

R-Type

Generated by Doxygen 1.9.8

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()	17
6.2.2.5 getWindowSize()	17
6.2.2.6 Incoming()	18
6.2.2.7 isConnected()	18
6.2.2.8 Send()	18
6.2.2.9 setPlayerId()	18
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	19
6.2.3.5 thrContext	19
6.2.3.6 windowSize	19
6.3 AllyComponent Struct Reference	20
6.4 AllyMissileComponent Struct Reference	20
6.5 AnimationComponent Struct Reference	20

6.5.1 Detailed Description	20
6.5.2 Constructor & Destructor Documentation	20
6.5.2.1 AnimationComponent()	20
6.5.3 Member Data Documentation	21
6.5.3.1 dimension	21
6.5.3.2 offset	21
6.6 AnimationSystem Class Reference	21
6.6.1 Detailed Description	22
6.6.2 Constructor & Destructor Documentation	22
6.6.2.1 AnimationSystem()	22
6.6.3 Member Function Documentation	22
6.6.3.1 animateBasicMonster()	22
6.6.3.2 animateBoss()	23
6.6.3.3 animateForceMissile()	23
6.6.3.4 animateForceWeapon()	23
6.6.3.5 animatePlayer()	24
6.6.3.6 AnimationEntities()	24
6.6.4 Member Data Documentation	25
6.6.4.1 _componentManager	25
6.6.4.2 _entityManager	25
6.7 AScenes Class Reference	25
6.7.1 Member Enumeration Documentation	28
6.7.1.1 Actions	28
6.7.1.2 DaltonismMode	28
6.7.1.3 GameMode	29
6.7.1.4 Scene	29
6.7.1.5 SpriteType	29
6.7.2 Constructor & Destructor Documentation	30
6.7.2.1 AScenes()	30
6.7.2.2 ~AScenes()	30
6.7.3 Member Function Documentation	30
6.7.3.1 getDaltonism()	30
6.7.3.2 getDisplayDaltonismChoice()	31
6.7.3.3 getDisplayGameModeChoice()	31
6.7.3.4 getDisplayKeyBindsChoice()	31
6.7.3.5 getGameMode()	31
6.7.3.6 getIp()	32
6.7.3.7 GetPlayerReady()	32
6.7.3.8 getPort()	32
6.7.3.9 getPreviousScene()	32
6.7.3.10 setDaltonism()	32
6.7.3.11 setDisplayDaltonismChoice()	33

6.7.3.12 setDisplayGameModeChoice()	33
6.7.3.13 setDisplayKeyBindsChoice()	33
6.7.3.14 setGameMode()	34
6.7.3.15 setIp()	34
6.7.3.16 SetPlayerReady()	34
6.7.3.17 setPort()	34
6.7.3.18 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	35
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 _playerReady	36
6.7.4.9 _port	36
6.7.4.10 _previousScene	36
6.7.4.11 buttons	37
6.7.4.12 filter	37
6.7.4.13 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()	41
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()	43
6.8.3.9 GetPlayerClientId()	43
6.8.3.10 InitBoss()	44
6.8.3.11 InitiateEnemyMissile()	44
6.8.3.12 InitiatePlayer()	44
6.8.3.13 InitiatePlayerMissile()	44
6.8.3.14 InitiateWeaponForce()	45
6.8.3.15 InitInfoBar()	45

6.8.3.16 MessageAllClients()	45
6.8.3.17 MessageClient()	46
6.8.3.18 OnClientConnect()	46
6.8.3.19 OnClientDisconnect()	46
6.8.3.20 OnClientValidated()	47
6.8.3.21 OnMessage()	47
6.8.3.22 RemoveBossTail()	47
6.8.3.23 RemoveEntity()	47
6.8.3.24 RemoveInfoBar()	48
6.8.3.25 RemovePlayer()	48
6.8.3.26 SavePlayerScore()	48
6.8.3.27 SetClock()	49
6.8.3.28 Start()	49
6.8.3.29 Stop()	49
6.8.3.30 Update()	50
6.8.3.31 UpdateInfoBar()	50
6.8.3.32 UpdatePlayerPosition()	50
6.8.3.33 WaitForClientMessage()	51
6.8.4 Member Data Documentation	51
6.8.4.1 _asioContext	51
6.8.4.2 _asioSocket	51
6.8.4.3 _background	52
6.8.4.4 _bossActive	52
6.8.4.5 _bossDefeated	52
6.8.4.6 _clientEndpoint	52
6.8.4.7 _clientInfoBarID	52
6.8.4.8 _clientPlayerID	52
6.8.4.9 _clock	53
6.8.4.10 _componentManager	53
6.8.4.11 _deqConnections	53
6.8.4.12 _endOfLevel	53
6.8.4.13 _entityFactory	53
6.8.4.14 _entityManager	54
6.8.4.15 _level	54
6.8.4.16 _nbrOfPlayers	54
6.8.4.17 _nIDCounter	54
6.8.4.18 _playerConnected	54
6.8.4.19 _playerReady	54
6.8.4.20 _port	54
6.8.4.21 _qMessagesIn	55
6.8.4.22 _tempBuffer	55
6.8.4.23 _threadContext	55

6.8.4.24 <code>_watingPlayersReady</code>	55
6.9 AudioManager Class Reference	55
6.9.1 Detailed Description	56
6.9.2 Member Function Documentation	56
6.9.2.1 <code>getSoundBuffer()</code>	56
6.9.3 Member Data Documentation	57
6.9.3.1 <code>soundBuffers</code>	57
6.10 AudioSystem Class Reference	57
6.10.1 Detailed Description	58
6.10.2 Constructor & Destructor Documentation	58
6.10.2.1 <code>AudioSystem()</code>	58
6.10.3 Member Function Documentation	59
6.10.3.1 <code>playBackgroundMusic()</code>	59
6.10.3.2 <code>playSoundEffect()</code>	59
6.10.3.3 <code>stopBackgroundMusic()</code>	59
6.10.4 Member Data Documentation	59
6.10.4.1 <code>_audioManager</code>	59
6.10.4.2 <code>_backgroundMusic</code>	60
6.10.4.3 <code>_currentMusicFilePath</code>	60
6.10.4.4 <code>_soundEffect</code>	60
6.11 AutoFireSystem Class Reference	60
6.11.1 Detailed Description	61
6.11.2 Constructor & Destructor Documentation	61
6.11.2.1 <code>AutoFireSystem()</code>	61
6.11.3 Member Function Documentation	61
6.11.3.1 <code>handleAutoFire()</code>	61
6.11.4 Member Data Documentation	62
6.11.4.1 <code>_componentManager</code>	62
6.11.4.2 <code>_entityManager</code>	62
6.12 BackgroundComponent Struct Reference	62
6.13 BasicMonsterComponent Struct Reference	62
6.14 BindComponent Struct Reference	63
6.14.1 Detailed Description	63
6.14.2 Constructor & Destructor Documentation	63
6.14.2.1 <code>BindComponent()</code>	63
6.14.3 Member Data Documentation	63
6.14.3.1 <code>bind</code>	63
6.14.3.2 <code>isHovered</code>	64
6.15 BossComponent Struct Reference	64
6.15.1 Member Data Documentation	64
6.15.1.1 <code>tailSegmentIds</code>	64
6.16 <code>r_type::net::Client</code> Class Reference	64

6.16.1 Member Function Documentation	66
6.16.1.1 addEntity()	66
6.16.1.2 animateEntity()	66
6.16.1.3 displayEndOfGame()	66
6.16.1.4 initInfoBar()	66
6.16.1.5 MessageAll()	66
6.16.1.6 moveEntity()	66
6.16.1.7 PingServer()	67
6.16.1.8 removeEntity()	67
6.16.1.9 updateInfoBar()	67
6.17 CollisionSystem Class Reference	67
6.17.1 Detailed Description	68
6.17.2 Constructor & Destructor Documentation	68
6.17.2.1 CollisionSystem()	68
6.17.3 Member Function Documentation	68
6.17.3.1 checkCollision()	68
6.17.3.2 checkOffScreen()	69
6.17.4 Member Data Documentation	69
6.17.4.1 _componentManager	69
6.17.4.2 _entityManager	69
6.18 ComponentManager Class Reference	70
6.18.1 Detailed Description	70
6.18.2 Member Function Documentation	70
6.18.2.1 addComponent()	70
6.18.2.2 getComponent()	71
6.18.2.3 getComponentMap()	71
6.18.2.4 removeAllComponents()	72
6.18.2.5 removeEntityFromAllComponents()	72
6.18.2.6 removeEntityFromComponent()	72
6.18.3 Member Data Documentation	72
6.18.3.1 components	72
6.19 componentNotFound Class Reference	73
6.19.1 Detailed Description	73
6.19.2 Member Function Documentation	73
6.19.2.1 what()	73
6.20 EnemyComponent Struct Reference	74
6.21 EnemyMissileComponent Struct Reference	74
6.22 Entity Class Reference	74
6.22.1 Detailed Description	74
6.22.2 Constructor & Destructor Documentation	74
6.22.2.1 Entity()	74
6.22.3 Member Function Documentation	75

6.22.3.1 getId()	75
6.22.4 Member Data Documentation	75
6.22.4.1 _id	75
6.23 EntityFactory Class Reference	75
6.23.1 Detailed Description	77
6.23.2 Member Function Documentation	77
6.23.2.1 backgroundFactory()	77
6.23.2.2 createBackgroundLevelOne()	77
6.23.2.3 createBackgroundLevelThree()	78
6.23.2.4 createBackgroundLevelTwo()	78
6.23.2.5 createBackgroundMenu()	79
6.23.2.6 createBasicMonster()	79
6.23.2.7 createBoss()	79
6.23.2.8 createButton()	80
6.23.2.9 createEnemyMissile()	81
6.23.2.10 createFilter()	81
6.23.2.11 createForceMissile()	81
6.23.2.12 createForceWeapon()	82
6.23.2.13 createInfoBar()	82
6.23.2.14 createPlayer()	83
6.23.2.15 createPlayerMissile()	83
6.23.2.16 createPowerUpBlueLaserCrystal()	84
6.23.2.17 createShooterEnemy()	84
6.23.2.18 createSmallButton()	85
6.23.2.19 createTailEnd()	85
6.23.2.20 createTailSegment()	85
6.23.2.21 createUpdateButton()	86
6.23.2.22 createWall()	86
6.24 EntityInformation Struct Reference	86
6.24.1 Detailed Description	87
6.24.2 Member Data Documentation	87
6.24.2.1 animationComponent	87
6.24.2.2 ratio	87
6.24.2.3 spriteData	87
6.24.2.4 uniqueID	87
6.24.2.5 vPos	87
6.25 EntityManager Class Reference	87
6.25.1 Detailed Description	88
6.25.2 Member Function Documentation	88
6.25.2.1 createEntity()	88
6.25.2.2 getAllEntities()	88
6.25.2.3 getEntity()	89

6.25.2.4 removeAllEntities()	90
6.25.2.5 removeEntity()	90
6.25.3 Member Data Documentation	90
6.25.3.1 entities	90
6.25.3.2 entityNb	91
6.26 entityNotFound Class Reference	91
6.26.1 Detailed Description	91
6.26.2 Member Function Documentation	91
6.26.2.1 what()	91
6.27 failedToCreateFile Class Reference	92
6.27.1 Detailed Description	92
6.27.2 Member Function Documentation	92
6.27.2.1 what()	92
6.28 failedToLoadFont Class Reference	92
6.28.1 Detailed Description	93
6.28.2 Member Function Documentation	93
6.28.2.1 what()	93
6.29 failedToLoadSound Class Reference	93
6.29.1 Detailed Description	93
6.29.2 Member Function Documentation	94
6.29.2.1 what()	94
6.30 failedToLoadTexture Class Reference	94
6.30.1 Detailed Description	94
6.30.2 Member Function Documentation	94
6.30.2.1 what()	94
6.31 failedToOpenFile Class Reference	95
6.31.1 Detailed Description	95
6.31.2 Member Function Documentation	95
6.31.2.1 what()	95
6.32 FontManager Class Reference	95
6.32.1 Detailed Description	96
6.32.2 Member Function Documentation	96
6.32.2.1 getFont()	96
6.32.2.2 releaseFont()	96
6.32.3 Member Data Documentation	97
6.32.3.1 fonts	97
6.33 ForceMissileComponent Struct Reference	97
6.33.1 Detailed Description	97
6.33.2 Member Data Documentation	97
6.33.2.1 forceId	97
6.34 ForceWeaponComponent Struct Reference	98
6.34.1 Detailed Description	98

6.34.2 Constructor & Destructor Documentation	98
6.34.2.1 ForceWeaponComponent()	98
6.34.3 Member Data Documentation	99
6.34.3.1 attached	99
6.34.3.2 level	99
6.34.3.3 playerId	99
6.35 FrontComponent Struct Reference	99
6.35.1 Detailed Description	99
6.35.2 Constructor & Destructor Documentation	100
6.35.2.1 FrontComponent()	100
6.35.3 Member Data Documentation	100
6.35.3.1 targetId	100
6.36 HealthComponent Struct Reference	100
6.36.1 Detailed Description	100
6.36.2 Member Data Documentation	100
6.36.2.1 lives	100
6.37 HitboxComponent Struct Reference	101
6.37.1 Detailed Description	101
6.37.2 Member Data Documentation	101
6.37.2.1 h	101
6.37.2.2 w	101
6.38 r_type::net::IClient< T > Class Template Reference	101
6.38.1 Constructor & Destructor Documentation	102
6.38.1.1 IClient()	102
6.38.1.2 ~IClient()	102
6.38.2 Member Function Documentation	102
6.38.2.1 Connect()	102
6.38.2.2 Disconnect()	103
6.38.2.3 Incoming()	103
6.38.2.4 IsConnected()	103
6.38.2.5 Send()	103
6.39 IEntityFactory Class Reference	104
6.39.1 Detailed Description	105
6.39.2 Member Enumeration Documentation	106
6.39.2.1 EnemyType	106
6.39.3 Constructor & Destructor Documentation	106
6.39.3.1 ~IEntityFactory()	106
6.39.4 Member Function Documentation	106
6.39.4.1 createBackgroundLevelOne()	106
6.39.4.2 createBackgroundLevelThree()	107
6.39.4.3 createBackgroundLevelTwo()	107
6.39.4.4 createBackgroundMenu()	107

6.39.4.5 createBasicMonster()	108
6.39.4.6 createButton()	108
6.39.4.7 createEnemyMissile()	109
6.39.4.8 createForceMissile()	109
6.39.4.9 createForceWeapon()	110
6.39.4.10 createInfoBar()	110
6.39.4.11 createPlayer()	111
6.39.4.12 createPlayerMissile()	111
6.39.4.13 createPowerUpBlueLaserCrystal()	112
6.39.4.14 createShooterEnemy()	112
6.39.4.15 createSmallButton()	112
6.39.4.16 createTailEnd()	113
6.39.4.17 createTailSegment()	113
6.39.4.18 createUpdateButton()	113
6.39.4.19 createWall()	114
6.40 InputComponent Struct Reference	114
6.40.1 Detailed Description	114
6.40.2 Member Data Documentation	115
6.40.2.1 input	115
6.41 IScenes Class Reference	115
6.41.1 Detailed Description	116
6.41.2 Constructor & Destructor Documentation	116
6.41.2.1 ~IScenes()	116
6.41.3 Member Function Documentation	116
6.41.3.1 difficultyChoices()	116
6.41.3.2 gameLoop()	116
6.41.3.3 getRenderWindow()	116
6.41.3.4 inGameMenu()	116
6.41.3.5 mainMenu()	117
6.41.3.6 render()	117
6.41.3.7 settingsMenu()	117
6.41.3.8 shouldQuit()	117
6.42 ISystem Class Reference	117
6.42.1 Detailed Description	118
6.42.2 Constructor & Destructor Documentation	118
6.42.2.1 ISystem()	118
6.42.2.2 ~ISystem()	118
6.43 labelComponent Struct Reference	118
6.43.1 Detailed Description	119
6.43.2 Member Data Documentation	119
6.43.2.1 name	119
6.43.2.2 x	119

6.43.2.3 y	119
6.44 r_type::Level< T > Class Template Reference	119
6.44.1 Detailed Description	121
6.44.2 Constructor & Destructor Documentation	121
6.44.2.1 Level()	121
6.44.2.2 ~Level()	122
6.44.3 Member Function Documentation	122
6.44.3.1 AnimationUpdate()	122
6.44.3.2 ChangeBackground()	122
6.44.3.3 ChangeLevel()	123
6.44.3.4 collisionAction()	123
6.44.3.5 CollisionUpdate()	124
6.44.3.6 EndOfGame()	124
6.44.3.7 FireUpdate()	124
6.44.3.8 GetEntityBackGround()	125
6.44.3.9 GetLevel()	125
6.44.3.10 InitiateBackground()	125
6.44.3.11 LevelOne()	126
6.44.3.12 LevelThree()	126
6.44.3.13 LevelTwo()	127
6.44.3.14 MoveUpdate()	127
6.44.3.15 SetGameParameters()	128
6.44.3.16 SetSystem()	128
6.44.3.17 SpawnEntity()	128
6.44.3.18 Update()	129
6.44.4 Member Data Documentation	129
6.44.4.1 _animationSystem	129
6.44.4.2 _autoFireSystem	130
6.44.4.3 _basicMonsterSpawnTime	130
6.44.4.4 _collisionSystem	130
6.44.4.5 _gameParameters	130
6.44.4.6 _moveSystem	130
6.44.4.7 _shooterEnemySpawnTime	131
6.44.4.8 _spawnTimeMonsterThree	131
6.44.4.9 _WallSpawnTime	131
6.45 LinkForceComponent Struct Reference	131
6.45.1 Detailed Description	132
6.45.2 Constructor & Destructor Documentation	132
6.45.2.1 LinkForceComponent()	132
6.45.3 Member Data Documentation	132
6.45.3.1 targetId	132
6.46 MovementComponent Struct Reference	132

