

## Cats vs Rats TD

Generated by Doxygen 1.9.1



<b>1 Contents</b>	<b>1</b>
<b>2 LIBS directory</b>	<b>3</b>
2.1 List of External Libs . . . . .	3
<b>3 Meeting Notes</b>	<b>5</b>
3.1 Meeting 09.11.2022 19::30 . . . . .	5
3.1.1 Summary of works . . . . .	5
3.1.2 Challenges . . . . .	5
3.1.3 Actions . . . . .	5
3.1.4 Project status . . . . .	6
3.1.4.1 TODOs . . . . .	6
3.2 Meeting 10.11.2022 14::00 . . . . .	6
3.2.1 Summary of works . . . . .	6
3.2.2 Challenges . . . . .	6
3.2.3 Actions . . . . .	6
3.2.4 Project status . . . . .	7
3.2.4.1 TODOs . . . . .	7
3.3 Meeting 16.11.2022 12::00 . . . . .	7
3.3.1 Summary of works . . . . .	7
3.3.2 Challenges . . . . .	7
3.3.3 Actions . . . . .	7
3.3.4 Project status . . . . .	7
3.3.4.1 TODOs . . . . .	8
3.4 Meeting 22.11.2022 18:00 . . . . .	8
3.4.1 Summary of works . . . . .	8
3.4.2 Challenges . . . . .	8
3.4.3 Actions . . . . .	8
3.4.4 Project status . . . . .	8
3.4.4.1 TODOs . . . . .	8
3.5 Meeting 04.12.2022 21:00 . . . . .	8
3.5.1 Summary of works . . . . .	9
3.5.2 Challenges . . . . .	9
3.5.3 Actions . . . . .	9
3.5.4 Project status . . . . .	9
3.5.4.1 TODOs . . . . .	9
3.6 Meeting 06.12.2022 13:30 . . . . .	9
3.6.1 Summary of works . . . . .	9
3.6.2 Challenges . . . . .	9
3.6.3 Actions . . . . .	10
3.6.4 Project status . . . . .	10
3.6.4.1 TODOs . . . . .	10
3.7 Meeting 08.12.2022 16:30 . . . . .	10

3.7.1 Summary of works	10
3.7.2 Challenges	10
3.7.3 Actions	10
3.7.4 Project status	10
3.7.4.1 TODOs	10
3.8 Meeting 10.12.2022 21:00	11
3.8.1 Summary of works	11
3.8.2 Challenges	11
3.8.3 Actions	11
3.8.4 Project status	11
3.8.4.1 TODOs	11
<b>4 Contents</b>	<b>13</b>
<b>5 Tower Defense</b>	<b>15</b>
5.1 Group	15
5.2 Repository organization	15
5.3 Project Implementation	15
5.4 Working practices	16
5.5 Source code documentation	16
<b>6 Source content</b>	<b>17</b>
<b>7 Cats vs Rats UI library</b>	<b>19</b>
<b>8 Test files</b>	<b>21</b>
8.1 Unit Tests	21
8.1.1 Test of MyClass	21
<b>9 Namespace Index</b>	<b>23</b>
9.1 Namespace List	23
<b>10 Hierarchical Index</b>	<b>25</b>
10.1 Class Hierarchy	25
<b>11 Class Index</b>	<b>27</b>
11.1 Class List	27
<b>12 File Index</b>	<b>29</b>
12.1 File List	29
<b>13 Namespace Documentation</b>	<b>31</b>
13.1 Fonts Namespace Reference	31
13.1.1 Enumeration Type Documentation	31
13.1.1.1 ID	31
13.2 Maps Namespace Reference	31

13.2.1 Enumeration Type Documentation	31
13.2.1.1 ID	32
13.3 Scenes Namespace Reference	32
13.3.1 Enumeration Type Documentation	32
13.3.1.1 ID	32
13.4 SoundBuffers Namespace Reference	32
13.4.1 Enumeration Type Documentation	33
13.4.1.1 ID	33
13.5 Textures Namespace Reference	33
13.5.1 Enumeration Type Documentation	33
13.5.1.1 ID	33
<b>14 Class Documentation</b>	<b>35</b>
14.1 Block Class Reference	35
14.1.1 Detailed Description	35
14.2 Button Class Reference	35
14.2.1 Detailed Description	36
14.2.2 Constructor & Destructor Documentation	36
14.2.2.1 Button()	36
14.2.2.2 ~Button()	36
14.2.3 Member Function Documentation	36
14.2.3.1 draw()	37
14.2.3.2 handleInput()	37
14.2.3.3 hasBeenClicked()	37
14.2.3.4 mouseHovers()	37
14.2.3.5 onClick()	38
14.2.3.6 update()	38
14.2.4 Member Data Documentation	38
14.2.4.1 clicked	38
14.2.4.2 icon	38
14.2.4.3 mouseIn	38
14.2.4.4 sprite	39
14.3 Contains Class Reference	39
14.3.1 Detailed Description	39
14.4 Enemy Class Reference	39
14.4.1 Detailed Description	40
14.4.2 Constructor & Destructor Documentation	40
14.4.2.1 Enemy()	41
14.4.3 Member Function Documentation	41
14.4.3.1 clone()	41
14.4.3.2 getHitPoints()	42
14.4.3.3 getPathMarkers()	42

14.4.3.4	getSpeed()	42
14.4.3.5	getTextureHolder()	43
14.4.3.6	getValue()	43
14.4.3.7	ifShouldRemove()	43
14.4.3.8	isAlive()	44
14.4.3.9	isAtFinish()	44
14.4.3.10	isNotFrozen()	44
14.4.3.11	make()	44
14.4.3.12	slowDown()	45
14.4.3.13	takeDamage()	45
14.4.3.14	update()	46
14.4.4	Member Data Documentation	47
14.4.4.1	atFinishTile	47
14.4.4.2	hitPoints_	47
14.4.4.3	isFrozen	47
14.4.4.4	nextMarker	47
14.4.4.5	pathMarkers_	47
14.4.4.6	slowdownCounter_	48
14.4.4.7	slowdownFactor_	48
14.4.4.8	slowDownTime	48
14.4.4.9	slowedDownAt_	48
14.4.4.10	speed_	48
14.4.4.11	textures_	48
14.4.4.12	value_	49
14.5	Entity Class Reference	49
14.5.1	Detailed Description	49
14.5.2	Constructor & Destructor Documentation	49
14.5.2.1	Entity()	49
14.5.3	Member Function Documentation	50
14.5.3.1	draw()	50
14.5.3.2	getPosition()	50
14.5.3.3	getTextureID()	51
14.5.3.4	update()	51
14.5.4	Member Data Documentation	51
14.5.4.1	entitySprite	51
14.5.4.2	mVelocity	51
14.5.4.3	textureID_	52
14.6	Game Class Reference	52
14.6.1	Detailed Description	53
14.6.2	Constructor & Destructor Documentation	53
14.6.2.1	Game()	53
14.6.3	Member Function Documentation	53

14.6.3.1 handleSceneChange()	53
14.6.3.2 load()	54
14.6.3.3 processEvents()	54
14.6.3.4 render()	54
14.6.3.5 run()	55
14.6.3.6 Update()	55
14.6.4 Member Data Documentation	55
14.6.4.1 clock	55
14.6.4.2 fonts	56
14.6.4.3 renderStates	56
14.6.4.4 scene	56
14.6.4.5 sounds	56
14.6.4.6 textures	56
14.6.4.7 videomode	57
14.6.4.8 window	57
14.7 GameCommandsMenu Class Reference	57
14.7.1 Detailed Description	57
14.7.2 Constructor & Destructor Documentation	58
14.7.2.1 GameCommandsMenu()	58
14.7.3 Member Function Documentation	59
14.7.3.1 draw()	59
14.7.4 Member Data Documentation	59
14.7.4.1 menuString_	59
14.7.4.2 towerIndexText_	59
14.7.4.3 towerSprites_	59
14.8 GameEnd Class Reference	60
14.8.1 Detailed Description	60
14.8.2 Constructor & Destructor Documentation	61
14.8.2.1 GameEnd()	61
14.8.3 Member Function Documentation	61
14.8.3.1 draw()	61
14.8.3.2 handleInput()	62
14.8.3.3 handlePlayerInput()	62
14.8.3.4 requestedScene()	63
14.8.3.5 sceneType()	63
14.8.3.6 update()	63
14.8.4 Member Data Documentation	64
14.8.4.1 fonts_	64
14.8.4.2 height	64
14.8.4.3 nextScene_	64
14.8.4.4 optionIndex	64
14.8.4.5 options	65

14.8.4.6 textGameOver_ . . . . .	65
14.8.4.7 textInstruction_ . . . . .	65
14.8.4.8 textures_ . . . . .	65
14.8.4.9 width . . . . .	65
14.9 GameStateMenu Class Reference . . . . .	66
14.9.1 Detailed Description . . . . .	66
14.9.2 Constructor & Destructor Documentation . . . . .	66
14.9.2.1 GameStateMenu() . . . . .	66
14.9.3 Member Function Documentation . . . . .	66
14.9.3.1 draw() . . . . .	66
14.9.3.2 update() . . . . .	67
14.9.4 Member Data Documentation . . . . .	67
14.9.4.1 menuString_ . . . . .	67
14.10 GameTitle Class Reference . . . . .	68
14.10.1 Detailed Description . . . . .	68
14.10.2 Constructor & Destructor Documentation . . . . .	69
14.10.2.1 GameTitle() . . . . .	69
14.10.3 Member Function Documentation . . . . .	69
14.10.3.1 draw() . . . . .	69
14.10.3.2 handleInput() . . . . .	70
14.10.3.3 requestedScene() . . . . .	70
14.10.3.4 sceneType() . . . . .	71
14.10.3.5 update() . . . . .	71
14.10.4 Member Data Documentation . . . . .	71
14.10.4.1 fonts_ . . . . .	71
14.10.4.2 grassSprite_ . . . . .	72
14.10.4.3 height . . . . .	72
14.10.4.4 mapSprite_ . . . . .	72
14.10.4.5 mapTex_ . . . . .	72
14.10.4.6 nextScene_ . . . . .	72
14.10.4.7 pathSprite . . . . .	72
14.10.4.8 showText_ . . . . .	73
14.10.4.9 text2_ . . . . .	73
14.10.4.10 text_ . . . . .	73
14.10.4.11 textEffectTime_ . . . . .	73
14.10.4.12 textures_ . . . . .	73
14.10.4.13 width . . . . .	73
14.11 Grid Class Reference . . . . .	74
14.11.1 Detailed Description . . . . .	74
14.11.2 Constructor & Destructor Documentation . . . . .	74
14.11.2.1 Grid() . . . . .	74
14.11.3 Member Function Documentation . . . . .	74



14.11.3.1 draw()	75
14.11.3.2 getSelectedSector()	75
14.11.4 Member Data Documentation	75
14.11.4.1 blocks	75
14.11.4.2 height	75
14.11.4.3 selectedSector	76
14.11.4.4 width	76
14.12 LevelSelect Class Reference	76
14.12.1 Detailed Description	77
14.12.2 Constructor & Destructor Documentation	77
14.12.2.1 LevelSelect()	77
14.12.3 Member Function Documentation	79
14.12.3.1 draw()	79
14.12.3.2 handleInput()	79
14.12.3.3 handlePlayerInput()	80
14.12.3.4 requestedScene()	80
14.12.3.5 sceneType()	81
14.12.3.6 update()	81
14.12.3.7 updateOptionText()	81
14.12.4 Member Data Documentation	82
14.12.4.1 fonts_	82
14.12.4.2 grassSprite_	82
14.12.4.3 height	82
14.12.4.4 mapSprite_	82
14.12.4.5 mapTex_	82
14.12.4.6 nextScene_	83
14.12.4.7 optionIndex	83
14.12.4.8 options	83
14.12.4.9 pathSprite	83
14.12.4.10 showText_	83
14.12.4.11 text_	83
14.12.4.12 textEffectTime_	84
14.12.4.13 textures_	84
14.12.4.14 width	84
14.13 Map Class Reference	84
14.13.1 Detailed Description	84
14.13.2 Member Data Documentation	84
14.13.2.1 diff	85
14.13.2.2 name	85
14.14 MapGrid Class Reference	85
14.14.1 Detailed Description	86
14.14.2 Constructor & Destructor Documentation	86

14.14.2.1 MapGrid()	86
14.14.3 Member Function Documentation	87
14.14.3.1 draw()	87
14.14.3.2 getBlockAt()	88
14.14.3.3 getHeight()	88
14.14.3.4 getWidth()	88
14.14.3.5 setBlockRow()	89
14.14.4 Member Data Documentation	89
14.14.4.1 diff	89
14.14.4.2 height	89
14.14.4.3 map	89
14.14.4.4 mapNum_	89
14.14.4.5 mapSprite_	90
14.14.4.6 mapTex_	90
14.14.4.7 name	90
14.14.4.8 pathMarkers_	90
14.14.4.9 textures_	90
14.14.4.10 width	90
14.15 MapScene Class Reference	91
14.15.1 Detailed Description	92
14.15.2 Constructor & Destructor Documentation	92
14.15.2.1 MapScene()	92
14.15.3 Member Function Documentation	92
14.15.3.1 draw()	92
14.15.3.2 handleInput()	93
14.15.3.3 requestedScene()	93
14.15.3.4 sceneType()	94
14.15.3.5 update()	94
14.15.4 Member Data Documentation	94
14.15.4.1 commandsMenu_	94
14.15.4.2 fonts_	95
14.15.4.3 fpsText	95
14.15.4.4 height	95
14.15.4.5 mapNum_	95
14.15.4.6 nextScene_	95
14.15.4.7 pathMarkers	95
14.15.4.8 soundBuffers_	96
14.15.4.9 statusMenu_	96
14.15.4.10 textures_	96
14.15.4.11 towerMenu_	96
14.15.4.12 waveController_	96
14.15.4.13 wavePause_	96

14.15.4.14 waveStart_ . . . . .	97
14.15.4.15 width . . . . .	97
14.15.4.16 world_ . . . . .	97
14.16 Price Class Reference . . . . .	97
14.16.1 Detailed Description . . . . .	98
14.16.2 Constructor & Destructor Documentation . . . . .	98
14.16.2.1 Price() . . . . .	98
14.16.3 Member Function Documentation . . . . .	98
14.16.3.1 draw() . . . . .	98
14.16.3.2 valueToString() . . . . .	98
14.16.4 Member Data Documentation . . . . .	98
14.16.4.1 price . . . . .	99
14.17 Projectile Class Reference . . . . .	99
14.17.1 Detailed Description . . . . .	100
14.17.2 Constructor & Destructor Documentation . . . . .	100
14.17.2.1 Projectile() . . . . .	100
14.17.3 Member Function Documentation . . . . .	101
14.17.3.1 ifShouldRemove() . . . . .	101
14.17.3.2 make() . . . . .	102
14.17.3.3 update() . . . . .	102
14.17.4 Member Data Documentation . . . . .	105
14.17.4.1 atFinish . . . . .	105
14.17.4.2 closestEnemies_ . . . . .	105
14.17.4.3 closestEnemy_ . . . . .	105
14.17.4.4 damage_ . . . . .	105
14.17.4.5 distance . . . . .	106
14.17.4.6 distanceCovered . . . . .	106
14.17.4.7 finish_ . . . . .	106
14.17.4.8 neighbourEnemies_ . . . . .	106
14.17.4.9 range_ . . . . .	106
14.17.4.10 start_ . . . . .	106
14.17.4.11 textures_ . . . . .	107
14.17.4.12 type_ . . . . .	107
14.17.4.13 velocity_ . . . . .	107
14.18 ProjectileType Class Reference . . . . .	107
14.18.1 Detailed Description . . . . .	107
14.19 request Struct Reference . . . . .	107
14.19.1 Detailed Description . . . . .	108
14.19.2 Member Data Documentation . . . . .	108
14.19.2.1 number . . . . .	108
14.19.2.2 scene . . . . .	108
14.20 ResourceHolder< Resource, Identifier > Class Template Reference . . . . .	108

14.20.1 Detailed Description	108
14.20.2 Member Function Documentation	109
14.20.2.1 get() [1/2]	109
14.20.2.2 get() [2/2]	109
14.20.2.3 load()	109
14.20.3 Member Data Documentation	110
14.20.3.1 mResourceMap	110
14.21 Scene Class Reference	110
14.21.1 Detailed Description	111
14.21.2 Member Function Documentation	111
14.21.2.1 handleInput()	111
14.21.2.2 requestedScene()	111
14.21.2.3 sceneType()	112
14.21.2.4 update()	112
14.21.3 Member Data Documentation	112
14.21.3.1 textures	112
14.22 SceneItem Class Reference	112
14.22.1 Detailed Description	113
14.22.2 Member Typedef Documentation	113
14.22.2.1 Ptr	113
14.22.3 Constructor & Destructor Documentation	113
14.22.3.1 SceneItem()	113
14.22.4 Member Function Documentation	113
14.22.4.1 draw()	114
14.23 Sector Struct Reference	114
14.23.1 Detailed Description	114
14.23.2 Member Function Documentation	115
14.23.2.1 fromCoords()	115
14.23.2.2 operator==()	115
14.23.2.3 upperLeftPoint()	115
14.23.3 Member Data Documentation	115
14.23.3.1 DiagVector	115
14.23.3.2 scale	116
14.23.3.3 Size	116
14.23.3.4 x	116
14.23.3.5 y	116
14.24 SelectTowerButton< T > Class Template Reference	116
14.24.1 Detailed Description	117
14.24.2 Constructor & Destructor Documentation	117
14.24.2.1 SelectTowerButton()	117
14.24.3 Member Function Documentation	117
14.24.3.1 draw()	117

14.24.3.2 onClick()	118
14.24.4 Member Data Documentation	118
14.24.4.1 towerSector	118
14.25 Tower Class Reference	118
14.25.1 Detailed Description	119
14.25.2 Constructor & Destructor Documentation	119
14.25.2.1 Tower()	120
14.25.3 Member Function Documentation	120
14.25.3.1 getClosestEnemies()	120
14.25.3.2 getClosestEnemy()	121
14.25.3.3 getNeighbourEnemies()	121
14.25.3.4 getPrice()	122
14.25.3.5 getRange()	122
14.25.3.6 getSector()	122
14.25.3.7 inRange()	123
14.25.3.8 update()	123
14.25.3.9 upgrade()	124
14.25.4 Member Data Documentation	124
14.25.4.1 bombRange	124
14.25.4.2 damage	125
14.25.4.3 price	125
14.25.4.4 range	125
14.25.4.5 sector	125
14.25.4.6 shootDelay	125
14.25.4.7 soundBomb_	125
14.25.4.8 soundBullet_	126
14.25.4.9 sounds_	126
14.25.4.10 soundSnow_	126
14.25.4.11 testii	126
14.25.4.12 textures_	126
14.25.4.13 timeSinceFiring	126
14.25.4.14 type	127
14.25.4.15 upgradeable	127
14.25.4.16 velocity_	127
14.26 TowerMenu Class Reference	127
14.26.1 Detailed Description	128
14.26.2 Constructor & Destructor Documentation	128
14.26.2.1 TowerMenu()	128
14.26.2.2 ~TowerMenu()	128
14.26.3 Member Function Documentation	128
14.26.3.1 draw()	128
14.26.3.2 handleInput()	129

14.26.3.3 update()	130
14.26.4 Member Data Documentation	130
14.26.4.1 hoverIndicator	130
14.26.4.2 rangeIndicator	130
14.26.4.3 sectorIndicator	130
14.26.4.4 selectedSector	131
14.26.4.5 showSectorIndicator	131
14.27 TowerType Class Reference	131
14.27.1 Detailed Description	131
14.28 Wave Class Reference	131
14.28.1 Detailed Description	132
14.28.2 Constructor & Destructor Documentation	132
14.28.2.1 Wave()	132
14.28.3 Member Function Documentation	133
14.28.3.1 getEnemiesLeft()	133
14.28.3.2 ifNextEnemy()	133
14.28.3.3 isEmpty()	134
14.28.4 Member Data Documentation	134
14.28.4.1 elapsed_	134
14.28.4.2 enemiesLeft	134
14.28.4.3 enemyTemplate	134
14.28.4.4 spacing_	134
14.28.4.5 SPACING_HUGE	135
14.28.4.6 SPACING_MEDIUM	135
14.28.4.7 SPACING_NARROW	135
14.28.4.8 SPACING_WIDE	135
14.29 WaveController Class Reference	135
14.29.1 Detailed Description	136
14.29.2 Constructor & Destructor Documentation	136
14.29.2.1 WaveController()	136
14.29.3 Member Function Documentation	137
14.29.3.1 getWaveEnemiesLeft()	137
14.29.3.2 getWaveNumber()	137
14.29.3.3 makeNewWave()	137
14.29.3.4 update()	139
14.29.4 Member Data Documentation	140
14.29.4.1 pathMarkers_	140
14.29.4.2 textureholder_	140
14.29.4.3 wave	140
14.29.4.4 waveNumber	140
14.30 WavePause Class Reference	141
14.30.1 Detailed Description	141

14.30.2 Constructor & Destructor Documentation	141
14.30.2.1 WavePause()	141
14.30.3 Member Function Documentation	142
14.30.3.1 draw()	142
14.30.3.2 handleInput()	142
14.30.3.3 update()	143
14.30.4 Member Data Documentation	143
14.30.4.1 menuString_	143
14.30.4.2 mousePosition_	143
14.30.4.3 square_	143
14.30.4.4 triangle_	143
14.31 WaveStart Class Reference	144
14.31.1 Detailed Description	144
14.31.2 Constructor & Destructor Documentation	144
14.31.2.1 WaveStart()	144
14.31.3 Member Function Documentation	145
14.31.3.1 draw()	145
14.31.3.2 handleInput()	145
14.31.3.3 update()	146
14.31.4 Member Data Documentation	146
14.31.4.1 menuString_	146
14.31.4.2 mousePosition_	146
14.31.4.3 square_	146
14.31.4.4 triangle_	146
14.32 World Class Reference	147
14.32.1 Detailed Description	148
14.32.2 Constructor & Destructor Documentation	148
14.32.2.1 World()	148
14.32.3 Member Function Documentation	149
14.32.3.1 addProjectile()	149
14.32.3.2 addTower()	150
14.32.3.3 clean() [1/2]	150
14.32.3.4 clean() [2/2]	150
14.32.3.5 draw() [1/3]	151
14.32.3.6 draw() [2/3]	151
14.32.3.7 draw() [3/3]	151
14.32.3.8 getEnemies()	152
14.32.3.9 getHP()	152
14.32.3.10 getMapGrid()	152
14.32.3.11 getMoney()	153
14.32.3.12 getSounds()	153
14.32.3.13 getTextures()	153

14.32.3.14 getTowerAt()	153
14.32.3.15 removeTower()	154
14.32.3.16 update() [1/3]	154
14.32.3.17 update() [2/3]	155
14.32.3.18 update() [3/3]	155
14.32.3.19 upgradeTower()	156
14.32.4 Member Data Documentation	156
14.32.4.1 deathSound_	156
14.32.4.2 endHouse	156
14.32.4.3 enemies	157
14.32.4.4 explosionSound_	157
14.32.4.5 grid_	157
14.32.4.6 hp_	157
14.32.4.7 isReadyForNextWave	157
14.32.4.8 mapNum_	157
14.32.4.9 money_	158
14.32.4.10 paused	158
14.32.4.11 projectile	158
14.32.4.12 projectiles	158
14.32.4.13 sounds	158
14.32.4.14 textures	158
14.32.4.15 towerGrid_	159
14.32.4.16 towers	159
<b>15 File Documentation</b>	<b>161</b>
15.1 doc/readme.md File Reference	161
15.2 libs/readme.md File Reference	161
15.3 plan/readme.md File Reference	161
15.4 src/readme.md File Reference	161
15.5 src/ui/readme.md File Reference	161
15.6 tests/readme.md File Reference	161
15.7 Meeting-notes.md File Reference	161
15.8 README.md File Reference	161
15.9 src/entity/Enemy.cpp File Reference	161
15.10 src/entity/Enemy.hpp File Reference	162
15.10.1 Typedef Documentation	162
15.10.1.1 Enemies	162
15.10.1.2 EnemyPtr	162
15.11 src/entity/Entity.cpp File Reference	162
15.12 src/entity/Entity.hpp File Reference	163
15.13 src/entity/Projectile.cpp File Reference	163
15.14 src/entity/Projectile.hpp File Reference	163



15.14.1 Typedef Documentation . . . . .	164
15.14.1.1 ProjectilePtr . . . . .	164
15.14.1.2 Projectiles . . . . .	164
15.14.2 Enumeration Type Documentation . . . . .	164
15.14.2.1 ProjectileType . . . . .	164
15.15 src/entity/Rat.cpp File Reference . . . . .	164
15.16 src/entity/Rat.hpp File Reference . . . . .	164
15.17 src/entity/Tower.cpp File Reference . . . . .	164
15.17.1 Variable Documentation . . . . .	165
15.17.1.1 frameDelay . . . . .	165
15.18 src/entity/Tower.hpp File Reference . . . . .	165
15.18.1 Typedef Documentation . . . . .	166
15.18.1.1 TowerPtr . . . . .	166
15.18.1.2 Towers . . . . .	166
15.18.2 Enumeration Type Documentation . . . . .	166
15.18.2.1 TowerType . . . . .	166
15.19 src/game/Block.cpp File Reference . . . . .	166
15.20 src/game/Block.hpp File Reference . . . . .	166
15.21 src/game/Game.cpp File Reference . . . . .	167
15.22 src/game/Game.hpp File Reference . . . . .	167
15.23 src/game/GameEnd.cpp File Reference . . . . .	167
15.24 src/game/GameEnd.hpp File Reference . . . . .	167
15.25 src/game/GameTitle.cpp File Reference . . . . .	168
15.26 src/game/GameTitle.hpp File Reference . . . . .	168
15.27 src/game/LevelSelect.cpp File Reference . . . . .	168
15.28 src/game/LevelSelect.hpp File Reference . . . . .	168
15.29 src/game/Map.cpp File Reference . . . . .	169
15.30 src/game/Map.hpp File Reference . . . . .	169
15.31 src/game/MapGrid.cpp File Reference . . . . .	169
15.32 src/game/MapGrid.hpp File Reference . . . . .	169
15.33 src/game/MapScene.cpp File Reference . . . . .	169
15.34 src/game/MapScene.hpp File Reference . . . . .	170
15.35 src/game/Scene.cpp File Reference . . . . .	170
15.36 src/game/Scene.hpp File Reference . . . . .	170
15.36.1 Typedef Documentation . . . . .	171
15.36.1.1 ScenePtr . . . . .	171
15.36.1.2 sceneRequest . . . . .	171
15.37 src/game/Wave.cpp File Reference . . . . .	171
15.38 src/game/Wave.hpp File Reference . . . . .	171
15.38.1 Typedef Documentation . . . . .	172
15.38.1.1 WavePtr . . . . .	172
15.39 src/game/WaveController.cpp File Reference . . . . .	172

15.39.1 Variable Documentation	172
15.39.1.1 delayTime	172
15.40 src/game/WaveController.hpp File Reference	172
15.41 src/game/World.cpp File Reference	173
15.42 src/game/World.hpp File Reference	173
15.43 src/main.cpp File Reference	173
15.43.1 Function Documentation	173
15.43.1.1 main()	174
15.44 src/resource/Resource.cpp File Reference	174
15.45 src/resource/Resource.hpp File Reference	174
15.45.1 Typedef Documentation	175
15.45.1.1 FontHolder	175
15.45.1.2 MapHolder	175
15.45.1.3 SoundBufferHolder	175
15.45.1.4 TextureHolder	175
15.46 src/sceneltem/Sceneltem.cpp File Reference	176
15.47 src/sceneltem/Sceneltem.hpp File Reference	176
15.48 src/ui/Button.cpp File Reference	176
15.49 src/ui/Button.hpp File Reference	176
15.49.1 Typedef Documentation	176
15.49.1.1 ButtonPtr	177
15.49.1.2 Buttons	177
15.50 src/ui/GameCommandsMenu.cpp File Reference	177
15.51 src/ui/GameCommandsMenu.hpp File Reference	177
15.52 src/ui/GameStatusMenu.cpp File Reference	177
15.53 src/ui/GameStatusMenu.hpp File Reference	177
15.54 src/ui/Grid.cpp File Reference	178
15.55 src/ui/Grid.hpp File Reference	178
15.56 src/ui/Price.cpp File Reference	178
15.57 src/ui/Price.hpp File Reference	178
15.58 src/ui/Sector.cpp File Reference	178
15.58.1 Function Documentation	179
15.58.1.1 operator+()	179
15.59 src/ui/Sector.hpp File Reference	179
15.59.1 Typedef Documentation	179
15.59.1.1 Path	180
15.59.2 Function Documentation	180
15.59.2.1 operator+()	180
15.60 src/ui/SelectTowerButton.hpp File Reference	180
15.61 src/ui/TowerMenu.cpp File Reference	180
15.62 src/ui/TowerMenu.hpp File Reference	181
15.63 src/ui/WavePause.cpp File Reference	181

---

15.64 src/ui/WavePause.hpp File Reference . . . . .	181
15.65 src/ui/WaveStart.cpp File Reference . . . . .	181
15.66 src/ui/WaveStart.hpp File Reference . . . . .	181
<b>Index</b>	<b>183</b>



# Chapter 1

## 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 2

# LIBS directory

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

### 2.1 List of External Libs

1. `Project1`
2. `Project2`

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





## Chapter 3

# Meeting Notes

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

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

### 3.1 Meeting 09.11.2022 19::30

#### Participants:

1. Henrik Turtinen
2. Antti Chen

#### 3.1.1 Summary of works

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

#### 3.1.2 Challenges

1. ...

