

R-Type

Generated by Doxygen 1.9.1

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	9
4.1 File List	9
5 Namespace Documentation	13
5.1 r_type Namespace Reference	13
5.2 r_type::net Namespace Reference	13
6 Class Documentation	15
6.1 AbstractScenes Class Reference	15
6.1.1 Detailed Description	15
6.2 r_type::net::AClient< T > Class Template Reference	15
6.2.1 Constructor & Destructor Documentation	16
6.2.1.1 AClient()	16
6.2.1.2 ~AClient()	16
6.2.2 Member Function Documentation	17
6.2.2.1 Connect()	17
6.2.2.2 Disconnect()	17
6.2.2.3 getConnection()	17
6.2.2.4 getPlayerId()	18
6.2.2.5 getWindowSize()	18
6.2.2.6 Incoming()	18
6.2.2.7 isConnected()	18
6.2.2.8 Send()	18
6.2.2.9 setPlayerId()	19
6.2.2.10 setWindowSize()	19
6.2.3 Member Data Documentation	19
6.2.3.1 m_connection	19
6.2.3.2 m_context	19
6.2.3.3 m_qMessagesIn	19
6.2.3.4 playerId	20
6.2.3.5 thrContext	20
6.2.3.6 windowSize	20
6.3 AllyComponent Struct Reference	20
6.4 AllyMissileComponent Struct Reference	20
6.5 AnimationComponent Struct Reference	20

6.5.1 Detailed Description	21
6.5.2 Constructor & Destructor Documentation	21
6.5.2.1 AnimationComponent()	21
6.5.3 Member Data Documentation	21
6.5.3.1 dimension	21
6.5.3.2 offset	22
6.6 AnimationSystem Class Reference	22
6.6.1 Detailed Description	23
6.6.2 Constructor & Destructor Documentation	23
6.6.2.1 AnimationSystem()	23
6.6.3 Member Function Documentation	23
6.6.3.1 animateBasicMonster()	23
6.6.3.2 animateForceMissile()	24
6.6.3.3 animateForceWeapon()	24
6.6.3.4 animatePlayer()	24
6.6.3.5 AnimationEntities()	25
6.6.4 Member Data Documentation	25
6.6.4.1 _componentManager	26
6.6.4.2 _entityManager	26
6.7 AScenes Class Reference	26
6.7.1 Member Enumeration Documentation	28
6.7.1.1 Actions	28
6.7.1.2 DaltonismMode	29
6.7.1.3 GameMode	29
6.7.1.4 Scene	29
6.7.1.5 SpriteType	30
6.7.2 Constructor & Destructor Documentation	30
6.7.2.1 AScenes()	30
6.7.2.2 ~AScenes()	31
6.7.3 Member Function Documentation	31
6.7.3.1 getDaltonism()	31
6.7.3.2 getDisplayDaltonismChoice()	31
6.7.3.3 getDisplayGameModeChoice()	31
6.7.3.4 getDisplayKeyBindsChoice()	32
6.7.3.5 getIp()	32
6.7.3.6 getPort()	32
6.7.3.7 getPreviousScene()	32
6.7.3.8 setDaltonism()	32
6.7.3.9 setDisplayDaltonismChoice()	33
6.7.3.10 setDisplayGameModeChoice()	33
6.7.3.11 setDisplayKeyBindsChoice()	33
6.7.3.12 setGameMode()	34

6.7.3.13 setIp()	34
6.7.3.14 setPort()	34
6.7.3.15 setScene()	35
6.7.4 Member Data Documentation	35
6.7.4.1 _currentDaltonismMode	35
6.7.4.2 _currentGameMode	35
6.7.4.3 _currentScene	36
6.7.4.4 _displayDaltonismChoice	36
6.7.4.5 _displayGameModeChoice	36
6.7.4.6 _displayKeyBindsChoice	36
6.7.4.7 _ip	36
6.7.4.8 _port	36
6.7.4.9 _previousScene	37
6.7.4.10 buttons	37
6.7.4.11 filter	37
6.7.4.12 keyBinds	37
6.8 r_type::net::AServer< T > Class Template Reference	38
6.8.1 Detailed Description	40
6.8.2 Constructor & Destructor Documentation	40
6.8.2.1 AServer()	40
6.8.2.2 ~AServer()	41
6.8.3 Member Function Documentation	41
6.8.3.1 FormatEntityInformation()	41
6.8.3.2 getClientById()	42
6.8.3.3 GetClientInfoBarId()	42
6.8.3.4 GetClientPlayerId()	42
6.8.3.5 GetClock()	42
6.8.3.6 GetComponentManager()	43
6.8.3.7 GetEntityFactory()	43
6.8.3.8 GetEntityManager()	44
6.8.3.9 GetPlayerClientId()	44
6.8.3.10 InitiateEnemyMissile()	44
6.8.3.11 InitiatePlayer()	45
6.8.3.12 InitiatePlayerMissile()	45
6.8.3.13 InitiateWeaponForce()	45
6.8.3.14 InitInfoBar()	46
6.8.3.15 MessageAllClients()	46
6.8.3.16 MessageClient()	46
6.8.3.17 OnClientConnect()	47
6.8.3.18 OnClientDisconnect()	47
6.8.3.19 OnClientValidated()	47
6.8.3.20 OnMessage()	48

6.8.3.21 RemoveEntity()	48
6.8.3.22 RemoveInfoBar()	48
6.8.3.23 RemovePlayer()	49
6.8.3.24 SavePlayerScore()	49
6.8.3.25 SetClock()	50
6.8.3.26 Start()	50
6.8.3.27 Stop()	50
6.8.3.28 Update()	51
6.8.3.29 UpdateInfoBar()	51
6.8.3.30 UpdatePlayerPosition()	52
6.8.3.31 WaitForClientMessage()	52
6.8.4 Member Data Documentation	52
6.8.4.1 _asioContext	52
6.8.4.2 _asioSocket	53
6.8.4.3 _background	53
6.8.4.4 _clientEndpoint	53
6.8.4.5 _clientInfoBarID	53
6.8.4.6 _clientPlayerID	53
6.8.4.7 _clock	54
6.8.4.8 _componentManager	54
6.8.4.9 _deqConnections	54
6.8.4.10 _entityFactory	54
6.8.4.11 _entityManager	55
6.8.4.12 _level	55
6.8.4.13 _nbrOfPlayers	55
6.8.4.14 _nIDCounter	55
6.8.4.15 _playerConnected	55
6.8.4.16 _port	55
6.8.4.17 _qMessagesIn	56
6.8.4.18 _tempBuffer	56
6.8.4.19 _threadContext	56
6.9 AudioManager Class Reference	56
6.9.1 Detailed Description	57
6.9.2 Member Function Documentation	57
6.9.2.1 getSoundBuffer()	57
6.9.3 Member Data Documentation	57
6.9.3.1 soundBuffers	58
6.10 AudioSystem Class Reference	58
6.10.1 Detailed Description	59
6.10.2 Constructor & Destructor Documentation	59
6.10.2.1 AudioSystem()	59
6.10.3 Member Function Documentation	59

6.10.3.1 playBackgroundMusic()	59
6.10.3.2 playSoundEffect()	61
6.10.3.3 stopBackgroundMusic()	61
6.10.4 Member Data Documentation	61
6.10.4.1 _audioManager	61
6.10.4.2 _backgroundMusic	62
6.10.4.3 _currentMusicFilePath	62
6.10.4.4 _soundEffect	62
6.11 AutoFireSystem Class Reference	62
6.11.1 Detailed Description	63
6.11.2 Constructor & Destructor Documentation	63
6.11.2.1 AutoFireSystem()	63
6.11.3 Member Function Documentation	63
6.11.3.1 handleAutoFire()	63
6.11.4 Member Data Documentation	64
6.11.4.1 _componentManager	64
6.11.4.2 _entityManager	64
6.12 BackgroundComponent Struct Reference	64
6.13 BasicMonsterComponent Struct Reference	65
6.14 BindComponent Struct Reference	65
6.14.1 Detailed Description	65
6.14.2 Constructor & Destructor Documentation	65
6.14.2.1 BindComponent()	65
6.14.3 Member Data Documentation	66
6.14.3.1 bind	66
6.14.3.2 isHovered	66
6.15 BossComponent Struct Reference	66
6.16 r_type::net::Client Class Reference	66
6.16.1 Member Function Documentation	67
6.16.1.1 addEntity()	67
6.16.1.2 animateEntity()	67
6.16.1.3 initInfoBar()	67
6.16.1.4 MessageAll()	68
6.16.1.5 moveEntity()	68
6.16.1.6 PingServer()	68
6.16.1.7 removeEntity()	68
6.16.1.8 updateInfoBar()	68
6.17 CollisionSystem Class Reference	69
6.17.1 Detailed Description	69
6.17.2 Constructor & Destructor Documentation	69
6.17.2.1 CollisionSystem()	70
6.17.3 Member Function Documentation	70

6.17.3.1 checkCollision()	70
6.17.3.2 checkOffScreen()	70
6.17.4 Member Data Documentation	71
6.17.4.1 _componentManager	71
6.17.4.2 _entityManager	71
6.18 ComponentManager Class Reference	71
6.18.1 Detailed Description	72
6.18.2 Member Function Documentation	72
6.18.2.1 addComponent()	72
6.18.2.2 getComponent()	72
6.18.2.3 getComponentMap()	73
6.18.2.4 removeAllComponents()	73
6.18.2.5 removeEntityFromAllComponents()	73
6.18.2.6 removeEntityFromComponent()	74
6.18.3 Member Data Documentation	74
6.18.3.1 components	74
6.19 componentNotFound Class Reference	75
6.19.1 Detailed Description	75
6.19.2 Member Function Documentation	75
6.19.2.1 what()	75
6.20 EnemyComponent Struct Reference	75
6.21 EnemyMissileComponent Struct Reference	76
6.22 Entity Class Reference	76
6.22.1 Detailed Description	76
6.22.2 Constructor & Destructor Documentation	76
6.22.2.1 Entity()	76
6.22.3 Member Function Documentation	77
6.22.3.1 getId()	77
6.22.4 Member Data Documentation	77
6.22.4.1 _id	77
6.23 EntityFactory Class Reference	77
6.23.1 Detailed Description	79
6.23.2 Member Function Documentation	79
6.23.2.1 backgroundFactory()	79
6.23.2.2 createBackgroundLevelOne()	79
6.23.2.3 createBackgroundLevelThree()	79
6.23.2.4 createBackgroundLevelTwo()	81
6.23.2.5 createBackgroundMenu()	81
6.23.2.6 createBasicMonster()	82
6.23.2.7 createButton()	82
6.23.2.8 createEnemyMissile()	83
6.23.2.9 createFilter()	83

6.23.2.10 createForceMissile()	85
6.23.2.11 createForceWeapon()	85
6.23.2.12 createInfoBar()	86
6.23.2.13 createPlayer()	86
6.23.2.14 createPlayerMissile()	87
6.23.2.15 createPowerUpBlueLaserCrystal()	87
6.23.2.16 createShooterEnemy()	88
6.23.2.17 createSmallButton()	88
6.23.2.18 createWall()	90
6.24 EntityInformation Struct Reference	91
6.24.1 Detailed Description	91
6.24.2 Member Data Documentation	91
6.24.2.1 animationComponent	91
6.24.2.2 ratio	91
6.24.2.3 spriteData	91
6.24.2.4 uniqueID	91
6.24.2.5 vPos	92
6.25 EntityManager Class Reference	92
6.25.1 Detailed Description	92
6.25.2 Member Function Documentation	92
6.25.2.1 createEntity()	93
6.25.2.2 getAllEntities()	93
6.25.2.3 getEntity()	93
6.25.2.4 removeAllEntities()	93
6.25.2.5 removeEntity()	94
6.25.3 Member Data Documentation	94
6.25.3.1 entities	94
6.25.3.2 entityNb	94
6.26 entityNotFound Class Reference	95
6.26.1 Detailed Description	95
6.26.2 Member Function Documentation	95
6.26.2.1 what()	95
6.27 failedToCreateFile Class Reference	95
6.27.1 Detailed Description	96
6.27.2 Member Function Documentation	96
6.27.2.1 what()	96
6.28 failedToLoadFont Class Reference	96
6.28.1 Detailed Description	96
6.28.2 Member Function Documentation	97
6.28.2.1 what()	97
6.29 failedToLoadSound Class Reference	97
6.29.1 Detailed Description	97

6.29.2 Member Function Documentation	97
6.29.2.1 what()	97
6.30 failedToLoadTexture Class Reference	98
6.30.1 Detailed Description	98
6.30.2 Member Function Documentation	98
6.30.2.1 what()	98
6.31 failedToOpenFile Class Reference	98
6.31.1 Detailed Description	99
6.31.2 Member Function Documentation	99
6.31.2.1 what()	99
6.32 FontManager Class Reference	99
6.32.1 Detailed Description	100
6.32.2 Member Function Documentation	100
6.32.2.1 getFont()	100
6.32.2.2 releaseFont()	100
6.32.3 Member Data Documentation	101
6.32.3.1 fonts	101
6.33 ForceMissileComponent Struct Reference	101
6.33.1 Detailed Description	101
6.33.2 Member Data Documentation	101
6.33.2.1 forceld	102
6.34 ForceWeaponComponent Struct Reference	102
6.34.1 Detailed Description	102
6.34.2 Constructor & Destructor Documentation	102
6.34.2.1 ForceWeaponComponent()	102
6.34.3 Member Data Documentation	103
6.34.3.1 attached	103
6.34.3.2 level	103
6.34.3.3 playerId	103
6.35 FrontComponent Struct Reference	103
6.35.1 Detailed Description	104
6.35.2 Constructor & Destructor Documentation	104
6.35.2.1 FrontComponent()	104
6.35.3 Member Data Documentation	104
6.35.3.1 targetId	104
6.36 HealthComponent Struct Reference	104
6.36.1 Detailed Description	105
6.36.2 Member Data Documentation	105
6.36.2.1 health	105
6.36.2.2 max_health	105
6.37 HitboxComponent Struct Reference	105
6.37.1 Detailed Description	105

6.37.2 Member Data Documentation	106
6.37.2.1 h	106
6.37.2.2 w	106
6.38 r_type::net::IClient< T > Class Template Reference	106
6.38.1 Constructor & Destructor Documentation	107
6.38.1.1 IClient()	107
6.38.1.2 ~IClient()	107
6.38.2 Member Function Documentation	107
6.38.2.1 Connect()	107
6.38.2.2 Disconnect()	107
6.38.2.3 Incoming()	108
6.38.2.4 IsConnected()	108
6.38.2.5 Send()	108
6.39 IEntityFactory Class Reference	109
6.39.1 Detailed Description	110
6.39.2 Member Enumeration Documentation	110
6.39.2.1 EnemyType	111
6.39.3 Constructor & Destructor Documentation	111
6.39.3.1 ~IEntityFactory()	111
6.39.4 Member Function Documentation	111
6.39.4.1 createBackgroundLevelOne()	111
6.39.4.2 createBackgroundLevelThree()	112
6.39.4.3 createBackgroundLevelTwo()	112
6.39.4.4 createBackgroundMenu()	113
6.39.4.5 createBasicMonster()	113
6.39.4.6 createButton()	113
6.39.4.7 createEnemyMissile()	114
6.39.4.8 createForceMissile()	115
6.39.4.9 createForceWeapon()	115
6.39.4.10 createInfoBar()	116
6.39.4.11 createPlayer()	116
6.39.4.12 createPlayerMissile()	116
6.39.4.13 createPowerUpBlueLaserCrystal()	117
6.39.4.14 createShooterEnemy()	117
6.39.4.15 createSmallButton()	118
6.39.4.16 createWall()	119
6.40 InputComponent Struct Reference	119
6.40.1 Detailed Description	119
6.40.2 Member Data Documentation	119
6.40.2.1 input	120
6.41 IScenes Class Reference	120
6.41.1 Detailed Description	121

6.41.2 Constructor & Destructor Documentation	121
6.41.2.1 ~IScenes()	121
6.41.3 Member Function Documentation	121
6.41.3.1 difficultyChoices()	121
6.41.3.2 gameLoop()	121
6.41.3.3 getRenderWindow()	121
6.41.3.4 inGameMenu()	122
6.41.3.5 mainMenu()	122
6.41.3.6 render()	122
6.41.3.7 settingsMenu()	122
6.41.3.8 shouldQuit()	122
6.42 ISystem Class Reference	123
6.42.1 Detailed Description	123
6.42.2 Constructor & Destructor Documentation	123
6.42.2.1 ISystem()	123
6.42.2.2 ~ISystem()	123
6.43 labelComponent Struct Reference	124
6.43.1 Detailed Description	124
6.43.2 Member Data Documentation	124
6.43.2.1 name	124
6.43.2.2 x	124
6.43.2.3 y	125
6.44 r_type::Level< T > Class Template Reference	125
6.44.1 Detailed Description	127
6.44.2 Constructor & Destructor Documentation	127
6.44.2.1 Level()	127
6.44.2.2 ~Level()	127
6.44.3 Member Function Documentation	127
6.44.3.1 AnimationUpdate()	127
6.44.3.2 collisionAction()	129
6.44.3.3 CollisionUpdate()	129
6.44.3.4 FireUpdate()	130
6.44.3.5 GetEntityBackGround()	130
6.44.3.6 InitiateBackground()	131
6.44.3.7 LevelOne()	131
6.44.3.8 LevelThree()	132
6.44.3.9 LevelTwo()	132
6.44.3.10 MoveUpdate()	133
6.44.3.11 SetGameParameters()	133
6.44.3.12 SetSystem()	133
6.44.3.13 SpawnEntity()	134
6.44.3.14 Update()	134

6.44.4 Member Data Documentation	135
6.44.4.1 _animationSystem	135
6.44.4.2 _autoFireSystem	135
6.44.4.3 _basicMonsterSpawnTime	136
6.44.4.4 _collisionSystem	136
6.44.4.5 _gameParameters	136
6.44.4.6 _moveSystem	136
6.44.4.7 _shooterEnemySpawnTime	137
6.44.4.8 _spawnTimeMonsterThree	137
6.44.4.9 _WallSpawnTime	137
6.45 LinkForceComponent Struct Reference	137
6.45.1 Detailed Description	138
6.45.2 Constructor & Destructor Documentation	138
6.45.2.1 LinkForceComponent()	138
6.45.3 Member Data Documentation	138
6.45.3.1 targetId	138
6.46 MovementComponent Struct Reference	138
6.46.1 Detailed Description	139
6.46.2 Constructor & Destructor Documentation	139
6.46.2.1 MovementComponent() [1/2]	139
6.46.2.2 MovementComponent() [2/2]	139
6.46.3 Member Data Documentation	140
6.46.3.1 index	140
6.46.3.2 move	140
6.46.3.3 movementType	140
6.47 MoveSystem Class Reference	140
6.47.1 Detailed Description	141
6.47.2 Constructor & Destructor Documentation	141
6.47.2.1 MoveSystem()	141
6.47.3 Member Function Documentation	141
6.47.3.1 moveEntities()	142
6.47.3.2 moveEntity()	142
6.47.4 Member Data Documentation	142
6.47.4.1 _componentManager	142
6.47.4.2 _entityManager	143
6.48 OffsetComponent Struct Reference	143
6.48.1 Detailed Description	143
6.48.2 Member Data Documentation	143
6.48.2.1 offset	143
6.49 OnClickComponent Struct Reference	144
6.49.1 Detailed Description	144
6.49.2 Constructor & Destructor Documentation	144

6.49.2.1 OnClickComponent()	144
6.49.3 Member Data Documentation	144
6.49.3.1 isClicked	145
6.49.3.2 onClick	145
6.50 PlayerComponent Struct Reference	145
6.51 playerIdNotFound Class Reference	145
6.51.1 Detailed Description	146
6.51.2 Member Function Documentation	146
6.51.2.1 what()	146
6.52 PlayerMissileComponent Struct Reference	146
6.52.1 Detailed Description	146
6.52.2 Member Data Documentation	146
6.52.2.1 playerId	147
6.53 PositionComponent Struct Reference	147
6.53.1 Detailed Description	147
6.53.2 Constructor & Destructor Documentation	147
6.53.2.1 PositionComponent()	147
6.53.3 Member Data Documentation	147
6.53.3.1 x	148
6.53.3.2 y	148
6.54 PowerUpComponent Struct Reference	148
6.55 RectangleShapeComponent Struct Reference	148
6.55.1 Detailed Description	148
6.55.2 Constructor & Destructor Documentation	148
6.55.2.1 RectangleShapeComponent()	148
6.55.3 Member Data Documentation	149
6.55.3.1 rectangleShape	149
6.56 RenderSystem Class Reference	149
6.56.1 Detailed Description	149
6.56.2 Constructor & Destructor Documentation	150
6.56.2.1 RenderSystem()	150
6.56.3 Member Function Documentation	150
6.56.3.1 render()	150
6.56.4 Member Data Documentation	150
6.56.4.1 _componentManager	150
6.56.4.2 _font	151
6.56.4.3 _window	151
6.57 Scenes Class Reference	151
6.57.1 Detailed Description	152
6.57.2 Constructor & Destructor Documentation	152
6.57.2.1 Scenes()	152
6.57.2.2 ~Scenes()	153

6.57.3 Member Function Documentation	153
6.57.3.1 difficultyChoices()	153
6.57.3.2 gameLoop()	153
6.57.3.3 getRenderWindow()	153
6.57.3.4 HandleMessage()	154
6.57.3.5 inGameMenu()	154
6.57.3.6 mainMenu()	155
6.57.3.7 render()	155
6.57.3.8 run()	155
6.57.3.9 settingsMenu()	156
6.57.3.10 shouldQuit()	156
6.57.3.11 StopGameLoop()	156
6.57.4 Member Data Documentation	156
6.57.4.1 _networkClient	156
6.57.4.2 _window	157
6.58 ScoreComponent Struct Reference	157
6.58.1 Detailed Description	157
6.58.2 Member Data Documentation	157
6.58.2.1 score	157
6.59 r_type::net::Server Class Reference	158
6.59.1 Detailed Description	158
6.59.2 Constructor & Destructor Documentation	159
6.59.2.1 Server()	159
6.59.2.2 ~Server()	159
6.59.3 Member Function Documentation	159
6.59.3.1 OnClientConnect()	159
6.59.3.2 OnClientDisconnect()	160
6.59.3.3 OnMessage()	160
6.60 ShaderComponent Struct Reference	161
6.60.1 Detailed Description	161
6.60.2 Constructor & Destructor Documentation	162
6.60.2.1 ShaderComponent()	162
6.60.3 Member Data Documentation	162
6.60.3.1 shader	162
6.61 ShootComponent Struct Reference	162
6.61.1 Detailed Description	163
6.61.2 Constructor & Destructor Documentation	163
6.61.2.1 ShootComponent()	163
6.61.3 Member Data Documentation	163
6.61.3.1 canShoot	163
6.61.3.2 cooldownTime	164
6.61.3.3 nextShootTime	164

6.62 SpriteComponent Struct Reference	164
6.62.1 Detailed Description	164
6.62.2 Constructor & Destructor Documentation	165
6.62.2.1 SpriteComponent()	165
6.62.3 Member Data Documentation	165
6.62.3.1 hitboxX	165
6.62.3.2 hitboxY	165
6.62.3.3 sprite	166
6.62.3.4 type	166
6.63 SpriteDataComponent Struct Reference	166
6.63.1 Detailed Description	166
6.63.2 Member Data Documentation	166
6.63.2.1 scale	167
6.63.2.2 spritePath	167
6.63.2.3 type	167
6.64 TextComponent Struct Reference	167
6.64.1 Detailed Description	167
6.64.2 Constructor & Destructor Documentation	168
6.64.2.1 TextComponent()	168
6.64.3 Member Data Documentation	168
6.64.3.1 text	168
6.65 TextDataComponent Struct Reference	168
6.65.1 Detailed Description	169
6.65.2 Member Data Documentation	169
6.65.2.1 categoryIds	169
6.65.2.2 categorySize	169
6.65.2.3 categoryTexts	169
6.65.2.4 charSize	170
6.65.2.5 fontPath	170
6.66 TextureManager Class Reference	170
6.66.1 Member Function Documentation	170
6.66.1.1 getTexture()	170
6.66.1.2 releaseTexture()	171
6.66.2 Member Data Documentation	171
6.66.2.1 textures	171
6.67 UIEntityInformation Struct Reference	172
6.67.1 Detailed Description	172
6.67.2 Member Data Documentation	172
6.67.2.1 lives	172
6.67.2.2 score	172
6.67.2.3 spriteData	173
6.67.2.4 textData	173

6.67.2.5 uniqueID	173
6.68 UpdateSystem Class Reference	173
6.68.1 Detailed Description	174
6.68.2 Constructor & Destructor Documentation	174
6.68.2.1 UpdateSystem()	174
6.68.3 Member Function Documentation	174
6.68.3.1 updateSpritePositions()	175
6.68.4 Member Data Documentation	175
6.68.4.1 _componentManager	175
6.68.4.2 _entityManager	175
6.68.4.3 _window	175
6.69 VelocityComponent Struct Reference	176
6.69.1 Detailed Description	176
6.69.2 Member Data Documentation	176
6.69.2.1 x	176
6.69.2.2 y	176
6.70 vf2d Struct Reference	176
6.70.1 Detailed Description	177
6.70.2 Member Data Documentation	177
6.70.2.1 x	177
6.70.2.2 y	177
6.71 WallComponent Struct Reference	177
7 File Documentation	179
7.1 /home/runner/work/R-Type/R-Type/Client/Interface/Include/mainmenu.hpp File Reference	179
7.1.1 Function Documentation	179
7.1.1.1 MainMenu()	179
7.2 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/a_client.hpp File Reference	179
7.3 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/client.hpp File Reference	180
7.4 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp File Reference	180
7.5 /home/runner/work/R-Type/R-Type/Client/Interface/Include/scenes.hpp File Reference	181
7.5.1 Function Documentation	181
7.5.1.1 keyToString()	181
7.6 /home/runner/work/R-Type/R-Type/Client/Src/keyToString.cpp File Reference	181
7.6.1 Function Documentation	181
7.6.1.1 keyToString()	182
7.7 /home/runner/work/R-Type/R-Type/Client/Src/main.cpp File Reference	182
7.7.1 Function Documentation	182
7.7.1.1 isValidIPv4()	182
7.7.1.2 isValidPort()	182
7.7.1.3 main()	182
7.8 /home/runner/work/R-Type/R-Type/Server/Src/main.cpp File Reference	183

7.8.1 Function Documentation	183
7.8.1.1 isValidPort()	183
7.8.1.2 main()	184
7.8.1.3 signal_handler()	184
7.8.2 Variable Documentation	184
7.8.2.1 loopRunning	185
7.9 /home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp File Reference	185
7.9.1 Function Documentation	185
7.9.1.1 createDaltonismChoiceButtons()	186
7.9.1.2 createKeyBindingButtons()	186
7.9.1.3 handleEvents()	186
7.9.1.4 reloadFilter()	187
7.9.1.5 waitForKey()	187
7.10 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp File Reference	187
7.11 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/audio_manager.hpp File Reference	187
7.12 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_component.hpp File Reference	187
7.12.1 Detailed Description	188
7.13 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_missile_component.hpp File Reference	188
7.13.1 Detailed Description	188
7.14 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/animation_component.hpp File Reference	188
7.14.1 Function Documentation	189
7.14.1.1 operator!=(=)	189
7.15 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/background_component.hpp File Reference	190
7.15.1 Detailed Description	190
7.16 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/basic_monster_component.hpp File Reference	190
7.16.1 Detailed Description	191
7.17 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/bind_component.hpp File Reference	191
7.18 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/boss_component.hpp File Reference	191
7.18.1 Detailed Description	191
7.19 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/component_manager.hpp File Reference	192
7.20 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/components.hpp File Reference	192
7.21 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_component.hpp File Reference	193
7.21.1 Detailed Description	193
7.22 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_missile_component.hpp File Reference	193

7.22.1 Detailed Description	193
7.23 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_missile_component.hpp File Reference	193
7.24 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_weapon_component.hpp File Reference	194
7.25 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/front_component.hpp File Reference	194
7.26 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/health_component.hpp File Reference	194
7.27 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/hitbox_component.hpp File Reference	194
7.28 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/input_component.hpp File Reference	195
7.28.1 Enumeration Type Documentation	195
7.28.1.1 InputType	195
7.29 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp File Reference	195
7.30 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link_force_component.hpp File Reference	196
7.31 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp File Reference	196
7.31.1 Enumeration Type Documentation	196
7.31.1.1 MovementType	196
7.32 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/offset_component.hpp File Reference	197
7.33 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/on_click_component.hpp File Reference	197
7.33.1 Detailed Description	197
7.34 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_component.hpp File Reference	197
7.34.1 Detailed Description	197
7.35 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_missile_component.hpp File Reference	198
7.36 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/position_component.hpp File Reference	198
7.37 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/power_up_component.hpp File Reference	198
7.37.1 Detailed Description	198
7.38 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/rectangleShapeComponent.hpp File Reference	198
7.39 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/score_component.hpp File Reference	199
7.39.1 Detailed Description	199
7.40 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shader_component.hpp File Reference	199
7.41 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_component.hpp File Reference	199

7.42	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_component.hpp File Reference	200
7.43	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_data_component.hpp File Reference	200
7.43.1	Function Documentation	200
7.43.1.1	operator<<()	200
7.44	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp File Reference	201
7.45	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_data_component.hpp File Reference	201
7.46	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/velocity_component.hpp File Reference	201
7.47	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wall_component.hpp File Reference	202
7.47.1	Detailed Description	202
7.48	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/creatable_client_object.hpp File Reference	202
7.48.1	Enumeration Type Documentation	202
7.48.1.1	CreatableClientObject	202
7.49	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hpp File Reference	203
7.50	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_factory.hpp File Reference	203
7.51	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp File Reference	203
7.52	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/i_entity_factory.hpp File Reference	203
7.53	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/entity_struct.hpp File Reference	204
7.54	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp File Reference	204
7.55	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_manager.hpp File Reference	205
7.56	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_path.hpp File Reference	205
7.56.1	Enumeration Type Documentation	205
7.56.1.1	FontPath	206
7.56.2	Function Documentation	206
7.56.2.1	FontFactory()	206
7.56.2.2	operator<<()	207
7.57	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/game_text.hpp File Reference	207
7.57.1	Enumeration Type Documentation	207
7.57.1.1	GameText	208
7.57.2	Function Documentation	208
7.57.2.1	GameTextFactory()	208
7.57.2.2	operator<<()	208
7.58	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/hitbox_tmp.hpp File Reference	209
7.58.1	Function Documentation	209
7.58.1.1	CheckEntityMovement()	209
7.58.1.2	CheckEntityPosition()	210
7.59	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/i_scenes.hpp File Reference	210
7.60	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/macros.hpp File Reference	210
7.60.1	Macro Definition Documentation	211

7.60.1.1	SCREEN_HEIGHT	211
7.60.1.2	SCREEN_WIDTH	211
7.61	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/sound_path.hpp File Reference	211
7.61.1	Enumeration Type Documentation	211
7.61.1.1	ActionType	212
7.61.2	Function Documentation	212
7.61.2.1	SoundFactory()	213
7.62	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/sprite_path.hpp File Reference	213
7.62.1	Enumeration Type Documentation	213
7.62.1.1	SpritePath	213
7.62.2	Function Documentation	214
7.62.2.1	operator<<()	214
7.62.2.2	SpriteFactory()	215
7.63	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/audio_system.hpp File Reference	215
7.64	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/auto_fire_system.hpp File Reference	215
7.65	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/collision_system.hpp File Reference	216
7.66	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/i_system.hpp File Reference	216
7.67	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/move_system.hpp File Reference	216
7.68	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/render_system.hpp File Reference	216
7.69	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/systems.hpp File Reference	217
7.70	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/update_system.hpp File Reference	217
7.71	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/texture_manager.hpp File Reference	217
7.72	/home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp File Reference	217
7.73	/home/runner/work/R-Type/R-Type/ECS/Src/Entities/entity_factory.cpp File Reference	218
7.73.1	Function Documentation	218
7.73.1.1	operator<<() [1/4]	218
7.73.1.2	operator<<() [2/4]	218
7.73.1.3	operator<<() [3/4]	218
7.73.1.4	operator<<() [4/4]	219
7.74	/home/runner/work/R-Type/R-Type/ECS/Src/font_path.cpp File Reference	219
7.74.1	Function Documentation	219
7.74.1.1	FontFactory()	220
7.75	/home/runner/work/R-Type/R-Type/ECS/Src/game_text.cpp File Reference	220
7.75.1	Function Documentation	220
7.75.1.1	GameTextFactory()	220
7.76	/home/runner/work/R-Type/R-Type/ECS/Src/hitbox_tmp.cpp File Reference	221
7.76.1	Function Documentation	221
7.76.1.1	CheckCollisionLogic()	221
7.76.1.2	CheckEntityMovement()	221
7.76.1.3	CheckEntityPosition()	222
7.77	/home/runner/work/R-Type/R-Type/ECS/Src/sound_path.cpp File Reference	222

7.77.1 Function Documentation	223
7.77.1.1 SoundFactory()	223
7.78 /home/runner/work/R-Type/R-Type/ECS/Src/sprite_path.cpp File Reference	223
7.78.1 Function Documentation	223
7.78.1.1 SpriteFactory()	223
7.79 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_system.cpp File Reference	224
7.80 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/auto_fire_system.cpp File Reference	224
7.81 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/collision_system.cpp File Reference	224
7.82 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/move_system.cpp File Reference	224
7.83 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/render_system.cpp File Reference	224
7.84 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/update_system.cpp File Reference	225
7.85 /home/runner/work/R-Type/R-Type/Server/Interface/Include/animation_system.hpp File Reference	225
7.85.1 Enumeration Type Documentation	226
7.85.1.1 AnimationBasicMonster	226
7.85.1.2 AnimationForceMissile1	227
7.85.1.3 AnimationForceMissile2	227
7.85.1.4 AnimationForceMissile3	227
7.85.1.5 AnimationForceWeapon1	228
7.85.1.6 AnimationForceWeapon2	228
7.85.1.7 AnimationForceWeapon3	228
7.85.1.8 AnimationShip	229
7.85.2 Function Documentation	229
7.85.2.1 animationShipFactory()	229
7.85.2.2 operator"!=()"	230
7.86 /home/runner/work/R-Type/R-Type/Server/Interface/Include/level.hpp File Reference	230
7.87 /home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/a_server.hpp File Reference	231
7.88 /home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/server.hpp File Reference	231
7.89 /home/runner/work/R-Type/R-Type/Server/Src/animation_system.cpp File Reference	232
7.89.1 Function Documentation	232
7.89.1.1 animateForceMissileLevel1()	232
7.89.1.2 animateForceMissileLevel2()	233
7.89.1.3 animateForceMissileLevel3()	233
7.89.1.4 animateForceWeaponLevel1()	233
7.89.1.5 animateForceWeaponLevel2()	233
7.89.1.6 animateForceWeaponLevel3()	233
7.89.1.7 animationBasicMonsterFactory()	233
7.89.1.8 animationForceMissile1Factory()	233
7.89.1.9 animationForceMissile2Factory()	234
7.89.1.10 animationForceMissile3Factory()	234
7.89.1.11 animationForceWeapon1Factory()	234
7.89.1.12 animationForceWeapon2Factory()	234
7.89.1.13 animationForceWeapon3Factory()	234

7.89.1.14 animationShipFactory()	234
7.89.1.15 operator"!=()	235
7.89.1.16 operator==(())	235
7.90 /home/runner/work/R-Type/R-Type/Server/Src/server.cpp File Reference	235
Index	237

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

r_type	13
r_type::net	13

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

AbstractScenes	15
AllyComponent	20
AllyMissileComponent	20
AnimationComponent	20
AudioManager	56
BackgroundComponent	64
BasicMonsterComponent	65
BindComponent	65
BossComponent	66
ComponentManager	71
EnemyComponent	75
EnemyMissileComponent	76
Entity	76
EntityInformation	91
EntityManager	92
std::exception	
componentNotFound	75
entityNotFound	95
failedToCreateFile	95
failedToLoadFont	96
failedToLoadSound	97
failedToLoadTexture	98
failedToOpenFile	98
playerIdNotFound	145
FontManager	99
ForceMissileComponent	101
ForceWeaponComponent	102
FrontComponent	103
HealthComponent	104
HitboxComponent	105
r_type::net::IClient< T >	106
r_type::net::AClient< TypeMessage >	15
r_type::net::Client	66
r_type::net::AClient< T >	15
IEntityFactory	109

EntityFactory	77
ILevel	
r_type::Level< T >	125
InputComponent	119
IScenes	120
AScenes	26
Scenes	151
r_type::net::IServer	
r_type::net::AServer< TypeMessage >	38
r_type::net::Server	158
r_type::net::AServer< T >	38
ISystem	123
AnimationSystem	22
AudioSystem	58
AutoFireSystem	62
CollisionSystem	69
MoveSystem	140
RenderSystem	149
UpdateSystem	173
labelComponent	124
LinkForceComponent	137
MovementComponent	138
OffsetComponent	143
OnClickComponent	144
PlayerComponent	145
PlayerMissileComponent	146
PositionComponent	147
PowerUpComponent	148
RectangleShapeComponent	148
ScoreComponent	157
ShaderComponent	161
ShootComponent	162
SpriteComponent	164
SpriteDataComponent	166
TextComponent	167
TextDataComponent	168
TextureManager	170
UIEntityInformation	172
VelocityComponent	176
vf2d	176
WallComponent	177

Chapter 3

Class Index

3.1 Class List

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

AbstractScenes	
An abstract class that provides a base for managing different scenes in a game	15
r_type::net::AClient< T >	15
AllyComponent	20
AllyMissileComponent	20
AnimationComponent	
A component that holds animation properties such as offset and dimension	20
AnimationSystem	
A system responsible for animating entities within the ECS framework	22
AScenes	26
r_type::net::AServer< T >	
AServer class template for managing server operations	38
AudioManager	
Manages and caches sound buffers for efficient audio playback	56
AudioSystem	
Manages audio playback within the application	58
AutoFireSystem	
A system that handles automatic firing mechanisms for entities	62
BackgroundComponent	64
BasicMonsterComponent	65
BindComponent	
A component that binds a function to handle scene transitions	65
BossComponent	66
r_type::net::Client	66
CollisionSystem	
Manages collision detection and response within the ECS framework	69
ComponentManager	
Manages the components of entities in an ECS system	71
componentNotFound	
Exception class for when a component is not found	75
EnemyComponent	75
EnemyMissileComponent	76
Entity	
Represents an entity in the ECS system	76
EntityFactory	
A factory class for creating various types of entities	77

EntityInformation	Represents information about an entity	91
EntityManager	Manages the creation, removal, and retrieval of entities	92
entityNotFound	Exception class for entity not found error	95
failedToCreateFile	Exception class for handling file creation failures	95
failedToLoadFont	Exception class for handling font loading failures	96
failedToLoadSound	Exception class for handling sound loading failures	97
failedToLoadTexture	Exception class for failed texture loading	98
failedToOpenFile	Exception class for handling file opening failures	98
FontManager	Manages the loading and retrieval of font resources	99
ForceMissileComponent	Component representing a force missile in the ECS system	101
ForceWeaponComponent	Represents a component for a force weapon in the game	102
FrontComponent	A component that represents the front of an entity	103
HealthComponent	Represents the health attributes of an entity	104
HitboxComponent	Represents the hitbox dimensions of an entity	105
r_type::net::IClient< T >		106
IEntityFactory	The interface for an entity factory	109
InputComponent	Component for handling input actions	119
IScenes	Interface for managing different scenes in a game	120
ISystem	Interface for all systems in the ECS (Entity Component System) architecture	123
labelComponent	Represents a label component with a name and position coordinates	124
r_type::Level< T >	The Level class template manages the game level, including updating game state, handling collisions, and managing entities	125
LinkForceComponent	Component that links an entity to a target entity by ID	137
MovementComponent	Represents a component that handles movement in the ECS system	138
MoveSystem	System responsible for moving entities within the ECS framework	140
OffsetComponent	Component that represents an offset value	143
OnClickComponent	Component that handles click events	144
PlayerComponent		145
playerIdNotFound	Exception class for handling cases where a player ID is not found	145
PlayerMissileComponent	Component that represents a missile belonging to a player	146

PositionComponent	
A component that represents the position of an entity in 2D space	147
PowerUpComponent	148
RectangleShapeComponent	
A component that holds an sf::RectangleShape	148
RenderSystem	
A system responsible for rendering entities in the ECS framework	149
Scenes	
Represents a class that manages different scenes in a game	151
ScoreComponent	
Component that holds the score of an entity	157
r_type::net::Server	
A server class that handles client connections and messaging	158
ShaderComponent	
A component that holds a shader	161
ShootComponent	
Component that handles shooting mechanics for an entity	162
SpriteComponent	
A component that represents a sprite in the ECS (Entity Component System)	164
SpriteDataComponent	
Component that holds data related to a sprite	166
TextComponent	
A component that encapsulates an SFML text object	167
TextDataComponent	
Component that holds text-related data for an entity	168
TextureManager	170
UIEntityInformation	
Represents the information of a UI entity in the game	172
UpdateSystem	
A system responsible for updating sprite positions in the game	173
VelocityComponent	
Represents the velocity of an entity in 2D space	176
vf2d	
Represents a 2D vector with x and y coordinates	176
WallComponent	177

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

/home/runner/work/R-Type/R-Type/Client/Interface/Include/mainmenu.hpp	179
/home/runner/work/R-Type/R-Type/Client/Interface/Include/scenes.hpp	181
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/a_client.hpp	179
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/client.hpp	180
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp	180
/home/runner/work/R-Type/R-Type/Client/Src/keyToString.cpp	181
/home/runner/work/R-Type/R-Type/Client/Src/main.cpp	182
/home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp	185
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp	187
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/audio_manager.hpp	187
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/creatable_client_object.hpp	202
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/entity_struct.hpp	204
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp	204
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_manager.hpp	205
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_path.hpp	205
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/game_text.hpp	207
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/hitbox_tmp.hpp	209
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/i_scenes.hpp	210
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/macros.hpp	210
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/sound_path.hpp	211
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/sprite_path.hpp	213
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/texture_manager.hpp	217
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_component.hpp	
Defines the AllyComponent structure	187
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_missile_component.hpp	
Defines the AllyMissileComponent structure	188
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/animation_component.hpp	188
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/background_component.hpp	
Defines the BackgroundComponent structure	190
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/basic_monster_component.hpp	
Defines the BasicMonsterComponent structure	190
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/bind_component.hpp	191
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/boss_component.hpp	
Defines the BossComponent structure	191
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/component_manager.hpp	192

/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/components.hpp	192
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_component.hpp	
Defines the EnemyComponent structure	193
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_missile_component.hpp	
Defines the EnemyMissileComponent structure	193
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_missile_component.hpp . .	193
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_weapon_component.hpp .	194
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/front_component.hpp	194
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/health_component.hpp	194
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/hitbox_component.hpp	194
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/input_component.hpp	195
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp	195
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link_force_component.hpp . . .	196
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp . . .	196
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/offset_component.hpp	197
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/on_click_component.hpp	
Defines the OnClickComponent structure used for handling click events in the ECS system . .	197
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_component.hpp	
Defines the PlayerComponent structure	197
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_missile_component.hpp .	198
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/position_component.hpp	198
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/power_up_component.hpp	
Defines the PowerUpComponent structure	198
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/rectangleShapeComponent.hpp .	198
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/score_component.hpp	
Defines the ScoreComponent struct used to store the score of an entity	199
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shader_component.hpp	199
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_component.hpp	199
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_component.hpp	200
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_data_component.hpp . . .	200
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp	201
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_data_component.hpp	201
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/velocity_component.hpp	201
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wall_component.hpp	
Defines the WallComponent structure	202
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hpp	203
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_factory.hpp	203
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp	203
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/i_entity_factory.hpp	203
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/audio_system.hpp	215
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/auto_fire_system.hpp	215
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/collision_system.hpp	216
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/i_system.hpp	216
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/move_system.hpp	216
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/render_system.hpp	216
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/systems.hpp	217
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/update_system.hpp	217
/home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp	217
/home/runner/work/R-Type/R-Type/ECS/Src/font_path.cpp	219
/home/runner/work/R-Type/R-Type/ECS/Src/game_text.cpp	220
/home/runner/work/R-Type/R-Type/ECS/Src/hitbox_tmp.cpp	221
/home/runner/work/R-Type/R-Type/ECS/Src/sound_path.cpp	222
/home/runner/work/R-Type/R-Type/ECS/Src/sprite_path.cpp	223
/home/runner/work/R-Type/R-Type/ECS/Src/Entities/entity_factory.cpp	218
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_system.cpp	224
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/auto_fire_system.cpp	224
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/collision_system.cpp	224
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/move_system.cpp	224

/home/runner/work/R-Type/R-Type/ECS/Src/Systems/ render_system.cpp	224
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/ update_system.cpp	225
/home/runner/work/R-Type/R-Type/Server/Interface/Include/ animation_system.hpp	225
/home/runner/work/R-Type/R-Type/Server/Interface/Include/ level.hpp	230
/home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/ a_server.hpp	231
/home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/ server.hpp	231
/home/runner/work/R-Type/R-Type/Server/Src/ animation_system.cpp	232
/home/runner/work/R-Type/R-Type/Server/Src/ main.cpp	183
/home/runner/work/R-Type/R-Type/Server/Src/ server.cpp	235

Chapter 5

Namespace Documentation

5.1 `r_type` Namespace Reference

Namespaces

- [net](#)

Classes

- class [Level](#)

The [Level](#) class template manages the game level, including updating game state, handling collisions, and managing entities.

