType, uint32_t > towerHealths`
The healths of the attacking towers.

Protected Member Functions

- `Tower (uint32_t range, const std::pair< int32_t, int32_t > &coords, const std::string &name, const std::vector< sf::Sprite > &sprites)`
Plain towers are not meant to be constructed, only the subclassess are.

Protected Attributes

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

13.21.1 Detailed Description

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

13.21.2 Constructor & Destructor Documentation

13.21.2.1 ~Tower()

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

Virtual destructor.

13.21.2.2 Tower()

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

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

Parameters

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

13.21.3 Member Function Documentation

13.21.3.1 GetCoords()

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

Get the coordinates of the tower.

Returns

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

13.21.3.2 GetRange()

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

Get the range of the tower.

Returns

```
uint32_t
```

13.21.3.3 IsUpgradeable()

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

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

Parameters

<i>money</i>	The amount of money the player has
--------------	------------------------------------

Returns

bool

Implemented in [AttackingTower](#), and [SupportTower](#).

13.21.4 Member Data Documentation

13.21.4.1 m_allSprites

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

13.21.4.2 m_coords

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

13.21.4.3 m_range

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

13.21.4.4 towerHealts

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

Initial value:

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

The healths of the attacking towers.

13.21.4.5 towerPowers

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

Initial value:

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

The base powers of the attacking towers.

13.21.4.6 towerPrices

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

Initial value:

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

The prices of the different kinds of towers.

13.21.4.7 towerRanges

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

Initial value:

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

The base ranges of the towers.

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

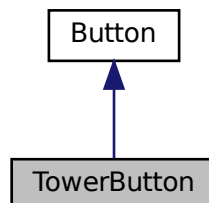
- [src/tower.hpp](#)
- [src/tower.cpp](#)

13.22 TowerButton Class Reference

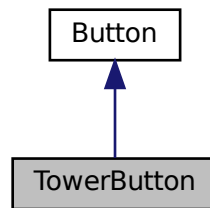
A subclass of buttons, for building the towers.

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

Inheritance diagram for TowerButton:



Collaboration diagram for TowerButton:



Public Member Functions

- `TowerButton (TowerType tower, int32_t x, int32_t y, sf::Font &font)`
Construct a new [Tower Button](#) object.
- `~TowerButton ()=default`
Default destructor.
- `void drawButton (sf::RenderWindow &window)`
Draws the button on the window.
- `void disableButton ()`
Makes the button grayed out.
- `void enableButton ()`
Restores the original look of the button.

Private Attributes

- `sf::Sprite m_sprite`
- `std::string m_name`

Additional Inherited Members

13.22.1 Detailed Description

A subclass of buttons, for building the towers.

13.22.2 Constructor & Destructor Documentation

13.22.2.1 TowerButton()

```

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

```

Construct a new [Tower Button](#) object.

Parameters

<i>tower</i>	The type of tower this is supposed to represent
<i>x</i>	The x-coordinate of the upper left corner
<i>y</i>	The y-coordinate of the upper left corner
<i>font</i>	The font used by the text

13.22.2.2 ~TowerButton()

```
TowerButton::~~TowerButton ( ) [default]
```

Default destructor.

13.22.3 Member Function Documentation**13.22.3.1 disableButton()**

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

Makes the button grayed out.

Reimplemented from [Button](#).

13.22.3.2 drawButton()

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

Draws the button on the window.

Parameters

<i>window</i>	A ref to the window where to draw
---------------	-----------------------------------

Reimplemented from [Button](#).

13.22.3.3 enableButton()

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

Restores the original look of the button.

Reimplemented from [Button](#).

13.22.4 Member Data Documentation

13.22.4.1 m_name

```
std::string TowerButton::m_name [private]
```

13.22.4.2 m_sprite

```
sf::Sprite TowerButton::m_sprite [private]
```

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

- [src/button.hpp](#)
- [src/button.cpp](#)

Chapter 14

File Documentation

14.1 build/CMakeCache.txt File Reference

14.2 build/CMakeFiles/3.16.3/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define C_DIALECT`

Functions

- `int main (int argc, char *argv[])`

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" "" "]"`
- `char const * info_platform = "INFO" ":" "platform[" "]"`
- `char const * info_arch = "INFO" ":" "arch[" "]"`
- `const char * info_language_dialect_default`

14.2.1 Macro Definition Documentation

14.2.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

14.2.1.2 C_DIALECT

```
#define C_DIALECT
```

14.2.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

14.2.1.4 DEC

```
#define DEC(  
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \
('0' + ((n) / 1000000) % 10), \
('0' + ((n) / 100000) % 10), \
('0' + ((n) / 10000) % 10), \
('0' + ((n) / 1000) % 10), \
('0' + ((n) / 100) % 10), \
('0' + ((n) / 10) % 10), \
('0' + ((n) % 10))
```

14.2.1.5 HEX

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \
('0' + ((n) >> 24 & 0xF)), \
('0' + ((n) >> 20 & 0xF)), \
('0' + ((n) >> 16 & 0xF)), \
('0' + ((n) >> 12 & 0xF)), \
('0' + ((n) >> 8 & 0xF)), \
('0' + ((n) >> 4 & 0xF)), \
('0' + ((n) & 0xF))
```


14.2.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

14.2.1.7 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY_HELPER(X)
```

14.2.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

14.2.2 Function Documentation

14.2.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

14.2.3 Variable Documentation

14.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" " "]"
```

14.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" " " "]"
```

14.2.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
= "INFO" ":" "dialect_default[" " "]"
```

14.2.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" " "]"
```

14.3 build/CMakeFiles/3.16.3/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define CXX_STD __cplusplus`

Functions

- `int main (int argc, char *argv[])`

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" " ""]"`
- `char const * info_platform = "INFO" ":" "platform[" ""]"`
- `char const * info_arch = "INFO" ":" "arch[" ""]"`
- `const char * info_language_dialect_default`

14.3.1 Macro Definition Documentation

14.3.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

14.3.1.2 COMPILER_ID

```
#define COMPILER_ID ""
```

14.3.1.3 CXX_STD

```
#define CXX_STD __cplusplus
```

14.3.1.4 DEC

```
#define DEC(
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \
('0' + ((n) / 1000000) % 10), \
('0' + ((n) / 100000) % 10), \
('0' + ((n) / 10000) % 10), \
('0' + ((n) / 1000) % 10), \
('0' + ((n) / 100) % 10), \
('0' + ((n) / 10) % 10), \
('0' + ((n) % 10))
```

14.3.1.5 HEX

```
#define HEX(
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \
('0' + ((n) >> 24 & 0xF)), \
('0' + ((n) >> 20 & 0xF)), \
('0' + ((n) >> 16 & 0xF)), \
('0' + ((n) >> 12 & 0xF)), \
('0' + ((n) >> 8 & 0xF)), \
('0' + ((n) >> 4 & 0xF)), \
('0' + ((n) & 0xF))
```

14.3.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

14.3.1.7 STRINGIFY

```
#define STRINGIFY(  
    X )  STRINGIFY_HELPER(X)
```

14.3.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X )  #X
```

14.3.2 Function Documentation

14.3.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

14.3.3 Variable Documentation

14.3.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" " ""]"
```

14.3.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" " " ""]"
```

14.3.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
= "INFO" ":" "dialect_default["  
    "98"  
    "]"
```

14.3.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" " ]"
```

14.4 build/CMakeFiles/TargetDirectories.txt File Reference

14.5 build/CMakeFiles/tower-defense.dir/link.txt File Reference

14.6 CMakeLists.txt File Reference

14.7 doc/readme.md File Reference

14.8 libs/readme.md File Reference

14.9 plan/readme.md File Reference

14.10 src/readme.md File Reference

14.11 tests/readme.md File Reference

14.12 highscores.txt File Reference

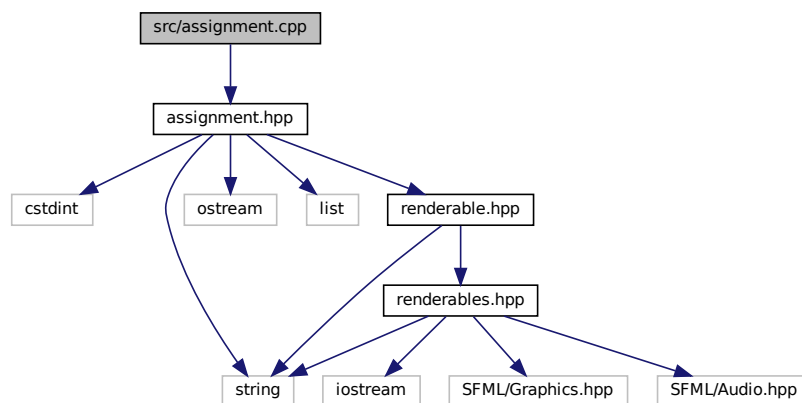
14.13 Meeting-notes.md File Reference

14.14 README.md File Reference

14.15 src/assignment.cpp File Reference

```
#include "assignment.hpp"
```

Include dependency graph for assignment.cpp:



Functions

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

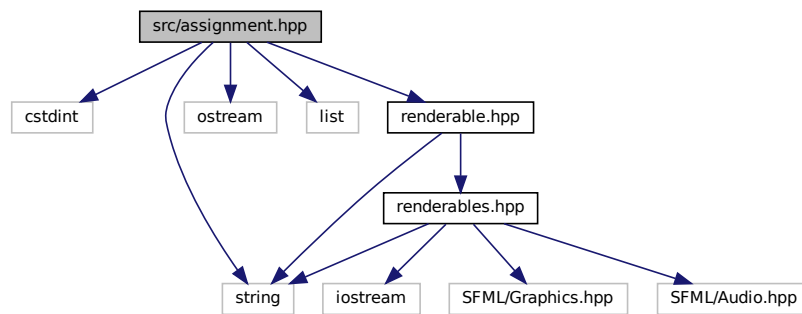
14.15.1 Function Documentation

14.15.1.1 `operator<<()`

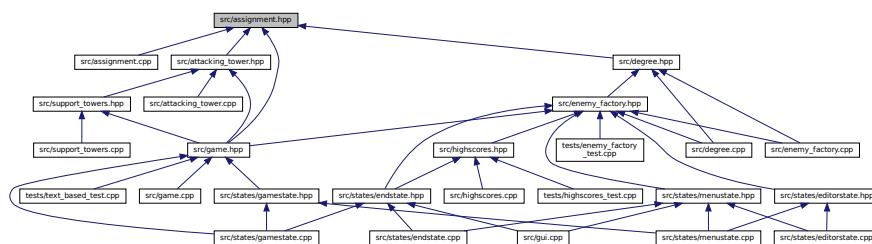
```
std::ostream& operator<< (
    std::ostream & os,
    const Assignment & as )
```

14.16 `src/assignment.hpp` File Reference

```
#include <cstdint>
#include <string>
#include <ostream>
#include <list>
#include "renderable.hpp"
Include dependency graph for assignment.hpp:
```



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



Classes

- class [Assignment](#)
Generic Enemy class, these just die when they are killed.

Enumerations

- enum [Enemy](#) {
 [Homework](#), [Essay](#), [Project](#), [B_Thesis](#),
 [M_Thesis](#), [D_Thesis](#), [BSc](#), [MSc](#),
 [DSc](#) }
An enumeration for the different enemy types.

14.16.1 Enumeration Type Documentation

14.16.1.1 Enemy

enum [Enemy](#)

An enumeration for the different enemy types.

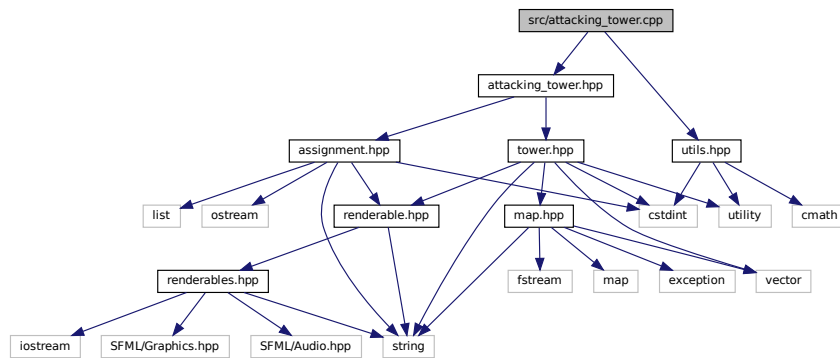
Enumerator

Homework	
Essay	
Project	
B_Thesis	
M_Thesis	
D_Thesis	
BSc	
MSc	
DSc	

14.17 src/attacking_tower.cpp File Reference

```
#include "attacking_tower.hpp"  
#include "utils.hpp"
```

Include dependency graph for `attacking_tower.cpp`:



Functions

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

14.17.1 Function Documentation

14.17.1.1 `operator<<()`

```

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

```

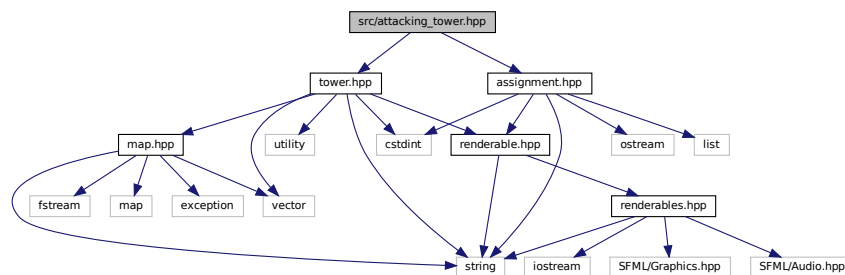
14.18 `src/attacking_tower.hpp` File Reference