6.46.1 Detailed Description	133
6.46.2 Constructor & Destructor Documentation	133
6.46.2.1 MovementComponent() [1/2]	133
6.46.2.2 MovementComponent() [2/2]	133
6.46.3 Member Data Documentation	134
6.46.3.1 index	134
6.46.3.2 move	134
6.46.3.3 movementType	134
6.47 MoveSystem Class Reference	134
6.47.1 Detailed Description	135
6.47.2 Constructor & Destructor Documentation	135
6.47.2.1 MoveSystem()	135
6.47.3 Member Function Documentation	135
6.47.3.1 moveEntities()	135
6.47.3.2 moveEntity()	136
6.47.4 Member Data Documentation	136
6.47.4.1 _componentManager	136
6.47.4.2 _entityManager	136
6.48 OffsetComponent Struct Reference	137
6.48.1 Detailed Description	137
6.48.2 Member Data Documentation	137
6.48.2.1 offset	137
6.49 OnClickComponent Struct Reference	137
6.49.1 Detailed Description	138
6.49.2 Constructor & Destructor Documentation	138
6.49.2.1 OnClickComponent()	138
6.49.3 Member Data Documentation	138
6.49.3.1 isClicked	138
6.49.3.2 onClick	138
6.50 PlayerComponent Struct Reference	138
6.51 playerIdNotFound Class Reference	139
6.51.1 Detailed Description	139
6.51.2 Member Function Documentation	139
6.51.2.1 what()	139
6.52 PlayerMissileComponent Struct Reference	139
6.52.1 Detailed Description	140
6.52.2 Member Data Documentation	140
6.52.2.1 playerId	140
6.53 PositionComponent Struct Reference	140
6.53.1 Detailed Description	140
6.53.2 Constructor & Destructor Documentation	140
6.53.2.1 PositionComponent()	140

6.53.3 Member Data Documentation	141
6.53.3.1 x	141
6.53.3.2 y	141
6.54 PowerUpComponent Struct Reference	141
6.55 RectangleShapeComponent Struct Reference	141
6.55.1 Detailed Description	141
6.55.2 Constructor & Destructor Documentation	141
6.55.2.1 RectangleShapeComponent()	141
6.55.3 Member Data Documentation	142
6.55.3.1 rectangleShape	142
6.56 RenderSystem Class Reference	142
6.56.1 Detailed Description	142
6.56.2 Constructor & Destructor Documentation	143
6.56.2.1 RenderSystem()	143
6.56.3 Member Function Documentation	143
6.56.3.1 render()	143
6.56.4 Member Data Documentation	143
6.56.4.1 _componentManager	143
6.56.4.2 _font	144
6.56.4.3 _window	144
6.57 Scenes Class Reference	144
6.57.1 Detailed Description	147
6.57.2 Constructor & Destructor Documentation	147
6.57.2.1 Scenes()	147
6.57.2.2 ~Scenes()	147
6.57.3 Member Function Documentation	148
6.57.3.1 difficultyChoices()	148
6.57.3.2 difficultyChoicesCustomization()	148
6.57.3.3 gameLoop()	148
6.57.3.4 getRenderWindow()	148
6.57.3.5 HandleMessage()	148
6.57.3.6 HandleTransitionLevelMessage()	148
6.57.3.7 inGameMenu()	149
6.57.3.8 mainMenu()	149
6.57.3.9 render()	150
6.57.3.10 run()	150
6.57.3.11 settingsMenu()	150
6.57.3.12 shouldQuit()	150
6.57.3.13 StopGameLoop()	150
6.57.3.14 TransitionLevel()	151
6.57.4 Member Data Documentation	151
6.57.4.1 _networkClient	151

6.57.4.2 _window	151
6.58 ScoreComponent Struct Reference	151
6.58.1 Detailed Description	151
6.58.2 Member Data Documentation	151
6.58.2.1 score	151
6.59 r_type::net::Server Class Reference	152
6.59.1 Detailed Description	155
6.59.2 Constructor & Destructor Documentation	155
6.59.2.1 Server()	155
6.59.2.2 ~Server()	155
6.59.3 Member Function Documentation	155
6.59.3.1 OnClientConnect()	155
6.59.3.2 OnClientDisconnect()	156
6.59.3.3 OnMessage()	156
6.60 ShaderComponent Struct Reference	157
6.60.1 Detailed Description	157
6.60.2 Constructor & Destructor Documentation	157
6.60.2.1 ShaderComponent()	157
6.60.3 Member Data Documentation	158
6.60.3.1 shader	158
6.61 ShootComponent Struct Reference	158
6.61.1 Detailed Description	158
6.61.2 Constructor & Destructor Documentation	158
6.61.2.1 ShootComponent()	158
6.61.3 Member Data Documentation	159
6.61.3.1 canShoot	159
6.61.3.2 cooldownTime	159
6.61.3.3 nextShootTime	159
6.62 SpriteComponent Struct Reference	159
6.62.1 Detailed Description	160
6.62.2 Constructor & Destructor Documentation	160
6.62.2.1 SpriteComponent()	160
6.62.3 Member Data Documentation	160
6.62.3.1 hitboxX	160
6.62.3.2 hitboxY	161
6.62.3.3 sprite	161
6.62.3.4 type	161
6.63 SpriteDataComponent Struct Reference	161
6.63.1 Detailed Description	161
6.63.2 Member Data Documentation	162
6.63.2.1 scale	162
6.63.2.2 spritePath	162

6.63.2.3 type	162
6.64 TailComponent Struct Reference	162
6.65 TextComponent Struct Reference	162
6.65.1 Detailed Description	163
6.65.2 Constructor & Destructor Documentation	163
6.65.2.1 TextComponent()	163
6.65.3 Member Data Documentation	163
6.65.3.1 text	163
6.66 TextDataComponent Struct Reference	163
6.66.1 Detailed Description	164
6.66.2 Member Data Documentation	164
6.66.2.1 categoryIds	164
6.66.2.2 categorySize	164
6.66.2.3 categoryTexts	164
6.66.2.4 charSize	164
6.66.2.5 fontPath	165
6.67 TextureManager Class Reference	165
6.67.1 Member Function Documentation	165
6.67.1.1 getTexture()	165
6.67.1.2 releaseTexture()	166
6.67.2 Member Data Documentation	166
6.67.2.1 textures	166
6.68 UIEntityInformation Struct Reference	166
6.68.1 Detailed Description	167
6.68.2 Member Data Documentation	167
6.68.2.1 lives	167
6.68.2.2 score	167
6.68.2.3 spriteData	167
6.68.2.4 textData	167
6.68.2.5 uniqueID	167
6.69 UpdateSystem Class Reference	168
6.69.1 Detailed Description	168
6.69.2 Constructor & Destructor Documentation	169
6.69.2.1 UpdateSystem()	169
6.69.3 Member Function Documentation	169
6.69.3.1 updateSpritePositions()	169
6.69.4 Member Data Documentation	169
6.69.4.1 _componentManager	169
6.69.4.2 _entityManager	170
6.69.4.3 _window	170
6.70 UpdateTextComponent Struct Reference	170
6.70.1 Constructor & Destructor Documentation	170

6.70.1.1 UpdateTextComponent()	170
6.70.2 Member Data Documentation	170
6.70.2.1 updateText	170
6.71 VelocityComponent Struct Reference	171
6.71.1 Detailed Description	171
6.71.2 Member Data Documentation	171
6.71.2.1 x	171
6.71.2.2 y	171
6.72 vf2d Struct Reference	171
6.72.1 Detailed Description	172
6.72.2 Member Data Documentation	172
6.72.2.1 x	172
6.72.2.2 y	172
6.73 WallComponent Struct Reference	172
7 File Documentation	173
7.1 /home/runner/work/R-Type/R-Type/Client/Interface/Include/mainmenu.hpp File Reference	173
7.1.1 Function Documentation	173
7.1.1.1 MainMenu()	173
7.2 mainmenu.hpp	173
7.3 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/a_client.hpp File Reference	174
7.4 a_client.hpp	174
7.5 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/client.hpp File Reference	175
7.6 client.hpp	176
7.7 /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp File Reference	178
7.8 i_client.hpp	178
7.9 /home/runner/work/R-Type/R-Type/Client/Interface/Include/scenes.hpp File Reference	179
7.9.1 Function Documentation	179
7.9.1.1 keyToString()	179
7.10 scenes.hpp	180
7.11 /home/runner/work/R-Type/R-Type/Client/Src/keyToString.cpp File Reference	180
7.11.1 Function Documentation	181
7.11.1.1 keyToString()	181
7.12 /home/runner/work/R-Type/R-Type/Client/Src/main.cpp File Reference	181
7.12.1 Function Documentation	181
7.12.1.1 isValidIPv4()	181
7.12.1.2 isValidPort()	181
7.12.1.3 main()	181
7.13 /home/runner/work/R-Type/R-Type/Server/Src/main.cpp File Reference	182
7.13.1 Function Documentation	182
7.13.1.1 isValidPort()	182
7.13.1.2 main()	183

7.13.1.3 <code>signal_handler()</code>	183
7.13.2 Variable Documentation	183
7.13.2.1 <code>loopRunning</code>	183
7.14 <code>/home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp</code> File Reference	184
7.14.1 Function Documentation	184
7.14.1.1 <code>createDaltonismChoiceButtons()</code>	184
7.14.1.2 <code>createKeyBindingButtons()</code>	184
7.14.1.3 <code>handleEvents()</code>	185
7.14.1.4 <code>reloadFilter()</code>	185
7.14.1.5 <code>waitForKey()</code>	185
7.15 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp</code> File Reference	185
7.16 <code>a_scenes.hpp</code>	186
7.17 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/audio_manager.hpp</code> File Reference	187
7.18 <code>audio_manager.hpp</code>	187
7.19 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_component.hpp</code> File Reference	188
7.19.1 Detailed Description	188
7.20 <code>ally_component.hpp</code>	188
7.21 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_missile_component.hpp</code> File Reference	189
7.21.1 Detailed Description	189
7.22 <code>ally_missile_component.hpp</code>	189
7.23 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/animation_component.hpp</code> File Reference	189
7.23.1 Function Documentation	189
7.23.1.1 <code>operator"!=()</code>	189
7.24 <code>animation_component.hpp</code>	190
7.25 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/background_component.hpp</code> File Reference	190
7.25.1 Detailed Description	191
7.26 <code>background_component.hpp</code>	191
7.27 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/basic_monster_component.hpp</code> File Reference	191
7.27.1 Detailed Description	191
7.28 <code>basic_monster_component.hpp</code>	191
7.29 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/bind_component.hpp</code> File Reference	192
7.30 <code>bind_component.hpp</code>	192
7.31 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/boss_component.hpp</code> File Reference	192
7.32 <code>boss_component.hpp</code>	192
7.33 <code>/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/component_manager.hpp</code> File Reference	193
7.34 <code>component_manager.hpp</code>	193

7.35	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/components.hpp File Reference	194
7.36	components.hpp	195
7.37	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_component.hpp File Reference	195
7.37.1	Detailed Description	196
7.38	enemy_component.hpp	196
7.39	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_missile_component.hpp File Reference	196
7.39.1	Detailed Description	196
7.40	enemy_missile_component.hpp	196
7.41	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_missile_component.hpp File Reference	197
7.42	force_missile_component.hpp	197
7.43	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_weapon_component.hpp File Reference	197
7.44	force_weapon_component.hpp	197
7.45	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/front_component.hpp File Reference	198
7.46	front_component.hpp	198
7.47	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/health_component.hpp File Reference	198
7.48	health_component.hpp	198
7.49	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/hitbox_component.hpp File Reference	199
7.50	hitbox_component.hpp	199
7.51	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/input_component.hpp File Reference	199
7.51.1	Enumeration Type Documentation	199
7.51.1.1	InputType	199
7.52	input_component.hpp	200
7.53	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp File Reference	200
7.54	label_component.hpp	201
7.55	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link_force_component.hpp File Reference	201
7.56	link_force_component.hpp	201
7.57	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp File Reference	201
7.57.1	Enumeration Type Documentation	202
7.57.1.1	MovementType	202
7.58	movement_component.hpp	202
7.59	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/offset_component.hpp File Reference	203
7.60	offset_component.hpp	203

7.61	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/on_click_component.hpp	
	File Reference	203
7.61.1	Detailed Description	203
7.62	on_click_component.hpp	204
7.63	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_component.hpp	File Reference
	Reference	204
7.63.1	Detailed Description	204
7.64	player_component.hpp	204
7.65	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_missile_component.hpp	File Reference
	Reference	204
7.66	player_missile_component.hpp	205
7.67	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/position_component.hpp	File Reference
	Reference	205
7.68	position_component.hpp	205
7.69	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/power_up_component.hpp	File Reference
	Reference	205
7.69.1	Detailed Description	206
7.70	power_up_component.hpp	206
7.71	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/rectangleShapeComponent.hpp	File Reference
	Reference	206
7.72	rectangleShapeComponent.hpp	206
7.73	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/score_component.hpp	File Reference
	Reference	207
7.73.1	Detailed Description	207
7.74	score_component.hpp	207
7.75	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shader_component.hpp	File Reference
	Reference	207
7.76	shader_component.hpp	208
7.77	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_component.hpp	File Reference
	Reference	208
7.78	shoot_component.hpp	208
7.79	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_component.hpp	File Reference
	Reference	209
7.80	sprite_component.hpp	209
7.81	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_data_component.hpp	File Reference
	Reference	209
7.81.1	Function Documentation	210
7.81.1.1	operator<<()	210
7.82	sprite_data_component.hpp	210
7.83	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/tail_component.hpp	File Reference
	Reference	211
7.84	tail_component.hpp	211
7.85	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp	File Reference
	Reference	211
7.86	text_component.hpp	211

7.87	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_data_component.hpp	
	File Reference	212
7.88	text_data_component.hpp	212
7.89	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/update_text_component.hpp	
	File Reference	212
7.90	update_text_component.hpp	213
7.91	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/velocity_component.hpp	
	File Reference	213
7.92	velocity_component.hpp	213
7.93	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wall_component.hpp	
	File Reference	213
	7.93.1 Detailed Description	214
7.94	wall_component.hpp	214
7.95	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/creatable_client_object.hpp	
	File Reference	214
	7.95.1 Enumeration Type Documentation	214
	7.95.1.1 CreatableClientObject	214
7.96	creatable_client_object.hpp	215
7.97	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hpp	
	File Reference	215
7.98	entity.hpp	215
7.99	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_factory.hpp	
	File Reference	215
7.100	entity_factory.hpp	216
7.101	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp	
	File Reference	217
7.102	entity_manager.hpp	217
7.103	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/i_entity_factory.hpp	
	File Reference	218
7.104	i_entity_factory.hpp	218
7.105	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/entity_struct.hpp	
	File Reference	219
7.106	entity_struct.hpp	220
7.107	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp	
	File Reference	220
7.108	error_handling.hpp	221
7.109	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_manager.hpp	
	File Reference	221
7.110	font_manager.hpp	222
7.111	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_path.hpp	
	File Reference	222
	7.111.1 Enumeration Type Documentation	223
	7.111.1.1 FontPath	223
	7.111.2 Function Documentation	223
	7.111.2.1 FontFactory()	223
	7.111.2.2 operator<<()	224
7.112	font_path.hpp	224
7.113	/home/runner/work/R-Type/R-Type/ECS/Interface/Include/game_text.hpp	
	File Reference	224
	7.113.1 Enumeration Type Documentation	225
	7.113.1.1 GameText	225
	7.113.2 Function Documentation	225
	7.113.2.1 GameTextFactory()	225

7.113.2.2 operator<<()	226
7.114 game_text.hpp	226
7.115 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/hitbox_tmp.hpp File Reference	226
7.115.1 Function Documentation	227
7.115.1.1 CheckEntityMovement()	227
7.115.1.2 CheckEntityPosition()	227
7.116 hitbox_tmp.hpp	227
7.117 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/i_scenes.hpp File Reference	228
7.118 i_scenes.hpp	228
7.119 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/macros.hpp File Reference	229
7.119.1 Macro Definition Documentation	229
7.119.1.1 SCREEN_HEIGHT	229
7.119.1.2 SCREEN_WIDTH	229
7.120 macros.hpp	229
7.121 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/sound_path.hpp File Reference	229
7.121.1 Enumeration Type Documentation	230
7.121.1.1 ActionType	230
7.121.2 Function Documentation	231
7.121.2.1 SoundFactory()	231
7.122 sound_path.hpp	231
7.123 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/sprite_path.hpp File Reference	232
7.123.1 Enumeration Type Documentation	232
7.123.1.1 SpritePath	232
7.123.2 Function Documentation	233
7.123.2.1 operator<<()	233
7.123.2.2 SpriteFactory()	233
7.124 sprite_path.hpp	234
7.125 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/audio_system.hpp File Reference	234
7.126 audio_system.hpp	235
7.127 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/auto_fire_system.hpp File Reference	235
7.128 auto_fire_system.hpp	235
7.129 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/collision_system.hpp File Reference	236
7.130 collision_system.hpp	236
7.131 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/i_system.hpp File Reference	236
7.132 i_system.hpp	237
7.133 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/move_system.hpp File Reference	237
7.134 move_system.hpp	237
7.135 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/render_system.hpp File Reference	238
7.136 render_system.hpp	238
7.137 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/systems.hpp File Reference	238

7.138 systems.hpp	239
7.139 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/update_system.hpp File Reference	239
7.140 update_system.hpp	239
7.141 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/texture_manager.hpp File Reference	240
7.142 texture_manager.hpp	240
7.143 /home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp File Reference	240
7.144 /home/runner/work/R-Type/R-Type/ECS/Src/Entities/entity_factory.cpp File Reference	241
7.144.1 Function Documentation	241
7.144.1.1 operator<<() [1/4]	241
7.144.1.2 operator<<() [2/4]	241
7.144.1.3 operator<<() [3/4]	241
7.144.1.4 operator<<() [4/4]	242
7.145 /home/runner/work/R-Type/R-Type/ECS/Src/font_path.cpp File Reference	242
7.145.1 Function Documentation	242
7.145.1.1 FontFactory()	242
7.146 /home/runner/work/R-Type/R-Type/ECS/Src/game_text.cpp File Reference	243
7.146.1 Function Documentation	243
7.146.1.1 GameTextFactory()	243
7.147 /home/runner/work/R-Type/R-Type/ECS/Src/hitbox_tmp.cpp File Reference	243
7.147.1 Function Documentation	244
7.147.1.1 CheckCollisionLogic()	244
7.147.1.2 CheckEntityMovement()	244
7.147.1.3 CheckEntityPosition()	244
7.148 /home/runner/work/R-Type/R-Type/ECS/Src/sound_path.cpp File Reference	245
7.148.1 Function Documentation	245
7.148.1.1 SoundFactory()	245
7.149 /home/runner/work/R-Type/R-Type/ECS/Src/sprite_path.cpp File Reference	246
7.149.1 Function Documentation	246
7.149.1.1 SpriteFactory()	246
7.150 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_system.cpp File Reference	246
7.151 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/auto_fire_system.cpp File Reference	246
7.152 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/collision_system.cpp File Reference	247
7.153 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/move_system.cpp File Reference	247
7.154 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/render_system.cpp File Reference	247
7.155 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/update_system.cpp File Reference	247
7.156 /home/runner/work/R-Type/R-Type/Server/Interface/Include/animation_system.hpp File Reference	247
7.156.1 Enumeration Type Documentation	248
7.156.1.1 AnimationBasicMonster	248
7.156.1.2 AnimationBoss	248
7.156.1.3 AnimationForceMissile1	249
7.156.1.4 AnimationForceMissile2	249