5.2 `r_type::net` Namespace Reference

Classes

- class [AClient](#)
- class [Client](#)
- class [IClient](#)
- class [AServer](#)

[AServer](#) class template for managing server operations.

- class [Server](#)

A server class that handles client connections and messaging.

Chapter 6

Class Documentation

6.1 AbstractScenes Class Reference

An abstract class that provides a base for managing different scenes in a game.

```
#include <a_scenes.hpp>
```

6.1.1 Detailed Description

An abstract class that provides a base for managing different scenes in a game.

This abstract class implements the ScenesInterface and provides some common functionality.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp](#)

6.2 r_type::net::AClient< T > Class Template Reference

```
#include <a_client.hpp>
```

Inheritance diagram for r_type::net::AClient< T >:



Public Member Functions

- [AClient](#) ()
- virtual [~AClient](#) ()
- bool [Connect](#) (const std::string &host, const uint16_t port)
Connects to a remote host using UDP protocol.
- void [Disconnect](#) ()
Disconnects the client from the server.
- bool [IsConnected](#) ()
Checks if the client is connected to the server.
- void [Send](#) (const Message< T > &msg)
Send message to server.
- ThreadSafeQueue< OwnedMessage< T > > & [Incoming](#) ()
get incoming messages
- const std::unique_ptr< Connection< T > > & [getConnection](#) ()
- void [setPlayerId](#) (uint32_t id)
- uint32_t [getPlayerId](#) ()
- void [setWindowSize](#) (sf::Vector2u size)
- sf::Vector2u [getWindowSize](#) ()

Protected Attributes

- asio::io_context [m_context](#)
- std::thread [thrContext](#)
- std::unique_ptr< Connection< T > > [m_connection](#)

Private Attributes

- ThreadSafeQueue< OwnedMessage< T > > [m_qMessagesIn](#)
- uint32_t [playerId](#) = 0
- sf::Vector2u [windowSize](#)

6.2.1 Constructor & Destructor Documentation

6.2.1.1 AClient()

```
template<typename T >
r_type::net::AClient< T >::AClient ( ) [inline]
```

6.2.1.2 ~AClient()

```
template<typename T >
virtual r_type::net::AClient< T >::~~AClient ( ) [inline], [virtual]
```


6.2.2 Member Function Documentation

6.2.2.1 `Connect()`

```
template<typename T >
bool r_type::net::AClient< T >::Connect (
    const std::string & host,
    const uint16_t port ) [inline], [virtual]
```

Connects to a remote host using UDP protocol.

Parameters

<i>host</i>	The IP address or hostname of the remote host.
<i>port</i>	The port number of the remote host.

Returns

true if the connection is successful, false otherwise.

Implements `r_type::net::IClient< T >`.

6.2.2.2 `Disconnect()`

```
template<typename T >
void r_type::net::AClient< T >::Disconnect ( ) [inline], [virtual]
```

Disconnects the client from the server.

This function disconnects the client from the server if it is currently connected. It stops the context and joins the context thread. It also releases the connection resource.

Implements `r_type::net::IClient< T >`.

6.2.2.3 `getConnection()`

```
template<typename T >
const std::unique_ptr<Connection<T> >& r_type::net::AClient< T >::getConnection ( ) [inline]
```

6.2.2.4 getPlayerId()

```
template<typename T >
uint32_t r_type::net::AClient< T >::getPlayerId ( ) [inline]
```

6.2.2.5 getWindowSize()

```
template<typename T >
sf::Vector2u r_type::net::AClient< T >::getWindowSize ( ) [inline]
```

6.2.2.6 Incoming()

```
template<typename T >
ThreadSafeQueue<OwnedMessage<T> >& r_type::net::AClient< T >::Incoming ( ) [inline], [virtual]
```

get incoming messages

Returns

ThreadSafeQueue<OwnedMessage<T>>&

Implements [r_type::net::IClient< T >](#).

6.2.2.7 isConnected()

```
template<typename T >
bool r_type::net::AClient< T >::IsConnected ( ) [inline], [virtual]
```

Checks if the client is connected to the server.

Returns

true

false

Implements [r_type::net::IClient< T >](#).

6.2.2.8 Send()

```
template<typename T >
void r_type::net::AClient< T >::Send (
    const Message< T > & msg ) [inline], [virtual]
```

Send message to server.

Parameters

<code>msg</code>	
------------------	--

Implements `r_type::net::IClient< T >`.

6.2.2.9 `setPlayerId()`

```
template<typename T >
void r_type::net::AClient< T >::setPlayerId (
    uint32_t id ) [inline]
```

6.2.2.10 `setWindowSize()`

```
template<typename T >
void r_type::net::AClient< T >::setWindowSize (
    sf::Vector2u size ) [inline]
```

6.2.3 Member Data Documentation

6.2.3.1 `m_connection`

```
template<typename T >
std::unique_ptr<Connection<T> > r_type::net::AClient< T >::m_connection [protected]
```

6.2.3.2 `m_context`

```
template<typename T >
asio::io_context r_type::net::AClient< T >::m_context [protected]
```

6.2.3.3 `m_qMessagesIn`

```
template<typename T >
ThreadSafeQueue<OwnedMessage<T> > r_type::net::AClient< T >::m_qMessagesIn [private]
```

6.2.3.4 playerId

```
template<typename T >
uint32_t r_type::net::AClient< T >::playerId = 0 [private]
```

6.2.3.5 thrContext

```
template<typename T >
std::thread r_type::net::AClient< T >::thrContext [protected]
```

6.2.3.6 windowSize

```
template<typename T >
sf::Vector2u r_type::net::AClient< T >::windowSize [private]
```

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

- [/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/a_client.hpp](#)

6.3 AllyComponent Struct Reference

```
#include <ally_component.hpp>
```

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_component.hpp](#)

6.4 AllyMissileComponent Struct Reference

```
#include <ally_missile_component.hpp>
```

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_missile_component.hpp](#)

6.5 AnimationComponent Struct Reference

A component that holds animation properties such as offset and dimension.

```
#include <animation_component.hpp>
```

Public Member Functions

- [AnimationComponent](#) ([vf2d _offset](#), [vf2d _dimension](#))

Constructs an [AnimationComponent](#) with the given offset and dimension.

Public Attributes

- [vf2d offset](#)

The offset of the animation.

- [vf2d dimension](#)

The dimension of the animation.

6.5.1 Detailed Description

A component that holds animation properties such as offset and dimension.

This component is used to define the properties of an animation, including its offset and dimension.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 AnimationComponent()

```
AnimationComponent::AnimationComponent (
    vf2d \_offset,
    vf2d \_dimension ) [inline]
```

Constructs an [AnimationComponent](#) with the given offset and dimension.

Parameters

_offset	The offset of the animation.
_dimension	The dimension of the animation.

6.5.3 Member Data Documentation

6.5.3.1 dimension

```
AnimationComponent::dimension
```

The dimension of the animation.

6.5.3.2 offset

`AnimationComponent::offset`

The offset of the animation.

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

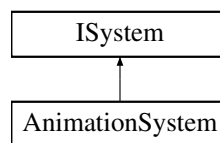
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/animation_component.hpp](#)

6.6 AnimationSystem Class Reference

A system responsible for animating entities within the ECS framework.

```
#include <animation_system.hpp>
```

Inheritance diagram for AnimationSystem:



Public Member Functions

- [AnimationSystem](#) ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager)
- void [AnimationEntities](#) ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, float deltaTime)
Animates entities.
- void [animatePlayer](#) (std::optional< [VelocityComponent](#) * > &velocity, std::optional< [AnimationComponent](#) * > &animation)
Animates the player based on their velocity.
- void [animateBasicMonster](#) (std::optional< [AnimationComponent](#) * > &animation)
Animates a basic monster entity.
- void [animateForceWeapon](#) (std::optional< [ForceWeaponComponent](#) * > &forceWeapon, std::optional< [AnimationComponent](#) * > &animation)
Animates the force weapon based on its current state.
- void [animateForceMissile](#) (std::optional< [ForceWeaponComponent](#) * > &forceWeapon, std::optional< [AnimationComponent](#) * > &animation)
Animates the force missile based on the provided components.

Private Attributes

- [ComponentManager](#) & [_componentManager](#)
Reference to the [ComponentManager](#) instance.
- [EntityManager](#) & [_entityManager](#)
Reference to the [EntityManager](#) instance.

6.6.1 Detailed Description

A system responsible for animating entities within the ECS framework.

The [AnimationSystem](#) class provides functionality to animate various entities such as players, basic monsters, force weapons, and force missiles. It interacts with the [ComponentManager](#) and [EntityManager](#) to access and update the relevant components required for animation.

The [AnimationSystem](#) class inherits from the [ISystem](#) interface and implements several methods to handle the animation of different types of entities. It processes entities based on their animation components and updates their animation states according to the provided delta time or specific component states.

Note

This class assumes the presence of specific components such as [VelocityComponent](#), [AnimationComponent](#), and [ForceWeaponComponent](#) to perform the animations. The methods use optional pointers to these components to ensure that animations are only performed when the components are available.

See also

[ISystem](#)
[ComponentManager](#)
[EntityManager](#)

6.6.2 Constructor & Destructor Documentation

6.6.2.1 AnimationSystem()

```
AnimationSystem::AnimationSystem (
    ComponentManager & componentManager,
    EntityManager & entityManager ) [inline]
```

6.6.3 Member Function Documentation

6.6.3.1 animateBasicMonster()

```
void AnimationSystem::animateBasicMonster (
    std::optional< AnimationComponent * > & animation )
```

Animates a basic monster entity.

This function updates the animation state of a basic monster entity based on the provided [AnimationComponent](#). The animation state is modified to reflect the current frame or sequence in the animation.

Parameters

<i>animation</i>	An optional pointer to the AnimationComponent associated with the basic monster entity. If the optional is empty, no animation will be performed.
------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.6.3.2 animateForceMissile()

```
void AnimationSystem::animateForceMissile (
    std::optional< ForceWeaponComponent * > & forceWeapon,
    std::optional< AnimationComponent * > & animation )
```

Animates the force missile based on the provided components.

This function updates the animation state of a force missile using the provided [ForceWeaponComponent](#) and [AnimationComponent](#). The function ensures that the animation reflects the current state of the force missile.

Parameters

<i>forceWeapon</i>	An optional reference to a ForceWeaponComponent pointer. This component contains the state and properties of the force missile weapon.
<i>animation</i>	An optional reference to an AnimationComponent pointer. This component handles the animation state and frames for the force missile.

6.6.3.3 animateForceWeapon()

```
void AnimationSystem::animateForceWeapon (
    std::optional< ForceWeaponComponent * > & forceWeapon,
    std::optional< AnimationComponent * > & animation )
```

Animates the force weapon based on its current state.

This function updates the animation of the force weapon component by modifying the associated animation component.

Parameters

<i>forceWeapon</i>	An optional reference to the ForceWeaponComponent .
<i>animation</i>	An optional reference to the AnimationComponent .

6.6.3.4 animatePlayer()

```
void AnimationSystem::animatePlayer (
    std::optional< VelocityComponent * > & velocity,
    std::optional< AnimationComponent * > & animation )
```


Animates the player based on their velocity.

This function updates the player's animation state according to the provided velocity component. If the velocity component indicates movement, the animation component will be updated to reflect the corresponding animation state.

Parameters

<i>velocity</i>	A reference to an optional VelocityComponent pointer. If the pointer is present, it contains the player's current velocity.
<i>animation</i>	A reference to an optional AnimationComponent pointer. If the pointer is present, it contains the player's current animation state.

6.6.3.5 AnimationEntities()

```
void AnimationSystem::AnimationEntities (
    ComponentManager & componentManager,
    EntityManager & entityManager,
    float deltaTime )
```

Animates entities.

Updates the animation states of entities based on their components.

This function animates entities based on their animation components. It processes each entity in the entity manager and updates their animation based on the delta time provided.

Parameters

<i>componentManager</i>	The component manager used to access entity components.
<i>entityManager</i>	The entity manager used to access entities.
<i>deltaTime</i>	The time elapsed since the last update, used to update animations.

This function iterates through all entities and updates their animation states based on the presence and values of specific components such as [AnimationComponent](#), [PlayerComponent](#), [VelocityComponent](#), and [BackgroundComponent](#).

Parameters

<i>componentManager</i>	Reference to the ComponentManager that handles components.
<i>entityManager</i>	Reference to the EntityManager that handles entities.
<i>deltaTime</i>	The time elapsed since the last update, used for time-based animations.

6.6.4 Member Data Documentation

6.6.4.1 `_componentManager`

`ComponentManager& AnimationSystem::_componentManager [private]`

Reference to the `ComponentManager` instance.

This member variable holds a reference to the `ComponentManager`, which is responsible for managing all the components within the ECS (`Entity` Component System). It provides functionality to add, remove, and query components associated with entities.

6.6.4.2 `_entityManager`

`EntityManager& AnimationSystem::_entityManager [private]`

Reference to the `EntityManager` instance.

This member variable holds a reference to the `EntityManager`, which is responsible for managing all entities within the ECS (`Entity` Component System). It provides functionalities such as entity creation, deletion, and retrieval.

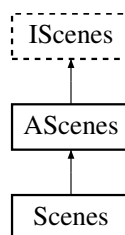
The documentation for this class was generated from the following files:

- `/home/runner/work/R-Type/R-Type/Server/Interface/Include/animation_system.hpp`
- `/home/runner/work/R-Type/R-Type/Server/Src/animation_system.cpp`

6.7 AScenes Class Reference

```
#include <a_scenes.hpp>
```

Inheritance diagram for AScenes:



Public Types

- enum class `Scene` {
`MAIN_MENU` , `GAME_LOOP` , `SETTINGS_MENU` , `IN_GAME_MENU` ,
`EXIT` }
Represents the different scenes in the R-Type client application.
- enum class `GameMode` { `EASY` , `MEDIUM` , `HARD` }
Enumeration to represent different game difficulty levels.
- enum class `DaltonismMode` { `NORMAL` , `TRITANOPIA` , `DEUTERANOPIA` , `PROTANOPIA` }
Enum representing different modes of color blindness (Daltonism).
- enum class `Actions` {
`UP` , `DOWN` , `LEFT` , `RIGHT` ,
`FIRE` , `PAUSE` , `QUIT` }
Enumeration representing possible actions in the game.
- enum class `SpriteType` {
`BACKGROUND` , `PLAYER` , `ALLY` , `ENEMY` ,
`FILTER` , `WEAPON` , `POWER_UP` , `UI` ,
`OTHER` }
Enumeration representing the type of sprite in the game.

Public Member Functions

- [AScenes](#) (std::string ip, int port)
- [~AScenes](#) ()=default
- void [setScene](#) ([Scene](#) scene)
Set the Scene object.
- [AScenes::Scene](#) [getPreviousScene](#) ()
Get the Previous Scene object.
- [DaltonismMode](#) [getDaltonism](#) () const
Get the Daltonism object.
- void [setDaltonism](#) ([DaltonismMode](#) const mode)
Set the Daltonism object.
- void [setGameMode](#) ([GameMode](#) const mode)
Set the Game Mode object.
- void [setDisplayDaltonismChoice](#) (bool const displayDaltonismChoice)
Sets the display option for Daltonism mode.
- bool [getDisplayDaltonismChoice](#) () const
Retrieves the current display setting for Daltonism (color blindness) mode.
- void [setDisplayGameModeChoice](#) (bool const displayGameModeChoice)
Sets the display state for the game mode choice.
- bool [getDisplayGameModeChoice](#) () const
Retrieves the current display game mode choice.
- void [setDisplayKeyBindsChoice](#) (bool const displayKeyBindsChoice)
Sets the display status for key binds choice.
- bool [getDisplayKeyBindsChoice](#) () const
Retrieves the current choice for displaying key bindings.
- void [setIp](#) (std::string ip)
Sets the IP address.
- void [setPort](#) (int port)
Sets the port number for the connection.
- std::string [getIp](#) () const
Retrieves the IP address.
- int [getPort](#) () const
Retrieves the port number.

Public Attributes

- std::map< [Actions](#), sf::Keyboard::Key > [keyBinds](#)
A map that binds game actions to specific keyboard keys.
- std::vector< std::shared_ptr< [Entity](#) > > [buttons](#)
A collection of shared pointers to [Entity](#) objects representing buttons.
- std::shared_ptr< [Entity](#) > [filter](#)
A shared pointer to an [Entity](#) object.

Protected Attributes

- `GameMode _currentGameMode = GameMode::MEDIUM`
Represents the current game mode.
- `DaltonismMode _currentDaltonismMode = DaltonismMode::NORMAL`
Enum representing different modes of Daltonism (color blindness).
- `Scene _currentScene = Scene::MAIN_MENU`
Represents the current scene in the application.
- `Scene _previousScene = Scene::MAIN_MENU`
Represents the previous scene in the application.
- `bool _displayDaltonismChoice = false`
Flag to indicate whether the Daltonism choice should be displayed.
- `bool _displayGameModeChoice = false`
Flag indicating whether the game mode choice should be displayed.
- `bool _displayKeyBindingsChoice = false`
Flag indicating whether the key bindings choice should be displayed.
- `std::string _ip`
The IP address of the server.
- `int _port`
The port number of the server.

6.7.1 Member Enumeration Documentation

6.7.1.1 Actions

```
enum AScenes::Actions [strong]
```

Enumeration representing possible actions in the game.

This enumeration defines the various actions that can be performed by the player in the game. The actions include:

- UP: Move up
- DOWN: Move down
- LEFT: Move left
- RIGHT: Move right
- FIRE: Fire a weapon
- PAUSE: Pause the game
- QUIT: Quit the game

Enumerator

UP	
DOWN	
LEFT	
RIGHT	
FIRE	
PAUSE	
QUIT	

6.7.1.2 DaltonismMode

```
enum AScenes::DaltonismMode [strong]
```

Enum representing different modes of color blindness (Daltonism).

This enum is used to specify the type of color blindness mode that can be applied.

Enumerator

NORMAL	Represents normal vision without any color blindness.
TRITANOPIA	Represents Tritanopia, a type of color blindness where blue and yellow colors are confused.
DEUTERANOPIA	Represents Deuteranopia, a type of color blindness where green and red colors are confused.
PROTANOPIA	Represents Protanopia, a type of color blindness where red and green colors are confused.

6.7.1.3 GameMode

```
enum AScenes::GameMode [strong]
```

Enumeration to represent different game difficulty levels.

This enumeration defines the various difficulty levels that can be selected in the game. The available modes are:

- EASY: Represents an easy difficulty level.
- MEDIUM: Represents a medium difficulty level.
- HARD: Represents a hard difficulty level.

Enumerator

EASY	
MEDIUM	
HARD	

6.7.1.4 Scene

```
enum AScenes::Scene [strong]
```

Represents the different scenes in the R-Type client application.

This enumeration defines the various scenes that the client can be in during its lifecycle.

Enumerator

MAIN_MENU	Represents the main menu scene.
GAME_LOOP	Represents the game loop scene where the main gameplay occurs.
SETTINGS_MENU	Represents the settings menu scene where the user can adjust settings.
IN_GAME_MENU	Represents the in-game menu scene that can be accessed during gameplay.
EXIT	Represents the exit scene where the application is closing.

6.7.1.5 SpriteType

```
enum AScenes::SpriteType [strong]
```

Enumeration representing the type of sprite in the game.

This enumeration defines the different sprite types that need to be identified in the game. The types include:

- BACKGROUND: Represents a background sprite.
- PLAYER: Represents a player sprite.
- ALLY: Represents an ally sprite.
- ENEMY: Represents an enemy sprite.
- OTHER: Represents any other type of sprite.

Enumerator

BACKGROUND	
PLAYER	
ALLY	
ENEMY	
FILTER	
WEAPON	
POWER_UP	
UI	
OTHER	

6.7.2 Constructor & Destructor Documentation

6.7.2.1 AScenes()

```
AScenes::AScenes (
    std::string ip,
    int port )
```

6.7.2.2 ~AScenes()

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

6.7.3 Member Function Documentation

6.7.3.1 getDaltonism()

```
DaltonismMode AScenes::getDaltonism ( ) const [inline]
```

Get the Daltonism object.

Returns

DaltonismMode

6.7.3.2 getDisplayDaltonismChoice()

```
bool AScenes::getDisplayDaltonismChoice ( ) const
```

Retrieves the current display setting for Daltonism (color blindness) mode.

Returns

true if Daltonism mode is enabled, false otherwise.

6.7.3.3 getDisplayGameModeChoice()

```
bool AScenes::getDisplayGameModeChoice ( ) const
```

Retrieves the current display game mode choice.

This function returns a boolean value indicating whether the game mode choice is currently set to be displayed.

Returns

true if the game mode choice is set to be displayed, false otherwise.

6.7.3.4 getDisplayKeyBindsChoice()

```
bool AScenes::getDisplayKeyBindsChoice ( ) const
```

Retrieves the current choice for displaying key bindings.

Returns

true if key bindings should be displayed, false otherwise.

6.7.3.5 getIp()

```
std::string AScenes::getIp ( ) const
```

Retrieves the IP address.

This function returns the IP address as a string.

Returns

std::string The IP address.

6.7.3.6 getPort()

```
int AScenes::getPort ( ) const
```

Retrieves the port number.

Returns

int The port number.

6.7.3.7 getPreviousScene()

```
AScenes::Scene AScenes::getPreviousScene ( )
```

Get the Previous Scene object.

Returns

Scene

6.7.3.8 setDaltonism()

```
void AScenes::setDaltonism (
    DaltonismMode const mode )
```

Set the Daltonism object.

Parameters

<i>mode</i>	The daltonism mode to set
-------------	---------------------------

6.7.3.9 setDisplayDaltonismChoice()

```
void AScenes::setDisplayDaltonismChoice (
    bool const displayDaltonismChoice )
```

Sets the display option for Daltonism mode.

This function enables or disables the display option for Daltonism mode based on the provided boolean value.

Parameters

<i>displayDaltonismChoice</i>	A boolean value indicating whether to display the Daltonism mode option (true) or not (false).
-------------------------------	------------------------------------------------------------------------------------------------

6.7.3.10 setDisplayGameModeChoice()

```
void AScenes::setDisplayGameModeChoice (
    bool const displayGameModeChoice )
```

Sets the display state for the game mode choice.

This function allows you to control whether the game mode choice should be displayed or not.

Parameters

<i>displayGameModeChoice</i>	A boolean value indicating whether the game mode choice should be displayed (true) or hidden (false).
------------------------------	-------------------------------------------------------------------------------------------------------

6.7.3.11 setDisplayKeyBindsChoice()

```
void AScenes::setDisplayKeyBindsChoice (
    bool const displayKeyBindsChoice )
```

Sets the display status for key binds choice.

This function allows you to enable or disable the display of key binds choice.

Parameters

<i>displayKeyBindsChoice</i>	A boolean value indicating whether to display the key binds choice (true) or not (false).
------------------------------	-------------------------------------------------------------------------------------------

6.7.3.12 setGameMode()

```
void AScenes::setGameMode (
    GameMode const mode )
```

Set the Game Mode object.

Parameters

<i>mode</i>	
-------------	--

6.7.3.13 setIp()

```
void AScenes::setIp (
    std::string ip )
```

Sets the IP address.

This function sets the IP address to the specified value.

Parameters

<i>ip</i>	The IP address to set as a string.
-----------	------------------------------------

6.7.3.14 setPort()

```
void AScenes::setPort (
    int port )
```

Sets the port number for the connection.

This function assigns the specified port number to be used for network communication.

Parameters

<i>port</i>	The port number to be set.
-------------	----------------------------

6.7.3.15 setScene()

```
void AScenes::setScene (
    AScenes::Scene scene )
```

Set the Scene object.

Parameters

<i>scene</i>	
--------------	--

6.7.4 Member Data Documentation

6.7.4.1 _currentDaltonismMode

```
DaltonismMode AScenes::_currentDaltonismMode = DaltonismMode::NORMAL [protected]
```

Enum representing different modes of Daltonism (color blindness).

This enum is used to specify the current Daltonism mode, which can be used to adjust the display settings for users with different types of color blindness.

Possible values:

- NORMAL: No color blindness.
- PROTANOPIA: Red color blindness.
- DEUTERANOPIA: Green color blindness.
- TRITANOPIA: Blue color blindness.

6.7.4.2 _currentGameMode

```
GameMode AScenes::_currentGameMode = GameMode::MEDIUM [protected]
```

Represents the current game mode.

This variable holds the current game mode of the game. It is initialized to `GameMode::MEDIUM` by default.

6.7.4.3 `_currentScene`

```
Scene AScenes::_currentScene = Scene::MAIN_MENU [protected]
```

Represents the current scene in the application.

This variable holds the current scene being displayed or interacted with in the application. It is initialized to the MAIN_MENU scene by default.

6.7.4.4 `_displayDaltonismChoice`

```
bool AScenes::_displayDaltonismChoice = false [protected]
```

Flag to indicate whether the Daltonism choice should be displayed.

6.7.4.5 `_displayGameModeChoice`

```
bool AScenes::_displayGameModeChoice = false [protected]
```

Flag indicating whether the game mode choice should be displayed.

6.7.4.6 `_displayKeyBindsChoice`

```
bool AScenes::_displayKeyBindsChoice = false [protected]
```

Flag indicating whether the key bindings choice should be displayed.

6.7.4.7 `_ip`

```
std::string AScenes::_ip [protected]
```

The IP address of the server.

This member variable stores the IP address of the server to which the client will connect. It is a string that contains the IP address in the format "xxx.xxx.xxx.xxx".

6.7.4.8 `_port`

```
int AScenes::_port [protected]
```

The port number of the server.

This member variable stores the port number of the server to which the client will connect. It is an integer that represents the port number on which the server is listening for incoming connections.

6.7.4.9 `_previousScene`

```
Scene AScenes::_previousScene = Scene::MAIN_MENU [protected]
```

Represents the previous scene in the application.

This variable holds the previous scene that was active before the current one. It is initialized to the MAIN_MENU scene by default.

6.7.4.10 `buttons`

```
std::vector<std::shared_ptr<Entity>> > AScenes::buttons
```

A collection of shared pointers to [Entity](#) objects representing buttons.

This vector holds shared pointers to [Entity](#) instances, which are used to represent buttons within the scene. The use of shared pointers ensures that the [Entity](#) objects are properly managed and their memory is automatically deallocated when they are no longer in use.

6.7.4.11 `filter`

```
std::shared_ptr<Entity> AScenes::filter
```

A shared pointer to an [Entity](#) object.

This smart pointer manages the lifetime of an [Entity](#) instance, ensuring that the [Entity](#) is properly deleted when no longer in use. It allows multiple parts of the program to share ownership of the [Entity](#).

6.7.4.12 `keyBinds`

```
std::map<Actions, sf::Keyboard::Key> AScenes::keyBinds
```

Initial value:

```
= {{Actions::UP, sf::Keyboard::Key::Up},
  {Actions::DOWN, sf::Keyboard::Key::Down}, {Actions::LEFT, sf::Keyboard::Key::Left},
  {Actions::RIGHT, sf::Keyboard::Key::Right}, {Actions::FIRE, sf::Keyboard::Key::Space},
  {Actions::PAUSE, sf::Keyboard::Key::Escape}, {Actions::QUIT, sf::Keyboard::Key::Q}}
```

A map that binds game actions to specific keyboard keys.

This map associates each action defined in the Actions enum with a corresponding key from the sf::Keyboard::Key enumeration. It is used to handle user input by mapping key presses to game actions.

The key bindings are as follows:

- [Actions::UP](#) -> sf::Keyboard::Key::Up
- [Actions::DOWN](#) -> sf::Keyboard::Key::Down
- [Actions::LEFT](#) -> sf::Keyboard::Key::Left
- [Actions::RIGHT](#) -> sf::Keyboard::Key::Right
- [Actions::FIRE](#) -> sf::Keyboard::Key::Space
- [Actions::PAUSE](#) -> sf::Keyboard::Key::Escape
- [Actions::QUIT](#) -> sf::Keyboard::Key::Q

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

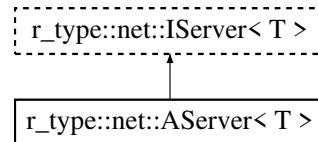
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp
- /home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp

6.8 r_type::net::AServer< T > Class Template Reference

[AServer](#) class template for managing server operations.

```
#include <a_server.hpp>
```

Inheritance diagram for r_type::net::AServer< T >:



Public Member Functions

- [AServer](#) (uint16_t port)
Constructs an [AServer](#) object with the specified port.
- [~AServer](#) ()
Destructor for the [AServer](#) class.
- bool [Start](#) ()
Starts the server and begins waiting for client messages.
- void [Stop](#) ()
Stops the server.
- void [WaitForClientMessage](#) ()
Waits for a client message asynchronously.
- void [MessageClient](#) (std::shared_ptr< Connection< T >> client, const Message< T > &msg)
Sends a message to a specific client if the client is connected.
- void [MessageAllClients](#) (const Message< T > &msg, std::shared_ptr< Connection< T >> plgnore← Client=nullptr)
Sends a message to all connected clients, optionally ignoring a specified client.
- [UIEntityInformation UpdateInfoBar](#) (int playerId)
Updates the information bar for a given player.
- void [Update](#) (size_t nMaxMessages=-1, bool bWait=false)
Updates the server state, processes incoming messages, and updates the game level.
- void [UpdatePlayerPosition](#) (PlayerMovement direction, uint32_t entityId) override
Updates the position of an entity based on the message received and the client ID.
- void [SavePlayerScore](#) (uint32_t playerId)
Saves the score of a player to a file.
- uint32_t [GetClientPlayerId](#) (uint32_t id)
Retrieves the entity ID associated with a client ID.
- uint32_t [GetPlayerClientId](#) (uint32_t id)
Retrieves the client ID associated with a given player ID.
- uint32_t [GetClientInfoBarId](#) (uint32_t id)
Retrieves the client info bar ID associated with a given client ID.
- void [RemovePlayer](#) (uint32_t id)
Removes a player from the server.
- void [RemoveEntity](#) (uint32_t id)
Removes an entity from the server.
- void [RemoveInfoBar](#) (uint32_t infoBarId)

- Removes an information bar and its associated entities.*
 - [EntityInformation InitiatePlayer](#) (int clientId)
 - Initializes a new player entity and assigns a random position.*
 - [UIEntityInformation InitInfoBar](#) (int clientId)
 - [EntityInformation FormatEntityInformation](#) (uint32_t entityId)
 - Formats the information of a given entity into an [EntityInformation](#) structure.*
 - [EntityInformation InitiatePlayerMissile](#) (int entityId)
 - Initializes a missile entity associated with a player.*
 - [EntityInformation InitiateEnemyMissile](#) (int enemyId)
 - [EntityInformation InitiateWeaponForce](#) (int entityId)
 - `std::shared_ptr< Connection< T > >` [getClientById](#) (const `std::deque< std::shared_ptr< Connection< T > >>` &connections, uint32_t clientId)
 - virtual void [OnClientValidated](#) (`std::shared_ptr< Connection< T > >` client)
 - Callback function that is called when a client has been successfully validated.*
 - [ComponentManager GetComponentManager](#) () override
 - Retrieves the component manager associated with the server.*
 - [EntityManager & GetEntityManager](#) () override
 - Retrieves the entity manager associated with the server.*
 - [EntityFactory & GetEntityFactory](#) () override
 - Retrieves the entity factory associated with the server.*
 - `std::chrono::system_clock::time_point` [GetClock](#) () override
 - Retrieves the current clock time of the server.*
 - void [SetClock](#) (`std::chrono::system_clock::time_point` clock)
 - Set the Clock object.*

Public Attributes

- `ThreadSafeQueue< OwnedMessage< T > >` [_qMessagesIn](#)
- Thread-safe queue to store incoming messages.*
- `std::deque< std::shared_ptr< Connection< T > > >` [_deqConnections](#)
- A deque that holds shared pointers to Connection objects.*
- `asio::io_context` [_asioContext](#)
- The io_context object provides I/O services, such as sockets, that the server will use.*
- `std::thread` [_threadContext](#)
- Thread object for managing the server's context operations.*
- `asio::ip::udp::socket` [_asioSocket](#)
- A socket for sending and receiving UDP datagrams.*
- `asio::ip::udp::endpoint` [_clientEndpoint](#)
- Represents the endpoint of a client in a UDP connection.*
- `std::array< uint8_t, 1024 >` [_tempBuffer](#)
- Temporary buffer used for storing data.*
- `uint32_t` [_nIDCounter](#) = 10000
- Counter for generating unique network IDs.*
- [ComponentManager _componentManager](#)
- Manages and maintains the lifecycle of various components within the server.*
- [EntityManager _entityManager](#)
- Manages the lifecycle and operations of entities within the server.*
- [EntityFactory _entityFactory](#)
- An instance of [EntityFactory](#) used to create and manage game entities.*
- `std::unordered_map< uint32_t, uint32_t >` [_clientPlayerID](#)

- A container that maps client IDs to player IDs.*
- `std::unordered_map< uint32_t, uint32_t > _clientInfoBarID`
- `int _nbrOfPlayers = 0`
Number of players currently connected to the server.
- `std::chrono::system_clock::time_point _clock = std::chrono::system_clock::now()`
Stores the current time point from the system clock.
- `bool _playerConnected = false`
- `EntityInformation _background`
Holds information about the background entity.
- `int _port`
- `r_type::Level< T > _level`

Protected Member Functions

- virtual `bool OnClientConnect (std::shared_ptr< Connection< T >> client)`
on client connect event
- virtual `void OnClientDisconnect (std::shared_ptr< Connection< T >> client)`
on client disconnect event
- virtual `void OnMessage (std::shared_ptr< Connection< T >> client, Message< T > &msg)`
on message event

6.8.1 Detailed Description

```
template<typename T>
class r_type::net::AServer< T >
```

[AServer](#) class template for managing server operations.

This class template provides a framework for creating and managing a server that handles client connections, messages, and entity updates. It uses the ASIO library for asynchronous network communication and provides various functions for server operations such as starting, stopping, and updating the server, as well as handling client messages and connections.

Template Parameters

<code>T</code>	The type of data that the server handles.
----------------	-------------------------------------------

6.8.2 Constructor & Destructor Documentation

6.8.2.1 AServer()

```
template<typename T >
r_type::net::AServer< T >::AServer (
    uint16_t port ) [inline]
```


Constructs an [AServer](#) object with the specified port.

This constructor initializes the server with the given port number and sets up the necessary components for the server to function. It initializes the ASIO socket with the provided port and creates instances of [EntityManager](#), [EntityFactory](#), and [ComponentManager](#). Additionally, it initiates the background process and creates three basic monster entities using the entity factory.

Parameters

<i>port</i>	The port number on which the server will listen for incoming connections.
-------------	---------------------------------------------------------------------------

6.8.2.2 `~AServer()`

```
template<typename T >
r_type::net::AServer< T >::~~AServer ( ) [inline]
```

Destructor for the [AServer](#) class.

This destructor ensures that the server is properly stopped by calling the [Stop\(\)](#) method when an instance of [AServer](#) is destroyed.

6.8.3 Member Function Documentation

6.8.3.1 `FormatEntityInformation()`

```
template<typename T >
EntityInformation r_type::net::AServer< T >::FormatEntityInformation (
    uint32_t entityId ) [inline]
```

Formats the information of a given entity into an [EntityInformation](#) structure.

This function retrieves the position and sprite data components of the specified entity and populates an [EntityInformation](#) structure with this data. If the entity has both position and sprite data components, their values are copied into the [EntityInformation](#) structure. If either component is missing, the [EntityInformation](#) structure will be returned with default values.

Parameters

<i>entity</i>	The entity whose information is to be formatted.
---------------	--------------------------------------------------

Returns

[EntityInformation](#) The formatted information of the entity.

6.8.3.2 getClientById()

```
template<typename T >
std::shared_ptr<Connection<T> > r_type::net::AServer< T >::getClientById (
    const std::deque< std::shared_ptr< Connection< T >>> & connections,
    uint32_t clientId ) [inline]
```

6.8.3.3 GetClientInfoBarId()

```
template<typename T >
uint32_t r_type::net::AServer< T >::GetClientInfoBarId (
    uint32_t id ) [inline]
```

Retrieves the client info bar ID associated with a given client ID.

Parameters

<i>id</i>	The client ID for which to retrieve the info bar ID.
-----------	------------------------------------------------------

Returns

uint32_t The info bar ID associated with the specified client ID.