#### 3.1.3 Actions

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

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

### 3.1.4 Project status

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

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

#### 3.1.4.1 TODOs

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

## 3.2 Meeting 10.11.2022 14::00

### Participants:

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

### 3.2.1 Summary of works

1. Henrik Turtinen  
Started the project plan. It includes sketch of the [Game](#) UI and the basic idea: cats against rodents. Created GitLab repo to version.aalto.fi
2. Antti Chen  
Project plan: same as above

### 3.2.2 Challenges

1. ...

### 3.2.3 Actions

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

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

### 3.2.4 Project status

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

Framework: SFML

#### 3.2.4.1 TODOs

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

## 3.3 Meeting 16.11.2022 12::00

### Participants:

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

### 3.3.1 Summary of works

No work done between this and the last meeting.

### 3.3.2 Challenges

### 3.3.3 Actions

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

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

### 3.3.4 Project status

Project plan was ok according to the advisor.

#### 3.3.4.1 TODOs

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

### 3.4 Meeting 22.11.2022 18:00

#### Participants:

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

#### 3.4.1 Summary of works

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

#### 3.4.2 Challenges

#### 3.4.3 Actions

#### 3.4.4 Project status

The project is running late on schedule

##### 3.4.4.1 TODOs

1. All: continue working on the project

### 3.5 Meeting 04.12.2022 21:00

#### Participants:

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

### 3.5.1 Summary of works

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

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

### 3.5.2 Challenges

### 3.5.3 Actions

### 3.5.4 Project status

The project is running late on schedule

#### 3.5.4.1 TODOs

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

## 3.6 Meeting 06.12.2022 13:30

### Participants:

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

### 3.6.1 Summary of works

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

### 3.6.2 Challenges

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

### 3.6.3 Actions

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

### 3.6.4 Project status

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

#### 3.6.4.1 TODOs

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

## 3.7 Meeting 08.12.2022 16:30

### Participants:

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

### 3.7.1 Summary of works

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

### 3.7.2 Challenges

### 3.7.3 Actions

### 3.7.4 Project status

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

#### 3.7.4.1 TODOs

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

## 3.8 Meeting 10.12.2022 21:00

### Participants:

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

### 3.8.1 Summary of works

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

### 3.8.2 Challenges

### 3.8.3 Actions

### 3.8.4 Project status

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

#### 3.8.4.1 TODOs

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





## Chapter 4

# Contents

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

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

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

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



## Chapter 5

# Tower Defense

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

### 5.1 Group

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

### 5.2 Repository organization

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

### 5.3 Project Implementation

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

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

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

## 5.4 Working practices

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

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

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

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

## 5.5 Source code documentation

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

## Chapter 6

# Source content

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

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

This folder has folders for the games source and media files

[Entity](#) folder includes code files related to creating entities, like towers and enemies

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

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

Resource folder includes the implementation of a resource holder.

[SceneItem](#) folder

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



## Chapter 7

# Cats vs Rats UI library

This subfolder contains classes that are used in [Game](#) scene.

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

Side menu uses the following classes:

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





## Chapter 8

# Test files

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

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

## 8.1 Unit Tests

### 8.1.1 Test of MyClass

**Involved Classes:**

**Test File:**

**Results:**



## Chapter 9

# Namespace Index

### 9.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Fonts	31
Maps	31
Scenes	32
SoundBuffers	32
Textures	33



## Chapter 10

# Hierarchical Index

### 10.1 Class Hierarchy

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

Block . . . . .	35
Contains . . . . .	39
sf::Drawable	
Button . . . . .	35
SelectTowerButton< T > . . . . .	116
Entity . . . . .	49
Enemy . . . . .	39
Projectile . . . . .	99
Tower . . . . .	118
GameCommandsMenu . . . . .	57
GameStatusMenu . . . . .	66
Grid . . . . .	74
MapGrid . . . . .	85
Price . . . . .	97
Scene . . . . .	110
GameEnd . . . . .	60
GameTitle . . . . .	68
LevelSelect . . . . .	76
MapScene . . . . .	91
ScenItem . . . . .	112
TowerMenu . . . . .	127
WavePause . . . . .	141
WaveStart . . . . .	144
World . . . . .	147
Game . . . . .	52
Map . . . . .	84
sf::NonCopyable	
ScenItem . . . . .	112
ProjectileType . . . . .	107
request . . . . .	107
ResourceHolder< Resource, Identifier > . . . . .	108
ResourceHolder< sf::Font, Fonts::ID > . . . . .	108
ResourceHolder< sf::SoundBuffer, SoundBuffers::ID > . . . . .	108
ResourceHolder< sf::Texture, Textures::ID > . . . . .	108
Sector . . . . .	114

TowerType . . . . .	131
sf::Transformable	
SceneItem . . . . .	112
Wave . . . . .	131
WaveController . . . . .	135

## Chapter 11

# Class Index

### 11.1 Class List

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

<a href="#">Block</a>	35
<a href="#">Button</a>	35
<a href="#">Contains</a>	39
<a href="#">Enemy</a>	
A class for ingame enemies. Derived from <a href="#">Entity</a> class	39
<a href="#">Entity</a>	
Visible entity on the map	49
<a href="#">Game</a>	
Class where the game, setting are set and loading and rendering is called from	52
<a href="#">GameCommandsMenu</a>	
Shows the player how to play the game (at the right side of the screen in a <a href="#">MapScene</a> )	57
<a href="#">GameEnd</a>	
A class that inherits <a href="#">Scene</a> , is shown when the player loses the game	60
<a href="#">GameStatusMenu</a>	
Shows the player information from the game world: hp, money, enemies left and wave number	66
<a href="#">GameTitle</a>	
Welcome screen for the game	68
<a href="#">Grid</a>	
<a href="#">Grid</a> is used to place the towers according to the visual tiles	74
<a href="#">LevelSelect</a>	
A class for level selection menu, inherits <a href="#">Scene</a> class	76
<a href="#">Map</a>	84
<a href="#">MapGrid</a>	
The game map consists of blocks, this class handles reading maps from file and rendering the grid	85
<a href="#">MapScene</a>	
A class used when ingame. Inherits from <a href="#">Scene</a> class. The main scene of the game	91
<a href="#">Price</a>	97
<a href="#">Projectile</a>	
Extends <a href="#">Entity</a> class, Bombs, Bullets etc	99
<a href="#">ProjectileType</a>	
<a href="#">Contains</a> identification for different projectiles	107
<a href="#">request</a>	107
<a href="#">ResourceHolder&lt; Resource, Identifier &gt;</a>	108
<a href="#">Scene</a>	
This is a class for different UI "pages" of the game such as main menu or the game itself	110

<a href="#">SceneItem</a>	112
<a href="#">Sector</a>	
A <a href="#">Sector</a> is a 64x64 pixel block in the game map The <a href="#">Sector</a> class is used to align towers properly	114
<a href="#">SelectTowerButton&lt; T &gt;</a>	116
<a href="#">Tower</a>	
<a href="#">Tower</a> to display on the map and shoot, extends <a href="#">Entity</a> class	118
<a href="#">TowerMenu</a>	
<a href="#">TowerMenu</a> class allows the player to buy and upgrade towers. Controls: Left click to select a <a href="#">Sector</a> . Right click to sell a tower at selected <a href="#">Sector</a> . 1 to buy tower 1 (GunCat) 2 to buy tower 2 (FreezeCat) 3 to buy tower 3 (BombCat) 4 to upgrade tower in selected <a href="#">Sector</a>	127
<a href="#">TowerType</a>	
Possible <a href="#">Tower</a> types	131
<a href="#">Wave</a>	
A class for a single wave of enemies. A wave can be started by a player with a button	131
<a href="#">WaveController</a>	
<a href="#">WaveController</a> controls the current wave and makes a new one when the player is ready	135
<a href="#">WavePause</a>	
A class for pause button to pause a wave of enemies	141
<a href="#">WaveStart</a>	
A class for the button to start next wave	144
<a href="#">World</a>	
The world class houses all of the things in a game level. Towers, enemies, map, map grid, etc.	
All those elements are used here to run and update the game	147



# Chapter 12

## File Index

### 12.1 File List

Here is a list of all files with brief descriptions:

src/main.cpp	173
src/entity/Enemy.cpp	161
src/entity/Enemy.hpp	162
src/entity/Entity.cpp	162
src/entity/Entity.hpp	163
src/entity/Projectile.cpp	163
src/entity/Projectile.hpp	163
src/entity/Rat.cpp	164
src/entity/Rat.hpp	164
src/entity/Tower.cpp	164
src/entity/Tower.hpp	165
src/game/Block.cpp	166
src/game/Block.hpp	166
src/game/Game.cpp	167
src/game/Game.hpp	167
src/game/GameEnd.cpp	167
src/game/GameEnd.hpp	167
src/game/GameTitle.cpp	168
src/game/GameTitle.hpp	168
src/game/LevelSelect.cpp	168
src/game/LevelSelect.hpp	168
src/game/Map.cpp	169
src/game/Map.hpp	169
src/game/MapGrid.cpp	169
src/game/MapGrid.hpp	169
src/game/MapScene.cpp	169
src/game/MapScene.hpp	170
src/game/Scene.cpp	170
src/game/Scene.hpp	170
src/game/Wave.cpp	171
src/game/Wave.hpp	171
src/game/WaveController.cpp	172
src/game/WaveController.hpp	172
src/game/World.cpp	173
src/game/World.hpp	173

src/resource/Resource.cpp	174
src/resource/Resource.hpp	174
src/sceneltem/Sceneltem.cpp	176
src/sceneltem/Sceneltem.hpp	176
src/ui/Button.cpp	176
src/ui/Button.hpp	176
src/ui/GameCommandsMenu.cpp	177
src/ui/GameCommandsMenu.hpp	177
src/ui/GameStatusMenu.cpp	177
src/ui/GameStatusMenu.hpp	177
src/ui/Grid.cpp	178
src/ui/Grid.hpp	178
src/ui/Price.cpp	178
src/ui/Price.hpp	178
src/ui/Sector.cpp	178
src/ui/Sector.hpp	179
src/ui/SelectTowerButton.hpp	180
src/ui/TowerMenu.cpp	180
src/ui/TowerMenu.hpp	181
src/ui/WavePause.cpp	181
src/ui/WavePause.hpp	181
src/ui/WaveStart.cpp	181
src/ui/WaveStart.hpp	181

## Chapter 13

# Namespace Documentation

### 13.1 Fonts Namespace Reference

#### Enumerations

- enum [ID](#) { [GameTitleFont](#) }

#### 13.1.1 Enumeration Type Documentation

##### 13.1.1.1 ID

```
enum Fonts::ID
```

#### Enumerator

GameTitleFont	
---------------	--

Definition at line 44 of file Resource.hpp.

```
44 { GameTitleFont };
```

### 13.2 Maps Namespace Reference

#### Enumerations

- enum [ID](#) { [Map](#) }

#### 13.2.1 Enumeration Type Documentation

### 13.2.1.1 ID

```
enum Maps::ID
```

#### Enumerator

Map	
-----	--

Definition at line 40 of file Resource.hpp.

```
40 { Map };
```

## 13.3 Scenes Namespace Reference

### Enumerations

- enum class ID { [GameTitle](#) , [LevelSelect](#) , [MapScene](#) , [GameEnd](#) }

### 13.3.1 Enumeration Type Documentation

#### 13.3.1.1 ID

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

#### Enumerator

GameTitle	
LevelSelect	Title menu of the game, has name and list of creators.
MapScene	Main menu for selecting map.
GameEnd	<a href="#">Game</a> and its GUI. For <a href="#">Game</a> Over <a href="#">Scene</a>

Definition at line 17 of file Scene.hpp.

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

## 13.4 SoundBuffers Namespace Reference

### Enumerations

- enum ID {  
[EnemyDeath](#) , [Explosion](#) , [GunCat](#) , [BombCatMeow](#) ,  
[FreezeCatMeow](#) }

### 13.4.1 Enumeration Type Documentation

#### 13.4.1.1 ID

```
enum SoundBuffers::ID
```

Enumerator

EnemyDeath	
Explosion	
GunCat	
BombCatMeow	
FreezeCatMeow	

Definition at line 47 of file Resource.hpp.

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

## 13.5 Textures Namespace Reference

### Enumerations

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

### 13.5.1 Enumeration Type Documentation

#### 13.5.1.1 ID

```
enum Textures::ID
```

Enumerator

PathTile	
GrassTile	
HouseTile	
GunCat	
UpgradedGunCat	
FreezeCat	
UpgradedFzeezeCat	

## Enumerator

BombCat	
UpgradedBombCat	
FatRat	
FastRat	
BasicRat	
Bullet	
Bomb	
Snowflake	
PlayButton	
Explosion	

Definition at line 18 of file Resource.hpp.

```
18     {
19     PathTile,
20     GrassTile,
21     HouseTile,
22     GunCat,
23     UpgradedGunCat,
24     FreezeCat,
25     UpgradedFzeezeCat,
26     BombCat,
27     UpgradedBombCat,
28     FatRat,
29     FastRat,
30     BasicRat,
31     Bullet,
32     Bomb,
33     Snowflake,
34     PlayButton,
35     Explosion
36 };
```

## Chapter 14

# Class Documentation

### 14.1 Block Class Reference

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

#### 14.1.1 Detailed Description

Definition at line 8 of file Block.hpp.

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

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

### 14.2 Button Class Reference

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

Inheritance diagram for Button:

Collaboration diagram for Button:

#### Public Member Functions

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

#### Protected Member Functions

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

## Private Attributes

- sf::CircleShape [sprite](#)
- sf::Sprite [icon](#)
- bool [mouseIn](#)
- bool [clicked](#)

### 14.2.1 Detailed Description

Definition at line 11 of file Button.hpp.

### 14.2.2 Constructor & Destructor Documentation

#### 14.2.2.1 Button()

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

Definition at line 9 of file Button.cpp.

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

#### 14.2.2.2 ~Button()

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

Definition at line 14 of file Button.hpp.

```
14 {};
```

### 14.2.3 Member Function Documentation



### 14.2.3.1 draw()

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

Reimplemented in [SelectTowerButton< T >](#).

Definition at line 35 of file Button.cpp.

```
35                                     {
36     target.draw(sprite, states);
37     target.draw(icon, states);
38 }
```

### 14.2.3.2 handleInput()

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

Definition at line 22 of file Button.cpp.

```
22                                     {
23     if (event.type == sf::Event::MouseButtonPressed
24         && event.mouseButton.button == sf::Mouse::Button::Left) {
25         onClick(world);
26         return true;
27     }
28     return false;
29 }
```

### 14.2.3.3 hasBeenClicked()

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

Definition at line 18 of file Button.hpp.

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

### 14.2.3.4 mouseHovers()

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

Definition at line 20 of file Button.hpp.

```
20 { return mouseIn; }
```

#### 14.2.3.5 onClick()

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

Reimplemented in [SelectTowerButton< T >](#).

Definition at line 40 of file Button.cpp.

```
40 {
41     clicked = true;
42 }
```

#### 14.2.3.6 update()

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

Definition at line 31 of file Button.cpp.

```
31 {
32     mouseIn = sprite.getGlobalBounds().contains(mousePosition);
33 }
```

### 14.2.4 Member Data Documentation

#### 14.2.4.1 clicked

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

Definition at line 26 of file Button.hpp.

#### 14.2.4.2 icon

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

Definition at line 24 of file Button.hpp.

#### 14.2.4.3 mouseIn

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

Definition at line 25 of file Button.hpp.

#### 14.2.4.4 sprite

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

Definition at line 23 of file Button.hpp.

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

- [src/ui/Button.hpp](#)
- [src/ui/Button.cpp](#)

## 14.3 Contains Class Reference

### 14.3.1 Detailed Description

of [Scenes](#)

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

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

## 14.4 Enemy Class Reference

A class for ingame enemies. Derived from [Entity](#) class.

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

Inheritance diagram for Enemy:

Collaboration diagram for Enemy:

### Public Member Functions

- [Enemy](#) ([TextureHolder](#) &textures, int textureID, std::map< int, std::pair< int, int >> &pathMarkers, int hitPoints, float speed)
- void [update](#) (sf::Time deltaTime, [World](#) &world)
- [EnemyPtr](#) [clone](#) ([TextureHolder](#) &textures, int textureID, std::map< int, std::pair< int, int >> &pathMarkers, int hitPoints, float speed) const  
*Copies an enemy.*
- void [takeDamage](#) (int damageAmount)  
*Makes this enemy instance take damage.*
- void [slowDown](#) ()  
*Slows the enemy down (caused by FreezeCat tower).*
- [TextureHolder](#) & [getTextureHolder](#) () const  
*returns a reference to the TextureHolder that contains the texture for this enemy.*
- std::map< int, std::pair< int, int >> & [getPathMarkers](#) () const  
*Gets a reference to this Enemy's pathmarkers list.*
- int [getHitPoints](#) () const

- returns hitpoints for this enemy (int)*
  - int [getValue](#) () const  
*Get value for this enemy (how much money will killing it give).*
  - float [getSpeed](#) () const  
*Get speed for this enemy.*
  - bool [isAtFinish](#) () const
  - bool [isAlive](#) () const  
*Is this enemy alive (hitpoints>0)?*
  - bool [ifShouldRemove](#) () const  
*Should this enemy be removed from the enemies list?*
  - bool [isNotFrozen](#) () const  
*Is this enemy not frozen (slowed down)?*

## Static Public Member Functions

- static [EnemyPtr](#) [make](#) ([TextureHolder](#) &textures, int textureID, std::map< int, std::pair< int, int >> &pathMarkers, int hitPoints, float speed)  
*Makes a new enemy. Returns a pointer to it.*

## Private Attributes

- std::map< int, std::pair< int, int > > & [pathMarkers\\_](#)
- [TextureHolder](#) & [textures\\_](#)
- float [speed\\_](#)
- float [slowdownFactor\\_](#) = 1.0
- int [slowdownCounter\\_](#)
- sf::Time [slowDownTime](#) = sf::seconds(.1f)
- sf::Time [slowedDownAt\\_](#) = sf::seconds(0.f)
- int [nextMarker](#)
- int [hitPoints\\_](#)
- int [value\\_](#)
- bool [atFinishTile](#)
- bool [isFrozen](#)

## Additional Inherited Members

### 14.4.1 Detailed Description

A class for ingame enemies. Derived from [Entity](#) class.

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

Definition at line 21 of file Enemy.hpp.

### 14.4.2 Constructor & Destructor Documentation

### 14.4.2.1 Enemy()

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

Definition at line 10 of file Enemy.cpp.

```
10
11     :
12     Entity(textureID, textures),
13     textures_(textures),
14     pathMarkers_(pathMarkers),
15     atFinishTile(false),
16     hitPoints_(hitPoints),
17     value_(hitPoints),
18     speed_(speed),
19     slowdownCounter_(0),
20     isFrozen(false)
21 {
22     int tileX = pathMarkers_.at(1).first;
23     int tileY = pathMarkers_.at(1).second;
24     float coordX = tileX * 64.f;
25     float coordY = tileY * 64.f;
26     entitySprite.setPosition(coordX, coordY);
27     nextMarker = 2;
28     //std::cout << "x: " << entitySprite.getPosition().x << " y: " << entitySprite.getPosition().y << std::endl;
29 }
30 }
```

## 14.4.3 Member Function Documentation

### 14.4.3.1 clone()

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

Copies an enemy.

#### Parameters

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

#### Returns

A shared pointer to the copied enemy.

Definition at line 100 of file Enemy.cpp.

```
103 {  
104     return std::make_shared<Enemy> (textures, textureID, pathMarkers, hitPoints, speed);  
105 }
```

#### 14.4.3.2 getHitPoints()

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

returns hitpoints for this enemy (int)

##### Returns

Hitpoints as int

Definition at line 76 of file Enemy.hpp.

```
76 { return hitPoints_; }
```

#### 14.4.3.3 getPathMarkers()

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

Gets a reference to this [Enemy](#)'s pathmarkers list.

##### Returns

Reference to the pathmarkers list.

Definition at line 71 of file Enemy.hpp.

```
71 { return pathMarkers_; }
```

#### 14.4.3.4 getSpeed()

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

Get speed for this enemy.

##### Returns

Speed as float.

Definition at line 86 of file Enemy.hpp.

```
86 { return speed_; }
```

#### 14.4.3.5 getTextureHolder()

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

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

##### Returns

Definition at line 66 of file Enemy.hpp.

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

#### 14.4.3.6 getValue()

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

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

##### Returns

Value as int.

Definition at line 81 of file Enemy.hpp.

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

#### 14.4.3.7 ifShouldRemove()

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

Should this enemy be removed from the enemies list?

##### Returns

true if enemy should be removed, otherwise false.

Definition at line 101 of file Enemy.hpp.

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

#### 14.4.3.8 isAlive()

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

Is this enemy alive (hitpoints>0)?

##### Returns

true if enemy is alive, otherwise false.

Definition at line 96 of file Enemy.hpp.

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

#### 14.4.3.9 isAtFinish()

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

Is this enemy at the finish line (house)?

##### Returns

true if enemy has reached the house, otherwise false.

Definition at line 91 of file Enemy.hpp.

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

#### 14.4.3.10 isNotFrozen()

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

Is this enemy not frozen (slowed down)?

##### Returns

false if enemy is frozen, otherwise true.

Definition at line 106 of file Enemy.hpp.

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

#### 14.4.3.11 make()

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

Makes a new enemy. Returns a pointer to it.



**Parameters**

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

**Returns**

A shared pointer to the created enemy.

Definition at line 39 of file Enemy.hpp.

```
39 {
40     return std::make_unique<Enemy>(textures, textureID, pathMarkers, hitPoints, speed);
41 }
```

**14.4.3.12 slowDown()**

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

Slows the enemy down (caused by FreezeCat tower).

Definition at line 31 of file Enemy.cpp.

```
31 {
32     slowdownCounter_ = 0;
33     slowdownFactor_ = 0.5f;
34     isFrozen = true;
35 }
```

**14.4.3.13 takeDamage()**

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

Makes this enemy instance take damage.

**Parameters**

<i>damageAmount</i>	Damage to take.
---------------------	-----------------

Definition at line 57 of file Enemy.hpp.

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

### 14.4.3.14 update()

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

#### Parameters

<i>deltaTime</i>	time since last update in <a href="#">Game</a> loop
<i>world</i>	<a href="#">World</a> class the enemy belongs to

Implements [Entity](#).

Definition at line 36 of file Enemy.cpp.

```
36         {
37
38     int finishTileX = pathMarkers_.at(pathMarkers_.size()).first;
39     int finishTileY = pathMarkers_.at(pathMarkers_.size()).second;
40     float finishCoordX = finishTileX * 64.f;
41     float finishCoordY = finishTileY * 64.f;
42
43     if (!(abs(finishCoordX - entitySprite.getPosition().x) <= 4.f && abs(finishCoordY -
44         entitySprite.getPosition().y) <= 4.f)) {
45
46         //Check if has reached next marker
47         int nextTileX = pathMarkers_.at(nextMarker).first;
48         int nextTileY = pathMarkers_.at(nextMarker).second;
49         float nextCoordX = nextTileX * 64.f;
50         float nextCoordY = nextTileY * 64.f;
51
52         if (pathMarkers_.size() != nextMarker) {
53             if (abs(nextCoordX - entitySprite.getPosition().x) <= 4.f && abs(nextCoordY -
54                 entitySprite.getPosition().y) <= 4.f) {
55                 entitySprite.setPosition(nextCoordX, nextCoordY);
56                 nextMarker++;
57             }
58
59             //Recalculate next marker coordinates in case of nextmarker++
60             nextTileX = pathMarkers_.at(nextMarker).first;
61             nextTileY = pathMarkers_.at(nextMarker).second;
62             nextCoordX = nextTileX * 64.f;
63             nextCoordY = nextTileY * 64.f;
64             //float movement = round(speed_ * slowdownFactor_);
65             float movement = speed_ * slowdownFactor_;
66
67             //Move either along x- or y-axis
68             if (entitySprite.getPosition().x == nextCoordX) {
69                 if (entitySprite.getPosition().y < nextCoordY) {
70                     entitySprite.move(0.0f, movement);
71                 } else {
72                     entitySprite.move(0.0f, -movement);
73                 }
74             } else if (entitySprite.getPosition().y == nextCoordY) {
75                 if (entitySprite.getPosition().x < nextCoordX) {
76                     entitySprite.move(movement, 0.0f);
77                 } else {
78                     entitySprite.move(-movement, 0.0f);
79                 }
80             }
81
82             // std::cout << "X: " << entitySprite.getPosition().x << ",Y: " << entitySprite.getPosition().y << "\n";
83             /*
84             std::cout << "getPos x: " << entitySprite.getPosition().x << " y: " << entitySprite.getPosition().y <<
85             std::ends;
86             std::cout << "    nextCr x: " << nextCoordX << " y: " << nextCoordY << std::ends;
87             std::cout << "    nextMarker: " << nextMarker << std::ends;
88             std::cout << "    finCr x: " << finishCoordX << " y: " << finishCoordY << std::endl;
89             */
90
91             slowdownCounter_++;
92             // handle slowdown
93             if (slowdownFactor_ < 1.0f && slowdownCounter_ >= 100) {
94                 slowdownFactor_ = 1.0;
95                 isFrozen = false;
96             }
97         } else {
98             atFinishTile = true;
99         }
100     }
```

## 14.4.4 Member Data Documentation

### 14.4.4.1 atFinishTile

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

Definition at line 119 of file Enemy.hpp.

### 14.4.4.2 hitPoints\_

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

Definition at line 117 of file Enemy.hpp.

### 14.4.4.3 isFrozen

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

Definition at line 120 of file Enemy.hpp.

### 14.4.4.4 nextMarker

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

Definition at line 116 of file Enemy.hpp.

### 14.4.4.5 pathMarkers\_

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

Definition at line 109 of file Enemy.hpp.

#### 14.4.4.6 slowdownCounter\_

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

Definition at line 113 of file Enemy.hpp.

#### 14.4.4.7 slowdownFactor\_

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

Definition at line 112 of file Enemy.hpp.

#### 14.4.4.8 slowDownTime

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

Definition at line 114 of file Enemy.hpp.

#### 14.4.4.9 slowedDownAt\_

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

Definition at line 115 of file Enemy.hpp.

#### 14.4.4.10 speed\_

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

Definition at line 111 of file Enemy.hpp.

#### 14.4.4.11 textures\_

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

Definition at line 110 of file Enemy.hpp.

#### 14.4.4.12 value\_

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

Definition at line 118 of file Enemy.hpp.

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

- src/entity/[Enemy.hpp](#)
- src/entity/[Enemy.cpp](#)

## 14.5 Entity Class Reference

Visible entity on the map.

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

Inheritance diagram for Entity:

Collaboration diagram for Entity:

### Public Member Functions

- [Entity](#) (int textureID, const [TextureHolder](#) &textures)
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const  
*draw this entity, structure according to SFML manual*
- virtual void [update](#) (sf::Time deltaTime, [World](#) &world)=0  
*Virtual function to update this [Entity](#), subclasses will implement.*
- sf::Vector2f [getPosition](#) () const  
*Get sprite position.*
- int [getTextureID](#) () const  
*Get textureID used for this [Entity](#).*

### Protected Attributes

- sf::Sprite [entitySprite](#)

### Private Attributes

- sf::Vector2f [mVelocity](#)
- int [textureID\\_](#)

### 14.5.1 Detailed Description

Visible entity on the map.

Definition at line 22 of file Entity.hpp.