```

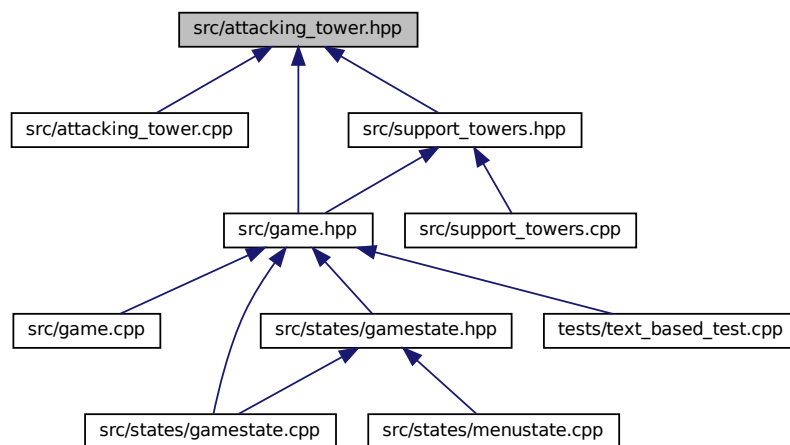
#include "tower.hpp"
#include "assignment.hpp"

```

Include dependency graph for `attacking_tower.hpp`:



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



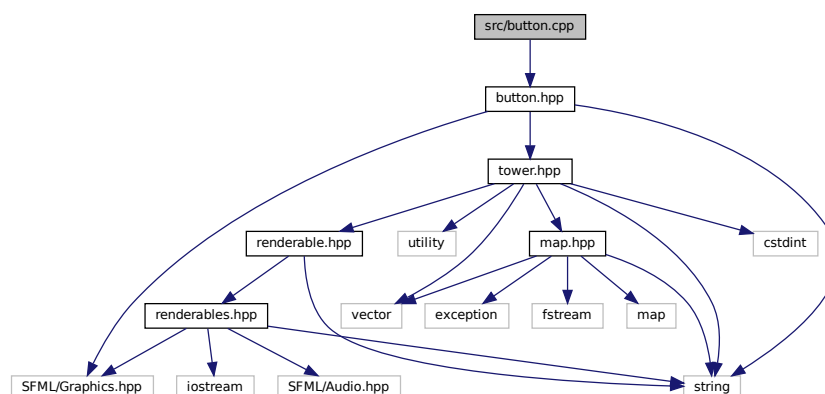
Classes

- class [AttackingTower](#)
A class for the offensive towers.

14.19 src/button.cpp File Reference

```
#include "button.hpp"
```

Include dependency graph for button.cpp:



Macros

- #define [TILE_SIZE](#) 30

14.19.1 Macro Definition Documentation

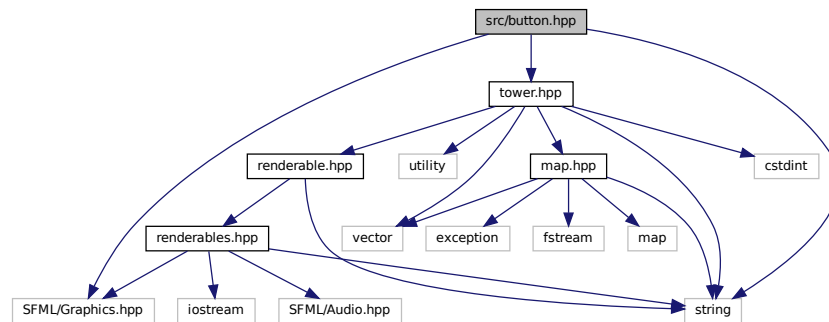
14.19.1.1 TILE_SIZE

```
#define TILE_SIZE 30
```

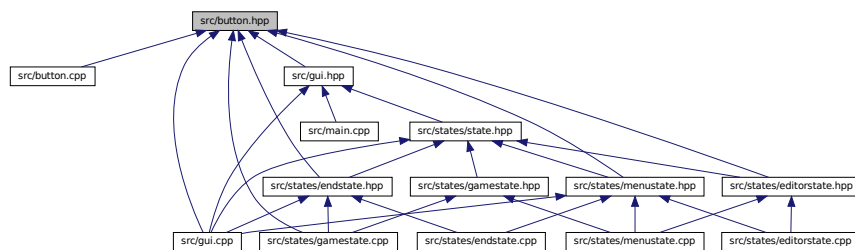
14.20 src/button.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <string>
#include "tower.hpp"
```

Include dependency graph for button.hpp:



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

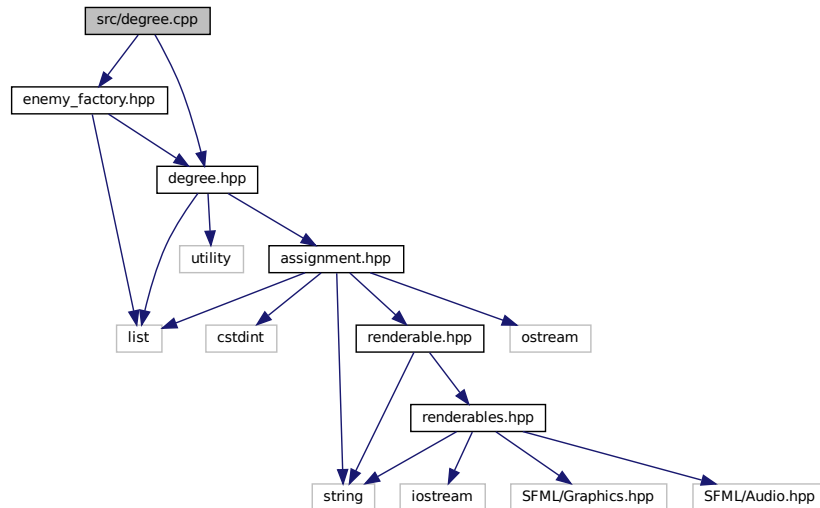


Classes

- class [Button](#)
A class for the buttons in the game.
- class [TowerButton](#)
A subclass of buttons, for building the towers.

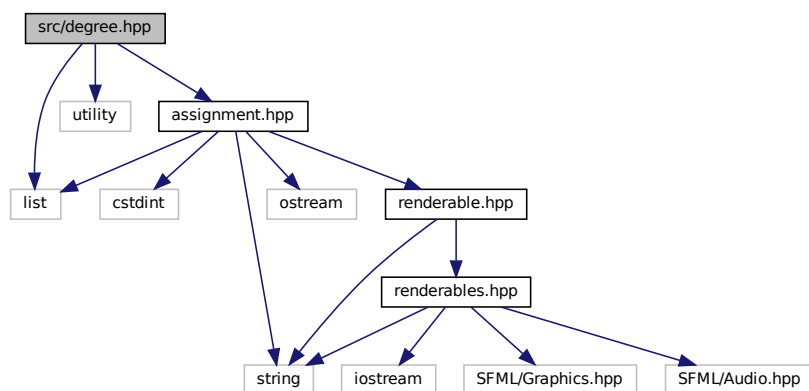
14.21 src/degree.cpp File Reference

```
#include "degree.hpp"
#include "enemy_factory.hpp"
Include dependency graph for degree.cpp:
```



14.22 src/degree.hpp File Reference

```
#include <list>
#include <utility>
#include "assignment.hpp"
Include dependency graph for degree.hpp:
```



Functions

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

14.23.1 Macro Definition Documentation

14.23.1.1 DETAILED_DEBUG_PRINT

```
#define DETAILED_DEBUG_PRINT 0
```

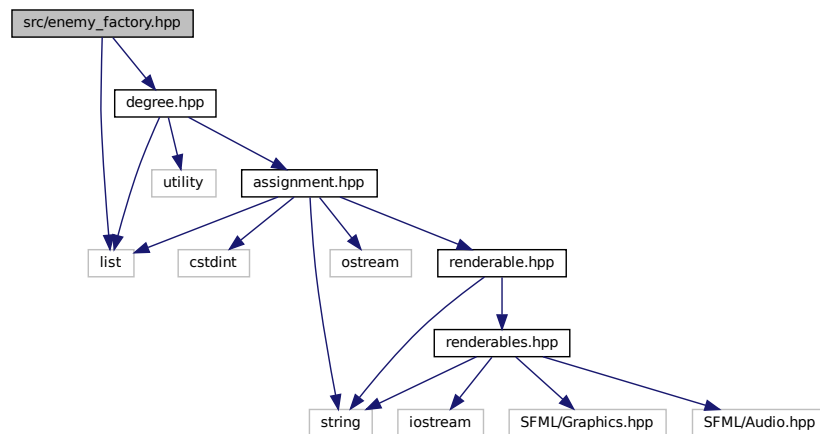
14.23.2 Function Documentation

14.23.2.1 operator<<()

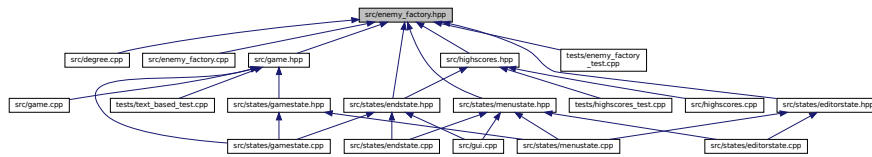
```
std::ostream& operator<< (
    std::ostream & os,
    const EnemyFactory & ef )
```

14.24 src/enemy_factory.hpp File Reference

```
#include <list>
#include "degree.hpp"
Include dependency graph for enemy_factory.hpp:
```



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



Classes

- class [EnemyFactory](#)

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

Enumerations

- enum [Difficulty](#) { [Easy](#), [Medium](#), [Hard](#) }

An enumeration for the game difficulties.

14.24.1 Enumeration Type Documentation

14.24.1.1 Difficulty

enum [Difficulty](#)

An enumeration for the game difficulties.

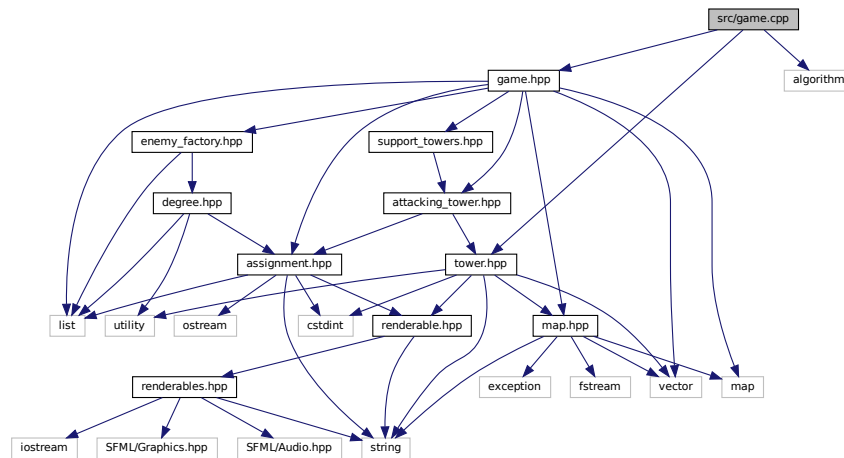
Enumerator

Easy	
Medium	
Hard	

14.25 src/game.cpp File Reference

```
#include "game.hpp"
#include <algorithm>
#include "tower.hpp"
```

Include dependency graph for game.cpp:



Functions

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

14.25.1 Function Documentation

14.25.1.1 operator<<()

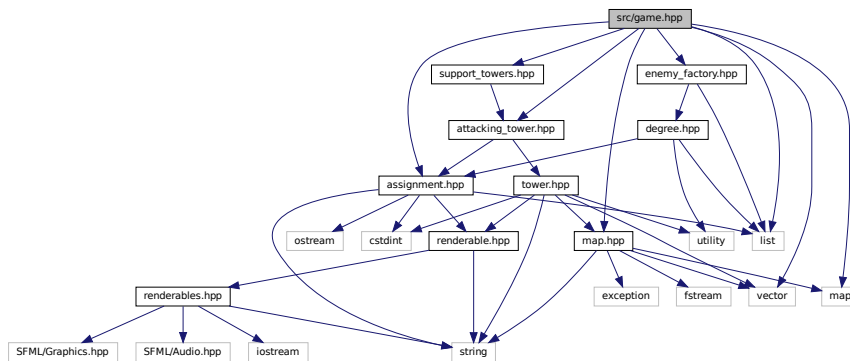
```
std::ostream& operator<< (
    std::ostream & os,
    const Game & game )
```

14.26 src/game.hpp File Reference

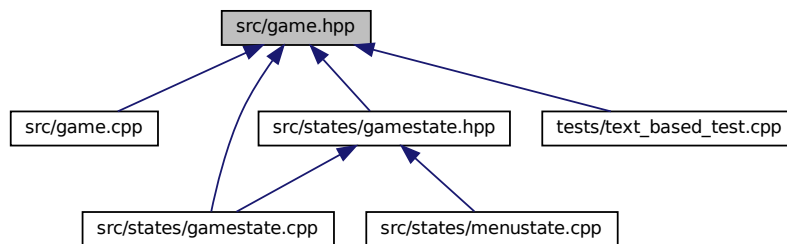
```
#include <list>
#include <map>
#include <vector>
#include "assignment.hpp"
#include "attacking_tower.hpp"
#include "enemy_factory.hpp"
#include "map.hpp"
```

```
#include "support_towers.hpp"
```

Include dependency graph for game.hpp:



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



Classes

- class [Game](#)
A class which holds the logic of the game.

Enumerations

- enum [Action](#) {
[BuyFreshman](#), [BuyTeekkari](#), [BuyBachelor](#), [BuyMaster](#),
[BuyDoctor](#), [BuyCalculator](#), [BuyCoffeeTable](#), [UpgradeTower](#),
[DestroyTower](#) }
 An enumeration for different actions on grid cells Used to test the possibility of actions.