6.8.3.4 GetClientPlayerId()

```
template<typename T >
uint32_t r_type::net::AServer< T >::GetClientPlayerId (
    uint32_t id ) [inline]
```

Retrieves the entity ID associated with a client ID.

Parameters

<i>id</i>	The client ID.
-----------	----------------

Returns

uint32_t The entity ID associated with the client.

6.8.3.5 GetClock()

```
template<typename T >
std::chrono::system_clock::time_point r_type::net::AServer< T >::GetClock ( ) [inline], [override]
```

Retrieves the current clock time of the server.

This function returns the current time point of the server's clock, which can be used for time-related calculations, such as updating game state, handling animations, or scheduling events. It provides a consistent reference point for the server's operations.

Returns

`std::chrono::system_clock::time_point` The current time point of the server's clock.

6.8.3.6 `GetComponentManager()`

```
template<typename T >
ComponentManager r_type::net::AServer< T >::GetComponentManager ( ) [inline], [override]
```

Retrieves the component manager associated with the server.

This function provides access to the component manager, which is responsible for managing the components associated with entities in the game. It allows for the retrieval and manipulation of entity components, enabling the game logic to interact with them as needed.

Returns

`ComponentManager`& A reference to the component manager instance.

6.8.3.7 `GetEntityFactory()`

```
template<typename T >
EntityFactory& r_type::net::AServer< T >::GetEntityFactory ( ) [inline], [override]
```

Retrieves the entity factory associated with the server.

This function provides access to the entity factory, which is responsible for creating new entities in the game. The entity factory provides methods to instantiate various types of entities, such as players, missiles, and background elements, ensuring that they are correctly initialized with the necessary components.

Returns

`EntityFactory`& A reference to the entity factory instance.

6.8.3.8 GetEntityManager()

```
template<typename T >
EntityManager& r_type::net::AServer< T >::GetEntityManager ( ) [inline], [override]
```

Retrieves the entity manager associated with the server.

This function returns the entity manager responsible for creating, managing, and removing entities in the game. The entity manager handles the lifecycle of entities and ensures that they are correctly processed within the game's systems.

Returns

[EntityManager](#)& A reference to the entity manager instance.

6.8.3.9 GetPlayerClientId()

```
template<typename T >
uint32_t r_type::net::AServer< T >::GetPlayerClientId (
    uint32_t id ) [inline]
```

Retrieves the client ID associated with a given player ID.

This function searches through the `_clientPlayerID` map to find the client ID that corresponds to the provided player ID. If the player ID is found, the associated client ID is returned. If the player ID is not found, a [playerIdNotFound](#) exception is thrown.

Parameters

<i>id</i>	The player ID for which the client ID is to be retrieved.
-----------	-----------------------------------------------------------

Returns

uint32_t The client ID associated with the given player ID.

Exceptions

playerIdNotFound	If the player ID is not found in the map.
----------------------------------	-------------------------------------------

6.8.3.10 InitiateEnemyMissile()

```
template<typename T >
EntityInformation r_type::net::AServer< T >::InitiateEnemyMissile (
    int enemyId ) [inline]
```

6.8.3.11 `InitiatePlayer()`

```
template<typename T >
EntityInformation r_type::net::AServer< T >::InitiatePlayer (
    int clientId ) [inline]
```

Initializes a new player entity and assigns a random position.

The function creates a new player entity, assigns it a random position, and ensures that it does not overlap with any other players.

Parameters

<i>clientId</i>	The client ID of the player being initialized.
-----------------	------------------------------------------------

Returns

[EntityInformation](#) The information of the newly created player entity.

6.8.3.12 `InitiatePlayerMissile()`

```
template<typename T >
EntityInformation r_type::net::AServer< T >::InitiatePlayerMissile (
    int entityId ) [inline]
```

Initializes a missile entity associated with a player.

The function creates a missile entity associated with a player and assigns its position based on the player's current position.

Parameters

<i>clientId</i>	The client ID of the player firing the missile.
-----------------	-------------------------------------------------

Returns

[EntityInformation](#) The information of the newly created missile entity.

6.8.3.13 `InitiateWeaponForce()`

```
template<typename T >
EntityInformation r_type::net::AServer< T >::InitiateWeaponForce (
    int entityId ) [inline]
```

6.8.3.14 InitInfoBar()

```
template<typename T >
UIEntityInformation r_type::net::AServer< T >::InitInfoBar (
    int clientId ) [inline]
```

6.8.3.15 MessageAllClients()

```
template<typename T >
void r_type::net::AServer< T >::MessageAllClients (
    const Message< T > & msg,
    std::shared_ptr< Connection< T >> pIgnoreClient = nullptr ) [inline]
```

Sends a message to all connected clients, optionally ignoring a specified client.

This function iterates through all the connections in the server and sends the provided message to each connected client, except for the client specified by `pIgnoreClient`. If a client is found to be disconnected, it triggers the disconnection handler and removes the client from the list of connections.

Template Parameters

<i>T</i>	The type of the message.
----------	--------------------------

Parameters

<i>msg</i>	The message to be sent to all clients.
<i>pIgnoreClient</i>	A shared pointer to a client connection that should be ignored. Defaults to nullptr.

6.8.3.16 MessageClient()

```
template<typename T >
void r_type::net::AServer< T >::MessageClient (
    std::shared_ptr< Connection< T >> client,
    const Message< T > & msg ) [inline]
```

Sends a message to a specific client if the client is connected.

If the client is not connected, it handles the client disconnection.

Template Parameters

<i>T</i>	The type of the message.
----------	--------------------------

Parameters

<i>client</i>	A shared pointer to the client connection.
---------------	--------------------------------------------

Parameters

<i>msg</i>	The message to be sent to the client.
------------	---------------------------------------

6.8.3.17 OnClientConnect()

```
template<typename T >
virtual bool r_type::net::AServer< T >::OnClientConnect (
    std::shared_ptr< Connection< T >> client ) [inline], [protected], [virtual]
```

on client connect event

Parameters

<i>client</i>	
---------------	--

Returns

true
false

6.8.3.18 OnClientDisconnect()

```
template<typename T >
virtual void r_type::net::AServer< T >::OnClientDisconnect (
    std::shared_ptr< Connection< T >> client ) [inline], [protected], [virtual]
```

on client disconnect event

Parameters

<i>client</i>	
---------------	--

6.8.3.19 OnClientValidated()

```
template<typename T >
virtual void r_type::net::AServer< T >::OnClientValidated (
    std::shared_ptr< Connection< T >> client ) [inline], [virtual]
```

Callback function that is called when a client has been successfully validated.

This function is intended to be overridden by derived classes to handle any specific actions that need to be taken when a client is validated.

Parameters

<i>client</i>	A shared pointer to the validated client connection.
---------------	------------------------------------------------------

6.8.3.20 OnMessage()

```
template<typename T >
virtual void r_type::net::AServer< T >::OnMessage (
    std::shared_ptr< Connection< T >> client,
    Message< T > & msg ) [inline], [protected], [virtual]
```

on message event

Parameters

<i>client</i>	
<i>msg</i>	

6.8.3.21 RemoveEntity()

```
template<typename T >
void r_type::net::AServer< T >::RemoveEntity (
    uint32_t id ) [inline]
```

Removes an entity from the server.

This function removes an entity identified by the given ID from the server. It first checks if the entity exists using the entity manager. If the entity is found, it removes the entity from all components using the component manager and then removes the entity itself from the entity manager.

Parameters

<i>id</i>	The unique identifier of the entity to be removed.
-----------	----------------------------------------------------

6.8.3.22 RemoveInfoBar()

```
template<typename T >
void r_type::net::AServer< T >::RemoveInfoBar (
    uint32_t infoBarId ) [inline]
```

Removes an information bar and its associated entities.

This function removes an information bar identified by the given `infoBarId`. It first checks if the information bar has a `TextDataComponent` and removes all entities associated with the categories listed in the `TextDataComponent`. Finally, it removes the information bar entity itself and erases its ID from the client information bar ID map.

Parameters

<i>info↔ BarId</i>	The ID of the information bar to be removed.
------------------------	----------------------------------------------

6.8.3.23 RemovePlayer()

```
template<typename T >
void r_type::net::AServer< T >::RemovePlayer (
    uint32_t id ) [inline]
```

Removes a player from the server.

This function removes a player identified by the given ID from the server's internal player list.

Parameters

<i>id</i>	The unique identifier of the player to be removed.
-----------	----------------------------------------------------

6.8.3.24 SavePlayerScore()

```
template<typename T >
void r_type::net::AServer< T >::SavePlayerScore (
    uint32_t playerId ) [inline]
```

Saves the score of a player to a file.

This function saves the score of a player identified by the given `playerId` to a file named "scores.txt" located in the "GameScores" directory. If the directory or file does not exist, they will be created. The score is appended to the file in the format "Player <playerId>: <score>".

Parameters

<i>player↔ Id</i>	The unique identifier of the player whose score is to be saved.
-----------------------	-----------------------------------------------------------------

Exceptions

<i>failedToCreateFile</i>	If the file cannot be created.
<i>failedToOpenFile</i>	If the file cannot be opened in append mode.

6.8.3.25 SetClock()

```
template<typename T >
void r_type::net::AServer< T >::SetClock (
    std::chrono::system_clock::time_point clock ) [inline]
```

Set the Clock object.

Parameters

<i>clock</i>	
--------------	--

6.8.3.26 Start()

```
template<typename T >
bool r_type::net::AServer< T >::Start ( ) [inline]
```

Starts the server and begins waiting for client messages.

This function attempts to start the server by waiting for client messages and running the ASIO context in a separate thread. If an exception occurs during this process, it will be caught, an error message will be printed to the standard error stream, and the function will return false.

Returns

true if the server started successfully, false otherwise.

6.8.3.27 Stop()

```
template<typename T >
void r_type::net::AServer< T >::Stop ( ) [inline]
```

Stops the server.

This function stops the server by stopping the ASIO context and joining the thread context. It also prints a message indicating that the server has been stopped.

6.8.3.28 `Update()`

```
template<typename T >
void r_type::net::AServer< T >::Update (
    size_t nMaxMessages = -1,
    bool bWait = false ) [inline]
```

Updates the server state, processes incoming messages, and updates the game level.

This function performs several tasks:

- If no players are connected, it returns immediately.
- If players are connected and the player connection flag is not set, it sets the flag and updates the clock.
- Spawns a thread to update the game level.
- Processes up to `nMaxMessages` from the incoming message queue.
- Joins the level update thread and updates the clock if entities were updated.

Parameters

<i>nMaxMessages</i>	The maximum number of messages to process from the incoming message queue. Default is -1 (process all messages).
<i>bWait</i>	A flag indicating whether to wait for messages. Default is false.

6.8.3.29 `UpdateInfoBar()`

```
template<typename T >
UIEntityInformation r_type::net::AServer< T >::UpdateInfoBar (
    int playerId ) [inline]
```

Updates the information bar for a given player.

This function retrieves the health and score components of the specified player, as well as the sprite and text data components of the player's information bar. It then updates the `UIEntityInformation` structure with these values.

Parameters

<i>playerId</i>	The ID of the player whose information bar is to be updated.
-----------------	--------------------------------------------------------------

Returns

`UIEntityInformation` The updated information for the player's information bar.

6.8.3.30 UpdatePlayerPosition()

```
template<typename T >
void r_type::net::AServer< T >::UpdatePlayerPosition (
    PlayerMovement direction,
    uint32_t entityId ) [inline], [override]
```

Updates the position of an entity based on the message received and the client ID.

This function updates the position of an entity. If the entity is not touching any other player, it updates its position and sends a message to all clients about the new position. If it touches another player, a destroy message is sent to all clients.

Parameters

<i>msg</i>	The message containing the new position of the entity.
<i>clientId</i>	The ID of the client sending the update.

6.8.3.31 WaitForClientMessage()

```
template<typename T >
void r_type::net::AServer< T >::WaitForClientMessage ( ) [inline]
```

Waits for a client message asynchronously.

This function waits for a client message by asynchronously receiving data from the socket. When a message is received, it checks if the client endpoint protocol is UDPv4. If the protocol is not UDPv4, it recursively calls itself to wait for another client message. If the protocol is UDPv4 and there are no errors, it prints the client endpoint and checks if a connection already exists. If a connection already exists, it returns without further processing. If a connection does not exist, it creates a new client socket, binds it to a local endpoint, and creates a new connection object. It then calls the OnClientConnect function to check if the client connection is approved. If the connection is approved, it adds the new connection to the list of connections, connects it to the client, and prints the connection ID. If the connection is denied, it prints a message indicating the connection was denied. If there is an error during the receive operation, it prints the error message..

6.8.4 Member Data Documentation

6.8.4.1 _asioContext

```
template<typename T >
asio::io_context r_type::net::AServer< T >::_asioContext
```

The io_context object provides I/O services, such as sockets, that the server will use.

This member variable is responsible for managing asynchronous I/O operations. It is part of the ASIO library, which is used for network programming.

6.8.4.2 `_asioSocket`

```
template<typename T >
asio::ip::udp::socket r_type::net::AServer< T >::_asioSocket
```

A socket for sending and receiving UDP datagrams.

This member variable represents a UDP socket using the ASIO library. It is used for network communication in the server.

6.8.4.3 `_background`

```
template<typename T >
EntityInformation r_type::net::AServer< T >::_background
```

Holds information about the background entity.

This member variable stores the details related to the background entity in the game. It includes properties such as position, texture, and other relevant attributes that define the background's appearance and behavior.

6.8.4.4 `_clientEndpoint`

```
template<typename T >
asio::ip::udp::endpoint r_type::net::AServer< T >::_clientEndpoint
```

Represents the endpoint of a client in a UDP connection.

This member variable holds the endpoint information (IP address and port) of a client in a UDP connection using the ASIO library.

6.8.4.5 `_clientInfoBarID`

```
template<typename T >
std::unordered_map<uint32_t, uint32_t> r_type::net::AServer< T >::_clientInfoBarID
```

6.8.4.6 `_clientPlayerID`

```
template<typename T >
std::unordered_map<uint32_t, uint32_t> r_type::net::AServer< T >::_clientPlayerID
```

A container that maps client IDs to player IDs.

left: client ID right: player ID

This unordered map is used to associate client IDs with their corresponding player IDs. The keys are of type `uint32_t` representing the client IDs, and the values are also of type `uint32_t` representing the player IDs.

6.8.4.7 `_clock`

```
template<typename T >
std::chrono::system_clock::time_point r_type::net::AServer< T >::_clock = std::chrono::system_↵
_clock::now()
```

Stores the current time point from the system clock.

This variable is initialized with the current time using `std::chrono::system_clock::now()` and represents a specific point in time according to the system clock.

6.8.4.8 `_componentManager`

```
template<typename T >
ComponentManager r_type::net::AServer< T >::_componentManager
```

Manages and maintains the lifecycle of various components within the server.

The `ComponentManager` is responsible for creating, updating, and destroying components as needed. It ensures that all components are properly managed and that their states are consistent throughout the server's operation.

6.8.4.9 `_deqConnections`

```
template<typename T >
std::deque<std::shared_ptr<Connection<T> > > r_type::net::AServer< T >::_deqConnections
```

A deque that holds shared pointers to `Connection` objects.

This member variable is used to manage a collection of active connections. The use of `std::shared_ptr` ensures that the `Connection` objects are reference-counted and automatically deallocated when no longer in use.

Template Parameters

<code>T</code>	The type of data that the <code>Connection</code> handles.
----------------	------------------------------------------------------------

6.8.4.10 `_entityFactory`

```
template<typename T >
EntityFactory r_type::net::AServer< T >::_entityFactory
```

An instance of `EntityFactory` used to create and manage game entities.

6.8.4.11 `_entityManager`

```
template<typename T >
EntityManger r_type::net::AServer< T >::_entityManager
```

Manages the lifecycle and operations of entities within the server.

The `EntityManager` is responsible for creating, updating, and deleting entities. It ensures that entities are properly managed and synchronized within the server's environment.

6.8.4.12 `_level`

```
template<typename T >
r_type::Level<T> r_type::net::AServer< T >::_level
```

6.8.4.13 `_nbrOfPlayers`

```
template<typename T >
int r_type::net::AServer< T >::_nbrOfPlayers = 0
```

Number of players currently connected to the server.

6.8.4.14 `_nIDCounter`

```
template<typename T >
uint32_t r_type::net::AServer< T >::_nIDCounter = 10000
```

Counter for generating unique network IDs.

This variable is used to keep track of the current ID to be assigned for network-related entities. It starts at 10000 and increments with each new ID generation.

6.8.4.15 `_playerConnected`

```
template<typename T >
bool r_type::net::AServer< T >::_playerConnected = false
```

6.8.4.16 `_port`

```
template<typename T >
int r_type::net::AServer< T >::_port
```

6.8.4.17 `_qMessagesIn`

```
template<typename T >
ThreadSafeQueue<OwnedMessage<T> > r_type::net::AServer< T >::_qMessagesIn
```

Thread-safe queue to store incoming messages.

This member variable is a thread-safe queue that holds messages of type `OwnedMessage<T>`. It ensures that messages can be safely accessed and modified by multiple threads concurrently.

6.8.4.18 `_tempBuffer`

```
template<typename T >
std::array<uint8_t, 1024> r_type::net::AServer< T >::_tempBuffer
```

Temporary buffer used for storing data.

This buffer is an array of 1024 bytes (`uint8_t`) used for temporary storage of data within the server's network interface.

6.8.4.19 `_threadContext`

```
template<typename T >
std::thread r_type::net::AServer< T >::_threadContext
```

Thread object for managing the server's context operations.

This member variable represents a thread that handles the server's context, allowing for concurrent execution of tasks related to the server's operation. It is used to ensure that the server can perform its duties without blocking the main execution flow.

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

- `/home/runner/work/R-Type/R-Type/Server/Interface/Include/level.hpp`
- `/home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/a_server.hpp`

6.9 AudioManager Class Reference

Manages and caches sound buffers for efficient audio playback.

```
#include <audio_manager.hpp>
```

Public Member Functions

- `sf::SoundBuffer & getSoundBuffer` (const `std::string &filePath`)
Retrieves a sound buffer from the specified file path.

Private Attributes

- `std::unordered_map< std::string, std::shared_ptr< sf::SoundBuffer > >` [soundBuffers](#)

A map that associates sound buffer names with their corresponding shared pointers to `sf::SoundBuffer` objects.

6.9.1 Detailed Description

Manages and caches sound buffers for efficient audio playback.

The [AudioManager](#) class is responsible for loading, caching, and retrieving sound buffers. It ensures that sound buffers are loaded only once and reused efficiently throughout the application.

Note

This class uses the SFML library for handling sound buffers.

6.9.2 Member Function Documentation

6.9.2.1 `getSoundBuffer()`

```
sf::SoundBuffer& AudioManager::getSoundBuffer (
    const std::string & filePath ) [inline]
```

Retrieves a sound buffer from the specified file path.

This function checks if the sound buffer is already cached. If it is, the cached sound buffer is returned. Otherwise, it loads the sound buffer from the file, caches it, and then returns it.

Parameters

<i>filePath</i>	The path to the sound file.
-----------------	-----------------------------

Returns

A reference to the sound buffer.

Exceptions

<i>std::runtime_error</i>	If the sound buffer fails to load from the file.
---------------------------	--------------------------------------------------

6.9.3 Member Data Documentation

6.9.3.1 soundBuffers

```
std::unordered_map<std::string, std::shared_ptr<sf::SoundBuffer> > AudioManager::soundBuffers
[private]
```

A map that associates sound buffer names with their corresponding shared pointers to sf::SoundBuffer objects.

This unordered map is used to store and manage sound buffers by their names, allowing for efficient retrieval and usage of sound resources in the application. Each entry in the map consists of a string key representing the name of the sound buffer and a shared pointer to an sf::SoundBuffer object, which ensures proper memory management and resource sharing.

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

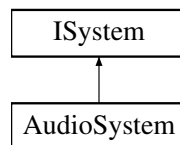
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/[audio_manager.hpp](#)

6.10 AudioSystem Class Reference

Manages audio playback within the application.

```
#include <audio_system.hpp>
```

Inheritance diagram for AudioSystem:



Public Member Functions

- [AudioSystem](#) (std::shared_ptr< [AudioManager](#) > audioManager)
Constructs an [AudioSystem](#) object.
- void [playBackgroundMusic](#) (const std::string &filePath)
Plays background music from the specified file.
- void [stopBackgroundMusic](#) ()
Stops the background music that is currently playing.
- void [playSoundEffect](#) (const std::string &filePath)
Plays a sound effect from the specified file.

Private Attributes

- std::shared_ptr< [AudioManager](#) > [_audioManager](#)
A shared pointer to the [AudioManager](#) instance.
- sf::Music [_backgroundMusic](#)
A class that provides functionality for playing music.
- std::string [_currentMusicFilePath](#)
Stores the file path of the currently playing music.
- sf::Sound [_soundEffect](#)
Represents a sound effect that can be played in the audio system.

6.10.1 Detailed Description

Manages audio playback within the application.

The [AudioSystem](#) class provides functionalities for playing background music and sound effects. It utilizes the [AudioManager](#) for managing audio resources and the SFML library for audio playback.

This class is responsible for handling audio playback in the application. It allows for playing background music and sound effects from specified file paths. The class ensures proper management of audio resources through the use of `std::shared_ptr` for the [AudioManager](#) instance.

Note

The [AudioSystem](#) class relies on the SFML library for audio playback functionalities. Ensure that the SFML library is properly included and linked in your project.

See also

[AudioManager](#)

`sf::Music`

`sf::Sound`

6.10.2 Constructor & Destructor Documentation

6.10.2.1 AudioSystem()

```
AudioSystem::AudioSystem (
    std::shared_ptr< AudioManager > audioManager ) [inline]
```

Constructs an [AudioSystem](#) object.

Parameters

<i>audioManager</i>	A shared pointer to an AudioManager instance.
---------------------	---------------------------------------------------------------

6.10.3 Member Function Documentation

6.10.3.1 playBackgroundMusic()

```
void AudioSystem::playBackgroundMusic (
    const std::string & filePath )
```

Plays background music from the specified file.

This function loads and plays background music from the given file path. It is typically used to provide ambient music for the application.

Parameters

<i>filePath</i>	The path to the audio file to be played as background music.
-----------------	--------------------------------------------------------------

6.10.3.2 playSoundEffect()

```
void AudioSystem::playSoundEffect (
    const std::string & filePath )
```

Plays a sound effect from the specified file.

This function loads and plays a sound effect from the given file path. It is useful for triggering sound effects in response to game events.

Parameters

<i>filePath</i>	The path to the sound effect file to be played.
-----------------	-------------------------------------------------

6.10.3.3 stopBackgroundMusic()

```
void AudioSystem::stopBackgroundMusic ( )
```

Stops the background music that is currently playing.

This function halts any background music that is being played by the audio system. It can be used to stop the music when it is no longer needed or when transitioning to a different scene or state in the application.

6.10.4 Member Data Documentation

6.10.4.1 _audioManager

```
std::shared_ptr<AudioManager> AudioSystem::_audioManager [private]
```

A shared pointer to the [AudioManager](#) instance.

This member variable holds a shared pointer to an [AudioManager](#) object, which is responsible for managing audio resources and playback within the system. The use of `std::shared_ptr` ensures that the [AudioManager](#) instance is properly managed and deallocated when no longer in use.

6.10.4.2 `_backgroundMusic`

```
sf::Music AudioSystem::_backgroundMusic [private]
```

A class that provides functionality for playing music.

The `sf::Music` class allows for streaming audio from a file or memory. It is particularly useful for playing large audio files, such as background music, as it does not load the entire file into memory.

6.10.4.3 `_currentMusicFilePath`

```
std::string AudioSystem::_currentMusicFilePath [private]
```

Stores the file path of the currently playing music.

6.10.4.4 `_soundEffect`

```
sf::Sound AudioSystem::_soundEffect [private]
```

Represents a sound effect that can be played in the audio system.

This member variable is an instance of the `sf::Sound` class from the SFML library, which is used to handle the playback of short sound effects. It provides functionalities to play, pause, stop, and manipulate sound properties such as volume, pitch, and loop status.

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

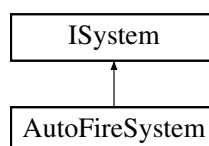
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/audio_system.hpp](#)
- [/home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_system.cpp](#)

6.11 AutoFireSystem Class Reference

A system that handles automatic firing mechanisms for entities.

```
#include <auto_fire_system.hpp>
```

Inheritance diagram for `AutoFireSystem`:



Public Member Functions

- `AutoFireSystem` ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager)
 - void `handleAutoFire` ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager)
- Handles the automatic firing mechanism for entities.*

Private Attributes

- [ComponentManager](#) & [_componentManager](#)
Reference to the [ComponentManager](#) instance.
- [EntityManager](#) & [_entityManager](#)
Reference to the [EntityManager](#) instance.

6.11.1 Detailed Description

A system that handles automatic firing mechanisms for entities.

System responsible for handling automatic firing mechanisms in entities.

The [AutoFireSystem](#) class is responsible for managing the automatic firing behavior of entities within the ECS framework. It interacts with the [ComponentManager](#) and [EntityManager](#) to update and control the firing state of entities.

Parameters

<i>componentManager</i>	Reference to the ComponentManager instance.
<i>entityManager</i>	Reference to the EntityManager instance.
<i>componentManager</i>	Reference to the ComponentManager that manages all components.
<i>entityManager</i>	Reference to the EntityManager that manages all entities.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 AutoFireSystem()

```
AutoFireSystem::AutoFireSystem (  
    ComponentManager & componentManager,  
    EntityManager & entityManager ) [inline]
```

6.11.3 Member Function Documentation

6.11.3.1 handleAutoFire()

```
void AutoFireSystem::handleAutoFire (  
    ComponentManager & componentManager,  
    EntityManager & entityManager )
```

Handles the automatic firing mechanism for entities.

This function processes entities that have the auto-fire capability and triggers their firing actions based on the game logic and conditions.

Parameters

<i>componentManager</i>	Reference to the ComponentManager that manages all components.
<i>entityManager</i>	Reference to the EntityManager that manages all entities.

6.11.4 Member Data Documentation

6.11.4.1 `_componentManager`

```
ComponentManager& AutoFireSystem::_componentManager [private]
```

Reference to the [ComponentManager](#) instance.

This member variable holds a reference to the [ComponentManager](#), which is responsible for managing all the components within the ECS ([Entity](#) Component System). It is used by the system to access and manipulate components associated with entities.

6.11.4.2 `_entityManager`

```
EntityManager& AutoFireSystem::_entityManager [private]
```

Reference to the [EntityManager](#) instance.

This member variable holds a reference to the [EntityManager](#), which is responsible for managing the entities within the ECS ([Entity](#) Component System). It is used to perform operations such as adding, removing, and querying entities.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/auto_fire_system.hpp](#)
- [/home/runner/work/R-Type/R-Type/ECS/Src/Systems/auto_fire_system.cpp](#)

6.12 BackgroundComponent Struct Reference

```
#include <background_component.hpp>
```

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/background_component.hpp](#)

6.13 BasicMonsterComponent Struct Reference

```
#include <basic_monster_component.hpp>
```

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/basic_monster_component.hpp](#)

6.14 BindComponent Struct Reference

A component that binds a function to handle scene transitions.

```
#include <bind_component.hpp>
```

Public Member Functions

- [BindComponent](#) (std::function< [IScenes](#) *([AScenes](#) *, [AScenes::Actions](#))> bindFunction)
Constructs a [BindComponent](#) with the given bind function.

Public Attributes

- bool [isHovered](#) = false
A boolean flag indicating whether the component is currently hovered.
- std::function< [IScenes](#) *([AScenes](#) *, [AScenes::Actions](#))> [bind](#)
A std::function that takes two [AScenes](#) pointers and an [AScenes::Actions](#), and returns a pointer to an [IScenes](#).

6.14.1 Detailed Description

A component that binds a function to handle scene transitions.

This component contains a function that takes two scene pointers and an action, and returns a pointer to a new scene. It also has a flag to indicate if the component is currently hovered.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 BindComponent()

```
BindComponent::BindComponent (
    std::function< IScenes *(AScenes *, AScenes::Actions)> bindFunction ) [inline]
```

Constructs a [BindComponent](#) with the given bind function.

Parameters

<i>bindFunction</i>	The function to bind for handling scene transitions.
---------------------	------------------------------------------------------

6.14.3 Member Data Documentation

6.14.3.1 bind

`BindComponent::bind`

A `std::function` that takes two [AScenes](#) pointers and an [AScenes::Actions](#), and returns a pointer to an [IScenes](#).

6.14.3.2 isHovered

`BindComponent::isHovered = false`

A boolean flag indicating whether the component is currently hovered.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/bind_component.hpp](#)

6.15 BossComponent Struct Reference

```
#include <boss_component.hpp>
```

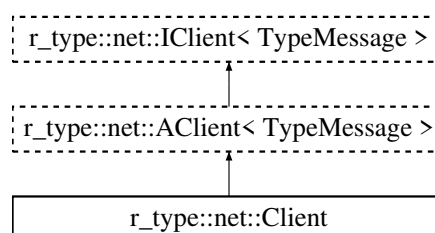
The documentation for this struct was generated from the following file:

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/boss_component.hpp](#)

6.16 r_type::net::Client Class Reference

```
#include <client.hpp>
```

Inheritance diagram for `r_type::net::Client`:



Public Member Functions

- void [PingServer](#) ()
Send a message to the server to get the ping.
- void [MessageAll](#) ()
Send a message to the server to all other clients.
- sf::Vector2u [initInfoBar](#) (UEntityInformation entity, ComponentManager &componentManager, TextureManager &textureManager, FontManager &fontManager, sf::Vector2u windowSize)
- void [updateInfoBar](#) (UEntityInformation entity, ComponentManager &componentManager, TextureManager &textureManager)
- void [addEntity](#) (EntityInformation entity, ComponentManager &componentManager, TextureManager &textureManager, sf::Vector2u windowSize)
- void [removeEntity](#) (int entityId, ComponentManager &componentManager)
- void [moveEntity](#) (uint32_t id, vf2d newPos, ComponentManager &componentManager, sf::Vector2u windowSize)
- void [animateEntity](#) (int entityId, AnimationComponent rect, ComponentManager &componentManager)

Additional Inherited Members

6.16.1 Member Function Documentation

6.16.1.1 addEntity()

```
void r_type::net::Client::addEntity (
    EntityInformation entity,
    ComponentManager & componentManager,
    TextureManager & textureManager,
    sf::Vector2u windowSize ) [inline]
```

6.16.1.2 animateEntity()

```
void r_type::net::Client::animateEntity (
    int entityId,
    AnimationComponent rect,
    ComponentManager & componentManager ) [inline]
```

6.16.1.3 initInfoBar()

```
sf::Vector2u r_type::net::Client::initInfoBar (
    UEntityInformation entity,
    ComponentManager & componentManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    sf::Vector2u windowSize ) [inline]
```

6.16.1.4 MessageAll()

```
void r_type::net::Client::MessageAll ( ) [inline]
```

Send a message to the server to all other clients.

6.16.1.5 moveEntity()

```
void r_type::net::Client::moveEntity (
    uint32_t id,
    vf2d newPos,
    ComponentManager & componentManager,
    sf::Vector2u windowSize ) [inline]
```

6.16.1.6 PingServer()

```
void r_type::net::Client::PingServer ( ) [inline]
```

Send a message to the server to get the ping.

6.16.1.7 removeEntity()

```
void r_type::net::Client::removeEntity (
    int entityId,
    ComponentManager & componentManager ) [inline]
```

6.16.1.8 updateInfoBar()

```
void r_type::net::Client::updateInfoBar (
    UIEntityInformation entity,
    ComponentManager & componentManager,
    TextureManager & textureManager ) [inline]
```

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

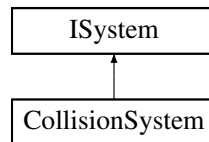
- /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/[client.hpp](#)

6.17 CollisionSystem Class Reference

Manages collision detection and response within the ECS framework.

```
#include <collision_system.hpp>
```

Inheritance diagram for CollisionSystem:



Public Member Functions

- [CollisionSystem](#) ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager)
- bool [checkCollision](#) ([ComponentManager](#) &componentManager, int entityId1, int entityId2)
Checks for a collision between two entities.
- bool [checkOffScreen](#) ([ComponentManager](#) &componentManager, int entityId)
Checks if the entity with the given ID is off the screen.

Private Attributes

- [ComponentManager](#) & [_componentManager](#)
Reference to the [ComponentManager](#) instance.
- [EntityManager](#) & [_entityManager](#)
Reference to the [EntityManager](#) instance.

6.17.1 Detailed Description

Manages collision detection and response within the ECS framework.

This system is responsible for handling all collision-related logic, including detecting collisions between entities and responding to them appropriately.

Parameters

<i>componentManager</i>	Reference to the ComponentManager that handles the components of the entities.
<i>entityManager</i>	Reference to the EntityManager that manages the entities in the system.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 CollisionSystem()

```
CollisionSystem::CollisionSystem (
    ComponentManager & componentManager,
    EntityManager & entityManager ) [inline]
```

6.17.3 Member Function Documentation

6.17.3.1 checkCollision()

```
bool CollisionSystem::checkCollision (
    ComponentManager & componentManager,
    int entityId1,
    int entityId2 )
```

Checks for a collision between two entities.

This function determines whether there is a collision between the components of two specified entities within the component manager.

Parameters

<i>componentManager</i>	Reference to the ComponentManager that holds the components of all entities.
<i>entityId1</i>	The ID of the first entity to check for collision.
<i>entityId2</i>	The ID of the second entity to check for collision.

Returns

true if a collision is detected between the two entities, false otherwise.

6.17.3.2 checkOffScreen()

```
bool CollisionSystem::checkOffScreen (
    ComponentManager & componentManager,
    int entityId )
```

Checks if the entity with the given ID is off the screen.

This function determines whether the specified entity is outside the visible screen area based on its components managed by the [ComponentManager](#).

Parameters

<i>componentManager</i>	Reference to the ComponentManager that manages the entity's components.
<i>entityId</i>	The ID of the entity to check.

Returns

true if the entity is off the screen, false otherwise.

6.17.4 Member Data Documentation**6.17.4.1 `_componentManager`**

`ComponentManager& CollisionSystem::_componentManager [private]`

Reference to the [ComponentManager](#) instance.

This member is used to manage and access various components within the ECS ([Entity](#) Component System).

6.17.4.2 `_entityManager`

`EntityManager& CollisionSystem::_entityManager [private]`

Reference to the [EntityManager](#) instance.

This member variable holds a reference to the [EntityManager](#), which is responsible for managing the entities within the ECS ([Entity](#) Component System). It is used to perform operations such as adding, removing, and querying entities.

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/collision_system.hpp`
- `/home/runner/work/R-Type/R-Type/ECS/Src/Systems/collision_system.cpp`

6.18 ComponentManager Class Reference

Manages the components of entities in an ECS system.

```
#include <component_manager.hpp>
```

Public Member Functions

- `template<typename ComponentType, typename... Args>`
`void addComponent (int entityId, Args &&...args)`
Adds a component to an entity.
- `template<typename ComponentType >`
`std::optional< ComponentType * > getComponent (int entityId)`
Retrieves the component of the specified type associated with the given entity ID.
- `template<typename ComponentType >`
`std::optional< std::unordered_map< int, std::any > * > getComponentMap ()`
Retrieves the component map for the specified component type.
- `template<typename ComponentType >`
`void removeEntityFromComponent (int entityId)`
Removes an entity from the specified component type.
- `void removeAllComponents ()`
Removes all components from the component manager.
- `void removeEntityFromAllComponents (int entityId)`
Removes the specified entity from all components.

Private Attributes

- `std::unordered_map< std::type_index, std::unordered_map< int, std::any > >` [components](#)

A component manager that stores components in an unordered map.

6.18.1 Detailed Description

Manages the components of entities in an ECS system.

The [ComponentManager](#) class provides functionality to add and retrieve components for entities in an ECS system. It uses an unordered map to store the components, where the key is the type of the component and the value is another unordered map that maps entity IDs to their corresponding component values.

6.18.2 Member Function Documentation

6.18.2.1 addComponent()

```
template<typename ComponentType , typename... Args>
void ComponentManager::addComponent (
    int entityId,
    Args &&... args ) [inline]
```

Adds a component to an entity.

Template Parameters

<i>ComponentType</i>	The type of the component to add.
<i>Args</i>	The types of the arguments to forward to the component's constructor.

Parameters

<i>entityId</i>	The ID of the entity to add the component to.
<i>args</i>	The arguments to forward to the component's constructor.

6.18.2.2 GetComponent()

```
template<typename ComponentType >
std::optional<ComponentType *> ComponentManager::GetComponent (
    int entityId ) [inline]
```

Retrieves the component of the specified type associated with the given entity ID.

Template Parameters

<i>ComponentType</i>	The type of the component to retrieve.
----------------------	----------------------------------------

Parameters

<i>entityId</i>	The ID of the entity.
-----------------	-----------------------

Returns

An optional pointer to the component if found, otherwise `std::nullopt`.

6.18.2.3 GetComponentMap()

```
template<typename ComponentType >
std::optional<std::unordered_map<int, std::any>*> ComponentManager::GetComponentMap ( )
[inline]
```

Retrieves the component map for the specified component type.

Template Parameters

<i>ComponentType</i>	The type of the component.
----------------------	----------------------------

Returns

`std::optional<std::unordered_map<int, std::any>*>` The component map if found, otherwise `std::nullopt`.

6.18.2.4 removeAllComponents()

```
void ComponentManager::removeAllComponents ( ) [inline]
```

Removes all components from the component manager.

6.18.2.5 removeEntityFromAllComponents()

```
void ComponentManager::removeEntityFromAllComponents (
    int entityId ) [inline]
```

Removes the specified entity from all components.

This function iterates through all components and removes the entity with the given ID from each component's collection.

Parameters

<i>entityId</i>	The ID of the entity to be removed from all components.
-----------------	---------------------------------------------------------

6.18.2.6 removeEntityFromComponent()

```
template<typename ComponentType >
void ComponentManager::removeEntityFromComponent (
    int entityId ) [inline]
```

Removes an entity from the specified component type.

This function searches for the component type in the components map using the typeid of the ComponentType. If the component type is found, it removes the entity with the given entityId from the component's entity list.

Template Parameters

<i>ComponentType</i>	The type of the component from which the entity should be removed.
----------------------	--------------------------------------------------------------------

Parameters

<i>entityId</i>	The ID of the entity to be removed from the component.
-----------------	--------------------------------------------------------

6.18.3 Member Data Documentation**6.18.3.1 components**

```
std::unordered_map<std::type_index, std::unordered_map<int, std::any> > ComponentManager←
::components [private]
```

A component manager that stores components in an unordered map.

This component manager uses an unordered map to store components. The keys of the outer map are of type `std::type_index`, which represents the type of the component. The values of the outer map are inner unordered maps, where the keys are of type `int` and represent the entity ID, and the values are of type `std::any`, which allows storing components of any type.

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

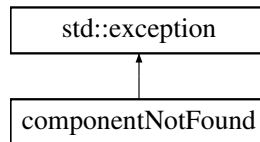
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/component_manager.hpp

6.19 componentNotFound Class Reference

Exception class for when a component is not found.

```
#include <error_handling.hpp>
```

Inheritance diagram for componentNotFound:



Private Member Functions

- `const char * what () const noexcept override`

6.19.1 Detailed Description

Exception class for when a component is not found.

This exception is thrown when a component is not found in the system. It inherits from `std::exception` and overrides the `what\(\)` method to provide a custom error message.

6.19.2 Member Function Documentation

6.19.2.1 `what()`

```
const char* componentNotFound::what ( ) const [inline], [override], [private], [noexcept]
```

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

6.20 EnemyComponent Struct Reference

```
#include <enemy_component.hpp>
```

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_component.hpp`

6.21 EnemyMissileComponent Struct Reference

```
#include <enemy_missile_component.hpp>
```

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_missile_component.hpp](#)

6.22 Entity Class Reference

Represents an entity in the ECS system.

```
#include <entity.hpp>
```

Public Member Functions

- [Entity](#) (int id)
Constructs an [Entity](#) with a specified ID.
- int [getId](#) () const
Retrieves the unique identifier of the entity.

Private Attributes

- int [_id](#)
Unique identifier for the entity.

6.22.1 Detailed Description

Represents an entity in the ECS system.

This class is a concrete implementation of the IEntity interface. It provides functionality to retrieve the ID of the entity.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Entity()

```
Entity::Entity (  
    int id ) [inline], [explicit]
```

Constructs an [Entity](#) with a specified ID.

Parameters

<i>id</i>	The unique identifier for the entity.
-----------	---------------------------------------

6.22.3 Member Function Documentation

6.22.3.1 getId()

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

Retrieves the unique identifier of the entity.

Returns

int The unique identifier of the entity.

6.22.4 Member Data Documentation

6.22.4.1 _id

```
int Entity::_id [private]
```

Unique identifier for the entity.