7.156.1.5 AnimationForceMissile3	249
7.156.1.6 AnimationForceWeapon1	249
7.156.1.7 AnimationForceWeapon2	250
7.156.1.8 AnimationForceWeapon3	250
7.156.1.9 AnimationShip	250
7.156.2 Function Documentation	251
7.156.2.1 operator"!=()"	251
7.157 animation_system.hpp	251
7.158 /home/runner/work/R-Type/R-Type/Server/Interface/Include/level.hpp File Reference	253
7.159 level.hpp	253
7.160 /home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/a_server.hpp File Reference	262
7.161 a_server.hpp	263
7.162 /home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/server.hpp File Reference	272
7.163 server.hpp	272
7.164 /home/runner/work/R-Type/R-Type/Server/Src/animation_system.cpp File Reference	272
7.164.1 Function Documentation	273
7.164.1.1 animateForceMissileLevel1()	273
7.164.1.2 animateForceMissileLevel2()	273
7.164.1.3 animateForceMissileLevel3()	273
7.164.1.4 animateForceWeaponLevel1()	273
7.164.1.5 animateForceWeaponLevel2()	273
7.164.1.6 animateForceWeaponLevel3()	274
7.164.1.7 animationBasicMonsterFactory()	274
7.164.1.8 animationBossFactory()	274
7.164.1.9 animationForceMissile1Factory()	274
7.164.1.10 animationForceMissile2Factory()	274
7.164.1.11 animationForceMissile3Factory()	274
7.164.1.12 animationForceWeapon1Factory()	274
7.164.1.13 animationForceWeapon2Factory()	274
7.164.1.14 animationForceWeapon3Factory()	274
7.164.1.15 animationShipFactory()	274
7.164.1.16 operator"!=()"	275
7.164.1.17 operator=="()"	275
7.165 /home/runner/work/R-Type/R-Type/Server/Src/server.cpp File Reference	275

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	55
BackgroundComponent	62
BasicMonsterComponent	62
BindComponent	63
BossComponent	64
ComponentManager	70
EnemyComponent	74
EnemyMissileComponent	74
Entity	74
EntityInformation	86
EntityManager	87
std::exception	
componentNotFound	73
entityNotFound	91
failedToCreateFile	92
failedToLoadFont	92
failedToLoadSound	93
failedToLoadTexture	94
failedToOpenFile	95
playerIdNotFound	139
FontManager	95
ForceMissileComponent	97
ForceWeaponComponent	98
FrontComponent	99
HealthComponent	100
HitboxComponent	101
r_type::net::IClient< T >	101
r_type::net::AClient< TypeMessage >	15
r_type::net::Client	64
r_type::net::AClient< T >	15
r_type::net::IClient< TypeMessage >	101

IEntityFactory	104
EntityFactory	75
ILevel	
r_type::Level< TypeMessage >	119
r_type::Level< T >	119
InputComponent	114
IScenes	115
AScenes	25
Scenes	144
r_type::net::IServer	
r_type::net::AServer< TypeMessage >	38
r_type::net::Server	152
r_type::net::AServer< T >	38
ISystem	117
AnimationSystem	21
AudioSystem	57
AutoFireSystem	60
CollisionSystem	67
MoveSystem	134
RenderSystem	142
UpdateSystem	168
labelComponent	118
LinkForceComponent	131
MovementComponent	132
OffsetComponent	137
OnClickComponent	137
PlayerComponent	138
PlayerMissileComponent	139
PositionComponent	140
PowerUpComponent	141
RectangleShapeComponent	141
ScoreComponent	151
ShaderComponent	157
ShootComponent	158
SpriteComponent	159
SpriteDataComponent	161
TailComponent	162
TextComponent	162
TextDataComponent	163
TextureManager	165
UIEntityInformation	166
UpdateTextComponent	170
VelocityComponent	171
vf2d	171
WallComponent	172

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	21
AScenes	25
r_type::net::AServer< T >	
AServer class template for managing server operations	38
AudioManager	
Manages and caches sound buffers for efficient audio playback	55
AudioSystem	
Manages audio playback within the application	57
AutoFireSystem	
A system that handles automatic firing mechanisms for entities	60
BackgroundComponent	62
BasicMonsterComponent	62
BindComponent	
A component that binds a function to handle scene transitions	63
BossComponent	64
r_type::net::Client	64
CollisionSystem	
Manages collision detection and response within the ECS framework	67
ComponentManager	
Manages the components of entities in an ECS system	70
componentNotFound	
Exception class for when a component is not found	73
EnemyComponent	74
EnemyMissileComponent	74
Entity	
Represents an entity in the ECS system	74
EntityFactory	
A factory class for creating various types of entities	75

EntityInformation	Represents information about an entity	86
EntityManager	Manages the creation, removal, and retrieval of entities	87
entityNotFound	Exception class for entity not found error	91
failedToCreateFile	Exception class for handling file creation failures	92
failedToLoadFont	Exception class for handling font loading failures	92
failedToLoadSound	Exception class for handling sound loading failures	93
failedToLoadTexture	Exception class for failed texture loading	94
failedToOpenFile	Exception class for handling file opening failures	95
FontManager	Manages the loading and retrieval of font resources	95
ForceMissileComponent	Component representing a force missile in the ECS system	97
ForceWeaponComponent	Represents a component for a force weapon in the game	98
FrontComponent	A component that represents the front of an entity	99
HealthComponent	Represents the health attributes of an entity	100
HitboxComponent	Represents the hitbox dimensions of an entity	101
r_type::net::IClient< T >	101
IEntityFactory	The interface for an entity factory	104
InputComponent	Component for handling input actions	114
IScenes	Interface for managing different scenes in a game	115
ISystem	Interface for all systems in the ECS (Entity Component System) architecture	117
labelComponent	Represents a label component with a name and position coordinates	118
r_type::Level< T >	The Level class template manages the game level, including updating game state, handling collisions, and managing entities	119
LinkForceComponent	Component that links an entity to a target entity by ID	131
MovementComponent	Represents a component that handles movement in the ECS system	132
MoveSystem	System responsible for moving entities within the ECS framework	134
OffsetComponent	Component that represents an offset value	137
OnClickComponent	Component that handles click events	137
PlayerComponent	138
playerIdNotFound	Exception class for handling cases where a player ID is not found	139
PlayerMissileComponent	Component that represents a missile belonging to a player	139

PositionComponent	
A component that represents the position of an entity in 2D space	140
PowerUpComponent	141
RectangleShapeComponent	
A component that holds an <code>sf::RectangleShape</code>	141
RenderSystem	
A system responsible for rendering entities in the ECS framework	142
Scenes	
Represents a class that manages different scenes in a game	144
ScoreComponent	
Component that holds the score of an entity	151
r_type::net::Server	
A server class that handles client connections and messaging	152
ShaderComponent	
A component that holds a shader	157
ShootComponent	
Component that handles shooting mechanics for an entity	158
SpriteComponent	
A component that represents a sprite in the ECS (Entity Component System)	159
SpriteDataComponent	
Component that holds data related to a sprite	161
TailComponent	162
TextComponent	
A component that encapsulates an SFML text object	162
TextDataComponent	
Component that holds text-related data for an entity	163
TextureManager	165
UIEntityInformation	
Represents the information of a UI entity in the game	166
UpdateSystem	
A system responsible for updating sprite positions in the game	168
UpdateTextComponent	170
VelocityComponent	
Represents the velocity of an entity in 2D space	171
vf2d	
Represents a 2D vector with x and y coordinates	171
WallComponent	172

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	173
/home/runner/work/R-Type/R-Type/Client/Interface/Include/scenes.hpp	179
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/a_client.hpp	174
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/client.hpp	175
/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp	178
/home/runner/work/R-Type/R-Type/Client/Src/keyToString.cpp	180
/home/runner/work/R-Type/R-Type/Client/Src/main.cpp	181
/home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp	184
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp	185
/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	214
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/entity_struct.hpp	219
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp	220
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_manager.hpp	221
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/font_path.hpp	222
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/game_text.hpp	224
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/hitbox_tmp.hpp	226
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/i_scenes.hpp	228
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/macros.hpp	229
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/sound_path.hpp	229
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/sprite_path.hpp	232
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/texture_manager.hpp	240
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_component.hpp	
Defines the AllyComponent structure	188
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_missile_component.hpp	
Defines the AllyMissileComponent structure	189
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/animation_component.hpp	189
/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	191
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/bind_component.hpp	192
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/boss_component.hpp	192
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/component_manager.hpp	193
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/components.hpp	194

/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_component.hpp	
Defines the EnemyComponent structure	195
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_missile_component.hpp	
Defines the EnemyMissileComponent structure	196
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_missile_component.hpp	197
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/force_weapon_component.hpp	197
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/front_component.hpp	198
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/health_component.hpp	198
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/hitbox_component.hpp	199
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/input_component.hpp	199
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp	200
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link_force_component.hpp	201
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp	201
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/offset_component.hpp	203
/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	203
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_component.hpp	
Defines the PlayerComponent structure	204
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_missile_component.hpp	204
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/position_component.hpp	205
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/power_up_component.hpp	
Defines the PowerUpComponent structure	205
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/rectangleShapeComponent.hpp	206
/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	207
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shader_component.hpp	207
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_component.hpp	208
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_component.hpp	209
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_data_component.hpp	209
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/tail_component.hpp	211
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp	211
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_data_component.hpp	212
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/update_text_component.hpp	212
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/velocity_component.hpp	213
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wall_component.hpp	
Defines the WallComponent structure	213
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hpp	215
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_factory.hpp	215
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp	217
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/i_entity_factory.hpp	218
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/audio_system.hpp	234
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/auto_fire_system.hpp	235
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/collision_system.hpp	236
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/i_system.hpp	236
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/move_system.hpp	237
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/render_system.hpp	238
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/systems.hpp	238
/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/update_system.hpp	239
/home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp	240
/home/runner/work/R-Type/R-Type/ECS/Src/font_path.cpp	242
/home/runner/work/R-Type/R-Type/ECS/Src/game_text.cpp	243
/home/runner/work/R-Type/R-Type/ECS/Src/hitbox_tmp.cpp	243
/home/runner/work/R-Type/R-Type/ECS/Src/sound_path.cpp	245
/home/runner/work/R-Type/R-Type/ECS/Src/sprite_path.cpp	246
/home/runner/work/R-Type/R-Type/ECS/Src/Entities/entity_factory.cpp	241
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_system.cpp	246
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/auto_fire_system.cpp	246
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/collision_system.cpp	247

/home/runner/work/R-Type/R-Type/ECS/Src/Systems/ move_system.cpp	247
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/ render_system.cpp	247
/home/runner/work/R-Type/R-Type/ECS/Src/Systems/ update_system.cpp	247
/home/runner/work/R-Type/R-Type/Server/Interface/Include/ animation_system.hpp	247
/home/runner/work/R-Type/R-Type/Server/Interface/Include/ level.hpp	253
/home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/ a_server.hpp	262
/home/runner/work/R-Type/R-Type/Server/Interface/Include/Net/ server.hpp	272
/home/runner/work/R-Type/R-Type/Server/Src/ animation_system.cpp	272
/home/runner/work/R-Type/R-Type/Server/Src/ main.cpp	182
/home/runner/work/R-Type/R-Type/Server/Src/ server.cpp	275

Chapter 5

Namespace Documentation

5.1 `r_type` Namespace Reference

Namespaces

- namespace [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 [AServer](#)

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

- class [Client](#)
- class [IClient](#)
- 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.

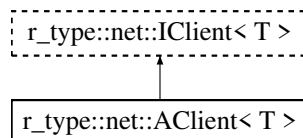
The documentation for this class was generated from the following file:

- [/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](#) ()

Public Member Functions inherited from [r_type::net::IClient< T >](#)

- [IClient](#) ()
- [virtual ~IClient](#) ()

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

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

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

<code>_offset</code>	The offset of the animation.
<code>_dimension</code>	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:

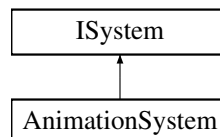
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/animation_component.hpp](#)

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, bool &endOfLevel)
- *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, std::optional< [HitboxComponent](#) * > &hitbox)
- *Animates the force weapon based on its current state.*
- void [animateForceMissile](#) (std::optional< [ForceWeaponComponent](#) * > &forceWeapon, std::optional< [AnimationComponent](#) * > &animation, std::optional< [HitboxComponent](#) * > &hitbox)
- *Animates the force missile based on the provided components.*
- void [animateBoss](#) (std::optional< [BossComponent](#) * > &boss, std::optional< [AnimationComponent](#) * > &animation)

Public Member Functions inherited from [ISystem](#)

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

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 animateBoss()

```
void AnimationSystem::animateBoss (
    std::optional< BossComponent * > & boss,
    std::optional< AnimationComponent * > & animation )
```

6.6.3.3 animateForceMissile()

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

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.

6.6.3.4 animateForceWeapon()

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

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 .

6.6.3.5 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.

6.6.3.6 AnimationEntities()

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

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.

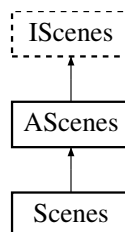
The documentation for this class was generated from the following files:

- `/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](#) ,
[CHOOSE_DIFFICULTY](#) , [CUSTOM_DIFFICULTY](#) , [TRANSITION_LEVEL](#) , [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](#) ([GameParameters](#) const mode)
Set the Game Mode object.
- [GameParameters](#) [getGameMode](#) () const
Get 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.
- void [SetPlayerReady](#) (bool ready)
Sets the player ready status.
- bool [GetPlayerReady](#) () const
Retrieves the player ready status.

Public Member Functions inherited from [IScenes](#)

- 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.

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

- GameParameters `_currentGameMode`
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 `_displayKeyBindsChoice` = 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.
- bool `_playerReady` = false

6.7.1 Member Enumeration Documentation

6.7.1.1 Actions

```
enum class 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 class 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 class 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 class 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.
CHOOSE_DIFFICULTY	
CUSTOM_DIFFICULTY	
TRANSITION_LEVEL	
EXIT	Represents the exit scene where the application is closing.

6.7.1.5 SpriteType

```
enum class 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 getGameMode()

```
GameParameters AScenes::getGameMode ( ) const [inline]
```

Get the Game Mode object.

Returns

GameParameters

6.7.3.6 `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.

6.7.3.7 `GetPlayerReady()`

```
bool AScenes::GetPlayerReady ( ) const [inline]
```

Retrieves the player ready status.

This function returns the player ready status.

Returns

bool The player ready status.

6.7.3.8 `getPort()`

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

Retrieves the port number.

Returns

int The port number.

6.7.3.9 `getPreviousScene()`

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

Get the Previous Scene object.

Returns

Scene

6.7.3.10 `setDaltonism()`

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

Set the Daltonism object.

Parameters

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

6.7.3.11 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).
-------------------------------	--

6.7.3.12 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).
------------------------------	---

6.7.3.13 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).
------------------------------	---

6.7.3.14 setGameMode()

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

Set the Game Mode object.

Parameters

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

6.7.3.15 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.
-----------	------------------------------------

6.7.3.16 SetPlayerReady()

```
void AScenes::SetPlayerReady (
    bool ready ) [inline]
```

Sets the player ready status.

This function sets the player ready status to the specified value.

Parameters

<i>ready</i>	The player ready status to set.
--------------	---------------------------------

6.7.3.17 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.
-------------	----------------------------

6.7.3.18 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

```
GameParameters AScenes::_currentGameMode [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 `_playerReady`

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

6.7.4.9 `_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.

6.7.4.10 `_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.

6.7.4.11 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.

6.7.4.12 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](#).

6.7.4.13 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:

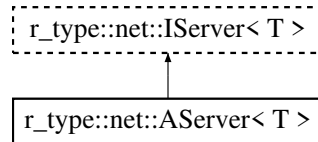
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_scenes.hpp
- /home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp

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 > > plgnoreClient=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.

- [void RemoveBossTail \(int bossId\)](#)
- [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 \(int entityId\)](#)
- [void InitBoss \(r_type::net::AServer< T > *server\)](#)
- [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.
- [bool _endOfLevel = false](#)

- `bool _bossActive = false`
- `bool _bossDefeated = false`
- `bool _watingPlayersReady = false`
- `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`
- `int _playerReady = 0`

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

<i>playerIdNotFound</i>	If the player ID is not found in the map.
---	---

6.8.3.10 InitBoss()

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

6.8.3.11 InitiateEnemyMissile()

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

6.8.3.12 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.

6.8.3.13 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.

6.8.3.14 InitiateWeaponForce()

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

6.8.3.15 InitInfoBar()

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

6.8.3.16 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.
------------	--

Parameters

<i>plgnoreClient</i>	A shared pointer to a client connection that should be ignored. Defaults to nullptr.
----------------------	--

6.8.3.17 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.
<i>msg</i>	The message to be sent to the client.

6.8.3.18 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

6.8.3.19 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>	
---------------	--

6.8.3.20 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.
---------------	--

6.8.3.21 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>	

6.8.3.22 RemoveBossTail()

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

6.8.3.23 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.
-----------	--

6.8.3.24 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.
------------------------	--

6.8.3.25 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.
-----------	--

6.8.3.26 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.

6.8.3.27 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>	
--------------	--

6.8.3.28 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.

6.8.3.29 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.

6.8.3.30 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.

6.8.3.31 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.

6.8.3.32 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.

6.8.3.33 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 `_bossActive`

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

6.8.4.5 `_bossDefeated`

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

6.8.4.6 `_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.

6.8.4.7 `_clientInfoBarID`

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

6.8.4.8 `_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.

6.8.4.9 `_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.

6.8.4.10 `_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.

6.8.4.11 `_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.
----------------	--

6.8.4.12 `_endOfLevel`

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

6.8.4.13 `_entityFactory`

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

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

6.8.4.14 `_entityManager`

```
template<typename T >
EntityManager 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.

6.8.4.15 `_level`

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

6.8.4.16 `_nbrOfPlayers`

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

Number of players currently connected to the server.

6.8.4.17 `_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.

6.8.4.18 `_playerConnected`

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

6.8.4.19 `_playerReady`

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

6.8.4.20 `_port`

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


6.8.4.21 `_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.

6.8.4.22 `_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.

6.8.4.23 `_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.

6.8.4.24 `_watingPlayersReady`

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

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

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

Returns

A reference to the sound buffer.

Exceptions

<code>std::runtime_error</code>	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:

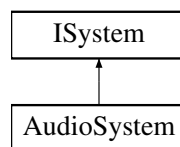
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/[audio_manager.hpp](#)

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.