14.26.1 Enumeration Type Documentation

14.26.1.1 Action

```
enum Action
```

An enumeration for different actions on grid cells Used to test the possibility of actions.

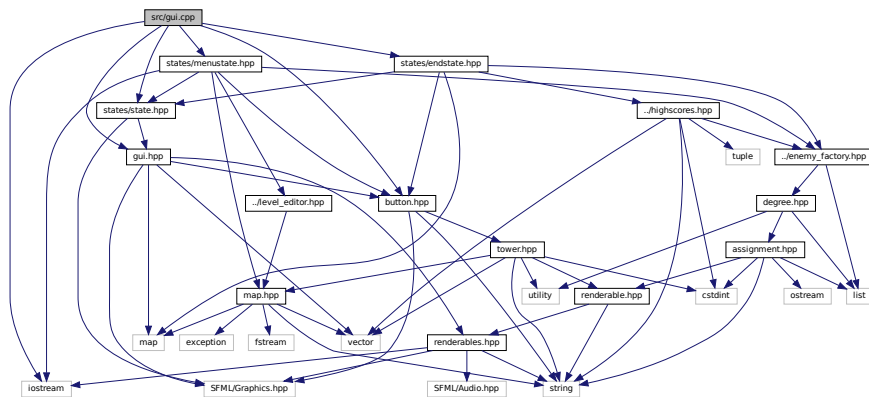
Enumerator

BuyFreshman	
BuyTeekkari	
BuyBachelor	
BuyMaster	
BuyDoctor	
BuyCalculator	
BuyCoffeeTable	
UpgradeTower	
DestroyTower	

14.27 src/gui.cpp File Reference

```
#include "gui.hpp"
#include <iostream>
#include "button.hpp"
#include "states/state.hpp"
#include "states/menustate.hpp"
#include "states/endstate.hpp"
```

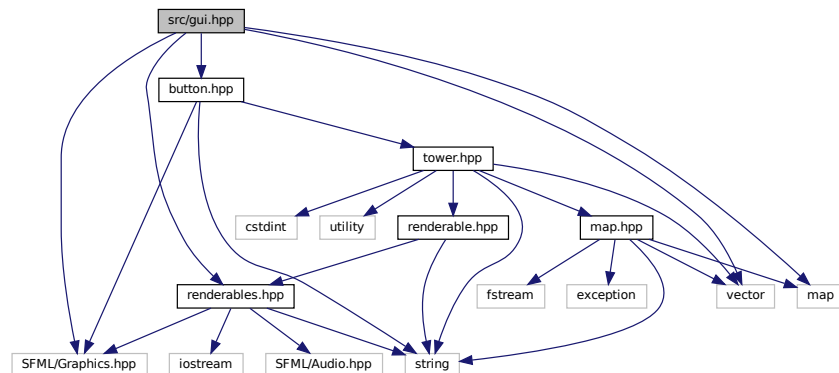
Include dependency graph for gui.cpp:



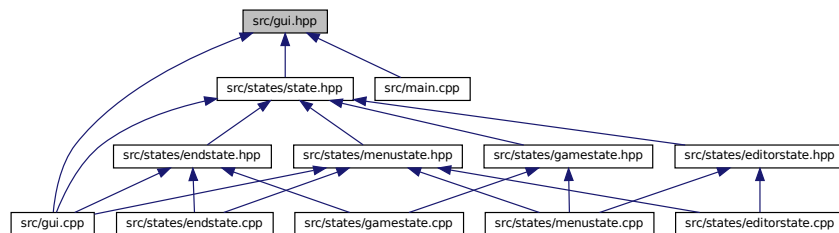
14.28 src/gui.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <map>
#include <vector>
#include "button.hpp"
```

```
#include "renderables.hpp"
Include dependency graph for gui.hpp:
```



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



Classes

- class [GUI](#)

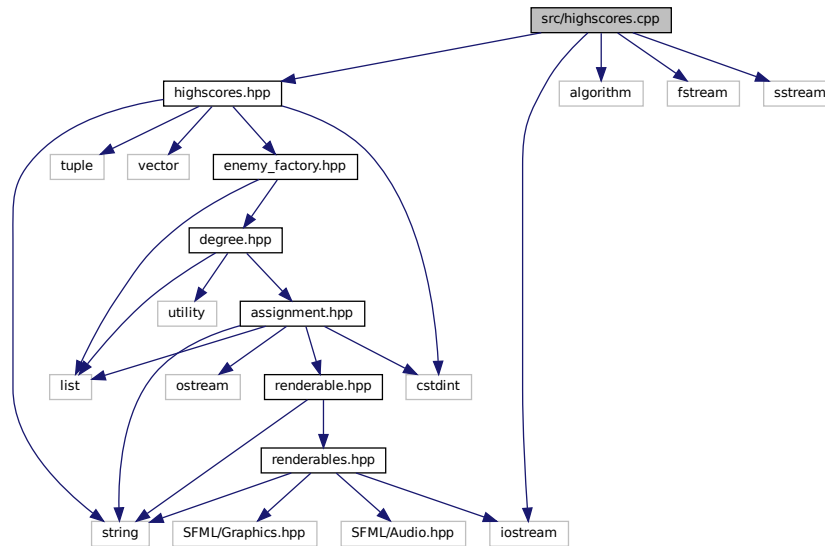
A class to add elements to the Graphical User Interface.

14.29 src/highscores.cpp File Reference

```
#include "highscores.hpp"
#include <algorithm>
#include <fstream>
#include <iostream>
```

```
#include <sstream>
```

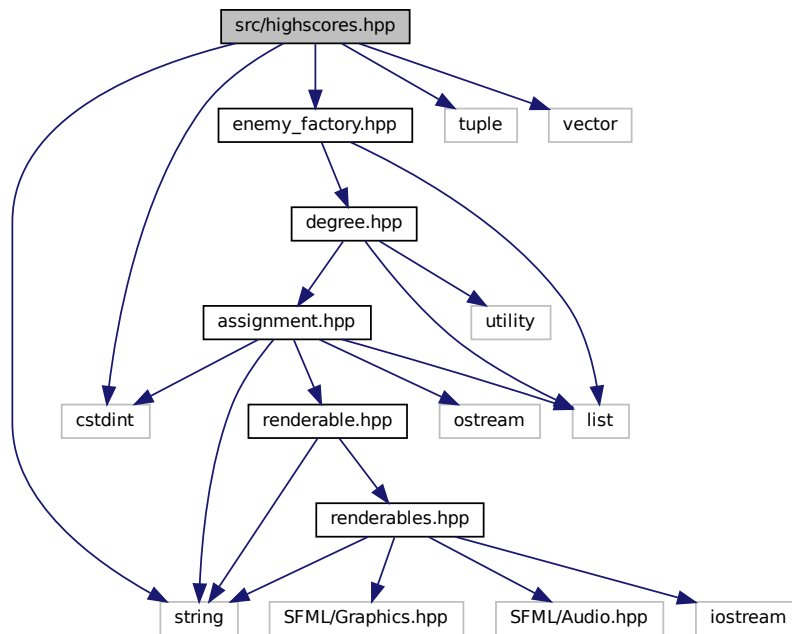
Include dependency graph for highscores.cpp:



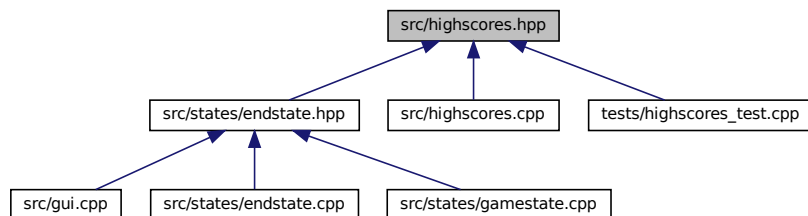
14.30 src/highscores.hpp File Reference

```
#include <cstdint>
#include <string>
#include <tuple>
#include <vector>
#include "enemy_factory.hpp"
```

Include dependency graph for `highscores.hpp`:



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



Classes

- class [Highscores](#)

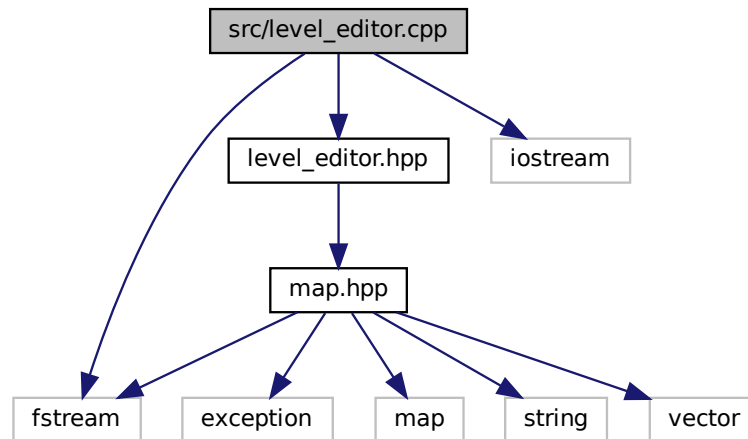
A class used to handle the high score savings at the end of game.

14.31 src/level_editor.cpp File Reference

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

```
#include <iostream>
```

Include dependency graph for level_editor.cpp:



Macros

- `#define PRINT_EDITOR_ERRORS false`

14.31.1 Macro Definition Documentation

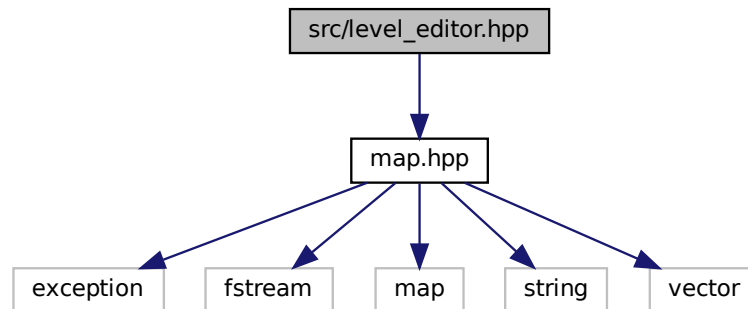
14.31.1.1 PRINT_EDITOR_ERRORS

```
#define PRINT_EDITOR_ERRORS false
```

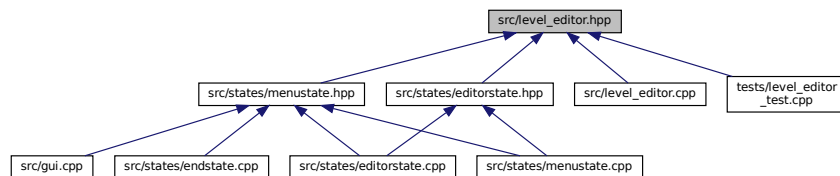
14.32 src/level_editor.hpp File Reference

```
#include "map.hpp"
```

Include dependency graph for level_editor.hpp:



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



Classes

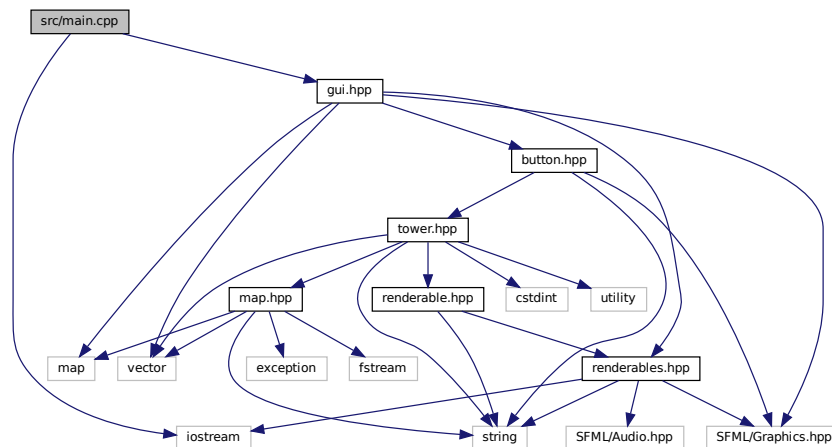
- class [LevelEditor](#)

The logical core of the level editor state.

14.33 src/main.cpp File Reference

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

Include dependency graph for main.cpp:



Functions

- int [main](#) ()

14.33.1 Function Documentation

14.33.1.1 main()

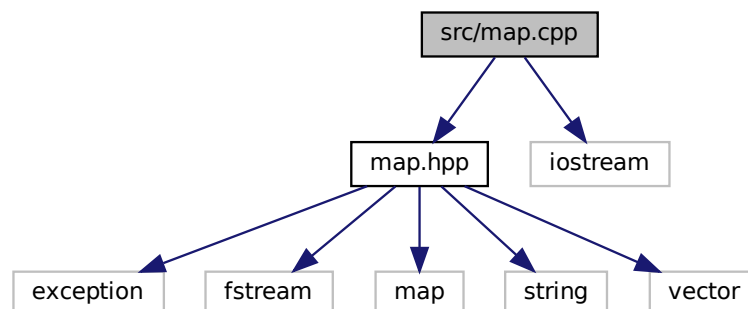
```
int main ( )
```

14.34 src/map.cpp File Reference