### 14.5.2 Constructor & Destructor Documentation

#### 14.5.2.1 Entity()

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

**Parameters**

<i>textureID</i>	Texture number to be used for the <a href="#">Entity</a>
<i>textures</i>	Reference to a TextureHolder class instance that holds the texture

Definition at line 8 of file Entity.cpp.

```

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

**14.5.3 Member Function Documentation****14.5.3.1 draw()**

```

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

draw this entity, structure according to SFML manual

**Parameters**

<i>target</i>	where to draw this <a href="#">Entity</a>
<i>states</i>	RenderStates class to use for this drawable

Definition at line 13 of file Entity.cpp.

```

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

**14.5.3.2 getPosition()**

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

Get sprite position.

**Returns**

SFML 2D vector, the position

Definition at line 46 of file Entity.hpp.

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

### 14.5.3.3 getTextureID()

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

Get textureID used for this [Entity](#).

#### Returns

integer, the textureID (index in TextureHolder list)

Definition at line 51 of file Entity.hpp.

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

### 14.5.3.4 update()

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

Virtual function to update this [Entity](#), subclasses will implement.

#### Parameters

<i>deltaTime</i>	Time since last frame
<i>world</i>	Reference to a world class where to update

Implemented in [Tower](#), [Projectile](#), and [Enemy](#).

## 14.5.4 Member Data Documentation

### 14.5.4.1 entitySprite

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

Definition at line 54 of file Entity.hpp.

### 14.5.4.2 mVelocity

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

Definition at line 57 of file Entity.hpp.

#### 14.5.4.3 textureID\_

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

Definition at line 58 of file Entity.hpp.

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

- [src/entity/Entity.hpp](#)
- [src/entity/Entity.cpp](#)

## 14.6 Game Class Reference

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

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

Collaboration diagram for Game:

### Public Member Functions

- [Game](#) ()
- void [load](#) ()  
*load all resources to Resource class instances*
- void [render](#) ()  
*call's current [Scene](#) class instance's draw function to display everything on screen*
- void [processEvents](#) ()  
*takes player input from window and forwards is to current [Scene](#) class*
- void [handleSceneChange](#) ()  
*Changes the current [Scene](#) class instance (for example when loading a level)*
- void [Update](#) (sf::Time deltaTime)  
*Updates everything on screen that should move etc.*
- void [run](#) ()  
*Run the game.*

### Public Attributes

- [TextureHolder](#) textures
- [FontHolder](#) fonts
- [SoundBufferHolder](#) sounds

### Private Attributes

- sf::VideoMode [videomode](#)
- sf::RenderWindow [window](#)
- std::unique\_ptr< [Scene](#) > [scene](#)
- sf::Clock [clock](#)
- sf::RenderStates [renderStates](#)



## 14.6.1 Detailed Description

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

Definition at line 13 of file Game.hpp.

## 14.6.2 Constructor & Destructor Documentation

### 14.6.2.1 Game()

Game::Game ( )

Definition at line 16 of file Game.cpp.

```
16         : videomode(1280, 1024),
17           window(videomode, "Cats vs Rats TD"),
18           textures(),
19           renderStates(sf::RenderStates::Default) {
20     load();
21     scene = std::make_unique<GameTitle>(textures, fonts);
22     //scene = std::make_unique<GameEnd>(textures, fonts);
23 }
```

## 14.6.3 Member Function Documentation

### 14.6.3.1 handleSceneChange()

void Game::handleSceneChange ( )

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

Definition at line 98 of file Game.cpp.

```
98     {
99     if (scene->requestedScene().scene != scene->sceneType()) {
100        //std::cout << "Requested: " << scene->requestedScene().scene << " ,sceneType: " <<scene->sceneType <<
101        "\n";
102        auto request = scene->requestedScene();
103        switch (request.scene) {
104            case Scenes::ID::GameTitle:scene = std::make_unique<GameTitle>(textures, fonts);
105            break;
106            case Scenes::ID::MapScene:scene = std::make_unique<MapScene>(textures, fonts, sounds,
107            request.number);
108            break;
109            case Scenes::ID::LevelSelect:scene = std::make_unique<LevelSelect>(textures, fonts);
110            break;
111            case Scenes::ID::GameEnd:scene = std::make_unique<GameEnd>(textures, fonts);
112            break;
113            default:scene = std::make_unique<GameTitle>(textures, fonts);
114            break;
115        }
116    }
```

### 14.6.3.2 load()

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

load all resources to Resource class instances

Definition at line 25 of file Game.cpp.

```
25 {
26     //load textures and sounds at start of game, return false if failed
27     //set renderStates
28     renderStates.transform.scale(1, 1);
29     textures.load(Textures::PathTile, "src/media/textures/patchtile.png");
30     textures.load(Textures::GrassTile, "src/media/textures/grasstile.png");
31     textures.load(Textures::HouseTile, "src/media/textures/house.png");
32     textures.load(Textures::FatRat, "src/media/textures/fatrat.png");
33     textures.load(Textures::BasicRat, "src/media/textures/basicrat.png");
34     textures.load(Textures::FastRat, "src/media/textures/fastrat.png");
35     textures.load(Textures::GunCat, "src/media/textures/guncat.png");
36     textures.load(Textures::UpgradedGunCat, "src/media/textures/level2guncat.png");
37     textures.load(Textures::FreezeCat, "src/media/textures/freezecat.png");
38     textures.load(Textures::UpgradedFzezeCat, "src/media/textures/level2freezecat.png");
39     textures.load(Textures::BombCat, "src/media/textures/bombcat.png");
40     textures.load(Textures::UpgradedBombCat, "src/media/textures/level2bombcat.png");
41     textures.load(Textures::Bullet, "src/media/textures/bullet.png");
42     textures.load(Textures::Snowflake, "src/media/textures/snowflake.png");
43     textures.load(Textures::Bomb, "src/media/textures/bomb.png");
44     textures.load(Textures::Explosion, "src/media/textures/explosion_128.png");
45     textures.load(Textures::PlayButton, "src/media/textures/playbutton.png");
46     //fonts.load(Fonts::GameTitleFont, "src/media/fonts/SnackerComic_PerosnalUseOnly.ttf");
47     fonts.load(Fonts::GameTitleFont, "src/media/fonts/BalonkuRegular-lalw.otf");
48     sounds.load(SoundBuffers::EnemyDeath, "src/media/sounds/44429_468340-lq.wav");
49     sounds.load(SoundBuffers::Explosion, "src/media/sounds/explosion_sound.mp3");
50     sounds.load(SoundBuffers::GunCat, "src/media/sounds/gun_sound.mp3");
51     sounds.load(SoundBuffers::BombCatMeow, "src/media/sounds/sadmeow_speedup.mp3");
52     sounds.load(SoundBuffers::FreezeCatMeow, "src/media/sounds/freezecat.mp3");
53 }
```

### 14.6.3.3 processEvents()

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

takes player input from window and forwards is to current [Scene](#) class

Definition at line 65 of file Game.cpp.

```
65 {
66     //handle input from player
67     sf::Event event;
68     while (window.pollEvent(event)) {
69         scene->handleInput(event);
70         if (event.type == sf::Event::Closed)
71             window.close();
72     }
73 }
```

### 14.6.3.4 render()

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

call's current [Scene](#) class instance's draw function to display everything on screen

Definition at line 55 of file Game.cpp.

```
55 {
56     //render everything to the screen
57     //sf::CircleShape shape(50);
58     //shape.setFillColor(sf::Color(150, 50, 250));
59     //window.draw(shape);
60     window.clear(sf::Color(250, 250, 250)); //????????????????
61     window.draw(*scene, renderStates);
62     window.display();
63 }
```

### 14.6.3.5 run()

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

Run the game.

Definition at line 80 of file Game.cpp.

```
80     {
81         // Counts time between frames
82         sf::Time TimePerFrame = sf::seconds(0.0167); // 1/60=0.0167
83         sf::Time deltaTime = sf::Time::Zero;
84         // Initialize time to zero
85         while (window.isOpen()) {
86             handleSceneChange();
87             processEvents();
88             deltaTime += clock.restart();
89             while (deltaTime > TimePerFrame) {
90                 deltaTime -= TimePerFrame;
91                 processEvents();
92                 Update(deltaTime);
93             }
94             window.clear(sf::Color::White);
95             render();
96         }
97     }
```

### 14.6.3.6 Update()

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

Updates everything on screen that should move etc.

Parameters

<i>deltaTime</i>	time since last frame
------------------	-----------------------

Definition at line 75 of file Game.cpp.

```
75     {
76         //update elements of the game, positions etc.
77         scene->update(deltaTime);
78     }
```

## 14.6.4 Member Data Documentation

### 14.6.4.1 clock

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

Definition at line 57 of file Game.hpp.

#### 14.6.4.2 fonts

`FontHolder` `Game::fonts`

Instance of class `Resource`, which stores fonts

Definition at line 48 of file `Game.hpp`.

#### 14.6.4.3 renderStates

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

Definition at line 58 of file `Game.hpp`.

#### 14.6.4.4 scene

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

Definition at line 56 of file `Game.hpp`.

#### 14.6.4.5 sounds

`SoundBufferHolder` `Game::sounds`

Instance of class `Resource`, which stores sounds

Definition at line 52 of file `Game.hpp`.

#### 14.6.4.6 textures

`TextureHolder` `Game::textures`

Instance of class `Resource`, which stores textures

Definition at line 44 of file `Game.hpp`.

#### 14.6.4.7 videomode

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

Definition at line 54 of file Game.hpp.

#### 14.6.4.8 window

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

Definition at line 55 of file Game.hpp.

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

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

## 14.7 GameCommandsMenu Class Reference

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

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

Inheritance diagram for GameCommandsMenu:

Collaboration diagram for GameCommandsMenu:

### Public Member Functions

- [GameCommandsMenu](#) (const sf::Vector2f &textPosition, const sf::Vector2f &towerPosition, [FontHolder](#) &fonts, [World](#) &world)  
*Constructor.*
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override

### Private Attributes

- sf::Text [menuString\\_](#)
- sf::Text [towerIndexText\\_](#)
- std::vector< sf::Sprite > [towerSprites\\_](#)

#### 14.7.1 Detailed Description

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

Definition at line 17 of file GameCommandsMenu.hpp.

## 14.7.2 Constructor & Destructor Documentation

### 14.7.2.1 GameCommandsMenu()

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

Constructor.

Parameters

<i>textPosition</i>	Coordinates for the guide text
<i>towerPosition</i>	Coordinates for the <a href="#">Tower</a> sprites position
<i>fonts</i>	Reference to a FontHolder to get fonts for text
<i>world</i>	Reference to the current <a href="#">Game World</a>

Definition at line 7 of file GameCommandsMenu.cpp.

```
10 {
11     menuString_.setFont(fonts.get(Fonts::GameTitleFont));
12     menuString_.setString("Controls:\n"
13         "Left Click:\n"
14         "    Select location\n"
15         "Right Click:\n"
16         "    sell\n"
17         "Number 1, 2, 3:\n"
18         "    Buy tower type\n"
19         "Number 4:\n"
20         "    Upgrade tower\n"
21         "    (Price:500)");
22     menuString_.setCharacterSize(24);
23     menuString_.setFillColor(sf::Color::Black);
24     menuString_.setPosition(textPosition);
25     menuString_.setLineSpacing(0.8f);
26
27     towerIndexText_.setFont(fonts.get(Fonts::GameTitleFont));
28     towerIndexText_.setString("1      GunCat\n"
29         "      Price:300\n"
30         "2      IceCat\n"
31         "      Price:400\n"
32         "3      BombCat\n"
33         "      Price:500");
34     towerIndexText_.setCharacterSize(24);
35     towerIndexText_.setFillColor(sf::Color::Black);
36     towerIndexText_.setPosition(towerPosition);
37     towerIndexText_.move(-10.f, 2.f);
38     towerIndexText_.setLineSpacing(0.8f);
39     // load and set textures
40     sf::Sprite tower1;
41     tower1.setTexture(world.getTextures().get(Textures::GunCat));
42     sf::Sprite tower2;
43     tower2.setTexture(world.getTextures().get(Textures::FreezeCat));
44     sf::Sprite tower3;
45     tower3.setTexture(world.getTextures().get(Textures::BombCat));
46
47     towerSprites_.push_back(tower1);
48     towerSprites_.push_back(tower2);
49     towerSprites_.push_back(tower3);
50     int moveAmount = 0;
51
52     for (auto &tower : towerSprites_) {
53         tower.setPosition(towerPosition);
54         tower.move(0, tileSize * float(moveAmount));
55         moveAmount += 1;
56     }
57
58 }
```

## 14.7.3 Member Function Documentation

### 14.7.3.1 draw()

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

Definition at line 59 of file GameCommandsMenu.cpp.

```
59
60 target.draw(menuString_);
61 for (auto tower : towerSprites_) {
62     target.draw(tower);
63 }
64 target.draw(towerIndexText_);
65 }
```

## 14.7.4 Member Data Documentation

### 14.7.4.1 menuString\_

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

Definition at line 32 of file GameCommandsMenu.hpp.

### 14.7.4.2 towerIndexText\_

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

Definition at line 33 of file GameCommandsMenu.hpp.

### 14.7.4.3 towerSprites\_

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

Definition at line 34 of file GameCommandsMenu.hpp.

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

- [src/ui/GameCommandsMenu.hpp](#)
- [src/ui/GameCommandsMenu.cpp](#)

## 14.8 GameEnd Class Reference

A class that inherits [Scene](#), is shown when the player loses the game.

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

Inheritance diagram for GameEnd:

Collaboration diagram for GameEnd:

### Public Member Functions

- [GameEnd](#) ([TextureHolder](#) &textures, [FontHolder](#) &fonts)  
*Constructor for the class.*
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override  
*draw the scene to target*
- void [handleInput](#) (const sf::Event &event) override  
*handle input for current scene*
- void [handlePlayerInput](#) (sf::Keyboard::Key key, bool isPressed)  
*Takes input from handleInput function.*
- [sceneRequest requestedScene](#) () override  
*Next scene from this one.*
- void [update](#) (sf::Time dt) override  
*Update scene function.*
- [Scenes::ID sceneType](#) () override  
*Return current scene's type.*

### Public Attributes

- std::vector< sf::Text > [options](#)  
*Options to choose from in this scene.*
- std::size\_t [optionIndex](#)  
*What option is currently selected.*

### Private Attributes

- int [width](#) = 16
- int [height](#) = 16
- [TextureHolder](#) & textures\_
- [FontHolder](#) & fonts\_
- sf::Text [textGameOver\\_](#)
- sf::Text [textInstruction\\_](#)
- [sceneRequest](#) nextScene\_

#### 14.8.1 Detailed Description

A class that inherits [Scene](#), is shown when the player loses the game.

Definition at line 16 of file GameEnd.hpp.



## 14.8.2 Constructor & Destructor Documentation

### 14.8.2.1 GameEnd()

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

Constructor for the class.

#### Parameters

<i>textures</i>	Reference to a TextureHolder instance
<i>fonts</i>	Reference to a FontHolder instance

Definition at line 8 of file GameEnd.cpp.

```
9      : textures_(textures), fonts_(fonts) {
10      // help text
11      textGameOver_.setFont(fonts_.get(Fonts::GameTitleFont));
12      textGameOver_.setString("Game Over!");
13      textGameOver_.setCharacterSize(50);
14      textGameOver_.setFillColor(sf::Color::Black);
15      textGameOver_.setPosition(tileSize * float(width) / 3.f, tileSize * float(height) / 4.1f);
16      textInstruction_.setFont(fonts_.get(Fonts::GameTitleFont));
17      textInstruction_.setString("press Enter to go back to main menu");
18      textInstruction_.setCharacterSize(24);
19      textInstruction_.setFillColor(sf::Color::Black);
20      textInstruction_.setPosition(tileSize * float(width) / 3.f, tileSize * float(height) / 3.4f);
21
22      optionIndex = 0;
23      //option for enter and going back to main menu
24      sf::Text backToMenuOption;
25      backToMenuOption.setFont(fonts_.get(Fonts::GameTitleFont));
26      backToMenuOption.setString("Back To Menu");
27      backToMenuOption.setPosition(tileSize * float(width) / 2.f, tileSize * float(height) / 3.f);
28      backToMenuOption.setFillColor(sf::Color::Red); // optionIndex = 0;
29      backToMenuOption.setCharacterSize(60);
30      options.push_back(backToMenuOption);
31
32      // nextScene struct for requestedScene
33      nextScene_.scene = Scenes::ID::GameEnd;
34      nextScene_.number = 1;
35 }
```

## 14.8.3 Member Function Documentation

### 14.8.3.1 draw()

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

draw the scene to target

## Parameters

<i>target</i>	sf::RenderTarget to draw the scene to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 39 of file GameEnd.cpp.

```

39                                     {
40     for (const auto &text : options) {
41         target.draw(text);
42     }
43     target.draw(textGameOver_);
44     target.draw(textInstruction_);
45 }
```

### 14.8.3.2 handleInput()

```

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

handle input for current scene

## Parameters

<i>event</i>	sf::Event (keypress etc.)
--------------	---------------------------

Implements [Scene](#).

Definition at line 46 of file GameEnd.cpp.

```

46                                     {
47     switch (event.type) {
48         case sf::Event::KeyPressed:handlePlayerInput(event.key.code, true);
49         break;
50         case sf::Event::KeyReleased:handlePlayerInput(event.key.code, false);
51         break;
52         default:break;
53     }
54 }
```

### 14.8.3.3 handlePlayerInput()

```

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

Takes input from handleInput function.

## Parameters

<i>key</i>	keyboard key that was pressed
<i>isPressed</i>	Is the key pressed currently?

Definition at line 57 of file GameEnd.cpp.

```

57                                     {
58     if (isPressed) {
59         if (key == sf::Keyboard::Enter) {
60             nextScene_.scene = Scenes::ID::LevelSelect;
61             nextScene_.number = optionIndex + 1;
62         }
63     }
64 }
```

#### 14.8.3.4 requestedScene()

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

Next scene from this one.

##### Returns

the request

Implements [Scene](#).

Definition at line 36 of file GameEnd.cpp.

```

36                                     {
37     return nextScene_;
38 }
```

#### 14.8.3.5 sceneType()

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

Return current scene's type.

##### Returns

current scene's type, [GameEnd](#)

Implements [Scene](#).

Definition at line 55 of file GameEnd.hpp.

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

#### 14.8.3.6 update()

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

Update scene function.

#### Parameters

<i>dt</i>	deltatime, time since last frame update
-----------	-----------------------------------------

Implements [Scene](#).

Definition at line 55 of file GameEnd.cpp.

```
55 {}
```

### 14.8.4 Member Data Documentation

#### 14.8.4.1 fonts\_

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

Definition at line 68 of file GameEnd.hpp.

#### 14.8.4.2 height

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

Definition at line 66 of file GameEnd.hpp.

#### 14.8.4.3 nextScene\_

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

Definition at line 71 of file GameEnd.hpp.

#### 14.8.4.4 optionIndex

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

What option is currently selected.

Definition at line 63 of file GameEnd.hpp.

#### 14.8.4.5 options

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

Options to choose from in this scene.

Definition at line 59 of file GameEnd.hpp.

#### 14.8.4.6 textGameOver\_

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

Definition at line 69 of file GameEnd.hpp.

#### 14.8.4.7 textInstruction\_

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

Definition at line 70 of file GameEnd.hpp.

#### 14.8.4.8 textures\_

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

Definition at line 67 of file GameEnd.hpp.

#### 14.8.4.9 width

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

Definition at line 65 of file GameEnd.hpp.

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

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

## 14.9 GameStateMenu Class Reference

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

```
#include <GameStateMenu.hpp>
```

Inheritance diagram for GameStateMenu:

Collaboration diagram for GameStateMenu:

### Public Member Functions

- [GameStateMenu](#) (const sf::Vector2f &position, [FontHolder](#) &fonts)
- void [update](#) ([World](#) &world, int enemiesNotSpawned, int waveNumber)  
*Update The status menu numbers.*
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override  
*draw the status menu to screen*

### Private Attributes

- sf::Text [menuString\\_](#)

#### 14.9.1 Detailed Description

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

Definition at line 13 of file GameStateMenu.hpp.

#### 14.9.2 Constructor & Destructor Documentation

##### 14.9.2.1 GameStateMenu()

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

Definition at line 7 of file GameStateMenu.cpp.

```
7
8  menuString_.setFont (fonts.get (Fonts::GameTitleFont));
9  menuString_.setString ("GameStateMenu");
10 menuString_.setCharacterSize (24);
11 menuString_.setFillColor (sf::Color::Black);
12 menuString_.setPosition (position);
13 }
```

#### 14.9.3 Member Function Documentation

##### 14.9.3.1 draw()

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

draw the status menu to screen

## Parameters

<i>target</i>	sf::RenderTarget to draw the statusmenu to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 23 of file GameStatusMenu.cpp.

```

23                                     {
24     target.draw(menuString_);
25 }
```

### 14.9.3.2 update()

```

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

Update The status menu numbers.

## Parameters

<i>world</i>	
<i>enemiesNotSpawned</i>	enemies left in current wave that haven't spawned yet
<i>waveNumber</i>	current wave number

Definition at line 14 of file GameStatusMenu.cpp.

```

14                                     {
15     std::stringstream ss;
16     ss << "Wave Number: " << waveNumber << "\n"
17         << "Enemies Left: " << (world.getEnemies().size() + enemiesNotSpawned) << "\n"
18         << "Hit Points: " << world.getHP() << "\n"
19         << "Money: " << world.getMoney() << "\n";
20     menuString_.setString(ss.str());
21
22 }
```

## 14.9.4 Member Data Documentation

### 14.9.4.1 menuString\_

sf::Text GameStatusMenu::menuString\_ [private]

Definition at line 30 of file GameStatusMenu.hpp.

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

- [src/ui/GameStatusMenu.hpp](#)
- [src/ui/GameStatusMenu.cpp](#)

## 14.10 GameTitle Class Reference

Welcome screen for the game.

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

Inheritance diagram for GameTitle:

Collaboration diagram for GameTitle:

### Public Member Functions

- [GameTitle](#) ([TextureHolder](#) &textures, [FontHolder](#) &fonts)  
*Constructor for the [GameTitle](#).*
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override  
*draw the scene to target*
- void [handleInput](#) (const sf::Event &event) override  
*handle input for current scene*
- [sceneRequest requestedScene](#) () override  
*Next scene from this one.*
- void [update](#) (sf::Time dt) override  
*Update scene function.*
- [Scenes::ID sceneType](#) () override  
*Return current scene's type.*

### Private Attributes

- int [width](#) = 20
- int [height](#) = 16
- [TextureHolder](#) & [textures\\_](#)
- [FontHolder](#) & [fonts\\_](#)
- sf::Time [textEffectTime\\_](#)
- sf::Text [text\\_](#)
- sf::Text [text2\\_](#)
- bool [showText\\_](#)
- sf::Sprite [grassSprite\\_](#)
- sf::Sprite [pathSprite](#)
- sf::RenderTexture [mapTex\\_](#)
- sf::Sprite [mapSprite\\_](#)
- [sceneRequest](#) [nextScene\\_](#)

### Additional Inherited Members

#### 14.10.1 Detailed Description

Welcome screen for the game.

Definition at line 14 of file GameTitle.hpp.



## 14.10.2 Constructor & Destructor Documentation

### 14.10.2.1 GameTitle()

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

Constructor for the [GameTitle](#).

#### Parameters

<i>textures</i>	Reference to a Resource class holding textures
<i>fonts</i>	Reference to a Resource class holding fonts

Definition at line 10 of file GameTitle.cpp.

```
10                                     : textures_(textures), fonts_(fonts) {
11     showText_ = true;
12     // load text resources
13     text_.setFont(fonts_.get(Fonts::GameTitleFont));
14     //textGameOver_.setFont(font);
15     text_.setString("Cats vs Rats TD\n"
16                     "by: Antti, Henrik, Kasper and Otso");
17     text_.setCharacterSize(43);
18     text_.setFillColor(sf::Color::Black);
19     text_.setPosition(tileSize * float(width) / 6.f, tileSize * float(height) / 3.f);
20     text2_.setFont(fonts_.get(Fonts::GameTitleFont));
21     //textGameOver_.setFont(font);
22     text2_.setString("Press Enter");
23     text2_.setCharacterSize(43);
24     text2_.setFillColor(sf::Color::Red);
25     text2_.setPosition(tileSize * float(width) / 2.8f, tileSize * float(height) / 2.f);
26     // load texture resources
27     grassSprite_.setTexture(textures_.get(Textures::GrassTile));
28
29     // create holder for map sprite that consists of multiple
30     // small tiles
31     mapTex_.clear();
32     mapTex_.create(1280, 1024);
33     for (int i = 0; i < height; ++i) {
34         for (int j = 0; j < width; ++j) {
35             grassSprite_.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
36             mapTex_.draw(grassSprite_);
37         }
38     }
39     mapSprite_.setTexture(mapTex_.getTexture());
40     mapSprite_.setOrigin({0, mapSprite_.getLocalBounds().height});
41     mapSprite_.setScale({1, -1});
42     mapSprite_.setColor(sf::Color(255, 255, 255, 64));
43
44     //scene request
45     nextScene_.scene = Scenes::ID::GameTitle;
46     nextScene_.number = 1;
47 }
```

## 14.10.3 Member Function Documentation

### 14.10.3.1 draw()

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

draw the scene to target

#### Parameters

<i>target</i>	sf::RenderTarget to draw the scene to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 49 of file GameTitle.cpp.

```

49                                     {
50     // draw background
51     target.draw(mapSprite_);
52     //if (showText_) {
53     //     target.draw(text_);
54     //}
55     target.draw(text_);
56     target.draw(text2_);
57
58 }
```

### 14.10.3.2 handleInput()

```

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

handle input for current scene

#### Parameters

<i>event</i>	sf::Event (keypress etc.)
--------------	---------------------------

Implements [Scene](#).

Definition at line 59 of file GameTitle.cpp.

```

59                                     {
60     if (event.type == sf::Event::EventType::KeyPressed) {
61         if (event.key.code == sf::Keyboard::Enter) {
62             nextScene_.scene = Scenes::ID::LevelSelect;
63         }
64     }
65     //std::cout << "input";
66     //requestSceneChange(SceneChangeRequest { nextScene_ });
67 }
```

### 14.10.3.3 requestedScene()

```

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

Next scene from this one.

#### Returns

the request

Implements [Scene](#).

Definition at line 76 of file GameTitle.cpp.

```

76                                     {
77     return nextScene_;
78 }
```

### 14.10.3.4 sceneType()

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

Return current scene's type.

#### Returns

current scene's type, [GameTitle](#)

Implements [Scene](#).

Definition at line 47 of file GameTitle.hpp.

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

### 14.10.3.5 update()

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

Update scene function.

#### Parameters

<i>dt</i>	deltatime, time since last frame update
-----------	-----------------------------------------

Implements [Scene](#).

Definition at line 68 of file GameTitle.cpp.

```
68 {
69     textEffectTime_ += dt;
70     if (textEffectTime_ >= sf::seconds(0.1)) {
71         showText_ = !showText_;
72         text_.setFillColor(showText_ ? sf::Color::Black : sf::Color::Red);
73         textEffectTime_ = sf::Time::Zero;
74     }
75 }
```

## 14.10.4 Member Data Documentation

### 14.10.4.1 fonts\_

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

Definition at line 52 of file GameTitle.hpp.

#### 14.10.4.2 grassSprite\_

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

Definition at line 57 of file GameTitle.hpp.

#### 14.10.4.3 height

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

Definition at line 50 of file GameTitle.hpp.

#### 14.10.4.4 mapSprite\_

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

Definition at line 60 of file GameTitle.hpp.

#### 14.10.4.5 mapTex\_

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

Definition at line 59 of file GameTitle.hpp.

#### 14.10.4.6 nextScene\_

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

Definition at line 61 of file GameTitle.hpp.

#### 14.10.4.7 pathSprite

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

Definition at line 58 of file GameTitle.hpp.

#### 14.10.4.8 showText\_

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

Definition at line 56 of file GameTitle.hpp.

#### 14.10.4.9 text2\_

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

Definition at line 55 of file GameTitle.hpp.

#### 14.10.4.10 text\_

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

Definition at line 54 of file GameTitle.hpp.

#### 14.10.4.11 textEffectTime\_

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

Definition at line 53 of file GameTitle.hpp.

#### 14.10.4.12 textures\_

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

Definition at line 51 of file GameTitle.hpp.

#### 14.10.4.13 width

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

Definition at line 49 of file GameTitle.hpp.

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

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

## 14.11 Grid Class Reference

[Grid](#) is used to place the towers according to the visual tiles.

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

Inheritance diagram for Grid:

Collaboration diagram for Grid:

### Public Member Functions

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

Get selected [Sector](#). Used in [TowerMenu](#) class to change where to place a tower.

### Private Attributes

- int [width](#)
- int [height](#)
- [Sector](#) \* [selectedSector](#)
- std::vector< std::vector< int > > [blocks](#)

#### 14.11.1 Detailed Description

[Grid](#) is used to place the towers according to the visual tiles.

Definition at line 10 of file Grid.hpp.

#### 14.11.2 Constructor & Destructor Documentation

##### 14.11.2.1 Grid()

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

Definition at line 5 of file Grid.cpp.

```
5      :  
6      width(16),  
7      height(16),  
8      blocks() {  
9  };
```

#### 14.11.3 Member Function Documentation

### 14.11.3.1 draw()

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

Definition at line 11 of file Grid.cpp.

```
11 {
12 }
```

### 14.11.3.2 getSelectedSector()

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

Get selected [Sector](#). Used in [TowerMenu](#) class to change where to place a tower.

#### Returns

Definition at line 13 of file Grid.cpp.

```
13 {
14     return selectedSector;
15 }
```

## 14.11.4 Member Data Documentation

### 14.11.4.1 blocks

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

Definition at line 23 of file Grid.hpp.