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

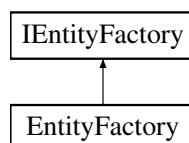
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hpp](#)

6.23 EntityFactory Class Reference

A factory class for creating various types of entities.

```
#include <entity_factory.hpp>
```

Inheritance diagram for EntityFactory:



Public Member Functions

- [Entity createBackgroundLevelOne](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
Create a Background Level One object.
- [Entity createBackgroundLevelTwo](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
Create a Background Level Two object.
- [Entity createBackgroundLevelThree](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
Create a Background Level Three object.
- [Entity createBackgroundMenu](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager) override
Create a Background Menu object.
- [Entity createInfoBar](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
Creates a bar entity.
- [Entity createPlayer](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int nbrOfPlayers) override
Creates a player entity.
- [Entity createShooterEnemy](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int posX, int posY) override
Creates a shooter enemy entity.
- [Entity createBasicMonster](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int posX, int posY) override
Creates a basic monster entity.
- [Entity createPlayerMissile](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId) override
Creates a player missile entity.
- [Entity createForceWeapon](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId) override
Creates a force weapon entity.
- [Entity createForceMissile](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId) override
Creates a force missile entity.
- [Entity createPowerUpBlueLaserCrystal](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
Creates a power-up blue laser crystal entity.
- [Entity createWall](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int posX, int posY) override
Creates a wall entity.
- [Entity createButton](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, std::string text, std::function< [IScenes](#) *([AScenes](#) *)> *onClick, float x=0, float y=0) override
Creates a button entity.
- [Entity createSmallButton](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, std::string text, std::function< [IScenes](#) *([AScenes](#) *, [AScenes::Actions](#))> *onClick, float x=0, float y=0) override
Creates a small button entity.
- [Entity createEnemyMissile](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId) override
Creates an enemy missile entity.
- [Entity createFilter](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [AScenes::DaltonismMode](#) mode) override
Creates a filter entity.
- [Entity backgroundFactory](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, GameState type)

Additional Inherited Members

6.23.1 Detailed Description

A factory class for creating various types of entities.

The [EntityFactory](#) class provides methods to create different types of entities such as background, player, enemies, missiles, buttons, and more. It utilizes the provided entity manager and component manager to create and initialize the entities with the necessary components.

6.23.2 Member Function Documentation

6.23.2.1 backgroundFactory()

```
Entity EntityFactory::backgroundFactory (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    GameState type )
```

6.23.2.2 createBackgroundLevelOne()

```
Entity EntityFactory::createBackgroundLevelOne (
    EntityManager & entityManager,
    ComponentManager & componentManager ) [override], [virtual]
```

Create a Background Level One object.

Parameters

<i>entityManager</i>	
<i>componentManager</i>	

Returns

[Entity](#)

Implements [IEntityFactory](#).

6.23.2.3 createBackgroundLevelThree()

```
Entity EntityFactory::createBackgroundLevelThree (
    EntityManager & entityManager,
    ComponentManager & componentManager ) [override], [virtual]
```

Create a Background Level Three object.

Parameters

<i>entityManager</i>	
<i>componentManager</i>	

Returns

[Entity](#)

Implements [IEntityFactory](#).

6.23.2.4 createBackgroundLevelTwo()

```
Entity EntityFactory::createBackgroundLevelTwo (  
    EntityManager & entityManager,  
    ComponentManager & componentManager ) [override], [virtual]
```

Create a Background Level Two object.

Parameters

<i>entityManager</i>	
<i>componentManager</i>	

Returns

[Entity](#)

Implements [IEntityFactory](#).

6.23.2.5 createBackgroundMenu()

```
Entity EntityFactory::createBackgroundMenu (  
    EntityManager & entityManager,  
    ComponentManager & componentManager,  
    TextureManager & textureManager ) [override], [virtual]
```

Create a Background Menu object.

Parameters

<i>entityManager</i>	
<i>componentManager</i>	

Returns

[Entity](#)Implements [IEntityFactory](#).**6.23.2.6 createBasicMonster()**

```
Entity EntityFactory::createBasicMonster (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int posX,
    int posY ) [override], [virtual]
```

Creates a basic monster entity.

This function creates a basic monster entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager used to create the entity.
<i>componentManager</i>	The component manager used to add components to the entity.
<i>posX</i>	The x-coordinate position of the basic monster.
<i>posY</i>	The y-coordinate position of the basic monster.

Returns

The created basic monster entity.

Implements [IEntityFactory](#).**6.23.2.7 createButton()**

```
Entity EntityFactory::createButton (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    std::string text,
    std::function< IScenes *(AScenes *)> * onClick,
    float x = 0,
    float y = 0 ) [override], [virtual]
```

Creates a button entity.

This function creates a button entity with the specified parameters.

Parameters

<i>entityManager</i>	The entity manager to create the entity.
<i>componentManager</i>	The component manager to add components to the entity.
<i>textureManager</i>	The texture manager to load the button texture.
<i>fontManager</i>	The font manager to load the button font.
<i>text</i>	The text to display on the button.
<i>onClick</i>	The function to be called when the button is clicked.
<i>x</i>	The x-coordinate position of the button.
<i>y</i>	The y-coordinate position of the button.

Returns

The created button entity.

Implements [IEntityFactory](#).

6.23.2.8 createEnemyMissile()

```
Entity EntityFactory::createEnemyMissile (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    uint32_t entityId ) [override], [virtual]
```

Creates an enemy missile entity.

This function creates an enemy missile entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager used to create the entity.
<i>componentManager</i>	The component manager used to add components to the entity.
<i>entityId</i>	The id of the entity that shoots the missile.

Returns

The created enemy missile entity.

Implements [IEntityFactory](#).

6.23.2.9 createFilter()

```
Entity EntityFactory::createFilter (
    EntityManager & entityManager,
```

```
ComponentManager & componentManager,  
AScenes::DaltonismMode mode )
```

Creates a filter entity.

This function creates a filter entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager to use for creating the entity.
<i>componentManager</i>	The component manager to use for adding components to the entity.
<i>mode</i>	The Daltonism mode for the filter.

Returns

The created filter entity.

6.23.2.10 createForceMissile()

```
Entity EntityFactory::createForceMissile (  
    EntityManager & entityManager,  
    ComponentManager & componentManager,  
    uint32_t entityId ) [override], [virtual]
```

Creates a force missile entity.

This function creates a force missile entity with the specified player ID and adds it to the entity manager. It also initializes the necessary components for the force missile entity using the component manager.

Parameters

<i>entityManager</i>	The entity manager to add the force missile entity to.
<i>componentManager</i>	The component manager to initialize the components for the force missile entity.
<i>entityId</i>	The id of the entity that shoots the force missile.

Returns

The created force missile entity.

Implements [IEntityFactory](#).

6.23.2.11 createForceWeapon()

```
Entity EntityFactory::createForceWeapon (  
    EntityManager & entityManager,  
    ComponentManager & componentManager,  
    uint32_t entityId ) [override], [virtual]
```

Creates a force weapon entity.

This function creates a force weapon entity with the specified player ID and adds it to the entity manager. It also initializes the necessary components for the force weapon entity using the component manager.

Parameters

<i>entityManager</i>	The entity manager to add the force weapon entity to.
<i>componentManager</i>	The component manager to initialize the components for the force weapon entity.
<i>entityId</i>	The id of the entity that uses the force weapon.

Returns

The created force weapon entity.

Implements [IEntityFactory](#).

6.23.2.12 createInfoBar()

```
Entity EntityFactory::createInfoBar (  
    EntityManager & entityManager,  
    ComponentManager & componentManager ) [override], [virtual]
```

Creates a bar entity.

This function creates a bar with text for displaying player information like health and score.

Parameters

<i>entityManager</i>	The entity manager to use for creating the entity.
<i>componentManager</i>	The component manager to use for adding components to the entity.

Returns

The created bar entity.

Implements [IEntityFactory](#).

6.23.2.13 createPlayer()

```
Entity EntityFactory::createPlayer (  
    EntityManager & entityManager,  
    ComponentManager & componentManager,  
    int nbrOfPlayers ) [override], [virtual]
```

Creates a player entity.

This function creates a player entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager to use for creating the entity.
<i>componentManager</i>	The component manager to use for adding components to the entity.
<i>nbrOfPlayers</i>	The number of players to create.

Returns

The created player entity.

Implements [IEntityFactory](#).

6.23.2.14 createPlayerMissile()

```
Entity EntityFactory::createPlayerMissile (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    uint32_t entityId ) [override], [virtual]
```

Creates a player missile entity.

This function creates a player missile entity with the specified player ID and adds it to the entity manager. It also initializes the necessary components for the player missile entity using the component manager.

Parameters

<i>entityManager</i>	The entity manager to add the player missile entity to.
<i>componentManager</i>	The component manager to initialize the components for the player missile entity.
<i>entityId</i>	The id of the entity that shoots the missile.

Returns

The created player missile entity.

Implements [IEntityFactory](#).

6.23.2.15 createPowerUpBlueLaserCrystal()

```
Entity EntityFactory::createPowerUpBlueLaserCrystal (
    EntityManager & entityManager,
    ComponentManager & componentManager ) [override], [virtual]
```

Creates a power-up blue laser crystal entity.

This function creates a power-up blue laser crystal entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager to use for creating the entity.
<i>componentManager</i>	The component manager to use for adding components to the entity.

Returns

The created power-up blue laser crystal entity.

Implements [IEntityFactory](#).

6.23.2.16 createShooterEnemy()

```
Entity EntityManager::createShooterEnemy (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int posX,
    int posY ) [override], [virtual]
```

Creates a shooter enemy entity.

This function creates a shooter enemy entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager used to create the entity.
<i>componentManager</i>	The component manager used to add components to the entity.
<i>posX</i>	The x-coordinate position of the shooter enemy.
<i>posY</i>	The y-coordinate position of the shooter enemy.

Returns

The created shooter enemy entity.

Implements [IEntityFactory](#).

6.23.2.17 createSmallButton()

```
Entity EntityManager::createSmallButton (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    std::string text,
    std::function< IScenes *(AScenes *, AScenes::Actions)> * onClick,
```



```
float x = 0,  
float y = 0 ) [override], [virtual]
```

Creates a small button entity.

This function creates a small button entity with the specified parameters.

Parameters

<i>entityManager</i>	The entity manager to create the entity.
<i>componentManager</i>	The component manager to add components to the entity.
<i>textureManager</i>	The texture manager to load the button texture.
<i>fontManager</i>	The font manager to load the button font.
<i>text</i>	The text to display on the button.
<i>onClick</i>	The function to be called when the button is clicked.
<i>x</i>	The x-coordinate position of the button.
<i>y</i>	The y-coordinate position of the button.

Returns

The created small button entity.

Implements [IEntityFactory](#).

6.23.2.18 createWall()

```
Entity IEntityFactory::createWall (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int posX,
    int posY ) [override], [virtual]
```

Creates a wall entity.

This function creates a wall entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager to use for creating the entity.
<i>componentManager</i>	The component manager to use for adding components to the entity.
<i>posX</i>	The x-coordinate position of the wall.
<i>posY</i>	The y-coordinate position of the wall.

Returns

The created wall entity.

Implements [IEntityFactory](#).

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_factory.hpp](#)
- [/home/runner/work/R-Type/R-Type/ECS/Src/Entities/entity_factory.cpp](#)

6.24 EntityInformation Struct Reference

Represents information about an entity.

```
#include <entity_struct.hpp>
```

Public Attributes

- `uint32_t` `uniqueID` = 0
- `vf2d` `ratio` = {0, 0}
- `SpriteDataComponent` `spriteData`
- `vf2d` `vPos` = {0, 0}
- `AnimationComponent` `animationComponent` = {{0, 0}, {0, 0}}

6.24.1 Detailed Description

Represents information about an entity.

6.24.2 Member Data Documentation

6.24.2.1 animationComponent

```
AnimationComponent EntityInformation::animationComponent = {{0, 0}, {0, 0}}
```

6.24.2.2 ratio

```
vf2d EntityInformation::ratio = {0, 0}
```

6.24.2.3 spriteData

```
SpriteDataComponent EntityInformation::spriteData
```

6.24.2.4 uniqueID

```
uint32_t EntityInformation::uniqueID = 0
```

6.24.2.5 vPos

```
vf2d EntityInformation::vPos = {0, 0}
```

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/[entity_struct.hpp](#)

6.25 EntityManager Class Reference

Manages the creation, removal, and retrieval of entities.

```
#include <entity_manager.hpp>
```

Public Member Functions

- [Entity createEntity](#) ()
Creates a new entity and adds it to the entity manager.
- void [removeEntity](#) (int entityId)
Remove an entity from the entity manager.
- void [removeAllEntities](#) ()
Remove all entities from the entity manager.
- std::optional< [Entity](#) * > [getEntity](#) (int entityId)
Get an entity by its ID.
- const std::vector< [Entity](#) > & [getAllEntities](#) () const
Get all entities in the entity manager.

Private Attributes

- int [entityNb](#) = 0
The number of entities in the entity manager.
- std::vector< [Entity](#) > [entities](#)
A container that holds a collection of [Entity](#) objects.

6.25.1 Detailed Description

Manages the creation, removal, and retrieval of entities.

The [EntityManager](#) class is responsible for managing entities within the system. It provides functionality to create new entities, remove existing ones, and retrieve entities by their ID. It also allows access to all entities currently managed by the entity manager.

6.25.2 Member Function Documentation

6.25.2.1 createEntity()

```
Entity EntityManager::createEntity ( ) [inline]
```

Creates a new entity and adds it to the entity manager.

This function increments the entity counter, assigns a new unique ID to the entity, and adds it to the list of managed entities.

Returns

[Entity](#) The newly created entity.

6.25.2.2 getAllEntities()

```
const std::vector<Entity>& EntityManager::getAllEntities ( ) const [inline]
```

Get all entities in the entity manager.

Returns

`const std::vector<Entity>&` A reference to the vector of entities.

This function returns a reference to the vector of entities in the entity manager.

6.25.2.3 getEntity()

```
std::optional<Entity *> EntityManager::getEntity (
    int entityId ) [inline]
```

Get an entity by its ID.

Parameters

<i>entityId</i>	The ID of the entity to retrieve.
-----------------	-----------------------------------

Returns

[Entity](#)& A reference to the entity with the specified ID.

This function retrieves the entity with the specified ID from the entity manager. If the entity is not found, an [entityNotFound](#) exception is thrown.

6.25.2.4 removeAllEntities()

```
void EntityManager::removeAllEntities ( ) [inline]
```

Remove all entities from the entity manager.

This function removes all entities from the entity manager.

6.25.2.5 removeEntity()

```
void EntityManager::removeEntity (
    int entityId ) [inline]
```

Remove an entity from the entity manager.

Parameters

<i>entityId</i>	The ID of the entity to remove.
-----------------	---------------------------------

This function removes the entity with the specified ID from the entity manager. If the entity is not found, an [entityNotFound](#) exception is thrown.

6.25.3 Member Data Documentation

6.25.3.1 entities

```
std::vector<Entity> EntityManager::entities [private]
```

A container that holds a collection of [Entity](#) objects.

This vector is used to manage and store all the entities within the [Entity](#) Component System (ECS). Each [Entity](#) represents a unique object within the ECS framework.

6.25.3.2 entityNb

```
int EntityManager::entityNb = 0 [private]
```

The number of entities in the entity manager.

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

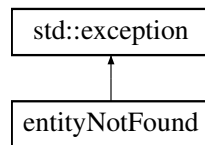
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp](#)

6.26 entityNotFound Class Reference

Exception class for entity not found error.

```
#include <error_handling.hpp>
```

Inheritance diagram for entityNotFound:



Private Member Functions

- `const char * what () const` noexcept override

6.26.1 Detailed Description

Exception class for entity not found error.

This exception is thrown when an entity is not found. It is derived from the `std::exception` class. The `what ()` function is overridden to provide a custom error message.

6.26.2 Member Function Documentation

6.26.2.1 what()

```
const char* entityNotFound::what ( ) const [inline], [override], [private], [noexcept]
```

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

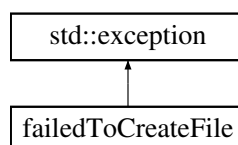
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

6.27 failedToCreateFile Class Reference

Exception class for handling file creation failures.

```
#include <error_handling.hpp>
```

Inheritance diagram for failedToCreateFile:



Private Member Functions

- `const char * what () const noexcept override`

6.27.1 Detailed Description

Exception class for handling file creation failures.

This exception is thrown when a file creation operation fails. It inherits from the standard `std::exception` class.

6.27.2 Member Function Documentation

6.27.2.1 `what()`

```
const char* failedToCreateFile::what ( ) const [inline], [override], [private], [noexcept]
```

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

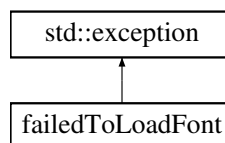
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

6.28 failedToLoadFont Class Reference

Exception class for handling font loading failures.

```
#include <error_handling.hpp>
```

Inheritance diagram for `failedToLoadFont`:



Private Member Functions

- `const char * what () const noexcept override`

6.28.1 Detailed Description

Exception class for handling font loading failures.

This exception is thrown when the application fails to load a font. It inherits from `std::exception` and overrides the `what\(\)` method to provide a specific error message.

6.28.2 Member Function Documentation

6.28.2.1 what()

```
const char* failedToLoadFont::what ( ) const [inline], [override], [private], [noexcept]
```

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

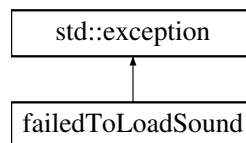
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp](#)

6.29 failedToLoadSound Class Reference

Exception class for handling sound loading failures.

```
#include <error_handling.hpp>
```

Inheritance diagram for failedToLoadSound:



Private Member Functions

- `const char * what () const noexcept override`

6.29.1 Detailed Description

Exception class for handling sound loading failures.

This exception is thrown when the application fails to load a sound file. It inherits from `std::exception` and overrides the [what\(\)](#) method to provide a specific error message.

6.29.2 Member Function Documentation

6.29.2.1 what()

```
const char* failedToLoadSound::what ( ) const [inline], [override], [private], [noexcept]
```

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

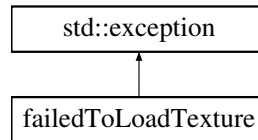
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp](#)

6.30 failedToLoadTexture Class Reference

Exception class for failed texture loading.

```
#include <error_handling.hpp>
```

Inheritance diagram for failedToLoadTexture:



Private Member Functions

- `const char * what () const` noexcept override

6.30.1 Detailed Description

Exception class for failed texture loading.

This exception is thrown when there is a failure to load a texture. It inherits from the `std::exception` class and overrides the `what\(\)` method to provide a custom error message.

6.30.2 Member Function Documentation

6.30.2.1 `what()`

```
const char* failedToLoadTexture::what ( ) const [inline], [override], [private], [noexcept]
```

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

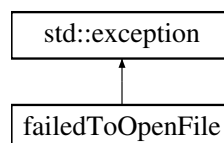
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

6.31 failedToOpenFile Class Reference

Exception class for handling file opening failures.

```
#include <error_handling.hpp>
```

Inheritance diagram for failedToOpenFile:



Private Member Functions

- `const char * what ()` `const noexcept` override

6.31.1 Detailed Description

Exception class for handling file opening failures.

This exception is thrown when a file cannot be opened. It inherits from `std::exception` and overrides the `what\(\)` method to provide a specific error message.

6.31.2 Member Function Documentation

6.31.2.1 `what()`

```
const char* failedToOpenFile::what ( ) const [inline], [override], [private], [noexcept]
```

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

6.32 FontManager Class Reference

Manages the loading and retrieval of font resources.

```
#include <font_manager.hpp>
```

Public Member Functions

- `sf::Font & getFont (const std::string &filePath)`
Retrieves a font from the font manager.
- `void releaseFont (const std::string &filePath)`
Releases the font associated with the given file path.

Private Attributes

- `std::unordered_map< std::string, sf::Font > fonts`
A map that associates font names with their corresponding sf::Font objects.

6.32.1 Detailed Description

Manages the loading and retrieval of font resources.

The [FontManager](#) class provides functionality to load, retrieve, and release font resources. It maintains an internal storage of fonts, allowing for efficient management and reuse of font resources.

Example usage:

```
FontManager fontManager;  
sf::Font &font = fontManager.getFont("path/to/font.ttf");  
// Use the font...  
fontManager.releaseFont("path/to/font.ttf");
```

6.32.2 Member Function Documentation

6.32.2.1 getFont()

```
sf::Font& FontManager::getFont (  
    const std::string & filePath ) [inline]
```

Retrieves a font from the font manager.

This function attempts to find and return a font associated with the given file path. If the font is not already loaded, it will attempt to load it from the specified file path. If loading the font fails, an exception is thrown.

Parameters

<i>filePath</i>	The path to the font file.
-----------------	----------------------------

Returns

A reference to the loaded `sf::Font` object.

Exceptions

failedToLoadFont	if the font cannot be loaded from the specified file path.
----------------------------------	------------------------------------------------------------

6.32.2.2 releaseFont()

```
void FontManager::releaseFont (  
    const std::string & filePath ) [inline]
```

Releases the font associated with the given file path.

This function removes the font from the internal storage, effectively releasing any resources associated with it.

Parameters

<i>filePath</i>	The file path of the font to be released.
-----------------	-------------------------------------------

6.32.3 Member Data Documentation

6.32.3.1 fonts

```
std::unordered_map<std::string, sf::Font> FontManager::fonts [private]
```

A map that associates font names with their corresponding sf::Font objects.

This unordered map uses strings as keys to store and retrieve sf::Font objects. It allows for efficient lookup, insertion, and deletion of font resources by name.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_manager.hpp](#)

6.33 ForceMissileComponent Struct Reference

Component representing a force missile in the ECS system.

```
#include <force_missile_component.hpp>
```

Public Attributes

- uint32_t [forceId](#)
Unique identifier for the force missile.

6.33.1 Detailed Description

Component representing a force missile in the ECS system.

This component is used to identify and manage force missiles within the Entity-Component-System (ECS) architecture.

6.33.2 Member Data Documentation

6.33.2.1 forceId

`ForceMissileComponent::forceId`

Unique identifier for the force missile.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_missile_component.hpp

6.34 ForceWeaponComponent Struct Reference

Represents a component for a force weapon in the game.

```
#include <force_weapon_component.hpp>
```

Public Member Functions

- [ForceWeaponComponent](#) (uint32_t _playerId, uint32_t _level, uint32_t _attached)
Constructs a new [ForceWeaponComponent](#).

Public Attributes

- uint32_t [playerId](#)
The ID of the player who owns the force weapon.
- uint32_t [level](#)
The level of the force weapon.
- bool [attached](#)
A flag indicating whether the force weapon is attached to the player.

6.34.1 Detailed Description

Represents a component for a force weapon in the game.

This component is used to manage the state and properties of a force weapon associated with a player.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 ForceWeaponComponent()

```
ForceWeaponComponent::ForceWeaponComponent (
    uint32_t _playerId,
    uint32_t _level,
    uint32_t _attached ) [inline]
```

Constructs a new [ForceWeaponComponent](#).

Parameters

<code>_playerId</code>	The ID of the player who owns the force weapon.
<code>_level</code>	The level of the force weapon.
<code>_attached</code>	A flag indicating whether the force weapon is attached to the player.

6.34.3 Member Data Documentation

6.34.3.1 attached

`ForceWeaponComponent::attached`

A flag indicating whether the force weapon is attached to the player.

6.34.3.2 level

`ForceWeaponComponent::level`

The level of the force weapon.

6.34.3.3 playerId

`ForceWeaponComponent::playerId`

The ID of the player who owns the force weapon.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_weapon_component.hpp](#)

6.35 FrontComponent Struct Reference

A component that represents the front of an entity.

```
#include <front_component.hpp>
```

Public Member Functions

- [FrontComponent](#) (int _targetId)

Public Attributes

- int [targetId](#)

6.35.1 Detailed Description

A component that represents the front of an entity.

This component is used to identify the target entity that this component is associated with.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 FrontComponent()

```
FrontComponent::FrontComponent (  
    int _targetId ) [inline]
```

6.35.3 Member Data Documentation

6.35.3.1 targetId

```
int FrontComponent::targetId
```

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/front_component.hpp

6.36 HealthComponent Struct Reference

Represents the health attributes of an entity.

```
#include <health_component.hpp>
```

Public Attributes

- int [max_health](#)
Maximum health value that the entity can have.
- int [health](#)
Current health value of the entity.

6.36.1 Detailed Description

Represents the health attributes of an entity.

This component is used to store and manage the health-related data of an entity.

6.36.2 Member Data Documentation

6.36.2.1 health

```
HealthComponent::health
```

Current health value of the entity.

6.36.2.2 max_health

```
HealthComponent::max_health
```

Maximum health value that the entity can have.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/health_component.hpp](#)

6.37 HitboxComponent Struct Reference

Represents the hitbox dimensions of an entity.

```
#include <hitbox_component.hpp>
```

Public Attributes

- int [w](#)
Width of the hitbox.
- int [h](#)
Height of the hitbox.

6.37.1 Detailed Description

Represents the hitbox dimensions of an entity.

This component is used to define the width and height of an entity's hitbox in the game. It is typically used for collision detection purposes.

6.37.2 Member Data Documentation

6.37.2.1 h

HitboxComponent::h

Height of the hitbox.

6.37.2.2 w

HitboxComponent::w

Width of the hitbox.

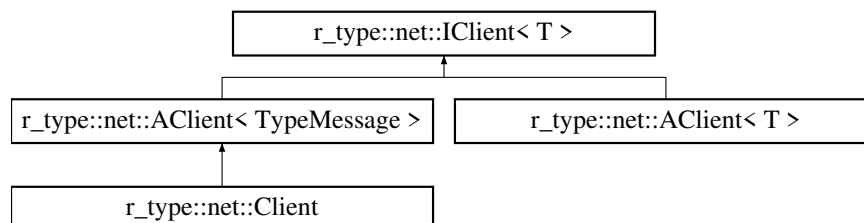
The documentation for this struct was generated from the following file:

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/hitbox_component.hpp

6.38 r_type::net::IClient< T > Class Template Reference

```
#include <i_client.hpp>
```

Inheritance diagram for r_type::net::IClient< T >:



Public Member Functions

- [IClient](#) ()
- virtual [~IClient](#) ()
- virtual bool [Connect](#) (const std::string &host, const uint16_t port)=0
Connects to a remote host using UDP protocol.
- virtual void [Disconnect](#) ()=0
Disconnects the client from the server.
- virtual bool [IsConnected](#) ()=0
Checks if the client is connected to the server.
- virtual void [Send](#) (const Message< T > &msg)=0
Send message to server.
- virtual ThreadSafeQueue< OwnedMessage< T > > & [Incoming](#) ()=0
get incoming messages

6.38.1 Constructor & Destructor Documentation

6.38.1.1 `IClient()`

```
template<typename T >
r_type::net::IClient< T >::IClient ( ) [inline]
```

6.38.1.2 `~IClient()`

```
template<typename T >
virtual r_type::net::IClient< T >::~~IClient ( ) [inline], [virtual]
```

6.38.2 Member Function Documentation

6.38.2.1 `Connect()`

```
template<typename T >
virtual bool r_type::net::IClient< T >::Connect (
    const std::string & host,
    const uint16_t port ) [pure virtual]
```

Connects to a remote host using UDP protocol.

Parameters

<i>host</i>	The IP address or hostname of the remote host.
<i>port</i>	The port number of the remote host.

Returns

true if the connection is successful
false otherwise.

Implemented in `r_type::net::AClient< T >`, and `r_type::net::AClient< TypeMessage >`.

6.38.2.2 `Disconnect()`

```
template<typename T >
virtual void r_type::net::IClient< T >::Disconnect ( ) [pure virtual]
```

Disconnects the client from the server.

This function disconnects the client from the server if it is currently connected. It stops the context and joins the context thread. It also releases the connection resource.

Implemented in [r_type::net::AClient< T >](#), and [r_type::net::AClient< TypeMessage >](#).

6.38.2.3 Incoming()

```
template<typename T >
virtual ThreadSafeQueue<OwnedMessage<T> >& r_type::net::IClient< T >::Incoming ( ) [pure
virtual]
```

get incoming messages

Returns

ThreadSafeQueue<OwnedMessage<T>>&

Implemented in [r_type::net::AClient< T >](#), and [r_type::net::AClient< TypeMessage >](#).

6.38.2.4 IsConnected()

```
template<typename T >
virtual bool r_type::net::IClient< T >::IsConnected ( ) [pure virtual]
```

Checks if the client is connected to the server.

Returns

true

false

Implemented in [r_type::net::AClient< T >](#), and [r_type::net::AClient< TypeMessage >](#).

6.38.2.5 Send()

```
template<typename T >
virtual void r_type::net::IClient< T >::Send (
    const Message< T > & msg ) [pure virtual]
```

Send message to server.

Parameters

<i>msg</i>	
------------	--

Implemented in [r_type::net::AClient< T >](#).

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

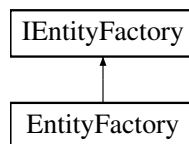
- [/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp](#)

6.39 IEntityFactory Class Reference

The interface for an entity factory.

```
#include <i_entity_factory.hpp>
```

Inheritance diagram for IEntityFactory:



Public Types

- enum [EnemyType](#) { [BasicMonster](#) , [ShooterEnemy](#) , [Wall](#) , [Boss](#) }
Enumeration representing different types of enemies in the game.

Public Member Functions

- virtual [~IEntityFactory](#) ()=default
Destroy the IEntityFactory object.
- virtual [Entity](#) [createBackgroundLevelOne](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager)=0
Creates a background entity.
- virtual [Entity](#) [createBackgroundLevelTwo](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager)=0
Create a Background Level Two object.
- virtual [Entity](#) [createBackgroundLevelThree](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager)=0
Create a Background Level Three object.
- virtual [Entity](#) [createBackgroundMenu](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager)=0
Create a Background Menu object.
- virtual [Entity](#) [createInfoBar](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager)=0
Creates a bar entity.

- virtual [Entity](#) [createPlayer](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int nbrOfPlayers)=0
Creates a player entity.
- virtual [Entity](#) [createShooterEnemy](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int posX, int posY)=0
Creates a shooter enemy entity.
- virtual [Entity](#) [createBasicMonster](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int posX, int posY)=0
Creates a basic monster entity.
- virtual [Entity](#) [createPlayerMissile](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId)=0
Creates a player missile entity.
- virtual [Entity](#) [createForceWeapon](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId)=0
Creates a Force Weapon entity.
- virtual [Entity](#) [createForceMissile](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId)=0
Creates a Force Missile entity.
- virtual [Entity](#) [createPowerUpBlueLaserCrystal](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager)=0
Creates a Power-Up Blue Laser Crystal entity.
- virtual [Entity](#) [createWall](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int posX, int posY)=0
Creates a wall entity with the specified position.
- virtual [Entity](#) [createEnemyMissile](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId)=0
Creates an enemy missile entity.
- virtual [Entity](#) [createButton](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, std::string text, std::function< [IScenes](#) *([AScenes](#) *)> *onClick, float x, float y)=0
Creates a button entity.
- virtual [Entity](#) [createSmallButton](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, std::string text, std::function< [IScenes](#) *([AScenes](#) *, [AScenes::Actions](#))> *onClick, float x=0, float y=0)=0
Creates a button entity with the specified properties.

6.39.1 Detailed Description

The interface for an entity factory.

This interface defines the methods for creating different types of entities in the game. Each method takes references to the entity manager, component manager, and other necessary parameters, and returns an entity object.

Note

This is an abstract base class and cannot be instantiated directly.

6.39.2 Member Enumeration Documentation

6.39.2.1 EnemyType

```
enum IEntityFactory::EnemyType
```

Enumeration representing different types of enemies in the game.

This enumeration defines the various enemy types that can be instantiated in the game. Each type corresponds to a specific kind of enemy with unique behaviors and characteristics.

Enumerator

BasicMonster	
ShooterEnemy	
Wall	
Boss	

6.39.3 Constructor & Destructor Documentation

6.39.3.1 ~IEntityFactory()

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

Destroy the [IEntityFactory](#) object.

6.39.4 Member Function Documentation

6.39.4.1 createBackgroundLevelOne()

```
virtual Entity IEntityFactory::createBackgroundLevelOne (
    EntityManager & entityManager,
    ComponentManager & componentManager ) [pure virtual]
```

Creates a background entity.

This function creates a background entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager to use for creating the entity.
<i>componentManager</i>	The component manager to use for adding components to the entity.

Returns

The created background entity.

Implemented in [EntityFactory](#).

6.39.4.2 createBackgroundLevelThree()

```
virtual Entity IEntityFactory::createBackgroundLevelThree (  
    EntityManager & entityManager,  
    ComponentManager & componentManager ) [pure virtual]
```

Create a Background Level Three object.

Parameters

<i>entityManager</i>	
<i>componentManager</i>	

Returns

[Entity](#)

Implemented in [EntityFactory](#).

6.39.4.3 createBackgroundLevelTwo()

```
virtual Entity IEntityFactory::createBackgroundLevelTwo (  
    EntityManager & entityManager,  
    ComponentManager & componentManager ) [pure virtual]
```

Create a Background Level Two object.

Parameters

<i>entityManager</i>	
<i>componentManager</i>	

Returns

[Entity](#)

Implemented in [EntityFactory](#).

6.39.4.4 createBackgroundMenu()

```
virtual Entity IEntityFactory::createBackgroundMenu (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    TextureManager & textureManager ) [pure virtual]
```

Create a Background Menu object.

Parameters

<i>entityManager</i>	
<i>componentManager</i>	

Returns

Entity

Implemented in [EntityFactory](#).

6.39.4.5 createBasicMonster()

```
virtual Entity IEntityFactory::createBasicMonster (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int posX,
    int posY ) [pure virtual]
```

Creates a basic monster entity.

This function creates a basic monster entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager used to create the entity.
<i>componentManager</i>	The component manager used to add components to the entity.

Returns

The created basic monster entity.

Implemented in [EntityFactory](#).

6.39.4.6 createButton()

```
virtual Entity IEntityFactory::createButton (
    EntityManager & entityManager,
```

```

    ComponentManager & componentManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    std::string text,
    std::function< IScenes *(AScenes *)> * onClick,
    float x,
    float y ) [pure virtual]

```

Creates a button entity.

This function creates a button entity using the provided entity manager, component manager, texture manager, text, and onClick function. The button entity represents a clickable button in the game.

Parameters

<i>entityManager</i>	The entity manager used to create the button entity.
<i>componentManager</i>	The component manager used to manage the components of the button entity.
<i>textureManager</i>	The texture manager used to load the textures for the button entity.
<i>text</i>	The text displayed on the button.
<i>onClick</i>	The function to be called when the button is clicked.

Returns

The created button entity.

Implemented in [EntityFactory](#).

6.39.4.7 createEnemyMissile()

```

virtual Entity IEntityFactory::createEnemyMissile (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    uint32_t entityId ) [pure virtual]

```

Creates an enemy missile entity.

This function creates an enemy missile entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager used to create the entity.
<i>componentManager</i>	The component manager used to add components to the entity.

Returns

The created enemy missile entity.

Implemented in [EntityFactory](#).

6.39.4.8 createForceMissile()

```
virtual Entity IEntityFactory::createForceMissile (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    uint32_t entityId ) [pure virtual]
```

Creates a Force Missile entity.

This function creates a Force Missile entity and registers it with the given [EntityManager](#) and [ComponentManager](#). The entity is identified by the provided entityId.

Parameters

<i>entityManager</i>	Reference to the EntityManager that will manage the entity.
<i>componentManager</i>	Reference to the ComponentManager that will manage the components of the entity.
<i>entityId</i>	The unique identifier for the entity to be created.

Returns

[Entity](#) The created Force Missile entity.

Implemented in [EntityFactory](#).

6.39.4.9 createForceWeapon()

```
virtual Entity IEntityFactory::createForceWeapon (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    uint32_t entityId ) [pure virtual]
```

Creates a Force Weapon entity.

This function is responsible for creating a Force Weapon entity and adding it to the provided [EntityManager](#) and [ComponentManager](#). The entity is identified by the given entityId.

Parameters

<i>entityManager</i>	Reference to the EntityManager that will manage the entity.
<i>componentManager</i>	Reference to the ComponentManager that will manage the components of the entity.
<i>entityId</i>	The unique identifier for the entity to be created.

Returns

[Entity](#) The created Force Weapon entity.

Implemented in [EntityFactory](#).

6.39.4.10 createInfoBar()

```
virtual Entity IEntityFactory::createInfoBar (
    EntityManager & entityManager,
    ComponentManager & componentManager ) [pure virtual]
```

Creates a bar entity.

This function creates a bar with text for displaying player information like health and score.

Parameters

<i>entityManager</i>	The entity manager to use for creating the entity.
<i>componentManager</i>	The component manager to use for adding components to the entity.

Returns

The created bar entity.

Implemented in [EntityFactory](#).

6.39.4.11 createPlayer()

```
virtual Entity IEntityFactory::createPlayer (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int nbrOfPlayers ) [pure virtual]
```

Creates a player entity.

This function creates a player entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager used to create the entity.
<i>componentManager</i>	The component manager used to add components to the entity.

Returns

The created player entity.

Implemented in [EntityFactory](#).

6.39.4.12 createPlayerMissile()

```
virtual Entity IEntityFactory::createPlayerMissile (
    EntityManager & entityManager,
```

```
ComponentManager & componentManager,  
uint32_t entityId ) [pure virtual]
```

Creates a player missile entity.

This function creates a player missile entity with the specified player ID and adds it to the entity manager. It also initializes the necessary components for the player missile entity using the component manager.

Parameters

<i>entityId</i>	The ID of the entity that shoot the missile.
<i>entityManager</i>	The entity manager to add the player missile entity to.
<i>componentManager</i>	The component manager to initialize the components for the player missile entity.

Returns

The created player missile entity.

Implemented in [EntityFactory](#).

6.39.4.13 createPowerUpBlueLaserCrystal()

```
virtual Entity IEntityFactory::createPowerUpBlueLaserCrystal (  
    EntityManager & entityManager,  
    ComponentManager & componentManager ) [pure virtual]
```

Creates a Power-Up Blue Laser Crystal entity.

This function is responsible for creating an entity that represents a Power-Up Blue Laser Crystal in the game. It initializes the entity with the necessary components and registers it with the provided [EntityManager](#) and [ComponentManager](#).

Parameters

<i>entityManager</i>	Reference to the EntityManager that will manage the entity.
<i>componentManager</i>	Reference to the ComponentManager that will manage the components of the entity.

Returns

[Entity](#) The created Power-Up Blue Laser Crystal entity.

Implemented in [EntityFactory](#).

6.39.4.14 createShooterEnemy()

```
virtual Entity IEntityFactory::createShooterEnemy (  
    EntityManager & entityManager,
```

```

    ComponentManager & componentManager,
    int posX,
    int posY ) [pure virtual]

```

Creates a shooter enemy entity.

This function creates a shooter enemy entity using the provided entity manager and component manager.

Parameters

<i>entityManager</i>	The entity manager used to create the entity.
<i>componentManager</i>	The component manager used to add components to the entity.

Returns

The created shooter enemy entity.

Implemented in [EntityFactory](#).

6.39.4.15 createSmallButton()

```

virtual Entity IEntityFactory::createSmallButton (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    std::string text,
    std::function< IScenes *(AScenes *, AScenes::Actions)> * onClick,
    float x = 0,
    float y = 0 ) [pure virtual]

```

Creates a button entity with the specified properties.

Parameters

<i>entityManager</i>	Reference to the EntityManager responsible for managing entities.
<i>componentManager</i>	Reference to the ComponentManager responsible for managing components.
<i>textureManager</i>	Reference to the TextureManager responsible for managing textures.
<i>fontManager</i>	Reference to the FontManager responsible for managing fonts.
<i>text</i>	The text to be displayed on the button.
<i>onClick</i>	A pointer to a function that will be called when the button is clicked.
<i>x</i>	The x-coordinate position of the button.
<i>y</i>	The y-coordinate position of the button.

Returns

[Entity](#) The created button entity.

Implemented in [EntityFactory](#).

6.39.4.16 createWall()

```
virtual Entity IEntityFactory::createWall (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int posX,
    int posY ) [pure virtual]
```

Creates a wall entity with the specified position.

Parameters

<i>entityManager</i>	Reference to the EntityManager that will manage the new entity.
<i>componentManager</i>	Reference to the ComponentManager that will manage the components of the new entity.
<i>posX</i>	The x-coordinate of the wall's position.
<i>posY</i>	The y-coordinate of the wall's position.

Returns

[Entity](#) The created wall entity.

Implemented in [EntityFactory](#).

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/i_entity_factory.hpp](#)

6.40 InputComponent Struct Reference

Component for handling input actions.

```
#include <input_component.hpp>
```

Public Attributes

- [InputType](#) *input*
The current input action of the entity.

6.40.1 Detailed Description

Component for handling input actions.

This structure is used to store the current input action of an entity.