Public Member Functions inherited from [ISystem](#)

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

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:

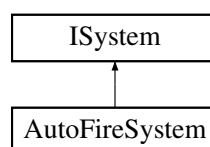
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/audio_system.hpp`
- `/home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_system.cpp`

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.

Public Member Functions inherited from [ISystem](#)

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

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>
```

Public Attributes

- std::vector< int > [tailSegmentIds](#)

6.15.1 Member Data Documentation

6.15.1.1 tailSegmentIds

```
std::vector<int> BossComponent::tailSegmentIds
```

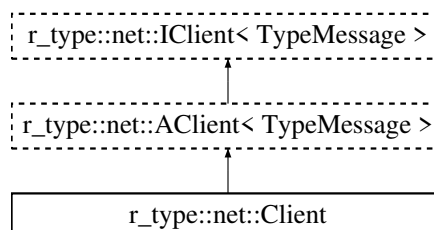
The documentation for this struct was generated from the following file:

- /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 (UIEntityInformation entity, ComponentManager &componentManager, TextureManager &textureManager, FontManager &fontManager, sf::Vector2u windowSize)`
- `void updateInfoBar (UIEntityInformation entity, ComponentManager &componentManager)`
- `void addEntity (EntityInformation entity, ComponentManager &componentManager, TextureManager &textureManager, sf::Vector2u windowSize)`
- `void removeEntity (int entityId, ComponentManager &componentManager)`
- `void moveEntity (uint32_t id, sf::Vector2f newPos, ComponentManager &componentManager, sf::Vector2u windowSize)`
- `void animateEntity (int entityId, AnimationComponent rect, ComponentManager &componentManager)`
- `void displayEndOfGame (ComponentManager &componentManager, TextureManager &textureManager, FontManager &fontManager, sf::Vector2u windowSize)`

Public Member Functions inherited from `r_type::net::AClient< TypeMessage >`

- `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< TypeMessage > &msg)`
Send message to server.
- `ThreadSafeQueue< OwnedMessage< TypeMessage > > & Incoming ()`
get incoming messages
- `const std::unique_ptr< Connection< TypeMessage > > & getConnection ()`
- `void setPlayerId (uint32_t id)`
- `uint32_t getPlayerId ()`
- `void setWindowSize (sf::Vector2u size)`
- `sf::Vector2u getWindowSize ()`

Public Member Functions inherited from `r_type::net::IClient< T >`

- `IClient ()`
- `virtual ~IClient ()`
- `virtual void Send (const Message< T > &msg)=0`
Send message to server.

Additional Inherited Members**Protected Attributes inherited from `r_type::net::AClient< TypeMessage >`**

- `asio::io_context m_context`
- `std::thread thrContext`
- `std::unique_ptr< Connection< TypeMessage > > m_connection`

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 displayEndOfGame()

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

6.16.1.4 initInfoBar()

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

6.16.1.5 MessageAll()

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

Send a message to the server to all other clients.

6.16.1.6 moveEntity()

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

6.16.1.7 PingServer()

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

Send a message to the server to get the ping.

6.16.1.8 removeEntity()

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

6.16.1.9 updateInfoBar()

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

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

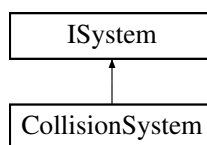
- /home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/[client.hpp](#)

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.

Public Member Functions inherited from [ISystem](#)

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

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:

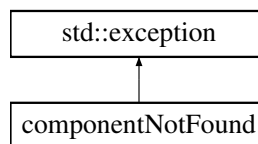
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/component_manager.hpp](#)

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:

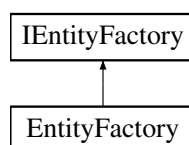
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/[entity.hpp](#)

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, int posX, int posY) 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 createUpdateButton](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, std::string text, std::function< [IScenes](#) *([AScenes](#) *)> *onClick, std::function< std::string([GameParameters](#))> *updateText, float x, float y) override
- [Entity createEnemyMissile](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, uint32_t entityId) override

- Creates an enemy missile entity.*
- [Entity createFilter](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [AScenes::DaltonismMode](#) mode)
- Creates a filter entity.*
- [Entity backgroundFactory](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, GameState type)
- [Entity createBoss](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [EntityFactory](#) &entityFactory)
- Creates a boss entity.*
- [Entity createTailSegment](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override
- [Entity createTailEnd](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager) override

Public Member Functions inherited from [IEntityFactory](#)

- virtual [~IEntityFactory](#) ()=default
- Destroy the [IEntityFactory](#) object.*

Additional Inherited Members

Public Types inherited from [IEntityFactory](#)

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

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 createBoss()

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

```

    ComponentManager & componentManager,
    EntityFactory & entityFactory )

```

Creates a boss entity.

This function creates a boss 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 boss entity.

6.23.2.8 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](#).

6.23.2.9 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](#).

6.23.2.10 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.

6.23.2.11 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](#).

6.23.2.12 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](#).

6.23.2.13 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](#).

6.23.2.14 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](#).

6.23.2.15 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](#).

6.23.2.16 createPowerUpBlueLaserCrystal()

```
Entity EntityFactory::createPowerUpBlueLaserCrystal (
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int posX,
    int posY ) [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](#).

6.23.2.17 createShooterEnemy()

```
Entity EntityFactory::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](#).

6.23.2.18 createSmallButton()

```
Entity EntityFactory::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](#).

6.23.2.19 createTailEnd()

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

Implements [IEntityFactory](#).

6.23.2.20 createTailSegment()

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

Implements [IEntityFactory](#).

6.23.2.21 createUpdateButton()

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

Implements [IEntityFactory](#).

6.23.2.22 createWall()

```
Entity EntityFactory::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:

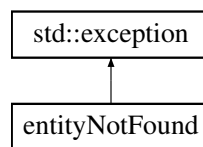
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp](#)

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:

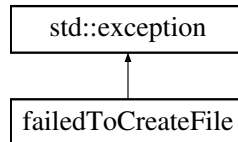
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp](#)

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:

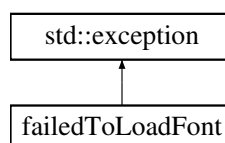
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

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:

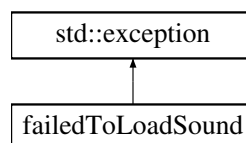
- `/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp`

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:

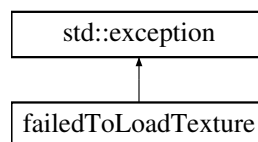
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp](#)

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:

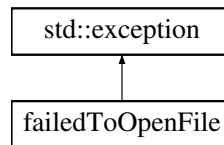
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/error_handling.hpp](#)

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 [lives](#)

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 lives

```
int HealthComponent::lives
```

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.

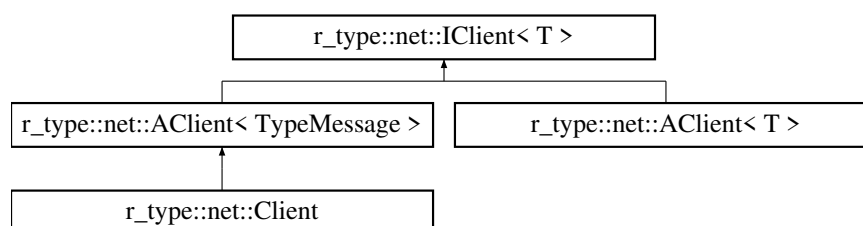
The documentation for this struct was generated from the following file:

- /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::IClient< T >](#), and [r_type::net::IClient< 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:

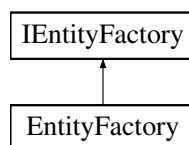
- [/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp](#)

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, int posX, int posY)=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.*
- virtual [Entity](#) [createUpdateButton](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager, [FontManager](#) &fontManager, std::string text, std::function< [IScenes](#) *([AScenes](#) *)> *onClick, std::function< std::string([GameParameters](#))> *updateText, float x, float y)=0
- virtual [Entity](#) [createTailSegment](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager)=0
- virtual [Entity](#) [createTailEnd](#) ([EntityManager](#) &entityManager, [ComponentManager](#) &componentManager)=0

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,
    int posX,
    int posY ) [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 createTailEnd()

```

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

```

Implemented in [EntityFactory](#).

6.39.4.17 createTailSegment()

```

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

```

Implemented in [EntityFactory](#).

6.39.4.18 createUpdateButton()

```

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

```

Implemented in [EntityFactory](#).

6.39.4.19 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:

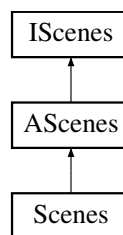
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/input_component.hpp

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:

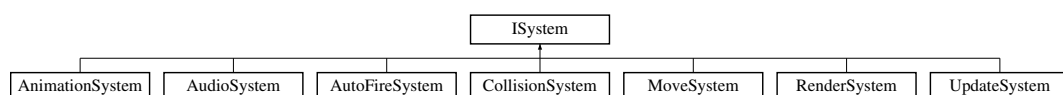
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/i_scenes.hpp](#)

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:

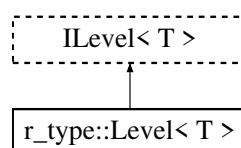
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp

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 EndOfGame \(r_type::net::AServer< T > *server, ComponentManager &componentManager, EntityManager &entityManager\) override](#)
- [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.
- [void ChangeBackground \(r_type::net::AServer< T > *server, EntityManager &entityManager, ComponentManager &componentManager\) override](#)
Changes the background in the game by removing the current background entity and creating a new one.
- [GameState GetLevel \(\) override](#)
- [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.
- [void ChangeLevel \(GameState state\) override](#)
Changes the level of the game based on the provided game state.

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 ChangeBackground()

```
template<typename T >
void r_type::Level< T >::ChangeBackground (
    r_type::net::AServer< T > * server,
    EntityManager & entityManager,
    ComponentManager & componentManager ) [inline], [override]
```

Changes the background in the game by removing the current background entity and creating a new one.

This function sends messages to all clients to destroy the current background entity and create a new one.

Template Parameters

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

Parameters

<code>server</code>	Pointer to the server instance.
<code>entityManager</code>	Reference to the EntityManager instance.
<code>componentManager</code>	Reference to the ComponentManager instance.

6.44.3.3 `ChangeLevel()`

```
template<typename T >
void r_type::Level< T >::ChangeLevel (
    GameState state ) [inline], [override]
```

Changes the level of the game based on the provided game state.

This function changes the level of the game based on the provided game state.

Parameters

<code>state</code>	The game state to change the level to.
--------------------	--

6.44.3.4 `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

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

Parameters

<code>server</code>	Pointer to the server instance.
---------------------	---------------------------------

Parameters

<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.

6.44.3.5 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.

6.44.3.6 EndOfGame()

```
template<typename T >
void r_type::Level< T >::EndOfGame (
    r_type::net::AServer< T > * server,
    ComponentManager & componentManager,
    EntityManager & entityManager ) [inline], [override]
```

6.44.3.7 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.

6.44.3.8 `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

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

Returns

[EntityInformation](#)

6.44.3.9 `GetLevel()`

```

template<typename T >
GameState r\_type::Level< T >::GetLevel ( ) [inline], [override]

```

6.44.3.10 `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.

6.44.3.11 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

<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.

6.44.3.12 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.

6.44.3.13 `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.

6.44.3.14 `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.

6.44.3.15 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.
-----------------------	--

6.44.3.16 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.

6.44.3.17 SpawnEntity()

```
template<typename T >
void r_type::Level< T >::SpawnEntity (
    r_type::net::AServer< T > * server,
    EntityManager & entityManager,
    ComponentManager & componentManager,
    int nbrOfEnemy,
    EntityFactory::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., <code>BasicMonster</code> , <code>ShooterEnemy</code>).

6.44.3.18 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:

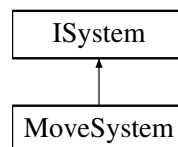
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp

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.

Public Member Functions inherited from [ISystem](#)

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

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:

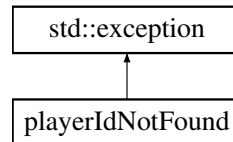
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/player_component.hpp](#)

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:

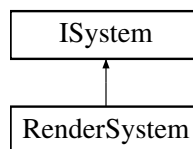
- </home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/rectangleShapeComponent.hpp>

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](#).

Public Member Functions inherited from [ISystem](#)

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

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:

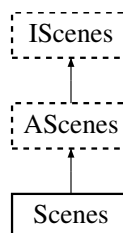
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Systems/render_system.hpp](#)
- [/home/runner/work/R-Type/R-Type/ECS/Src/Systems/render_system.cpp](#)

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 [difficultyChoicesCustomization](#) ()
- 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 [TransitionLevel](#) ()
- void [HandleTransitionLevelMessage](#) (r_type::net::Message< TypeMessage > &msg, [ComponentManager](#) &componentManager, [TextureManager](#) &textureManager)
- void [run](#) ()

Public Member Functions inherited from [AScenes](#)

- [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](#) ([GameParameters](#) const mode)
Set the Game Mode object.
- [GameParameters](#) [getGameMode](#) () const
Get 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.
- void [SetPlayerReady](#) (bool ready)
Sets the player ready status.
- bool [GetPlayerReady](#) () const
Retrieves the player ready status.

Public Member Functions inherited from [IScenes](#)

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

Public Attributes

- sf::RenderWindow [_window](#)
- r_type::net::Client [_networkClient](#)

Public Attributes inherited from [AScenes](#)

- 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.

Additional Inherited Members

Public Types inherited from [AScenes](#)

- enum class [Scene](#) {
[MAIN_MENU](#) , [GAME_LOOP](#) , [SETTINGS_MENU](#) , [IN_GAME_MENU](#) ,
[CHOOSE_DIFFICULTY](#) , [CUSTOM_DIFFICULTY](#) , [TRANSITION_LEVEL](#) , [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.

Protected Attributes inherited from [AScenes](#)

- `GameParameters _currentGameMode`
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 _displayKeyBindsChoice = 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.
- `bool _playerReady = false`

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

<code>window</code>	
---------------------	--

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 difficultyChoicesCustomization()

```
void Scenes::difficultyChoicesCustomization ( )
```

6.57.3.3 gameLoop()

```
void Scenes::gameLoop ( ) [virtual]
```

displays the main game loop, creates all the necessary entities

Implements [IScenes](#).

6.57.3.4 getRenderWindow()

```
sf::RenderWindow * Scenes::getRenderWindow ( ) [inline], [virtual]
```

Get the RenderWindow object.

Returns

sf::RenderWindow*

Implements [IScenes](#).

6.57.3.5 HandleMessage()

```
void Scenes::HandleMessage (
    r_type::net::Message< TypeMessage > & msg,
    ComponentManager & componentManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    std::shared_ptr< AudioSystem > & audioSystem )
```

6.57.3.6 HandleTransitionLevelMessage()

```
void Scenes::HandleTransitionLevelMessage (
    r_type::net::Message< TypeMessage > & msg,
    ComponentManager & componentManager,
    TextureManager & textureManager )
```

6.57.3.7 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](#).

6.57.3.8 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](#).

6.57.3.9 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](#).

6.57.3.10 run()

```
void Scenes::run ( )
```

6.57.3.11 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](#).

6.57.3.12 shouldQuit()

```
bool Scenes::shouldQuit ( ) [inline], [virtual]
```

check if game should stop running

Returns

true

false

Implements [IScenes](#).

6.57.3.13 StopGameLoop()

```
void Scenes::StopGameLoop (
    std::shared_ptr< AudioSystem > & audioSystem )
```


6.57.3.14 TransitionLevel()

```
void Scenes::TransitionLevel ( )
```

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:

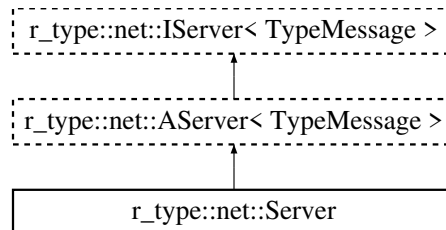
- /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/[score_component.hpp](#)

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.

Public Member Functions inherited from [r_type::net::AServer< TypeMessage >](#)

- [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](#)< [TypeMessage](#) > > client, const [Message](#)< [TypeMessage](#) > &msg)
Sends a message to a specific client if the client is connected.
- [void](#) MessageAllClients (const [Message](#)< [TypeMessage](#) > &msg, std::shared_ptr< [Connection](#)< [TypeMessage](#) > > plgnoreClient=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.
- [void RemoveBossTail \(int bossId\)](#)
- [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 \(int entityId\)](#)
- [void InitBoss \(r_type::net::AServer< \[TypeMessage\]\(#\) > *server\)](#)
- [std::shared_ptr< \[Connection\]\(#\)< \[TypeMessage\]\(#\) > > getClientById \(const std::deque< std::shared_ptr< \[Connection\]\(#\)< \[TypeMessage\]\(#\) > > &connections, uint32_t clientId\)](#)
- [virtual void OnClientValidated \(std::shared_ptr< \[Connection\]\(#\)< \[TypeMessage\]\(#\) > > 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.

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.

Protected Member Functions inherited from [r_type::net::AServer< TypeMessage >](#)

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

Additional Inherited Members

Public Attributes inherited from [r_type::net::AServer< TypeMessage >](#)

- [ThreadSafeQueue< OwnedMessage< TypeMessage > > _qMessagesIn](#)
Thread-safe queue to store incoming messages.
- [std::deque< std::shared_ptr< Connection< TypeMessage > > > _dequeConnections](#)
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](#)
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.
- [bool _endOfLevel](#)
- [bool _bossActive](#)
- [bool _bossDefeated](#)
- [bool _waitingPlayersReady](#)
- [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](#)
Number of players currently connected to the server.
- [std::chrono::system_clock::time_point _clock](#)
Stores the current time point from the system clock.
- [bool _playerConnected](#)
- [EntityInformation _background](#)
Holds information about the background entity.
- [int _port](#)
- [r_type::Level< TypeMessage > _level](#)
- [int _playerReady](#)

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

<code>cooldown</code>	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.
- `A Scenes::SpriteType` [type](#)
The type of the sprite, defined by the [A Scenes](#) 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,
    A Scenes::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

```
AScenes::SpriteType SpriteComponent::type
```

The type of the sprite, defined by the [AScenes](#) 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.
- [AScenes::SpriteType](#) `type`
Type of the sprite as defined in [AScenes::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 TailComponent Struct Reference

```
#include <tail_component.hpp>
```

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

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

6.65 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.

6.65.1 Detailed Description

A component that encapsulates an SFML text object.

This component is used to manage and render text in an SFML application.

6.65.2 Constructor & Destructor Documentation

6.65.2.1 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.