### 14.11.4.2 height

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

Definition at line 21 of file Grid.hpp.

#### 14.11.4.3 selectedSector

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

Definition at line 22 of file Grid.hpp.

#### 14.11.4.4 width

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

Definition at line 20 of file Grid.hpp.

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

- [src/ui/Grid.hpp](#)
- [src/ui/Grid.cpp](#)

## 14.12 LevelSelect Class Reference

A class for level selection menu, inherits [Scene](#) class.

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

Inheritance diagram for LevelSelect:

Collaboration diagram for LevelSelect:

### Public Member Functions

- [LevelSelect](#) ([TextureHolder](#) &textures, [FontHolder](#) &fonts)  
*Constructor for the class.*
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override  
*draw the scene to target*
- void [handleInput](#) (const sf::Event &event) override  
*handle input for current scene*
- void [handlePlayerInput](#) (sf::Keyboard::Key key, bool isPressed)  
*Takes input from handleInput function.*
- [sceneRequest requestedScene](#) () override  
*Next scene from this one.*
- void [update](#) (sf::Time dt) override  
*Update scene function.*
- void [updateOptionText](#) ()  
*Update, which option is selected (and so should be colored red)*
- [Scenes::ID sceneType](#) () override  
*Return current scene's type.*



## Public Attributes

- `std::vector< sf::Text > options`  
*Options to choose from in this scene, so maps.*
- `std::size_t optionIndex`  
*What option is currently selected.*

## Private Attributes

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

### 14.12.1 Detailed Description

A class for level selection menu, inherits [Scene](#) class.

Definition at line 18 of file `LevelSelect.hpp`.

### 14.12.2 Constructor & Destructor Documentation

#### 14.12.2.1 LevelSelect()

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

Constructor for the class.

#### Parameters

<i>textures</i>	Reference to a TextureHolder instance
<i>fonts</i>	Reference to a FontHolder instance

Definition at line 7 of file `LevelSelect.cpp`.

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

```

11     std::string mapfile = "src/media/maps/map";
12     mapfile.append(std::to_string(i + 1));
13     mapfile.append(".txt");
14     std::ifstream infile(mapfile);
15     std::string line; //line to store a line of the text file
16     std::getline(infile, line); //first line, map name
17     maps.push_back(line);
18     std::getline(infile, line); //second line, difficulty
19     maps[i] = maps[i] + " (" + line + ")";
20     infile.close();
21 }
22 showText_ = true;
23 optionIndex = 0;
24 // load text resources
25 text_.setFont(fonts_.get(Fonts::GameTitleFont));
26 //textGameOver_.setFont(font);
27 text_.setString("Use Up and Down keys to navigate,\n"
28               "choose Map and press Enter");
29 text_.setCharacterSize(30);
30 text_.setFillColor(sf::Color::Black);
31 text_.setPosition(tileSize * float(width) / 3.8f, tileSize * float(height) / 5.f);
32 // load texture resources
33 grassSprite_.setTexture(textures_.get(Textures::GrassTile));
34
35 // create holder for map sprite that consists of multiple
36 // small tiles
37 mapTex_.clear();
38 mapTex_.create(1280, 1024);
39 for (int i = 0; i < height; ++i) {
40     for (int j = 0; j < width; ++j) {
41         grassSprite_.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
42         mapTex_.draw(grassSprite_);
43     }
44 }
45 mapSprite_.setTexture(mapTex_.getTexture());
46 mapSprite_.setOrigin({0, mapSprite_.getLocalBounds().height});
47 mapSprite_.setScale({1, -1});
48 mapSprite_.setColor(sf::Color(255, 255, 255, 128)); // opacity 50%
49
50 // initialize menu options
51 sf::Text mapOption1;
52 mapOption1.setFont(fonts_.get(Fonts::GameTitleFont));
53 mapOption1.setString(maps[0]);
54 mapOption1.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f);
55 mapOption1.setFillColor(sf::Color::Red); // optionIndex = 0;
56 mapOption1.setCharacterSize(60);
57 sf::Text mapOption2;
58 mapOption2.setFont(fonts_.get(Fonts::GameTitleFont));
59 mapOption2.setString(maps[1]);
60 mapOption2.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 100);
61 mapOption2.setFillColor(sf::Color::Black);
62 mapOption2.setCharacterSize(60);
63 sf::Text mapOption3;
64 mapOption3.setFont(fonts_.get(Fonts::GameTitleFont));
65 mapOption3.setString(maps[2]);
66 mapOption3.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 200);
67 mapOption3.setFillColor(sf::Color::Black);
68 mapOption3.setCharacterSize(60);
69 sf::Text mapOption4;
70 mapOption4.setFont(fonts_.get(Fonts::GameTitleFont));
71 mapOption4.setString(maps[3]);
72 mapOption4.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 300);
73 mapOption4.setFillColor(sf::Color::Black);
74 mapOption4.setCharacterSize(60);
75 sf::Text mapOption5;
76 mapOption5.setFont(fonts_.get(Fonts::GameTitleFont));
77 mapOption5.setString(maps[4]);
78 mapOption5.setPosition(tileSize * float(width) / 3.5f, tileSize * float(height) / 3.5f + 400);
79 mapOption5.setFillColor(sf::Color::Black);
80 mapOption5.setCharacterSize(60);
81
82 options.push_back(mapOption1);
83 options.push_back(mapOption2);
84 options.push_back(mapOption3);
85 options.push_back(mapOption4);
86 options.push_back(mapOption5);
87
88
89 //sf::Text
90 // next scene:
91 // sceneType.scene = Scenes::ID::LevelSelect;
92 // sceneType.number = 0;
93 nextScene_.scene = Scenes::ID::LevelSelect;
94 nextScene_.number = 1;
95 }

```

### 14.12.3 Member Function Documentation

#### 14.12.3.1 draw()

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

draw the scene to target

##### Parameters

<i>target</i>	sf::RenderTarget to draw the scene to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 99 of file LevelSelect.cpp.

```
99
100     target.draw(mapSprite_);
101     for (const auto &text : options) {
102         target.draw(text);
103     }
104     target.draw(text_);
105 }
```

#### 14.12.3.2 handleInput()

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

handle input for current scene

##### Parameters

<i>event</i>	sf::Event (keypress etc.)
--------------	---------------------------

Implements [Scene](#).

Definition at line 106 of file LevelSelect.cpp.

```
106
107     switch (event.type) {
108         case sf::Event::KeyPressed:handlePlayerInput (event.key.code, true);
109             break;
110         case sf::Event::KeyReleased:handlePlayerInput (event.key.code, false);
111             break;
112         default:break;
113         //case sf::Event::MouseButtonPressed:handlePlayerInput (event.mouseButton, true);
114     }
115 }
```

### 14.12.3.3 handlePlayerInput()

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

Takes input from handleInput function.

#### Parameters

<i>key</i>	keyboard key that was pressed
<i>isPressed</i>	Is the key pressed currently?

Definition at line 134 of file LevelSelect.cpp.

```
134 {
135     //std::cout << "input: " << key << ", isPressed: " << isPressed << "\n";
136     if (isPressed) {
137         //std::cout << "START: index is: " << optionIndex << ", key is " << key << "\n";
138         if (key == sf::Keyboard::Up) {
139             if (optionIndex > 0)
140                 optionIndex--;
141             else
142                 optionIndex = options.size() - 1;
143             updateOptionText();
144         } else if (key == sf::Keyboard::Down) {
145             if (optionIndex < options.size() - 1)
146                 optionIndex++;
147             else
148                 optionIndex = 0;
149             updateOptionText();
150         } else if (key == sf::Keyboard::Enter) {
151             nextScene_.scene = Scenes::ID::MapScene;
152             nextScene_.number = optionIndex + 1;
153             //updateOptionText();
154         }
155         //std::cout << "END: index is: " << optionIndex << "key is" << key << "\n";
156     }
157
158     // switch (event.type) {
159     //     case sf::Event::MouseButtonPressed:
160     //         //handlePlayerInput(event.mouseButton, false);
161     //         break;
162     //     case sf::Event::MouseButtonReleased:
163     //         //handlePlayerInput(event.mouseButton, false);
164     //         break;
165     // case sf::Event::MouseMove:handlePlayerInput(event.)
166     //     case sf::Event::Closed:
167     //         window.close();
168     //         break;
169     // }
170 }
```

### 14.12.3.4 requestedScene()

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

Next scene from this one.

#### Returns

the request

Implements [Scene](#).

Definition at line 96 of file LevelSelect.cpp.

```
96 {
97     return nextScene_;
98 }
```

### 14.12.3.5 sceneType()

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

Return current scene's type.

#### Returns

current scene's type, [GameEnd](#)

Implements [Scene](#).

Definition at line 61 of file LevelSelect.hpp.

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

### 14.12.3.6 update()

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

Update scene function.

#### Parameters

<i>dt</i>	deltatime, time since last frame update
-----------	-----------------------------------------

Implements [Scene](#).

Definition at line 116 of file LevelSelect.cpp.

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

### 14.12.3.7 updateOptionText()

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

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

Definition at line 124 of file LevelSelect.cpp.

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

## 14.12.4 Member Data Documentation

### 14.12.4.1 fonts\_

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

Definition at line 74 of file LevelSelect.hpp.

### 14.12.4.2 grassSprite\_

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

Definition at line 78 of file LevelSelect.hpp.

### 14.12.4.3 height

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

Definition at line 72 of file LevelSelect.hpp.

### 14.12.4.4 mapSprite\_

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

Definition at line 81 of file LevelSelect.hpp.

### 14.12.4.5 mapTex\_

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

Definition at line 80 of file LevelSelect.hpp.

#### 14.12.4.6 nextScene\_

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

Definition at line 82 of file LevelSelect.hpp.

#### 14.12.4.7 optionIndex

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

What option is currently selected.

Definition at line 69 of file LevelSelect.hpp.

#### 14.12.4.8 options

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

Options to choose from in this scene, so maps.

Definition at line 65 of file LevelSelect.hpp.

#### 14.12.4.9 pathSprite

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

Definition at line 79 of file LevelSelect.hpp.

#### 14.12.4.10 showText\_

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

Definition at line 77 of file LevelSelect.hpp.

#### 14.12.4.11 text\_

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

Definition at line 76 of file LevelSelect.hpp.

#### 14.12.4.12 `textEffectTime_`

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

Definition at line 75 of file `LevelSelect.hpp`.

#### 14.12.4.13 `textures_`

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

Definition at line 73 of file `LevelSelect.hpp`.

#### 14.12.4.14 `width`

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

Definition at line 71 of file `LevelSelect.hpp`.

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

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

## 14.13 Map Class Reference

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

### Private Attributes

- `std::string` [name](#)
- `std::string` [diff](#)

#### 14.13.1 Detailed Description

Definition at line 8 of file `Map.hpp`.

#### 14.13.2 Member Data Documentation



**14.13.2.1 diff**

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

Definition at line 12 of file Map.hpp.

**14.13.2.2 name**

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

Definition at line 11 of file Map.hpp.

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

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

**14.14 MapGrid Class Reference**

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

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

Inheritance diagram for MapGrid:

Collaboration diagram for MapGrid:

**Public Member Functions**

- [MapGrid](#) ([TextureHolder](#) &textureholder, int mapNum, std::map< int, std::pair< int, int >> &pathMarkers)  
*Constructor for the class.*
- int [getBlockAt](#) (int x, int y) const  
*Find out if a block is road or not.*
- void [setBlockRow](#) (std::vector< int > &row)  
*set a row of map blocks. Used by the constructor when reading a map*
- virtual void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override  
*draw the scene to target*
- int [getWidth](#) () const  
*Get map width.*
- int [getHeight](#) () const  
*Get map height.*

## Private Attributes

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

### 14.14.1 Detailed Description

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

Definition at line 17 of file MapGrid.hpp.

### 14.14.2 Constructor & Destructor Documentation

#### 14.14.2.1 MapGrid()

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

Constructor for the class.

#### Parameters

<i>textureholder</i>	Reference to Resource class instance that holds textures for the map
<i>mapNum</i>	Which map to load from file (file=map(mapNum).txt)
<i>pathMarkers</i>	<a href="#">Map</a> that will hold the enemy pathMarkers for this map

Definition at line 15 of file MapGrid.cpp.

```
15                                     :
16     textures_(
17     textureholder), mapNum_(mapNum), pathMarkers_(pathMarkers) {
18     std::string mapfile = "src/media/maps/map";
19     mapfile.append(std::to_string(mapNum));
20     mapfile.append(".txt");
21     std::ifstream infile(mapfile);
22     std::string line; //line to store a line of the text file
23     std::getline(infile, line); //first line, map name
24     this->name = line;
25     std::getline(infile, line); //second line, difficulty
26     this->diff = line;
27     std::getline(infile, line); //third line, map width
28     this->width = std::stoi(line);
29     std::getline(infile, line); //fourth line, map height
30     this->height = std::stoi(line);
```

```

30 //read the map
31 for (int i = 0; i < height; ++i) {
32     std::getline(infile, line);
33     std::vector<int> row;
34     int offset = 0;
35     for (int j = 0; j < line.length() + 1; ++j) {
36         if (line[j] == '+') {
37             row.emplace_back(0);
38         } else if (line[j] == '(') {
39             row.emplace_back(1);
40             std::string it;
41             j++;
42             while (line[j] != ')') {
43                 it += line[j];
44                 j++;
45                 offset++;
46             }
47             offset++;
48             int key = std::stoi(it);
49             pathMarkers_.insert(std::make_pair(key, std::make_pair(j - offset, i)));
50         } else if (line[j] == '-') {
51             row.emplace_back(1);
52         }
53     }
54     setBlockRow(row);
55 }
56 infile.close();
57 /*
58 for (int i = 0; i < pathMarkers.size(); i++) {
59     std::cout << i + 1 << ": " << pathMarkers.at(i + 1).first << ", " << pathMarkers.at(i + 1).second <<
60     std::endl;
61 }
62 */
63 mapTex_.clear();
64 mapTex_.create(1280, 1024);
65 sf::Sprite grassSprite;
66 grassSprite.setTexture(textures_.get(Textures::GrassTile));
67 sf::Sprite pathSprite;
68 pathSprite.setTexture(textures_.get(Textures::PathTile));
69 for (int i = 0; i < height; ++i) {
70     for (int j = 0; j < width; ++j) {
71         if (getBlockAt(j, i) == 0) {
72             grassSprite.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
73             mapTex_.draw(grassSprite);
74         } else if (getBlockAt(j, i) == 1) {
75             pathSprite.setPosition(static_cast<float>(tileSize * j), static_cast<float>(tileSize * i));
76             mapTex_.draw(pathSprite);
77         }
78     }
79 }
80 mapSprite_.setTexture(mapTex_.getTexture());
81 mapSprite_.setOrigin({0, mapSprite_.getLocalBounds().height});
82 mapSprite_.setScale({1, -1});
83 }

```

## 14.14.3 Member Function Documentation

### 14.14.3.1 draw()

```

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

```

draw the scene to target

#### Parameters

<i>target</i>	sf::RenderTarget to draw the scene to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 12 of file MapGrid.cpp.

```
12
13     target.draw(mapSprite_);
14 }
```

#### 14.14.3.2 getBlockAt()

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

Find out if a block is road or not.

##### Parameters

<i>x</i>	x-coordinate of the block (in blocks)
<i>y</i>	y-coordinate of the block (in blocks)

##### Returns

0=grass,1=road

Definition at line 32 of file MapGrid.hpp.

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

#### 14.14.3.3 getHeight()

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

Get map height.

##### Returns

map height

Definition at line 53 of file MapGrid.hpp.

```
53 { return height; }
```

#### 14.14.3.4 getWidth()

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

Get map width.

##### Returns

map width

Definition at line 48 of file MapGrid.hpp.

```
48 { return width; }
```

#### 14.14.3.5 setBlockRow()

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

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

##### Parameters

<i>row</i>	row number to change
------------	----------------------

Definition at line 37 of file MapGrid.hpp.

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

### 14.14.4 Member Data Documentation

#### 14.14.4.1 diff

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

Definition at line 57 of file MapGrid.hpp.

#### 14.14.4.2 height

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

Definition at line 56 of file MapGrid.hpp.

#### 14.14.4.3 map

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

Definition at line 59 of file MapGrid.hpp.

#### 14.14.4.4 mapNum\_

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

Definition at line 64 of file MapGrid.hpp.

#### 14.14.4.5 mapSprite\_

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

Definition at line 63 of file MapGrid.hpp.

#### 14.14.4.6 mapTex\_

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

Definition at line 62 of file MapGrid.hpp.

#### 14.14.4.7 name

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

Definition at line 58 of file MapGrid.hpp.

#### 14.14.4.8 pathMarkers\_

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

Definition at line 60 of file MapGrid.hpp.

#### 14.14.4.9 textures\_

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

Definition at line 61 of file MapGrid.hpp.

#### 14.14.4.10 width

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

Definition at line 55 of file MapGrid.hpp.

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

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

## 14.15 MapScene Class Reference

A class used when ingame. Inherits from [Scene](#) class. The main scene of the game.

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

Inheritance diagram for MapScene:

Collaboration diagram for MapScene:

### Public Member Functions

- [MapScene](#) ([TextureHolder](#) &textures, [FontHolder](#) &fonts, [SoundBufferHolder](#) &soundBuffers, int mapNum)  
*Constructor for the class.*
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override  
*draw the scene to target*
- void [handleInput](#) (const sf::Event &event) override  
*handle input for current scene*
- void [update](#) (sf::Time delta) override  
*Update scene function.*
- [sceneRequest requestedScene](#) () override  
*Next scene from this one (GameOverScene)*
- [Scenes::ID sceneType](#) () override  
*Return current scene's type.*

### Public Attributes

- int [mapNum\\_](#)  
*Current Map number.*
- std::map< int, std::pair< int, int > > [pathMarkers](#)  
*A map that contains enemy pathMarkers for current level (coordinates enemies go through)*
- [TowerMenu towerMenu\\_](#)  
*The towermenu class that is rendered with maps.*

### Private Attributes

- [World world\\_](#)
- [TextureHolder](#) & textures\_
- [SoundBufferHolder](#) & soundBuffers\_
- [FontHolder](#) & fonts\_
- [WaveController](#) waveController\_
- sf::Text [fpsText](#)
- int [width](#) = 16
- int [height](#) = 16
- [GameStatusMenu](#) statusMenu\_
- [GameCommandsMenu](#) commandsMenu\_
- [WaveStart](#) waveStart\_
- [WavePause](#) wavePause\_
- [request](#) nextScene\_

### 14.15.1 Detailed Description

A class used when ingame. Inherits from [Scene](#) class. The main scene of the game.

Definition at line 19 of file MapScene.hpp.

### 14.15.2 Constructor & Destructor Documentation

#### 14.15.2.1 MapScene()

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

Constructor for the class.

##### Parameters

<i>textures</i>	Reference to a TextureHolder instance
<i>fonts</i>	Reference to a FontHolder instance
<i>soundBuffers</i>	Reference to a SoundBufferHolder instance
<i>mapNum</i>	<a href="#">Map</a> number

Definition at line 9 of file MapScene.cpp.

```
10 : textures_(textures), soundBuffers_(soundBuffers), world_(textures, soundBuffers, mapNum,
    pathMarkers),
11 mapNum_(mapNum), waveController_(textures, pathMarkers), fonts_(fonts), towerMenu_(world_),
12 statusMenu_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 60.f), fonts),
13 commandsMenu_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 2.2f),
14 sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 1.35f),
15 fonts, world_),
16 waveStart_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 5.f), fonts),
17 wavePause_(sf::Vector2f(tileSize * float(width) / 0.98f, tileSize * float(height) / 2.9f), fonts) {
18 //world_ = World(textures_, soundBuffers, mapNum, pathMarkers);
19 fpsText.setFont(fonts_.get(Fonts::GameTitleFont));
20 fpsText.setString("updates / second");
21 fpsText.setCharacterSize(30);
22 fpsText.setFillColor(sf::Color::Black);
23 fpsText.setPosition(tileSize * float(width) / 30.f, tileSize * float(height) / 60.f);
24 // initialize scene request struct
25 nextScene_.scene = Scenes::ID::MapScene;
26 nextScene_.number = 1;
27 }
```

### 14.15.3 Member Function Documentation

#### 14.15.3.1 draw()

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



draw the scene to target

#### Parameters

<i>target</i>	sf::RenderTarget to draw the scene to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 29 of file MapScene.cpp.

```

29
30 world_.draw(target, states);
31 //target.draw(fpsText);
32 towerMenu_.draw(target, states);
33 statusMenu_.draw(target, states);
34 commandsMenu_.draw(target, states);
35 waveStart_.draw(target, states);
36 wavePause_.draw(target, states);
37 }
```

### 14.15.3.2 handleInput()

```

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

handle input for current scene

#### Parameters

<i>event</i>	sf::Event (keypress etc.)
--------------	---------------------------

Implements [Scene](#).

Definition at line 38 of file MapScene.cpp.

```

38
39 //towerMenu_.handleInput(event, world_);
40 towerMenu_.handleInput(event, world_);
41 waveStart_.handleInput(event, world_);
42 wavePause_.handleInput(event, world_);
43 }
```

### 14.15.3.3 requestedScene()

```

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

Next scene from this one (GameOverScene)

#### Returns

the request

Implements [Scene](#).

Definition at line 56 of file MapScene.cpp.

```

56
57 return nextScene_;
58 }
```

### 14.15.3.4 sceneType()

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

Return current scene's type.

#### Returns

current scene's type, [MapScene](#)

Implements [Scene](#).

Definition at line 54 of file MapScene.hpp.

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

### 14.15.3.5 update()

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

Update scene function.

#### Parameters

<i>delta</i>	deltatime, time since last frame update
--------------	-----------------------------------------

Implements [Scene](#).

Definition at line 44 of file MapScene.cpp.

```
44 {
45     world_.update(deltaTime);
46     if (!world_.paused) waveController_.update(deltaTime, world_);
47     double fps = 1.0 / deltaTime.asSeconds();
48     fpsText.setString(std::to_string(fps));
49     statusMenu_.update(world_, waveController_.getWaveEnemiesLeft(), waveController_.getWaveNumber());
50     waveStart_.update(world_, waveController_.getWaveEnemiesLeft());
51     // check if game is over
52     if (world_.getHP() <= 0) {
53         nextScene_.scene = Scenes::ID::GameEnd;
54     }
55 }
```

## 14.15.4 Member Data Documentation

### 14.15.4.1 commandsMenu\_

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

Definition at line 77 of file MapScene.hpp.

#### 14.15.4.2 fonts\_

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

Definition at line 71 of file MapScene.hpp.

#### 14.15.4.3 fpsText

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

Definition at line 73 of file MapScene.hpp.

#### 14.15.4.4 height

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

Definition at line 75 of file MapScene.hpp.

#### 14.15.4.5 mapNum\_

```
int MapScene::mapNum_
```

Current [Map](#) number.

Definition at line 58 of file MapScene.hpp.

#### 14.15.4.6 nextScene\_

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

Definition at line 80 of file MapScene.hpp.

#### 14.15.4.7 pathMarkers

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

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

Definition at line 62 of file MapScene.hpp.

#### 14.15.4.8 soundBuffers\_

`SoundBufferHolder& MapScene::soundBuffers_ [private]`

Definition at line 70 of file MapScene.hpp.

#### 14.15.4.9 statusMenu\_

`GameStatusMenu MapScene::statusMenu_ [private]`

Definition at line 76 of file MapScene.hpp.

#### 14.15.4.10 textures\_

`TextureHolder& MapScene::textures_ [private]`

Definition at line 69 of file MapScene.hpp.

#### 14.15.4.11 towerMenu\_

`TowerMenu MapScene::towerMenu_`

The towermenu class that is rendered with maps.

Definition at line 66 of file MapScene.hpp.

#### 14.15.4.12 waveController\_

`WaveController MapScene::waveController_ [private]`

Definition at line 72 of file MapScene.hpp.

#### 14.15.4.13 wavePause\_

`WavePause MapScene::wavePause_ [private]`

Definition at line 79 of file MapScene.hpp.

#### 14.15.4.14 waveStart\_

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

Definition at line 78 of file MapScene.hpp.

#### 14.15.4.15 width

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

Definition at line 74 of file MapScene.hpp.

#### 14.15.4.16 world\_

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

Definition at line 68 of file MapScene.hpp.

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

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

## 14.16 Price Class Reference

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

Inheritance diagram for Price:

Collaboration diagram for Price:

### Public Member Functions

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

### Static Private Member Functions

- static std::string [valueToString](#) (int value)

### Private Attributes

- sf::Text [price](#)

### 14.16.1 Detailed Description

Definition at line 5 of file Price.hpp.

### 14.16.2 Constructor & Destructor Documentation

#### 14.16.2.1 Price()

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

Definition at line 6 of file Price.cpp.

```
6                                     {
7     price.setFillColor(PriceColor);
8     price.setCharacterSize(fontSize);
9     price.setString(valueToString(value));
10    price.setPosition(position);
11    //have to set font
12 }
```

### 14.16.3 Member Function Documentation

#### 14.16.3.1 draw()

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

Definition at line 14 of file Price.cpp.

```
14                                     {
15 }
```

#### 14.16.3.2 valueToString()

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

Definition at line 17 of file Price.cpp.

```
17                                     {
18     return ((value >= 0) ? "+" : "-") + std::to_string(abs(value));
19 }
```

### 14.16.4 Member Data Documentation

### 14.16.4.1 price

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

Definition at line 10 of file Price.hpp.

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

- [src/ui/Price.hpp](#)
- [src/ui/Price.cpp](#)

## 14.17 Projectile Class Reference

Extends [Entity](#) class, Bombs, Bullets etc.

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

Inheritance diagram for Projectile:

Collaboration diagram for Projectile:

### Public Member Functions

- [Projectile](#) ([TextureHolder](#) &textures, int textureID, [ProjectileType](#) type, [Sector](#) start, const std::pair< float, std::shared\_ptr< [Enemy](#) >> &closestEnemy, const std::map< int, std::pair< float, std::shared\_ptr< [Enemy](#) >>> &closestEnemies, std::map< int, std::pair< float, std::shared\_ptr< [Enemy](#) >>> neighbourEnemies, float range, int damage, float velocity)
- void [update](#) (sf::Time deltaTime, [World](#) &world)  
*update projectile in [World](#)*
- bool [ifShouldRemove](#) () const  
*Should the projectile be removed, has it hit something?*

### Static Public Member Functions

- static [ProjectilePtr](#) [make](#) ([TextureHolder](#) &textures, int textureID, [ProjectileType](#) type, [Sector](#) start, const std::pair< float, std::shared\_ptr< [Enemy](#) >> &closestEnemy, const std::map< int, std::pair< float, std::shared\_ptr< [Enemy](#) >>> &closestEnemies, const std::map< int, std::pair< float, std::shared\_ptr< [Enemy](#) >>> &neighbourEnemies, float range, int damage, float velocity)  
*make new projectile*

### Public Attributes

- [ProjectileType](#) type\_  
*type of this projectile*

## Private Attributes

- [Sector start\\_](#)
- [Sector finish\\_](#)
- `std::pair< float, std::shared_ptr< Enemy > > closestEnemy\_`
- `std::map< int, std::pair< float, std::shared_ptr< Enemy > > > closestEnemies\_`
- `std::map< int, std::pair< float, std::shared_ptr< Enemy > > > neighbourEnemies\_`
- [TextureHolder](#) & [textures\\_](#)
- `float range\_`
- `int damage\_`
- `float velocity\_`
- `float distanceCovered`
- `float distance`
- `bool atFinish`

## Additional Inherited Members

### 14.17.1 Detailed Description

Extends [Entity](#) class, Bombs, Bullets etc.

Definition at line 31 of file Projectile.hpp.

### 14.17.2 Constructor & Destructor Documentation

#### 14.17.2.1 Projectile()

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

#### Parameters

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



Definition at line 5 of file Projectile.cpp.

5

```

        :
6     Entity(textureID, textures),
7     textures_(textures),
8     type_(type),
9     start_(start),
10    closestEnemy_(closestEnemy),
11    neighbourEnemies_(std::move(neighbourEnemies)),
12    range_(range),
13    damage_(damage),
14    velocity_(velocity),
15    distanceCovered(0.f),
16    distance(closestEnemy.first),
17    atFinish(false)
18 {
19
20     for (auto const& [key, value] : closestEnemies) {
21         std::shared_ptr pTemp (value.second);
22         std::pair<int, std::pair<float, std::shared_ptr<Enemy>>> p = {key, {value.first, pTemp}};
23         closestEnemies_.insert(p);
24     }
25     for (auto const& [key, value] : neighbourEnemies_) {
26         std::shared_ptr pTemp (value.second);
27         std::pair<int, std::pair<float, std::shared_ptr<Enemy>>> p = {key, {value.first, pTemp}};
28         neighbourEnemies_.insert(p);
29     }
30
31     if (type == ProjectileType::Bomb) {
32         entitySprite.setPosition((float)start_.x , (float)start_.y);
33     }
34     else {
35         entitySprite.setPosition((float)start_.x + 32.f , (float)start_.y + 32.f);
36     }
37 }
38 }
```

### 14.17.3 Member Function Documentation

#### 14.17.3.1 ifShouldRemove()

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

Should the projectile be removed, has it hit something?