6.40.2 Member Data Documentation

6.40.2.1 input

`InputComponent::input`

The current input action of the entity.

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

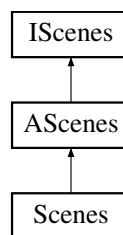
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/input_component.hpp

6.41 IScenes Class Reference

Interface for managing different scenes in a game.

```
#include <i_scenes.hpp>
```

Inheritance diagram for IScenes:



Public Member Functions

- virtual `~IScenes()`=default
- virtual void `mainMenu()`=0
Displays the main menu and creates necessary entities.
- virtual void `gameLoop()`=0
Displays the main game loop and creates necessary entities.
- virtual void `settingsMenu()`=0
Displays the settings menu and creates necessary entities.
- virtual void `inGameMenu()`=0
Displays the in-game menu and creates necessary entities.
- virtual void `difficultyChoices()`=0
Displays the difficulty choices.
- virtual void `render()`=0
Displays the current scene and manages its components.
- virtual bool `shouldQuit()`=0
Checks if the game should quit.
- virtual `sf::RenderWindow * getRenderWindow()`=0
Gets the render window.

6.41.1 Detailed Description

Interface for managing different scenes in a game.

This interface declares the methods for displaying and managing various scenes in a game, such as the main menu, game loop, settings menu, and in-game menu.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 ~IScenes()

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

6.41.3 Member Function Documentation

6.41.3.1 difficultyChoices()

```
virtual void IScenes::difficultyChoices ( ) [pure virtual]
```

Displays the difficulty choices.

Implemented in [Scenes](#).

6.41.3.2 gameLoop()

```
virtual void IScenes::gameLoop ( ) [pure virtual]
```

Displays the main game loop and creates necessary entities.

Implemented in [Scenes](#).

6.41.3.3 getRenderWindow()

```
virtual sf::RenderWindow* IScenes::getRenderWindow ( ) [pure virtual]
```

Gets the render window.

Returns

Pointer to the sf::RenderWindow.

Implemented in [Scenes](#).

6.41.3.4 inGameMenu()

```
virtual void IScenes::inGameMenu ( ) [pure virtual]
```

Displays the in-game menu and creates necessary entities.

Implemented in [Scenes](#).

6.41.3.5 mainMenu()

```
virtual void IScenes::mainMenu ( ) [pure virtual]
```

Displays the main menu and creates necessary entities.

Implemented in [Scenes](#).

6.41.3.6 render()

```
virtual void IScenes::render ( ) [pure virtual]
```

Displays the current scene and manages its components.

Implemented in [Scenes](#).

6.41.3.7 settingsMenu()

```
virtual void IScenes::settingsMenu ( ) [pure virtual]
```

Displays the settings menu and creates necessary entities.

Implemented in [Scenes](#).

6.41.3.8 shouldQuit()

```
virtual bool IScenes::shouldQuit ( ) [pure virtual]
```

Checks if the game should quit.

Returns

True if the game should quit, false otherwise.

Implemented in [Scenes](#).

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

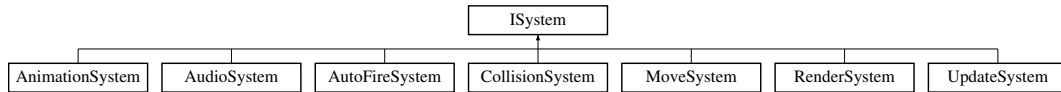
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/i_scenes.hpp](#)

6.42 ISystem Class Reference

Interface for all systems in the ECS ([Entity](#) Component System) architecture.

```
#include <i_system.hpp>
```

Inheritance diagram for ISystem:



Public Member Functions

- [ISystem](#) ()=default
- virtual [~ISystem](#) ()=default

6.42.1 Detailed Description

Interface for all systems in the ECS ([Entity](#) Component System) architecture.

This class serves as a base class for all systems within the ECS framework. Systems are responsible for processing entities that possess a specific set of components.

Note

This is an abstract class and should not be instantiated directly.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 ISystem()

```
ISystem::ISystem ( ) [default]
```

6.42.2.2 ~ISystem()

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

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/i_system.hpp](#)

6.43 labelComponent Struct Reference

Represents a label component with a name and position coordinates.

```
#include <label_component.hpp>
```

Public Attributes

- `std::string` [name](#)
The name of the label.
- `int` [x](#)
The x-coordinate of the label's position.
- `int` [y](#)
The y-coordinate of the label's position.

6.43.1 Detailed Description

Represents a label component with a name and position coordinates.

This structure is used to define a label component in the ECS ([Entity](#) Component System). It contains a name and the x and y coordinates for positioning.

6.43.2 Member Data Documentation

6.43.2.1 name

```
labelComponent::name
```

The name of the label.

6.43.2.2 x

```
labelComponent::x
```

The x-coordinate of the label's position.

6.43.2.3 y

```
labelComponent::y
```

The y-coordinate of the label's position.

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

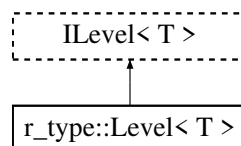
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp

6.44 r_type::Level< T > Class Template Reference

The [Level](#) class template manages the game level, including updating game state, handling collisions, and managing entities.

```
#include <level.hpp>
```

Inheritance diagram for r_type::Level< T >:



Public Member Functions

- [Level](#) ()=default
- [~Level](#) ()=default
- void [Update](#) (r_type::net::AServer< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock, bool *bUpdateEntities) override
Updates the game state by processing entity movements, handling collisions, and sending messages to clients.
- void [SetSystem](#) ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager) override
Initializes and sets up various systems for the level.
- void [MoveUpdate](#) (r_type::net::AServer< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock) override
Updates the positions of entities and notifies clients of any changes.
- void [CollisionUpdate](#) (r_type::net::AServer< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock) override
Updates the collision status of entities in the game.
- void [AnimationUpdate](#) (r_type::net::AServer< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock) override
Updates the animations of entities and sends messages to clients if animations have changed.
- void [FireUpdate](#) (r_type::net::AServer< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock) override
Updates the firing mechanism of entities in the game.
- void [LevelOne](#) (r_type::net::AServer< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock) override
Handles the spawning of entities for [Level](#) One.

- void [LevelTwo](#) ([r_type::net::AServer](#)< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock) override
Handles the spawning of entities for [Level Two](#).
- void [LevelThree](#) ([r_type::net::AServer](#)< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::chrono::system_clock::time_point newClock) override
Handles the spawning of entities for [Level Three](#).
- void [SpawnEntity](#) ([r_type::net::AServer](#)< T > *server, [EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, int nbrOfEnemy, [EntityFactory::EnemyType](#) enemyType)
Spawns a specified number of enemy entities in the game.
- [EntityInformation](#) [GetEntityBackGround](#) ([r_type::net::AServer](#)< T > *server, [EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
Get the [Entity](#) Back Ground object.
- [EntityInformation](#) [InitiateBackground](#) ([r_type::net::AServer](#)< T > *server, [EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
Initializes a background entity.
- void [SetGameParameters](#) (GameParameters gameParameters)
Sets the game difficulty based on the provided game parameters.

Static Public Member Functions

- static bool [collisionAction](#) ([r_type::net::AServer](#)< T > *server, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, std::vector< int > &entitiesToRemove, std::vector< int > &entitiesToAdd, uint32_t entityId1, uint32_t entityId2)
Handles the collision action between two entities in the game.

Protected Attributes

- std::shared_ptr< [MoveSystem](#) > [_moveSystem](#)
A shared pointer to the [MoveSystem](#) instance.
- std::shared_ptr< [CollisionSystem](#) > [_collisionSystem](#)
A shared pointer to the [CollisionSystem](#).
- std::shared_ptr< [AnimationSystem](#) > [_animationSystem](#)
A shared pointer to the [AnimationSystem](#).
- std::shared_ptr< [AutoFireSystem](#) > [_autoFireSystem](#)
A shared pointer to an instance of [AutoFireSystem](#).
- std::chrono::system_clock::time_point [_basicMonsterSpawnTime](#)
Represents the time point at which a basic monster is spawned.
- std::chrono::system_clock::time_point [_shooterEnemySpawnTime](#)
Represents the time point when the shooter enemy is spawned.
- std::chrono::system_clock::time_point [_WallSpawnTime](#) = std::chrono::system_clock::now()
Stores the time point when the wall was spawned.
- std::chrono::system_clock::time_point [_spawnTimeMonsterThree](#)
The time point at which the third type of monster is spawned.
- GameParameters [_gameParameters](#)
Holds the parameters for the game configuration.

6.44.1 Detailed Description

```
template<typename T>
class r_type::Level< T >
```

The `Level` class template manages the game level, including updating game state, handling collisions, and managing entities.

This class template provides methods to update the game state, handle collisions, manage animations, and control the firing mechanisms of entities. It also includes functionality to spawn entities and set game parameters.

Template Parameters

<code>T</code>	The type of the server.
----------------	-------------------------

6.44.2 Constructor & Destructor Documentation

6.44.2.1 `Level()`

```
template<typename T >
r_type::Level< T >::Level ( ) [default]
```

6.44.2.2 `~Level()`

```
template<typename T >
r_type::Level< T >::~~Level ( ) [default]
```

6.44.3 Member Function Documentation

6.44.3.1 `AnimationUpdate()`

```
template<typename T >
void r_type::Level< T >::AnimationUpdate (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::chrono::system_clock::time_point newClock ) [inline], [override]
```

Updates the animations of entities and sends messages to clients if animations have changed.

This function performs the following steps:

1. Retrieves the current animation components from the component manager.
2. Saves the current state of animations.
3. Updates the animations using the animation system.
4. Compares the new state of animations with the previous state.
5. Sends messages to all clients if any animations have changed.

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the component manager.
<i>entityManager</i>	Reference to the entity manager.
<i>newClock</i>	The current time point.

6.44.3.2 `collisionAction()`

```
template<typename T >
static bool r_type::Level< T >::collisionAction (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::vector< int > & entitiesToRemove,
    std::vector< int > & entitiesToAdd,
    uint32_t entityId1,
    uint32_t entityId2 ) [inline], [static]
```

Handles the collision action between two entities in the game.

This function determines the type of collision between two entities and performs the appropriate actions based on the components of the entities involved. It updates the health, score, and other relevant components, and manages the addition and removal of entities from the game.

Template Parameters

<i>T</i>	The type of the server.
----------	-------------------------

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the ComponentManager .
<i>entityManager</i>	Reference to the EntityManager .
<i>entitiesToRemove</i>	Vector of entity IDs to be removed from the game.
<i>entitiesToAdd</i>	Vector of entity IDs to be added to the game.
<i>entityId1</i>	The ID of the first entity involved in the collision.
<i>entityId2</i>	The ID of the second entity involved in the collision.

Returns

True if a collision was handled, false otherwise.

6.44.3.3 `CollisionUpdate()`

```
template<typename T >
void r_type::Level< T >::CollisionUpdate (
```

```

r_type::net::AServer< T > * server,
ComponentManager & componentManager,
EntityManager & entityManager,
std::chrono::system_clock::time_point newClock ) [inline], [override]

```

Updates the collision status of entities in the game.

This function checks for collisions between entities and handles the consequences of those collisions, such as updating health, removing entities, and adding new entities. It also handles entities that go off-screen.

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the component manager.
<i>entityManager</i>	Reference to the entity manager.
<i>newClock</i>	The current time point for the update.

6.44.3.4 FireUpdate()

```

template<typename T >
void r_type::Level< T >::FireUpdate (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::chrono::system_clock::time_point newClock ) [inline], [override]

```

Updates the firing mechanism of entities in the game.

This function handles the automatic firing system and processes the firing logic for entities. It retrieves all entities and checks if they can shoot. If an entity can shoot, it sends a message to all clients to create an enemy missile and sets the entity's canShoot flag to false.

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the ComponentManager handling components.
<i>entityManager</i>	Reference to the EntityManager handling entities.
<i>newClock</i>	The current time point used for timing events.

6.44.3.5 GetEntityBackGround()

```

template<typename T >
EntityInformation r_type::Level< T >::GetEntityBackGround (
    r_type::net::AServer< T > * server,
    EntityManager & entityManager,
    ComponentManager & componentManager ) [inline], [override]

```

Get the [Entity](#) Back Ground object.

Parameters

<code>server</code>	
<code>entityManager</code>	
<code>componentManager</code>	

Returns

[EntityInformation](#)

6.44.3.6 `InitiateBackground()`

```
template<typename T >
EntityInformation r_type::Level< T >::InitiateBackground (
    r_type::net::AServer< T > * server,
    EntityManager & entityManager,
    ComponentManager & componentManager ) [inline], [override]
```

Initializes a background entity.

The function creates and returns information about the background entity.

Returns

[EntityInformation](#) The information of the background entity.

6.44.3.7 `LevelOne()`

```
template<typename T >
void r_type::Level< T >::LevelOne (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::chrono::system_clock::time_point newClock ) [inline], [override]
```

Handles the spawning of entities for [Level One](#).

This function is responsible for spawning basic monsters and shooter enemies at specific intervals defined by the game parameters. It checks the elapsed time since the last spawn of each entity type and spawns new entities if the required time has passed.

Parameters

<code>server</code>	Pointer to the server instance.
<code>componentManager</code>	Reference to the ComponentManager instance.
<code>entityManager</code>	Reference to the EntityManager instance.
<code>newClock</code>	The current time point used for timing calculations.

6.44.3.8 LevelThree()

```
template<typename T >
void r_type::Level< T >::LevelThree (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::chrono::system_clock::time_point newClock ) [inline], [override]
```

Handles the spawning of entities for [Level](#) Three.

This function is responsible for spawning basic monsters and shooter enemies at specific intervals defined by the game parameters. It checks the elapsed time since the last spawn of each entity type and spawns new entities if the required time has passed.

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the ComponentManager instance.
<i>entityManager</i>	Reference to the EntityManager instance.
<i>newClock</i>	The current time point used for timing calculations.

6.44.3.9 LevelTwo()

```
template<typename T >
void r_type::Level< T >::LevelTwo (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::chrono::system_clock::time_point newClock ) [inline], [override]
```

Handles the spawning of entities for [Level](#) Two.

This function is responsible for spawning basic monsters and shooter enemies at specific intervals defined by the game parameters. It checks the elapsed time since the last spawn of each entity type and spawns new entities if the required time has passed.

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the ComponentManager instance.
<i>entityManager</i>	Reference to the EntityManager instance.
<i>newClock</i>	The current time point used for timing calculations.

6.44.3.10 `MoveUpdate()`

```
template<typename T >
void r_type::Level< T >::MoveUpdate (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::chrono::system_clock::time_point newClock ) [inline], [override]
```

Updates the positions of entities and notifies clients of any changes.

This function performs the following steps:

1. Retrieves the current positions of entities and stores them.
2. Moves the entities using the move system.
3. Compares the new positions with the previous positions.
4. If an entity's position has changed, sends an update message to all clients.

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the ComponentManager .
<i>entityManager</i>	Reference to the EntityManager .
<i>newClock</i>	The current time point.

6.44.3.11 `SetGameParameters()`

```
template<typename T >
void r_type::Level< T >::SetGameParameters (
    GameParameters gameParameters ) [inline]
```

Sets the game difficulty based on the provided game parameters.

This function sets the game difficulty based on the provided game parameters.

Parameters

<i>gameParameters</i>	The game parameters to set the difficulty.
-----------------------	--------------------------------------------

6.44.3.12 `SetSystem()`

```
template<typename T >
void r_type::Level< T >::SetSystem (
```

```

    ComponentManager & componentManager,
    EntityManager & entityManager ) [inline], [override]

```

Initializes and sets up various systems for the level.

This function overrides a base class method to initialize and set up the [MoveSystem](#), [CollisionSystem](#), [AnimationSystem](#), and [AutoFireSystem](#) using the provided [ComponentManager](#) and [EntityManager](#).

Parameters

<i>componentManager</i>	Reference to the ComponentManager used to manage components.
<i>entityManager</i>	Reference to the EntityManager used to manage entities.

6.44.3.13 SpawnEntity()

```

template<typename T >
void r\_type::Level< T >::SpawnEntity (
    r\_type::net::AServer< T > * server,
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int nbrOfEnemy,
    EnemyFactory::EnemyType enemyType ) [inline]

```

Spawns a specified number of enemy entities in the game.

This function creates and spawns a specified number of enemy entities of a given type at random positions within the game world. The enemy entities are then broadcasted to all connected clients.

Template Parameters

<i>T</i>	The type of the server.
----------	-------------------------

Parameters

<i>server</i>	A pointer to the server instance.
<i>entityManager</i>	Reference to the EntityManager responsible for managing entities.
<i>componentManager</i>	Reference to the ComponentManager responsible for managing components.
<i>nbrOfEnemy</i>	The number of enemy entities to spawn.
<i>enemyType</i>	The type of enemy to spawn (e.g., BasicMonster , ShooterEnemy).

6.44.3.14 Update()

```

template<typename T >
void r\_type::Level< T >::Update (
    r\_type::net::AServer< T > * server,

```

```

    ComponentManager & componentManager,
    EntityManager & entityManager,
    std::chrono::system_clock::time_point newClock,
    bool * bUpdateEntities ) [inline], [override]

```

Updates the game state by processing entity movements, handling collisions, and sending messages to clients.

This function performs several tasks to update the game state:

- Moves entities based on the elapsed time.
- Handles collisions between entities.
- Sends messages to clients about destroyed entities.
- Updates animations and firing mechanisms.

Parameters

<i>server</i>	Pointer to the server instance.
<i>componentManager</i>	Reference to the ComponentManager handling game components.
<i>entityManager</i>	Reference to the EntityManager handling game entities.
<i>newClock</i>	The current time point used to calculate elapsed time.
<i>bUpdateEntities</i>	Pointer to a boolean flag indicating whether entities should be updated.

6.44.4 Member Data Documentation

6.44.4.1 `_animationSystem`

```

template<typename T >
std::shared_ptr<AnimationSystem> r\_type::Level< T >::\_animationSystem [protected]

```

A shared pointer to the [AnimationSystem](#).

This member variable holds a shared pointer to an instance of the [AnimationSystem](#). The [AnimationSystem](#) is responsible for managing and updating animations within the game. Using a shared pointer ensures that the [AnimationSystem](#) instance is properly managed and its lifetime is controlled, preventing memory leaks and dangling pointers.

6.44.4.2 `_autoFireSystem`

```

template<typename T >
std::shared_ptr<AutoFireSystem> r\_type::Level< T >::\_autoFireSystem [protected]

```

A shared pointer to an instance of [AutoFireSystem](#).

This member variable holds a shared pointer to an [AutoFireSystem](#) object, which is responsible for managing the automatic firing mechanism in the game. The use of `std::shared_ptr` ensures that the [AutoFireSystem](#) instance is properly managed and deallocated when no longer in use.

6.44.4.3 `_basicMonsterSpawnTime`

```
template<typename T >
std::chrono::system_clock::time_point r\_type::Level< T >::_basicMonsterSpawnTime [protected]
```

Initial value:

```
=
    std::chrono::system_clock::now()
```

Represents the time point at which a basic monster is spawned.

This variable is initialized to the current system time using `std::chrono::system_clock::now()`. It is used to track the spawn time of a basic monster in the game.

6.44.4.4 `_collisionSystem`

```
template<typename T >
std::shared_ptr<CollisionSystem> r\_type::Level< T >::_collisionSystem [protected]
```

A shared pointer to the [CollisionSystem](#).

This member variable holds a shared pointer to an instance of the [CollisionSystem](#), which is responsible for handling collision detection and response within the game. Using a shared pointer ensures that the [CollisionSystem](#) instance is properly managed and its memory is automatically deallocated when no longer in use.

6.44.4.5 `_gameParameters`

```
template<typename T >
GameParameters r\_type::Level< T >::_gameParameters [protected]
```

Holds the parameters for the game configuration.

This member variable stores an instance of the `GameParameters` class, which contains various settings and configurations for the game.

6.44.4.6 `_moveSystem`

```
template<typename T >
std::shared_ptr<MoveSystem> r\_type::Level< T >::_moveSystem [protected]
```

A shared pointer to the [MoveSystem](#) instance.

This member variable holds a shared pointer to an instance of the [MoveSystem](#) class, which is responsible for handling movement logic within the system. Using a shared pointer ensures that the [MoveSystem](#) instance is properly managed and its lifetime is tied to the number of references to it.

6.44.4.7 _shooterEnemySpawnTime

```
template<typename T >
std::chrono::system_clock::time_point r_type::Level< T >::_shooterEnemySpawnTime [protected]
```

Initial value:

```
=
    std::chrono::system_clock::now()
```

Represents the time point when the shooter enemy is spawned.

This variable is initialized to the current system time using `std::chrono::system_clock::now()`. It is used to track the exact moment when the shooter enemy is spawned in the game.

6.44.4.8 _spawnTimeMonsterThree

```
template<typename T >
std::chrono::system_clock::time_point r_type::Level< T >::_spawnTimeMonsterThree [protected]
```

The time point at which the third type of monster is spawned.

This member variable holds the spawn time for the third type of monster using the system clock's time point. It is used to schedule or track when the third type of monster should appear in the game.

6.44.4.9 _WallSpawnTime

```
template<typename T >
std::chrono::system_clock::time_point r_type::Level< T >::_WallSpawnTime = std::chrono::system_↵
_clock::now() [protected]
```

Stores the time point when the wall was spawned.

This member variable holds the time point, using the system clock, at which the wall was spawned. It is initialized to the current time when the object is created.

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

- [/home/runner/work/R-Type/R-Type/Server/Interface/Include/level.hpp](#)

6.45 LinkForceComponent Struct Reference

Component that links an entity to a target entity by ID.

```
#include <link_force_component.hpp>
```

Public Member Functions

- [LinkForceComponent](#) (int _targetId)
Constructs a [LinkForceComponent](#) with the specified target entity ID.

Public Attributes

- int [targetId](#)

The ID of the target entity to which this entity is linked.

6.45.1 Detailed Description

Component that links an entity to a target entity by ID.

This component is used to establish a link between the current entity and a target entity identified by the `targetId`. This can be useful in scenarios where entities need to interact or be aware of each other.


6.45.2 Constructor & Destructor Documentation

6.45.2.1 LinkForceComponent()

```
LinkForceComponent::LinkForceComponent (
    int _targetId ) [inline]
```

Constructs a [LinkForceComponent](#) with the specified target entity ID.

Parameters

 <i>targetId</i>	The ID of the target entity.
--------------------------------------------------------------------------------------------------------	------------------------------

6.45.3 Member Data Documentation

6.45.3.1 targetId

```
LinkForceComponent::targetId
```

The ID of the target entity to which this entity is linked.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link_force_component.hpp](#)

6.46 MovementComponent Struct Reference

Represents a component that handles movement in the ECS system.

```
#include <movement_component.hpp>
```

Public Member Functions

- [MovementComponent](#) ()
- [MovementComponent](#) ([MovementType](#) *movementType*, [uint32_t](#) *index*, [bool](#) *move*)

Constructs a [MovementComponent](#) with the specified parameters.

Public Attributes

- [MovementType](#) *movementType*
The type of movement to be applied.
- [uint32_t](#) *index*
An index used to identify the movement component.
- [bool](#) *move*
A boolean flag indicating whether the entity should move.

6.46.1 Detailed Description

Represents a component that handles movement in the ECS system.

This component is used to define the movement behavior of an entity.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 MovementComponent() [1/2]

```
MovementComponent::MovementComponent ( ) [inline]
```

6.46.2.2 MovementComponent() [2/2]

```
MovementComponent::MovementComponent (
    MovementType movementType,
    uint32\_t index,
    bool move ) [inline]
```

Constructs a [MovementComponent](#) with the specified parameters.

Parameters

<i>movementType</i>	The type of movement to be applied.
<i>index</i>	An index used to identify the movement component.
<i>move</i>	A boolean flag indicating whether the entity should move.

6.46.3 Member Data Documentation

6.46.3.1 index

`MovementComponent::index`

An index used to identify the movement component.

6.46.3.2 move

`MovementComponent::move`

A boolean flag indicating whether the entity should move.

6.46.3.3 movementType

`MovementComponent::movementType`

The type of movement to be applied.

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

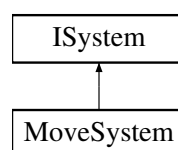
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp

6.47 MoveSystem Class Reference

System responsible for moving entities within the ECS framework.

```
#include <move_system.hpp>
```

Inheritance diagram for MoveSystem:



Public Member Functions

- [MoveSystem](#) ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager)
Constructs a new [MoveSystem](#) object.
- void [moveEntities](#) ([ComponentManager](#) &componentManager, [EntityManager](#) &entityManager)
Moves all entities managed by the system.
- void [moveEntity](#) ([ComponentManager](#) &componentManager, int entityId)
Moves a single entity.

Private Attributes

- [ComponentManager](#) &_componentManager
Reference to the [ComponentManager](#) instance.
- [EntityManager](#) &_entityManager
Reference to the [EntityManager](#) instance.

6.47.1 Detailed Description

System responsible for moving entities within the ECS framework.

The [MoveSystem](#) class handles the movement logic for entities in the game. It interacts with the [ComponentManager](#) and [EntityManager](#) to update the positions of entities based on their movement components.

6.47.2 Constructor & Destructor Documentation

6.47.2.1 MoveSystem()

```
MoveSystem::MoveSystem (  
    ComponentManager & componentManager,  
    EntityManager & entityManager ) [inline]
```

Constructs a new [MoveSystem](#) object.

Parameters

<i>componentManager</i>	Reference to the ComponentManager .
<i>entityManager</i>	Reference to the EntityManager .

6.47.3 Member Function Documentation

6.47.3.1 moveEntities()

```
void MoveSystem::moveEntities (
    ComponentManager & componentManager,
    EntityManager & entityManager )
```

Moves all entities managed by the system.

This function iterates through all entities and updates their positions based on their movement components.

Parameters

<i>componentManager</i>	Reference to the ComponentManager .
<i>entityManager</i>	Reference to the EntityManager .

6.47.3.2 moveEntity()

```
void MoveSystem::moveEntity (
    ComponentManager & componentManager,
    int entityId )
```

Moves a single entity.

This function updates the position of a specific entity based on its movement component.

Parameters

<i>componentManager</i>	Reference to the ComponentManager .
<i>entityId</i>	The ID of the entity to be moved.

6.47.4 Member Data Documentation

6.47.4.1 _componentManager

```
ComponentManager& MoveSystem::_componentManager [private]
```

Reference to the [ComponentManager](#) instance.

This member variable holds a reference to the [ComponentManager](#), which is responsible for managing all the components within the ECS ([Entity](#) Component System). It is used by the system to access and manipulate components associated with entities.

6.47.4.2 `_entityManager`

```
EntityManager& MoveSystem::_entityManager [private]
```

Reference to the [EntityManager](#) instance.

This member variable holds a reference to the [EntityManager](#), which is responsible for managing all entities within the ECS ([Entity](#) Component System). It provides functionalities to create, destroy, and manage entities and their components.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/move_system.hpp](#)
- [/home/runner/work/R-Type/R-Type/ECS/Src/Systems/move_system.cpp](#)

6.48 OffsetComponent Struct Reference

Component that represents an offset value.

```
#include <offset_component.hpp>
```

Public Attributes

- float [offset](#)

6.48.1 Detailed Description

Component that represents an offset value.

This component is used to store a floating-point offset value.

6.48.2 Member Data Documentation

6.48.2.1 `offset`

```
float OffsetComponent::offset
```

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/offset_component.hpp](#)

6.49 OnClickComponent Struct Reference

Component that handles click events.

```
#include <on_click_component.hpp>
```

Public Member Functions

- [OnClickComponent](#) (std::function< [IScenes](#) *([AScenes](#) *)> onClickfunction)
Constructs an [OnClickComponent](#) with the specified click handler function.

Public Attributes

- bool [isClicked](#) = false
Indicates whether the entity has been clicked.
- std::function< [IScenes](#) *([AScenes](#) *)> [onClick](#)
A function that is executed when the entity is clicked.

6.49.1 Detailed Description

Component that handles click events.

This component is used to determine if an entity has been clicked and to execute a specified function when the entity is clicked.

6.49.2 Constructor & Destructor Documentation

6.49.2.1 OnClickComponent()

```
OnClickComponent::OnClickComponent (
    std::function< IScenes *(AScenes *)> onClickfunction )    [inline]
```

Constructs an [OnClickComponent](#) with the specified click handler function.

Parameters

<i>onClickfunction</i>	The function to be executed when the entity is clicked.
------------------------	---------------------------------------------------------

6.49.3 Member Data Documentation

6.49.3.1 isClicked

```
OnClickComponent::isClicked = false
```

Indicates whether the entity has been clicked.

6.49.3.2 onClick

```
OnClickComponent::onClick
```

A function that is executed when the entity is clicked.

The function takes a pointer to an [AScenes](#) object and returns a pointer to an [IScenes](#) object.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/on_click_component.hpp

6.50 PlayerComponent Struct Reference

```
#include <player_component.hpp>
```

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

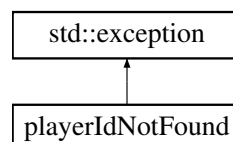
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_component.hpp

6.51 playerIdNotFound Class Reference

Exception class for handling cases where a player ID is not found.

```
#include <error_handling.hpp>
```

Inheritance diagram for playerIdNotFound:



Private Member Functions

- `const char * what () const` noexcept override

6.51.1 Detailed Description

Exception class for handling cases where a player ID is not found.

This exception is thrown when a requested player ID cannot be found in the system. It inherits from the standard `std::exception` class and overrides the `what()` method to provide a specific error message.

6.51.2 Member Function Documentation

6.51.2.1 `what()`

```
const char* playerIdNotFound::what ( ) const [inline], [override], [private], [noexcept]
```

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

6.52 PlayerMissileComponent Struct Reference

Component that represents a missile belonging to a player.

```
#include <player_missile_component.hpp>
```

Public Attributes

- `uint32_t playerId`
The unique identifier of the player who fired the missile.

6.52.1 Detailed Description

Component that represents a missile belonging to a player.

This component is used to identify missiles that are fired by players in the game.

6.52.2 Member Data Documentation

6.52.2.1 playerId

```
PlayerMissileComponent::playerId
```

The unique identifier of the player who fired the missile.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_missile_component.hpp

6.53 PositionComponent Struct Reference

A component that represents the position of an entity in 2D space.

```
#include <position_component.hpp>
```

Public Member Functions

- [PositionComponent](#) (float _x, float _y)

Public Attributes

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

6.53.1 Detailed Description

A component that represents the position of an entity in 2D space.

6.53.2 Constructor & Destructor Documentation

6.53.2.1 PositionComponent()

```
PositionComponent::PositionComponent (
    float _x,
    float _y ) [inline]
```

6.53.3 Member Data Documentation

6.53.3.1 x

```
float PositionComponent::x
```

6.53.3.2 y

```
float PositionComponent::y
```

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/[position_component.hpp](#)

6.54 PowerUpComponent Struct Reference

```
#include <power_up_component.hpp>
```

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/[power_up_component.hpp](#)

6.55 RectangleShapeComponent Struct Reference

A component that holds an sf::RectangleShape.

```
#include <rectangleShapeComponent.hpp>
```

Public Member Functions

- [RectangleShapeComponent](#) (sf::RectangleShape &[rectangleShape](#))
Constructs a new [RectangleShapeComponent](#).

Public Attributes

- sf::RectangleShape [rectangleShape](#)

6.55.1 Detailed Description

A component that holds an sf::RectangleShape.

This component is used to store and manage an sf::RectangleShape object.

6.55.2 Constructor & Destructor Documentation

6.55.2.1 RectangleShapeComponent()

```
RectangleShapeComponent::RectangleShapeComponent (  
    sf::RectangleShape & rectangleShape ) [inline]
```

Constructs a new [RectangleShapeComponent](#).

Parameters

<code>rectangleShape</code>	A reference to an <code>sf::RectangleShape</code> object.
-----------------------------	-----------------------------------------------------------

6.55.3 Member Data Documentation

6.55.3.1 rectangleShape

```
sf::RectangleShape RectangleShapeComponent::rectangleShape
```

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

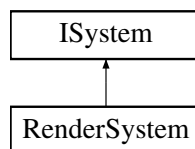
- </home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/rectangleShapeComponent.hpp>

6.56 RenderSystem Class Reference

A system responsible for rendering entities in the ECS framework.

```
#include <render_system.hpp>
```

Inheritance diagram for RenderSystem:



Public Member Functions

- [RenderSystem](#) (`sf::RenderWindow` &window, [ComponentManager](#) &componentManager)
- void [render](#) ([ComponentManager](#) &componentManager)
Renders the components managed by the [ComponentManager](#).

Private Attributes

- `sf::RenderWindow` & [_window](#)
Reference to the SFML RenderWindow used for rendering.
- [ComponentManager](#) & [_componentManager](#)
Reference to the [ComponentManager](#) instance.
- `sf::Font` [_font](#)
A font object used for drawing text in SFML.

6.56.1 Detailed Description

A system responsible for rendering entities in the ECS framework.

This class handles the rendering of entities using the SFML library. It requires a reference to an SFML `RenderWindow` and a [ComponentManager](#).

Parameters

<i>window</i>	Reference to the SFML <code>RenderWindow</code> where entities will be rendered.
<i>componentManager</i>	Reference to the ComponentManager that manages entity components.

Exceptions

failedToLoadFont	Exception thrown if the font file cannot be loaded.
----------------------------------	-----------------------------------------------------

6.56.2 Constructor & Destructor Documentation

6.56.2.1 `RenderSystem()`

```
RenderSystem::RenderSystem (
    sf::RenderWindow & window,
    ComponentManager & componentManager ) [inline]
```

6.56.3 Member Function Documentation

6.56.3.1 `render()`

```
void RenderSystem::render (
    ComponentManager & componentManager )
```

Renders the components managed by the [ComponentManager](#).

This function iterates through the components in the provided [ComponentManager](#) and performs rendering operations on them. It is typically called once per frame to update the visual representation of the components.

Parameters

<i>componentManager</i>	A reference to the ComponentManager that holds the components to be rendered.
-------------------------	-----------------------------------------------------------------------------------------------

6.56.4 Member Data Documentation

6.56.4.1 `_componentManager`

```
ComponentManager& RenderSystem::_componentManager [private]
```

Reference to the [ComponentManager](#) instance.

This member variable holds a reference to the [ComponentManager](#), which is responsible for managing all the components within the ECS ([Entity](#) Component System). It provides functionalities to add, remove, and access components associated with entities.

6.56.4.2 `_font`

```
sf::Font RenderSystem::_font [private]
```

A font object used for drawing text in SFML.

This member variable holds an instance of `sf::Font`, which is used to load and manage font resources for rendering text in the application. The `sf::Font` class provides functionality to load fonts from files, memory, or streams, and to retrieve font metrics and glyphs for text rendering.

6.56.4.3 `_window`

```
sf::RenderWindow& RenderSystem::_window [private]
```

Reference to the SFML `RenderWindow` used for rendering.

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

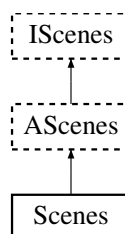
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/render_system.hpp`
- `/home/runner/work/R-Type/R-Type/ECS/Src/Systems/render_system.cpp`

6.57 Scenes Class Reference

Represents a class that manages different scenes in a game.

```
#include <scenes.hpp>
```

Inheritance diagram for Scenes:



Public Member Functions

- [Scenes](#) (std::string ip, int port)
Construct a new [Scenes](#) object.
- [~Scenes](#) ()=default
Destroy the [Scenes](#) object.
- void [mainMenu](#) ()
displays the main menu, creates all the necessary entities
- void [gameLoop](#) ()
displays the main game loop, creates all the necessary entities
- void [HandleMessage](#) (r_type::net::Message< TypeMessage > &msg, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, std::shared_ptr< [AudioSystem](#) > &audioSystem)
- void [StopGameLoop](#) (std::shared_ptr< [AudioSystem](#) > &audioSystem)
- void [settingsMenu](#) ()
displays the settings menu, creates all the necessary entities
- void [inGameMenu](#) ()
displays the in game menu, creates all the necessary entities
- void [difficultyChoices](#) ()
displays the difficulty choices, creates all the necessary entities
- void [render](#) ()
display what must be displayed (main menu, game loop, settings menu, in game menu), creates all the components needed and manages them
- bool [shouldQuit](#) ()
check if game should stop running
- sf::RenderWindow * [getRenderWindow](#) ()
Get the RenderWindow object.
- void [run](#) ()

Public Attributes

- sf::RenderWindow [_window](#)
- r_type::net::Client [_networkClient](#)

Additional Inherited Members

6.57.1 Detailed Description

Represents a class that manages different scenes in a game.

The [Scenes](#) class provides functionality to display and manage various scenes in a game, such as the main menu, game loop, settings menu, and in-game menu. It also allows setting the game mode and daltonism mode.

6.57.2 Constructor & Destructor Documentation

6.57.2.1 Scenes()

```
Scenes::Scenes (
    std::string ip,
    int port )
```

Construct a new [Scenes](#) object.

Parameters

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

6.57.2.2 ~Scenes()

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

Destroy the [Scenes](#) object.

6.57.3 Member Function Documentation

6.57.3.1 difficultyChoices()

```
void Scenes::difficultyChoices ( ) [virtual]
```

displays the difficulty choices, creates all the necessary entities

Implements [IScenes](#).

6.57.3.2 gameLoop()

```
void Scenes::gameLoop ( ) [virtual]
```

displays the main game loop, creates all the necessary entities

Implements [IScenes](#).

6.57.3.3 getRenderWindow()

```
sf::RenderWindow* Scenes::getRenderWindow ( ) [inline], [virtual]
```

Get the RenderWindow object.

Returns

sf::RenderWindow*

Implements [IScenes](#).

6.57.3.4 HandleMessage()

```
void Scenes::HandleMessage (
    r_type::net::Message< TypeMessage > & msg,
    ComponentManager & componentManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    std::shared_ptr< AudioSystem > & audioSystem )
```

6.57.3.5 inGameMenu()

```
void Scenes::inGameMenu ( ) [virtual]
```

displays the in game menu, creates all the necessary entities

This function handles the main game loop for the [Scenes](#) class.

It contains the logic for connecting to a server, updating entities, handling user input, and rendering the game.

The game loop performs the following steps:

1. Connects to a server using the [r_type::net::Client](#) class.
2. Initializes the [ComponentManager](#), [TextureManager](#), and [EntityManager](#).
3. Creates a background entity and sets its sprite component.
4. Defines lambda functions for updating player position and firing missiles.
5. Enters the main loop, which continues until the window is closed.
6. Within the loop, it checks for user input events and handles them accordingly.
7. If the server is connected, it processes incoming messages and updates entities accordingly.
8. It then updates the entities using the [UpdateSystem](#) and renders them using the [RenderSystem](#).

Note

This code assumes the presence of the [r_type::net::Client](#), [ComponentManager](#), [TextureManager](#), [EntityManager](#), [UpdateSystem](#), and [RenderSystem](#) classes.

See also

[r_type::net::Client](#)
[ComponentManager](#)
[TextureManager](#)
[EntityManager](#)
[UpdateSystem](#)
[RenderSystem](#)

Displays the in-game menu.

Implements [IScenes](#).

6.57.3.6 mainMenu()

```
void Scenes::mainMenu ( ) [virtual]
```

displays the main menu, creates all the necessary entities

Displays the main menu scene.

This function creates the main menu scene, including the background, buttons, and event handling. The main menu scene allows the user to navigate to different scenes by clicking on the buttons. The buttons include "Play", "↩ Settings", and "Quit". The function continuously updates and renders the scene until the user closes the window or navigates to a different scene.

Returns

void

Implements [IScenes](#).

6.57.3.7 render()

```
void Scenes::render ( ) [virtual]
```

display what must be displayed (main menu, game loop, settings menu, in game menu), creates all the components needed and manages them

Renders the current scene based on the value of currentScene.

The render function uses a switch statement to determine which scene to render. It calls the corresponding member function based on the value of currentScene.

Note

The currentScene variable must be set before calling this function.

Implements [IScenes](#).

6.57.3.8 run()

```
void Scenes::run ( )
```

6.57.3.9 settingsMenu()

```
void Scenes::settingsMenu ( ) [virtual]
```

displays the settings menu, creates all the necessary entities

Displays the settings menu.

This function is responsible for displaying the settings menu in the game. It does not return any value.

Implements [IScenes](#).

6.57.3.10 shouldQuit()

```
bool Scenes::shouldQuit ( ) [inline], [virtual]
```

check if game should stop running

Returns

true

false

Implements [IScenes](#).

6.57.3.11 StopGameLoop()

```
void Scenes::StopGameLoop (
    std::shared_ptr< AudioSystem > & audioSystem )
```

6.57.4 Member Data Documentation

6.57.4.1 _networkClient

```
r\_type::net::Client Scenes::_networkClient
```

6.57.4.2 _window

```
sf::RenderWindow Scenes::_window
```

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

- [/home/runner/work/R-Type/R-Type/Client/Interface/Include/scenes.hpp](#)
- [/home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp](#)

6.58 ScoreComponent Struct Reference

Component that holds the score of an entity.

```
#include <score_component.hpp>
```

Public Attributes

- `int` [score](#)

6.58.1 Detailed Description

Component that holds the score of an entity.