6.65.3 Member Data Documentation

6.65.3.1 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`

6.66 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.

6.66.1 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.

6.66.2 Member Data Documentation

6.66.2.1 `categoryIds`

```
TextDataComponent::categoryIds = {0}
```

Array of category IDs associated with the text.

6.66.2.2 `categorySize`

```
TextDataComponent::categorySize = 0
```

Number of categories available.

6.66.2.3 `categoryTexts`

```
TextDataComponent::categoryTexts
```

Array of GameText objects representing the text for each category.

6.66.2.4 `charSize`

```
TextDataComponent::charSize = 0
```

Size of the characters to be rendered.

6.66.2.5 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`

6.67 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.

6.67.1 Member Function Documentation

6.67.1.1 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

<code>failedToLoadTexture</code>	If the texture fails to load from the file path.
----------------------------------	--

Parameters

<code>filePath</code>	The file path of the texture to retrieve.
-----------------------	---

Returns

sf::Texture& A reference to the retrieved texture.

6.67.1.2 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.
-----------------	--

6.67.2 Member Data Documentation**6.67.2.1 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](#)

6.68 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.

6.68.1 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.

6.68.2 Member Data Documentation

6.68.2.1 lives

```
uint32_t UIEntityInformation::lives = 0
```

Number of lives the UI entity has.

6.68.2.2 score

```
uint32_t UIEntityInformation::score = 0
```

Score associated with the UI entity.

6.68.2.3 spriteData

```
SpriteDataComponent UIEntityInformation::spriteData
```

Data related to the sprite of the UI entity.

6.68.2.4 textData

```
TextDataComponent UIEntityInformation::textData
```

Data related to the text of the UI entity.

6.68.2.5 uniqueID

```
uint32_t UIEntityInformation::uniqueID = 0
```

Unique identifier for the UI entity.

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

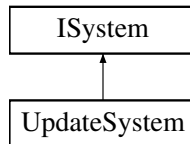
- [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/entity_struct.hpp](#)

6.69 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.

Public Member Functions inherited from [ISystem](#)

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

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.

6.69.1 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

<i>window</i>	Reference to the SFML RenderWindow object.
<i>componentManager</i>	Reference to the ComponentManager object.
<i>entityManager</i>	Reference to the EntityManager object.

6.69.2 Constructor & Destructor Documentation

6.69.2.1 UpdateSystem()

```
UpdateSystem::UpdateSystem (
    sf::RenderWindow & window,
    ComponentManager & componentManager,
    EntityManager & entityManager ) [inline]
```

Manages the update logic for entities within the ECS framework.

Parameters

<i>window</i>	Reference to the SFML RenderWindow used for rendering.
<i>componentManager</i>	Reference to the ComponentManager that handles components.
<i>entityManager</i>	Reference to the EntityManager that handles entities.

6.69.3 Member Function Documentation

6.69.3.1 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.

6.69.4 Member Data Documentation

6.69.4.1 _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).

6.69.4.2 `_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.

6.69.4.3 `_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](#)

6.70 UpdateTextComponent Struct Reference

```
#include <update_text_component.hpp>
```

Public Member Functions

- [UpdateTextComponent](#) (std::function< std::string(GameParameters)> updateTextFunction)

Public Attributes

- std::function< std::string(GameParameters)> [updateText](#)

6.70.1 Constructor & Destructor Documentation

6.70.1.1 UpdateTextComponent()

```
UpdateTextComponent::UpdateTextComponent (
    std::function< std::string(GameParameters)> updateTextFunction ) [inline]
```

6.70.2 Member Data Documentation

6.70.2.1 updateText

```
std::function<std::string(GameParameters)> UpdateTextComponent::updateText
```

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

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

6.71 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.

6.71.1 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.

6.71.2 Member Data Documentation

6.71.2.1 [x](#)

VelocityComponent::x

The velocity along the x-axis.

6.71.2.2 [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](#)

6.72 vf2d Struct Reference

Represents a 2D vector with x and y coordinates.

```
#include <macros.hpp>
```

Public Attributes

- float `x` = 0
- float `y` = 0

6.72.1 Detailed Description

Represents a 2D vector with x and y coordinates.

6.72.2 Member Data Documentation

6.72.2.1 `x`

```
float vf2d::x = 0
```

6.72.2.2 `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`

6.73 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 mainmenu.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** mainmenu
00006 */
00007
00008 #pragma once
00009
00010 #include <SFML/Graphics.hpp>
00011 #include <r_type_client.hpp>
00012
00013 int MainMenu(sf::RenderWindow *window, Rtype *rtype);
```

7.3 /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

- namespace `r_type`
- namespace `r_type::net`

7.4 a_client.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** netClient
00006 */
00007
00008 #pragma once
00009
00010 #include <Components/component_manager.hpp>
00011 #include <Components/components.hpp>
00012 #include <Net/i_client.hpp>
00013 #include <SFML/Graphics.hpp>
00014 #include <entity_struct.hpp>
00015 #include <font_manager.hpp>
00016 #include <texture_manager.hpp>
00017 #include <unordered_map>
00018
00019 namespace r_type {
00020 namespace net {
00021 template <typename T> class AClient : virtual public IClient<T> {
00022 public:
00023     AClient() { m_connection = nullptr; }
00024
00025     virtual ~AClient() { Disconnect(); }
00026
00027 public:
00035     bool Connect(const std::string &host, const uint16_t port)
00036     {
00037         try {
00038             asio::ip::udp::endpoint remote_endpoint =
00039                 asio::ip::udp::endpoint(asio::ip::make_address(host), port);
00040             // std::cout << "Remote endpoint: " << remote_endpoint << std::endl;
00041
00042             asio::ip::udp::socket socket(
00043                 m_context, asio::ip::udp::endpoint(asio::ip::udp::v4(), 0));
00044             m_connection = std::make_unique<Connection<T>>(Connection<T>::owner::client, m_context,
00045                 std::move(socket), std::move(remote_endpoint), m_qMessagesIn);
00046             m_connection->ConnectToServer();
00047         }
```



```

00048         // std::cout << "Connection: " << *(m_connection.get()) << std::endl;
00049
00050         thrContext = std::thread([this]() { m_context.run(); });
00051     } catch (std::exception &e) {
00052         std::cerr << "Client Exception: " << e.what() << std::endl;
00053         return false;
00054     }
00055     return true;
00056 }
00057
00065 void Disconnect()
00066 {
00067     if (IsConnected()) {
00068         m_connection->Disconnect();
00069     }
00070
00071     m_context.stop();
00072     if (thrContext.joinable())
00073         thrContext.join();
00074
00075     m_connection.release();
00076 }
00077
00084 bool IsConnected()
00085 {
00086     if (m_connection)
00087         return m_connection->IsConnected();
00088     else
00089         return false;
00090 }
00091
00092 public:
00098 void Send(const Message<T> &msg)
00099 {
00100     if (IsConnected())
00101         m_connection->Send(msg);
00102 }
00103
00109 ThreadSafeQueue<OwnedMessage<T> &Incoming() { return m_qMessagesIn; }
00110
00111 const std::unique_ptr<Connection<T> &getConnection() { return m_connection; }
00112
00113 void setPlayerId(uint32_t id) { playerId = id; }
00114 uint32_t getPlayerId() { return playerId; }
00115
00116 void setWindowSize(sf::Vector2u size) { windowSize = size; }
00117 sf::Vector2u getWindowSize() { return windowSize; }
00118
00119 protected:
00120     asio::io_context m_context;
00121     std::thread thrContext;
00122     std::unique_ptr<Connection<T> m_connection;
00123
00124 private:
00125     ThreadSafeQueue<OwnedMessage<T> m_qMessagesIn;
00126     uint32_t playerId = 0;
00127     sf::Vector2u windowSize;
00128 };
00129 } // namespace net
00130 } // namespace r_type

```

7.5 /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 <fstream>
#include <iostream>

```

Classes

- class [r_type::net::Client](#)

Namespaces

- namespace `r_type`
- namespace `r_type::net`

7.6 client.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** simpleClient
00006 */
00007
00008 #pragma once
00009
00010 #include <Net/a_client.hpp>
00011 #include <SFML/Graphics.hpp>
00012 #include <fstream>
00013 #include <iostream>
00014
00015 namespace r_type {
00016     namespace net {
00017         class Client : virtual public r_type::net::AClient<TypeMessage> {
00018             public:
00023                 void PingServer()
00024                 {
00025                     r_type::net::Message<TypeMessage> msg;
00026                     msg.header.id = TypeMessage::ServerPing;
00027
00028                     std::chrono::system_clock::time_point timeNow = std::chrono::system_clock::now();
00029
00030                     msg << timeNow;
00031                     Send(msg);
00032                 }
00033
00038                 void MessageAll()
00039                 {
00040                     r_type::net::Message<TypeMessage> msg;
00041                     msg.header.id = TypeMessage::MessageAll;
00042                     Send(msg);
00043                 }
00044
00045                 sf::Vector2u initInfoBar(UIEntityInformation entity, ComponentManager &componentManager,
00046                                         TextureManager &textureManager, FontManager &fontManager, sf::Vector2u windowSize)
00047                 {
00048                     float windowWidth = static_cast<float>(windowSize.x);
00049                     float windowHeight = static_cast<float>(windowSize.y);
00050
00051                     sf::Texture &texture =
00052                         textureManager.getTexture(SpriteFactory(entity.spriteData.spritePath));
00053                     float desiredWidth = windowWidth;
00054                     float desiredHeight = windowHeight * 0.10f;
00055                     sf::Vector2f scale(
00056                         desiredWidth / texture.getSize().x, desiredHeight / texture.getSize().y);
00057                     SpriteComponent spriteComponent(texture, 0, 0, scale, entity.spriteData.type);
00058                     componentManager.addComponent<SpriteComponent>(entity.uniqueID, spriteComponent);
00059
00060                     if (auto spriteEntity = componentManager.getComponent<SpriteComponent>(entity.uniqueID)) {
00061                         spriteEntity.value()->sprite.setPosition(windowWidth / 2, windowHeight * 0.95f);
00062                         sf::Font &font = fontManager.getFont(FontFactory(entity.textData.fontPath));
00063
00064                         sf::FloatRect spriteBounds = spriteEntity.value()->sprite.getLocalBounds();
00065                         float barWidth = spriteBounds.width * scale.x;
00066                         float barHeight = spriteBounds.height * scale.y;
00067                         float barPosX = spriteEntity.value()->sprite.getPosition().x;
00068                         float barPosY = spriteEntity.value()->sprite.getPosition().y;
00069
00070                         for (size_t i = 0; i < entity.textData.categorySize; i++) {
00071                             std::string displayText = GameTextFactory(entity.textData.categoryTexts[i]);
00072                             TextComponent textComponent(font, displayText, 0, 0, entity.textData.charSize);
00073                             componentManager.addComponent<TextComponent>(
00074                                 entity.textData.categoryIds[i], textComponent);
00075
00076                             if (auto textComponent = componentManager.getComponent<TextComponent>(
00077                                 entity.textData.categoryIds[i])) {
00078                                 sf::FloatRect textBounds = textComponent.value()->text.getLocalBounds();
00079
00080                                 float posX = barPosX;
```

```

00081         if (entity.textData.categoryTexts[i] == GameText::Lives) {
00082             posX -= (barWidth / 4);
00083             displayText += std::to_string(entity.lives);
00084         } else if (entity.textData.categoryTexts[i] == GameText::Score) {
00085             posX += (barWidth / 4);
00086             displayText += std::to_string(entity.score);
00087         }
00088         float posY =
00089             barPosY - (barHeight / 2) + (barHeight / 2) - (textBounds.height / 2);
00090
00091         textComponent.value()->text.setPosition(posX, posY);
00092         textComponent.value()->text.setString(displayText);
00093     }
00094 }
00095 sf::Vector2u newWindowSize = {
00096     windowSize.x, static_cast<unsigned int>(windowHeight - desiredHeight));
00097 return newWindowSize;
00098 }
00099 return windowSize;
00100 }
00101
00102 void updateInfoBar(UIEntityInformation entity, ComponentManager &componentManager)
00103 {
00104     for (size_t i = 0; i < entity.textData.categorySize; i++) {
00105         if (auto textComponent =
00106             componentManager.getComponent<TextComponent>(entity.textData.categoryIds[i])) {
00107             std::string displayText = GameTextFactory(entity.textData.categoryTexts[i]);
00108             if (entity.textData.categoryTexts[i] == GameText::Lives) {
00109                 displayText += std::to_string(entity.lives);
00110             } else if (entity.textData.categoryTexts[i] == GameText::Score) {
00111                 displayText += std::to_string(entity.score);
00112             }
00113             textComponent.value()->text.setString(displayText);
00114         }
00115     }
00116 }
00117
00118 void addEntity(EntityInformation entity, ComponentManager &componentManager,
00119     TextureManager &textureManager, sf::Vector2u windowSize)
00120 {
00121     if (entity.spriteData.type == AScenes::SpriteType::UI) {
00122         return;
00123     }
00124     float posX = windowSize.x * (entity.vPos.x / 100.0f);
00125     float posY = windowSize.y * (entity.vPos.y / 100.0f);
00126     float scaleX = (entity.ratio.x * windowSize.x) / entity.animationComponent.dimension.x;
00127     float scaleY = (entity.ratio.y * windowSize.y) / entity.animationComponent.dimension.y;
00128     sf::Texture &texture =
00129         textureManager.getTexture(SpriteFactory(entity.spriteData.spritePath));
00130     sf::Vector2f scale(scaleX, scaleY);
00131     sf::IntRect rect(entity.animationComponent.offset.x, entity.animationComponent.offset.y,
00132         entity.animationComponent.dimension.x, entity.animationComponent.dimension.y);
00133     SpriteComponent sprite(texture, posX, posY, scale, entity.spriteData.type, rect);
00134     componentManager.addComponent<SpriteComponent>(entity.uniqueID, sprite);
00135 }
00136
00137 void removeEntity(int entityId, ComponentManager &componentManager)
00138 {
00139     componentManager.removeEntityFromComponent<SpriteComponent>(entityId);
00140 }
00141
00142 void moveEntity(
00143     uint32_t id, vf2d newPos, ComponentManager &componentManager, sf::Vector2u windowSize)
00144 {
00145     auto spriteEntity = componentManager.getComponent<SpriteComponent>(id);
00146     if (spriteEntity) {
00147         float posX = windowSize.x * (newPos.x / 100.0f);
00148         float posY = windowSize.y * (newPos.y / 100.0f);
00149         spriteEntity.value()->sprite.setPosition(posX, posY);
00150     } else {
00151         std::cerr << "Entity not found, id: " << id << std::endl;
00152     }
00153 }
00154
00155 void animateEntity(int entityId, AnimationComponent rect, ComponentManager &componentManager)
00156 {
00157     if (auto spritesOpt = componentManager.getComponentMap<SpriteComponent>()) {
00158         auto &sprites = **spritesOpt;
00159         auto entitySpriteIt = sprites.find(entityId);
00160         if (entitySpriteIt != sprites.end()) {
00161             auto &spriteComponent = entitySpriteIt->second;
00162             if (auto entitySprite = std::any_cast<SpriteComponent>(&spriteComponent)) {
00163                 sf::IntRect newRect(
00164                     rect.offset.x, rect.offset.y, rect.dimension.x, rect.dimension.y);
00165                 entitySprite->sprite.setTextureRect(newRect);
00166             }
00167         }
00168     }
}

```

```

00168     }
00169 }
00170
00171 void displayEndOfGame(ComponentManager &componentManager, TextureManager &textureManager,
00172 FontManager &fontManager, sf::Vector2u windowSize)
00173 {
00174     float yPos = 50;
00175     float xPos = windowSize.x / 2;
00176     sf::Font &font = fontManager.getFont(FontFactory(FontPath::MAIN));
00177     const std::string winText = "You Win!";
00178     TextComponent textComponent(font, winText, xPos, yPos, 80);
00179     componentManager.addComponent<TextComponent>(3, textComponent);
00180
00181     std::vector<std::string> scores;
00182     std::ifstream file("GameScores/scores.txt");
00183     if (file.is_open()) {
00184         std::string line;
00185         while (std::getline(file, line)) {
00186             scores.push_back(line);
00187         }
00188         file.close();
00189     } else {
00190         throw failedToOpenFile();
00191     }
00192     int id = 4;
00193     for (const auto &score : scores) {
00194         yPos += 80;
00195         TextComponent textComponent(font, score, xPos, yPos, 50);
00196         componentManager.addComponent<TextComponent>(id, textComponent);
00197         id++;
00198     }
00199 }
00200 };
00201 } // namespace net
00202 } // namespace r_type

```

7.7 /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

- namespace [r_type](#)
- namespace [r_type::net](#)

7.8 i_client.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** netClient
00006  */
00007
00008 #pragma once

```

```

00009
00010 #include <Net/common.hpp>
00011 #include <Net/connection.hpp>
00012 #include <Net/thread_safe_queue.hpp>
00013
00014 namespace r_type {
00015 namespace net {
00016 template <typename T> class IClient {
00017 public:
00018     IClient() {}
00019
00020     virtual ~IClient() {}
00021
00022 public:
00031     virtual bool Connect(const std::string &host, const uint16_t port) = 0;
00032
00040     virtual void Disconnect() = 0;
00041
00048     virtual bool IsConnected() = 0;
00049
00050 public:
00056     virtual void Send(const Message<T> &msg) = 0;
00057
00063     virtual ThreadSafeQueue<OwnedMessage<T> &Incoming() = 0;
00064 };
00065 } // namespace net
00066 } // namespace r_type

```

7.9 /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)

7.9.1 Function Documentation

7.9.1.1 keyToString()

```

std::string keyToString (
    sf::Keyboard::Key key )

```

7.10 scenes.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** scenes
00006  */
00007
00008  #pragma once
00009
00010  #include <Entities/entity.hpp>
00011  #include <Net/client.hpp>
00012  #include <SFML/Graphics.hpp>
00013  #include <Systems/systems.hpp>
00014  #include <a_scenes.hpp>
00015  #include <memory>
00016  #include <vector>
00017
00018  std::string keyToString(sf::Keyboard::Key key);
00027  class Scenes : virtual public AScenes {
00028
00029  public:
00035      Scenes(std::string ip, int port);
00036
00041      ~Scenes() = default;
00042
00047      void mainMenu();
00048
00053      void gameLoop();
00054
00055      void HandleMessage(r_type::net::Message<TypeMessage> &msg, ComponentManager &componentManager,
00056                        TextureManager &textureManager, FontManager &fontManager,
00057                        std::shared_ptr<AudioSystem> &audioSystem);
00058
00059      void StopGameLoop(std::shared_ptr<AudioSystem> &audioSystem);
00060
00065      void settingsMenu();
00066
00071      void inGameMenu();
00072
00077      void difficultyChoices();
00078
00079      void difficultyChoicesCustomization();
00080
00086      void render();
00087
00094      bool shouldQuit() { return _currentScene == Scene::EXIT; }
00095
00101      sf::RenderWindow *getRenderWindow() { return &_amp;window; }
00102
00103      void TransitionLevel();
00104
00105      void HandleTransitionLevelMessage(r_type::net::Message<TypeMessage> &msg,
00106                                       ComponentManager &componentManager, TextureManager &textureManager);
00107
00108      void run();
00109
00110      sf::RenderWindow _window;
00111
00112      r_type::net::Client _networkClient;
00113  };

```

7.11 /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)

7.11.1 Function Documentation

7.11.1.1 keyToString()

```
std::string keyToString (
    sf::Keyboard::Key key )
```

7.12 /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.