#### Returns

Whether the projectile should be removed, true is should

Definition at line 72 of file Projectile.hpp.

```
72 { return atFinish; }
```

### 14.17.3.2 make()

```

ProjectilePtr Projectile::make (
    TextureHolder & textures,
    int textureID,
    ProjectileType type,
    Sector start,
    const std::pair< float, std::shared_ptr< Enemy >> & closestEnemy,
    const std::map< int, std::pair< float, std::shared_ptr< Enemy >>> & closest←
Enemies,

    const std::map< int, std::pair< float, std::shared_ptr< Enemy >>> & neighbour←
Enemies,

    float range,
    int damage,
    float velocity ) [static]

```

make new projectile

#### Parameters

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

#### Returns

unique\_ptr to the [Projectile](#)

Definition at line 137 of file Projectile.cpp.

```

146         {
147     std::map<int, std::pair<float, std::shared_ptr<Enemy>> closestEnemiesTemp;
148     for (auto const& [key, value] : neighbourEnemies) {
149         std::shared_ptr pTemp (value.second);
150         std::pair <int, std::pair<float, std::shared_ptr<Enemy>> p = {key, {value.first, pTemp}};
151         closestEnemiesTemp.insert(p);
152     }
153     std::map<int, std::pair<float, std::shared_ptr<Enemy>> neighbourEnemiesTemp;
154     for (auto const& [key, value] : neighbourEnemies) {
155         std::shared_ptr pTemp (value.second);
156         std::pair <int, std::pair<float, std::shared_ptr<Enemy>> p = {key, {value.first, pTemp}};
157         neighbourEnemiesTemp.insert(p);
158     }
159     return std::make_unique<Projectile>(textures, textureID, type, start, closestEnemy,
        closestEnemiesTemp, neighbourEnemiesTemp, range, damage, velocity);
160 }

```

### 14.17.3.3 update()

```

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

```

update projectile in [World](#)

## Parameters

<i>deltaTime</i>	time since last frame
<i>world</i>	reference to <a href="#">World</a> class where the <a href="#">Projectile</a> is

Implements [Entity](#).

Definition at line 40 of file Projectile.cpp.

```

40
41 //closestEnemies_ = getClosestEnemies(world);
42 if(closestEnemy_.second==nullptr) {return;}
43 float enemyX = closestEnemy_.second->getPosition().x;
44 float enemyY = closestEnemy_.second->getPosition().y;
45 //std::shared_ptr<Enemy> freezeEnemy=nullptr;
46 std::shared_ptr<Enemy> freezeEnemy (closestEnemy_.second);
47 if (type_ == ProjectileType::FreezeGun) {
48     for (int i = 0; i < closestEnemies_.size(); i++) {
49         if (closestEnemies_[i].second!=nullptr && closestEnemies_[i].first <= range_) {
50             if (closestEnemies_[i].second->isNotFrozen()) {
51                 enemyX = closestEnemies_[i].second->getPosition().x;
52                 enemyY = closestEnemies_[i].second->getPosition().y;
53                 freezeEnemy = closestEnemies_[i].second;
54                 break;
55             }
56         }
57     }
58 }
59
60 auto towerX = float(start_.x);
61 auto towerY = float(start_.y);
62 float distanceX = abs(enemyX - towerX);
63 float distanceY = abs(enemyY - towerY);
64
65 float factorX = distanceX / distance;
66 float factorY = distanceY / distance;
67
68 float velocityX = velocity_ * factorX;
69 float velocityY = velocity_ * factorY;
70
71 if (enemyX < towerX) {
72     velocityX = -velocityX;
73 }
74 if (enemyY < towerY) {
75     velocityY = -velocityY;
76 }
77
78 //std::cout << "DistanceCovered: " << distanceCovered << " Distance: " << distance << std::endl;
79 //std::cout << entitySprite.getPosition().x << "," << entitySprite.getPosition().y << std::endl;
80 //std::cout << velocityX << "," << velocityY << std::endl;
81
82 if (type_ == ProjectileType::Bullet) {
83     if (distanceCovered <= distance) {
84         entitySprite.move(velocityX, velocityY);
85         distanceCovered += velocity_;
86     }
87     else {
88         if (closestEnemy_.second != nullptr) {
89             closestEnemy_.second->takeDamage(damage_);
90         }
91         atFinish = true;
92     }
93 }
94 else if (type_ == ProjectileType::FreezeGun) {
95     if (distanceCovered <= distance) {
96         entitySprite.move(velocityX, velocityY);
97         distanceCovered += velocity_;
98     }
99     else {
100         if (freezeEnemy != nullptr) {
101             freezeEnemy->slowDown();
102             freezeEnemy->takeDamage(damage_);
103         }
104         atFinish = true;
105     }
106 }
107 else if (type_ == ProjectileType::Bomb) {
108     //Explosion texture
109     if (distance-distanceCovered <= 30.f) {
110         entitySprite.setTexture(textures_.get(Textures::Explosion));
111     }
112     //Check if at finish
113     if (distanceCovered <= distance) {

```

```

114     entitySprite.move(velocityX, velocityY);
115     distanceCovered += velocity_;
116 }
117 else {
118     //Closest enemy takes damage
119     if (closestEnemy_.second != nullptr) {
120         closestEnemy_.second->takeDamage(damage_);
121     }
122
123     //Other enemies inside range take damage as well
124     for (auto enemy : neighbourEnemies_) {
125         //std::cout << enemy.second.first << std::endl;
126         if (enemy.second.first <= range_) {
127             if (enemy.second.second != nullptr) {
128                 enemy.second.second->takeDamage(damage_);
129             }
130         }
131     }
132     atFinish = true;
133 }
134 }
135 //std::cout << "DistanceCovered: " << distanceCovered << "    Distance: " << distance << std::endl;
136 }

```

## 14.17.4 Member Data Documentation

### 14.17.4.1 atFinish

bool Projectile::atFinish [private]

Definition at line 90 of file Projectile.hpp.

### 14.17.4.2 closestEnemies\_

std::map<int, std::pair<float, std::shared\_ptr<Enemy> > > Projectile::closestEnemies\_↵  
[private]

Definition at line 82 of file Projectile.hpp.

### 14.17.4.3 closestEnemy\_

std::pair<float, std::shared\_ptr<Enemy> > Projectile::closestEnemy\_ [private]

Definition at line 81 of file Projectile.hpp.

### 14.17.4.4 damage\_

int Projectile::damage\_ [private]

Definition at line 86 of file Projectile.hpp.

#### 14.17.4.5 distance

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

Definition at line 89 of file Projectile.hpp.

#### 14.17.4.6 distanceCovered

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

Definition at line 88 of file Projectile.hpp.

#### 14.17.4.7 finish\_

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

Definition at line 80 of file Projectile.hpp.

#### 14.17.4.8 neighbourEnemies\_

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

Definition at line 83 of file Projectile.hpp.

#### 14.17.4.9 range\_

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

Definition at line 85 of file Projectile.hpp.

#### 14.17.4.10 start\_

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

Definition at line 79 of file Projectile.hpp.

#### 14.17.4.11 textures\_

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

Definition at line 84 of file Projectile.hpp.

#### 14.17.4.12 type\_

```
ProjectileType Projectile::type_
```

type of this projectile

Definition at line 76 of file Projectile.hpp.

#### 14.17.4.13 velocity\_

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

Definition at line 87 of file Projectile.hpp.

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

- [src/entity/Projectile.hpp](#)
- [src/entity/Projectile.cpp](#)

## 14.18 ProjectileType Class Reference

[Contains](#) identification for different projectiles.

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

### 14.18.1 Detailed Description

[Contains](#) identification for different projectiles.

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

- [src/entity/Projectile.hpp](#)

## 14.19 request Struct Reference

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

## Public Attributes

- [Scenes::ID](#) `scene`
- `size_t` `number`

### 14.19.1 Detailed Description

Definition at line 25 of file `Scene.hpp`.

### 14.19.2 Member Data Documentation

#### 14.19.2.1 `number`

```
size_t request::number
```

Definition at line 27 of file `Scene.hpp`.

#### 14.19.2.2 `scene`

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

Definition at line 26 of file `Scene.hpp`.

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

- `src/game/Scene.hpp`

## 14.20 ResourceHolder< Resource, Identifier > Class Template Reference

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

### Public Member Functions

- `void load` (`Identifier id`, `const std::string &filename`)  
*A template function for a Resource.*
- `Resource & get` (`Identifier id`)
- `const Resource & get` (`Identifier id`) `const`

### Private Attributes

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

### 14.20.1 Detailed Description

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



## Template Parameters

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

Definition at line 55 of file Resource.hpp.

## 14.20.2 Member Function Documentation

### 14.20.2.1 get() [1/2]

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

## Parameters

<i>id</i>	Identifier for wanted resource
-----------	--------------------------------

## Returns

Reference to wanted resource

Definition at line 92 of file Resource.hpp.

```
92                                     {
93     auto found = mResourceMap.find(id);
94     assert(found != mResourceMap.end());
95     return *found->second;
96 }
```

### 14.20.2.2 get() [2/2]

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

Definition at line 98 of file Resource.hpp.

```
98                                     {
99     auto found = mResourceMap.find(id);
100     assert(found != mResourceMap.end());
101     return *found->second;
102 }
```

### 14.20.2.3 load()

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

A template function for a Resource.

## Parameters

<i>id</i>	Identifier for this texture
<i>filename</i>	File to load to this identifier

## Template Parameters

<i>Resource</i>	
<i>Identifier</i>	Identifier for resources

## Parameters

<i>id</i>	Identifier name, used when accessing a Resource
<i>filename</i>	File to load from

Definition at line 82 of file Resource.hpp.

```

83                                     {
84     std::unique_ptr<Resource> resource(new Resource());
85     if (!resource->loadFromFile(filename))
86         throw std::runtime_error("ResourceHolder::load - Failed to load " + filename);
87     auto inserted = mResourceMap.insert(
88         std::make_pair(id, std::move(resource)));
89     assert(inserted.second);
90 }
```

## 14.20.3 Member Data Documentation

### 14.20.3.1 mResourceMap

```

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

Definition at line 72 of file Resource.hpp.

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

- src/resource/[Resource.hpp](#)

## 14.21 Scene Class Reference

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

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

Inheritance diagram for Scene:

Collaboration diagram for Scene:

## Public Member Functions

- virtual void [handleInput](#) (const sf::Event &event)=0
- virtual void [update](#) (sf::Time deltaTime)=0  
*updates everything is a scene including UI elements and entities*
- virtual [sceneRequest requestedScene](#) ()=0  
*Should the current scene be changed.*
- virtual [Scenes::ID sceneType](#) ()=0  
*Returns the ID the scene, does not change.*

## Public Attributes

- [TextureHolder textures](#)

### 14.21.1 Detailed Description

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

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

Definition at line 37 of file Scene.hpp.

### 14.21.2 Member Function Documentation

#### 14.21.2.1 [handleInput\(\)](#)

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

Implemented in [MapScene](#), [LevelSelect](#), [GameTitle](#), and [GameEnd](#).

#### 14.21.2.2 [requestedScene\(\)](#)

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

Should the current scene be changed.

#### Returns

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

Implemented in [MapScene](#), [LevelSelect](#), [GameTitle](#), and [GameEnd](#).

### 14.21.2.3 sceneType()

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

Returns the ID the scene, does not change.

#### Returns

Does not change after scene creation

Implemented in [MapScene](#), [LevelSelect](#), [GameTitle](#), and [GameEnd](#).

### 14.21.2.4 update()

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

updates everything in a scene including UI elements and entities

<

#### Parameters

<i>deltaTime</i>	time since last update in <a href="#">Game</a> loop
------------------	-----------------------------------------------------

Implemented in [LevelSelect](#), [GameTitle](#), [GameEnd](#), and [MapScene](#).

## 14.21.3 Member Data Documentation

### 14.21.3.1 textures

```
TextureHolder Scene::textures
```

Definition at line 55 of file [Scene.hpp](#).

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

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

## 14.22 SceneItem Class Reference

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

Inheritance diagram for SceneItem:

Collaboration diagram for SceneItem:

## Public Types

- typedef std::unique\_ptr< [SceneItem](#) > [Ptr](#)

## Public Member Functions

- [SceneItem](#) ()

## Private Member Functions

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

### 14.22.1 Detailed Description

Definition at line 10 of file SceneItem.hpp.

### 14.22.2 Member Typedef Documentation

#### 14.22.2.1 Ptr

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

Definition at line 12 of file SceneItem.hpp.

### 14.22.3 Constructor & Destructor Documentation

#### 14.22.3.1 SceneItem()

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

### 14.22.4 Member Function Documentation

#### 14.22.4.1 draw()

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

Definition at line 7 of file SceneItem.cpp.

```
8
9 // see book p.60
10 states.transform *= getTransform();
11 }
```

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

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

## 14.23 Sector Struct Reference

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

```
#include <Sector.hpp>
```

### Public Member Functions

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

### Static Public Member Functions

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

### Public Attributes

- int [x](#)
- int [y](#)

### Static Public Attributes

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

#### 14.23.1 Detailed Description

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

Definition at line 12 of file Sector.hpp.

## 14.23.2 Member Function Documentation

### 14.23.2.1 fromCoords()

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

Definition at line 30 of file Sector.hpp.

```
30 {
31     int xCoordMod = xCoord % 64;
32     int yCoordMod = yCoord % 64;
33     return Sector{static_cast<int>(xCoord - xCoordMod), static_cast<int>(yCoord - yCoordMod)};
34 }
```

### 14.23.2.2 operator==( )

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

Definition at line 15 of file Sector.cpp.

```
15 {
16     return x == rhs.x && y == rhs.y;
17 }
```

### 14.23.2.3 upperLeftPoint()

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

Definition at line 11 of file Sector.cpp.

```
11 {
12     return sf::Vector2f(static_cast<float>(x), static_cast<float>(y));
13 }
```

## 14.23.3 Member Data Documentation

### 14.23.3.1 DiagVector

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

Definition at line 21 of file Sector.hpp.

#### 14.23.3.2 scale

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

Definition at line 22 of file Sector.hpp.

#### 14.23.3.3 Size

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

Definition at line 20 of file Sector.hpp.

#### 14.23.3.4 x

```
int Sector::x
```

Definition at line 13 of file Sector.hpp.

#### 14.23.3.5 y

```
int Sector::y
```

Definition at line 14 of file Sector.hpp.

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

- [src/ui/Sector.hpp](#)
- [src/ui/Sector.cpp](#)

## 14.24 SelectTowerButton< T > Class Template Reference

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

Inheritance diagram for SelectTowerButton< T >:

Collaboration diagram for SelectTowerButton< T >:

### Public Member Functions

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



## Protected Member Functions

- virtual void [onClick](#) ([World](#) &world) override

## Protected Attributes

- [Sector](#) [towerSector](#)

### 14.24.1 Detailed Description

```
template<typename T>
class SelectTowerButton< T >
```

Definition at line 11 of file SelectTowerButton.hpp.

### 14.24.2 Constructor & Destructor Documentation

#### 14.24.2.1 SelectTowerButton()

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

Definition at line 22 of file SelectTowerButton.hpp.

```
22
23     Button(position, T::TEXTURE_ID),
24     towerSector(towerSector) {
25 }
```

### 14.24.3 Member Function Documentation

#### 14.24.3.1 draw()

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

Reimplemented from [Button](#).

Definition at line 28 of file SelectTowerButton.hpp.

```
28
29
30     Button::draw(target, states);
31 }
```

### 14.24.3.2 onClick()

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

Reimplemented from [Button](#).

Definition at line 34 of file SelectTowerButton.hpp.

```
34 {
35     Button::onClick(world);
36 }
```

## 14.24.4 Member Data Documentation

### 14.24.4.1 towerSector

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

Definition at line 17 of file SelectTowerButton.hpp.

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

- src/ui/[SelectTowerButton.hpp](#)

## 14.25 Tower Class Reference

[Tower](#) to display on the map and shoot, extends [Entity](#) class.

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

Inheritance diagram for Tower:

Collaboration diagram for Tower:

### Public Member Functions

- [Tower](#) ([TextureHolder](#) &textures, [SoundBufferHolder](#) &sounds, int textureID, [Sector](#) sector, int damage, float range, sf::Time shootDelay, int price, [TowerType](#) towerType)
- void [update](#) (sf::Time deltaTime, [World](#) &world)
  - Update the projectile after a frame change.*
- bool [upgrade](#) ()
  - Function to call when this tower is upgraded.*
- float [getRange](#) () const
  - Get this tower's range as a float value.*
- int [getPrice](#) () const
  - Get this tower's price.*
- const [Sector](#) & [getSector](#) () const
  - Get a reference to the [Sector](#) this tower is in.*

## Public Attributes

- [TowerType](#) type

*This tower's type.*

## Private Member Functions

- bool [inRange](#) (const sf::Vector2f enemyPos, float [range](#))
- std::pair< float, std::shared\_ptr< [Enemy](#) > > [getClosestEnemy](#) ([World](#) &world)
- std::map< int, std::pair< float, std::shared\_ptr< [Enemy](#) > > > [getClosestEnemies](#) ([World](#) &world)
- std::map< int, std::pair< float, std::shared\_ptr< [Enemy](#) > > > [getNeighbourEnemies](#) (std::pair< float, std::shared\_ptr< [Enemy](#) > > ClosestEnemy, [World](#) &world)

## Private Attributes

- bool [upgradeable](#)
- [Sector](#) [sector](#)
- [TextureHolder](#) & [textures\\_](#)
- [SoundBufferHolder](#) & [sounds\\_](#)
- std::map< int, std::pair< int, int > > [testii](#)
- float [bombRange](#)
- int [damage](#)
- float [range](#)
- int [price](#)
- float [velocity\\_](#)
- sf::Sound [soundBullet\\_](#)
- sf::Sound [soundBomb\\_](#)
- sf::Sound [soundSnow\\_](#)
- sf::Time [shootDelay](#)
- sf::Time [timeSinceFiring](#)

## Additional Inherited Members

### 14.25.1 Detailed Description

[Tower](#) to display on the map and shoot, extends [Entity](#) class.

Definition at line 33 of file Tower.hpp.

### 14.25.2 Constructor & Destructor Documentation

### 14.25.2.1 Tower()

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

Definition at line 9 of file Tower.cpp.

```
17                                     :
18 Entity(textureID, textures),
19 textures_(textures),
20 sounds_(sounds),
21 sector(sector),
22 damage(damage),
23 range(range),
24 shootDelay(shootDelay),
25 timeSinceFiring(sf::Time::Zero),
26 price(price),
27 type(towerType),
28 upgradeable(true),
29 bombRange(150.f),
30 velocity_(10.f)
31 {
32     entitySprite.setPosition(sector.x, sector.y);
33     soundBullet_.setBuffer(sounds.get(SoundBuffers::GunCat));
34     soundBullet_.setVolume(50.f);
35     soundBomb_.setBuffer(sounds.get(SoundBuffers::BombCatMeow));
36     soundSnow_.setBuffer(sounds.get(SoundBuffers::FreezeCatMeow));
37     if (towerType == TowerType::FreezeCat) {
38         upgradeable = false;
39     }
40 }
```

## 14.25.3 Member Function Documentation

### 14.25.3.1 getClosestEnemies()

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

Definition at line 102 of file Tower.cpp.

```
102                                     {
103     std::map<float, std::shared_ptr<Enemy>> enemyDistances;
104
105     float towerX = entitySprite.getPosition().x;
106     float towerY = entitySprite.getPosition().y;
107     for (const auto& enemy : world.getEnemies()) {
108         float enemyX = enemy->getPosition().x;
109         float enemyY = enemy->getPosition().y;
110         float distanceX = abs(enemyX - towerX);
111         float distanceY = abs(enemyY - towerY);
112         float distance = sqrt(distanceX * distanceX + distanceY * distanceY);
113         enemyDistances.insert(std::make_pair(distance, std::shared_ptr<Enemy>(enemy)));
114     }
115     std::map<int, std::pair<float, std::shared_ptr<Enemy>> > closestEnemies;
116     int counter = 0;
117     for (std::map<float, std::shared_ptr<Enemy>::iterator enemy = enemyDistances.begin(); enemy !=
118         enemyDistances.end(); enemy++) {
119         // Sort into result
120         closestEnemies[counter] = std::make_pair(enemy->first, std::shared_ptr<Enemy>(enemy->second));
121         counter++;
122     }
123     return closestEnemies;
124 }
```

### 14.25.3.2 getClosestEnemy()

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

Definition at line 76 of file Tower.cpp.

```
76
77     std::shared_ptr<Enemy> closestEnemy = nullptr;
78     float closestDistance = 0.0;
79
80     float towerX = entitySprite.getPosition().x;
81     float towerY = entitySprite.getPosition().y;
82
83     for (auto&& enemy : world.getEnemies()) {
84
85         float enemyX = enemy->getPosition().x;
86         float enemyY = enemy->getPosition().y;
87
88         float distanceX = abs(enemyX - towerX);
89         float distanceY = abs(enemyY - towerY);
90
91         float distance = sqrt(distanceX * distanceX + distanceY * distanceY);
92
93         if (closestDistance == 0.0 || distance < closestDistance) {
94             closestEnemy = enemy;
95             closestDistance = distance;
96         }
97     }
98     //std::cout << closestDistance << std::endl;
99     return std::make_pair(closestDistance, closestEnemy);
100 }
```

### 14.25.3.3 getNeighbourEnemies()

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

Definition at line 126 of file Tower.cpp.

```
126
127     {
128         std::map<float, std::shared_ptr<Enemy>> enemyDistances;
129
130         float closestEnemyX = closestEnemy.second->getPosition().x;
131         float closestEnemyY = closestEnemy.second->getPosition().y;
132         for (auto& enemy : world.getEnemies()) {
133             float enemyX = enemy->getPosition().x;
134             float enemyY = enemy->getPosition().y;
135             float distanceX = abs(enemyX - closestEnemyX);
136             float distanceY = abs(enemyY - closestEnemyY);
137             float distance = sqrt(distanceX * distanceX + distanceY * distanceY);
138             if (enemy != closestEnemy.second) {
139                 enemyDistances.insert(std::make_pair(distance, std::shared_ptr<Enemy>(enemy)));
140             }
141         }
142         std::map<int, std::pair<float, std::shared_ptr<Enemy>>> neighbourEnemies;
143         int counter = 0;
144         for (std::map<float, std::shared_ptr<Enemy>>::iterator enemy = enemyDistances.begin(); enemy !=
145             enemyDistances.end(); enemy++) {
146             // Sort into result
147             neighbourEnemies[counter] = std::make_pair(enemy->first, std::shared_ptr<Enemy>(enemy->second));
148             counter++;
149         }
150         return neighbourEnemies;
151     }
```

#### 14.25.3.4 getPrice()

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

Get this tower's price.

##### Returns

[Price](#) as int

Definition at line 64 of file Tower.hpp.

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

#### 14.25.3.5 getRange()

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

Get this tower's range as a float value.

##### Returns

Radius of the range as float

Definition at line 59 of file Tower.hpp.

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

#### 14.25.3.6 getSector()

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

Get a reference to the [Sector](#) this tower is in.

##### Returns

Reference to location [Sector](#)

Definition at line 69 of file Tower.hpp.

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

### 14.25.3.7 inRange()

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

Definition at line 152 of file Tower.cpp.

```
152                                     {
153     float enemyX = enemyPos.x;
154     float enemyY = enemyPos.y;
155     float towerX = entitySprite.getPosition().x;
156     float towerY = entitySprite.getPosition().y;
157     float distanceX = abs(enemyX - towerX);
158     float distanceY = abs(enemyY - towerY);
159     float distance = sqrt(distanceX * distanceX + distanceY * distanceY);
160     if (distance <= range) {
161         return true;
162     }
163     else
164         return false;
165 }
```

### 14.25.3.8 update()

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

Update the projectile after a frame change.

#### Parameters

<i>deltaTime</i>	Time since last frame
<i>world</i>	Reference to <a href="#">World</a> class where the projectile is displayed

Implements [Entity](#).

Definition at line 42 of file Tower.cpp.

```
42                                     {
43     timeSinceFiring += frameDelay;
44     //std::cout << "timeSinceFiring: " << timeSinceFiring.asSeconds() << "    deltaTime: " <<
45         deltaTime.asSeconds() << "    shootDelay: " << shootDelay.asSeconds() << std::endl;
46     if (timeSinceFiring < shootDelay) {
47         return;
48     }
49
50     if (timeSinceFiring >= shootDelay) {
51         std::pair<float, std::shared_ptr<Enemy>> closestEnemy = getClosestEnemy(world);
52         if (closestEnemy.second != nullptr && inRange(closestEnemy.second->getPosition(), range)) {
53             if (type == TowerType::GunCat) {
54                 // Shoot Bullet
55                 world.addProjectile(Projectile::make(textures_, Textures::Bullet, ProjectileType::Bullet, sector,
56                     closestEnemy, getClosestEnemies(world), getNeighbourEnemies(closestEnemy, world), 1.f, damage,
57                     velocity_));
58                 soundBullet_.play();
59                 timeSinceFiring = sf::Time::Zero;
60             }
61             else if (type == TowerType::FreezeCat) {
62                 // Shoot Freeze
63                 world.addProjectile(Projectile::make(textures_, Textures::Snowflake, ProjectileType::FreezeGun,
64                     sector, closestEnemy, getClosestEnemies(world), getNeighbourEnemies(closestEnemy, world), range,
65                     damage, velocity_));
66                 soundSnow_.play();
67                 timeSinceFiring = sf::Time::Zero;
68             }
69         }
70     }
```

```

65         else if (type == TowerType::BombCat) {
66             // Shoot Bomb
67             world.addProjectile(Projectile::make(textures_, Textures::Bomb, ProjectileType::Bomb, sector,
closestEnemy, getClosestEnemies(world), getNeighbourEnemies(closestEnemy, world), bombRange, damage,
velocity_));
68             soundBomb_.play();
69             timeSinceFiring = sf::Time::Zero;
70         }
71     }
72     timeSinceFiring = sf::Time::Zero;
73 }
74 }

```

### 14.25.3.9 upgrade()

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

Function to call when this tower is upgraded.

#### Returns

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

Definition at line 167 of file Tower.cpp.

```

167     {
168         if (type == TowerType::GunCat) {
169             damage = damage * 2;
170             velocity_ = velocity_ * 1.5;
171             shootDelay = sf::seconds(1);
172             entitySprite.setTexture(textures_.get(Textures::UpgradedGunCat));
173             upgradeable = false;
174             return true;
175         }
176         else if (type == TowerType::FreezeCat) {
177             range = 350.f;
178             velocity_ = velocity_ * 1.5;
179             shootDelay = sf::seconds(1);
180             entitySprite.setTexture(textures_.get(Textures::UpgradedFreezeCat));
181             upgradeable = false;
182             return true;
183         }
184         else if (type == TowerType::BombCat) {
185             bombRange = 200.f;
186             shootDelay = sf::seconds(2);
187             entitySprite.setTexture(textures_.get(Textures::UpgradedBombCat));
188             upgradeable = false;
189             return true;
190         }
191         else
192             return false;
193     }

```

## 14.25.4 Member Data Documentation

### 14.25.4.1 bombRange

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

Definition at line 86 of file Tower.hpp.



#### 14.25.4.2 damage

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

Definition at line 87 of file Tower.hpp.

#### 14.25.4.3 price

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

Definition at line 89 of file Tower.hpp.

#### 14.25.4.4 range

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

Definition at line 88 of file Tower.hpp.

#### 14.25.4.5 sector

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

Definition at line 82 of file Tower.hpp.

#### 14.25.4.6 shootDelay

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

Definition at line 94 of file Tower.hpp.

#### 14.25.4.7 soundBomb\_

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

Definition at line 92 of file Tower.hpp.

#### 14.25.4.8 soundBullet\_

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

Definition at line 91 of file Tower.hpp.

#### 14.25.4.9 sounds\_

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

Definition at line 84 of file Tower.hpp.

#### 14.25.4.10 soundSnow\_

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

Definition at line 93 of file Tower.hpp.

#### 14.25.4.11 testii

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

Definition at line 85 of file Tower.hpp.

#### 14.25.4.12 textures\_

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

Definition at line 83 of file Tower.hpp.

#### 14.25.4.13 timeSinceFiring

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

Definition at line 95 of file Tower.hpp.

**14.25.4.14 type**

```
TowerType Tower::type
```

This tower's type.

Definition at line 73 of file Tower.hpp.

**14.25.4.15 upgradeable**

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

Definition at line 76 of file Tower.hpp.

**14.25.4.16 velocity\_**

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

Definition at line 90 of file Tower.hpp.

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

- src/entity/[Tower.hpp](#)
- src/entity/[Tower.cpp](#)

**14.26 TowerMenu Class Reference**

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

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

Inheritance diagram for TowerMenu:

Collaboration diagram for TowerMenu:

**Public Member Functions**

- [TowerMenu](#) ([World](#) &world)
- virtual [~TowerMenu](#) ()
- void [handleInput](#) (const sf::Event &event, [World](#) &world)  
*handle player input*
- void [update](#) ([World](#) &world, const sf::Vector2f &mousePosition)
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override  
*Draw a square to show which [Sector](#) is selected.*

## Private Attributes

- [Sector](#) `selectedSector`
- `sf::RectangleShape` `hoverIndicator`
- `bool` `showSectorIndicator`
- `sf::RectangleShape` `sectorIndicator`
- `sf::CircleShape` `rangeIndicator`

### 14.26.1 Detailed Description

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

Definition at line 16 of file TowerMenu.hpp.

### 14.26.2 Constructor & Destructor Documentation

#### 14.26.2.1 TowerMenu()

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

Definition at line 9 of file TowerMenu.cpp.