```
#include "map.hpp"
```

```
#include <iostream>
```

Include dependency graph for map.cpp:



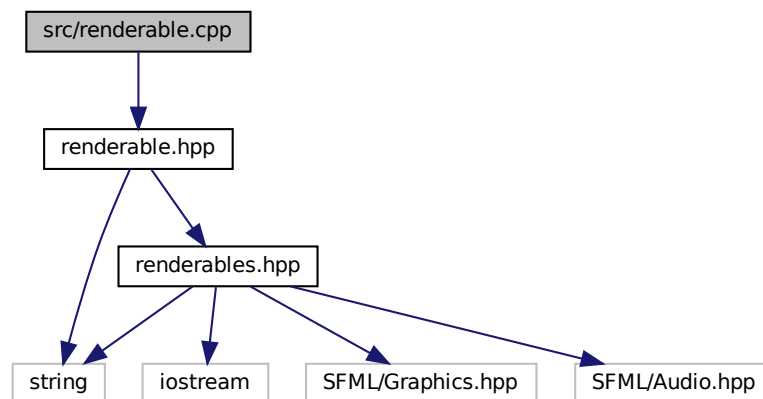
Enumerator

towerTile	
startTile	
pathTile	
endTile	

14.36 src/maps/map1.txt File Reference**14.37 src/maps/map2.txt File Reference****14.38 src/maps/map3.txt File Reference****14.39 src/renderable.cpp File Reference**

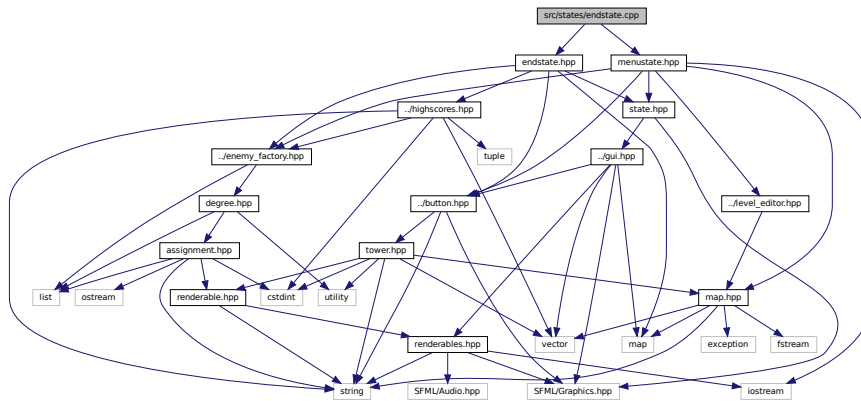
```
#include "renderable.hpp"
```

Include dependency graph for renderable.cpp:

**14.40 src/renderable.hpp File Reference**

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

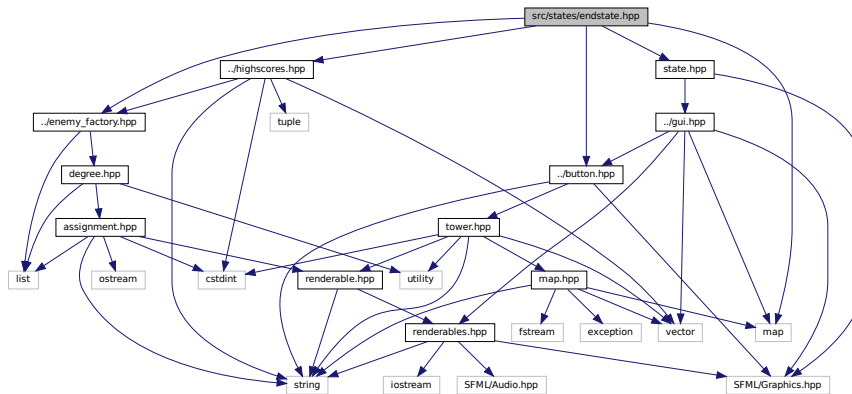

Include dependency graph for endstate.cpp:



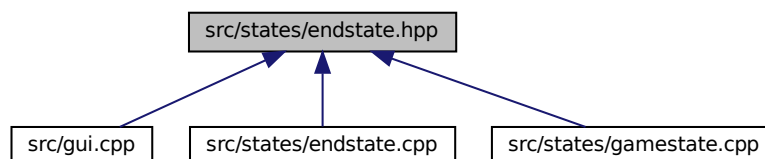
14.46 src/states/endstate.hpp File Reference

```
#include <map>
#include "../button.hpp"
#include "../enemy_factory.hpp"
#include "../highscores.hpp"
#include "state.hpp"
```

Include dependency graph for endstate.hpp:



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

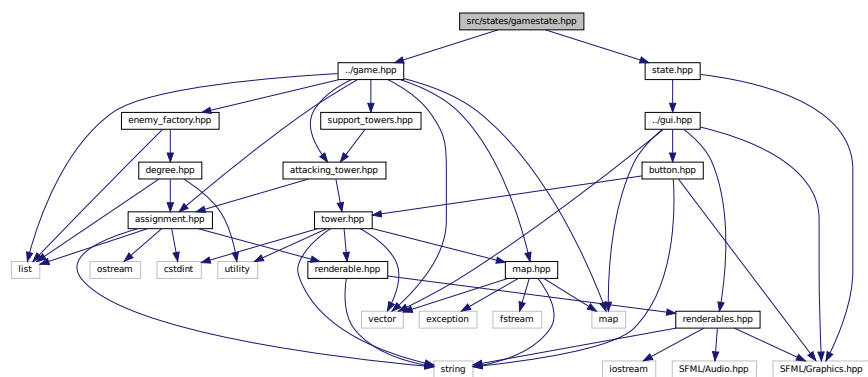


14.47.1.3 TILE_SIZE

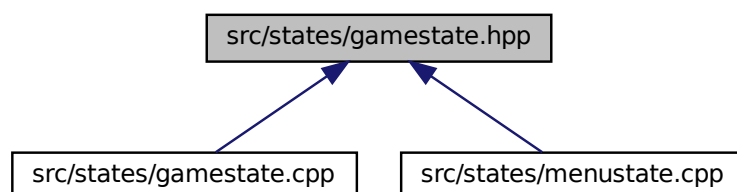
```
#define TILE_SIZE 30
```

14.48 src/states/gamestate.hpp File Reference

```
#include "../game.hpp"
#include "state.hpp"
Include dependency graph for gamestate.hpp:
```



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



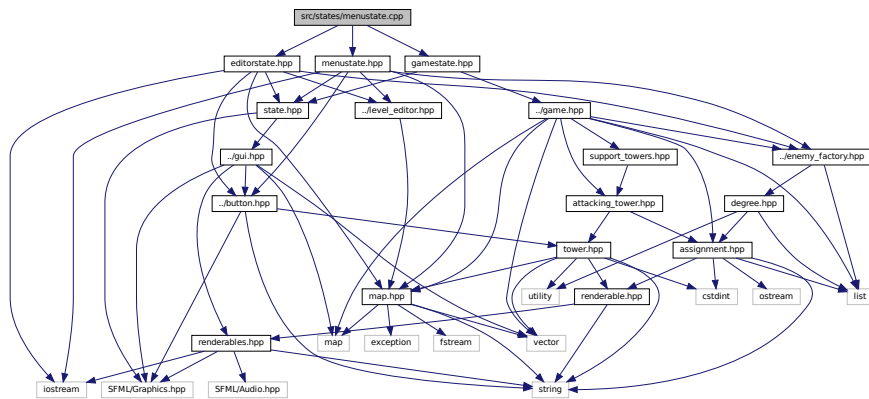
Classes

- class [GameState](#)

[GameState](#) class runs and draws the game part of the software.

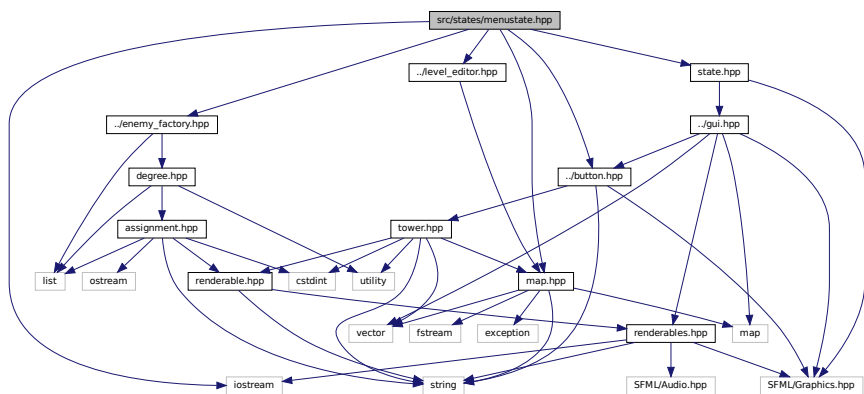
14.49 src/states/menustate.cpp File Reference

```
#include "menustate.hpp"
#include "editorstate.hpp"
#include "gamestate.hpp"
Include dependency graph for menustate.cpp:
```

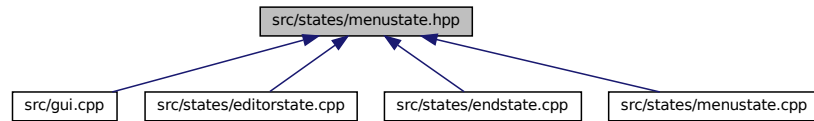


14.50 src/states/menustate.hpp File Reference

```
#include <iostream>
#include "../button.hpp"
#include "../enemy_factory.hpp"
#include "../level_editor.hpp"
#include "../map.hpp"
#include "state.hpp"
Include dependency graph for menustate.hpp:
```



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



Classes

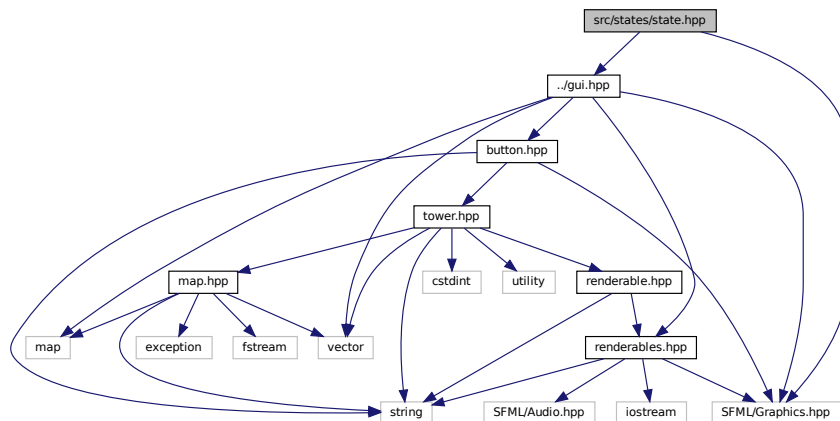
- class [MenuState](#)

A state run by [GUI](#) corresponding to the main menu state.

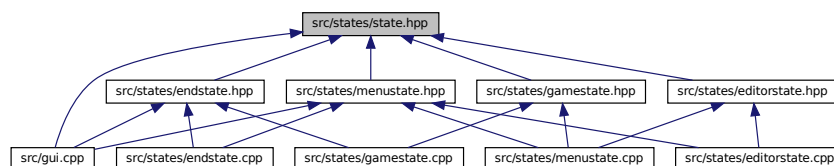
14.51 src/states/state.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "../gui.hpp"
```

Include dependency graph for state.hpp:



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



Classes

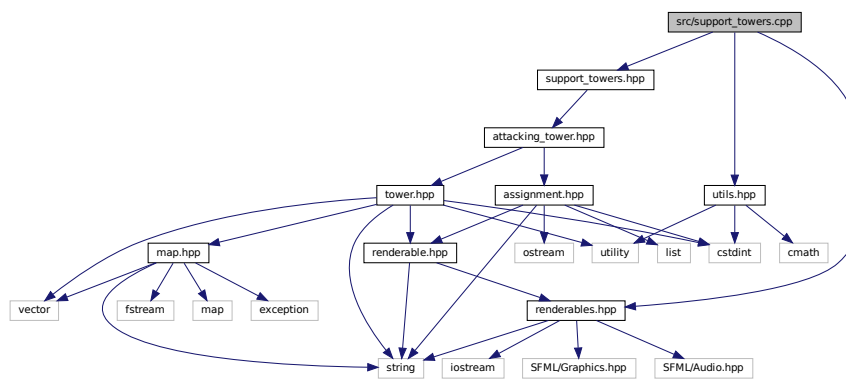
- class [State](#)

Abstract [State](#) class represents the classes that run and draw the different software states.

14.52 src/support_towers.cpp File Reference

```
#include "support_towers.hpp"
#include "utils.hpp"
#include "renderables.hpp"
```

Include dependency graph for support_towers.cpp:



Functions

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

14.52.1 Function Documentation

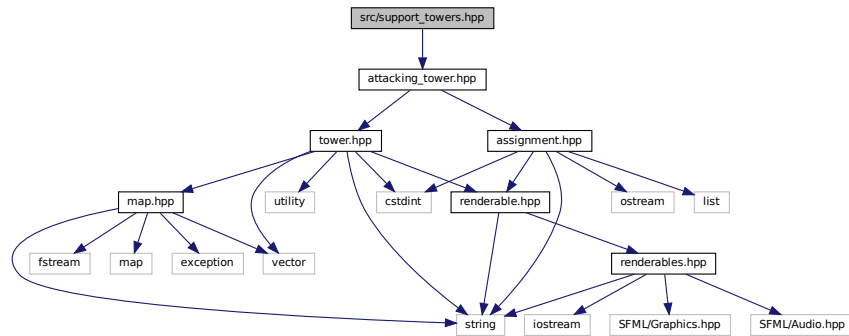
14.52.1.1 `operator<<()`

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

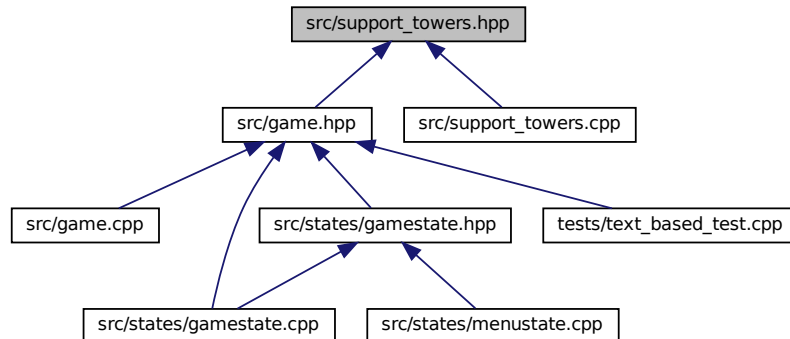
14.53 src/support_towers.hpp File Reference

```
#include "attacking_tower.hpp"
```

Include dependency graph for support_towers.hpp:



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



Classes

- class [SupportTower](#)

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

- class [BuffTower](#)

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

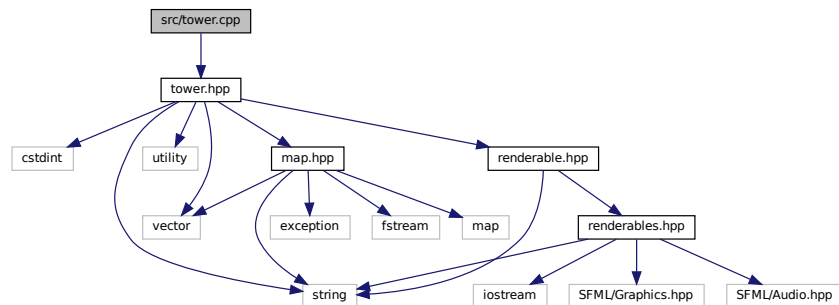
- class [HealTower](#)

A tower which heals attacking towers.