The [ScoreComponent](#) is used within the ECS framework to keep track of the score associated with a particular entity.

6.58.2 Member Data Documentation

6.58.2.1 score

```
int ScoreComponent::score
```

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

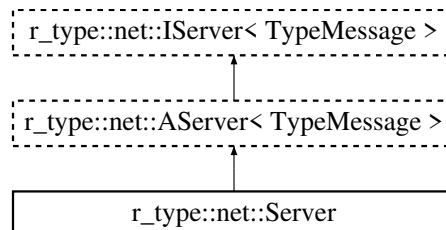
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/score_component.hpp](#)

6.59 r_type::net::Server Class Reference

A server class that handles client connections and messaging.

```
#include <server.hpp>
```

Inheritance diagram for r_type::net::Server:



Public Member Functions

- [Server](#) (uint16_t nPort)
Constructs a new [Server](#) object with the specified port number.
- [~Server](#) ()
Destructor for the [Server](#) class.

Protected Member Functions

- bool [OnClientConnect](#) (std::shared_ptr< r_type::net::Connection< TypeMessage >> client)
Handles the event when a client attempts to connect to the server.
- void [OnClientDisconnect](#) (std::shared_ptr< r_type::net::Connection< TypeMessage >> client, r_type::net::Message< TypeMessage > &msg)
Handles the event when a client disconnects from the server.
- void [OnMessage](#) (std::shared_ptr< r_type::net::Connection< TypeMessage >> client, r_type::net::Message< TypeMessage > &msg)
Handles incoming messages from a client.

Additional Inherited Members

6.59.1 Detailed Description

A server class that handles client connections and messaging.

This class inherits from r_type::net::AServer<TypeMessage> and provides implementations for handling client connections, disconnections, and message reception.

Template Parameters

<i>TypeMessage</i>	The type of message that the server will handle.
--------------------	--------------------------------------------------

6.59.2 Constructor & Destructor Documentation

6.59.2.1 Server()

```
r_type::net::Server::Server (
    uint16_t nPort ) [inline]
```

Constructs a new [Server](#) object with the specified port number.

This constructor initializes the [Server](#) object by calling the constructors of the base classes [r_type::net::IServer<TypeMessage>](#) and [r_type::net::AServer<TypeMessage>](#) with the provided port number.

Parameters

<i>nPort</i>	The port number on which the server will listen for incoming connections.
--------------	---------------------------------------------------------------------------

6.59.2.2 ~Server()

```
r_type::net::Server::~~Server ( ) [inline]
```

Destructor for the [Server](#) class.

This destructor is responsible for cleaning up any resources allocated by the [Server](#) class. Currently, it does not perform any specific actions.

6.59.3 Member Function Documentation

6.59.3.1 OnClientConnect()

```
bool r_type::net::Server::OnClientConnect (
    std::shared_ptr< r_type::net::Connection< TypeMessage >> client ) [protected]
```

Handles the event when a client attempts to connect to the server.

Parameters

<i>client</i>	A shared pointer to the client's connection object.
---------------	-----------------------------------------------------

Returns

true if the client is allowed to connect, false otherwise.

This function checks if the maximum number of players (4) has been reached. If so, it sends a denial message to the client and returns false. Otherwise, it sends an acceptance message to the client, increments the number of players, sets the client's status to INITIALISATION, assigns the last message sent to the client, and initializes the client's entities.

Parameters

<i>client</i>	A shared pointer to the client connection attempting to connect.
---------------	------------------------------------------------------------------

Returns

true if the client is accepted, false if the client is denied.

6.59.3.2 OnClientDisconnect()

```
void r_type::net::Server::OnClientDisconnect (
    std::shared_ptr< r_type::net::Connection< TypeMessage >> client,
    r_type::net::Message< TypeMessage > & msg ) [protected]
```

Handles the event when a client disconnects from the server.

Handles the disconnection of a client from the server.

Parameters

<i>client</i>	A shared pointer to the client's connection object.
<i>msg</i>	A reference to the message object containing information about the disconnection.

This function is called when a client disconnects from the server. It performs several tasks including removing the client, saving the player's score, removing associated entities, and notifying all other clients about the disconnection.

Parameters

<i>client</i>	A shared pointer to the connection object representing the client.
<i>msg</i>	A reference to the message object containing information about the disconnection.

6.59.3.3 OnMessage()

```
void r_type::net::Server::OnMessage (
    std::shared_ptr< r_type::net::Connection< TypeMessage >> client,
    r_type::net::Message< TypeMessage > & msg ) [protected]
```


Handles incoming messages from a client.

Handles the reception of a message from a client.

This function is called whenever a message is received from a client. It processes the message and performs the necessary actions based on the message content.

Parameters

<i>client</i>	A shared pointer to the client connection that sent the message.
<i>msg</i>	The message received from the client.

This function is called when a message is received from a client. It processes the message based on the client's status and the message's ID. The function performs different actions based on the message ID, such as sending a response message, updating player positions, creating entities, or destroying entities.

Parameters

<i>client</i>	A shared pointer to the connection object representing the client.
<i>msg</i>	A reference to the message object containing information sent by the client.

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

- [/home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/server.hpp](#)
- [/home/runner/work/R-Type/R-Type/Server/Src/server.cpp](#)

6.60 ShaderComponent Struct Reference

A component that holds a shader.

```
#include <shader_component.hpp>
```

Public Member Functions

- [ShaderComponent](#) (std::string path)
Constructs a [ShaderComponent](#) and loads a shader from a file.

Public Attributes

- std::shared_ptr< sf::Shader > [shader](#)
The shader object.

6.60.1 Detailed Description

A component that holds a shader.

This component is used to manage a shader in the ECS ([Entity](#) Component System). It loads a shader from a file and stores it in a shared pointer.

6.60.2 Constructor & Destructor Documentation

6.60.2.1 ShaderComponent()

```
ShaderComponent::ShaderComponent (
    std::string path ) [inline]
```

Constructs a [ShaderComponent](#) and loads a shader from a file.

Parameters

<i>path</i>	The file path to the shader.
-------------	------------------------------

This constructor creates a new `sf::Shader` object and attempts to load a shader from the specified file path. If the shader fails to load, an error message is printed to the standard error stream.

6.60.3 Member Data Documentation

6.60.3.1 shader

```
std::shared_ptr<sf::Shader> ShaderComponent::shader
```

The shader object.

This is a shared pointer to an `sf::Shader` object.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shader_component.hpp

6.61 ShootComponent Struct Reference

Component that handles shooting mechanics for an entity.

```
#include <shoot_component.hpp>
```

Public Member Functions

- [ShootComponent](#) (std::chrono::milliseconds cooldown)
Constructs a [ShootComponent](#) with a specified cooldown time.

Public Attributes

- `std::chrono::system_clock::time_point` [nextShootTime](#)
The time point when the entity is next allowed to shoot.
- `std::chrono::milliseconds` [cooldownTime](#)
The cooldown duration between consecutive shots.
- `bool` [canShoot](#)
A flag indicating whether the entity is currently allowed to shoot.

6.61.1 Detailed Description

Component that handles shooting mechanics for an entity.

This component keeps track of the next allowed shooting time, the cooldown period between shots, and whether the entity can shoot.

6.61.2 Constructor & Destructor Documentation

6.61.2.1 ShootComponent()

```
ShootComponent::ShootComponent (
    std::chrono::milliseconds cooldown )    [inline]
```

Constructs a [ShootComponent](#) with a specified cooldown time.

Initializes the `nextShootTime` to the current time and sets the `cooldownTime` to the provided value.

Parameters

<i>cooldown</i>	The cooldown duration between consecutive shots.
-----------------	--------------------------------------------------

6.61.3 Member Data Documentation

6.61.3.1 canShoot

```
ShootComponent::canShoot
```

A flag indicating whether the entity is currently allowed to shoot.

6.61.3.2 cooldownTime

`ShootComponent::cooldownTime`

The cooldown duration between consecutive shots.

6.61.3.3 nextShootTime

`ShootComponent::nextShootTime`

The time point when the entity is next allowed to shoot.

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_component.hpp`

6.62 SpriteComponent Struct Reference

A component that represents a sprite in the ECS ([Entity](#) Component System).

```
#include <sprite_component.hpp>
```

Public Member Functions

- [SpriteComponent](#) (`sf::Texture &texture, const float posX, float posY, const sf::Vector2f &scale, AScenes::SpriteType typeNb, sf::IntRect rect=sf::IntRect(0, 0, 0, 0)`)
Constructs a [SpriteComponent](#) with the given parameters.

Public Attributes

- `sf::Sprite` [sprite](#)
The SFML sprite object.
- [AScenes::SpriteType](#) `type`
The type of the sprite, defined by the [AScenes](#) namespace.
- `int` [hitboxX](#)
The width of the sprite's hitbox.
- `int` [hitboxY](#)
The height of the sprite's hitbox.

6.62.1 Detailed Description

A component that represents a sprite in the ECS ([Entity](#) Component System).

This component holds a sprite, its type, and hitbox dimensions. It provides functionality to initialize the sprite with a texture, position, scale, and optional texture rectangle.

6.62.2 Constructor & Destructor Documentation

6.62.2.1 SpriteComponent()

```
SpriteComponent::SpriteComponent (
    sf::Texture & texture,
    const float posX,
    float posY,
    const sf::Vector2f & scale,
    AScenes::SpriteType typeNb,
    sf::IntRect rect = sf::IntRect(0, 0, 0, 0) ) [inline]
```

Constructs a [SpriteComponent](#) with the given parameters.

Parameters

<i>texture</i>	The texture to be used for the sprite.
<i>posX</i>	The X position of the sprite.
<i>posY</i>	The Y position of the sprite.
<i>scale</i>	The scale of the sprite.
<i>typeNb</i>	The type of the sprite.
<i>rect</i>	The texture rectangle to be used for the sprite (default is an empty rectangle).

6.62.3 Member Data Documentation

6.62.3.1 hitboxX

```
int SpriteComponent::hitboxX
```

The width of the sprite's hitbox.

6.62.3.2 hitboxY

```
int SpriteComponent::hitboxY
```

The height of the sprite's hitbox.

6.62.3.3 sprite

```
sf::Sprite SpriteComponent::sprite
```

The SFML sprite object.

6.62.3.4 type

```
A::SpriteType SpriteComponent::type
```

The type of the sprite, defined by the [A::SpriteType](#) namespace.

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_component.hpp](#)

6.63 SpriteDataComponent Struct Reference

Component that holds data related to a sprite.

```
#include <sprite_data_component.hpp>
```

Public Attributes

- [SpritePath spritePath](#)
Path to the sprite resource.
- [vf2d scale](#)
Scale factor for the sprite.
- [A::SpriteType type](#)
Type of the sprite as defined in [A::SpriteType](#).

6.63.1 Detailed Description

Component that holds data related to a sprite.

This component contains information about the sprite's path, scale, and type.

6.63.2 Member Data Documentation

6.63.2.1 scale

```
SpriteDataComponent::scale
```

Scale factor for the sprite.

6.63.2.2 spritePath

```
SpriteDataComponent::spritePath
```

Path to the sprite resource.

6.63.2.3 type

```
SpriteDataComponent::type
```

Type of the sprite as defined in [AScenes::SpriteType](#).

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

- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_data_component.hpp](#)

6.64 TextComponent Struct Reference

A component that encapsulates an SFML text object.

```
#include <text_component.hpp>
```

Public Member Functions

- [TextComponent](#) (sf::Font &font, const std::string &string, float posX, float posY, int size=30)
Constructs a [TextComponent](#) with the specified parameters.

Public Attributes

- sf::Text [text](#)
The SFML text object that this component encapsulates.

6.64.1 Detailed Description

A component that encapsulates an SFML text object.

This component is used to manage and render text in an SFML application.

6.64.2 Constructor & Destructor Documentation

6.64.2.1 TextComponent()

```
TextComponent::TextComponent (
    sf::Font & font,
    const std::string & string,
    float posX,
    float posY,
    int size = 30 ) [inline]
```

Constructs a [TextComponent](#) with the specified parameters.

Parameters

<i>font</i>	The font to be used for the text.
<i>string</i>	The string to be displayed.
<i>posX</i>	The x-coordinate of the text's position.
<i>posY</i>	The y-coordinate of the text's position.
<i>size</i>	The character size of the text. Default is 30.

6.64.3 Member Data Documentation

6.64.3.1 text

```
sf::Text TextComponent::text
```

The SFML text object that this component encapsulates.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp

6.65 TextDataComponent Struct Reference

Component that holds text-related data for an entity.

```
#include <text_data_component.hpp>
```


Public Attributes

- [FontPath](#) `fontPath`
Path to the font file used for rendering the text.
- `uint32_t` `charSize` = 0
Size of the characters to be rendered.
- `uint32_t` `categoryIds` [5] = {0}
Array of category IDs associated with the text.
- [GameText](#) `categoryTexts` [5]
Array of GameText objects representing the text for each category.
- `uint32_t` `categorySize` = 0
Number of categories available.

6.65.1 Detailed Description

Component that holds text-related data for an entity.

This component is used to store information about text that can be rendered in the game, including font path, character size, category IDs, and category texts.

6.65.2 Member Data Documentation

6.65.2.1 categoryIds

```
TextDataComponent::categoryIds = {0}
```

Array of category IDs associated with the text.

6.65.2.2 categorySize

```
TextDataComponent::categorySize = 0
```

Number of categories available.

6.65.2.3 categoryTexts

```
TextDataComponent::categoryTexts
```

Array of GameText objects representing the text for each category.

6.65.2.4 charSize

```
TextDataComponent::charSize = 0
```

Size of the characters to be rendered.

6.65.2.5 fontPath

```
TextDataComponent::fontPath
```

Path to the font file used for rendering the text.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_data_component.hpp

6.66 TextureManager Class Reference

```
#include <texture_manager.hpp>
```

Public Member Functions

- `sf::Texture & getTexture (const std::string &filePath)`
Retrieves a texture from the texture manager.
- `void releaseTexture (const std::string &filePath)`
Releases the texture associated with the given file path.

Private Attributes

- `std::unordered_map< std::string, sf::Texture > textures`
A container for storing textures with string keys.

6.66.1 Member Function Documentation

6.66.1.1 `getTexture()`

```
sf::Texture& TextureManager::getTexture (  
    const std::string & filePath ) [inline]
```

Retrieves a texture from the texture manager.

This function attempts to find the texture associated with the given file path in the texture manager. If the texture is found, it is returned. Otherwise, a new texture is loaded from the file path and added to the texture manager before being returned.

Exceptions

<i>failedToLoadTexture</i>	If the texture fails to load from the file path.
--------------------------------------------	--------------------------------------------------

Parameters

<i>filePath</i>	The file path of the texture to retrieve.
-----------------	-------------------------------------------

Returns

sf::Texture& A reference to the retrieved texture.

6.66.1.2 releaseTexture()

```
void TextureManager::releaseTexture (
    const std::string & filePath ) [inline]
```

Releases the texture associated with the given file path.

This function removes the texture from the internal texture storage, effectively releasing any resources associated with it.

Parameters

<i>filePath</i>	The file path of the texture to be released.
-----------------	----------------------------------------------

6.66.2 Member Data Documentation

6.66.2.1 textures

```
std::unordered_map<std::string, sf::Texture> TextureManager::textures [private]
```

A container for storing textures with string keys.

This unordered map allows you to associate a string key with an sf::Texture object. It provides fast access to textures based on their keys.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/[texture_manager.hpp](#)

6.67 UIEntityInformation Struct Reference

Represents the information of a UI entity in the game.

```
#include <entity_struct.hpp>
```

Public Attributes

- uint32_t [uniqueID](#) = 0
Unique identifier for the UI entity.
- uint32_t [lives](#) = 0
Number of lives the UI entity has.
- uint32_t [score](#) = 0
Score associated with the UI entity.
- [SpriteDataComponent](#) [spriteData](#)
Data related to the sprite of the UI entity.
- [TextDataComponent](#) [textData](#)
Data related to the text of the UI entity.

6.67.1 Detailed Description

Represents the information of a UI entity in the game.

This structure holds various attributes related to a UI entity, including its unique identifier, lives, score, and associated sprite and text data components.

6.67.2 Member Data Documentation

6.67.2.1 lives

```
uint32_t UIEntityInformation::lives = 0
```

Number of lives the UI entity has.

6.67.2.2 score

```
uint32_t UIEntityInformation::score = 0
```

Score associated with the UI entity.

6.67.2.3 spriteData

`SpriteDataComponent` `UIEntityInformation::spriteData`

Data related to the sprite of the UI entity.

6.67.2.4 textData

`TextDataComponent` `UIEntityInformation::textData`

Data related to the text of the UI entity.

6.67.2.5 uniqueID

`uint32_t` `UIEntityInformation::uniqueID = 0`

Unique identifier for the UI entity.

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

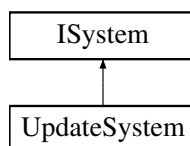
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/entity_struct.hpp`

6.68 UpdateSystem Class Reference

A system responsible for updating sprite positions in the game.

```
#include <update_system.hpp>
```

Inheritance diagram for UpdateSystem:



Public Member Functions

- `UpdateSystem` (`sf::RenderWindow &window`, `ComponentManager &componentManager`, `EntityManager &entityManager`)
Manages the update logic for entities within the ECS framework.
- `void updateSpritePositions` (`ComponentManager &componentManager`, `EntityManager &entityManager`)
Updates the positions of all sprite components in the game.

Private Attributes

- `sf::RenderWindow` & `_window`
Reference to the SFML `RenderWindow` used for rendering.
- `ComponentManager` & `_componentManager`
Reference to the `ComponentManager` instance.
- `EntityManager` & `_entityManager`
Reference to the `EntityManager` instance.

6.68.1 Detailed Description

A system responsible for updating sprite positions in the game.

The `UpdateSystem` class inherits from the `ISystem` interface and is responsible for updating the positions of sprites in the game. It interacts with the `ComponentManager` and `EntityManager` to manage and update the components and entities.

Parameters

<code>window</code>	Reference to the SFML <code>RenderWindow</code> object.
<code>componentManager</code>	Reference to the <code>ComponentManager</code> object.
<code>entityManager</code>	Reference to the <code>EntityManager</code> object.

6.68.2 Constructor & Destructor Documentation

6.68.2.1 UpdateSystem()

```
UpdateSystem::UpdateSystem (
    sf::RenderWindow & window,
    ComponentManager & componentManager,
    EntityManager & entityManager ) [inline]
```

Manages the update logic for entities within the ECS framework.

Parameters

<code>window</code>	Reference to the SFML <code>RenderWindow</code> used for rendering.
<code>componentManager</code>	Reference to the <code>ComponentManager</code> that handles components.
<code>entityManager</code>	Reference to the <code>EntityManager</code> that handles entities.

6.68.3 Member Function Documentation

6.68.3.1 updateSpritePositions()

```
void UpdateSystem::updateSpritePositions (
    ComponentManager & componentManager,
    EntityManager & entityManager )
```

Updates the positions of all sprite components in the game.

This function iterates through all entities that have sprite components and updates their positions based on their current velocities and other relevant factors.

Parameters

<i>componentManager</i>	Reference to the ComponentManager that manages all components.
<i>entityManager</i>	Reference to the EntityManager that manages all entities.

6.68.4 Member Data Documentation

6.68.4.1 _componentManager

```
ComponentManager& UpdateSystem::_componentManager [private]
```

Reference to the [ComponentManager](#) instance.

This member is used to manage and access various components within the ECS ([Entity](#) Component System).

6.68.4.2 _entityManager

```
EntityManager& UpdateSystem::_entityManager [private]
```

Reference to the [EntityManager](#) instance.

This member variable holds a reference to the [EntityManager](#), which is responsible for managing all entities within the ECS ([Entity](#) Component System). It provides functionalities to create, destroy, and query entities.

6.68.4.3 _window

```
sf::RenderWindow& UpdateSystem::_window [private]
```

Reference to the SFML `RenderWindow` used for rendering.

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/update_system.hpp`
- `/home/runner/work/R-Type/R-Type/ECS/Src/Systems/update_system.cpp`

6.69 VelocityComponent Struct Reference

Represents the velocity of an entity in 2D space.

```
#include <velocity_component.hpp>
```

Public Attributes

- float [x](#)
The velocity along the x-axis.
- float [y](#)
The velocity along the y-axis.

6.69.1 Detailed Description

Represents the velocity of an entity in 2D space.

This component stores the velocity of an entity along the x and y axes. It can be used to update the position of the entity based on its speed and direction.

6.69.2 Member Data Documentation

6.69.2.1 [x](#)

```
VelocityComponent::x
```

The velocity along the x-axis.

6.69.2.2 [y](#)

```
VelocityComponent::y
```

The velocity along the y-axis.

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

- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/velocity_component.hpp

6.70 vf2d Struct Reference

Represents a 2D vector with x and y coordinates.

```
#include <macros.hpp>
```


Public Attributes

- float `x` = 0
- float `y` = 0

6.70.1 Detailed Description

Represents a 2D vector with x and y coordinates.

6.70.2 Member Data Documentation

6.70.2.1 `x`

```
float vf2d::x = 0
```

6.70.2.2 `y`

```
float vf2d::y = 0
```

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/macros.hpp`

6.71 WallComponent Struct Reference

```
#include <wall_component.hpp>
```

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

- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wall_component.hpp`

Chapter 7

File Documentation

7.1 /home/runner/work/R-Type/R-Type/Client/Interface/↵ Include/mainmenu.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <r_type_client.hpp>
```

Functions

- int [MainMenu](#) (sf::RenderWindow *window, Rtype *rtype)

7.1.1 Function Documentation

7.1.1.1 MainMenu()

```
int MainMenu (
    sf::RenderWindow * window,
    Rtype * rtype )
```

7.2 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/a_↵ client.hpp File Reference

```
#include <Components/component_manager.hpp>
#include <Components/components.hpp>
#include <Net/i_client.hpp>
#include <SFML/Graphics.hpp>
#include <entity_struct.hpp>
#include <font_manager.hpp>
#include <texture_manager.hpp>
#include <unordered_map>
```

Classes

- class [r_type::net::AClient< T >](#)

Namespaces

- [r_type](#)
- [r_type::net](#)

7.3 /home/runner/work/R-Type/R-Type/Client/Interface/Include/↵ Net/client.hpp File Reference

```
#include <Net/a_client.hpp>
#include <SFML/Graphics.hpp>
#include <iostream>
```

Classes

- class [r_type::net::Client](#)

Namespaces

- [r_type](#)
- [r_type::net](#)

7.4 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_↵ client.hpp File Reference

```
#include <Net/common.hpp>
#include <Net/connection.hpp>
#include <Net/thread_safe_queue.hpp>
```

Classes

- class [r_type::net::IClient< T >](#)

Namespaces

- [r_type](#)
- [r_type::net](#)

7.5 /home/runner/work/R-Type/R-Type/Client/Interface/↵ Include/scenes.hpp File Reference

```
#include <Entities/entity.hpp>
#include <Net/client.hpp>
#include <SFML/Graphics.hpp>
#include <Systems/systems.hpp>
#include <a_scenes.hpp>
#include <memory>
#include <vector>
```

Classes

- class [Scenes](#)
Represents a class that manages different scenes in a game.

Functions

- std::string [keyToString](#) (sf::Keyboard::Key key)

7.5.1 Function Documentation

7.5.1.1 keyToString()

```
std::string keyToString (
    sf::Keyboard::Key key )
```

7.6 /home/runner/work/R-Type/R-Type/Client/Src/keyToString.cpp File Reference

```
#include <SFML/Window/Keyboard.hpp>
#include <iostream>
```

Functions

- std::string [keyToString](#) (sf::Keyboard::Key key)

7.6.1 Function Documentation

7.6.1.1 keyToString()

```
std::string keyToString (
    sf::Keyboard::Key key )
```

7.7 /home/runner/work/R-Type/R-Type/Client/Src/main.cpp File Reference

```
#include <iostream>
#include <macro.hpp>
#include <scenes.hpp>
#include <sstream>
```

Functions

- static bool [isValidIPv4](#) (const std::string &ip)
- static bool [isValidPort](#) (const std::string &portStr)
- int [main](#) (int const argc, char const *const *argv)

The entry point of the program.

7.7.1 Function Documentation

7.7.1.1 isValidIPv4()

```
static bool isValidIPv4 (
    const std::string & ip ) [static]
```

7.7.1.2 isValidPort()

```
static bool isValidPort (
    const std::string & portStr ) [static]
```

7.7.1.3 main()

```
int main (
    int const argc,
    char const *const * argv )
```

The entry point of the program.

This function initializes the Rtype object and runs the game.

Returns

0 indicating successful program execution.
int

7.8 /home/runner/work/R-Type/R-Type/Server/Src/main.cpp File Reference

```
#include <Net/server.hpp>
#include <iostream>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
```

Functions

- void `signal_handler` (int signal)
Signal handler for SIGINT.
- static bool `isValidPort` (const std::string &portStr)
Validates if a given string represents a valid port number.
- int `main` (int const argc, char const *const *const argv)
Entry point for the server application.

Variables

- static bool `loopRunning` = true
A static boolean flag to control the main loop execution.

7.8.1 Function Documentation

7.8.1.1 isValidPort()

```
static bool isValidPort (
    const std::string & portStr ) [static]
```

Validates if a given string represents a valid port number.

This function checks if the provided string is a valid port number within the range of 1024 to 65535. It performs the following checks:

- The string is not empty and does not exceed 5 characters in length.
- The string contains only digit characters.
- The integer value of the string is within the valid port range.

Parameters

<code>portStr</code>	The string representation of the port number to validate.
----------------------	-----------------------------------------------------------

Returns

true if the string is a valid port number within the range 1024-65535, false otherwise.

7.8.1.2 main()

```
int main (
    int const argc,
    char const *const *const argv )
```

Entry point for the server application.

This function initializes the server, sets up signal handling, and enters the main loop.

Parameters

<i>argc</i>	The number of command-line arguments.
<i>argv</i>	The array of command-line arguments. The first argument should be the port number on which the server will listen.

Returns

Returns an error code if the usage is incorrect or the port number is invalid. Returns OK upon successful execution.

7.8.1.3 signal_handler()

```
void signal_handler (
    int signal )
```

Signal handler for SIGINT.

This function is called when the program receives a SIGINT signal (usually generated by pressing Ctrl+C). It sets the global variable `loopRunning` to false, which can be used to gracefully terminate a running loop.

Parameters

<i>signal</i>	The signal number received by the handler.
---------------	--------------------------------------------

7.8.2 Variable Documentation

7.8.2.1 loopRunning

```
bool loopRunning = true [static]
```

A static boolean flag to control the main loop execution.

This variable is used to determine whether the main loop should continue running. It is set to true initially, and can be modified to false to stop the loop.

7.9 /home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp File Reference

```
#include <Components/components.hpp>
#include <Entities/entity_factory.hpp>
#include <Entities/entity_manager.hpp>
#include <Net/client.hpp>
#include <Systems/systems.hpp>
#include <audio_manager.hpp>
#include <chrono>
#include <creatable_client_object.hpp>
#include <font_manager.hpp>
#include <functional>
#include <iostream>
#include <scenes.hpp>
#include <sound_path.hpp>
#include <texture_manager.hpp>
```

Functions

- void [reloadFilter](#) (sf::RectangleShape &rectangle, [AScenes::DaltonismMode](#) mode)
- void [handleEvents](#) (sf::Event event, [ComponentManager](#) &componentManager, sf::RenderWindow *_window, std::vector< std::shared_ptr< [Entity](#) >> buttons, [Scenes](#) *scenes)
Handles events for the scene, including window close and mouse button press events.
- void [createDaltonismChoiceButtons](#) (std::vector< std::shared_ptr< [Entity](#) >> &buttons, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, [EntityFactory](#) &entityFactory)
- sf::Keyboard::Key [waitForKey](#) (sf::RenderWindow *_window)
- void [createKeyBindingButtons](#) (std::vector< std::shared_ptr< [Entity](#) >> &buttons, [ComponentManager](#) &componentManager, [EntityManager](#) &entityManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, [EntityFactory](#) &entityFactory, std::map< [Scenes::Actions](#), sf::Keyboard::Key > &keyBinds)

7.9.1 Function Documentation

7.9.1.1 createDaltonismChoiceButtons()

```
void createDaltonismChoiceButtons (
    std::vector< std::shared_ptr< Entity >> & buttons,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    EntityFactory & entityFactory )
```

7.9.1.2 createKeyBindingButtons()

```
void createKeyBindingButtons (
    std::vector< std::shared_ptr< Entity >> & buttons,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    EntityFactory & entityFactory,
    std::map< Scenes::Actions, sf::Keyboard::Key > & keyBinds )
```

7.9.1.3 handleEvents()

```
void handleEvents (
    sf::Event event,
    ComponentManager & componentManager,
    sf::RenderWindow * _window,
    std::vector< std::shared_ptr< Entity >> buttons,
    Scenes * scenes )
```

Handles events for the scene, including window close and mouse button press events.

This function processes events from the given `RenderWindow` and performs actions based on the type of event. It handles window close events and mouse button press events. For mouse button press events, it checks if the left mouse button was pressed and if the click occurred within the bounds of any button entities. If a button is clicked, it triggers the associated `OnClickComponent` or `BindComponent` actions.

Parameters

<i>event</i>	The event to handle.
<i>componentManager</i>	Reference to the <code>ComponentManager</code> to access components of entities.
<i>_window</i>	Pointer to the <code>RenderWindow</code> where events are polled from.
<i>buttons</i>	Vector of shared pointers to <code>Entity</code> objects representing buttons.

7.9.1.4 reloadFilter()

```
void reloadFilter (
    sf::RectangleShape & rectangle,
    AScenes::DaltonismMode mode )
```

7.9.1.5 waitForKey()

```
sf::Keyboard::Key waitForKey (
    sf::RenderWindow * _window )
```

7.10 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp File Reference

```
#include "Entities/entity.hpp"
#include "i_scenes.hpp"
#include <memory>
```

Classes

- class [AScenes](#)

7.11 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/audio_manager.hpp File Reference

```
#include "error_handling.hpp"
#include <SFML/Audio.hpp>
#include <memory>
#include <string>
#include <unordered_map>
```

Classes

- class [AudioManager](#)
Manages and caches sound buffers for efficient audio playback.

7.12 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_component.hpp File Reference

Defines the [AllyComponent](#) structure.

Classes

- struct [AllyComponent](#)

7.12.1 Detailed Description

Defines the [AllyComponent](#) structure.

The [AllyComponent](#) is used to mark entities as allies within the ECS framework.

7.13 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_missile_component.hpp](#) File Reference

Defines the [AllyMissileComponent](#) structure.

Classes

- struct [AllyMissileComponent](#)

7.13.1 Detailed Description

Defines the [AllyMissileComponent](#) structure.

The [AllyMissileComponent](#) is used to represent a missile fired by an ally in the game. This component can be attached to an entity to give it the behavior and properties of an ally missile.

7.14 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/animation_component.hpp](#) File Reference

```
#include <macros.hpp>
```

Classes

- struct [AnimationComponent](#)
A component that holds animation properties such as offset and dimension.

Functions

- bool [operator!=](#) ([AnimationComponent](#) animation, [AnimationComponent](#) other)
Inequality operator for [AnimationComponent](#).

7.14.1 Function Documentation

7.14.1.1 `operator!=()`

```
bool operator!= (
    AnimationComponent animation,
    AnimationComponent other )
```

Inequality operator for [AnimationComponent](#).

This operator checks if two [AnimationComponent](#) instances are not equal.

Parameters

<i>animation</i>	The first AnimationComponent instance.
<i>other</i>	The second AnimationComponent instance.

Returns

true if the two [AnimationComponent](#) instances are not equal, false otherwise.

This operator compares two [AnimationComponent](#) objects to determine if they are not equal. Two [AnimationComponent](#) objects are considered not equal if any of their respective offset or dimension coordinates differ.

Parameters

<i>animation</i>	The first AnimationComponent to compare.
<i>other</i>	The second AnimationComponent to compare.

Returns

true if the [AnimationComponent](#) objects are not equal, false otherwise.

7.15 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/background_component.hpp File Reference

Defines the [BackgroundComponent](#) structure.

Classes

- struct [BackgroundComponent](#)

7.15.1 Detailed Description

Defines the [BackgroundComponent](#) structure.

The [BackgroundComponent](#) is used to represent the background in the ECS ([Entity](#) Component System). This component can be attached to entities that require a background.

7.16 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/basic_monster_component.hpp File Reference

Defines the [BasicMonsterComponent](#) structure.

Classes

- struct [BasicMonsterComponent](#)

7.16.1 Detailed Description

Defines the [BasicMonsterComponent](#) structure.

This component is used to represent basic monster entities in the game.

7.17 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/bind_component.hpp File Reference

```
#include "a_scenes.hpp"
#include "i_scenes.hpp"
#include <functional>
```

Classes

- struct [BindComponent](#)
A component that binds a function to handle scene transitions.

7.18 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/boss_component.hpp File Reference

Defines the [BossComponent](#) structure.

Classes

- struct [BossComponent](#)

7.18.1 Detailed Description

Defines the [BossComponent](#) structure.

This file contains the definition of the [BossComponent](#) structure, which is used to represent a boss entity in the ECS ([Entity Component System](#)) framework.

7.19 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/component_manager.hpp File Reference

```
#include "components.hpp"
#include "texture_manager.hpp"
#include <any>
#include <iostream>
#include <memory>
#include <optional>
#include <typeindex>
#include <unordered_map>
```

Classes

- class [ComponentManager](#)
Manages the components of entities in an ECS system.

7.20 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/components.hpp File Reference

```
#include "ally_component.hpp"
#include "ally_missile_component.hpp"
#include "animation_component.hpp"
#include "background_component.hpp"
#include "basic_monster_component.hpp"
#include "bind_component.hpp"
#include "enemy_component.hpp"
#include "enemy_missile_component.hpp"
#include "force_missile_component.hpp"
#include "force_weapon_component.hpp"
#include "front_component.hpp"
#include "health_component.hpp"
#include "hitbox_component.hpp"
#include "input_component.hpp"
#include "link_force_component.hpp"
#include "movement_component.hpp"
#include "offset_component.hpp"
#include "on_click_component.hpp"
#include "player_component.hpp"
#include "player_missile_component.hpp"
#include "position_component.hpp"
#include "power_up_component.hpp"
#include "rectangleShapeComponent.hpp"
#include "score_component.hpp"
#include "shoot_component.hpp"
#include "sprite_component.hpp"
#include "sprite_data_component.hpp"
#include "text_component.hpp"
#include "text_data_component.hpp"
#include "velocity_component.hpp"
#include "wall_component.hpp"
```


7.21 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_component.hpp File Reference

Defines the [EnemyComponent](#) structure.

Classes

- struct [EnemyComponent](#)

7.21.1 Detailed Description

Defines the [EnemyComponent](#) structure.

This file contains the definition of the [EnemyComponent](#) structure, which is used to represent an enemy entity in the game. The structure itself is currently empty, but it can be extended in the future to include properties and behaviors specific to enemies.

7.22 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_missile_component.hpp File Reference

Defines the [EnemyMissileComponent](#) structure.

Classes

- struct [EnemyMissileComponent](#)

7.22.1 Detailed Description

Defines the [EnemyMissileComponent](#) structure.

This component is used to represent an enemy missile in the game.

7.23 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_missile_component.hpp File Reference

```
#include <stdint>
```

Classes

- struct [ForceMissileComponent](#)
Component representing a force missile in the ECS system.

7.24 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_weapon_component.hpp](#) File Reference

```
#include <stdint>
```

Classes

- struct [ForceWeaponComponent](#)
Represents a component for a force weapon in the game.

7.25 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/front_component.hpp](#) File Reference

```
#include <Entities/entity.hpp>  
#include <memory>
```

Classes

- struct [FrontComponent](#)
A component that represents the front of an entity.

7.26 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/health_component.hpp](#) File Reference

Classes

- struct [HealthComponent](#)
Represents the health attributes of an entity.

7.27 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/hitbox_component.hpp](#) File Reference

Classes

- struct [HitboxComponent](#)
Represents the hitbox dimensions of an entity.

7.28 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/input_component.hpp File Reference

Classes

- struct [InputComponent](#)
Component for handling input actions.

Enumerations

- enum class [InputType](#) {
 [UP](#) , [DOWN](#) , [LEFT](#) , [RIGHT](#) ,
 [SHOOT](#) , [QUIT](#) , [NONE](#) }
Enumeration of possible input actions.

7.28.1 Enumeration Type Documentation

7.28.1.1 InputType

```
enum InputType [strong]
```

Enumeration of possible input actions.

This enumeration defines the different types of inputs that can be handled by the [InputComponent](#).

Enumerator

UP	Represents the "up" input action.
DOWN	Represents the "down" input action.
LEFT	Represents the "left" input action.
RIGHT	Represents the "right" input action.
SHOOT	Represents the "shoot" input action.
QUIT	Represents the "quit" input action.
NONE	Represents no input action.

7.29 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp File Reference

```
#include <string>
```

Classes

- struct [labelComponent](#)

Represents a label component with a name and position coordinates.

7.30 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link_force_component.hpp](#) File Reference

Classes

- struct [LinkForceComponent](#)

Component that links an entity to a target entity by ID.

7.31 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp](#) File Reference

```
#include <stdint>
```

Classes

- struct [MovementComponent](#)

Represents a component that handles movement in the ECS system.

Enumerations

- enum class [MovementType](#) { [WIGGLE](#) , [DIAGONAL](#) , [CIRCLE](#) , [STRAIGHT](#) }

Enumeration of different types of movement behaviors.

7.31.1 Enumeration Type Documentation

7.31.1.1 MovementType

```
enum MovementType [strong]
```

Enumeration of different types of movement behaviors.

Enumerator

WIGGLE	Represents a wiggling movement pattern.
DIAGONAL	Represents a diagonal movement pattern.
CIRCLE	Represents a circular movement pattern.
STRAIGHT	

7.32 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/offset_component.hpp File Reference

Classes

- struct [OffsetComponent](#)
Component that represents an offset value.

7.33 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/on_click_component.hpp File Reference

Defines the [OnClickComponent](#) structure used for handling click events in the ECS system.

```
#include <a_scenes.hpp>
#include <functional>
#include <i_scenes.hpp>
```

Classes

- struct [OnClickComponent](#)
Component that handles click events.

7.33.1 Detailed Description

Defines the [OnClickComponent](#) structure used for handling click events in the ECS system.

7.34 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_component.hpp File Reference

Defines the [PlayerComponent](#) structure.

Classes

- struct [PlayerComponent](#)

7.34.1 Detailed Description

Defines the [PlayerComponent](#) structure.

The [PlayerComponent](#) structure is used to represent a player entity within the ECS ([Entity](#) Component System) framework of the R-Type project.

7.35 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_missile_component.hpp](#) File Reference

```
#include <stdint>
```

Classes

- struct [PlayerMissileComponent](#)
Component that represents a missile belonging to a player.

7.36 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/position_component.hpp](#) File Reference

Classes

- struct [PositionComponent](#)
A component that represents the position of an entity in 2D space.

7.37 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/power_up_component.hpp](#) File Reference

Defines the [PowerUpComponent](#) structure.

Classes

- struct [PowerUpComponent](#)

7.37.1 Detailed Description

Defines the [PowerUpComponent](#) structure.

The [PowerUpComponent](#) structure is used to represent a power-up in the game. It can be attached to entities to give them special abilities or enhancements.

7.38 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/rectangleShapeComponent.hpp](#) File Reference

```
#include <SFML/Graphics.hpp>
```

Classes

- struct [RectangleShapeComponent](#)
A component that holds an sf::RectangleShape.

7.39 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/score_component.hpp File Reference

Defines the [ScoreComponent](#) struct used to store the score of an entity.

Classes

- struct [ScoreComponent](#)
Component that holds the score of an entity.

7.39.1 Detailed Description

Defines the [ScoreComponent](#) struct used to store the score of an entity.

7.40 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shader_component.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <memory>
```

Classes

- struct [ShaderComponent](#)
A component that holds a shader.

7.41 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_component.hpp File Reference

```
#include <chrono>
```

Classes

- struct [ShootComponent](#)
Component that handles shooting mechanics for an entity.

7.42 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/sprite_component.hpp File Reference

```
#include "a_scenes.hpp"
#include <SFML/Graphics.hpp>
#include <string>
```

Classes

- struct [SpriteComponent](#)

A component that represents a sprite in the ECS ([Entity](#) Component System).

7.43 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/sprite_data_component.hpp File Reference

```
#include "../error_handling.hpp"
#include "../sprite_path.hpp"
#include "animation_component.hpp"
#include "position_component.hpp"
#include <SFML/Graphics.hpp>
#include <a_scenes.hpp>
#include <cstdint>
#include <macros.hpp>
#include <string>
```

Classes

- struct [SpriteDataComponent](#)

Component that holds data related to a sprite.