```
9      :
10      showSectorIndicator (false),
11      sectorIndicator(),
12      rangeIndicator() {
13      sectorIndicator.setSize(Sector::DiagVector);
14      sectorIndicator.setFillColor(sf::Color::Transparent);
15      sectorIndicator.setOutlineColor(hoverColor);
16      sectorIndicator.setOutlineThickness(3.f);
17      rangeIndicator.setRadius(0);
18      rangeIndicator.setFillColor(rangeIndicatorColor);
19      rangeIndicator.setOutlineColor(hoverColor);
20      rangeIndicator.setOutlineThickness(3.f);
21 }
```

#### 14.26.2.2 ~TowerMenu()

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

Definition at line 19 of file TowerMenu.hpp.

```
19 {}
```

### 14.26.3 Member Function Documentation

#### 14.26.3.1 draw()

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

Draw a square to show which [Sector](#) is selected.

## Parameters

<i>target</i>	sf::RenderTarget to draw the scene to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 81 of file TowerMenu.cpp.

```

81                                     {
82     target.draw(sectorIndicator, states);
83     target.draw(rangeIndicator, states);
84 }
```

## 14.26.3.2 handleInput()

```

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

handle player input

## Parameters

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

Definition at line 23 of file TowerMenu.cpp.

```

23                                     {
24     if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button == sf::Mouse::Button::Left)
25     {
26         showSectorIndicator = true;
27         Sector sector = Sector::fromCoords(event.mouseButton.x, event.mouseButton.y);
28         if ((sector.x / Sector::Size) <= world.getMapGrid().getHeight() - 1
29             && (sector.y / Sector::Size) <= world.getMapGrid().getWidth() - 1
30             && world.getMapGrid().getBlockAt(sector.x / Sector::Size, sector.y / Sector::Size) == 0) {
31             auto t = world.getTowerAt(sector);
32             float selectedTowerRange;
33             if (t != nullptr) selectedTowerRange = t->getRange(); else selectedTowerRange = 0;
34             sectorIndicator.setPosition(sector.upperLeftPoint());
35             rangeIndicator.setPosition(sector.upperLeftPoint().x - selectedTowerRange + ((float) Sector::Size / 2),
36                                     sector.upperLeftPoint().y - selectedTowerRange + ((float) Sector::Size / 2));
37             rangeIndicator.setRadius(selectedTowerRange);
38             selectedSector = sector;
39         } else if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button ==
40                     sf::Mouse::Button::Right) {
41             Sector sector = Sector::fromCoords(event.mouseButton.x, event.mouseButton.y);
42             if (world.getTowerAt(sector) != nullptr)
43                 world.removeTower(const_cast<Tower*>(world.getTowerAt(sector)),
44                                   (int) (0.8 * world.getTowerAt(sector)->getPrice()));
45         } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num1) {
46             if (world.getTowerAt(selectedSector) == nullptr) {
47                 TowerPtr
48                 t = std::make_unique<Tower>(world.getTextures(), world.getSounds(), Textures::GunCat,
49                                             selectedSector, 2, 250.f, sf::seconds(2), 300, TowerType::GunCat);
50                 world.addTower(std::move(t), t->getPrice());
51             } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num2
52                         && world.getMapGrid().getBlockAt(selectedSector.x / 64, selectedSector.y / 64) == 0) {
53                 if (world.getTowerAt(selectedSector) == nullptr) {
54                     TowerPtr
55                     t = std::make_unique<Tower>(world.getTextures(), world.getSounds(), Textures::FreezeCat,
56                                                 selectedSector, 1, 250.f, sf::seconds(2), 400,
57                                                 TowerType::FreezeCat);
```

```

57     world.addTower(std::move(t), t->getPrice());
58 }
59 } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::P) {
60     world.paused = !world.paused;
61 } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num3
62     && world.getMapGrid().getBlockAt(selectedSector.x / 64, selectedSector.y / 64) == 0) {
63     if (world.getTowerAt(selectedSector) == nullptr) {
64         TowerPtr
65         t = std::make_unique<Tower>(world.getTextures(), world.getSounds(), Textures::BombCat,
66             selectedSector, 1, 250.f, sf::seconds(3), 500, TowerType::BombCat);
67         world.addTower(std::move(t), t->getPrice());
68     }
69 } else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num4
70     && world.getMapGrid().getBlockAt(selectedSector.x / 64, selectedSector.y / 64) == 0) {
71     if (world.getTowerAt(selectedSector) != nullptr) {
72         world.upgradeTower(world.getTowerAt(selectedSector), 500);
73     }
74 }
75 }

```

### 14.26.3.3 update()

```

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

```

Definition at line 77 of file TowerMenu.cpp.

```

77 {
78
79 }

```

## 14.26.4 Member Data Documentation

### 14.26.4.1 hoverIndicator

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

Definition at line 41 of file TowerMenu.hpp.

### 14.26.4.2 rangeIndicator

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

Definition at line 44 of file TowerMenu.hpp.

### 14.26.4.3 sectorIndicator

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

Definition at line 43 of file TowerMenu.hpp.

#### 14.26.4.4 selectedSector

`Sector TowerMenu::selectedSector [private]`

Definition at line 40 of file TowerMenu.hpp.

#### 14.26.4.5 showSectorIndicator

`bool TowerMenu::showSectorIndicator [private]`

Definition at line 42 of file TowerMenu.hpp.

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

- [src/ui/TowerMenu.hpp](#)
- [src/ui/TowerMenu.cpp](#)

## 14.27 TowerType Class Reference

contains possible [Tower](#) types

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

### 14.27.1 Detailed Description

contains possible [Tower](#) types

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

- [src/entity/Tower.hpp](#)

## 14.28 Wave Class Reference

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

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

### Public Member Functions

- [Wave](#) ([TextureHolder](#) &textureholder, int textureID, int count, sf::Time spacing, std::map< int, std::pair< int, int >> &pathMarkers, int hitPoints, float speed)  
*Constructor for a [Wave](#).*
- [EnemyPtr ifNextEnemy](#) (sf::Time deltaTime)  
*if next enemy should spawn this function returns it*
- bool [isEmpty](#) () const  
*is the current wave done*
- int [getEnemiesLeft](#) ()  
*return amount of enemies left during this wave (shown in UI etc.)*

## Static Public Attributes

- static const sf::Time [SPACING\\_HUGE](#) = sf::milliseconds(1400)  
*Large time difference for spawning enemies.*
- static const sf::Time [SPACING\\_WIDE](#) = sf::milliseconds(1000)
- static const sf::Time [SPACING\\_MEDIUM](#) = sf::milliseconds(400)
- static const sf::Time [SPACING\\_NARROW](#) = sf::milliseconds(200)  
*narrow time difference for spawning enemies*

## Private Attributes

- int [enemiesLeft](#)
- [EnemyPtr](#) [enemyTemplate](#)
- sf::Time [spacing\\_](#)
- sf::Time [elapsed\\_](#)

### 14.28.1 Detailed Description

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

Definition at line 16 of file Wave.hpp.

### 14.28.2 Constructor & Destructor Documentation

#### 14.28.2.1 Wave()

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

Constructor for a [Wave](#).

#### Parameters

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



Definition at line 12 of file Wave.cpp.

```

18         :
19     enemyTemplate(Enemy::make(textureholder, textureID, pathMarkers, hitPoints, speed)),
20     enemiesLeft(count),
21     elapsed_(sf::Time::Zero),
22     spacing_(spacing) {
23 }
```

## 14.28.3 Member Function Documentation

### 14.28.3.1 getEnemiesLeft()

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

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

#### Returns

integer, how many enemies left

Definition at line 60 of file Wave.hpp.

```
60 { return enemiesLeft; }
```

### 14.28.3.2 ifNextEnemy()

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

if next enemy should spawn this function returns it

#### Parameters

<i>deltaTime</i>	time since last frame
------------------	-----------------------

#### Returns

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

Definition at line 25 of file Wave.cpp.

```

25                                     {
26     elapsed_ += deltaTime;
27     if (enemiesLeft && elapsed_ >= deltaTime) {
28         elapsed_ -= spacing_;
29         enemiesLeft--;
30         return enemyTemplate->clone(enemyTemplate->getTextureHolder(),
31                                     enemyTemplate->getTextureID(),
32                                     enemyTemplate->getPathMarkers(),
33                                     enemyTemplate->getHitPoints(),
34                                     enemyTemplate->getSpeed());
35     }
36     return nullptr;
37 }
```

### 14.28.3.3 isEmpty()

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

is the current wave done

#### Returns

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

Definition at line 55 of file Wave.hpp.

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

## 14.28.4 Member Data Documentation

### 14.28.4.1 elapsed\_

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

Definition at line 66 of file Wave.hpp.

### 14.28.4.2 enemiesLeft

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

Definition at line 63 of file Wave.hpp.

### 14.28.4.3 enemyTemplate

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

Definition at line 64 of file Wave.hpp.

### 14.28.4.4 spacing\_

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

Definition at line 65 of file Wave.hpp.

#### 14.28.4.5 SPACING\_HUGE

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

Large time difference for spawning enemies.

Definition at line 21 of file Wave.hpp.

#### 14.28.4.6 SPACING\_MEDIUM

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

Definition at line 23 of file Wave.hpp.

#### 14.28.4.7 SPACING\_NARROW

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

narrow time difference for spawning enemies

Definition at line 27 of file Wave.hpp.

#### 14.28.4.8 SPACING\_WIDE

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

Definition at line 22 of file Wave.hpp.

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

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

## 14.29 WaveController Class Reference

[WaveController](#) controls the current wave and makes a new one when the player is ready.

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

Collaboration diagram for WaveController:

## Public Member Functions

- [WaveController](#) ([TextureHolder](#) &textureholder, std::map< int, std::pair< int, int >> &pathMarkers)  
*Constructor for [WaveController](#).*
- void [update](#) (sf::Time deltaTime, [World](#) &world)  
*Checks wave status If player has ordered a new wave, starts it, or if enemies still left in current wave, spawns the next one.*
- [WavePtr](#) [makeNewWave](#) (int waveNumber)  
*Make a new wave to the world.*
- int [getWaveNumber](#) () const  
*Get current wave number.*
- int [getWaveEnemiesLeft](#) ()  
*Get how many enemies there are left in the current wave.*

## Private Attributes

- [TextureHolder](#) & [textureholder\\_](#)
- std::map< int, std::pair< int, int >> & [pathMarkers\\_](#)
- int [waveNumber](#)
- [WavePtr](#) [wave](#)

### 14.29.1 Detailed Description

[WaveController](#) controls the current wave and makes a new one when the player is ready.

Definition at line 14 of file WaveController.hpp.

### 14.29.2 Constructor & Destructor Documentation

#### 14.29.2.1 WaveController()

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

Constructor for [WaveController](#).

#### Parameters

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

Definition at line 21 of file WaveController.hpp.

```
21
22     waveNumber(0), wave(), textureholder\_(textureholder), pathMarkers\_(pathMarkers) {};
```

## 14.29.3 Member Function Documentation

### 14.29.3.1 getWaveEnemiesLeft()

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

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

#### Returns

Amount of enemies as an integer

Definition at line 45 of file WaveController.hpp.

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

### 14.29.3.2 getWaveNumber()

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

Get current wave number.

#### Returns

current wave number

Definition at line 40 of file WaveController.hpp.

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

### 14.29.3.3 makeNewWave()

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

Make a new wave to the world.

#### Parameters

<i>waveNumber</i>	goes to a switch case, determines how hard the wave will be
-------------------	-------------------------------------------------------------

#### Returns

Pointer to the new [Wave](#) object

Definition at line 29 of file WaveController.cpp.

```

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

```

```

114         return std::make_unique<Wave>(textureholder_,
115                                         Textures::FatRat,
116                                         20,
117                                         Wave::SPACING_WIDE,
118                                         pathMarkers_,
119                                         40,
120                                         1.f);
121     case 12:
122         return std::make_unique<Wave>(textureholder_,
123                                         Textures::FatRat,
124                                         40,
125                                         Wave::SPACING_WIDE,
126                                         pathMarkers_,
127                                         50,
128                                         1.f);
129     case 20:
130         return std::make_unique<Wave>(textureholder_,
131                                         Textures::FatRat,
132                                         5,
133                                         Wave::SPACING_HUGE,
134                                         pathMarkers_,
135                                         10000,
136                                         4.f);
137     default:
138         if (waveNumber % 2)
139             return std::make_unique<Wave>(textureholder_,
140                                             Textures::FatRat,
141                                             50,
142                                             Wave::SPACING_WIDE,
143                                             pathMarkers_,
144                                             waveNumber * 10,
145                                             1.f);
146         else
147             return std::make_unique<Wave>(textureholder_,
148                                             Textures::FastRat,
149                                             waveNumber * 20,
150                                             Wave::SPACING_NARROW,
151                                             pathMarkers_,
152                                             waveNumber,
153                                             4.f);
154     }
155 }
156 }

```

#### 14.29.3.4 update()

```

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

```

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

##### Parameters

<i>deltaTime</i>	time since last update
<i>world</i>	world to spawn enemies in

Definition at line 10 of file WaveController.cpp.

```

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

```

```
21  auto next = wave->ifNextEnemy(delayTime);
22
23  //if not nullptr
24  if (next && !world.paused) {
25      world.getEnemies().push_back(std::move(next));
26  }
27 }
```

## 14.29.4 Member Data Documentation

### 14.29.4.1 pathMarkers\_

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

Definition at line 49 of file WaveController.hpp.

### 14.29.4.2 textureholder\_

`TextureHolder& WaveController::textureholder_ [private]`

Definition at line 48 of file WaveController.hpp.

### 14.29.4.3 wave

`WavePtr WaveController::wave [private]`

Definition at line 51 of file WaveController.hpp.

### 14.29.4.4 waveNumber

`int WaveController::waveNumber [private]`

Definition at line 50 of file WaveController.hpp.

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

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



## 14.30 WavePause Class Reference

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

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

Inheritance diagram for WavePause:

Collaboration diagram for WavePause:

### Public Member Functions

- [WavePause](#) (const sf::Vector2f &position, [FontHolder](#) &fonts)  
*The constructor for the [WavePause](#) button.*
- void [update](#) ([World](#) &world, int enemiesNotSpawned)
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override
- void [handleInput](#) (const sf::Event &event, [World](#) &world)

### Private Attributes

- sf::Text [menuString\\_](#)
- sf::CircleShape [triangle\\_](#) = sf::CircleShape(40, 3)
- sf::RectangleShape [square\\_](#) = sf::RectangleShape(sf::Vector2f(100, 100))
- sf::Vector2< float > [mousePosition\\_](#) = sf::Vector2f(0.f, 0.f)

### 14.30.1 Detailed Description

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

Definition at line 13 of file WavePause.hpp.

### 14.30.2 Constructor & Destructor Documentation

#### 14.30.2.1 WavePause()

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

The constructor for the [WavePause](#) button.

Parameters

<i>position</i>	<a href="#">Button</a> coordinates
<i>fonts</i>	Reference to a FontHolder

Definition at line 6 of file WavePause.cpp.

```

6                                     :
7     mousePosition_(sf::Vector2f(0.f, 0.f)) {
8     //mousePosition_ = sf::V
9     menuString_.setFont(fonts.get(Fonts::GameTitleFont));
10    menuString_.setString("Pause Game");
11    menuString_.setCharacterSize(24);
12    menuString_.setFillColor(sf::Color::Black);
13    menuString_.setPosition(position);
14    menuString_.move(0.f, -40.f);
15
16    triangle_.setPosition(position);
17    triangle_.setOutlineThickness(5);
18    triangle_.setFillColor(sf::Color::Green);
19    triangle_.setOutlineColor(sf::Color::Black);
20    triangle_.setRotation(90.f);
21    triangle_.move(80.f, 12.f);
22    square_.setPosition(position);
23    square_.setOutlineThickness(3);
24    square_.setOutlineColor(sf::Color::Black);
25 }
```

### 14.30.3 Member Function Documentation

#### 14.30.3.1 draw()

```

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

Definition at line 30 of file WavePause.cpp.

```

30                                     {
31     target.draw(menuString_);
32     target.draw(square_);
33     target.draw(triangle_);
34 }
```

#### 14.30.3.2 handleInput()

```

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

Definition at line 35 of file WavePause.cpp.

```

35                                     {
36     // left click
37     if (event.type == sf::Event::MouseButtonPressed
38         && event.mouseButton.button == sf::Mouse::Button::Left) {
39         // inside square bounds
40         if (square_.getGlobalBounds().contains(
41             sf::Vector2f(event.mouseButton.x, event.mouseButton.y))) {
42             world.paused = !world.paused;
43             triangle_.setFillColor(world.paused ? sf::Color(200, 200, 200) : sf::Color::Green);
44             menuString_.setString(world.paused ? "Paused" : "Pause Game");
45         }
46     }
47 }
```

### 14.30.3.3 update()

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

Definition at line 26 of file WavePause.cpp.

```
26
27 //std::cout << "Is ready: " << world.isReadyForNextWave << "\n";
28 triangle_.setFillColor(sf::Color::Green);
29 }
```

## 14.30.4 Member Data Documentation

### 14.30.4.1 menuString\_

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

Definition at line 25 of file WavePause.hpp.

### 14.30.4.2 mousePosition\_

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

Definition at line 28 of file WavePause.hpp.

### 14.30.4.3 square\_

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

Definition at line 27 of file WavePause.hpp.

### 14.30.4.4 triangle\_

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

Definition at line 26 of file WavePause.hpp.

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

- [src/ui/WavePause.hpp](#)
- [src/ui/WavePause.cpp](#)

## 14.31 WaveStart Class Reference

A class for the button to start next wave.

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

Inheritance diagram for WaveStart:

Collaboration diagram for WaveStart:

### Public Member Functions

- [WaveStart](#) (const sf::Vector2f &position, [FontHolder](#) &fonts)  
*The constructor for the [WaveStart](#) button.*
- void [update](#) ([World](#) &world, int enemiesNotSpawned)
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const override
- void [handleInput](#) (const sf::Event &event, [World](#) &world)

### Private Attributes

- sf::Text [menuString\\_](#)
- sf::CircleShape [triangle\\_](#) = sf::CircleShape(40, 3)
- sf::RectangleShape [square\\_](#) = sf::RectangleShape(sf::Vector2f(100, 100))
- sf::Vector2< float > [mousePosition\\_](#) = sf::Vector2f(0.f, 0.f)

#### 14.31.1 Detailed Description

A class for the button to start next wave.

Definition at line 13 of file WaveStart.hpp.

#### 14.31.2 Constructor & Destructor Documentation

##### 14.31.2.1 WaveStart()

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

The constructor for the [WaveStart](#) button.

Parameters

<i>position</i>	<a href="#">Button</a> coordinates
<i>fonts</i>	Reference to a FontHolder

Definition at line 6 of file WaveStart.cpp.

```

6                                     :
7     mousePosition_(sf::Vector2f(0.f, 0.f)) {
8     //mousePosition_ = sf::V
9     menuString_.setFont(fonts.get(Fonts::GameTitleFont));
10    menuString_.setString("Start Wave");
11    menuString_.setCharacterSize(24);
12    menuString_.setFillColor(sf::Color::Black);
13    menuString_.setPosition(position);
14    menuString_.move(0.f, -40.f);
15
16    triangle_.setPosition(position);
17    triangle_.setOutlineThickness(5);
18    triangle_.setFillColor(sf::Color::Green);
19    triangle_.setOutlineColor(sf::Color::Black);
20    triangle_.setRotation(90.f);
21    triangle_.move(80.f, 12.f);
22    square_.setPosition(position);
23    square_.setOutlineThickness(3);
24    square_.setOutlineColor(sf::Color::Black);
25 }
```

### 14.31.3 Member Function Documentation

#### 14.31.3.1 draw()

```

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

Definition at line 36 of file WaveStart.cpp.

```

36                                     {
37     target.draw(menuString_);
38     target.draw(square_);
39     target.draw(triangle_);
40 }
```

#### 14.31.3.2 handleInput()

```

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

Definition at line 41 of file WaveStart.cpp.

```

41                                     {
42     // left click
43     if (event.type == sf::Event::MouseButtonPressed
44         && event.mouseButton.button == sf::Mouse::Button::Left) {
45         // inside square bounds
46         if (square_.getGlobalBounds().contains(
47             sf::Vector2f(event.mouseButton.x, event.mouseButton.y))) {
48             world.isReadyForNextWave = true;
49             menuString_.setString("Wave incoming...");
50         }
51     }
52 }
```

### 14.31.3.3 update()

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

Definition at line 26 of file WaveStart.cpp.

```
26
27 //std::cout << "Is ready: " << world.isReadyForNextWave << "\n";
28 if ((world.getEnemies().size() + enemiesNotSpawned) > 0) {
29     world.isReadyForNextWave = false;
30     triangle_.setFillColor(sf::Color(200, 200, 200));
31 } else {
32     triangle_.setFillColor(sf::Color::Green);
33     menuString_.setString("Start wave");
34 }
35 }
```

## 14.31.4 Member Data Documentation

### 14.31.4.1 menuString\_

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

Definition at line 25 of file WaveStart.hpp.

### 14.31.4.2 mousePosition\_

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

Definition at line 28 of file WaveStart.hpp.

### 14.31.4.3 square\_

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

Definition at line 27 of file WaveStart.hpp.

### 14.31.4.4 triangle\_

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

Definition at line 26 of file WaveStart.hpp.

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

- [src/ui/WaveStart.hpp](#)
- [src/ui/WaveStart.cpp](#)

## 14.32 World Class Reference

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

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

Inheritance diagram for World:

Collaboration diagram for World:

### Public Member Functions

- [World](#) ([TextureHolder](#) &textureholder, [SoundBufferHolder](#) &soundBufferHolder, int mapNum, std::map< int, std::pair< int, int >> &pathMarkers)  
*Constructor for world.*
- void [update](#) (sf::Time delta)  
*Update all enemies, towers and projectiles.*
- void [draw](#) (sf::RenderTarget &target, sf::RenderStates states) const  
*call other classes' draw functions. [Map](#), towers, enemies, house, and projectiles.*
- void [addProjectile](#) ([ProjectilePtr](#) &&projectile)  
*Add a projectile to the world (a tower shot one).*
- bool [addTower](#) ([TowerPtr](#) &&tower, int price)  
*Add a tower to the world (when player buys one)*
- bool [upgradeTower](#) ([Tower](#) \*tower, int price)  
*Upgrade a tower.*
- void [removeTower](#) ([Tower](#) \*tower, int price)  
*Find a tower and remove it.*
- [Enemies](#) & [getEnemies](#) ()  
*Get the enemies currently on the map.*
- int [getMoney](#) () const  
*Get the amount of money the player has.*
- int [getHP](#) () const  
*Get the player hp.*
- const [MapGrid](#) & [getMapGrid](#) () const  
*Get the current map's [MapGrid](#).*
- [Tower](#) \* [getTowerAt](#) (const [Sector](#) &target) const  
*Find a tower at given coordinates.*
- [TextureHolder](#) & [getTextures](#) ()  
*Get world TextureHolder.*
- [SoundBufferHolder](#) & [getSounds](#) ()  
*Get world SoundBufferHolder.*

### Public Attributes

- bool [isReadyForNextWave](#) = false  
*Is the player ready for next wave (pressed the button)?*
- bool [paused](#) = false  
*Is the game paused? If so, dont move anything.*

## Private Member Functions

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

## Private Attributes

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

### 14.32.1 Detailed Description

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

Definition at line 26 of file `World.hpp`.

### 14.32.2 Constructor & Destructor Documentation

#### 14.32.2.1 World()

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

Constructor for world.



## Parameters

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

Definition at line 12 of file World.cpp.

```

15                                     :
16     grid_(textureholder, mapNum, pathMarkers),
17     money_(startMoney),
18     hp_(startHP),
19     textures(textureholder),
20     sounds(soundBufferHolder),
21     mapNum_(mapNum),
22     towerGrid_() {
23     deathSound_.setBuffer(soundBufferHolder.get(SoundBuffers::EnemyDeath));
24     explosionSound_.setBuffer(soundBufferHolder.get(SoundBuffers::Explosion));
25     endHouse.setTexture(textures.get(Textures::HouseTile));
26     // x coords
27     float x = float(pathMarkers.rbegin()->second.first) * tileSize;
28     // y coords
29     float y = float(pathMarkers.rbegin()->second.second) * tileSize;
30     endHouse.setPosition(x, y);
31     endHouse.setScale(2.f, 2.f);
32     endHouse.move(-55.f, -64.f);
33 }
```

## 14.32.3 Member Function Documentation

### 14.32.3.1 addProjectile()

```

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

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

## Parameters

<i>projectile</i>	ProjectilePtr, pointer to the projectile to add
-------------------	-------------------------------------------------

## Returns

Definition at line 79 of file World.cpp.

```

79                                     {
80     projectiles.push_back(std::move(projectile));
81 }
```

### 14.32.3.2 addTower()

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

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

#### Parameters

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

Definition at line 83 of file World.cpp.

```
83                                     {
84     if (money_ >= price) {
85         money_ -= price;
86         //std::cout << "Money left: " << money_ << std::endl;
87         //std::cout << "HP left: " << hp_ << std::endl;
88         towers.push_back(std::move(tower));
89         return true;
90     } else
91         return false;
92 }
```

### 14.32.3.3 clean() [1/2]

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

Definition at line 205 of file World.hpp.

```
205                                     {
206     for (auto elem = vec.begin(); elem != vec.end(); ) {
207         if ((*elem)->ifShouldRemove()) {
208             elem = vec.erase(elem);
209         } else {
210             ++elem;
211         }
212     }
213 }
```

### 14.32.3.4 clean() [2/2]

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

Definition at line 192 of file World.hpp.

```
192                                     {
193     if (!vec.empty()) {
194         vec.erase(std::remove_if(vec.begin(),
195                                 vec.end(),
196                                 [](const std::unique_ptr<T> &entity) {
197                                     if (entity != nullptr) {
198                                         return entity->ifShouldRemove();
199                                     } else return false;
200                                 }), vec.end());
201     }
202 }
```

**14.32.3.5 draw()** [1/3]

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

Definition at line 183 of file World.hpp.

```
185                                     {
186     for (auto &&entity : vec) {
187         target.draw(*entity, states);
188     }
189 }
```

**14.32.3.6 draw()** [2/3]

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

Definition at line 175 of file World.hpp.

```
177                                     {
178     for (auto &&entity : vec) {
179         target.draw(*entity, states);
180     }
181 }
```

**14.32.3.7 draw()** [3/3]

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

call other classes' draw functions. [Map](#), towers, enemies, house, and projectiles.

**Parameters**

<i>target</i>	sf::RenderTarget to draw the scene to
<i>states</i>	sf::RenderStates object for drawing

Definition at line 71 of file World.cpp.

```
71                                     {
72     target.draw(grid_); //draw map
73     draw(towers, target, states); //draw towers
74     draw(enemies, target, states); // draw enemies
75     target.draw(endHouse);
76     draw(projectiles, target, states); // draw projectiles
77 }
```

#### 14.32.3.8 getEnemies()

```
Enemies& World::getEnemies ( ) [inline]
```

Get the enemies currently on the map.

##### Returns

A vector of enemies.

Definition at line 81 of file World.hpp.

```
81 { return enemies; }
```

#### 14.32.3.9 getHP()

```
int World::getHP ( ) const [inline]
```

Get the player hp.

##### Returns

integer, hp

Definition at line 91 of file World.hpp.

```
91 { return hp_; }
```

#### 14.32.3.10 getMapGrid()

```
const MapGrid& World::getMapGrid ( ) const [inline]
```

Get the current map's [MapGrid](#).

##### Returns

the map's [MapGrid](#) class instance

Definition at line 96 of file World.hpp.

```
96 { return grid_; }
```

#### 14.32.3.11 `getMoney()`

```
int World::getMoney ( ) const [inline]
```

Get the amount of money the player has.

##### Returns

integer, amount of money

Definition at line 86 of file World.hpp.