7.12.1 Function Documentation

7.12.1.1 isValidIPv4()

```
static bool isValidIPv4 (
    const std::string & ip ) [static]
```

7.12.1.2 isValidPort()

```
static bool isValidPort (
    const std::string & portStr ) [static]
```

7.12.1.3 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

7.13 /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.

7.13.1 Function Documentation

7.13.1.1 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

<i>portStr</i>	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.

7.13.1.2 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.

7.13.1.3 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.
---------------	--

7.13.2 Variable Documentation

7.13.2.1 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.

7.14 /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)

7.14.1 Function Documentation

7.14.1.1 createDaltonismChoiceButtons()

```
void createDaltonismChoiceButtons (
    std::vector< std::shared_ptr< Entity > > & buttons,
    ComponentManager & componentManager,
    EntityManager & entityManager,
    TextureManager & textureManager,
    FontManager & fontManager,
    EntityFactory & entityFactory )
```

7.14.1.2 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 )
```

7.14.1.3 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 ComponentManager to access components of entities.
<i>_window</i>	Pointer to the RenderWindow where events are polled from.
<i>buttons</i>	Vector of shared pointers to Entity objects representing buttons.

7.14.1.4 reloadFilter()

```
void reloadFilter (
    sf::RectangleShape & rectangle,
    AScenes::DaltonismMode mode )
```

7.14.1.5 waitForKey()

```
sf::Keyboard::Key waitForKey (
    sf::RenderWindow * _window )
```

7.15 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/a_↵ scenes.hpp File Reference

```
#include "Entities/entity.hpp"
#include "i_scenes.hpp"
#include <game_struct.hpp>
#include <memory>
```

Classes

- class [AScenes](#)

7.16 a_scenes.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** a_scenes
00006  */
00007
00008  #pragma once
00009
00010  #include "Entities/entity.hpp"
00011  #include "i_scenes.hpp"
00012  #include <game_struct.hpp>
00013  #include <memory>
00014
00021  class AScenes : virtual public IScenes {
00022  public:
00023      AScenes(std::string ip, int port);
00024      ~AScenes() = default;
00025
00047      enum class Scene
00048      {
00049          MAIN_MENU,
00050          GAME_LOOP,
00051          SETTINGS_MENU,
00052          IN_GAME_MENU,
00053          CHOOSE_DIFFICULTY,
00054          CUSTOM_DIFFICULTY,
00055          TRANSITION_LEVEL,
00056          EXIT
00057      };
00058
00069      enum class GameMode
00070      {
00071          EASY,
00072          MEDIUM,
00073          HARD
00074      };
00075
00094      enum class DaltonismMode
00095      {
00096          NORMAL,
00097          TRITANOPIA,
00098          DEUTERANOPIA,
00099          PROTANOPIA
00100      };
00101
00116      enum class Actions
00117      {
00118          UP,
00119          DOWN,
00120          LEFT,
00121          RIGHT,
00122          FIRE,
00123          PAUSE,
00124          QUIT
00125      };
00126
00139      enum class SpriteType
00140      {
00141          BACKGROUND,
00142          PLAYER,
00143          ALLY,
00144          ENEMY,
00145          FILTER,
00146          WEAPON,
00147          POWER_UP,
00148          UI,
00149          OTHER
00150      };
00151
00168      std::map<Actions, sf::Keyboard::Key> keyBinds = {{Actions::UP, sf::Keyboard::Key::Up},
00169      {Actions::DOWN, sf::Keyboard::Key::Down}, {Actions::LEFT, sf::Keyboard::Key::Left},
00170      {Actions::RIGHT, sf::Keyboard::Key::Right}, {Actions::FIRE, sf::Keyboard::Key::Space},
00171      {Actions::PAUSE, sf::Keyboard::Key::Escape}, {Actions::QUIT, sf::Keyboard::Key::Q}};
00172
00178      void setScene(Scene scene);
00179
00185      AScenes::Scene getPreviousScene();
00186
00192      DaltonismMode getDaltonism() const { return _currentDaltonismMode; };
00193
00199      void setDaltonism(DaltonismMode const mode);

```

```

00200
00206     void setGameMode(GameParameters const mode);
00207
00213     GameParameters getGameMode() const { return _currentGameMode; };
00214
00224     void setDisplayDaltonismChoice(bool const displayDaltonismChoice);
00225
00231     bool getDisplayDaltonismChoice() const;
00232
00242     void setDisplayGameModeChoice(bool const displayGameModeChoice);
00243
00252     bool getDisplayGameModeChoice() const;
00253
00262     void setDisplayKeyBindsChoice(bool const displayKeyBindsChoice);
00263
00269     bool getDisplayKeyBindsChoice() const;
00270
00279     std::vector<std::shared_ptr<Entity>> buttons;
00287     std::shared_ptr<Entity> filter;
00295     void setIp(std::string ip);
00303     void setPort(int port);
00304
00312     std::string getIp() const;
00318     int getPort() const;
00319
00327     void SetPlayerReady(bool ready) { _playerReady = ready; };
00328
00336     bool GetPlayerReady() const { return _playerReady; };
00337
00338 protected:
00345     GameParameters _currentGameMode;
00358     DaltonismMode _currentDaltonismMode = DaltonismMode::NORMAL;
00365     Scene _currentScene = Scene::MAIN_MENU;
00372     Scene _previousScene = Scene::MAIN_MENU;
00376     bool _displayDaltonismChoice = false;
00380     bool _displayGameModeChoice = false;
00384     bool _displayKeyBindsChoice = false;
00385
00393     std::string _ip;
00401     int _port;
00402
00403     bool _playerReady = false;
00404 };

```

7.17 /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.

7.18 audio_manager.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** audio_manager

```

```

00006 */
00007
00008 #pragma once
00009
00010 #include "error_handling.hpp"
00011 #include <SFML/Audio.hpp>
00012 #include <memory>
00013 #include <string>
00014 #include <unordered_map>
00015
00045 class AudioManager {
00046 public:
00058     sf::SoundBuffer &getSoundBuffer(const std::string &filePath)
00059     {
00060         // Check if sound buffer is already cached
00061         auto it = soundBuffers.find(filePath);
00062         if (it != soundBuffers.end()) {
00063             return *it->second;
00064         }
00065
00066         // Load and cache the sound buffer
00067         auto buffer = std::make_shared<sf::SoundBuffer>();
00068         if (!buffer->loadFromFile(filePath)) {
00069             throw std::runtime_error("Failed to load sound from " + filePath);
00070         }
00071         soundBuffers[filePath] = buffer;
00072         return *buffer;
00073     }
00074
00075 private:
00085     std::unordered_map<std::string, std::shared_ptr<sf::SoundBuffer>> soundBuffers;
00086 };

```

7.19 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_component.hpp File Reference ↩

Defines the [AllyComponent](#) structure.

Classes

- struct [AllyComponent](#)

7.19.1 Detailed Description

Defines the [AllyComponent](#) structure.

The [AllyComponent](#) is used to mark entities as allies within the ECS framework.

7.20 ally_component.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** ally_component
00006 */
00007
00008 #pragma once
00009
00016 struct AllyComponent {};

```

7.21 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ally_missile_component.hpp File Reference

Defines the [AllyMissileComponent](#) structure.

Classes

- struct [AllyMissileComponent](#)

7.21.1 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.

7.22 ally_missile_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** ally_missile_component
00006 */
00007
00008 #pragma once
00009
00018 struct AllyMissileComponent {};
```

7.23 /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](#).

7.23.1 Function Documentation

7.23.1.1 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.

7.24 [animation_component.hpp](#)

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** velocity_component
00006 */
00007
00008 #pragma once
00009
00010 #include <macros.hpp>
00011
00031 struct AnimationComponent {
00032     vf2d offset;
00033     vf2d dimension;
00034
00035     AnimationComponent(vf2d _offset, vf2d _dimension) : offset(_offset), dimension(_dimension) {}
00036 };
00037
00047 bool operator!=(AnimationComponent animation, AnimationComponent other);

```

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

Defines the [BackgroundComponent](#) structure.

Classes

- struct [BackgroundComponent](#)

7.25.1 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.

7.26 background_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** background_component
00006 */
00007
00008 #pragma once
00009
00017 struct BackgroundComponent {};
```

7.27 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/basic_monster_component.hpp File Reference

Defines the [BasicMonsterComponent](#) structure.

Classes

- struct [BasicMonsterComponent](#)

7.27.1 Detailed Description

Defines the [BasicMonsterComponent](#) structure.

This component is used to represent basic monster entities in the game.

7.28 basic_monster_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** basic_monster_component
00006 */
00007
00008 #pragma once
00009
00016 struct BasicMonsterComponent {};
```

7.29 /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.

7.30 bind_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** bind_component
00006 */
00007
00008 #pragma once
00009
00010 #include "a_scenes.hpp"
00011 #include "i_scenes.hpp"
00012 #include <functional>
00013
00035 struct BindComponent {
00036     bool isHovered = false;
00037     std::function<IScenes *(AScenes *, AScenes::Actions)> bind;
00038
00039     BindComponent(std::function<IScenes *(AScenes *, AScenes::Actions)> bindFunction)
00040         : bind(bindFunction){};
00041 };
```

7.31 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/boss_component.hpp File Reference

```
#include <vector>
```

Classes

- struct [BossComponent](#)

7.32 boss_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** boss_component
00006 */
00007
00008 #pragma once
00009
00010 #include <vector>
00011
00012 struct BossComponent {
00013     std::vector<int> tailSegmentIds;
00014 };
```

7.33 /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.

7.34 component_manager.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** component_manager
00006 */
00007
00008 #pragma once
00009
00010 #include "components.hpp"
00011 #include "texture_manager.hpp"
00012 #include <any>
00013 #include <iostream>
00014 #include <memory>
00015 #include <optional>
00016 #include <typeindex>
00017 #include <unordered_map>
00018
00019 class ComponentManager {
00020 public:
00021     // Usage ex: CMName.addComponent<NameOfComponent>(entityId, arg1, arg2);
00022     template <typename ComponentType, typename... Args>
00023     void addComponent(int entityId, Args &&...args)
00024     {
00025         ComponentType component(std::forward<Args>(args)...);
00026         components[typeid(ComponentType)][entityId] =
00027             std::make_any<ComponentType>(std::move(component));
00028     }
00029
00030     template <typename ComponentType> std::optional<ComponentType *> getComponent(int entityId)
00031     {
00032         if (components.find(typeid(ComponentType)) != components.end()) {
00033             auto &entityComponents = components[typeid(ComponentType)];
00034             if (entityComponents.find(entityId) != entityComponents.end()) {
00035                 return std::any_cast<ComponentType*>(&entityComponents[entityId]);
00036             }
00037         }
00038         return std::nullopt; // Return nullopt if not found
00039     }
00040
00041     template <typename ComponentType>
00042     std::optional<std::unordered_map<int, std::any *>> getComponentMap()
00043     {
00044         auto it = components.find(typeid(ComponentType));
00045         if (it != components.end()) {
00046             return &it->second;
00047         }
00048         return std::nullopt;
00049     }
00050 }
```

```

00080     }
00081
00092     template <typename ComponentType> void removeEntityFromComponent(int entityId)
00093     {
00094         auto it = components.find(typeid(ComponentType));
00095         if (it != components.end()) {
00096             auto &entityComponents = it->second;
00097             entityComponents.erase(entityId);
00098         }
00099     }
00100
00104     void removeAllComponents()
00105     {
00106         if (components.empty())
00107             return;
00108         std::cout << "Removing all components" << std::endl;
00109         components.clear();
00110     }
00111
00120     void removeEntityFromAllComponents(int entityId)
00121     {
00122         for (auto &component : components) {
00123             component.second.erase(entityId);
00124         }
00125     }
00126
00127 private:
00128     // unordered map of <componentType, it's unordered map of <entityId, it's
00129     // values for the componentType>
00138     std::unordered_map<std::type_index, std::unordered_map<int, std::any>> components;
00139 };

```

7.35 /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 "boss_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 "tail_component.hpp"
#include "text_component.hpp"

```

```
#include "text_data_component.hpp"
#include "update_text_component.hpp"
#include "velocity_component.hpp"
#include "wall_component.hpp"
```

7.36 components.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** components
00006 */
00007
00008 #pragma once
00009
00010 #include "ally_component.hpp"
00011 #include "ally_missile_component.hpp"
00012 #include "animation_component.hpp"
00013 #include "background_component.hpp"
00014 #include "basic_monster_component.hpp"
00015 #include "bind_component.hpp"
00016 #include "boss_component.hpp"
00017 #include "enemy_component.hpp"
00018 #include "enemy_missile_component.hpp"
00019 #include "force_missile_component.hpp"
00020 #include "force_weapon_component.hpp"
00021 #include "front_component.hpp"
00022 #include "health_component.hpp"
00023 #include "hitbox_component.hpp"
00024 #include "input_component.hpp"
00025 #include "link_force_component.hpp"
00026 #include "movement_component.hpp"
00027 #include "offset_component.hpp"
00028 #include "on_click_component.hpp"
00029 #include "player_component.hpp"
00030 #include "player_missile_component.hpp"
00031 #include "position_component.hpp"
00032 #include "power_up_component.hpp"
00033 #include "rectangleShapeComponent.hpp"
00034 #include "score_component.hpp"
00035 #include "shoot_component.hpp"
00036 #include "sprite_component.hpp"
00037 #include "sprite_data_component.hpp"
00038 #include "tail_component.hpp"
00039 #include "text_component.hpp"
00040 #include "text_data_component.hpp"
00041 #include "update_text_component.hpp"
00042 #include "velocity_component.hpp"
00043 #include "wall_component.hpp"
```

7.37 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/enemy_component.hpp File Reference

Defines the [EnemyComponent](#) structure.

Classes

- struct [EnemyComponent](#)

7.37.1 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.

7.38 enemy_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** enemy_component
00006 */
00007
00008 #pragma once
00009
00019 struct EnemyComponent {};
```

7.39 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/enemy_missile_component.hpp File Reference

Defines the [EnemyMissileComponent](#) structure.

Classes

- struct [EnemyMissileComponent](#)

7.39.1 Detailed Description

Defines the [EnemyMissileComponent](#) structure.

This component is used to represent an enemy missile in the game.

7.40 enemy_missile_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** enemy_missile_component
00006 */
00007
00008 #pragma once
00009
00016 struct EnemyMissileComponent {};
```

7.41 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/force_missile_component.hpp File Reference

```
#include <cstdint>
```

Classes

- struct [ForceMissileComponent](#)
Component representing a force missile in the ECS system.

7.42 force_missile_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*  
00002 ** EPITECH PROJECT, 2024  
00003 ** R-Type  
00004 ** File description:  
00005 ** missile_component  
00006 */  
00007  
00008 #pragma once  
00009  
00010 #include <cstdint>  
00011  
00022 struct ForceMissileComponent {  
00023     uint32_t forceId;  
00024 };
```

7.43 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/force_weapon_component.hpp File Reference

```
#include <cstdint>
```

Classes

- struct [ForceWeaponComponent](#)
Represents a component for a force weapon in the game.

7.44 force_weapon_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*  
00002 ** EPITECH PROJECT, 2024  
00003 ** R-Type  
00004 ** File description:  
00005 ** weapon_component  
00006 */  
00007  
00008 #pragma once  
00009  
00010 #include <cstdint>  
00011  
00036 struct ForceWeaponComponent {  
00037     uint32_t playerId;  
00038     uint32_t level;  
00039     bool attached;  
00040  
00041     ForceWeaponComponent(uint32_t _playerId, uint32_t _level, uint32_t _attached)  
00042         : playerId(_playerId), level(_level), attached(_attached)  
00043     {  
00044     }  
00045 };
```

7.45 /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.

7.46 front_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** ally_component
00006 */
00007
00008 #pragma once
00009
00010 #include <Entities/entity.hpp>
00011 #include <memory>
00012
00018 struct FrontComponent {
00019     int targetId;
00020
00021     FrontComponent(int _targetId) : targetId(_targetId) {}
00022 };
```

7.47 /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.

7.48 health_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** health_component
00006 */
00007
00008 #pragma once
00009
00022 struct HealthComponent {
00023     int lives;
00024 };
```


7.49 /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.

7.50 hitbox_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** hitbox_component
00006 */
00007
00008 #pragma once
00009
00023 struct HitboxComponent {
00024     int w;
00025     int h;
00026 };
```

7.51 /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.

7.51.1 Enumeration Type Documentation

7.51.1.1 InputType

```
enum class 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.

7.52 input_component.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** input_component
00006  */
00007
00008  #pragma once
00009
00038  enum class InputType
00039  {
00040      UP,
00041      DOWN,
00042      LEFT,
00043      RIGHT,
00044      SHOOT,
00045      QUIT,
00046      NONE
00047  };
00048
00058  struct InputComponent {
00059      InputType input;
00060  };

```

7.53 /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.

7.54 label_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** label_component
00006 */
00007
00008 #pragma once
00009
00010 #include <string>
00011
00028 struct labelComponent {
00029     std::string name;
00030     int x;
00031     int y;
00032 };
```

7.55 /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.

7.56 link_force_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** ally_component
00006 */
00007
00008 #pragma once
00009
00025 struct LinkForceComponent {
00026     int targetId;
00027
00028     LinkForceComponent(int _targetId) : targetId(_targetId) {}
00029 };
```

7.57 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/movement_component.hpp File Reference

```
#include <cstdint>
```

Classes

- struct [MovementComponent](#)

Represents a component that handles movement in the ECS system.

Enumerations

- enum class [MovementType](#) {
[WIGGLE](#) , [DIAGONAL](#) , [CIRCLE](#) , [STRAIGHT](#) ,
[SWEEPING](#) , [NONE](#) }

Enumeration of different types of movement behaviors.