14.54 src/tower.cpp File Reference

```
#include "tower.hpp"
```

Include dependency graph for tower.cpp:



14.55 src/tower.hpp File Reference

```
#include <cstdint>
```

```
#include <string>
```

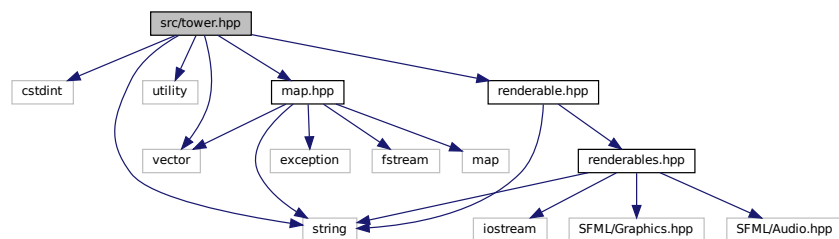
```
#include <utility>
```

```
#include <vector>
```

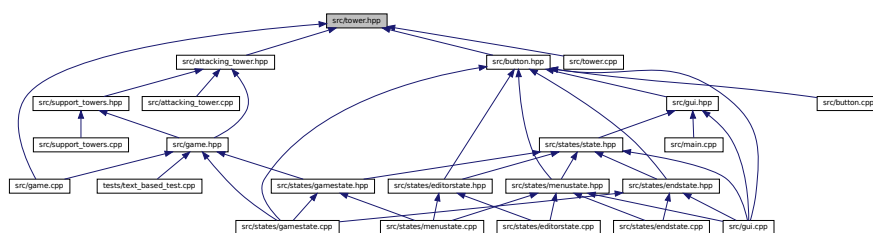
```
#include "map.hpp"
```

```
#include "renderable.hpp"
```

Include dependency graph for tower.hpp:



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



Classes

- class [Tower](#)

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

Macros

- `#define TILE_SIZE 30`

Enumerations

- enum [TowerType](#) {
 [Freshman](#), [Teekkari](#), [Bachelor](#), [Master](#),
 [Doctor](#), [Calculator](#), [CoffeeTable](#) }

An enumeration for the different towertypes.

14.55.1 Macro Definition Documentation

14.55.1.1 [TILE_SIZE](#)

```
#define TILE\_SIZE 30
```

14.55.2 Enumeration Type Documentation

14.55.2.1 [TowerType](#)

```
enum TowerType
```

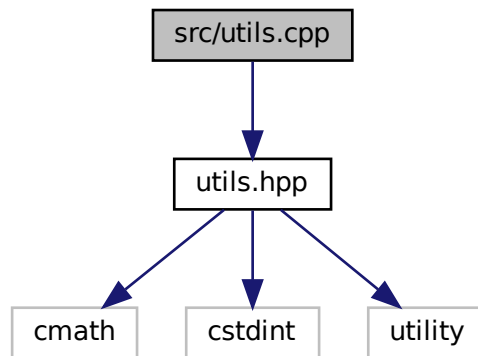
An enumeration for the different towertypes.

Enumerator

Freshman	
Teekkari	
Bachelor	
Master	
Doctor	
Calculator	
CoffeeTable	

14.56 src/utls.cpp File Reference

```
#include "utls.hpp"
Include dependency graph for utls.cpp:
```



Namespaces

- [UtilFunctions](#)

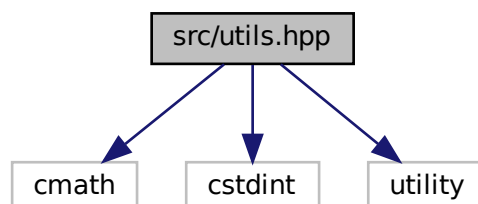
A namespace containing some utility functions needed in the project.

Functions

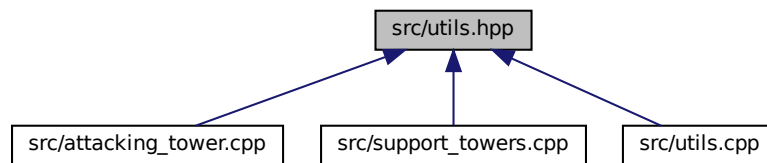
- float [UtilFunctions::distance](#) (const std::pair< int32_t, int32_t > &c1, const std::pair< int32_t, int32_t > &c2)
Calculates the Euclidean distance between two coordinate pairs.

14.57 src/utls.hpp File Reference

```
#include <cmath>
#include <cstdint>
#include <utility>
Include dependency graph for utls.hpp:
```



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



Namespaces

- [UtilFunctions](#)

A namespace containing some utility functions needed in the project.

Functions

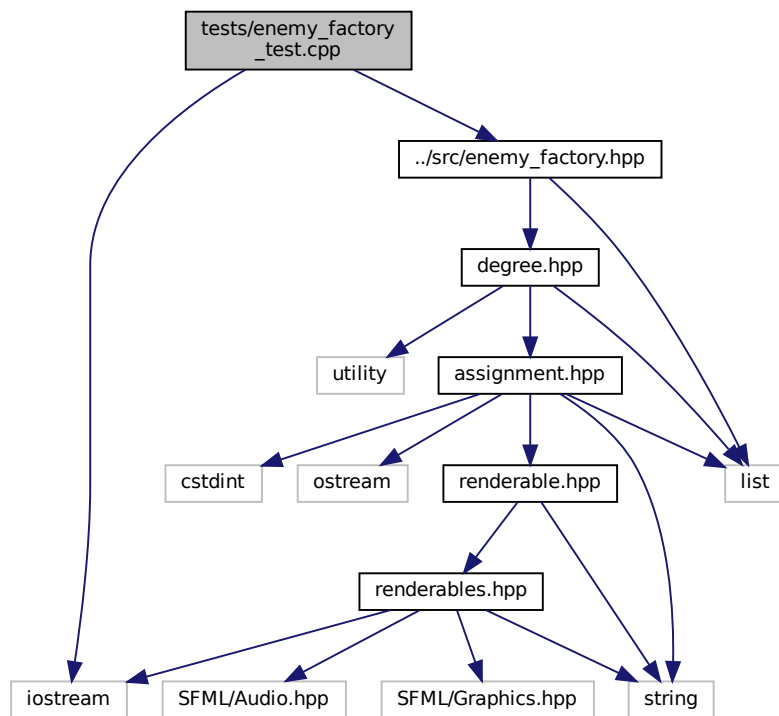
- float [UtilFunctions::distance](#) (const std::pair< int32_t, int32_t > &c1, const std::pair< int32_t, int32_t > &c2)

Calculates the Euclidean distance between two coordinate pairs.

14.58 tests/enemy_factory_test.cpp File Reference

```
#include <iostream>
#include "../src/enemy_factory.hpp"
```


Include dependency graph for enemy_factory_test.cpp:



Functions

- int `main` ()

14.58.1 Function Documentation

14.58.1.1 `main()`

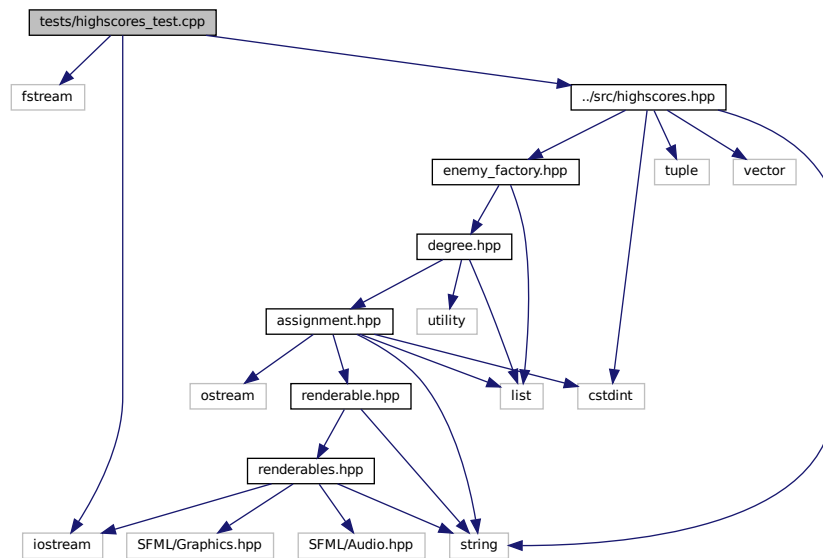
```
int main ( )
```

14.59 tests/highscores_test.cpp File Reference

```
#include <fstream>
#include <iostream>
```

```
#include "../src/highscores.hpp"
```

Include dependency graph for `highscores_test.cpp`:



Functions

- `int main ()`

14.59.1 Function Documentation

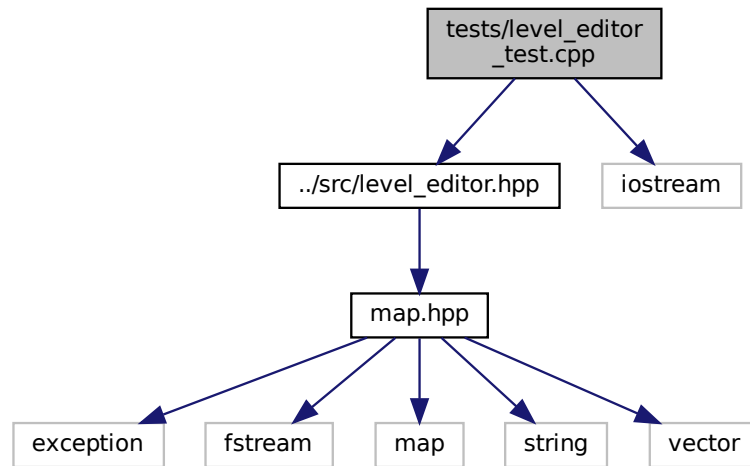
14.59.1.1 `main()`

```
int main ( )
```

14.60 tests/level_editor_test.cpp File Reference

Used to test the [LevelEditor](#) class.

```
#include "../src/level_editor.hpp"
#include <iostream>
Include dependency graph for level_editor_test.cpp:
```



Functions

- int `main` (void)

14.60.1 Detailed Description

Used to test the `LevelEditor` class.

14.60.2 Function Documentation

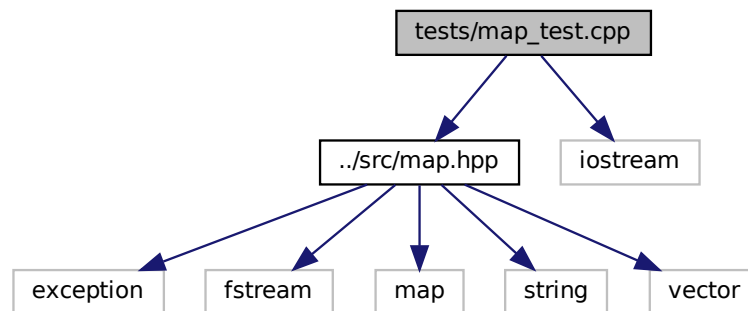
14.60.2.1 `main()`

```
int main (
    void )
```

14.61 tests/map_test.cpp File Reference

Used to test the [Map](#) class.

```
#include "../src/map.hpp"  
#include <iostream>  
Include dependency graph for map_test.cpp:
```



Functions

- `int main ()`

14.61.1 Detailed Description

Used to test the [Map](#) class.

14.61.2 Function Documentation

14.61.2.1 `main()`

```
int main ( )
```

14.62 tests/testmap1.txt File Reference

14.63 tests/testmap2.txt File Reference

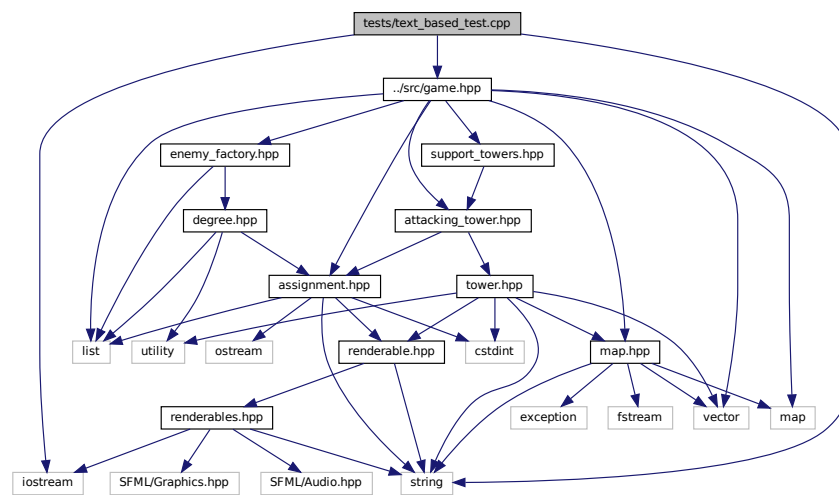
14.64 tests/testmapWrong.txt File Reference

14.65 tests/text_based_test.cpp File Reference

Used to test the game running with text output to terminal.

```
#include <iostream>
#include <string>
#include "../src/game.hpp"
```

Include dependency graph for text_based_test.cpp:



Functions

- int [main](#) (void)

14.65.1 Detailed Description

Used to test the game running with text output to terminal.