```
86 { return money_; }
```

#### 14.32.3.12 `getSounds()`

```
SoundBufferHolder& World::getSounds ( ) [inline]
```

Get world SoundBufferHolder.

##### Returns

reference to a SoundBufferHolder

Definition at line 112 of file World.hpp.

```
112 { return sounds; }
```

#### 14.32.3.13 `getTextures()`

```
TextureHolder& World::getTextures ( ) [inline]
```

Get world TextureHolder.

##### Returns

reference to a TextureHolder

Definition at line 107 of file World.hpp.

```
107 { return textures; }
```

#### 14.32.3.14 `getTowerAt()`

```
Tower * World::getTowerAt (
    const Sector & target ) const
```

Find a tower at given coordinates.

## Parameters

<i>target</i>	Target coordinates (using <a href="#">Sector</a> class)
---------------	---------------------------------------------------------

## Returns

Pointer to the tower

Definition at line 107 of file World.cpp.

```

107                                     {
108     auto found = std::find_if(
109         towers.cbegin(),
110         towers.cend(),
111         [&target](const TowerPtr &tower) { return tower->getSector() == target; }
112     );
113     return found == towers.cend() ? nullptr : &*found;
114 }
```

**14.32.3.15 removeTower()**

```

void World::removeTower (
    Tower * tower,
    int price )
```

Find a tower and remove it.

## Parameters

<i>tower</i>	A pointer to a tower that should be removed
<i>price</i>	how much money is returned to the player

Definition at line 94 of file World.cpp.

```

94                                     {
95     towers.erase(
96         std::find_if(
97             towers.begin(),
98             towers.end(),
99             [tower](const TowerPtr &other) { return other.get() == tower; }
100         )
101     );
102     money_ += price;
103     //std::cout << "Money left: " << money_ << std::endl;
104     //std::cout << "HP left: " << hp_ << std::endl;
105 }
```

**14.32.3.16 update() [1/3]**

```

template<typename T >
void World::update (
    const std::vector< std::shared_ptr< T >> & vec,
    sf::Time delta ) [inline], [private]
```

Definition at line 166 of file World.hpp.

```

166                                     {
```

```

167     for (auto &&entity : vec) {
168         if (entity != nullptr) {
169             entity->update(deltaTime, *this);
170         }
171     }
172 }

```

### 14.32.3.17 update() [2/3]

```

template<typename T >
void World::update (
    const std::vector< std::unique_ptr< T >> & vec,
    sf::Time delta ) [inline], [private]

```

Definition at line 157 of file World.hpp.

```

157     {
158     for (auto &&entity : vec) {
159         if (entity != nullptr) {
160             entity->update(deltaTime, *this);
161         }
162     }
163 }

```

### 14.32.3.18 update() [3/3]

```

void World::update (
    sf::Time delta )

```

Update all enemies, towers and projectiles.

#### Parameters

<i>delta</i>	time since last frame Updates positions of everything. If game is paused, stops enemies, towers and projectiles. Checks for dead enemies and those that have reached the cats' house.
--------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Definition at line 35 of file World.cpp.

```

35     {
36     if (!paused) {
37         //Update all enemies
38         update(enemies, deltaTime);
39         //Update all towers
40         update(towers, deltaTime);
41         //Update all projectiles
42         update(projectiles, deltaTime);
43     }
44     //Check which enemies are dead or at finish -> changes money amount
45     for (auto &&enemy : enemies) {
46         if (!enemy->isAlive()) {
47             deathSound_.play();
48             this->money_ += enemy->getValue() * 5;
49         }
50         if (enemy->isAtFinish()) {
51             this->hp_ -= enemy->getValue();
52         }
53     }
54
55     //Clean dead enemies from enemies vector
56     clean(enemies);
57
58     //Clean dead projectiles from projectiles vector
59     for (auto &&projectile : projectiles) {

```

```

60     if (projectile->type_ == ProjectileType::Bomb) {
61         if (projectile->ifShouldRemove()) {
62             explosionSound_.play();
63         }
64     }
65 }
66 clean(projectiles);
67
68 // std::cout << "Enemies: " << enemies.size() << " Towers: " << towers.size() << " Projectiles: " <<
    projectiles.size() << std::endl;
69 }

```

#### 14.32.3.19 upgradeTower()

```

bool World::upgradeTower (
    Tower * tower,
    int price )

```

Upgrade a tower.

##### Parameters

<i>tower</i>	<a href="#">Tower</a> to upgrade, calls its function for upgrading
<i>price</i>	How much the upgrade costs

##### Returns

Whether the upgrade succeeded (true if there was an upgrade available and the player had enough money)

Definition at line 115 of file World.cpp.

```

115 {
116     if (money_ >= price) {
117         if (tower->upgrade()) money_ -= price;
118         return true;
119     } else return false;
120 }

```

### 14.32.4 Member Data Documentation

#### 14.32.4.1 deathSound\_

```
sf::Sound World::deathSound_ [private]
```

Definition at line 125 of file World.hpp.

#### 14.32.4.2 endHouse

```
sf::Sprite World::endHouse [private]
```

Definition at line 123 of file World.hpp.



#### 14.32.4.3 enemies

```
Enemies World::enemies [private]
```

Definition at line 131 of file World.hpp.

#### 14.32.4.4 explosionSound\_

```
sf::Sound World::explosionSound_ [private]
```

Definition at line 126 of file World.hpp.

#### 14.32.4.5 grid\_

```
MapGrid World::grid_ [private]
```

Definition at line 127 of file World.hpp.

#### 14.32.4.6 hp\_

```
int World::hp_ [private]
```

Definition at line 129 of file World.hpp.

#### 14.32.4.7 isReadyForNextWave

```
bool World::isReadyForNextWave = false
```

Is the player ready for next wave (pressed the button)?

Definition at line 116 of file World.hpp.

#### 14.32.4.8 mapNum\_

```
int World::mapNum_ [private]
```

Definition at line 130 of file World.hpp.

#### 14.32.4.9 money\_

```
int World::money_ [private]
```

Definition at line 128 of file World.hpp.

#### 14.32.4.10 paused

```
bool World::paused = false
```

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

Definition at line 120 of file World.hpp.

#### 14.32.4.11 projectile

```
ProjectilePtr World::projectile [private]
```

Definition at line 134 of file World.hpp.

#### 14.32.4.12 projectiles

```
Projectiles World::projectiles [private]
```

Definition at line 133 of file World.hpp.

#### 14.32.4.13 sounds

```
SoundBufferHolder& World::sounds [private]
```

Definition at line 124 of file World.hpp.

#### 14.32.4.14 textures

```
TextureHolder& World::textures [private]
```

Definition at line 122 of file World.hpp.

**14.32.4.15 towerGrid\_**

`Grid` World::towerGrid\_ [private]

Definition at line 135 of file World.hpp.

**14.32.4.16 towers**

`Towers` World::towers [private]

Definition at line 132 of file World.hpp.

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

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



## Chapter 15

# File Documentation

### 15.1 doc/readme.md File Reference

### 15.2 libs/readme.md File Reference

### 15.3 plan/readme.md File Reference

### 15.4 src/readme.md File Reference

### 15.5 src/ui/readme.md File Reference

### 15.6 tests/readme.md File Reference

### 15.7 Meeting-notes.md File Reference

### 15.8 README.md File Reference

### 15.9 src/entity/Enemy.cpp File Reference

```
#include <cmath>
#include <memory>
#include "Enemy.hpp"
#include "SFML/System/Time.hpp"
Include dependency graph for Enemy.cpp:
```

## 15.10 src/entity/Enemy.hpp File Reference

```
#include "Entity.hpp"
#include "SFML/System/Time.hpp"
#include <iostream>
```

Include dependency graph for Enemy.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [Enemy](#)  
*A class for ingame enemies. Derived from [Entity](#) class.*

### Typedefs

- using [EnemyPtr](#) = std::shared\_ptr< [Enemy](#) >
- using [Enemies](#) = std::vector< [EnemyPtr](#) >

### 15.10.1 Typedef Documentation

#### 15.10.1.1 Enemies

```
using Enemies = std::vector<EnemyPtr>
```

Definition at line 14 of file Enemy.hpp.

#### 15.10.1.2 EnemyPtr

```
using EnemyPtr = std::shared_ptr<Enemy>
```

Definition at line 13 of file Enemy.hpp.

## 15.11 src/entity/Entity.cpp File Reference

```
#include "Entity.hpp"
#include <SFML/Graphics/RenderTarget.hpp>
```

Include dependency graph for Entity.cpp:

## 15.12 src/entity/Entity.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include <SFML/Graphics.hpp>
#include "SFML/Graphics/Sprite.hpp"
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include "resource/Resource.hpp"
#include <vector>
```

Include dependency graph for Entity.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [Entity](#)

*Visible entity on the map.*

## 15.13 src/entity/Projectile.cpp File Reference

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

Include dependency graph for Projectile.cpp:

## 15.14 src/entity/Projectile.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include "ui/Sector.hpp"
#include "Entity.hpp"
#include "Enemy.hpp"
#include <vector>
#include <memory>
```

Include dependency graph for Projectile.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [Projectile](#)

*Extends [Entity](#) class, Bombs, Bullets etc.*

### Typedefs

- using [ProjectilePtr](#) = std::unique\_ptr< [Projectile](#) >
- using [Projectiles](#) = std::vector< [ProjectilePtr](#) >

### Enumerations

- enum class [ProjectileType](#) { [Bullet](#) , [FreezeGun](#) , [Bomb](#) }

## 15.14.1 Typedef Documentation

### 15.14.1.1 ProjectilePtr

```
using ProjectilePtr = std::unique_ptr<Projectile>
```

Definition at line 25 of file Projectile.hpp.

### 15.14.1.2 Projectiles

```
using Projectiles = std::vector<ProjectilePtr>
```

Definition at line 26 of file Projectile.hpp.

## 15.14.2 Enumeration Type Documentation

### 15.14.2.1 ProjectileType

```
enum ProjectileType [strong]
```

Enumerator

Bullet	
FreezeGun	
Bomb	

Definition at line 18 of file Projectile.hpp.

```
18 {  
19     Bullet,  
20     FreezeGun,  
21     Bomb  
22 };
```

## 15.15 src/entity/Rat.cpp File Reference

## 15.16 src/entity/Rat.hpp File Reference

## 15.17 src/entity/Tower.cpp File Reference

```
#include "Tower.hpp"  
#include <SFML/Graphics/RenderTarget.hpp>
```



```
#include "game/World.hpp"
#include "SFML/System/Time.hpp"
#include "SFML/Audio/Sound.hpp"
Include dependency graph for Tower.cpp:
```

## Variables

- sf::Time [frameDelay](#) = sf::seconds(0.0167)

### 15.17.1 Variable Documentation

#### 15.17.1.1 frameDelay

```
sf::Time frameDelay = sf::seconds(0.0167)
```

Definition at line 7 of file Tower.cpp.

## 15.18 src/entity/Tower.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include "ui/Sector.hpp"
#include "Entity.hpp"
#include "Enemy.hpp"
#include "Projectile.hpp"
#include <vector>
#include <memory>
#include "SFML/Audio/Sound.hpp"
```

Include dependency graph for Tower.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [Tower](#)  
*[Tower](#) to display on the map and shoot, extends [Entity](#) class.*

## Typedefs

- using [TowerPtr](#) = std::unique\_ptr< [Tower](#) >
- using [Towers](#) = std::vector< [TowerPtr](#) >

## Enumerations

- enum class [TowerType](#) { [GunCat](#) , [FreezeCat](#) , [BombCat](#) }

## 15.18.1 Typedef Documentation

### 15.18.1.1 TowerPtr

```
using TowerPtr = std::unique_ptr<Tower>
```

Definition at line 27 of file Tower.hpp.

### 15.18.1.2 Towers

```
using Towers = std::vector<TowerPtr>
```

Definition at line 28 of file Tower.hpp.

## 15.18.2 Enumeration Type Documentation

### 15.18.2.1 TowerType

```
enum TowerType [strong]
```

Enumerator

GunCat	
FreezeCat	
BombCat	

Definition at line 20 of file Tower.hpp.

```
20         {  
21     GunCat,  
22     FreezeCat,  
23     BombCat  
24 };
```

## 15.19 src/game/Block.cpp File Reference

```
#include "Block.hpp"
```

Include dependency graph for Block.cpp:

## 15.20 src/game/Block.hpp File Reference

This graph shows which files directly or indirectly include this file:

## Classes

- class [Block](#)

## 15.21 src/game/Game.cpp File Reference

```
#include "Game.hpp"
#include <SFML/Graphics.hpp>
#include "MapGrid.hpp"
#include "World.hpp"
#include "MapScene.hpp"
#include "LevelSelect.hpp"
#include "GameTitle.hpp"
#include "GameEnd.hpp"
#include "resource/Resource.hpp"
#include <iostream>
```

Include dependency graph for Game.cpp:

## 15.22 src/game/Game.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "Scene.hpp"
#include "resource/Resource.hpp"
```

Include dependency graph for Game.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [Game](#)

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

## 15.23 src/game/GameEnd.cpp File Reference

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

Include dependency graph for GameEnd.cpp:

## 15.24 src/game/GameEnd.hpp File Reference

```
#include "Scene.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/Sprite.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
```

Include dependency graph for GameEnd.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [GameEnd](#)

*A class that inherits [Scene](#), is shown when the player loses the game.*

## 15.25 src/game/GameTitle.cpp File Reference

```
#include <iostream>
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include "GameTitle.hpp"
```

Include dependency graph for GameTitle.cpp:

## 15.26 src/game/GameTitle.hpp File Reference

```
#include "Scene.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
```

Include dependency graph for GameTitle.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [GameTitle](#)

*Welcome screen for the game.*

## 15.27 src/game/LevelSelect.cpp File Reference

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

Include dependency graph for LevelSelect.cpp:

## 15.28 src/game/LevelSelect.hpp File Reference

```
#include "Scene.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/Sprite.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
#include <fstream>
#include <iostream>
```

Include dependency graph for LevelSelect.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [LevelSelect](#)

*A class for level selection menu, inherits [Scene](#) class.*

## 15.29 src/game/Map.cpp File Reference

```
#include "Map.hpp"
```

Include dependency graph for Map.cpp:

## 15.30 src/game/Map.hpp File Reference

```
#include <string>
```

Include dependency graph for Map.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [Map](#)

## 15.31 src/game/MapGrid.cpp File Reference

```
#include "MapGrid.hpp"
#include "SFML/Graphics/Sprite.hpp"
#include "SFML/Graphics/Texture.hpp"
#include "SFML/Graphics/CircleShape.hpp"
#include <fstream>
#include <iostream>
```

Include dependency graph for MapGrid.cpp:

## 15.32 src/game/MapGrid.hpp File Reference

```
#include <vector>
#include "SFML/Graphics/RenderTarget.hpp"
#include "SFML/Graphics/Drawable.hpp"
#include "resource/Resource.hpp"
#include "SFML/Graphics/RenderTexture.hpp"
#include "SFML/Graphics/Sprite.hpp"
```

Include dependency graph for MapGrid.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [MapGrid](#)

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

## 15.33 src/game/MapScene.cpp File Reference

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

Include dependency graph for MapScene.cpp:

## 15.34 src/game/MapScene.hpp File Reference

```
#include "Scene.hpp"
#include "World.hpp"
#include "WaveController.hpp"
#include "ui/TowerMenu.hpp"
#include "ui/GameStatusMenu.hpp"
#include "ui/WaveStart.hpp"
#include "ui/WavePause.hpp"
#include "ui/GameCommandsMenu.hpp"
```

Include dependency graph for MapScene.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [MapScene](#)

*A class used when ingame. Inherits from [Scene](#) class. The main scene of the game.*

## 15.35 src/game/Scene.cpp File Reference

```
#include "Scene.hpp"
```

Include dependency graph for Scene.cpp:

## 15.36 src/game/Scene.hpp File Reference

```
#include <SFML/System/Time.hpp>
#include <SFML/Window/Event.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include "resource/Resource.hpp"
```

Include dependency graph for Scene.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- struct [request](#)
- class [Scene](#)

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

### Namespaces

- [Scenes](#)

### Typedefs

- typedef [request](#) [sceneRequest](#)
- using [ScenePtr](#) = std::unique\_ptr< [Scene](#) >

## Enumerations

- enum class [Scenes::ID](#) { [Scenes::GameTitle](#) , [Scenes::LevelSelect](#) , [Scenes::MapScene](#) , [Scenes::GameEnd](#) }

## 15.36.1 Typedef Documentation

### 15.36.1.1 ScenePtr

```
using ScenePtr = std::unique_ptr<Scene>
```

Definition at line 59 of file Scene.hpp.

### 15.36.1.2 sceneRequest

```
typedef request sceneRequest
```

Definition at line 29 of file Scene.hpp.

## 15.37 src/game/Wave.cpp File Reference

```
#include "Wave.hpp"
```

Include dependency graph for Wave.cpp:

## 15.38 src/game/Wave.hpp File Reference

```
#include "entity/Enemy.hpp"  
#include <SFML/System/Time.hpp>  
#include <iostream>
```

Include dependency graph for Wave.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [Wave](#)  
*A class for a single wave of enemies. A wave can be started by a player with a button.*

## Typedefs

- using [WavePtr](#) = std::unique\_ptr< [Wave](#) >

## 15.38.1 Typedef Documentation

### 15.38.1.1 WavePtr

```
using WavePtr = std::unique_ptr<Wave>
```

Definition at line 11 of file Wave.hpp.

## 15.39 src/game/WaveController.cpp File Reference

```
#include "WaveController.hpp"
#include <iostream>
Include dependency graph for WaveController.cpp:
```

### Variables

- sf::Time `delayTime` = sf::seconds(0.0167)

### 15.39.1 Variable Documentation

#### 15.39.1.1 delayTime

```
sf::Time delayTime = sf::seconds(0.0167)
```

Definition at line 8 of file WaveController.cpp.

## 15.40 src/game/WaveController.hpp File Reference

```
#include "World.hpp"
#include "Wave.hpp"
Include dependency graph for WaveController.hpp: This graph shows which files directly or indirectly include this file:
```

### Classes

- class `WaveController`  
*WaveController controls the current wave and makes a new one when the player is ready.*



## 15.41 src/game/World.cpp File Reference

```
#include "World.hpp"
#include "SFML/Graphics/RectangleShape.hpp"
#include "SFML/Graphics/CircleShape.hpp"
Include dependency graph for World.cpp:
```

## 15.42 src/game/World.hpp File Reference

```
#include <SFML/Graphics/RenderStates.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/System/Time.hpp>
#include <SFML/Graphics/Drawable.hpp>
#include <algorithm>
#include "ui/Sector.hpp"
#include "MapGrid.hpp"
#include "resource/Resource.hpp"
#include "entity/Enemy.hpp"
#include "entity/Tower.hpp"
#include "entity/Projectile.hpp"
#include "ui/Grid.hpp"
#include "SFML/Audio/Sound.hpp"
```

Include dependency graph for World.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [World](#)

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

## 15.43 src/main.cpp File Reference

```
#include <iostream>
#include "game/Game.hpp"
Include dependency graph for main.cpp:
```

### Functions

- int [main](#) ()

### 15.43.1 Function Documentation

### 15.43.1.1 main()

```
int main ( )
```

Definition at line 4 of file main.cpp.

```
4     {
5     Game newGame; //create game object
6     newGame.run(); //run the game
7     return 0;
8 }
```

## 15.44 src/resource/Resource.cpp File Reference

```
#include "Resource.hpp"
```

Include dependency graph for Resource.cpp:

## 15.45 src/resource/Resource.hpp File Reference

```
#include "string"
#include <SFML/System/Time.hpp>
#include "map"
#include <stdexcept>
#include <memory>
#include <cassert>
#include "SFML/Graphics/Texture.hpp"
#include "SFML/Graphics/Font.hpp"
#include "SFML/Audio/SoundBuffer.hpp"
```

Include dependency graph for Resource.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [ResourceHolder< Resource, Identifier >](#)

## Namespaces

- [Textures](#)
- [Maps](#)
- [Fonts](#)
- [SoundBuffers](#)

## Typedefs

- typedef [ResourceHolder< sf::Texture, Textures::ID >](#) [TextureHolder](#)
- typedef [ResourceHolder< std::string, Maps::ID >](#) [MapHolder](#)
- typedef [ResourceHolder< sf::Font, Fonts::ID >](#) [FontHolder](#)
- typedef [ResourceHolder< sf::SoundBuffer, SoundBuffers::ID >](#) [SoundBufferHolder](#)

## Enumerations

- enum `Textures::ID` {  
`Textures::PathTile` , `Textures::GrassTile` , `Textures::HouseTile` , `Textures::GunCat` ,  
`Textures::UpgradedGunCat` , `Textures::FreezeCat` , `Textures::UpgradedFzeezeCat` , `Textures::BombCat` ,  
`Textures::UpgradedBombCat` , `Textures::FatRat` , `Textures::FastRat` , `Textures::BasicRat` ,  
`Textures::Bullet` , `Textures::Bomb` , `Textures::Snowflake` , `Textures::PlayButton` ,  
`Textures::Explosion` }
- enum `Maps::ID` { `Maps::Map` }
- enum `Fonts::ID` { `Fonts::GameTitleFont` }
- enum `SoundBuffers::ID` {  
`SoundBuffers::EnemyDeath` , `SoundBuffers::Explosion` , `SoundBuffers::GunCat` , `SoundBuffers::BombCatMeow`  
, `SoundBuffers::FreezeCatMeow` }

### 15.45.1 Typedef Documentation

#### 15.45.1.1 FontHolder

```
typedef ResourceHolder<sf::Font, Fonts::ID> FontHolder
```

Definition at line 106 of file Resource.hpp.

#### 15.45.1.2 MapHolder

```
typedef ResourceHolder<std::string, Maps::ID> MapHolder
```

Definition at line 105 of file Resource.hpp.

#### 15.45.1.3 SoundBufferHolder

```
typedef ResourceHolder<sf::SoundBuffer, SoundBuffers::ID> SoundBufferHolder
```

Definition at line 107 of file Resource.hpp.

#### 15.45.1.4 TextureHolder

```
typedef ResourceHolder<sf::Texture, Textures::ID> TextureHolder
```

Definition at line 104 of file Resource.hpp.

## 15.46 src/sceneltem/Sceneltem.cpp File Reference

```
#include "SceneItem.hpp"
```

Include dependency graph for Sceneltem.cpp:

## 15.47 src/sceneltem/Sceneltem.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Include dependency graph for Sceneltem.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [Sceneltem](#)

## 15.48 src/ui/Button.cpp File Reference

```
#include "Button.hpp"
```

```
#include <SFML/Graphics/RenderTarget.hpp>
```

Include dependency graph for Button.cpp:

## 15.49 src/ui/Button.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
```

```
#include <SFML/Graphics/Sprite.hpp>
```

```
#include <SFML/Window/Event.hpp>
```

```
#include <vector>
```

```
#include <memory>
```

```
#include <SFML/Graphics/CircleShape.hpp>
```

Include dependency graph for Button.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [Button](#)

### Typedefs

- using [ButtonPtr](#) = std::unique\_ptr< [Button](#) >
- using [Buttons](#) = std::vector< [ButtonPtr](#) >

### 15.49.1 Typedef Documentation

### 15.49.1.1 ButtonPtr

```
using ButtonPtr = std::unique_ptr<Button>
```

Definition at line 29 of file Button.hpp.

### 15.49.1.2 Buttons

```
using Buttons = std::vector<ButtonPtr>
```

Definition at line 30 of file Button.hpp.

## 15.50 src/ui/GameCommandsMenu.cpp File Reference

```
#include "GameCommandsMenu.hpp"
```

Include dependency graph for GameCommandsMenu.cpp:

## 15.51 src/ui/GameCommandsMenu.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "resource/Resource.hpp"
#include "SFML/Graphics/Text.hpp"
#include "SFML/Graphics/Texture.hpp"
#include <SFML/Graphics/RenderStates.hpp>
#include "game/World.hpp"
```

Include dependency graph for GameCommandsMenu.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [GameCommandsMenu](#)

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

## 15.52 src/ui/GameStatusMenu.cpp File Reference

```
#include "GameStatusMenu.hpp"
#include <sstream>
```

Include dependency graph for GameStatusMenu.cpp:

## 15.53 src/ui/GameStatusMenu.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "game/World.hpp"
```

Include dependency graph for GameStatusMenu.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [GameStatusMenu](#)

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

## 15.54 src/ui/Grid.cpp File Reference

```
#include "Grid.hpp"
#include <SFML/Graphics/Sprite.hpp>
#include <SFML/Graphics/RenderTarget.hpp>
Include dependency graph for Grid.cpp:
```

## 15.55 src/ui/Grid.hpp File Reference

```
#include "Sector.hpp"
#include <vector>
#include <SFML/Graphics/Drawable.hpp>
#include "entity/Tower.hpp"
Include dependency graph for Grid.hpp: This graph shows which files directly or indirectly include this file:
```

## Classes

- class [Grid](#)

*[Grid](#) is used to place the towers according to the visual tiles.*

## 15.56 src/ui/Price.cpp File Reference

```
#include "Price.hpp"
Include dependency graph for Price.cpp:
```

## 15.57 src/ui/Price.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include <SFML/Graphics/Text.hpp>
Include dependency graph for Price.hpp: This graph shows which files directly or indirectly include this file:
```

## Classes

- class [Price](#)

## 15.58 src/ui/Sector.cpp File Reference

```
#include "Sector.hpp"
Include dependency graph for Sector.cpp:
```

## Functions

- [Sector operator+](#) (const [Sector](#) &lhs, const [Sector](#) &rhs)

### 15.58.1 Function Documentation

#### 15.58.1.1 operator+()

```
Sector operator+ (  
    const Sector & lhs,  
    const Sector & rhs )
```

Definition at line 7 of file Sector.cpp.

```
7  
8   return Sector{lhs.x + rhs.x, lhs.y + rhs.y};  
9 }
```

## 15.59 src/ui/Sector.hpp File Reference

```
#include <SFML/System/Vector2.hpp>  
#include <SFML/Graphics/Rect.hpp>  
#include <SFML/System/Time.hpp>  
#include <vector>  
#include <cmath>
```

Include dependency graph for Sector.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- struct [Sector](#)

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

## Typedefs

- using [Path](#) = std::vector< [Sector](#) >

## Functions

- [Sector operator+](#) (const [Sector](#) &lhs, const [Sector](#) &rhs)

### 15.59.1 Typedef Documentation

### 15.59.1.1 Path

```
using Path = std::vector<Sector>
```

Definition at line 27 of file Sector.hpp.

## 15.59.2 Function Documentation

### 15.59.2.1 operator+()

```
Sector operator+ (
    const Sector & lhs,
    const Sector & rhs )
```

Definition at line 7 of file Sector.cpp.