7.57.1 Enumeration Type Documentation

7.57.1.1 MovementType

```
enum class 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	
SWEEPING	
NONE	

7.58 movement_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** player_component
00006 */
00007
00008 #pragma once
00009
00010 #include <cstdlib>
00011
00026 enum class MovementType
00027 {
00028     WIGGLE,
00029     DIAGONAL,
00030     CIRCLE,
00031     STRAIGHT,
00032     SWEEPING,
00033     NONE
00034 };
00035
00058 struct MovementComponent {
00059     MovementType movementType;
00060     uint32_t index;
00061     bool move;
00062
00063     MovementComponent() : movementType(MovementType::STRAIGHT), index(0), move(true) {}
00064     MovementComponent(MovementType movementType, uint32_t index, bool move)
00065         : movementType(movementType), index(index), move(move)
00066     {
00067     }
00068 };
```

7.59 /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.

7.60 offset_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** offset_component
00006 */
00007
00008 #pragma once
00009
00015 struct OffsetComponent {
00016     float offset;
00017 };
```

7.61 /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.

7.61.1 Detailed Description

Defines the [OnClickComponent](#) structure used for handling click events in the ECS system.

7.62 on_click_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** on_click_component
00006 */
00007
00008 #pragma once
00009 #include <a_scenes.hpp>
00010 #include <functional>
00011 #include <i_scenes.hpp>
00012
00037 struct OnClickComponent {
00038     bool isClicked = false;
00039     std::function<IScenes *(AScenes *)> onClick;
00040
00041     OnClickComponent(std::function<IScenes *(AScenes *)> onClickFunction)
00042         : onClick(onClickFunction) {}
00043 };
```

7.63 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/player_component.hpp File Reference

Defines the [PlayerComponent](#) structure.

Classes

- struct [PlayerComponent](#)

7.63.1 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.

7.64 player_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** player_component
00006 */
00007
00008 #pragma once
00009
00017 struct PlayerComponent {};
```

7.65 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/player_missile_component.hpp File Reference

```
#include <cstdint>
```

Classes

- struct [PlayerMissileComponent](#)

Component that represents a missile belonging to a player.

7.66 player_missile_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** missile_component
00006 */
00007
00008 #pragma once
00009
00010 #include <stdint>
00011
00021 struct PlayerMissileComponent {
00022     uint32_t playerId;
00023 };
```

7.67 /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.

7.68 position_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** position_component
00006 */
00007
00008 #pragma once
00009
00013 struct PositionComponent {
00014     float x;
00015     float y;
00016
00017     PositionComponent(float _x, float _y) : x(_x), y(_y) {}
00018 };
```

7.69 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↔ Components/power_up_component.hpp File Reference

Defines the [PowerUpComponent](#) structure.

Classes

- struct [PowerUpComponent](#)

7.69.1 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.

7.70 power_up_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** player_component
00006 */
00007
00008 #pragma once
00009
00017 struct PowerUpComponent {};
```

7.71 /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`.

7.72 rectangleShapeComponent.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** rectangleShapeComponent
00006 */
00007
00008 #pragma once
00009 #include <SFML/Graphics.hpp>
00010
00017 struct RectangleShapeComponent {
00018     sf::RectangleShape rectangleShape;
00019
00025     RectangleShapeComponent(sf::RectangleShape &rectangleShape)
00026     {
00027         this->rectangleShape = rectangleShape;
00028     }
00029 };
```


7.73 /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.

7.73.1 Detailed Description

Defines the [ScoreComponent](#) struct used to store the score of an entity.

7.74 score_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** score_component
00006  */
00007
00008 #pragma once
00009
00022 struct ScoreComponent {
00023     int score;
00024 };
```

7.75 /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.

7.76 shader_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** shader_component
00006 */
00007
00008 #pragma once
00009 #include <SFML/Graphics.hpp>
00010 #include <iostream>
00011 #include <memory>
00012
00019 struct ShaderComponent {
00025     std::shared_ptr<sf::Shader> shader;
00026
00036     ShaderComponent(std::string path)
00037     {
00038         shader = std::make_shared<sf::Shader>();
00039         if (!shader->loadFromFile(path, sf::Shader::Fragment)) {
00040             std::cerr << "Error loading shader" << std::endl;
00041         }
00042     }
00043 };
```

7.77 /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.

7.78 shoot_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** shoot_component
00006 */
00007
00008 #pragma once
00009
00010 #include <chrono>
00011
00036 struct ShootComponent {
00037     std::chrono::system_clock::time_point nextShootTime;
00038     std::chrono::milliseconds cooldownTime;
00039     bool canShoot;
00040
00041     ShootComponent(std::chrono::milliseconds cooldown)
00042         : nextShootTime(std::chrono::system_clock::now()), cooldownTime(cooldown)
00043     {
00044     }
00045 };
```

7.79 /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](#)).

7.80 sprite_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** sprite_component
00006 */
00007
00008 #pragma once
00009
00010 #include "a_scenes.hpp"
00011 #include <SFML/Graphics.hpp>
00012 #include <string>
00013
00045 struct SpriteComponent {
00046     sf::Sprite sprite;
00047     AScenes::SpriteType type;
00048     int hitboxX;
00049     int hitboxY;
00050
00051     SpriteComponent(sf::Texture &texture, const float posX, float posY, const sf::Vector2f &scale,
00052         AScenes::SpriteType typeNb, sf::IntRect rect = sf::IntRect(0, 0, 0, 0))
00053     {
00054         type = typeNb;
00055         sprite.setTexture(texture);
00056         sprite.setPosition(posX, posY);
00057         sprite.setScale(scale);
00058         if (rect != sf::IntRect(0, 0, 0, 0))
00059             sprite.setTextureRect(rect);
00060         sprite.setOrigin(sprite.getLocalBounds().width / 2, sprite.getLocalBounds().height / 2);
00061         hitboxX = rect.width;
00062         hitboxY = rect.height;
00063     }
00064 };
```

7.81 /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.

7.81.1 Function Documentation

7.81.1.1 operator<<()

```
std::ostream & operator<< (
    std::ostream & os,
    const SpriteDataComponent & spriteData )
```

Overloads the << operator to output the contents of a [SpriteDataComponent](#) to an ostream.

Parameters

<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](#).

7.82 sprite_data_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** sprite_component
00006 */
00007
00008 #pragma once
00009
00010 #include "../error_handling.hpp"
00011 #include "../sprite_path.hpp"
00012 #include "animation_component.hpp"
00013 #include "position_component.hpp"
00014 #include <SFML/Graphics.hpp>
00015 #include <a_scenes.hpp>
00016 #include <cstdlib>
00017 #include <macros.hpp>
00018 #include <string>
00019
00035 struct SpriteDataComponent {
00036     SpritePath spritePath;
00037     vf2d scale;
00038     AScenes::SpriteType type;
00039 };
00040
00048 std::ostream &operator<<(std::ostream &os, const SpriteDataComponent &spriteData);
```

7.83 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/tail_component.hpp File Reference

Classes

- struct [TailComponent](#)

7.84 tail_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** tail_component
00006 */
00007
00008 #pragma once
00009
00010 struct TailComponent {};
```

7.85 /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.

7.86 text_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** text_component
00006 */
00007
00008 #pragma once
00009
00010 #include <SFML/Graphics.hpp>
00011
00031 struct TextComponent {
00032     sf::Text text;
00033
00034     TextComponent(sf::Font &font, const std::string &string, float posX, float posY, int size = 30)
00035     {
00036         text.setCharacterSize(size);
00037         text.setFont(font);
00038         text.setString(string);
00039         text.setPosition(posX, posY);
00040         text.setFillColor(sf::Color::White);
00041         text.setStyle(sf::Text::Bold);
00042     }
00043 };
```

7.87 /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.

7.88 text_data_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** text_data_component
00006 */
00007
00008 #pragma once
00009
00010 #include "../font_path.hpp"
00011 #include "../game_text.hpp"
00012
00035 struct TextDataComponent {
00036     FontPath fontPath;
00037     uint32_t charSize = 0;
00038     uint32_t categoryIds[5] = {0};
00039     GameText categoryTexts[5];
00040     uint32_t categorySize = 0;
00041 };
```

7.89 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/↵ Components/update_text_component.hpp File Reference

```
#include <a_scenes.hpp>
#include <functional>
#include <i_scenes.hpp>
#include <string>
```

Classes

- struct [UpdateTextComponent](#)

7.90 update_text_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** on_click_component
00006 */
00007
00008 #pragma once
00009 #include <a_scenes.hpp>
00010 #include <functional>
00011 #include <i_scenes.hpp>
00012 #include <string>
00013
00014 struct UpdateTextComponent {
00015     std::function<std::string(GameParameters)> updateText;
00016
00017     UpdateTextComponent (std::function<std::string(GameParameters)> updateTextFunction)
00018         : updateText (updateTextFunction) {};
00019 };
```

7.91 /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.

7.92 velocity_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** velocity_component
00006 */
00007
00008 #pragma once
00009
00023 struct VelocityComponent {
00024     float x;
00025     float y;
00026 };
```

7.93 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/wall_component.hpp File Reference ↩

Defines the [WallComponent](#) structure.

Classes

- struct [WallComponent](#)

7.93.1 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](#)).

7.94 wall_component.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** enemy_missile_component
00006 */
00007
00008 #pragma once
00009
00017 struct WallComponent {};
```

7.95 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/creatable_client_object.hpp File Reference

```
#include <stdint>
```

Enumerations

- enum class [CreatableClientObject](#) : uint32_t { [PLAYERMISSILE](#) , [NONE](#) }

Enum representing the types of client objects that can be created.

7.95.1 Enumeration Type Documentation

7.95.1.1 CreatableClientObject

```
enum class 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.

7.96 creatable_client_object.hpp

[Go to the documentation of this file.](#)

```
00001
00002 /*
00003 ** EPITECH PROJECT, 2024
00004 ** R-Type
00005 ** File description:
00006 ** creatable_client_object
00007 */
00008
00009 #pragma once
00010
00011 #include <stdint>
00012
00026 enum class CreatableClientObject : uint32_t
00027 {
00028     PLAYERMISSILE,
00029     NONE
00030 };
```

7.97 /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.

7.98 entity.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** entity
00006 */
00007
00008 #pragma once
00009
00018 class Entity {
00019     public:
00025         explicit Entity(int id) : _id(id) {}
00026
00032         int getId() const { return _id; }
00033
00034     private:
00038         int _id;
00039 };
```

7.99 /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.hpp>
```

Classes

- class [EntityFactory](#)

A factory class for creating various types of entities.

7.100 entity_factory.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** entity_factory
00006 */
00007
00008 #pragma once
00009
00010 #include "a_scenes.hpp"
00011 #include "i_entity_factory.hpp"
00012 #include "i_scenes.hpp"
00013 #include <functional>
00014 #include <game_struct.hpp>
00015
00025 class EntityFactory : public IEntityFactory {
00026 public:
00034     Entity createBackgroundLevelOne(
00035         EntityManager &entityManager, ComponentManager &componentManager) override;
00036
00044     Entity createBackgroundLevelTwo(
00045         EntityManager &entityManager, ComponentManager &componentManager) override;
00046
00054     Entity createBackgroundLevelThree(
00055         EntityManager &entityManager, ComponentManager &componentManager) override;
00056
00064     Entity createBackgroundMenu(EntityManager &entityManager, ComponentManager &componentManager,
00065         TextureManager &textureManager) override;
00066
00077     Entity createInfoBar(
00078         EntityManager &entityManager, ComponentManager &componentManager) override;
00079
00091     Entity createPlayer(EntityManager &entityManager, ComponentManager &componentManager,
00092         int nbrOfPlayers) override;
00093
00106     Entity createShooterEnemy(EntityManager &entityManager, ComponentManager &componentManager,
00107         int posX, int posY) override;
00108
00121     Entity createBasicMonster(EntityManager &entityManager, ComponentManager &componentManager,
00122         int posX, int posY) override;
00123
00137     Entity createPlayerMissile(EntityManager &entityManager, ComponentManager &componentManager,
00138         uint32_t entityId) override;
00139
00153     Entity createForceWeapon(EntityManager &entityManager, ComponentManager &componentManager,
00154         uint32_t entityId) override;
00155
00169     Entity createForceMissile(EntityManager &entityManager, ComponentManager &componentManager,
00170         uint32_t entityId) override;
00171
00182     Entity createPowerUpBlueLaserCrystal(EntityManager &entityManager,
00183         ComponentManager &componentManager, int posX, int posY) override;
00184
00196     Entity createWall(EntityManager &entityManager, ComponentManager &componentManager, int posX,
00197         int posY) override;
00198
00214     Entity createButton(EntityManager &entityManager, ComponentManager &componentManager,
00215         TextureManager &textureManager, FontManager &fontManager, std::string text,
00216         std::function<IScenes *(AScenes *)> *onClick, float x = 0, float y = 0) override;
00217
00233     Entity createSmallButton(EntityManager &entityManager, ComponentManager &componentManager,
00234         TextureManager &textureManager, FontManager &fontManager, std::string text,
00235         std::function<IScenes *(AScenes *, AScenes::Actions)> *onClick, float x = 0,
00236         float y = 0) override;
00237
00238     Entity createUpdateButton(EntityManager &entityManager, ComponentManager &componentManager,
00239         TextureManager &textureManager, FontManager &fontManager, std::string text,
00240         std::function<IScenes *(AScenes *)> *onClick,
00241         std::function<std::string(GameParameters)> *updateText, float x, float y) override;
00242
00255     Entity createEnemyMissile(EntityManager &entityManager, ComponentManager &componentManager,

```

```

00256         uint32_t entityId) override;
00257
00269     Entity createFilter(EntityManager &entityManager, ComponentManager &componentManager,
00270         AScenes::DaltonismMode mode);
00271
00272     Entity backgroundFactory(
00273         EntityManager &entityManager, ComponentManager &componentManager, GameState type);
00274
00284     Entity createBoss(EntityManager &entityManager, ComponentManager &componentManager,
00285         EntityFactory &entityFactory);
00286
00287     Entity createTailSegment(
00288         EntityManager &entityManager, ComponentManager &componentManager) override;
00289
00290     Entity createTailEnd(
00291         EntityManager &entityManager, ComponentManager &componentManager) override;
00292 };

```

7.101 /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.

7.102 entity_manager.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** entity_manager
00006 */
00007
00008 #pragma once
00009
00010 #include "../error_handling.hpp"
00011 #include "entity.hpp"
00012 #include <algorithm>
00013 #include <memory>
00014 #include <optional>
00015 #include <vector>
00016
00025 class EntityManager {
00026 public:
00035     Entity createEntity()
00036     {
00037         int id = (entityNb += 1);
00038         entities.emplace_back(id);
00039         return entities.back();
00040     }
00041
00050     void removeEntity(int entityId)
00051     {
00052         auto it = std::remove_if(entities.begin(), entities.end(),
00053             [entityId](const Entity &entity) { return entity.getId() == entityId; });

```

```

00054
00055     if (it != entities.end()) {
00056         entities.erase(it, entities.end());
00057     } else
00058         throw entityNotFound();
00059 }
00060
00066 void removeAllEntities()
00067 {
00068     if (entities.empty())
00069         return;
00070     std::cout << "Removing all entities" << std::endl;
00071     entities.clear();
00072     entityNb = 0;
00073 }
00074
00084 std::optional<Entity *> getEntity(int entityId)
00085 {
00086     for (auto &entity : entities) {
00087         if (entity.getId() == entityId) {
00088             return &entity;
00089         }
00090     }
00091     return std::nullopt;
00092 }
00093
00101 const std::vector<Entity> &getAllEntities() const { return entities; }
00102
00103 private:
00108     int entityNb = 0;
00116     std::vector<Entity> entities;
00117 };

```

7.103 /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.

7.104 i_entity_factory.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** i_entity_factory
00006  */
00007
00008 #pragma once
00009
00010 #include "Components/component_manager.hpp"
00011 #include "entity.hpp"
00012 #include "entity_manager.hpp"
00013 #include "font_manager.hpp"
00014 #include "texture_manager.hpp"
00015
00016 // Abstract Entity Factory

```

```

00028 class IEntityFactory {
00029 public:
00034     virtual ~IEntityFactory() = default;
00035
00046     virtual Entity createBackgroundLevelOne(
00047         EntityManager &entityManager, ComponentManager &componentManager) = 0;
00048
00056     virtual Entity createBackgroundLevelTwo(
00057         EntityManager &entityManager, ComponentManager &componentManager) = 0;
00058
00066     virtual Entity createBackgroundLevelThree(
00067         EntityManager &entityManager, ComponentManager &componentManager) = 0;
00068
00076     virtual Entity createBackgroundMenu(EntityManager &entityManager,
00077         ComponentManager &componentManager, TextureManager &textureManager) = 0;
00078
00089     virtual Entity createInfoBar(
00090         EntityManager &entityManager, ComponentManager &componentManager) = 0;
00091
00102     virtual Entity createPlayer(
00103         EntityManager &entityManager, ComponentManager &componentManager, int nbrOfPlayers) = 0;
00104
00115     virtual Entity createShooterEnemy(
00116         EntityManager &entityManager, ComponentManager &componentManager, int posX, int posY) = 0;
00117
00128     virtual Entity createBasicMonster(
00129         EntityManager &entityManager, ComponentManager &componentManager, int posX, int posY) = 0;
00130
00144     virtual Entity createPlayerMissile(
00145         EntityManager &entityManager, ComponentManager &componentManager, uint32_t entityId) = 0;
00146
00159     virtual Entity createForceWeapon(
00160         EntityManager &entityManager, ComponentManager &componentManager, uint32_t entityId) = 0;
00161
00175     virtual Entity createForceMissile(
00176         EntityManager &entityManager, ComponentManager &componentManager, uint32_t entityId) = 0;
00177
00191     virtual Entity createPowerUpBlueLaserCrystal(
00192         EntityManager &entityManager, ComponentManager &componentManager, int posX, int posY) = 0;
00193
00204     virtual Entity createWall(
00205         EntityManager &entityManager, ComponentManager &componentManager, int posX, int posY) = 0;
00206
00217     virtual Entity createEnemyMissile(
00218         EntityManager &entityManager, ComponentManager &componentManager, uint32_t entityId) = 0;
00219
00235     virtual Entity createButton(EntityManager &entityManager, ComponentManager &componentManager,
00236         TextureManager &textureManager, FontManager &fontManager, std::string text,
00237         std::function<IScenes *(AScenes *)> *onClick, float x, float y) = 0;
00238
00253     virtual Entity createSmallButton(EntityManager &entityManager,
00254         ComponentManager &componentManager, TextureManager &textureManager,
00255         FontManager &fontManager, std::string text,
00256         std::function<IScenes *(AScenes *, AScenes::Actions)> *onClick, float x = 0,
00257         float y = 0) = 0;
00258
00259     virtual Entity createUpdateButton(EntityManager &entityManager,
00260         ComponentManager &componentManager, TextureManager &textureManager,
00261         FontManager &fontManager, std::string text, std::function<IScenes *(AScenes *)> *onClick,
00262         std::function<std::string(GameParameters)> *updateText, float x, float y) = 0;
00263
00264     virtual Entity createTailSegment(
00265         EntityManager &entityManager, ComponentManager &componentManager) = 0;
00266
00267     virtual Entity createTailEnd(
00268         EntityManager &entityManager, ComponentManager &componentManager) = 0;
00269
00290     enum EnemyType
00291     {
00292         BasicMonster,
00293         ShooterEnemy,
00294         Wall,
00295         Boss,
00296     };
00297 };

```

7.105 /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 <stdint>
#include <macros.hpp>
```

Classes

- struct [EntityInformation](#)
Represents information about an entity.
- struct [UIEntityInformation](#)
Represents the information of a UI entity in the game.

7.106 entity_struct.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** entities_struct
00006 */
00007
00008 #pragma once
00009
00010 #include "Components/sprite_data_component.hpp"
00011 #include "Components/text_data_component.hpp"
00012 #include <stdint>
00013 #include <macros.hpp>
00014
00022 struct EntityInformation {
00023     uint32_t uniqueID = 0;
00024     vf2d ratio = {0, 0};
00025     SpriteDataComponent spriteData;
00026     vf2d vPos = {0, 0};
00027     AnimationComponent animationComponent = {{0, 0}, {0, 0}};
00028 };
00029
00053 struct UIEntityInformation {
00054     uint32_t uniqueID = 0;
00055     uint32_t lives = 0;
00056     uint32_t score = 0;
00057     SpriteDataComponent spriteData;
00058     TextDataComponent textData;
00059 };
```

7.107 /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.