14.65.2 Function Documentation

14.65.2.1 main()

```
int main (
    void )
```


Index

- ~Assignment
 - Assignment, [33](#)
- ~AttackingTower
 - AttackingTower, [39](#)
- ~BuffTower
 - BuffTower, [47](#)
- ~Button
 - Button, [49](#)
- ~Degree
 - Degree, [53](#)
- ~EditorState
 - EditorState, [57](#)
- ~EndState
 - EndState, [61](#)
- ~EnemyFactory
 - EnemyFactory, [65](#)
- ~GUI
 - GUI, [86](#)
- ~Game
 - Game, [70](#)
- ~GameState
 - GameState, [80](#)
- ~HealTower
 - HealTower, [92](#)
- ~Highscores
 - Highscores, [94](#)
- ~LevelEditor
 - LevelEditor, [98](#)
- ~Map
 - Map, [102](#)
- ~MenuState
 - MenuState, [108](#)
- ~Renderable
 - Renderable, [112](#)
- ~State
 - State, [131](#)
- ~SupportTower
 - SupportTower, [135](#)
- ~Tower
 - Tower, [138](#)
- ~TowerButton
 - TowerButton, [144](#)
- Act
 - BuffTower, [47](#)
 - HealTower, [93](#)
 - SupportTower, [135](#)
- Action
 - game.hpp, [164](#)
- addHighlight
 - Button, [49](#)
- AddScore
 - Highscores, [95](#)
- AddTower
 - Game, [71](#)
- Advance
 - Assignment, [33](#)
- ANIMATION_LENGTH
 - gamestate.cpp, [179](#)
- ApplyBuff
 - AttackingTower, [39](#)
- ARCHITECTURE_ID
 - CMakeCCompilerId.c, [147](#)
 - CMakeCXXCompilerId.cpp, [150](#)
- Assignment, [31](#)
 - ~Assignment, [33](#)
 - Advance, [33](#)
 - Assignment, [33](#)
 - CrLeft, [33](#)
 - GetCredits, [34](#)
 - IsAlive, [34](#)
 - m_curCr, [35](#)
 - m_maxCr, [35](#)
 - m_movedLastTick, [35](#)
 - m_timeRemainder, [35](#)
 - m_timeToMove, [36](#)
 - MovedLastTick, [34](#)
 - operator<<, [35](#)
 - TakeDmg, [34](#)
- assignment.cpp
 - operator<<, [154](#)
- assignment.hpp
 - B_Thesis, [155](#)
 - BSc, [155](#)
 - D_Thesis, [155](#)
 - DSc, [155](#)
 - Enemy, [155](#)
 - Essay, [155](#)
 - Homework, [155](#)
 - M_Thesis, [155](#)
 - MSc, [155](#)
 - Project, [155](#)
- Attack
 - AttackingTower, [40](#)
- attack
 - Renderables, [121](#)
- attack_sound
 - Renderables, [121](#)
- attacking_tower.cpp

- operator<<, 156
- AttackingTower, 36
 - ~AttackingTower, 39
 - ApplyBuff, 39
 - Attack, 40
 - AttackingTower, 38
 - Bachelor, 40
 - Doctor, 40
 - Freshman, 41
 - GetUpgradeCost, 41
 - Heal, 41
 - IsFunctional, 42
 - IsUpgradeable, 42
 - m_basePower, 44
 - m_buffs, 44
 - m_health, 44
 - m_inRangeInd, 44
 - m_level, 44
 - m_map, 44
 - m_maxHealth, 45
 - m_upgCost, 45
 - Master, 42
 - operator<<, 44
 - Priv_UpdateRange, 43
 - Teekkari, 43
 - Upgrade, 43
- B_Thesis
 - assignment.hpp, 155
- Bachelor
 - AttackingTower, 40
 - tower.hpp, 186
- bachelor_1
 - Renderables, 121
- bachelor_1_sprite
 - Renderables, 121
- bachelor_2
 - Renderables, 121
- bachelor_2_sprite
 - Renderables, 121
- bachelor_3
 - Renderables, 121
- bachelor_3_sprite
 - Renderables, 122
- bachelors_thesis
 - Renderables, 122
- bachelors_thesis_sprite
 - Renderables, 122
- background
 - Renderables, 122
- background_sprite
 - Renderables, 122
- BSc
 - assignment.hpp, 155
- bsc
 - Renderables, 122
- bsc_sprite
 - Renderables, 122
- BuffTower, 45
 - ~BuffTower, 47
 - Act, 47
 - BuffTower, 46
 - m_buffStrength, 47
- build/CMakeCache.txt, 147
- build/CMakeFiles/3.16.3/CompilerIdC/CMakeCCompilerId.c, 147
- build/CMakeFiles/3.16.3/CompilerIdCXX/CMakeCXXCompilerId.cpp, 150
- build/CMakeFiles/TargetDirectories.txt, 153
- build/CMakeFiles/tower-defense.dir/link.txt, 153
- BuildPath
 - Map, 102
- Button, 48
 - ~Button, 49
 - addHighlight, 49
 - Button, 49
 - changeText, 50
 - disableButton, 50
 - drawButton, 50
 - enableButton, 50
 - getGlobalBounds, 51
 - m_button, 51
 - m_font, 51
 - m_text, 51
 - removeHighlight, 51
- button.cpp
 - TILE_SIZE, 158
- BuyBachelor
 - game.hpp, 165
- BuyCalculator
 - game.hpp, 165
- BuyCoffeeTable
 - game.hpp, 165
- BuyDoctor
 - game.hpp, 165
- BuyFreshman
 - game.hpp, 165
- BuyMaster
 - game.hpp, 165
- BuyTeekkari
 - game.hpp, 165
- C_DIALECT
 - CMakeCCompilerId.c, 148
- Calculator
 - SupportTower, 135
 - tower.hpp, 186
- calculator
 - Renderables, 122
- calculator_sprite
 - Renderables, 123
- ChangeState
 - GUI, 87
- changeText
 - Button, 50
- CMakeCCompilerId.c
 - ARCHITECTURE_ID, 147
 - C_DIALECT, 148

- COMPILER_ID, 148
- DEC, 148
- HEX, 148
- info_arch, 149
- info_compiler, 149
- info_language_dialect_default, 149
- info_platform, 150
- main, 149
- PLATFORM_ID, 148
- STRINGIFY, 149
- STRINGIFY_HELPER, 149
- CMakeCXXCompilerId.cpp
 - ARCHITECTURE_ID, 150
 - COMPILER_ID, 150
 - CXX_STD, 151
 - DEC, 151
 - HEX, 151
 - info_arch, 152
 - info_compiler, 152
 - info_language_dialect_default, 152
 - info_platform, 152
 - main, 152
 - PLATFORM_ID, 151
 - STRINGIFY, 151
 - STRINGIFY_HELPER, 152
- CMakeLists.txt, 153
- coffee_table
 - Renderables, 123
- coffee_table_sprite
 - Renderables, 123
- CoffeeTable
 - SupportTower, 136
 - tower.hpp, 186
- COMPILER_ID
 - CMakeCCompilerId.c, 148
 - CMakeCXXCompilerId.cpp, 150
- CreateButton
 - GUI, 87
- CreateEnemy
 - EnemyFactory, 65
- CreateTower
 - Game, 71
- CreateTowerButton
 - GUI, 87
- CrLeft
 - Assignment, 33
- CXX_STD
 - CMakeCXXCompilerId.cpp, 151
- D_Thesis
 - assignment.hpp, 155
- DEC
 - CMakeCCompilerId.c, 148
 - CMakeCXXCompilerId.cpp, 151
- Degree, 52
 - ~Degree, 53
 - Degree, 53
 - m_decendants, 54
 - m_ef, 54
 - TakeDmg, 54
- DestroyTower
 - Game, 72
 - game.hpp, 165
- DETAILED_DEBUG_PRINT
 - enemy_factory.cpp, 161
- Difficulty
 - enemy_factory.hpp, 162
- disableButton
 - Button, 50
 - TowerButton, 144
- distance
 - UtilFunctions, 29
- doc/readme.md, 153
- Doctor
 - AttackingTower, 40
 - tower.hpp, 186
- doctor_1
 - Renderables, 123
- doctor_1_sprite
 - Renderables, 123
- doctor_2
 - Renderables, 123
- doctor_2_sprite
 - Renderables, 123
- doctor_3
 - Renderables, 123
- doctor_3_sprite
 - Renderables, 124
- doctoral_thesis
 - Renderables, 124
- doctoral_thesis_sprite
 - Renderables, 124
- Draw
 - MenuState, 108
- drawButton
 - Button, 50
 - TowerButton, 144
- DSc
 - assignment.hpp, 155
- dsc
 - Renderables, 124
- dsc_sprite
 - Renderables, 124
- Easy
 - enemy_factory.hpp, 162
- Edit
 - LevelEditor, 98
 - Map, 102
- EditorState, 55
 - ~EditorState, 57
 - EditorState, 56
 - m_buttons, 57
 - m_drawSelectedShape, 57
 - m_editor, 58
 - m_instructions, 58
 - m_mapPath, 58
 - m_mapTileSprites, 58

- m_selectedButton, 58
 - m_selectedShape, 58
 - m_selX, 58
 - m_selY, 58
 - m_unvalidated, 59
 - m_validated, 59
 - Priv_Draw, 57
 - Priv_PollEvents, 57
 - Run, 57
- editorstate.cpp
 - TILE_SIZE, 176
- enableButton
 - Button, 50
 - TowerButton, 144
- EndState, 59
 - ~EndState, 61
 - EndState, 60
 - m_buttons, 62
 - m_difficulty, 62
 - m_font, 62
 - m_highscores, 62
 - m_input, 62
 - m_score, 62
 - m_text_highscores, 62
 - m_text_name, 62
 - m_text_score, 63
 - Priv_Draw, 61
 - Priv_PollEvents, 61
 - Run, 61
- endTile
 - map.hpp, 173
 - Renderables, 124
- endTile_sprite
 - Renderables, 124
- enemies
 - GUI, 89
- EnemiesLeft
 - EnemyFactory, 65
- Enemy
 - assignment.hpp, 155
- enemy_factory.cpp
 - DETAILED_DEBUG_PRINT, 161
 - operator<<, 161
- enemy_factory.hpp
 - Difficulty, 162
 - Easy, 162
 - Hard, 162
 - Medium, 162
- enemy_factory_test.cpp
 - main, 189
- EnemyFactory, 63
 - ~EnemyFactory, 65
 - CreateEnemy, 65
 - EnemiesLeft, 65
 - EnemyFactory, 64, 65
 - GetDifficulty, 65
 - m_batchSizeDeltas, 67
 - m_batchSizes, 67
 - m_diff, 67
 - m_nums, 68
 - m_round, 68
 - m_roundEnemies, 68
 - NextRoundInit, 66
 - NextTick, 66
 - operator<<, 67
 - operator=, 66
 - Priv_Free, 66
 - Priv_NextNum, 67
- EnemyTurn
 - Game, 72
- EntityName
 - Renderable, 112
- Essay
 - assignment.hpp, 155
- essay
 - Renderables, 124
- essay_sprite
 - Renderables, 125
- Freshman
 - AttackingTower, 41
 - tower.hpp, 186
- freshman_1
 - Renderables, 125
- freshman_1_sprite
 - Renderables, 125
- freshman_2
 - Renderables, 125
- freshman_2_sprite
 - Renderables, 125
- freshman_3
 - Renderables, 125
- freshman_3_sprite
 - Renderables, 125
- Game, 68
 - ~Game, 70
 - AddTower, 71
 - CreateTower, 71
 - DestroyTower, 72
 - EnemyTurn, 72
 - Game, 70, 71
 - GetAttackingTowers, 72
 - GetAttacks, 72
 - GetDifficulty, 72
 - GetEnemies, 73
 - GetHealth, 73
 - GetMap, 73
 - GetMoney, 73
 - GetScore, 74
 - GetSupportTowers, 74
 - GetTower, 74
 - IsActionPossible, 75
 - m_attackingTowers, 77
 - m_enemies, 77
 - m_enemyFactory, 77
 - m_map, 77