```
7
8     return Sector{lhs.x + rhs.x, lhs.y + rhs.y};
9 }
```

## 15.60 src/ui/SelectTowerButton.hpp File Reference

```
#include "Button.hpp"
#include <SFML/Graphics/CircleShape.hpp>
#include <SFML/Graphics/Text.hpp>
#include <memory>
#include "game/World.hpp"
#include "Sector.hpp"
```

Include dependency graph for SelectTowerButton.hpp:

### Classes

- class [SelectTowerButton< T >](#)

## 15.61 src/ui/TowerMenu.cpp File Reference

```
#include "TowerMenu.hpp"
#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/Window/Event.hpp>
#include "game/World.hpp"
```

Include dependency graph for TowerMenu.cpp:



## 15.62 src/ui/TowerMenu.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "Sector.hpp"
#include <SFML/Graphics/RectangleShape.hpp>
#include "entity/Tower.hpp"
```

Include dependency graph for TowerMenu.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [TowerMenu](#)

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

## 15.63 src/ui/WavePause.cpp File Reference

```
#include "WavePause.hpp"
```

Include dependency graph for WavePause.cpp:

## 15.64 src/ui/WavePause.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "game/World.hpp"
```

Include dependency graph for WavePause.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [WavePause](#)

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

## 15.65 src/ui/WaveStart.cpp File Reference

```
#include "WaveStart.hpp"
```

Include dependency graph for WaveStart.cpp:

## 15.66 src/ui/WaveStart.hpp File Reference

```
#include <SFML/Graphics/Drawable.hpp>
#include "game/World.hpp"
```

Include dependency graph for WaveStart.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [WaveStart](#)

*A class for the button to start next wave.*



# Index

- ~Button
  - Button, [36](#)
- ~TowerMenu
  - TowerMenu, [128](#)
- addProjectile
  - World, [149](#)
- addTower
  - World, [149](#)
- atFinish
  - Projectile, [105](#)
- atFinishTile
  - Enemy, [47](#)
- BasicRat
  - Textures, [34](#)
- Block, [35](#)
- blocks
  - Grid, [75](#)
- Bomb
  - Projectile.hpp, [164](#)
  - Textures, [34](#)
- BombCat
  - Textures, [34](#)
  - Tower.hpp, [166](#)
- BombCatMeow
  - SoundBuffers, [33](#)
- bombRange
  - Tower, [124](#)
- Bullet
  - Projectile.hpp, [164](#)
  - Textures, [34](#)
- Button, [35](#)
  - ~Button, [36](#)
  - Button, [36](#)
  - clicked, [38](#)
  - draw, [36](#)
  - handleInput, [37](#)
  - hasBeenClicked, [37](#)
  - icon, [38](#)
  - mouseHovers, [37](#)
  - mouseIn, [38](#)
  - onClick, [37](#)
  - sprite, [38](#)
  - update, [38](#)
- Button.hpp
  - ButtonPtr, [176](#)
  - Buttons, [177](#)
- ButtonPtr
  - Button.hpp, [176](#)
- Buttons
  - Button.hpp, [177](#)
- clean
  - World, [150](#)
- clicked
  - Button, [38](#)
- clock
  - Game, [55](#)
- clone
  - Enemy, [41](#)
- closestEnemies\_
  - Projectile, [105](#)
- closestEnemy\_
  - Projectile, [105](#)
- commandsMenu\_
  - MapScene, [94](#)
- Contains, [39](#)
- damage
  - Tower, [124](#)
- damage\_
  - Projectile, [105](#)
- deathSound\_
  - World, [156](#)
- delayTime
  - WaveController.cpp, [172](#)
- DiagVector
  - Sector, [115](#)
- diff
  - Map, [84](#)
  - MapGrid, [89](#)
- distance
  - Projectile, [105](#)
- distanceCovered
  - Projectile, [106](#)
- doc/readme.md, [161](#)
- draw
  - Button, [36](#)
  - Entity, [50](#)
  - GameCommandsMenu, [59](#)
  - GameEnd, [61](#)
  - GameStatusMenu, [66](#)
  - GameTitle, [69](#)
  - Grid, [74](#)
  - LevelSelect, [79](#)
  - MapGrid, [87](#)
  - MapScene, [92](#)
  - Price, [98](#)
  - SceneItem, [113](#)

- SelectTowerButton< T >, 117
- TowerMenu, 128
- WavePause, 142
- WaveStart, 145
- World, 150, 151
- elapsed\_
  - Wave, 134
- endHouse
  - World, 156
- Enemies
  - Enemy.hpp, 162
- enemies
  - World, 156
- enemiesLeft
  - Wave, 134
- Enemy, 39
  - atFinishTile, 47
  - clone, 41
  - Enemy, 40
  - getHitPoints, 42
  - getPathMarkers, 42
  - getSpeed, 42
  - getTextureHolder, 42
  - getValue, 43
  - hitPoints\_, 47
  - ifShouldRemove, 43
  - isAlive, 43
  - isAtFinish, 44
  - isFrozen, 47
  - isNotFrozen, 44
  - make, 44
  - nextMarker, 47
  - pathMarkers\_, 47
  - slowDown, 45
  - slowdownCounter\_, 47
  - slowdownFactor\_, 48
  - slowDownTime, 48
  - slowedDownAt\_, 48
  - speed\_, 48
  - takeDamage, 45
  - textures\_, 48
  - update, 45
  - value\_, 48
- Enemy.hpp
  - Enemies, 162
  - EnemyPtr, 162
- EnemyDeath
  - SoundBuffers, 33
- EnemyPtr
  - Enemy.hpp, 162
- enemyTemplate
  - Wave, 134
- Entity, 49
  - draw, 50
  - Entity, 49
  - entitySprite, 51
  - getPosition, 50
  - getTextureID, 50
  - mVelocity, 51
  - textureID\_, 51
  - update, 51
- entitySprite
  - Entity, 51
- Explosion
  - SoundBuffers, 33
  - Textures, 34
- explosionSound\_
  - World, 157
- FastRat
  - Textures, 34
- FatRat
  - Textures, 34
- finish\_
  - Projectile, 106
- FontHolder
  - Resource.hpp, 175
- Fonts, 31
  - GameTitleFont, 31
  - ID, 31
- fonts
  - Game, 55
- fonts\_
  - GameEnd, 64
  - GameTitle, 71
  - LevelSelect, 82
  - MapScene, 94
- fpsText
  - MapScene, 95
- frameDelay
  - Tower.cpp, 165
- FreezeCat
  - Textures, 33
  - Tower.hpp, 166
- FreezeCatMeow
  - SoundBuffers, 33
- FreezeGun
  - Projectile.hpp, 164
- fromCoords
  - Sector, 115
- Game, 52
  - clock, 55
  - fonts, 55
  - Game, 53
  - handleSceneChange, 53
  - load, 53
  - processEvents, 54
  - render, 54
  - renderStates, 56
  - run, 54
  - scene, 56
  - sounds, 56
  - textures, 56
  - Update, 55
  - videomode, 56
  - window, 57

- GameCommandsMenu, 57
  - draw, 59
  - GameCommandsMenu, 58
  - menuString\_, 59
  - towerIndexText\_, 59
  - towerSprites\_, 59
- GameEnd, 60
  - draw, 61
  - fonts\_, 64
  - GameEnd, 61
  - handleInput, 62
  - handlePlayerInput, 62
  - height, 64
  - nextScene\_, 64
  - optionIndex, 64
  - options, 64
  - requestedScene, 63
  - Scenes, 32
  - sceneType, 63
  - textGameOver\_, 65
  - textInstruction\_, 65
  - textures\_, 65
  - update, 63
  - width, 65
- GameStatusMenu, 66
  - draw, 66
  - GameStatusMenu, 66
  - menuString\_, 67
  - update, 67
- GameTitle, 68
  - draw, 69
  - fonts\_, 71
  - GameTitle, 69
  - grassSprite\_, 71
  - handleInput, 70
  - height, 72
  - mapSprite\_, 72
  - mapTex\_, 72
  - nextScene\_, 72
  - pathSprite, 72
  - requestedScene, 70
  - Scenes, 32
  - sceneType, 70
  - showText\_, 72
  - text2\_, 73
  - text\_, 73
  - textEffectTime\_, 73
  - textures\_, 73
  - update, 71
  - width, 73
- GameTitleFont
  - Fonts, 31
- get
  - ResourceHolder< Resource, Identifier >, 109
- getBlockAt
  - MapGrid, 88
- getClosestEnemies
  - Tower, 120
- getClosestEnemy
  - Tower, 120
- getEnemies
  - World, 151
- getEnemiesLeft
  - Wave, 133
- getHeight
  - MapGrid, 88
- getHitPoints
  - Enemy, 42
- getHP
  - World, 152
- getMapGrid
  - World, 152
- getMoney
  - World, 152
- getNeighbourEnemies
  - Tower, 121
- getPathMarkers
  - Enemy, 42
- getPosition
  - Entity, 50
- getPrice
  - Tower, 121
- getRange
  - Tower, 122
- getSector
  - Tower, 122
- getSelectedSector
  - Grid, 75
- getSounds
  - World, 153
- getSpeed
  - Enemy, 42
- getTextureHolder
  - Enemy, 42
- getTextureID
  - Entity, 50
- getTextures
  - World, 153
- getTowerAt
  - World, 153
- getValue
  - Enemy, 43
- getWaveEnemiesLeft
  - WaveController, 137
- getWaveNumber
  - WaveController, 137
- getWidth
  - MapGrid, 88
- grassSprite\_
  - GameTitle, 71
  - LevelSelect, 82
- GrassTile
  - Textures, 33
- Grid, 74
  - blocks, 75
  - draw, 74

- getSelectedSector, [75](#)
  - Grid, [74](#)
  - height, [75](#)
  - selectedSector, [75](#)
  - width, [76](#)
- grid\_
  - World, [157](#)
- GunCat
  - SoundBuffers, [33](#)
  - Textures, [33](#)
  - Tower.hpp, [166](#)
- handleInput
  - Button, [37](#)
  - GameEnd, [62](#)
  - GameTitle, [70](#)
  - LevelSelect, [79](#)
  - MapScene, [93](#)
  - Scene, [111](#)
  - TowerMenu, [129](#)
  - WavePause, [142](#)
  - WaveStart, [145](#)
- handlePlayerInput
  - GameEnd, [62](#)
  - LevelSelect, [79](#)
- handleSceneChange
  - Game, [53](#)
- hasBeenClicked
  - Button, [37](#)
- height
  - GameEnd, [64](#)
  - GameTitle, [72](#)
  - Grid, [75](#)
  - LevelSelect, [82](#)
  - MapGrid, [89](#)
  - MapScene, [95](#)
- hitPoints\_
  - Enemy, [47](#)
- HouseTile
  - Textures, [33](#)
- hoverIndicator
  - TowerMenu, [130](#)
- hp\_
  - World, [157](#)
- icon
  - Button, [38](#)
- ID
  - Fonts, [31](#)
  - Maps, [31](#)
  - Scenes, [32](#)
  - SoundBuffers, [33](#)
  - Textures, [33](#)
- ifNextEnemy
  - Wave, [133](#)
- ifShouldRemove
  - Enemy, [43](#)
  - Projectile, [101](#)
- inRange
  - Tower, [122](#)
- isAlive
  - Enemy, [43](#)
- isAtFinish
  - Enemy, [44](#)
- isEmpty
  - Wave, [133](#)
- isFrozen
  - Enemy, [47](#)
- isNotFrozen
  - Enemy, [44](#)
- isReadyForNextWave
  - World, [157](#)
- LevelSelect, [76](#)
  - draw, [79](#)
  - fonts\_, [82](#)
  - grassSprite\_, [82](#)
  - handleInput, [79](#)
  - handlePlayerInput, [79](#)
  - height, [82](#)
  - LevelSelect, [77](#)
  - mapSprite\_, [82](#)
  - mapTex\_, [82](#)
  - nextScene\_, [82](#)
  - optionIndex, [83](#)
  - options, [83](#)
  - pathSprite, [83](#)
  - requestedScene, [80](#)
  - Scenes, [32](#)
  - sceneType, [80](#)
  - showText\_, [83](#)
  - text\_, [83](#)
  - textEffectTime\_, [83](#)
  - textures\_, [84](#)
  - update, [81](#)
  - updateOptionText, [81](#)
  - width, [84](#)
- libs/readme.md, [161](#)
- load
  - Game, [53](#)
  - ResourceHolder< Resource, Identifier >, [109](#)
- main
  - main.cpp, [173](#)
- main.cpp
  - main, [173](#)
- make
  - Enemy, [44](#)
  - Projectile, [101](#)
- makeNewWave
  - WaveController, [137](#)
- Map, [84](#)
  - diff, [84](#)
  - Maps, [32](#)
  - name, [85](#)
- map
  - MapGrid, [89](#)
- MapGrid, [85](#)

- diff, [89](#)
- draw, [87](#)
- getBlockAt, [88](#)
- getHeight, [88](#)
- getWidth, [88](#)
- height, [89](#)
- map, [89](#)
- MapGrid, [86](#)
- mapNum\_, [89](#)
- mapSprite\_, [89](#)
- mapTex\_, [90](#)
- name, [90](#)
- pathMarkers\_, [90](#)
- setBlockRow, [88](#)
- textures\_, [90](#)
- width, [90](#)
- MapHolder
  - Resource.hpp, [175](#)
- mapNum\_
  - MapGrid, [89](#)
  - MapScene, [95](#)
  - World, [157](#)
- Maps, [31](#)
  - ID, [31](#)
  - Map, [32](#)
- MapScene, [91](#)
  - commandsMenu\_, [94](#)
  - draw, [92](#)
  - fonts\_, [94](#)
  - fpsText, [95](#)
  - handleInput, [93](#)
  - height, [95](#)
  - mapNum\_, [95](#)
  - MapScene, [92](#)
  - nextScene\_, [95](#)
  - pathMarkers, [95](#)
  - requestedScene, [93](#)
  - Scenes, [32](#)
  - sceneType, [93](#)
  - soundBuffers\_, [95](#)
  - statusMenu\_, [96](#)
  - textures\_, [96](#)
  - towerMenu\_, [96](#)
  - update, [94](#)
  - waveController\_, [96](#)
  - wavePause\_, [96](#)
  - waveStart\_, [96](#)
  - width, [97](#)
  - world\_, [97](#)
- mapSprite\_
  - GameTitle, [72](#)
  - LevelSelect, [82](#)
  - MapGrid, [89](#)
- mapTex\_
  - GameTitle, [72](#)
  - LevelSelect, [82](#)
  - MapGrid, [90](#)
- Meeting-notes.md, [161](#)
- menuString\_
  - GameCommandsMenu, [59](#)
  - GameStatusMenu, [67](#)
  - WavePause, [143](#)
  - WaveStart, [146](#)
- money\_
  - World, [157](#)
- mouseHovers
  - Button, [37](#)
- mouseIn
  - Button, [38](#)
- mousePosition\_
  - WavePause, [143](#)
  - WaveStart, [146](#)
- mResourceMap
  - ResourceHolder< Resource, Identifier >, [110](#)
- mVelocity
  - Entity, [51](#)
- name
  - Map, [85](#)
  - MapGrid, [90](#)
- neighbourEnemies\_
  - Projectile, [106](#)
- nextMarker
  - Enemy, [47](#)
- nextScene\_
  - GameEnd, [64](#)
  - GameTitle, [72](#)
  - LevelSelect, [82](#)
  - MapScene, [95](#)
- number
  - request, [108](#)
- onClick
  - Button, [37](#)
  - SelectTowerButton< T >, [117](#)
- operator+
  - Sector.cpp, [179](#)
  - Sector.hpp, [180](#)
- operator==
  - Sector, [115](#)
- optionIndex
  - GameEnd, [64](#)
  - LevelSelect, [83](#)
- options
  - GameEnd, [64](#)
  - LevelSelect, [83](#)
- Path
  - Sector.hpp, [179](#)
- pathMarkers
  - MapScene, [95](#)
- pathMarkers\_
  - Enemy, [47](#)
  - MapGrid, [90](#)
  - WaveController, [140](#)
- pathSprite
  - GameTitle, [72](#)

- LevelSelect, [83](#)
- PathTile
  - Textures, [33](#)
- paused
  - World, [158](#)
- plan/readme.md, [161](#)
- PlayButton
  - Textures, [34](#)
- Price, [97](#)
  - draw, [98](#)
  - Price, [98](#)
  - price, [98](#)
  - valueToString, [98](#)
- price
  - Price, [98](#)
  - Tower, [125](#)
- processEvents
  - Game, [54](#)
- Projectile, [99](#)
  - atFinish, [105](#)
  - closestEnemies\_, [105](#)
  - closestEnemy\_, [105](#)
  - damage\_, [105](#)
  - distance, [105](#)
  - distanceCovered, [106](#)
  - finish\_, [106](#)
  - ifShouldRemove, [101](#)
  - make, [101](#)
  - neighbourEnemies\_, [106](#)
  - Projectile, [100](#)
  - range\_, [106](#)
  - start\_, [106](#)
  - textures\_, [106](#)
  - type\_, [107](#)
  - update, [102](#)
  - velocity\_, [107](#)
- projectile
  - World, [158](#)
- Projectile.hpp
  - Bomb, [164](#)
  - Bullet, [164](#)
  - FreezeGun, [164](#)
  - ProjectilePtr, [164](#)
  - Projectiles, [164](#)
  - ProjectileType, [164](#)
- ProjectilePtr
  - Projectile.hpp, [164](#)
- Projectiles
  - Projectile.hpp, [164](#)
- projectiles
  - World, [158](#)
- ProjectileType, [107](#)
  - Projectile.hpp, [164](#)
- Ptr
  - SceneItem, [113](#)
- range
  - Tower, [125](#)
- range\_
  - Projectile, [106](#)
  - rangeIndicator
    - TowerMenu, [130](#)
  - README.md, [161](#)
  - removeTower
    - World, [154](#)
  - render
    - Game, [54](#)
  - renderStates
    - Game, [56](#)
  - request, [107](#)
    - number, [108](#)
    - scene, [108](#)
  - requestedScene
    - GameEnd, [63](#)
    - GameTitle, [70](#)
    - LevelSelect, [80](#)
    - MapScene, [93](#)
    - Scene, [111](#)
  - Resource.hpp
    - FontHolder, [175](#)
    - MapHolder, [175](#)
    - SoundBufferHolder, [175](#)
    - TextureHolder, [175](#)
  - ResourceHolder< Resource, Identifier >, [108](#)
    - get, [109](#)
    - load, [109](#)
    - mResourceMap, [110](#)
  - run
    - Game, [54](#)
  - scale
    - Sector, [115](#)
  - Scene, [110](#)
    - handleInput, [111](#)
    - requestedScene, [111](#)
    - sceneType, [111](#)
    - textures, [112](#)
    - update, [112](#)
  - scene
    - Game, [56](#)
    - request, [108](#)
  - Scene.hpp
    - ScenePtr, [171](#)
    - sceneRequest, [171](#)
  - SceneItem, [112](#)
    - draw, [113](#)
    - Ptr, [113](#)
    - SceneItem, [113](#)
  - ScenePtr
    - Scene.hpp, [171](#)
  - sceneRequest
    - Scene.hpp, [171](#)
  - Scenes, [32](#)
    - GameEnd, [32](#)
    - GameTitle, [32](#)
    - ID, [32](#)
    - LevelSelect, [32](#)
    - MapScene, [32](#)



- sceneType
  - GameEnd, [63](#)
  - GameTitle, [70](#)
  - LevelSelect, [80](#)
  - MapScene, [93](#)
  - Scene, [111](#)
- Sector, [114](#)
  - DiagVector, [115](#)
  - fromCoords, [115](#)
  - operator==, [115](#)
  - scale, [115](#)
  - Size, [116](#)
  - upperLeftPoint, [115](#)
  - x, [116](#)
  - y, [116](#)
- sector
  - Tower, [125](#)
- Sector.cpp
  - operator+, [179](#)
- Sector.hpp
  - operator+, [180](#)
  - Path, [179](#)
- sectorIndicator
  - TowerMenu, [130](#)
- selectedSector
  - Grid, [75](#)
  - TowerMenu, [130](#)
- SelectTowerButton
  - SelectTowerButton< T >, [117](#)
- SelectTowerButton< T >, [116](#)
  - draw, [117](#)
  - onClick, [117](#)
  - SelectTowerButton, [117](#)
  - towerSector, [118](#)
- setBlockRow
  - MapGrid, [88](#)
- shootDelay
  - Tower, [125](#)
- showSectorIndicator
  - TowerMenu, [131](#)
- showText\_
  - GameTitle, [72](#)
  - LevelSelect, [83](#)
- Size
  - Sector, [116](#)
- slowDown
  - Enemy, [45](#)
- slowdownCounter\_
  - Enemy, [47](#)
- slowdownFactor\_
  - Enemy, [48](#)
- slowDownTime
  - Enemy, [48](#)
- slowedDownAt\_
  - Enemy, [48](#)
- Snowflake
  - Textures, [34](#)
- soundBomb\_
  - Tower, [125](#)
- SoundBufferHolder
  - Resource.hpp, [175](#)
- SoundBuffers, [32](#)
  - BombCatMeow, [33](#)
  - EnemyDeath, [33](#)
  - Explosion, [33](#)
  - FreezeCatMeow, [33](#)
  - GunCat, [33](#)
  - ID, [33](#)
- soundBuffers\_
  - MapScene, [95](#)
- soundBullet\_
  - Tower, [125](#)
- sounds
  - Game, [56](#)
  - World, [158](#)
- sounds\_
  - Tower, [126](#)
- soundSnow\_
  - Tower, [126](#)
- spacing\_
  - Wave, [134](#)
- SPACING\_HUGE
  - Wave, [134](#)
- SPACING\_MEDIUM
  - Wave, [135](#)
- SPACING\_NARROW
  - Wave, [135](#)
- SPACING\_WIDE
  - Wave, [135](#)
- speed\_
  - Enemy, [48](#)
- sprite
  - Button, [38](#)
- square\_
  - WavePause, [143](#)
  - WaveStart, [146](#)
- src/entity/Enemy.cpp, [161](#)
- src/entity/Enemy.hpp, [162](#)
- src/entity/Entity.cpp, [162](#)
- src/entity/Entity.hpp, [163](#)
- src/entity/Projectile.cpp, [163](#)
- src/entity/Projectile.hpp, [163](#)
- src/entity/Rat.cpp, [164](#)
- src/entity/Rat.hpp, [164](#)
- src/entity/Tower.cpp, [164](#)
- src/entity/Tower.hpp, [165](#)
- src/game/Block.cpp, [166](#)
- src/game/Block.hpp, [166](#)
- src/game/Game.cpp, [167](#)
- src/game/Game.hpp, [167](#)
- src/game/GameEnd.cpp, [167](#)
- src/game/GameEnd.hpp, [167](#)
- src/game/GameTitle.cpp, [168](#)
- src/game/GameTitle.hpp, [168](#)
- src/game/LevelSelect.cpp, [168](#)
- src/game/LevelSelect.hpp, [168](#)

- src/game/Map.cpp, 169
- src/game/Map.hpp, 169
- src/game/MapGrid.cpp, 169
- src/game/MapGrid.hpp, 169
- src/game/MapScene.cpp, 169
- src/game/MapScene.hpp, 170
- src/game/Scene.cpp, 170
- src/game/Scene.hpp, 170
- src/game/Wave.cpp, 171
- src/game/Wave.hpp, 171
- src/game/WaveController.cpp, 172
- src/game/WaveController.hpp, 172
- src/game/World.cpp, 173
- src/game/World.hpp, 173
- src/main.cpp, 173
- src/readme.md, 161
- src/resource/Resource.cpp, 174
- src/resource/Resource.hpp, 174
- src/sceneltem/Sceneltem.cpp, 176
- src/sceneltem/Sceneltem.hpp, 176
- src/ui/Button.cpp, 176
- src/ui/Button.hpp, 176
- src/ui/GameCommandsMenu.cpp, 177
- src/ui/GameCommandsMenu.hpp, 177
- src/ui/GameStatusMenu.cpp, 177
- src/ui/GameStatusMenu.hpp, 177
- src/ui/Grid.cpp, 178
- src/ui/Grid.hpp, 178
- src/ui/Price.cpp, 178
- src/ui/Price.hpp, 178
- src/ui/readme.md, 161
- src/ui/Sector.cpp, 178
- src/ui/Sector.hpp, 179
- src/ui/SelectTowerButton.hpp, 180
- src/ui/TowerMenu.cpp, 180
- src/ui/TowerMenu.hpp, 181
- src/ui/WavePause.cpp, 181
- src/ui/WavePause.hpp, 181
- src/ui/WaveStart.cpp, 181
- src/ui/WaveStart.hpp, 181
- start\_
  - Projectile, 106
- statusMenu\_
  - MapScene, 96
- takeDamage
  - Enemy, 45
- testii
  - Tower, 126
- tests/readme.md, 161
- text2\_
  - GameTitle, 73
- text\_
  - GameTitle, 73
  - LevelSelect, 83
- textEffectTime\_
  - GameTitle, 73
  - LevelSelect, 83
- textGameOver\_
  - GameEnd, 65
- textInstruction\_
  - GameEnd, 65
- TextureHolder
  - Resource.hpp, 175
- textureholder\_
  - WaveController, 140
- textureID\_
  - Entity, 51
- Textures, 33
  - BasicRat, 34
  - Bomb, 34
  - BombCat, 34
  - Bullet, 34
  - Explosion, 34
  - FastRat, 34
  - FatRat, 34
  - FreezeCat, 33
  - GrassTile, 33
  - GunCat, 33
  - HouseTile, 33
  - ID, 33
  - PathTile, 33
  - PlayButton, 34
  - Snowflake, 34
  - UpgradedBombCat, 34
  - UpgradedFzeezeCat, 33
  - UpgradedGunCat, 33
- textures
  - Game, 56
  - Scene, 112
  - World, 158
- textures\_
  - Enemy, 48
  - GameEnd, 65
  - GameTitle, 73
  - LevelSelect, 84
  - MapGrid, 90
  - MapScene, 96
  - Projectile, 106
  - Tower, 126
- timeSinceFiring
  - Tower, 126
- Tower, 118
  - bombRange, 124
  - damage, 124
  - getClosestEnemies, 120
  - getClosestEnemy, 120
  - getNeighbourEnemies, 121
  - getPrice, 121
  - getRange, 122
  - getSector, 122
  - inRange, 122
  - price, 125
  - range, 125
  - sector, 125
  - shootDelay, 125
  - soundBomb\_, 125

- soundBullet\_, 125
- sounds\_, 126
- soundSnow\_, 126
- testii, 126
- textures\_, 126
- timeSinceFiring, 126
- Tower, 119
- type, 126
- update, 123
- upgrade, 124
- upgradeable, 127
- velocity\_, 127
- Tower.cpp
  - frameDelay, 165
- Tower.hpp
  - BombCat, 166
  - FreezeCat, 166
  - GunCat, 166
  - TowerPtr, 166
  - Towers, 166
  - TowerType, 166
- towerGrid\_
  - World, 158
- towerIndexText\_
  - GameCommandsMenu, 59
- TowerMenu, 127
  - ~TowerMenu, 128
  - draw, 128
  - handleInput, 129
  - hoverIndicator, 130
  - rangeIndicator, 130
  - sectorIndicator, 130
  - selectedSector, 130
  - showSectorIndicator, 131
  - TowerMenu, 128
  - update, 130
- towerMenu\_
  - MapScene, 96
- TowerPtr
  - Tower.hpp, 166
- Towers
  - Tower.hpp, 166
- towers
  - World, 159
- towerSector
  - SelectTowerButton< T >, 118
- towerSprites\_
  - GameCommandsMenu, 59
- TowerType, 131
  - Tower.hpp, 166
- triangle\_
  - WavePause, 143
  - WaveStart, 146
- type
  - Tower, 126
- type\_
  - Projectile, 107
- Update
  - Game, 55
- update
  - Button, 38
  - Enemy, 45
  - Entity, 51
  - GameEnd, 63
  - GameStatusMenu, 67
  - GameTitle, 71
  - LevelSelect, 81
  - MapScene, 94
  - Projectile, 102
  - Scene, 112
  - Tower, 123
  - TowerMenu, 130
  - WaveController, 139
  - WavePause, 142
  - WaveStart, 145
  - World, 154, 155
- updateOptionText
  - LevelSelect, 81
- upgrade
  - Tower, 124
- upgradeable
  - Tower, 127
- UpgradedBombCat
  - Textures, 34
- UpgradedFzeezeCat
  - Textures, 33
- UpgradedGunCat
  - Textures, 33
- upgradeTower
  - World, 156
- upperLeftPoint
  - Sector, 115
- value\_
  - Enemy, 48
- valueToString
  - Price, 98
- velocity\_
  - Projectile, 107
  - Tower, 127
- videomode
  - Game, 56
- Wave, 131
  - elapsed\_, 134
  - enemiesLeft, 134
  - enemyTemplate, 134
  - getEnemiesLeft, 133
  - ifNextEnemy, 133
  - isEmpty, 133
  - spacing\_, 134
  - SPACING\_HUGE, 134
  - SPACING\_MEDIUM, 135
  - SPACING\_NARROW, 135
  - SPACING\_WIDE, 135
  - Wave, 132
- wave

- WaveController, 140
- Wave.hpp
  - WavePtr, 172
- WaveController, 135
  - getWaveEnemiesLeft, 137
  - getWaveNumber, 137
  - makeNewWave, 137
  - pathMarkers\_, 140
  - textureholder\_, 140
  - update, 139
  - wave, 140
  - WaveController, 136
  - waveNumber, 140
- WaveController.cpp
  - delayTime, 172
- waveController\_
  - MapScene, 96
- waveNumber
  - WaveController, 140
- WavePause, 141
  - draw, 142
  - handleInput, 142
  - menuString\_, 143
  - mousePosition\_, 143
  - square\_, 143
  - triangle\_, 143
  - update, 142
  - WavePause, 141
- wavePause\_
  - MapScene, 96
- WavePtr
  - Wave.hpp, 172
- WaveStart, 144
  - draw, 145
  - handleInput, 145
  - menuString\_, 146
  - mousePosition\_, 146
  - square\_, 146
  - triangle\_, 146
  - update, 145
  - WaveStart, 144
- waveStart\_
  - MapScene, 96
- width
  - GameEnd, 65
  - GameTitle, 73
  - Grid, 76
  - LevelSelect, 84
  - MapGrid, 90
  - MapScene, 97
- window
  - Game, 57
- World, 147
  - addProjectile, 149
  - addTower, 149
  - clean, 150
  - deathSound\_, 156
  - draw, 150, 151
  - endHouse, 156
  - enemies, 156
  - explosionSound\_, 157
  - getEnemies, 151
  - getHP, 152
  - getMapGrid, 152
  - getMoney, 152
  - getSounds, 153
  - getTextures, 153
  - getTowerAt, 153
  - grid\_, 157
  - hp\_, 157
  - isReadyForNextWave, 157
  - mapNum\_, 157
  - money\_, 157
  - paused, 158
  - projectile, 158
  - projectiles, 158
  - removeTower, 154
  - sounds, 158
  - textures, 158
  - towerGrid\_, 158
  - towers, 159
  - update, 154, 155
  - upgradeTower, 156
  - World, 148
- world\_
  - MapScene, 97
- x
  - Sector, 116
- y
  - Sector, 116