Functions

- `std::ostream & operator<< (std::ostream &os, const SpriteDataComponent &spriteData)`
Overloads the << operator to output the contents of a [SpriteDataComponent](#) to an ostream.

7.43.1 Function Documentation

7.43.1.1 `operator<<()`

```
std::ostream& operator<< (
    std::ostream & os,
    const SpriteDataComponent & spriteData )
```

Overloads the << operator to output the contents of a [SpriteDataComponent](#) to an ostream.

Parameters

<code>os</code>	The output stream to which the SpriteDataComponent will be written.
<code>spriteData</code>	The SpriteDataComponent instance to be written to the output stream.

Returns

`std::ostream&` The output stream after writing the [SpriteDataComponent](#).

7.44 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Classes

- struct [TextComponent](#)
A component that encapsulates an SFML text object.

7.45 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_data_component.hpp File Reference

```
#include "../font_path.hpp"  
#include "../game_text.hpp"
```

Classes

- struct [TextDataComponent](#)
Component that holds text-related data for an entity.

7.46 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/velocity_component.hpp File Reference

Classes

- struct [VelocityComponent](#)
Represents the velocity of an entity in 2D space.

7.47 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wall_component.hpp](#) File Reference

Defines the [WallComponent](#) structure.

Classes

- struct [WallComponent](#)

7.47.1 Detailed Description

Defines the [WallComponent](#) structure.

The [WallComponent](#) is a marker component used to identify entities that represent walls in the ECS ([Entity](#) Component System).

7.48 [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/creatable_client_object.hpp](#) File Reference

```
#include <cstdint>
```

Enumerations

- enum class [CreatableClientObject](#) : `uint32_t` { [PLAYERMISSILE](#) , [NONE](#) }
Enum representing the types of client objects that can be created.

7.48.1 Enumeration Type Documentation

7.48.1.1 CreatableClientObject

```
enum CreatableClientObject : uint32_t [strong]
```

Enum representing the types of client objects that can be created.

This enum is used to specify the different types of objects that can be instantiated on the client side in the R-Type game.

Enumerator

PLAYERMISSILE	Represents a missile fired by the player.
NONE	Represents the absence of a creatable client object.

7.49 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hpp File Reference

Classes

- class [Entity](#)

Represents an entity in the ECS system.

7.50 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_factory.hpp File Reference

```
#include "a_scenes.hpp"
#include "i_entity_factory.hpp"
#include "i_scenes.hpp"
#include <functional>
#include <game_struct.h>
```

Classes

- class [EntityFactory](#)

A factory class for creating various types of entities.

7.51 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp File Reference

```
#include "../error_handling.hpp"
#include "entity.hpp"
#include <algorithm>
#include <memory>
#include <optional>
#include <vector>
```

Classes

- class [EntityManager](#)

Manages the creation, removal, and retrieval of entities.

7.52 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/i_entity_factory.hpp File Reference

```
#include "Components/component_manager.hpp"
#include "entity.hpp"
#include "entity_manager.hpp"
#include "font_manager.hpp"
#include "texture_manager.hpp"
```

Classes

- class [IEntityFactory](#)
The interface for an entity factory.

7.53 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/entity_↵ struct.hpp File Reference

```
#include "Components/sprite_data_component.hpp"  
#include "Components/text_data_component.hpp"  
#include <cstdint>  
#include <macros.hpp>
```

Classes

- struct [EntityInformation](#)
Represents information about an entity.
- struct [UIEntityInformation](#)
Represents the information of a UI entity in the game.

7.54 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_↵ handling.hpp File Reference

```
#include <exception>
```

Classes

- class [componentNotFound](#)
Exception class for when a component is not found.
- class [entityNotFound](#)
Exception class for entity not found error.
- class [failedToLoadTexture](#)
Exception class for failed texture loading.
- class [failedToLoadSound](#)
Exception class for handling sound loading failures.
- class [failedToLoadFont](#)
Exception class for handling font loading failures.
- class [playerIdNotFound](#)
Exception class for handling cases where a player ID is not found.
- class [failedToCreateFile](#)
Exception class for handling file creation failures.
- class [failedToOpenFile](#)
Exception class for handling file opening failures.

7.55 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_manager.hpp File Reference

```
#include "error_handling.hpp"
#include <SFML/Graphics.hpp>
#include <string>
#include <unordered_map>
```

Classes

- class [FontManager](#)
Manages the loading and retrieval of font resources.

7.56 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_path.hpp File Reference

```
#include <cstdint>
#include <string>
```

Enumerations

- enum class [FontPath](#) : uint32_t { [MAIN](#) , [NONE](#) }
Enumeration of font paths.

Functions

- std::string [FontFactory](#) ([FontPath](#) font)
Creates a font object from the given font path.
- std::ostream & [operator<<](#) (std::ostream &os, const [FontPath](#) &fontPath)
Overloads the stream insertion operator to output the FontPath object.

7.56.1 Enumeration Type Documentation

7.56.1.1 FontPath

```
enum FontPath : uint32_t [strong]
```

Enumeration of font paths.

The FontPath enumeration contains a list of font paths that can be used to

specify the location of a font resource. Each font path corresponds to a specific font file that can be loaded and used by the application.

Example usage:

```
FontPath fontPath = FontPath::MAIN;  
std::string font = FontFactory(fontPath);
```

See also

[FontFactory](#)
[operator<<](#)
[FontManager](#)
[FontPath](#)

Note

The NONE font path is used to indicate that no font should be loaded.

Enumerator

MAIN	
NONE	

7.56.2 Function Documentation

7.56.2.1 FontFactory()

```
std::string FontFactory (  
    FontPath font )
```

Creates a font object from the given font path.

This function takes a FontPath object and returns a string representation of the font. The FontPath object should contain the necessary information to locate and load the font.

Parameters

<i>font</i>	The FontPath object containing the path to the font.
-------------	------------------------------------------------------

Returns

std::string The string representation of the font.

7.56.2.2 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const FontPath & fontPath )
```

Overloads the stream insertion operator to output the FontPath object.

This function allows the FontPath object to be output to an ostream, such as std::cout or any other output stream, by using the << operator.

Parameters

<i>os</i>	The output stream to which the FontPath object will be written.
<i>fontPath</i>	The FontPath object to be written to the output stream.

Returns

A reference to the output stream after the FontPath object has been written.

7.57 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/game_text.hpp File Reference

```
#include <stdint>
#include <string>
```

Enumerations

- enum class [GameText](#) : uint32_t { [Lives](#) , [Score](#) , [NONE](#) }
Enumeration for different types of game text.

Functions

- std::string [GameTextFactory](#) ([GameText](#) text)
Factory function to convert GameText enum to a string.
- std::ostream & [operator<<](#) (std::ostream &os, const [GameText](#) &text)
Overloaded stream insertion operator for GameText.

7.57.1 Enumeration Type Documentation

7.57.1.1 GameText

```
enum GameText : uint32_t [strong]
```

Enumeration for different types of game text.

This enumeration defines the different types of text that can be displayed in the game.

Enumerator

Lives	Represents the number of lives left.
Score	Represents the player's score.
NONE	Represents no text.

7.57.2 Function Documentation

7.57.2.1 GameTextFactory()

```
std::string GameTextFactory (  
    GameText text )
```

Factory function to convert GameText enum to a string.

Parameters

<i>text</i>	The GameText enum value.
-------------	--------------------------

Returns

A string representation of the GameText value.

7.57.2.2 operator<<()

```
std::ostream& operator<< (  
    std::ostream & os,  
    const GameText & text )
```

Overloaded stream insertion operator for GameText.

Parameters

<i>os</i>	The output stream.
<i>text</i>	The GameText enum value.

Returns

The output stream with the GameText value inserted.

7.58 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/hitbox_tmp.hpp File Reference

```
#include <Components/component_manager.hpp>
#include <Entities/entity.hpp>
#include <Entities/entity_manager.hpp>
#include <entity_struct.hpp>
```

Functions

- int [CheckEntityPosition](#) (uint32_t entityId, [ComponentManager](#) componentManager, [EntityManager](#) entityManager)

Checks the position of an entity within the game world.
- int [CheckEntityMovement](#) ([EntityInformation](#) desc, [ComponentManager](#) componentManager, [EntityManager](#) entityManager)

Checks the movement of an entity within the game.

7.58.1 Function Documentation

7.58.1.1 CheckEntityMovement()

```
int CheckEntityMovement (
    EntityInformation desc,
    ComponentManager componentManager,
    EntityManager entityManager )
```

Checks the movement of an entity within the game.

Parameters

<i>desc</i>	An EntityInformation object containing details about the entity.
<i>componentManager</i>	A ComponentManager object to manage the components of entities.
<i>entityManager</i>	An EntityManager object to manage the entities.

Returns

An integer indicating the result of the movement check.

7.58.1.2 CheckEntityPosition()

```
int CheckEntityPosition (
    uint32_t entityId,
    ComponentManager componentManager,
    EntityManager entityManager )
```

Checks the position of an entity within the game world.

This function retrieves and checks the position of the specified entity using the provided component and entity managers.

Parameters

<i>entityId</i>	The unique identifier of the entity whose position is to be checked.
<i>componentManager</i>	The manager responsible for handling components of entities.
<i>entityManager</i>	The manager responsible for handling entities.

Returns

An integer representing the status or result of the position check.

7.59 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/i_↵ scenes.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Classes

- class [IScenes](#)
Interface for managing different scenes in a game.

7.60 /home/runner/work/R-Type/R-Type/ECS/Interface/↵ Include/macros.hpp File Reference

Classes

- struct [vf2d](#)
Represents a 2D vector with x and y coordinates.

Macros

- #define [SCREEN_WIDTH](#) 1920
- #define [SCREEN_HEIGHT](#) 1080

7.60.1 Macro Definition Documentation

7.60.1.1 SCREEN_HEIGHT

```
#define SCREEN_HEIGHT 1080
```

7.60.1.2 SCREEN_WIDTH

```
#define SCREEN_WIDTH 1920
```

7.61 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/sound_path.hpp File Reference

```
#include <stdint>
#include <string>
```

Enumerations

- enum class [ActionType](#) : uint32_t {
 [Win](#) , [Shot](#) , [Boss](#) , [PowerUp](#) ,
 [GameOver](#) , [BossDeath](#) , [Explosion](#) , [Background](#) ,
 [NONE](#) }

This header file defines the ActionType enumeration and declares the SoundFactory function.

Functions

- std::string [SoundFactory](#) ([ActionType](#) action)

Generates the file path for a sound based on the given action type.

7.61.1 Enumeration Type Documentation

7.61.1.1 ActionType

```
enum ActionType : uint32_t [strong]
```

This header file defines the ActionType enumeration and declares the SoundFactory function.

ActionType: An enumeration representing different types of actions that can trigger sounds in the game. The possible values are:

- Win: Represents a winning action.
- Shot: Represents a shooting action.
- Boss: Represents a boss-related action.
- PowerUp: Represents a power-up action.
- GameOver: Represents a game over action.
- BossDeath: Represents a boss death action.
- Explosion: Represents an explosion action.
- Background: Represents background music or sound.
- NONE: Represents no action.

SoundFactory: A function that takes an ActionType as a parameter and returns a string representing the path to the corresponding sound file.

Parameters

<i>action</i>	The ActionType for which the sound path is required.
---------------	------------------------------------------------------

Returns

A string representing the path to the sound file corresponding to the given action.

Enumerator

Win	
Shot	
Boss	
PowerUp	
GameOver	
BossDeath	
Explosion	
Background	
NONE	

7.61.2 Function Documentation

7.61.2.1 SoundFactory()

```
std::string SoundFactory (
    ActionType action )
```

Generates the file path for a sound based on the given action type.

This function takes an ActionType enumeration value and returns a corresponding file path as a string. The file path points to the sound file associated with the specified action.

Parameters

<i>action</i>	The action type for which the sound file path is needed.
---------------	----------------------------------------------------------

Returns

std::string The file path of the sound associated with the given action.

7.62 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/sprite_path.hpp File Reference

```
#include <cstdint>
#include <string>
```

Enumerations

- enum class [SpritePath](#) : uint32_t {
[Ship1](#) , [Ship2](#) , [Ship3](#) , [Ship4](#) ,
[Enemy1](#) , [Enemy2](#) , [Enemy3](#) , [Missile](#) ,
[ForceWeapon](#) , [ForceMissile](#) , [BlueLaserCrystal](#) , [Background1](#) ,
[Background2](#) , [Background3](#) , [Boss](#) , [BossBullet](#) ,
[Bar](#) , [Wall](#) }

Enum class representing various sprite paths used in the game.

Functions

- std::string [SpriteFactory](#) ([SpritePath](#) sprite)
Factory function to get the string representation of a sprite path.
- std::ostream & [operator<<](#) (std::ostream &os, const [SpritePath](#) &spritePath)
Overloaded output stream operator for SpritePath.

7.62.1 Enumeration Type Documentation

7.62.1.1 SpritePath

```
enum SpritePath : uint32_t [strong]
```

Enum class representing various sprite paths used in the game.

This enum class defines a set of constants representing different sprite paths that can be used in the game. Each constant corresponds to a specific sprite.

Enumerator

Ship1	Represents the path for the first ship sprite.
Ship2	Represents the path for the second ship sprite.
Ship3	Represents the path for the third ship sprite.
Ship4	Represents the path for the fourth ship sprite.
Enemy1	Represents the path for the first enemy sprite.
Enemy2	Represents the path for the second enemy sprite.
Enemy3	Represents the path for the third enemy sprite.
Missile	Represents the path for the missile sprite.
ForceWeapon	Represents the path for the force weapon sprite.
ForceMissile	Represents the path for the force missile sprite.
BlueLaserCrystal	Represents the path for the blue laser crystal sprite.
Background1	Represents the path for the first background sprite.
Background2	Represents the path for the second background sprite.
Background3	Represents the path for the third background sprite.
Boss	Represents the path for the boss sprite.
BossBullet	Represents the path for the boss bullet sprite.
Bar	Represents the path for the bar sprite.
Wall	Represents the path for the wall sprite.

7.62.2 Function Documentation

7.62.2.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const SpritePath & spritePath )
```

Overloaded output stream operator for SpritePath.

This operator allows the SpritePath enum value to be output to an output stream, such as std::cout.

Parameters

<i>os</i>	The output stream.
<i>spritePath</i>	The SpritePath enum value.

Returns

std::ostream& The output stream with the sprite path written to it.

7.62.2.2 SpriteFactory()

```
std::string SpriteFactory (
    SpritePath sprite )
```

Factory function to get the string representation of a sprite path.

This function takes a SpritePath enum value and returns the corresponding string representation of the sprite path.

Parameters

<i>sprite</i>	The SpritePath enum value.
---------------	----------------------------

Returns

std::string The string representation of the sprite path.

7.63 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/audio_system.hpp File Reference

```
#include <SFML/Audio.hpp>
#include <Systems/i_system.hpp>
#include <audio_manager.hpp>
#include <error_handling.hpp>
#include <memory>
#include <string>
```

Classes

- class [AudioSystem](#)
Manages audio playback within the application.

7.64 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/auto_fire_system.hpp File Reference

```
#include "Systems/i_system.hpp"
```

Classes

- class [AutoFireSystem](#)
A system that handles automatic firing mechanisms for entities.

7.65 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/collision_system.hpp File Reference

```
#include "Systems/i_system.hpp"
```

Classes

- class [CollisionSystem](#)
Manages collision detection and response within the ECS framework.

7.66 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/i_system.hpp File Reference

```
#include "Components/component_manager.hpp"  
#include "Entities/entity_manager.hpp"  
#include <SFML/Graphics.hpp>
```

Classes

- class [ISystem](#)
Interface for all systems in the ECS ([Entity](#) Component System) architecture.

7.67 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/move_system.hpp File Reference

```
#include "i_system.hpp"
```

Classes

- class [MoveSystem](#)
System responsible for moving entities within the ECS framework.

7.68 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/render_system.hpp File Reference

```
#include "Systems/i_system.hpp"  
#include <error_handling.hpp>
```


Classes

- class [RenderSystem](#)
A system responsible for rendering entities in the ECS framework.

7.69 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/systems.hpp File Reference

```
#include <Systems/audio_system.hpp>
#include <Systems/auto_fire_system.hpp>
#include <Systems/collision_system.hpp>
#include <Systems/move_system.hpp>
#include <Systems/render_system.hpp>
#include <Systems/update_system.hpp>
```

7.70 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Systems/update_system.hpp File Reference

```
#include "Systems/i_system.hpp"
```

Classes

- class [UpdateSystem](#)
A system responsible for updating sprite positions in the game.

7.71 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/texture_↵ manager.hpp File Reference

```
#include "error_handling.hpp"
#include <SFML/Graphics.hpp>
#include <string>
#include <unordered_map>
```

Classes

- class [TextureManager](#)

7.72 /home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp File Reference

```
#include <a_scenes.hpp>
```

7.73 /home/runner/work/R-Type/R-Type/ECS/Src/Entities/entity_↔ factory.cpp File Reference

```
#include "hitbox_tmp.hpp"
#include <Components/components.hpp>
#include <Entities/entity_factory.hpp>
#include <SFML/Graphics.hpp>
#include <stdint>
#include <stdlib>
#include <macros.hpp>
```

Functions

- `std::ostream & operator<< (std::ostream &os, const SpritePath &spritePath)`
Overloaded output stream operator for [SpritePath](#).
- `std::ostream & operator<< (std::ostream &os, const AScenes::SpriteType &spriteType)`
- `std::ostream & operator<< (std::ostream &os, const GameState &gameState)`
- `std::ostream & operator<< (std::ostream &os, const SpriteDataComponent &spriteData)`
Overloads the << operator to output the contents of a [SpriteDataComponent](#) to an ostream.

7.73.1 Function Documentation

7.73.1.1 `operator<<()` [1/4]

```
std::ostream& operator<< (
    std::ostream & os,
    const AScenes::SpriteType & spriteType )
```

7.73.1.2 `operator<<()` [2/4]

```
std::ostream& operator<< (
    std::ostream & os,
    const GameState & gameState )
```

7.73.1.3 `operator<<()` [3/4]

```
std::ostream& operator<< (
    std::ostream & os,
    const SpriteDataComponent & spriteData )
```

Overloads the << operator to output the contents of a [SpriteDataComponent](#) to an ostream.

Parameters

<i>os</i>	The output stream to which the SpriteDataComponent will be written.
<i>spriteData</i>	The SpriteDataComponent instance to be written to the output stream.

Returns

std::ostream& The output stream after writing the [SpriteDataComponent](#).

7.73.1.4 operator<<() [4/4]

```
std::ostream& operator<< (
    std::ostream & os,
    const SpritePath & spritePath )
```

Overloaded output stream operator for SpritePath.

This operator allows the SpritePath enum value to be output to an output stream, such as std::cout.

Parameters

<i>os</i>	The output stream.
<i>spritePath</i>	The SpritePath enum value.

Returns

std::ostream& The output stream with the sprite path written to it.

7.74 /home/runner/work/R-Type/R-Type/ECS/Src/font_path.cpp File Reference

```
#include <font_path.hpp>
```

Functions

- std::string [FontFactory](#) ([FontPath](#) font)
Creates a font object from the given font path.

7.74.1 Function Documentation

7.74.1.1 FontFactory()

```
std::string FontFactory (
    FontPath font )
```

Creates a font object from the given font path.

This function takes a FontPath object and returns a string representation of the font. The FontPath object should contain the necessary information to locate and load the font.

Parameters

<i>font</i>	The FontPath object containing the path to the font.
-------------	------------------------------------------------------

Returns

std::string The string representation of the font.

7.75 /home/runner/work/R-Type/R-Type/ECS/Src/game_text.cpp File Reference

```
#include <game_text.hpp>
```

Functions

- std::string [GameTextFactory](#) ([GameText](#) text)
Factory function to convert GameText enum to a string.

7.75.1 Function Documentation

7.75.1.1 GameTextFactory()

```
std::string GameTextFactory (
    GameText text )
```

Factory function to convert GameText enum to a string.

Parameters

<i>text</i>	The GameText enum value.
-------------	--------------------------

Returns

A string representation of the GameText value.

7.76 /home/runner/work/R-Type/R-Type/ECS/Src/hitbox_tmp.cpp File Reference

```
#include "hitbox_tmp.hpp"
#include <macros.hpp>
```

Functions

- static int [CheckCollisionLogic](#) (float descLeft, float descRight, float descTop, float descBottom, [ComponentManager](#) componentManager, [EntityManager](#) entityManager, int entityId)
- int [CheckEntityPosition](#) (uint32_t entityId, [ComponentManager](#) componentManager, [EntityManager](#) entityManager)
Checks the position of an entity within the game world.
- int [CheckEntityMovement](#) ([EntityInformation](#) desc, [ComponentManager](#) componentManager, [EntityManager](#) entityManager)
Checks the movement of an entity within the game.

7.76.1 Function Documentation

7.76.1.1 CheckCollisionLogic()

```
static int CheckCollisionLogic (
    float descLeft,
    float descRight,
    float descTop,
    float descBottom,
    ComponentManager componentManager,
    EntityManager entityManager,
    int entityId ) [static]
```

7.76.1.2 CheckEntityMovement()

```
int CheckEntityMovement (
    EntityInformation desc,
    ComponentManager componentManager,
    EntityManager entityManager )
```

Checks the movement of an entity within the game.

Parameters

<i>desc</i>	An EntityInformation object containing details about the entity.
<i>componentManager</i>	A ComponentManager object to manage the components of entities.
<i>entityManager</i>	An EntityManager object to manage the entities.

Returns

An integer indicating the result of the movement check.

7.76.1.3 CheckEntityPosition()

```
int CheckEntityPosition (
    uint32_t entityId,
    ComponentManager componentManager,
    EntityManager entityManager )
```

Checks the position of an entity within the game world.

This function retrieves and checks the position of the specified entity using the provided component and entity managers.

Parameters

<i>entityId</i>	The unique identifier of the entity whose position is to be checked.
<i>componentManager</i>	The manager responsible for handling components of entities.
<i>entityManager</i>	The manager responsible for handling entities.

Returns

An integer representing the status or result of the position check.

7.77 /home/runner/work/R-Type/R-Type/ECS/Src/sound_path.cpp File Reference

```
#include <sound_path.hpp>
```

Functions

- `std::string SoundFactory (ActionType action)`
Generates the file path for a sound based on the given action type.

7.77.1 Function Documentation

7.77.1.1 SoundFactory()

```
std::string SoundFactory (
    ActionType action )
```

Generates the file path for a sound based on the given action type.

This function takes an ActionType enumeration value and returns a corresponding file path as a string. The file path points to the sound file associated with the specified action.

Parameters

<i>action</i>	The action type for which the sound file path is needed.
---------------	----------------------------------------------------------

Returns

std::string The file path of the sound associated with the given action.

7.78 /home/runner/work/R-Type/R-Type/ECS/Src/sprite_path.cpp File Reference

```
#include <sprite_path.hpp>
```

Functions

- std::string [SpriteFactory](#) ([SpritePath](#) sprite)
Factory function to get the string representation of a sprite path.

7.78.1 Function Documentation

7.78.1.1 SpriteFactory()

```
std::string SpriteFactory (
    SpritePath sprite )
```

Factory function to get the string representation of a sprite path.

This function takes a SpritePath enum value and returns the corresponding string representation of the sprite path.

Parameters

<i>sprite</i>	The SpritePath enum value.
---------------	----------------------------

Returns

std::string The string representation of the sprite path.

7.79 [↩](#) /home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_system.cpp File Reference

```
#include <Systems/audio_system.hpp>
```

7.80 [↩](#) /home/runner/work/R-Type/R-Type/ECS/Src/Systems/auto_fire_system.cpp File Reference

```
#include <Systems/auto_fire_system.hpp>
```

7.81 [↩](#) /home/runner/work/R-Type/R-Type/ECS/Src/Systems/collision_system.cpp File Reference

```
#include <Systems/collision_system.hpp>
#include <macros.hpp>
#include <vector>
```

7.82 [↩](#) /home/runner/work/R-Type/R-Type/ECS/Src/Systems/move_system.cpp File Reference

```
#include <Systems/move_system.hpp>
#include <cmath>
```

7.83 [↩](#) /home/runner/work/R-Type/R-Type/ECS/Src/Systems/render_system.cpp File Reference

```
#include <Systems/render_system.hpp>
```


7.84 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/update_system.cpp File Reference

```
#include "Systems/update_system.hpp"
```

7.85 /home/runner/work/R-Type/R-Type/Server/Interface/Include/animation_system.hpp File Reference

```
#include "Systems/i_system.hpp"
#include <entity_struct.hpp>
```

Classes

- class [AnimationSystem](#)
A system responsible for animating entities within the ECS framework.

Enumerations

- enum class [AnimationShip](#) : uint32_t {
SHIP_DOWN , SHIP_FLIP_DOWN , SHIP_STRAIT , SHIP_FLIP_UP ,
SHIP_UP }
- enum class [AnimationBasicMonster](#) : uint32_t {
BASIC_MONSTER_DEFAULT , BASIC_MONSTER_1 , BASIC_MONSTER_2 , BASIC_MONSTER_3 ,
BASIC_MONSTER_4 , BASIC_MONSTER_5 , BASIC_MONSTER_6 , BASIC_MONSTER_7 }
- enum class [AnimationForceWeapon1](#) : uint32_t {
FORCE_WEAPON_DEFAULT , FORCE_WEAPON_1 , FORCE_WEAPON_2 , FORCE_WEAPON_3 ,
FORCE_WEAPON_4 , FORCE_WEAPON_5 }
- enum class [AnimationForceWeapon2](#) : uint32_t {
FORCE_WEAPON_DEFAULT , FORCE_WEAPON_1 , FORCE_WEAPON_2 , FORCE_WEAPON_3 ,
FORCE_WEAPON_4 , FORCE_WEAPON_5 }
- enum class [AnimationForceWeapon3](#) : uint32_t { FORCE_WEAPON_DEFAULT , FORCE_WEAPON_1 ,
FORCE_WEAPON_2 , FORCE_WEAPON_3 }
- enum class [AnimationForceMissile1](#) : uint32_t { FORCE_MISSILE_DEFAULT }
- enum class [AnimationForceMissile2](#) : uint32_t { FORCE_MISSILE_DEFAULT }
- enum class [AnimationForceMissile3](#) : uint32_t {
FORCE_MISSILE_DEFAULT , FORCE_MISSILE_1 , FORCE_MISSILE_2 , FORCE_MISSILE_3 ,
FORCE_MISSILE_4 , FORCE_MISSILE_5 , FORCE_MISSILE_6 , FORCE_MISSILE_7 }

Functions

- bool [operator!=](#) ([AnimationComponent](#) animation, [AnimationComponent](#) other)
get if two animations are different.
- [vf2d animationShipFactory](#) ([AnimationShip](#) animation)
Factory function to create a ship animation.

7.85.1 Enumeration Type Documentation

7.85.1.1 AnimationBasicMonster

```
enum AnimationBasicMonster : uint32_t [strong]
```

Enumerator

BASIC_MONSTER_DEFAULT	
BASIC_MONSTER_1	
BASIC_MONSTER_2	
BASIC_MONSTER_3	
BASIC_MONSTER_4	
BASIC_MONSTER_5	
BASIC_MONSTER_6	
BASIC_MONSTER_7	

7.85.1.2 AnimationForceMissile1

```
enum AnimationForceMissile1 : uint32_t [strong]
```

Enumerator

FORCE_MISSILE_DEFAULT	
-----------------------	--

7.85.1.3 AnimationForceMissile2

```
enum AnimationForceMissile2 : uint32_t [strong]
```

Enumerator

FORCE_MISSILE_DEFAULT	
-----------------------	--

7.85.1.4 AnimationForceMissile3

```
enum AnimationForceMissile3 : uint32_t [strong]
```

Enumerator

FORCE_MISSILE_DEFAULT	
FORCE_MISSILE_1	
FORCE_MISSILE_2	
FORCE_MISSILE_3	
FORCE_MISSILE_4	
FORCE_MISSILE_5	
FORCE_MISSILE_6	
FORCE_MISSILE_7	

7.85.1.5 AnimationForceWeapon1

```
enum AnimationForceWeapon1 : uint32_t [strong]
```

Enumerator

FORCE_WEAPON_DEFAULT	
FORCE_WEAPON_1	
FORCE_WEAPON_2	
FORCE_WEAPON_3	
FORCE_WEAPON_4	
FORCE_WEAPON_5	

7.85.1.6 AnimationForceWeapon2

```
enum AnimationForceWeapon2 : uint32_t [strong]
```

Enumerator

FORCE_WEAPON_DEFAULT	
FORCE_WEAPON_1	
FORCE_WEAPON_2	
FORCE_WEAPON_3	
FORCE_WEAPON_4	
FORCE_WEAPON_5	

7.85.1.7 AnimationForceWeapon3

```
enum AnimationForceWeapon3 : uint32_t [strong]
```

Enumerator

FORCE_WEAPON_DEFAULT	
FORCE_WEAPON_1	
FORCE_WEAPON_2	
FORCE_WEAPON_3	

7.85.1.8 AnimationShip

```
enum AnimationShip : uint32_t [strong]
```

Enumerator

SHIP_DOWN	Ship animation when going down.
SHIP_FLIP_DOWN	Ship animation when flipping down.
SHIP_STRAIT	Ship animation when going strait.
SHIP_FLIP_UP	Ship animation when flipping up.
SHIP_UP	Ship animation when going up.

7.85.2 Function Documentation

7.85.2.1 animationShipFactory()

```
vf2d animationShipFactory (
    AnimationShip animation )
```

Factory function to create a ship animation.

This function takes an AnimationShip object and generates a corresponding [vf2d](#) object that represents the animation of the ship.

Parameters

<i>animation</i>	The AnimationShip object containing the animation details.
------------------	------------------------------------------------------------

Returns

[vf2d](#) The generated animation for the ship.

Factory function to create a ship animation.

This function takes an AnimationShip enumeration value and returns a [vf2d](#) vector that corresponds to the animation state of the ship.

Parameters

<i>animation</i>	The animation state of the ship, represented by the AnimationShip enumeration.
------------------	--------------------------------------------------------------------------------

Returns

[vf2d](#) A vector representing the animation state of the ship. The x-coordinate of the vector corresponds to the frame position, and the y-coordinate is always -1 for valid states. If the animation state is not recognized, the function returns {0, 0}.

7.85.2.2 operator"!=()

```
bool operator!= (
    AnimationComponent animation,
    AnimationComponent other )
```

get if two animations are different.

Parameters

<i>animation</i>	The first animation.
<i>other</i>	The second animation.

Returns

bool true if the animations are different, false otherwise.

get if two animations are different.

This operator compares two [AnimationComponent](#) objects to determine if they are not equal. Two [AnimationComponent](#) objects are considered not equal if any of their respective offset or dimension coordinates differ.

Parameters

<i>animation</i>	The first AnimationComponent to compare.
<i>other</i>	The second AnimationComponent to compare.

Returns

true if the [AnimationComponent](#) objects are not equal, false otherwise.

7.86 /home/runner/work/R-Type/R-Type/Server/Interface/↵ Include/level.hpp File Reference

```
#include <Components/component_manager.hpp>
#include <Components/components.hpp>
#include <animation_system.hpp>
#include <cmath>
#include <game_struct.h>
#include <i_level.hpp>
```

Classes

- class [r_type::Level< T >](#)

The [Level](#) class template manages the game level, including updating game state, handling collisions, and managing entities.

Namespaces

- [r_type](#)
- [r_type::net](#)

7.87 /home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/a_server.hpp File Reference

```
#include <Components/component_manager.hpp>
#include <Components/components.hpp>
#include <Entities/entity_factory.hpp>
#include <Entities/entity_manager.hpp>
#include <Net/i_server.hpp>
#include <Systems/systems.hpp>
#include <cmath>
#include <entity_struct.hpp>
#include <error_handling.hpp>
#include <filesystem>
#include <fstream>
#include <game_struct.h>
#include <iostream>
#include <level.hpp>
#include <macros.hpp>
#include <unordered_map>
```

Classes

- class [r_type::net::AServer< T >](#)
AServer class template for managing server operations.

Namespaces

- [r_type](#)
- [r_type::net](#)

7.88 /home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/server.hpp File Reference

```
#include "a_server.hpp"
```

Classes

- class [r_type::net::Server](#)
A server class that handles client connections and messaging.

Namespaces

- [r_type](#)
- [r_type::net](#)

7.89 /home/runner/work/R-Type/R-Type/Server/Src/animation_↵ system.cpp File Reference

```
#include <Systems/systems.hpp>
#include <animation_system.hpp>
```

Functions

- `bool operator==` (const [vf2d](#) &lhs, const [vf2d](#) &rhs)
- `vf2d animationShipFactory` ([AnimationShip](#) animation)
Generates a vector representing the animation state of a ship.
- `vf2d animationBasicMonsterFactory` ([AnimationBasicMonster](#) animation)
- `static vf2d animationForceWeapon1Factory` ([AnimationForceWeapon1](#) animation)
- `static vf2d animationForceWeapon2Factory` ([AnimationForceWeapon2](#) animation)
- `static vf2d animationForceWeapon3Factory` ([AnimationForceWeapon3](#) animation)
- `static vf2d animationForceMissile1Factory` ([AnimationForceMissile1](#) animation)
- `static vf2d animationForceMissile2Factory` ([AnimationForceMissile2](#) animation)
- `static vf2d animationForceMissile3Factory` ([AnimationForceMissile3](#) animation)
- `bool operator!=` ([AnimationComponent](#) animation, [AnimationComponent](#) other)
Inequality operator for [AnimationComponent](#).
- `static void animateForceWeaponLevel1` (std::optional< [AnimationComponent](#) * > &animation)
- `static void animateForceWeaponLevel2` (std::optional< [AnimationComponent](#) * > &animation)
- `static void animateForceWeaponLevel3` (std::optional< [AnimationComponent](#) * > &animation)
- `static void animateForceMissileLevel1` (std::optional< [AnimationComponent](#) * > &animation)
- `static void animateForceMissileLevel2` (std::optional< [AnimationComponent](#) * > &animation)
- `static void animateForceMissileLevel3` (std::optional< [AnimationComponent](#) * > &animation)

7.89.1 Function Documentation

7.89.1.1 animateForceMissileLevel1()

```
static void animateForceMissileLevel1 (
    std::optional< AnimationComponent * > & animation ) [static]
```


7.89.1.2 animateForceMissileLevel2()

```
static void animateForceMissileLevel2 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.89.1.3 animateForceMissileLevel3()

```
static void animateForceMissileLevel3 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.89.1.4 animateForceWeaponLevel1()

```
static void animateForceWeaponLevel1 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.89.1.5 animateForceWeaponLevel2()

```
static void animateForceWeaponLevel2 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.89.1.6 animateForceWeaponLevel3()

```
static void animateForceWeaponLevel3 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.89.1.7 animationBasicMonsterFactory()

```
vf2d animationBasicMonsterFactory (
    AnimationBasicMonster animation )
```

7.89.1.8 animationForceMissile1Factory()

```
static vf2d animationForceMissile1Factory (
    AnimationForceMissile1 animation ) [static]
```

7.89.1.9 animationForceMissile2Factory()

```
static vf2d animationForceMissile2Factory (
    AnimationForceMissile2 animation ) [static]
```

7.89.1.10 animationForceMissile3Factory()

```
static vf2d animationForceMissile3Factory (
    AnimationForceMissile3 animation ) [static]
```

7.89.1.11 animationForceWeapon1Factory()

```
static vf2d animationForceWeapon1Factory (
    AnimationForceWeapon1 animation ) [static]
```

7.89.1.12 animationForceWeapon2Factory()

```
static vf2d animationForceWeapon2Factory (
    AnimationForceWeapon2 animation ) [static]
```

7.89.1.13 animationForceWeapon3Factory()

```
static vf2d animationForceWeapon3Factory (
    AnimationForceWeapon3 animation ) [static]
```

7.89.1.14 animationShipFactory()

```
vf2d animationShipFactory (
    AnimationShip animation )
```

Generates a vector representing the animation state of a ship.

Factory function to create a ship animation.

This function takes an AnimationShip enumeration value and returns a vf2d vector that corresponds to the animation state of the ship.

Parameters

<i>animation</i>	The animation state of the ship, represented by the AnimationShip enumeration.
------------------	--------------------------------------------------------------------------------

Returns

[vf2d](#) A vector representing the animation state of the ship. The x-coordinate of the vector corresponds to the frame position, and the y-coordinate is always -1 for valid states. If the animation state is not recognized, the function returns {0, 0}.

7.89.1.15 operator!=(())

```
bool operator!=(
    AnimationComponent animation,
    AnimationComponent other )
```

Inequality operator for [AnimationComponent](#).

get if two animations are different.

This operator compares two [AnimationComponent](#) objects to determine if they are not equal. Two [AnimationComponent](#) objects are considered not equal if any of their respective offset or dimension coordinates differ.

Parameters

<i>animation</i>	The first AnimationComponent to compare.
<i>other</i>	The second AnimationComponent to compare.

Returns

true if the [AnimationComponent](#) objects are not equal, false otherwise.

7.89.1.16 operator==(())

```
bool operator==(
    const vf2d & lhs,
    const vf2d & rhs )
```

7.90 /home/runner/work/R-Type/R-Type/Server/Src/server.cpp File Reference