- m_money, [77](#)
- m_playerHealth, [77](#)
- m_score, [77](#)
- m_supportingTowers, [77](#)
- m_tickAttacks, [78](#)
- operator<<, [76](#)
- operator=, [75](#)
- RoundsFinished, [75](#)
- StartNextRound, [75](#)
- TowerTurn, [76](#)
- UpgradeTower, [76](#)
- game.cpp
 - operator<<, [163](#)
- game.hpp
 - Action, [164](#)
 - BuyBachelor, [165](#)
 - BuyCalculator, [165](#)
 - BuyCoffeeTable, [165](#)
 - BuyDoctor, [165](#)
 - BuyFreshman, [165](#)
 - BuyMaster, [165](#)
 - BuyTeekkari, [165](#)
 - DestroyTower, [165](#)
 - UpgradeTower, [165](#)
- GameState, [78](#)
 - ~GameState, [80](#)
 - GameState, [80](#)
 - m_buildPhase, [82](#)
 - m_buttons, [82](#)
 - m_drawRange, [82](#)
 - m_drawUpgradeRange, [82](#)
 - m_frameInTick, [83](#)
 - m_gameLogic, [83](#)
 - m_gameOver, [83](#)
 - m_gameSpeed, [83](#)
 - m_healthText, [83](#)
 - m_mapTileSprites, [83](#)
 - m_moneyText, [83](#)
 - m_projectile, [83](#)
 - m_rangeCircle, [84](#)
 - m_roundNum, [84](#)
 - m_roundNumText, [84](#)
 - m_scoreText, [84](#)
 - m_selectedShape, [84](#)
 - m_selX, [84](#)
 - m_selY, [84](#)
 - m_upgradeRange, [84](#)
 - Priv_ChangeCircle, [80](#)
 - Priv_ClearSpeedHighlights, [81](#)
 - Priv_Draw, [81](#)
 - Priv_DrawBCG, [81](#)
 - Priv_DrawMap, [81](#)
 - Priv_InitializeText, [81](#)
 - Priv_PollEvents, [82](#)
 - Run, [82](#)
- gamestate.cpp
 - ANIMATION_LENGTH, [179](#)
 - PROJECTILE_RADIUS, [179](#)
 - TILE_SIZE, [179](#)
- GetAttackingTowers
 - Game, [72](#)
- GetAttacks
 - Game, [72](#)
- getAttackSound
 - Renderables, [116](#)
- getBachelor1Sprite
 - Renderables, [116](#)
- getBachelor2Sprite
 - Renderables, [117](#)
- getBachelor3Sprite
 - Renderables, [117](#)
- getBachelorsThesisSprite
 - Renderables, [117](#)
- getBackgroundSprite
 - Renderables, [117](#)
- getBscSprite
 - Renderables, [117](#)
- getCalculatorSprite
 - Renderables, [117](#)
- getCoffeeTableSprite
 - Renderables, [117](#)
- GetCoords
 - Tower, [140](#)
- GetCredits
 - Assignment, [34](#)
- GetDifficulty
 - EnemyFactory, [65](#)
 - Game, [72](#)
- getDoctor1Sprite
 - Renderables, [117](#)
- getDoctor2Sprite
 - Renderables, [118](#)
- getDoctor3Sprite
 - Renderables, [118](#)
- getDoctoralThesisSprite
 - Renderables, [118](#)
- getDscSprite
 - Renderables, [118](#)
- GetEnd
 - Map, [103](#)
- getEndtileSprite
 - Renderables, [118](#)
- GetEnemies
 - Game, [73](#)
- getEssaySprite
 - Renderables, [118](#)
- GetFont
 - GUI, [88](#)
- getFreshman1Sprite
 - Renderables, [118](#)
- getFreshman2Sprite
 - Renderables, [118](#)
- getFreshman3Sprite
 - Renderables, [119](#)
- getGlobalBounds
 - Button, [51](#)

- GetGrid
 - Map, [103](#)
- GetHealth
 - Game, [73](#)
- getHomeworkSprite
 - Renderables, [119](#)
- GetMap
 - Game, [73](#)
 - LevelEditor, [99](#)
- getMaster1Sprite
 - Renderables, [119](#)
- getMaster2Sprite
 - Renderables, [119](#)
- getMaster3Sprite
 - Renderables, [119](#)
- getMastersThesisSprite
 - Renderables, [119](#)
- getMenuBackgroundSprite
 - Renderables, [119](#)
- GetMoney
 - Game, [73](#)
- getMscSprite
 - Renderables, [119](#)
- GetNeighbors
 - Map, [103](#)
- GetPath
 - Map, [103](#)
- getPathtileSprite
 - Renderables, [120](#)
- GetPos
 - Map, [104](#)
- getProjectSprite
 - Renderables, [120](#)
- GetRange
 - Tower, [140](#)
- GetScore
 - Game, [74](#)
- getSelectSound
 - Renderables, [120](#)
- GetSprite
 - Renderable, [112](#)
- GetStart
 - Map, [104](#)
- getStarttileSprite
 - Renderables, [120](#)
- GetSupportTowers
 - Game, [74](#)
- getTeekkari1Sprite
 - Renderables, [120](#)
- getTeekkari2Sprite
 - Renderables, [120](#)
- getTeekkari3Sprite
 - Renderables, [120](#)
- GetTop10
 - Highscores, [95](#)
- GetTop10asString
 - Highscores, [95](#)
- GetTower
 - Game, [74](#)
- getTowertileSprite
 - Renderables, [120](#)
- GetUpgradeCost
 - AttackingTower, [41](#)
- GUI, [85](#)
 - ~GUI, [86](#)
 - ChangeState, [87](#)
 - CreateButton, [87](#)
 - CreateTowerButton, [87](#)
 - enemies, [89](#)
 - GetFont, [88](#)
 - GUI, [86](#)
 - init, [88](#)
 - m_event, [89](#)
 - m_font, [89](#)
 - m_new_state, [89](#)
 - m_renderables, [89](#)
 - m_state, [90](#)
 - m_videoMode, [90](#)
 - m_window, [90](#)
 - Priv_DeleteState, [88](#)
 - running, [88](#)
 - start, [90](#)
 - update, [89](#)
 - x_velo, [90](#)
 - y_velo, [90](#)
- Hard
 - enemy_factory.hpp, [162](#)
- Heal
 - AttackingTower, [41](#)
- HealTower, [91](#)
 - ~HealTower, [92](#)
 - Act, [93](#)
 - HealTower, [92](#)
 - m_healStrength, [93](#)
- HEX
 - CMakeCCompilerId.c, [148](#)
 - CMakeCXXCompilerId.cpp, [151](#)
- Highscores, [93](#)
 - ~Highscores, [94](#)
 - AddScore, [95](#)
 - GetTop10, [95](#)
 - GetTop10asString, [95](#)
 - Highscores, [94](#)
 - m_filename, [96](#)
 - m_highscores, [96](#)
 - m_saved, [96](#)
 - Priv_LoadHighscores, [96](#)
 - Priv_SortHighscores, [96](#)
- highscores.txt, [153](#)
- highscores_test.cpp
 - main, [190](#)
- Homework
 - assignment.hpp, [155](#)
- homework
 - Renderables, [125](#)
- homework_sprite

- Renderables, [126](#)
- info_arch
 - CMakeCCompilerId.c, [149](#)
 - CMakeCXXCompilerId.cpp, [152](#)
- info_compiler
 - CMakeCCompilerId.c, [149](#)
 - CMakeCXXCompilerId.cpp, [152](#)
- info_language_dialect_default
 - CMakeCCompilerId.c, [149](#)
 - CMakeCXXCompilerId.cpp, [152](#)
- info_platform
 - CMakeCCompilerId.c, [150](#)
 - CMakeCXXCompilerId.cpp, [152](#)
- init
 - GUI, [88](#)
- InitializeMap
 - Map, [104](#)
- IsActionPossible
 - Game, [75](#)
- IsAlive
 - Assignment, [34](#)
- IsFunctional
 - AttackingTower, [42](#)
- IsUpgradeable
 - AttackingTower, [42](#)
 - SupportTower, [136](#)
 - Tower, [140](#)
- level_editor.cpp
 - PRINT_EDITOR_ERRORS, [169](#)
- level_editor_test.cpp
 - main, [191](#)
- LevelEditor, [97](#)
 - ~LevelEditor, [98](#)
 - Edit, [98](#)
 - GetMap, [99](#)
 - LevelEditor, [98](#)
 - m_height, [99](#)
 - m_map, [100](#)
 - m_mapPath, [100](#)
 - m_width, [100](#)
 - Save, [99](#)
 - Validate, [99](#)
- libs/readme.md, [153](#)
- m_allSprites
 - Tower, [141](#)
- m_attackingTowers
 - Game, [77](#)
- m_basePower
 - AttackingTower, [44](#)
- m_batchSizeDeltas
 - EnemyFactory, [67](#)
- m_batchSizes
 - EnemyFactory, [67](#)
- m_buffs
 - AttackingTower, [44](#)
- m_buffStrength
 - BuffTower, [47](#)
- m_buildPhase
 - GameState, [82](#)
- m_button
 - Button, [51](#)
- m_buttons
 - EditorState, [57](#)
 - EndState, [62](#)
 - GameState, [82](#)
 - MenuState, [109](#)
- m_coords
 - Tower, [141](#)
- m_curCr
 - Assignment, [35](#)
- m_decendants
 - Degree, [54](#)
- m_diff
 - EnemyFactory, [67](#)
- m_difficulty
 - EndState, [62](#)
 - MenuState, [109](#)
- m_drawRange
 - GameState, [82](#)
- m_drawSelectedShape
 - EditorState, [57](#)
- m_drawUpgradeRange
 - GameState, [82](#)
- m_editing
 - MenuState, [109](#)
- m_editor
 - EditorState, [58](#)
- m_ef
 - Degree, [54](#)
- m_end
 - Map, [105](#)
- m_enemies
 - Game, [77](#)
- m_enemyFactory
 - Game, [77](#)
- m_entityName
 - Renderable, [113](#)
- m_event
 - GUI, [89](#)
 - State, [132](#)
- m_filename
 - Highscores, [96](#)
- m_font
 - Button, [51](#)
 - EndState, [62](#)
 - GUI, [89](#)
- m_frameInTick
 - GameState, [83](#)
- m_gameLogic
 - GameState, [83](#)
- m_gameOver
 - GameState, [83](#)
- m_gameSpeed
 - GameState, [83](#)

- m_grid
 - Map, [106](#)
- m_gui
 - State, [132](#)
- m_healStrength
 - HealTower, [93](#)
- m_health
 - AttackingTower, [44](#)
- m_healthText
 - GameState, [83](#)
- m_height
 - LevelEditor, [99](#)
 - Map, [106](#)
 - MenuState, [110](#)
- m_highscores
 - EndState, [62](#)
 - Highscores, [96](#)
- m_input
 - EndState, [62](#)
- m_inRangeInd
 - AttackingTower, [44](#)
- m_instructions
 - EditorState, [58](#)
- m_level
 - AttackingTower, [44](#)
- m_map
 - AttackingTower, [44](#)
 - Game, [77](#)
 - LevelEditor, [100](#)
- m_mapPath
 - EditorState, [58](#)
 - LevelEditor, [100](#)
- m_mapTileSprites
 - EditorState, [58](#)
 - GameState, [83](#)
- m_maxCr
 - Assignment, [35](#)
- m_maxHealth
 - AttackingTower, [45](#)
- m_money
 - Game, [77](#)
- m_moneyText
 - GameState, [83](#)
- m_movedLastTick
 - Assignment, [35](#)
- m_name
 - TowerButton, [145](#)
- m_new_state
 - GUI, [89](#)
- m_nums
 - EnemyFactory, [68](#)
- m_path
 - Map, [106](#)
- m_playerHealth
 - Game, [77](#)
- m_projectile
 - GameState, [83](#)
- m_range
 - Tower, [141](#)
- m_rangeCircle
 - GameState, [84](#)
- m_renderables
 - GUI, [89](#)
- m_round
 - EnemyFactory, [68](#)
- m_roundEnemies
 - EnemyFactory, [68](#)
- m_roundNum
 - GameState, [84](#)
- m_roundNumText
 - GameState, [84](#)
- m_saved
 - Highscores, [96](#)
- m_score
 - EndState, [62](#)
 - Game, [77](#)
- m_scoreText
 - GameState, [84](#)
- m_selectedButton
 - EditorState, [58](#)
- m_selectedMap
 - MenuState, [110](#)
- m_selectedShape
 - EditorState, [58](#)
 - GameState, [84](#)
- m_selX
 - EditorState, [58](#)
 - GameState, [84](#)
- m_selY
 - EditorState, [58](#)
 - GameState, [84](#)
- m_sprite
 - Renderable, [113](#)
 - TowerButton, [145](#)
- m_start
 - Map, [106](#)
- m_state
 - GUI, [90](#)
- m_supportingTowers
 - Game, [77](#)
- m_text
 - Button, [51](#)
- m_text_highscores
 - EndState, [62](#)
- m_text_name
 - EndState, [62](#)
- m_text_score
 - EndState, [63](#)
- m_texts
 - MenuState, [110](#)
- M_Thesis
 - assignment.hpp, [155](#)
- m_tickAttacks
 - Game, [78](#)
- m_timeRemainder
 - Assignment, [35](#)

- m_timeToMove
 - Assignment, 36
- m_unvalidated
 - EditorState, 59
- m_upgCost
 - AttackingTower, 45
- m_upgradeRange
 - GameState, 84
- m_validated
 - EditorState, 59
- m_videoMode
 - GUI, 90
- m_width
 - LevelEditor, 100
 - Map, 106
 - MenuState, 110
- m_window
 - GUI, 90
 - State, 132
- main
 - CMakeCCompilerId.c, 149
 - CMakeCXXCompilerId.cpp, 152
 - enemy_factory_test.cpp, 189
 - highscores_test.cpp, 190
 - level_editor_test.cpp, 191
 - main.cpp, 171
 - map_test.cpp, 192
 - text_based_test.cpp, 193
- main.cpp
 - main, 171
- Map, 100
 - ~Map, 102
 - BuildPath, 102
 - Edit, 102
 - GetEnd, 103
 - GetGrid, 103
 - GetNeighbors, 103
 - GetPath, 103
 - GetPos, 104
 - GetStart, 104
 - InitializeMap, 104
 - m_end, 105
 - m_grid, 106
 - m_height, 106
 - m_path, 106
 - m_start, 106
 - m_width, 106
 - Map, 101
 - TestTilePos, 105
 - ValidateMap, 105
- map.hpp
 - endTile, 173
 - pathTile, 173
 - startTile, 173
 - TileType, 172
 - towerTile, 173
- map_test.cpp
 - main, 192
- Master
 - AttackingTower, 42
 - tower.hpp, 186
- master_1
 - Renderables, 126
- master_1_sprite
 - Renderables, 126
- master_2
 - Renderables, 126
- master_2_sprite
 - Renderables, 126
- master_3
 - Renderables, 126
- master_3_sprite
 - Renderables, 126
- masters_thesis
 - Renderables, 126
- masters_thesis_sprite
 - Renderables, 127
- Medium
 - enemy_factory.hpp, 162
- Meeting-notes.md, 153
- menu_background
 - Renderables, 127
- menu_background_sprite
 - Renderables, 127
- MenuState, 107
 - ~MenuState, 108
 - Draw, 108
 - m_buttons, 109
 - m_difficulty, 109
 - m_editing, 109
 - m_height, 110
 - m_selectedMap, 110
 - m_texts, 110
 - m_width, 110
 - MenuState, 108
 - PollEvents, 109
 - Run, 109
 - RunLevelEditor, 109
- MovedLastTick
 - Assignment, 34
- MSc
 - assignment.hpp, 155
- msc
 - Renderables, 127
- msc_sprite
 - Renderables, 127
- NextRoundInit
 - EnemyFactory, 66
- NextTick
 - EnemyFactory, 66
- operator<<
 - Assignment, 35
 - assignment.cpp, 154
 - attacking_tower.cpp, 156
 - AttackingTower, 44

- enemy_factory.cpp, [161](#)
 - EnemyFactory, [67](#)
 - Game, [76](#)
 - game.cpp, [163](#)
 - support_towers.cpp, [183](#)
 - SupportTower, [136](#)
- operator=
 - EnemyFactory, [66](#)
 - Game, [75](#)
- pathTile
 - map.hpp, [173](#)
 - Renderables, [127](#)
- pathTile_sprite
 - Renderables, [127](#)
- plan/readme.md, [153](#)
- PLATFORM_ID
 - CMakeCCompilerId.c, [148](#)
 - CMakeCXXCompilerId.cpp, [151](#)
- PollEvents
 - MenuState, [109](#)
- PRINT_EDITOR_ERRORS
 - level_editor.cpp, [169](#)
- Priv_ChangeCircle
 - GameState, [80](#)
- Priv_ClearSpeedHighlights
 - GameState, [81](#)
- Priv_DeleteState
 - GUI, [88](#)
- Priv_Draw
 - EditorState, [57](#)
 - EndState, [61](#)
 - GameState, [81](#)
- Priv_DrawBCG
 - GameState, [81](#)
- Priv_DrawMap
 - GameState, [81](#)
- Priv_Free
 - EnemyFactory, [66](#)
- Priv_InitializeText
 - GameState, [81](#)
- Priv_LoadHighscores
 - Highscores, [96](#)
- Priv_NextNum
 - EnemyFactory, [67](#)
- Priv_PollEvents
 - EditorState, [57](#)
 - EndState, [61](#)
 - GameState, [82](#)
- Priv_SortHighscores
 - Highscores, [96](#)
- Priv_UpdateRange
 - AttackingTower, [43](#)
- Project
 - assignment.hpp, [155](#)
- project
 - Renderables, [127](#)
- project_sprite
 - Renderables, [128](#)
- PROJECTILE_RADIUS
 - gamestate.cpp, [179](#)
- README.md, [153](#)
- removeHighlight
 - Button, [51](#)
- Renderable, [111](#)
 - ~Renderable, [112](#)
 - EntityName, [112](#)
 - GetSprite, [112](#)
 - m_entityName, [113](#)
 - m_sprite, [113](#)
 - Renderable, [112](#)
 - SetSprite, [113](#)
- Renderables, [114](#)
 - attack, [121](#)
 - attack_sound, [121](#)
 - bachelor_1, [121](#)
 - bachelor_1_sprite, [121](#)
 - bachelor_2, [121](#)
 - bachelor_2_sprite, [121](#)
 - bachelor_3, [121](#)
 - bachelor_3_sprite, [122](#)
 - bachelors_thesis, [122](#)
 - bachelors_thesis_sprite, [122](#)
 - background, [122](#)
 - background_sprite, [122](#)
 - bsc, [122](#)
 - bsc_sprite, [122](#)
 - calculator, [122](#)
 - calculator_sprite, [123](#)
 - coffee_table, [123](#)
 - coffee_table_sprite, [123](#)
 - doctor_1, [123](#)
 - doctor_1_sprite, [123](#)
 - doctor_2, [123](#)
 - doctor_2_sprite, [123](#)
 - doctor_3, [123](#)
 - doctor_3_sprite, [124](#)
 - doctoral_thesis, [124](#)
 - doctoral_thesis_sprite, [124](#)
 - dsc, [124](#)
 - dsc_sprite, [124](#)
 - endTile, [124](#)
 - endTile_sprite, [124](#)
 - essay, [124](#)
 - essay_sprite, [125](#)
 - freshman_1, [125](#)
 - freshman_1_sprite, [125](#)
 - freshman_2, [125](#)
 - freshman_2_sprite, [125](#)
 - freshman_3, [125](#)
 - freshman_3_sprite, [125](#)
 - getAttackSound, [116](#)
 - getBachelor1Sprite, [116](#)
 - getBachelor2Sprite, [117](#)
 - getBachelor3Sprite, [117](#)
 - getBachelorsThesisSprite, [117](#)
 - getBackgroundSprite, [117](#)

- getBscSprite, [117](#)
- getCalculatorSprite, [117](#)
- getCoffeeTableSprite, [117](#)
- getDoctor1Sprite, [117](#)
- getDoctor2Sprite, [118](#)
- getDoctor3Sprite, [118](#)
- getDoctoralThesisSprite, [118](#)
- getDscSprite, [118](#)
- getEndtileSprite, [118](#)
- getEssaySprite, [118](#)
- getFreshman1Sprite, [118](#)
- getFreshman2Sprite, [118](#)
- getFreshman3Sprite, [119](#)
- getHomeworkSprite, [119](#)
- getMaster1Sprite, [119](#)
- getMaster2Sprite, [119](#)
- getMaster3Sprite, [119](#)
- getMastersThesisSprite, [119](#)
- getMenuBackgroundSprite, [119](#)
- getMscSprite, [119](#)
- getPathtileSprite, [120](#)
- getProjectSprite, [120](#)
- getSelectSound, [120](#)
- getStarttileSprite, [120](#)
- getTeekkari1Sprite, [120](#)
- getTeekkari2Sprite, [120](#)
- getTeekkari3Sprite, [120](#)
- getTowertileSprite, [120](#)
- homework, [125](#)
- homework_sprite, [126](#)
- master_1, [126](#)
- master_1_sprite, [126](#)
- master_2, [126](#)
- master_2_sprite, [126](#)
- master_3, [126](#)
- master_3_sprite, [126](#)
- masters_thesis, [126](#)
- masters_thesis_sprite, [127](#)
- menu_background, [127](#)
- menu_background_sprite, [127](#)
- msc, [127](#)
- msc_sprite, [127](#)
- pathTile, [127](#)
- pathTile_sprite, [127](#)
- project, [127](#)
- project_sprite, [128](#)
- Renderables, [116](#)
- s_instance, [128](#)
- select, [128](#)
- select_sound, [128](#)
- startTile, [128](#)
- startTile_sprite, [128](#)
- teekkari_1, [128](#)
- teekkari_1_sprite, [129](#)
- teekkari_2, [129](#)
- teekkari_2_sprite, [129](#)
- teekkari_3, [129](#)
- teekkari_3_sprite, [129](#)
- towerTile, [129](#)
- towerTile_sprite, [129](#)
- RoundIsFinished
 - Game, [75](#)
- Run
 - EditorState, [57](#)
 - EndState, [61](#)
 - GameState, [82](#)
 - MenuState, [109](#)
 - State, [132](#)
- RunLevelEditor
 - MenuState, [109](#)
- running
 - GUI, [88](#)
- s_instance
 - Renderables, [128](#)
- Save
 - LevelEditor, [99](#)
- select
 - Renderables, [128](#)
- select_sound
 - Renderables, [128](#)
- SetSprite
 - Renderable, [113](#)
- src/assignment.cpp, [153](#)
- src/assignment.hpp, [154](#)
- src/attacking_tower.cpp, [155](#)
- src/attacking_tower.hpp, [156](#)
- src/button.cpp, [157](#)
- src/button.hpp, [158](#)
- src/degree.cpp, [159](#)
- src/degree.hpp, [159](#)
- src/enemy_factory.cpp, [160](#)
- src/enemy_factory.hpp, [161](#)
- src/game.cpp, [162](#)
- src/game.hpp, [163](#)
- src/gui.cpp, [165](#)
- src/gui.hpp, [165](#)
- src/highscores.cpp, [166](#)
- src/highscores.hpp, [167](#)
- src/level_editor.cpp, [168](#)
- src/level_editor.hpp, [170](#)
- src/main.cpp, [170](#)
- src/map.cpp, [171](#)
- src/map.hpp, [172](#)
- src/maps/map1.txt, [173](#)
- src/maps/map2.txt, [173](#)
- src/maps/map3.txt, [173](#)
- src/readme.md, [153](#)
- src/renderable.cpp, [173](#)
- src/renderable.hpp, [173](#)
- src/renderables.cpp, [174](#)
- src/renderables.hpp, [175](#)
- src/states/editorstate.cpp, [175](#)
- src/states/editorstate.hpp, [176](#)
- src/states/endstate.cpp, [177](#)
- src/states/endstate.hpp, [178](#)
- src/states/gamestate.cpp, [179](#)

- src/states/gamestate.hpp, 180
- src/states/menustate.cpp, 181
- src/states/menustate.hpp, 181
- src/states/state.hpp, 182
- src/support_towers.cpp, 183
- src/support_towers.hpp, 184
- src/tower.cpp, 185
- src/tower.hpp, 185
- src/utlis.cpp, 187
- src/utlis.hpp, 187
- start
 - GUI, 90
- StartNextRound
 - Game, 75
- startTile
 - map.hpp, 173
 - Renderables, 128
- startTile_sprite
 - Renderables, 128
- State, 130
 - ~State, 131
 - m_event, 132
 - m_gui, 132
 - m_window, 132
 - Run, 132
 - State, 131
- STRINGIFY
 - CMakeCCompilerId.c, 149
 - CMakeCXXCompilerId.cpp, 151
- STRINGIFY_HELPER
 - CMakeCCompilerId.c, 149
 - CMakeCXXCompilerId.cpp, 152
- support_towers.cpp
 - operator<<, 183
- SupportTower, 133
 - ~SupportTower, 135
 - Act, 135
 - Calculator, 135
 - CoffeeTable, 136
 - IsUpgradeable, 136
 - operator<<, 136
 - SupportTower, 134
- TakeDmg
 - Assignment, 34
 - Degree, 54
- Teekkari
 - AttackingTower, 43
 - tower.hpp, 186
- teekkari_1
 - Renderables, 128
- teekkari_1_sprite
 - Renderables, 129
- teekkari_2
 - Renderables, 129
- teekkari_2_sprite
 - Renderables, 129
- teekkari_3
 - Renderables, 129
- teekkari_3_sprite
 - Renderables, 129
- tests/enemy_factory_test.cpp, 188
- tests/highscores_test.cpp, 189
- tests/level_editor_test.cpp, 190
- tests/map_test.cpp, 192
- tests/readme.md, 153
- tests/testmap1.txt, 193
- tests/testmap2.txt, 193
- tests/testmapWrong.txt, 193
- tests/text_based_test.cpp, 193
- TestTilePos
 - Map, 105
- text_based_test.cpp
 - main, 193
- TILE_SIZE
 - button.cpp, 158
 - editorstate.cpp, 176
 - gamestate.cpp, 179
 - tower.hpp, 186
- TileType
 - map.hpp, 172
- Tower, 137
 - ~Tower, 138
 - GetCoords, 140
 - GetRange, 140
 - IsUpgradeable, 140
 - m_allSprites, 141
 - m_coords, 141
 - m_range, 141
 - Tower, 138
 - towerHealts, 141
 - towerPowers, 141
 - towerPrices, 141
 - towerRanges, 142
- tower.hpp
 - Bachelor, 186
 - Calculator, 186
 - CoffeeTable, 186
 - Doctor, 186
 - Freshman, 186
 - Master, 186
 - Teekkari, 186
 - TILE_SIZE, 186
 - TowerType, 186
- TowerButton, 142
 - ~TowerButton, 144
 - disableButton, 144
 - drawButton, 144
 - enableButton, 144
 - m_name, 145
 - m_sprite, 145
 - TowerButton, 143
- towerHealts
 - Tower, 141
- towerPowers
 - Tower, 141
- towerPrices

- Tower, [141](#)
- towerRanges
 - Tower, [142](#)
- towerTile
 - map.hpp, [173](#)
 - Renderables, [129](#)
- towerTile_sprite
 - Renderables, [129](#)
- TowerTurn
 - Game, [76](#)
- TowerType
 - tower.hpp, [186](#)
- update
 - GUI, [89](#)
- Upgrade
 - AttackingTower, [43](#)
- UpgradeTower
 - Game, [76](#)
 - game.hpp, [165](#)
- UtilFunctions, [29](#)
 - distance, [29](#)
- Validate
 - LevelEditor, [99](#)
- ValidateMap
 - Map, [105](#)
- x_velo
 - GUI, [90](#)
- y_velo
 - GUI, [90](#)