```
#include <Net/server.hpp>
#include <creatable_client_object.hpp>
```


Index

/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/a_client.hpp, 185
179 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/lab
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/client.hpp, 185
180 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp, 186
180 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/mo
/home/runner/work/R-Type/R-Type/Client/Interface/Include/mainmenu.hpp, 186
179 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/off
/home/runner/work/R-Type/R-Type/Client/Interface/Include/scenes.hpp, 187
181 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/on
/home/runner/work/R-Type/R-Type/Client/Src/keyToString.cpp, 197
181 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/pla
/home/runner/work/R-Type/R-Type/Client/Src/main.cpp, 197
182 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/pla
/home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp, 198
185 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/po
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component ally_component.hpp, 198
187 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/po
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component ally_missile_component.hpp, 198
188 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/reo
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component animation_component.hpp, 198
188 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sco
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component background_component.hpp, 198
190 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sha
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component basic_monster_component.hpp, 198
190 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sha
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component bind_component.hpp, 198
191 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/spi
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component boss_component.hpp, 198
191 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/spi
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component component_manager.hpp, 198
192 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/te
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component components.hpp, 198
192 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/te
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component enemy_component.hpp, 198
193 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/vel
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component enemy_missile_component.hpp, 198
193 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wa
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component force_missile_component.hpp, 198
193 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hp
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component force_weapon_component.hpp, 198
194 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_fa
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component front_component.hpp, 198
194 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_m
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component health_component.hpp, 198
194 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/i_entity
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component hitbox_component.hpp, 198
194 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/audio_
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Component input_component.hpp, 198

- RenderSystem, 150
- UpdateSystem, 175
- _currentDaltonismMode
 - AScenes, 35
- _currentGameMode
 - AScenes, 35
- _currentMusicFilePath
 - AudioSystem, 62
- _currentScene
 - AScenes, 35
- _deqConnections
 - r_type::net::AServer< T >, 54
- _displayDaltonismChoice
 - AScenes, 36
- _displayGameModeChoice
 - AScenes, 36
- _displayKeyBindsChoice
 - AScenes, 36
- _entityFactory
 - r_type::net::AServer< T >, 54
- _entityManager
 - AnimationSystem, 26
 - AutoFireSystem, 64
 - CollisionSystem, 71
 - MoveSystem, 142
 - r_type::net::AServer< T >, 54
 - UpdateSystem, 175
- _font
 - RenderSystem, 151
- _gameParameters
 - r_type::Level< T >, 136
- _id
 - Entity, 77
- _ip
 - AScenes, 36
- _level
 - r_type::net::AServer< T >, 55
- _moveSystem
 - r_type::Level< T >, 136
- _nIDCounter
 - r_type::net::AServer< T >, 55
- _nbrOfPlayers
 - r_type::net::AServer< T >, 55
- _networkClient
 - Scenes, 156
- _playerConnected
 - r_type::net::AServer< T >, 55
- _port
 - AScenes, 36
 - r_type::net::AServer< T >, 55
- _previousScene
 - AScenes, 36
- _qMessagesIn
 - r_type::net::AServer< T >, 55
- _shooterEnemySpawnTime
 - r_type::Level< T >, 136
- _soundEffect
 - AudioSystem, 62
- _spawnTimeMonsterThree
 - r_type::Level< T >, 137
- _tempBuffer
 - r_type::net::AServer< T >, 56
- _threadContext
 - r_type::net::AServer< T >, 56
- _window
 - RenderSystem, 151
 - Scenes, 156
 - UpdateSystem, 175
- ~AClient
 - r_type::net::AClient< T >, 16
- ~AScenes
 - AScenes, 30
- ~AServer
 - r_type::net::AServer< T >, 41
- ~IClient
 - r_type::net::IClient< T >, 107
- ~IEntityFactory
 - IEntityFactory, 111
- ~IScenes
 - IScenes, 121
- ~ISystem
 - ISystem, 123
- ~Level
 - r_type::Level< T >, 127
- ~Scenes
 - Scenes, 153
- ~Server
 - r_type::net::Server, 159
- AbstractScenes, 15
- AClient
 - r_type::net::AClient< T >, 16
- Actions
 - AScenes, 28
- ActionType
 - sound_path.hpp, 211
- addComponent
 - ComponentManager, 72
- addEntity
 - r_type::net::Client, 67
- ALLY
 - AScenes, 30
- AllyComponent, 20
- AllyMissileComponent, 20
- animateBasicMonster
 - AnimationSystem, 23
- animateEntity
 - r_type::net::Client, 67
- animateForceMissile
 - AnimationSystem, 24
- animateForceMissileLevel1
 - animation_system.cpp, 232
- animateForceMissileLevel2
 - animation_system.cpp, 232
- animateForceMissileLevel3
 - animation_system.cpp, 233
- animateForceWeapon

- AnimationSystem, [24](#)
- animateForceWeaponLevel1
 - animation_system.cpp, [233](#)
- animateForceWeaponLevel2
 - animation_system.cpp, [233](#)
- animateForceWeaponLevel3
 - animation_system.cpp, [233](#)
- animatePlayer
 - AnimationSystem, [24](#)
- animation_component.hpp
 - operator!=, [189](#)
- animation_system.cpp
 - animateForceMissileLevel1, [232](#)
 - animateForceMissileLevel2, [232](#)
 - animateForceMissileLevel3, [233](#)
 - animateForceWeaponLevel1, [233](#)
 - animateForceWeaponLevel2, [233](#)
 - animateForceWeaponLevel3, [233](#)
 - animationBasicMonsterFactory, [233](#)
 - animationForceMissile1Factory, [233](#)
 - animationForceMissile2Factory, [233](#)
 - animationForceMissile3Factory, [234](#)
 - animationForceWeapon1Factory, [234](#)
 - animationForceWeapon2Factory, [234](#)
 - animationForceWeapon3Factory, [234](#)
 - animationShipFactory, [234](#)
 - operator!=, [235](#)
 - operator==, [235](#)
- animation_system.hpp
 - AnimationBasicMonster, [226](#)
 - AnimationForceMissile1, [227](#)
 - AnimationForceMissile2, [227](#)
 - AnimationForceMissile3, [227](#)
 - AnimationForceWeapon1, [228](#)
 - AnimationForceWeapon2, [228](#)
 - AnimationForceWeapon3, [228](#)
 - AnimationShip, [228](#)
 - animationShipFactory, [229](#)
 - BASIC_MONSTER_1, [227](#)
 - BASIC_MONSTER_2, [227](#)
 - BASIC_MONSTER_3, [227](#)
 - BASIC_MONSTER_4, [227](#)
 - BASIC_MONSTER_5, [227](#)
 - BASIC_MONSTER_6, [227](#)
 - BASIC_MONSTER_7, [227](#)
 - BASIC_MONSTER_DEFAULT, [227](#)
 - FORCE_MISSILE_1, [227](#)
 - FORCE_MISSILE_2, [227](#)
 - FORCE_MISSILE_3, [227](#)
 - FORCE_MISSILE_4, [227](#)
 - FORCE_MISSILE_5, [227](#)
 - FORCE_MISSILE_6, [227](#)
 - FORCE_MISSILE_7, [227](#)
 - FORCE_MISSILE_DEFAULT, [227](#)
 - FORCE_WEAPON_1, [228](#)
 - FORCE_WEAPON_2, [228](#)
 - FORCE_WEAPON_3, [228](#)
 - FORCE_WEAPON_4, [228](#)
 - FORCE_WEAPON_5, [228](#)
 - FORCE_WEAPON_DEFAULT, [228](#)
 - operator!=, [230](#)
 - SHIP_DOWN, [229](#)
 - SHIP_FLIP_DOWN, [229](#)
 - SHIP_FLIP_UP, [229](#)
 - SHIP_STRAIT, [229](#)
 - SHIP_UP, [229](#)
- AnimationBasicMonster
 - animation_system.hpp, [226](#)
- animationBasicMonsterFactory
 - animation_system.cpp, [233](#)
- AnimationComponent, [20](#)
 - AnimationComponent, [21](#)
 - dimension, [21](#)
 - offset, [21](#)
- animationComponent
 - EntityInformation, [91](#)
- AnimationEntities
 - AnimationSystem, [25](#)
- AnimationForceMissile1
 - animation_system.hpp, [227](#)
- animationForceMissile1Factory
 - animation_system.cpp, [233](#)
- AnimationForceMissile2
 - animation_system.hpp, [227](#)
- animationForceMissile2Factory
 - animation_system.cpp, [233](#)
- AnimationForceMissile3
 - animation_system.hpp, [227](#)
- animationForceMissile3Factory
 - animation_system.cpp, [234](#)
- AnimationForceWeapon1
 - animation_system.hpp, [228](#)
- animationForceWeapon1Factory
 - animation_system.cpp, [234](#)
- AnimationForceWeapon2
 - animation_system.hpp, [228](#)
- animationForceWeapon2Factory
 - animation_system.cpp, [234](#)
- AnimationForceWeapon3
 - animation_system.hpp, [228](#)
- animationForceWeapon3Factory
 - animation_system.cpp, [234](#)
- AnimationShip
 - animation_system.hpp, [228](#)
- animationShipFactory
 - animation_system.cpp, [234](#)
 - animation_system.hpp, [229](#)
- AnimationSystem, [22](#)
 - _componentManager, [25](#)
 - _entityManager, [26](#)
 - animateBasicMonster, [23](#)
 - animateForceMissile, [24](#)
 - animateForceWeapon, [24](#)
 - animatePlayer, [24](#)
 - AnimationEntities, [25](#)
 - AnimationSystem, [23](#)

AnimationUpdate
 r_type::Level< T >, 127
 AScenes, 26
 _currentDaltonismMode, 35
 _currentGameMode, 35
 _currentScene, 35
 _displayDaltonismChoice, 36
 _displayGameModeChoice, 36
 _displayKeyBindsChoice, 36
 _ip, 36
 _port, 36
 _previousScene, 36
 ~AScenes, 30
 Actions, 28
 ALLY, 30
 AScenes, 30
 BACKGROUND, 30
 buttons, 37
 DaltonismMode, 29
 DEUTERANOPIA, 29
 DOWN, 28
 EASY, 29
 ENEMY, 30
 EXIT, 30
 FILTER, 30
 filter, 37
 FIRE, 28
 GAME_LOOP, 30
 GameMode, 29
 getDaltonism, 31
 getDisplayDaltonismChoice, 31
 getDisplayGameModeChoice, 31
 getDisplayKeyBindsChoice, 31
 getIp, 32
 getPort, 32
 getPreviousScene, 32
 HARD, 29
 IN_GAME_MENU, 30
 keyBinds, 37
 LEFT, 28
 MAIN_MENU, 30
 MEDIUM, 29
 NORMAL, 29
 OTHER, 30
 PAUSE, 28
 PLAYER, 30
 POWER_UP, 30
 PROTANOPIA, 29
 QUIT, 28
 RIGHT, 28
 Scene, 29
 setDaltonism, 32
 setDisplayDaltonismChoice, 33
 setDisplayGameModeChoice, 33
 setDisplayKeyBindsChoice, 33
 setGameMode, 34
 setIp, 34
 setPort, 34
 setScene, 35
 SETTINGS_MENU, 30
 SpriteType, 30
 TRITANOPIA, 29
 UI, 30
 UP, 28
 WEAPON, 30
 AServer
 r_type::net::AServer< T >, 40
 attached
 ForceWeaponComponent, 103
 AudioManager, 56
 getSoundBuffer, 57
 soundBuffers, 57
 AudioSystem, 58
 _audioManager, 61
 _backgroundMusic, 61
 _currentMusicFilePath, 62
 _soundEffect, 62
 AudioSystem, 59
 playBackgroundMusic, 59
 playSoundEffect, 61
 stopBackgroundMusic, 61
 AutoFireSystem, 62
 _componentManager, 64
 _entityManager, 64
 AutoFireSystem, 63
 handleAutoFire, 63
 BACKGROUND
 AScenes, 30
 Background
 sound_path.hpp, 212
 Background1
 sprite_path.hpp, 214
 Background2
 sprite_path.hpp, 214
 Background3
 sprite_path.hpp, 214
 BackgroundComponent, 64
 backgroundFactory
 EntityFactory, 79
 Bar
 sprite_path.hpp, 214
 BASIC_MONSTER_1
 animation_system.hpp, 227
 BASIC_MONSTER_2
 animation_system.hpp, 227
 BASIC_MONSTER_3
 animation_system.hpp, 227
 BASIC_MONSTER_4
 animation_system.hpp, 227
 BASIC_MONSTER_5
 animation_system.hpp, 227
 BASIC_MONSTER_6
 animation_system.hpp, 227
 BASIC_MONSTER_7
 animation_system.hpp, 227
 BASIC_MONSTER_DEFAULT

- animation_system.hpp, 227
- BasicMonster
 - IEntityFactory, 111
- BasicMonsterComponent, 65
- bind
 - BindComponent, 66
- BindComponent, 65
 - bind, 66
 - BindComponent, 65
 - isHovered, 66
- BlueLaserCrystal
 - sprite_path.hpp, 214
- Boss
 - IEntityFactory, 111
 - sound_path.hpp, 212
 - sprite_path.hpp, 214
- BossBullet
 - sprite_path.hpp, 214
- BossComponent, 66
- BossDeath
 - sound_path.hpp, 212
- buttons
 - AScenes, 37
- canShoot
 - ShootComponent, 163
- categoryIds
 - TextDataComponent, 169
- categorySize
 - TextDataComponent, 169
- categoryTexts
 - TextDataComponent, 169
- charSize
 - TextDataComponent, 169
- checkCollision
 - CollisionSystem, 70
- CheckCollisionLogic
 - hitbox_tmp.cpp, 221
- CheckEntityMovement
 - hitbox_tmp.cpp, 221
 - hitbox_tmp.hpp, 209
- CheckEntityPosition
 - hitbox_tmp.cpp, 222
 - hitbox_tmp.hpp, 209
- checkOffScreen
 - CollisionSystem, 70
- CIRCLE
 - movement_component.hpp, 196
- collisionAction
 - r_type::Level< T >, 129
- CollisionSystem, 69
 - _componentManager, 71
 - _entityManager, 71
 - checkCollision, 70
 - checkOffScreen, 70
 - CollisionSystem, 69
- CollisionUpdate
 - r_type::Level< T >, 129
- ComponentManager, 71
 - addComponent, 72
 - components, 74
 - getComponent, 72
 - getComponentMap, 73
 - removeAllComponents, 73
 - removeEntityFromAllComponents, 73
 - removeEntityFromComponent, 74
- componentNotFound, 75
 - what, 75
- components
 - ComponentManager, 74
- Connect
 - r_type::net::AClient< T >, 17
 - r_type::net::IClient< T >, 107
- cooldownTime
 - ShootComponent, 163
- creatable_client_object.hpp
 - CreatableClientObject, 202
 - NONE, 202
 - PLAYERMISSILE, 202
- CreatableClientObject
 - creatable_client_object.hpp, 202
- createBackgroundLevelOne
 - EntityFactory, 79
 - IEntityFactory, 111
- createBackgroundLevelThree
 - EntityFactory, 79
 - IEntityFactory, 112
- createBackgroundLevelTwo
 - EntityFactory, 81
 - IEntityFactory, 112
- createBackgroundMenu
 - EntityFactory, 81
 - IEntityFactory, 112
- createBasicMonster
 - EntityFactory, 82
 - IEntityFactory, 113
- createButton
 - EntityFactory, 82
 - IEntityFactory, 113
- createDaltonismChoiceButtons
 - scenes.cpp, 185
- createEnemyMissile
 - EntityFactory, 83
 - IEntityFactory, 114
- createEntity
 - EntityManager, 92
- createFilter
 - EntityFactory, 83
- createForceMissile
 - EntityFactory, 85
 - IEntityFactory, 114
- createForceWeapon
 - EntityFactory, 85
 - IEntityFactory, 115
- createInfoBar
 - EntityFactory, 86
 - IEntityFactory, 115

- createKeyBindingButtons
 - scenes.cpp, 186
- createPlayer
 - EntityFactory, 86
 - IEntityFactory, 116
- createPlayerMissile
 - EntityFactory, 87
 - IEntityFactory, 116
- createPowerUpBlueLaserCrystal
 - EntityFactory, 87
 - IEntityFactory, 117
- createShooterEnemy
 - EntityFactory, 88
 - IEntityFactory, 117
- createSmallButton
 - EntityFactory, 88
 - IEntityFactory, 118
- createWall
 - EntityFactory, 90
 - IEntityFactory, 118
- DaltonismMode
 - AScenes, 29
- DEUTERANOPIA
 - AScenes, 29
- DIAGONAL
 - movement_component.hpp, 196
- difficultyChoices
 - IScenes, 121
 - Scenes, 153
- dimension
 - AnimationComponent, 21
- Disconnect
 - r_type::net::AClient< T >, 17
 - r_type::net::IClient< T >, 107
- DOWN
 - AScenes, 28
 - input_component.hpp, 195
- EASY
 - AScenes, 29
- ENEMY
 - AScenes, 30
- Enemy1
 - sprite_path.hpp, 214
- Enemy2
 - sprite_path.hpp, 214
- Enemy3
 - sprite_path.hpp, 214
- EnemyComponent, 75
- EnemyMissileComponent, 76
- EnemyType
 - IEntityFactory, 110
- entities
 - EntityManager, 94
- Entity, 76
 - _id, 77
 - Entity, 76
 - getId, 77
- entity_factory.cpp
 - operator<<, 218, 219
- EntityFactory, 77
 - backgroundFactory, 79
 - createBackgroundLevelOne, 79
 - createBackgroundLevelThree, 79
 - createBackgroundLevelTwo, 81
 - createBackgroundMenu, 81
 - createBasicMonster, 82
 - createButton, 82
 - createEnemyMissile, 83
 - createFilter, 83
 - createForceMissile, 85
 - createForceWeapon, 85
 - createInfoBar, 86
 - createPlayer, 86
 - createPlayerMissile, 87
 - createPowerUpBlueLaserCrystal, 87
 - createShooterEnemy, 88
 - createSmallButton, 88
 - createWall, 90
- EntityInformation, 91
 - animationComponent, 91
 - ratio, 91
 - spriteData, 91
 - uniqueID, 91
 - vPos, 91
- EntityManager, 92
 - createEntity, 92
 - entities, 94
 - entityNb, 94
 - getAllEntities, 93
 - getEntity, 93
 - removeAllEntities, 93
 - removeEntity, 94
- entityNb
 - EntityManager, 94
- entityNotFound, 95
 - what, 95
- EXIT
 - AScenes, 30
- Explosion
 - sound_path.hpp, 212
- failedToCreateFile, 95
 - what, 96
- failedToLoadFont, 96
 - what, 97
- failedToLoadSound, 97
 - what, 97
- failedToLoadTexture, 98
 - what, 98
- failedToOpenFile, 98
 - what, 99
- FILTER
 - AScenes, 30
- filter
 - AScenes, 37
- FIRE

- AScenes, 28
- FireUpdate
 - r_type::Level< T >, 130
- font_path.cpp
 - FontFactory, 219
- font_path.hpp
 - FontFactory, 206
 - FontPath, 205
 - MAIN, 206
 - NONE, 206
 - operator<<, 207
- FontFactory
 - font_path.cpp, 219
 - font_path.hpp, 206
- FontManager, 99
 - fonts, 101
 - getFont, 100
 - releaseFont, 100
- FontPath
 - font_path.hpp, 205
- fontPath
 - TextDataComponent, 170
- fonts
 - FontManager, 101
- FORCE_MISSILE_1
 - animation_system.hpp, 227
- FORCE_MISSILE_2
 - animation_system.hpp, 227
- FORCE_MISSILE_3
 - animation_system.hpp, 227
- FORCE_MISSILE_4
 - animation_system.hpp, 227
- FORCE_MISSILE_5
 - animation_system.hpp, 227
- FORCE_MISSILE_6
 - animation_system.hpp, 227
- FORCE_MISSILE_7
 - animation_system.hpp, 227
- FORCE_MISSILE_DEFAULT
 - animation_system.hpp, 227
- FORCE_WEAPON_1
 - animation_system.hpp, 228
- FORCE_WEAPON_2
 - animation_system.hpp, 228
- FORCE_WEAPON_3
 - animation_system.hpp, 228
- FORCE_WEAPON_4
 - animation_system.hpp, 228
- FORCE_WEAPON_5
 - animation_system.hpp, 228
- FORCE_WEAPON_DEFAULT
 - animation_system.hpp, 228
- forceld
 - ForceMissileComponent, 101
- ForceMissile
 - sprite_path.hpp, 214
- ForceMissileComponent, 101
 - forceld, 101
- ForceWeapon
 - sprite_path.hpp, 214
- ForceWeaponComponent, 102
 - attached, 103
 - ForceWeaponComponent, 102
 - level, 103
 - playerId, 103
- FormatEntityInformation
 - r_type::net::AServer< T >, 41
- FrontComponent, 103
 - FrontComponent, 104
 - targetId, 104
- GAME_LOOP
 - AScenes, 30
- game_text.cpp
 - GameTextFactory, 220
- game_text.hpp
 - GameText, 207
 - GameTextFactory, 208
 - Lives, 208
 - NONE, 208
 - operator<<, 208
 - Score, 208
- gameLoop
 - IScenes, 121
 - Scenes, 153
- GameMode
 - AScenes, 29
- GameOver
 - sound_path.hpp, 212
- GameText
 - game_text.hpp, 207
- GameTextFactory
 - game_text.cpp, 220
 - game_text.hpp, 208
- getAllEntities
 - EntityManager, 93
- getClientById
 - r_type::net::AServer< T >, 41
- GetClientInfoBarId
 - r_type::net::AServer< T >, 42
- GetClientPlayerId
 - r_type::net::AServer< T >, 42
- GetClock
 - r_type::net::AServer< T >, 42
- getComponent
 - ComponentManager, 72
- GetComponentManager
 - r_type::net::AServer< T >, 43
- getComponentMap
 - ComponentManager, 73
- getConnection
 - r_type::net::AClient< T >, 17
- getDaltonism
 - AScenes, 31
- getDisplayDaltonismChoice
 - AScenes, 31
- getDisplayGameModeChoice

- AScenes, 31
- getDisplayKeyBindsChoice
 - AScenes, 31
- getEntity
 - EntityManager, 93
- GetEntityBackGround
 - r_type::Level< T >, 130
- GetEntityFactory
 - r_type::net::AServer< T >, 43
- GetEntityManager
 - r_type::net::AServer< T >, 43
- getFont
 - FontManager, 100
- getId
 - Entity, 77
- getIp
 - AScenes, 32
- GetPlayerClientId
 - r_type::net::AServer< T >, 44
- getPlayerId
 - r_type::net::AClient< T >, 17
- getPort
 - AScenes, 32
- getPreviousScene
 - AScenes, 32
- getRenderWindow
 - IScenes, 121
 - Scenes, 153
- getSoundBuffer
 - AudioManager, 57
- getTexture
 - TextureManager, 170
- getWindowSize
 - r_type::net::AClient< T >, 18
- h
 - HitboxComponent, 106
- handleAutoFire
 - AutoFireSystem, 63
- handleEvents
 - scenes.cpp, 186
- HandleMessage
 - Scenes, 153
- HARD
 - AScenes, 29
- health
 - HealthComponent, 105
- HealthComponent, 104
 - health, 105
 - max_health, 105
- hitbox_tmp.cpp
 - CheckCollisionLogic, 221
 - CheckEntityMovement, 221
 - CheckEntityPosition, 222
- hitbox_tmp.hpp
 - CheckEntityMovement, 209
 - CheckEntityPosition, 209
- HitboxComponent, 105
 - h, 106
 - w, 106
- hitboxX
 - SpriteComponent, 165
- hitboxY
 - SpriteComponent, 165
- IClient
 - r_type::net::IClient< T >, 107
- IEntityFactory, 109
 - ~IEntityFactory, 111
 - BasicMonster, 111
 - Boss, 111
 - createBackgroundLevelOne, 111
 - createBackgroundLevelThree, 112
 - createBackgroundLevelTwo, 112
 - createBackgroundMenu, 112
 - createBasicMonster, 113
 - createButton, 113
 - createEnemyMissile, 114
 - createForceMissile, 114
 - createForceWeapon, 115
 - createInfoBar, 115
 - createPlayer, 116
 - createPlayerMissile, 116
 - createPowerUpBlueLaserCrystal, 117
 - createShooterEnemy, 117
 - createSmallButton, 118
 - createWall, 118
 - EnemyType, 110
 - ShooterEnemy, 111
 - Wall, 111
- IN_GAME_MENU
 - AScenes, 30
- Incoming
 - r_type::net::AClient< T >, 18
 - r_type::net::IClient< T >, 108
- index
 - MovementComponent, 140
- inGameMenu
 - IScenes, 121
 - Scenes, 154
- InitiateBackground
 - r_type::Level< T >, 131
- InitiateEnemyMissile
 - r_type::net::AServer< T >, 44
- InitiatePlayer
 - r_type::net::AServer< T >, 44
- InitiatePlayerMissile
 - r_type::net::AServer< T >, 45
- InitiateWeaponForce
 - r_type::net::AServer< T >, 45
- InitInfoBar
 - r_type::net::AServer< T >, 45
- initInfoBar
 - r_type::net::Client, 67
- input
 - InputComponent, 119
- input_component.hpp
 - DOWN, 195

- InputType, 195
- LEFT, 195
- NONE, 195
- QUIT, 195
- RIGHT, 195
- SHOOT, 195
- UP, 195
- InputComponent, 119
 - input, 119
- InputType
 - input_component.hpp, 195
- IScenes, 120
 - ~IScenes, 121
 - difficultyChoices, 121
 - gameLoop, 121
 - getRenderWindow, 121
 - inGameMenu, 121
 - mainMenu, 122
 - render, 122
 - settingsMenu, 122
 - shouldQuit, 122
- isClicked
 - OnClickComponent, 144
- IsConnected
 - r_type::net::AClient< T >, 18
 - r_type::net::IClient< T >, 108
- isHovered
 - BindComponent, 66
- isValidIPv4
 - main.cpp, 182
- isValidPort
 - main.cpp, 182, 183
- ISystem, 123
 - ~ISystem, 123
 - ISystem, 123
- keyBinds
 - AScenes, 37
- keyToString
 - keyToString.cpp, 181
 - scenes.hpp, 181
- keyToString.cpp
 - keyToString, 181
- labelComponent, 124
 - name, 124
 - x, 124
 - y, 124
- LEFT
 - AScenes, 28
 - input_component.hpp, 195
- Level
 - r_type::Level< T >, 127
- level
 - ForceWeaponComponent, 103
- LevelOne
 - r_type::Level< T >, 131
- LevelThree
 - r_type::Level< T >, 132
- LevelTwo
 - r_type::Level< T >, 132
- LinkForceComponent, 137
 - LinkForceComponent, 138
 - targetId, 138
- Lives
 - game_text.hpp, 208
- lives
 - UIEntityInformation, 172
- loopRunning
 - main.cpp, 184
- m_connection
 - r_type::net::AClient< T >, 19
- m_context
 - r_type::net::AClient< T >, 19
- m_qMessagesIn
 - r_type::net::AClient< T >, 19
- macros.hpp
 - SCREEN_HEIGHT, 211
 - SCREEN_WIDTH, 211
- MAIN
 - font_path.hpp, 206
- main
 - main.cpp, 182, 184
- main.cpp
 - isValidIPv4, 182
 - isValidPort, 182, 183
 - loopRunning, 184
 - main, 182, 184
 - signal_handler, 184
- MAIN_MENU
 - AScenes, 30
- MainMenu
 - mainmenu.hpp, 179
- mainMenu
 - IScenes, 122
 - Scenes, 154
- mainmenu.hpp
 - MainMenu, 179
- max_health
 - HealthComponent, 105
- MEDIUM
 - AScenes, 29
- MessageAll
 - r_type::net::Client, 67
- MessageAllClients
 - r_type::net::AServer< T >, 46
- MessageClient
 - r_type::net::AServer< T >, 46
- Missile
 - sprite_path.hpp, 214
- move
 - MovementComponent, 140
- moveEntities
 - MoveSystem, 141
- moveEntity
 - MoveSystem, 142
 - r_type::net::Client, 68

- movement_component.hpp
 - CIRCLE, 196
 - DIAGONAL, 196
 - MovementType, 196
 - STRAIGHT, 196
 - WIGGLE, 196
- MovementComponent, 138
 - index, 140
 - move, 140
 - MovementComponent, 139
 - movementType, 140
- MovementType
 - movement_component.hpp, 196
- movementType
 - MovementComponent, 140
- MoveSystem, 140
 - _componentManager, 142
 - _entityManager, 142
 - moveEntities, 141
 - moveEntity, 142
 - MoveSystem, 141
- MoveUpdate
 - r_type::Level< T >, 132
- name
 - labelComponent, 124
- nextShootTime
 - ShootComponent, 164
- NONE
 - creatable_client_object.hpp, 202
 - font_path.hpp, 206
 - game_text.hpp, 208
 - input_component.hpp, 195
 - sound_path.hpp, 212
- NORMAL
 - AScenes, 29
- offset
 - AnimationComponent, 21
 - OffsetComponent, 143
- OffsetComponent, 143
 - offset, 143
- onClick
 - OnClickComponent, 145
- OnClickComponent, 144
 - isClicked, 144
 - onClick, 145
 - OnClickComponent, 144
- OnClientConnect
 - r_type::net::AServer< T >, 47
 - r_type::net::Server, 159
- OnClientDisconnect
 - r_type::net::AServer< T >, 47
 - r_type::net::Server, 160
- OnClientValidated
 - r_type::net::AServer< T >, 47
- OnMessage
 - r_type::net::AServer< T >, 48
 - r_type::net::Server, 160
- operator!=
 - animation_component.hpp, 189
 - animation_system.cpp, 235
 - animation_system.hpp, 230
- operator<<
 - entity_factory.cpp, 218, 219
 - font_path.hpp, 207
 - game_text.hpp, 208
 - sprite_data_component.hpp, 200
 - sprite_path.hpp, 214
- operator==
 - animation_system.cpp, 235
- OTHER
 - AScenes, 30
- PAUSE
 - AScenes, 28
- PingServer
 - r_type::net::Client, 68
- playBackgroundMusic
 - AudioSystem, 59
- PLAYER
 - AScenes, 30
- PlayerComponent, 145
- playerId
 - ForceWeaponComponent, 103
 - PlayerMissileComponent, 146
 - r_type::net::AClient< T >, 19
- playerIdNotFound, 145
 - what, 146
- PLAYERMISSILE
 - creatable_client_object.hpp, 202
- PlayerMissileComponent, 146
 - playerId, 146
- playSoundEffect
 - AudioSystem, 61
- PositionComponent, 147
 - PositionComponent, 147
 - x, 147
 - y, 148
- POWER_UP
 - AScenes, 30
- PowerUp
 - sound_path.hpp, 212
- PowerUpComponent, 148
- PROTANOPIA
 - AScenes, 29
- QUIT
 - AScenes, 28
 - input_component.hpp, 195
- r_type, 13
- r_type::Level< T >, 125
 - _WallSpawnTime, 137
 - _animationSystem, 135
 - _autoFireSystem, 135
 - _basicMonsterSpawnTime, 135
 - _collisionSystem, 136

- `_gameParameters`, 136
- `_moveSystem`, 136
- `_shooterEnemySpawnTime`, 136
- `_spawnTimeMonsterThree`, 137
- `~Level`, 127
- `AnimationUpdate`, 127
- `collisionAction`, 129
- `CollisionUpdate`, 129
- `FireUpdate`, 130
- `GetEntityBackGround`, 130
- `InitiateBackground`, 131
- `Level`, 127
- `LevelOne`, 131
- `LevelThree`, 132
- `LevelTwo`, 132
- `MoveUpdate`, 132
- `SetGameParameters`, 133
- `SetSystem`, 133
- `SpawnEntity`, 134
- `Update`, 134
- `r_type::net`, 13
- `r_type::net::AClient< T >`, 15
 - `~AClient`, 16
 - `AClient`, 16
 - `Connect`, 17
 - `Disconnect`, 17
 - `getConnection`, 17
 - `getPlayerId`, 17
 - `getWindowSize`, 18
 - `Incoming`, 18
 - `IsConnected`, 18
 - `m_connection`, 19
 - `m_context`, 19
 - `m_qMessagesIn`, 19
 - `playerId`, 19
 - `Send`, 18
 - `setPlayerId`, 19
 - `setWindowSize`, 19
 - `thrContext`, 20
 - `windowSize`, 20
- `r_type::net::AServer< T >`, 38
 - `_asioContext`, 52
 - `_asioSocket`, 52
 - `_background`, 53
 - `_clientEndpoint`, 53
 - `_clientInfoBarID`, 53
 - `_clientPlayerID`, 53
 - `_clock`, 53
 - `_componentManager`, 54
 - `_deqConnections`, 54
 - `_entityFactory`, 54
 - `_entityManager`, 54
 - `_level`, 55
 - `_nIDCounter`, 55
 - `_nbrOfPlayers`, 55
 - `_playerConnected`, 55
 - `_port`, 55
 - `_qMessagesIn`, 55
 - `_tempBuffer`, 56
 - `_threadContext`, 56
 - `~AServer`, 41
 - `AServer`, 40
 - `FormatEntityInformation`, 41
 - `getClientById`, 41
 - `getClientInfoBarId`, 42
 - `getClientPlayerId`, 42
 - `GetClock`, 42
 - `GetComponentManager`, 43
 - `GetEntityFactory`, 43
 - `GetEntityManager`, 43
 - `GetPlayerClientId`, 44
 - `InitiateEnemyMissile`, 44
 - `InitiatePlayer`, 44
 - `InitiatePlayerMissile`, 45
 - `InitiateWeaponForce`, 45
 - `InitInfoBar`, 45
 - `MessageAllClients`, 46
 - `MessageClient`, 46
 - `OnClientConnect`, 47
 - `OnClientDisconnect`, 47
 - `OnClientValidated`, 47
 - `OnMessage`, 48
 - `RemoveEntity`, 48
 - `RemoveInfoBar`, 48
 - `RemovePlayer`, 49
 - `SavePlayerScore`, 49
 - `SetClock`, 50
 - `Start`, 50
 - `Stop`, 50
 - `Update`, 50
 - `UpdateInfoBar`, 51
 - `UpdatePlayerPosition`, 51
 - `WaitForClientMessage`, 52
- `r_type::net::Client`, 66
 - `addEntity`, 67
 - `animateEntity`, 67
 - `initInfoBar`, 67
 - `MessageAll`, 67
 - `moveEntity`, 68
 - `PingServer`, 68
 - `removeEntity`, 68
 - `updateInfoBar`, 68
- `r_type::net::IClient< T >`, 106
 - `~IClient`, 107
 - `Connect`, 107
 - `Disconnect`, 107
 - `IClient`, 107
 - `Incoming`, 108
 - `IsConnected`, 108
 - `Send`, 108
- `r_type::net::Server`, 158
 - `~Server`, 159
 - `OnClientConnect`, 159
 - `OnClientDisconnect`, 160
 - `OnMessage`, 160
 - `Server`, 159

- ratio
 - EntityInformation, 91
- rectangleShape
 - RectangleShapeComponent, 149
- RectangleShapeComponent, 148
 - rectangleShape, 149
 - RectangleShapeComponent, 148
- releaseFont
 - FontManager, 100
- releaseTexture
 - TextureManager, 171
- reloadFilter
 - scenes.cpp, 186
- removeAllComponents
 - ComponentManager, 73
- removeAllEntities
 - EntityManager, 93
- RemoveEntity
 - r_type::net::AServer< T >, 48
- removeEntity
 - EntityManager, 94
 - r_type::net::Client, 68
- removeEntityFromAllComponents
 - ComponentManager, 73
- removeEntityFromComponent
 - ComponentManager, 74
- RemoveInfoBar
 - r_type::net::AServer< T >, 48
- RemovePlayer
 - r_type::net::AServer< T >, 49
- render
 - IScenes, 122
 - RenderSystem, 150
 - Scenes, 155
- RenderSystem, 149
 - _componentManager, 150
 - _font, 151
 - _window, 151
 - render, 150
 - RenderSystem, 150
- RIGHT
 - AScenes, 28
 - input_component.hpp, 195
- run
 - Scenes, 155
- SavePlayerScore
 - r_type::net::AServer< T >, 49
- scale
 - SpriteDataComponent, 166
- Scene
 - AScenes, 29
- Scenes, 151
 - _networkClient, 156
 - _window, 156
 - ~Scenes, 153
 - difficultyChoices, 153
 - gameLoop, 153
 - getRenderWindow, 153
 - HandleMessage, 153
 - inGameMenu, 154
 - mainMenu, 154
 - render, 155
 - run, 155
 - Scenes, 152
 - settingsMenu, 155
 - shouldQuit, 156
 - StopGameLoop, 156
- scenes.cpp
 - createDaltonismChoiceButtons, 185
 - createKeyBindingButtons, 186
 - handleEvents, 186
 - reloadFilter, 186
 - waitForKey, 187
- scenes.hpp
 - keyToString, 181
- Score
 - game_text.hpp, 208
- score
 - ScoreComponent, 157
 - UIEntityInformation, 172
- ScoreComponent, 157
 - score, 157
- SCREEN_HEIGHT
 - macros.hpp, 211
- SCREEN_WIDTH
 - macros.hpp, 211
- Send
 - r_type::net::AClient< T >, 18
 - r_type::net::IClient< T >, 108
- Server
 - r_type::net::Server, 159
- SetClock
 - r_type::net::AServer< T >, 50
- setDaltonism
 - AScenes, 32
- setDisplayDaltonismChoice
 - AScenes, 33
- setDisplayGameModeChoice
 - AScenes, 33
- setDisplayKeyBindsChoice
 - AScenes, 33
- setGameMode
 - AScenes, 34
- SetGameParameters
 - r_type::Level< T >, 133
- setIp
 - AScenes, 34
- setPlayerId
 - r_type::net::AClient< T >, 19
- setPort
 - AScenes, 34
- setScene
 - AScenes, 35
- SetSystem
 - r_type::Level< T >, 133
- SETTINGS_MENU

- AScenes, [30](#)
- settingsMenu
 - IScenes, [122](#)
 - Scenes, [155](#)
- setWindowSize
 - r_type::net::AClient< T >, [19](#)
- shader
 - ShaderComponent, [162](#)
- ShaderComponent, [161](#)
 - shader, [162](#)
 - ShaderComponent, [162](#)
- Ship1
 - sprite_path.hpp, [214](#)
- Ship2
 - sprite_path.hpp, [214](#)
- Ship3
 - sprite_path.hpp, [214](#)
- Ship4
 - sprite_path.hpp, [214](#)
- SHIP_DOWN
 - animation_system.hpp, [229](#)
- SHIP_FLIP_DOWN
 - animation_system.hpp, [229](#)
- SHIP_FLIP_UP
 - animation_system.hpp, [229](#)
- SHIP_STRAIT
 - animation_system.hpp, [229](#)
- SHIP_UP
 - animation_system.hpp, [229](#)
- SHOOT
 - input_component.hpp, [195](#)
- ShootComponent, [162](#)
 - canShoot, [163](#)
 - cooldownTime, [163](#)
 - nextShootTime, [164](#)
 - ShootComponent, [163](#)
- ShooterEnemy
 - IEntityFactory, [111](#)
- Shot
 - sound_path.hpp, [212](#)
- shouldQuit
 - IScenes, [122](#)
 - Scenes, [156](#)
- signal_handler
 - main.cpp, [184](#)
- sound_path.cpp
 - SoundFactory, [223](#)
- sound_path.hpp
 - ActionType, [211](#)
 - Background, [212](#)
 - Boss, [212](#)
 - BossDeath, [212](#)
 - Explosion, [212](#)
 - GameOver, [212](#)
 - NONE, [212](#)
 - PowerUp, [212](#)
 - Shot, [212](#)
 - SoundFactory, [212](#)
 - Win, [212](#)
- soundBuffers
 - AudioManager, [57](#)
- SoundFactory
 - sound_path.cpp, [223](#)
 - sound_path.hpp, [212](#)
- SpawnEntity
 - r_type::Level< T >, [134](#)
- sprite
 - SpriteComponent, [165](#)
- sprite_data_component.hpp
 - operator<<, [200](#)
- sprite_path.cpp
 - SpriteFactory, [223](#)
- sprite_path.hpp
 - Background1, [214](#)
 - Background2, [214](#)
 - Background3, [214](#)
 - Bar, [214](#)
 - BlueLaserCrystal, [214](#)
 - Boss, [214](#)
 - BossBullet, [214](#)
 - Enemy1, [214](#)
 - Enemy2, [214](#)
 - Enemy3, [214](#)
 - ForceMissile, [214](#)
 - ForceWeapon, [214](#)
 - Missile, [214](#)
 - operator<<, [214](#)
 - Ship1, [214](#)
 - Ship2, [214](#)
 - Ship3, [214](#)
 - Ship4, [214](#)
 - SpriteFactory, [214](#)
 - SpritePath, [213](#)
 - Wall, [214](#)
- SpriteComponent, [164](#)
 - hitboxX, [165](#)
 - hitboxY, [165](#)
 - sprite, [165](#)
 - SpriteComponent, [165](#)
 - type, [166](#)
- spriteData
 - EntityInformation, [91](#)
 - UIEntityInformation, [172](#)
- SpriteDataComponent, [166](#)
 - scale, [166](#)
 - spritePath, [167](#)
 - type, [167](#)
- SpriteFactory
 - sprite_path.cpp, [223](#)
 - sprite_path.hpp, [214](#)
- SpritePath
 - sprite_path.hpp, [213](#)
- spritePath
 - SpriteDataComponent, [167](#)
- SpriteType
 - AScenes, [30](#)

- Start
 - `r_type::net::AServer< T >`, 50
- Stop
 - `r_type::net::AServer< T >`, 50
- stopBackgroundMusic
 - AudioSystem, 61
- StopGameLoop
 - Scenes, 156
- STRAIGHT
 - `movement_component.hpp`, 196
- targetId
 - FrontComponent, 104
 - LinkForceComponent, 138
- text
 - TextComponent, 168
- TextComponent, 167
 - text, 168
 - TextComponent, 168
- textData
 - UIEntityInformation, 173
- TextDataComponent, 168
 - categoryIds, 169
 - categorySize, 169
 - categoryTexts, 169
 - charSize, 169
 - fontPath, 170
- TextureManager, 170
 - getTexture, 170
 - releaseTexture, 171
 - textures, 171
- textures
 - TextureManager, 171
- thrContext
 - `r_type::net::AClient< T >`, 20
- TRITANOPIA
 - AScenes, 29
- type
 - SpriteComponent, 166
 - SpriteDataComponent, 167
- UI
 - AScenes, 30
- UIEntityInformation, 172
 - lives, 172
 - score, 172
 - spriteData, 172
 - textData, 173
 - uniqueId, 173
- uniqueId
 - EntityInformation, 91
 - UIEntityInformation, 173
- UP
 - AScenes, 28
 - `input_component.hpp`, 195
- Update
 - `r_type::Level< T >`, 134
 - `r_type::net::AServer< T >`, 50
- UpdateInfoBar
 - `r_type::net::AServer< T >`, 51
- updateInfoBar
 - `r_type::net::Client`, 68
- UpdatePlayerPosition
 - `r_type::net::AServer< T >`, 51
- updateSpritePositions
 - UpdateSystem, 174
- UpdateSystem, 173
 - _componentManager, 175
 - _entityManager, 175
 - _window, 175
 - updateSpritePositions, 174
 - UpdateSystem, 174
- VelocityComponent, 176
 - x, 176
 - y, 176
- vf2d, 176
 - x, 177
 - y, 177
- vPos
 - EntityInformation, 91
- w
 - HitboxComponent, 106
- WaitForClientMessage
 - `r_type::net::AServer< T >`, 52
- waitForKey
 - `scenes.cpp`, 187
- Wall
 - IEntityFactory, 111
 - `sprite_path.hpp`, 214
- WallComponent, 177
- WEAPON
 - AScenes, 30
- what
 - componentNotFound, 75
 - entityNotFound, 95
 - failedToCreateFile, 96
 - failedToLoadFont, 97
 - failedToLoadSound, 97
 - failedToLoadTexture, 98
 - failedToOpenFile, 99
 - playerIdNotFound, 146
- WIGGLE
 - `movement_component.hpp`, 196
- Win
 - `sound_path.hpp`, 212
- windowSize
 - `r_type::net::AClient< T >`, 20
- x
 - labelComponent, 124
 - PositionComponent, 147
 - VelocityComponent, 176
 - vf2d, 177
- y
 - labelComponent, 124

PositionComponent, [148](#)

VelocityComponent, [176](#)

vf2d, [177](#)