7.108 error_handling.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** error_handling
00006 */
00007
00008 #pragma once
00009
00010 #include <exception>
00011
00020 class componentNotFound : public std::exception {
00021     const char *what() const noexcept override { return "Component not found"; }
00022 };
00023
00032 class entityNotFound : public std::exception {
00033     const char *what() const noexcept override { return "Entity not found"; }
00034 };
00035
00044 class failedToLoadTexture : public std::exception {
00045     const char *what() const noexcept override { return "Failed to load texture"; }
00046 };
00047
00056 class failedToLoadSound : public std::exception {
00057     const char *what() const noexcept override { return "Failed to load sound"; }
00058 };
00059
00068 class failedToLoadFont : public std::exception {
00069     const char *what() const noexcept override { return "Failed to load font"; }
00070 };
00071
00080 class playerIdNotFound : public std::exception {
00081     const char *what() const noexcept override { return "Player ID not found"; }
00082 };
00083
00091 class failedToCreateFile : public std::exception {
00092     const char *what() const noexcept override { return "Failed to create file"; }
00093 };
00094
00103 class failedToOpenFile : public std::exception {
00104     const char *what() const noexcept override { return "Failed to open file"; }
00105 };

```

7.109 /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.

7.110 font_manager.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** font_manager
00006  */
00007
00008 #pragma once
00009
00010 #include "error_handling.hpp"
00011 #include <SFML/Graphics.hpp>
00012 #include <string>
00013 #include <unordered_map>
00014
00031 class FontManager {
00032 public:
00044     sf::Font &getFont(const std::string &filePath)
00045     {
00046         auto it = fonts.find(filePath);
00047         if (it != fonts.end()) {
00048             return it->second;
00049         }
00050
00051         auto &font = fonts[filePath];
00052         if (!font.loadFromFile(filePath)) {
00053             fonts.erase(filePath);
00054             throw failedToLoadFont();
00055         }
00056
00057         return fonts[filePath];
00058     }
00059
00068     void releaseFont(const std::string &filePath) { fonts.erase(filePath); }
00069
00070 private:
00077     std::unordered_map<std::string, sf::Font> fonts;
00078 };
```

7.111 /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.

7.111.1 Enumeration Type Documentation

7.111.1.1 FontPath

```
enum class 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	

7.111.2 Function Documentation

7.111.2.1 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.

7.111.2.2 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.

7.112 font_path.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** font_path
00006 */
00007
00008 #pragma once
00009
00010 #include <stdint>
00011 #include <string>
00012
00035 enum class FontPath : uint32_t
00036 {
00037     MAIN,
00038     NONE
00039 };
00040
00051 std::string FontFactory(FontPath font);
00052
00063 std::ostream &operator<<(std::ostream &os, const FontPath &fontPath);
```

7.113 /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.

7.113.1 Enumeration Type Documentation

7.113.1.1 GameText

```
enum class 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.

7.113.2 Function Documentation

7.113.2.1 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.

7.113.2.2 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.

7.114 game_text.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** game_text
00006  */
00007
00008 #pragma once
00009
00010 #include <stdint>
00011 #include <string>
00012
00019 enum class GameText : uint32_t
00020 {
00021     Lives,
00022     Score,
00023     NONE,
00024 };
00025
00032 std::string GameTextFactory(GameText text);
00033
00041 std::ostream &operator<<(std::ostream &os, const GameText &text);
```

7.115 /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](#) entity↵
Manager)
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.

7.115.1 Function Documentation

7.115.1.1 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.

7.115.1.2 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.

7.116 hitbox_tmp.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
```

```

00004 ** File description:
00005 ** hitbox_tmp
00006 */
00007
00008 #pragma once
00009 #include <Components/component_manager.hpp>
00010 #include <Entities/entity.hpp>
00011 #include <Entities/entity_manager.hpp>
00012 #include <entity_struct.hpp>
00013
00025 int CheckEntityPosition(
00026     uint32_t entityId, ComponentManager componentManager, EntityManager entityManager);
00027
00036 int CheckEntityMovement(
00037     EntityInformation desc, ComponentManager componentManager, EntityManager entityManager);

```

7.117 /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.

7.118 i_scenes.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** i_scenes
00006 */
00007
00008 #pragma once
00009
00010 #include <SFML/Graphics.hpp>
00011
00019 class IScenes {
00020     public:
00021         virtual ~IScenes() = default;
00022
00026         virtual void mainMenu() = 0;
00027
00031         virtual void gameLoop() = 0;
00032
00036         virtual void settingsMenu() = 0;
00037
00041         virtual void inGameMenu() = 0;
00042
00046         virtual void difficultyChoices() = 0;
00047
00051         virtual void render() = 0;
00052
00057         virtual bool shouldQuit() = 0;
00058
00063         virtual sf::RenderWindow *getRenderWindow() = 0;
00064 };

```

7.119 /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

7.119.1 Macro Definition Documentation

7.119.1.1 SCREEN_HEIGHT

```
#define SCREEN_HEIGHT 1080
```

7.119.1.2 SCREEN_WIDTH

```
#define SCREEN_WIDTH 1920
```

7.120 macros.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** macros
00006 */
00007
00008 #pragma once
00009
00010 #define SCREEN_WIDTH 1920
00011 #define SCREEN_HEIGHT 1080
00012
00019 struct vf2d {
00020     float x = 0;
00021     float y = 0;
00022 };
```

7.121 /home/runner/work/R-Type/R-Type/ECS/Interface/Include/sound_↵ path.hpp File Reference

```
#include <cstdint>
#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.

7.121.1 Enumeration Type Documentation

7.121.1.1 ActionType

```
enum class 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 <code>ActionType</code> 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	

7.121.2 Function Documentation

7.121.2.1 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.

7.122 sound_path.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** sound_path
00006 */
00007
00008 #pragma once
00009
00010 #include <stdint>
00011 #include <string>
00012
00039 enum class ActionType : uint32_t
00040 {
00041     Win,
00042     Shot,
00043     Boss,
00044     PowerUp,
00045     GameOver,
00046     BossDeath,
00047     Explosion,
```

```

00048     Background,
00049     NONE,
00050 };
00051
00062 std::string SoundFactory(ActionType action);

```

7.123 /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.

7.123.1 Enumeration Type Documentation

7.123.1.1 SpritePath

```
enum class 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.

Enumerator

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.

7.123.2 Function Documentation**7.123.2.1 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.

7.123.2.2 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.

7.124 sprite_path.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** spriteData
00006  */
00007
00008  #pragma once
00009
00010  #include <stdint>
00011  #include <string>
00012
00092  enum class SpritePath : uint32_t
00093  {
00094      Ship1,
00095      Ship2,
00096      Ship3,
00097      Ship4,
00098      Enemy1,
00099      Enemy2,
00100      Enemy3,
00101      Missile,
00102      ForceWeapon,
00103      ForceMissile,
00104      BlueLaserCrystal,
00105      Background1,
00106      Background2,
00107      Background3,
00108      // Explosion,
00109      // PowerUp,
00110      Boss,
00111      BossBullet,
00112      Bar,
00113      Wall,
00114  };
00115
00125  std::string SpriteFactory(SpritePath sprite);
00126
00137  std::ostream &operator<<(std::ostream &os, const SpritePath &spritePath);

```

7.125 /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.

7.126 audio_system.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** audio_system
00006  */
00007
00008  #pragma once
00009
00010  #include <SFML/Audio.hpp>
00011  #include <Systems/i_system.hpp>
00012  #include <audio_manager.hpp>
00013  #include <error_handling.hpp>
00014  #include <memory>
00015  #include <string>
00016
00039  class AudioSystem : public ISystem {
00040  public:
00046      AudioSystem(std::shared_ptr<AudioManager> audioManager) : _audioManager(audioManager) {}
00047
00056      void playBackgroundMusic(const std::string &filePath);
00057
00065      void stopBackgroundMusic();
00066
00075      void playSoundEffect(const std::string &filePath);
00076
00077  private:
00086      std::shared_ptr<AudioManager> _audioManager;
00094      sf::Music _backgroundMusic;
00098      std::string _currentMusicFilePath;
00107      sf::Sound _soundEffect;
00108  };

```

7.127 /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.

7.128 auto_fire_system.hpp

[Go to the documentation of this file.](#)

```

00001  /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** auto_fire_system
00006  */
00007
00008  #pragma once
00009
00010  #include "Systems/i_system.hpp"
00011
00024  class AutoFireSystem : public ISystem {
00025  public:
00033      AutoFireSystem(ComponentManager &componentManager, EntityManager &entityManager)
00034          : _componentManager(componentManager), _entityManager(entityManager)
00035      {

```

```

00036     }
00037
00047     void handleAutoFire(ComponentManager &componentManager, EntityManager &entityManager);
00048
00049 private:
00057     ComponentManager &_componentManager;
00065     EntityManager &_entityManager;
00066 };

```

7.129 /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.

7.130 collision_system.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** collision_system
00006 */
00007
00008 #pragma once
00009
00010 #include "Systems/i_system.hpp"
00011
00012 class CollisionSystem : public ISystem {
00013 public:
00027     CollisionSystem(ComponentManager &componentManager, EntityManager &entityManager)
00028         : _componentManager(componentManager), _entityManager(entityManager)
00029     {
00030     }
00031
00044     bool checkCollision(ComponentManager &componentManager, int entityId1, int entityId2);
00045
00057     bool checkOffScreen(ComponentManager &componentManager, int entityId);
00058
00059 private:
00066     ComponentManager &_componentManager;
00074     EntityManager &_entityManager;
00075 };

```

7.131 /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.

7.132 i_system.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** i_system
00006 */
00007
00008 #pragma once
00009
00010 #include "Components/component_manager.hpp"
00011 #include "Entities/entity_manager.hpp"
00012 #include <SFML/Graphics.hpp>
00013
00023 class ISystem {
00024 public:
00025     ISystem() = default;
00026     virtual ~ISystem() = default;
00027 };
```

**7.133 /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.

7.134 move_system.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** move_system
00006 */
00007
00008 #pragma once
00009
00010 #include "i_system.hpp"
00011
00020 class MoveSystem : public ISystem {
00021 public:
00028     MoveSystem(ComponentManager &componentManager, EntityManager &entityManager)
00029         : _componentManager(componentManager), _entityManager(entityManager) {}
00039     void moveEntities(ComponentManager &componentManager, EntityManager &entityManager);
00040
00051     void moveEntity(ComponentManager &componentManager, int entityId);
00052
00053 private:
00061     ComponentManager &_componentManager;
00069     EntityManager &_entityManager;
00070 };
```

7.135 /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.

7.136 render_system.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** render_system
00006  */
00007
00008 #pragma once
00009
00010 #include "Systems/i_system.hpp"
00011 #include <error_handling.hpp>
00012
00013 class RenderSystem : public ISystem {
00014 public:
00027     RenderSystem(sf::RenderWindow &window, ComponentManager &componentManager)
00028         : _window(window), _componentManager(componentManager)
00029     {
00030     }
00031
00042     void render(ComponentManager &componentManager);
00043
00044 private:
00048     sf::RenderWindow &_window;
00056     ComponentManager &_componentManager;
00065     sf::Font _font;
00066 };
```

7.137 /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>
```


7.138 systems.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** system
00006  */
00007
00008 #pragma once
00009
00010 #include <Systems/audio_system.hpp>
00011 #include <Systems/auto_fire_system.hpp>
00012 #include <Systems/collision_system.hpp>
00013 #include <Systems/move_system.hpp>
00014 #include <Systems/render_system.hpp>
00015 #include <Systems/update_system.hpp>
```

7.139 /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.

7.140 update_system.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** update_system
00006  */
00007
00008 #pragma once
00009
00010 #include "Systems/i_system.hpp"
00011
00012 class UpdateSystem : public ISystem {
00013 public:
00014     UpdateSystem(
00015         sf::RenderWindow &window, ComponentManager &componentManager, EntityManager &entityManager)
00016         : _window(window), _componentManager(componentManager), _entityManager(entityManager)
00017     {
00018     }
00019
00020     void updateSpritePositions(ComponentManager &componentManager, EntityManager &entityManager);
00021
00022 private:
00023     sf::RenderWindow &_window;
00024     ComponentManager &_componentManager;
00025     EntityManager &_entityManager;
00026 };
```

7.141 /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](#)

7.142 texture_manager.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** texture_manager
00006 */
00007
00008 #pragma once
00009
00010 #include "error_handling.hpp"
00011 #include <SFML/Graphics.hpp>
00012 #include <string>
00013 #include <unordered_map>
00014
00015 class TextureManager {
00016 public:
00029     sf::Texture &getTexture(const std::string &filePath)
00030     {
00031         auto it = textures.find(filePath);
00032         if (it != textures.end()) {
00033             return it->second;
00034         }
00035
00036         auto &texture = textures[filePath];
00037         if (!texture.loadFromFile(filePath)) {
00038             textures.erase(filePath);
00039             throw failedToLoadTexture();
00040         }
00041
00042         return textures[filePath];
00043     }
00044
00053     void releaseTexture(const std::string &filePath) { textures.erase(filePath); }
00054
00055 private:
00062     std::unordered_map<std::string, sf::Texture> textures;
00063 };
```

7.143 /home/runner/work/R-Type/R-Type/ECS/Src/a_scenes.cpp File Reference

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

7.144 /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.

7.144.1 Function Documentation

7.144.1.1 operator<<() [1/4]

```
std::ostream & operator<< (
    std::ostream & os,
    const AScenes::SpriteType & spriteType )
```

7.144.1.2 operator<<() [2/4]

```
std::ostream & operator<< (
    std::ostream & os,
    const GameState & gameState )
```

7.144.1.3 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

<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](#).

7.144.1.4 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.

7.145 /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.

7.145.1 Function Documentation**7.145.1.1 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.

7.146 /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.

7.146.1 Function Documentation

7.146.1.1 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.

7.147 /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.

7.147.1 Function Documentation

7.147.1.1 CheckCollisionLogic()

```
static int CheckCollisionLogic (
    float descLeft,
    float descRight,
    float descTop,
    float descBottom,
    ComponentManager componentManager,
    EntityManager entityManager,
    int entityId ) [static]
```

7.147.1.2 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.

7.147.1.3 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.

7.148 /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.

7.148.1 Function Documentation

7.148.1.1 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.

7.149 `/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.

7.149.1 Function Documentation

7.149.1.1 `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 <code>SpritePath</code> enum value.
---------------	---

Returns

std::string The string representation of the sprite path.

7.150 `/home/runner/work/R-Type/R-Type/ECS/Src/Systems/audio_↵system.cpp` File Reference

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

7.151 `/home/runner/work/R-Type/R-Type/ECS/Src/Systems/auto_fire_↵system.cpp` File Reference

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


7.152 /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>
```

7.153 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/move_system.cpp File Reference

```
#include <Systems/move_system.hpp>
#include <cmath>
```

7.154 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/render_system.cpp File Reference

```
#include <Systems/render_system.hpp>
```

7.155 /home/runner/work/R-Type/R-Type/ECS/Src/Systems/update_system.cpp File Reference

```
#include "Systems/update_system.hpp"
```

7.156 /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 }
- enum class [AnimationBoss](#) : uint32_t { BOSS_DEFAULT , BOSS_1 , BOSS_2 , BOSS_3 }

Functions

- bool [operator!=](#) ([AnimationComponent](#) animation, [AnimationComponent](#) other)
get if two animations are different.

7.156.1 Enumeration Type Documentation

7.156.1.1 AnimationBasicMonster

```
enum class 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	

7.156.1.2 AnimationBoss

```
enum class AnimationBoss : uint32_t [strong]
```

Enumerator

BOSS_DEFAULT	
BOSS_1	
BOSS_2	
BOSS_3	

7.156.1.3 AnimationForceMissile1

```
enum class AnimationForceMissile1 : uint32_t [strong]
```

Enumerator

FORCE_MISSILE_DEFAULT	
-----------------------	--

7.156.1.4 AnimationForceMissile2

```
enum class AnimationForceMissile2 : uint32_t [strong]
```

Enumerator

FORCE_MISSILE_DEFAULT	
-----------------------	--

7.156.1.5 AnimationForceMissile3

```
enum class 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	

7.156.1.6 AnimationForceWeapon1

```
enum class AnimationForceWeapon1 : uint32_t [strong]
```

Enumerator

FORCE_WEAPON_DEFAULT	
----------------------	--

Enumerator

FORCE_WEAPON_1	
FORCE_WEAPON_2	
FORCE_WEAPON_3	
FORCE_WEAPON_4	
FORCE_WEAPON_5	

7.156.1.7 AnimationForceWeapon2

```
enum class AnimationForceWeapon2 : uint32_t [strong]
```

Enumerator

FORCE_WEAPON_DEFAULT	
FORCE_WEAPON_1	
FORCE_WEAPON_2	
FORCE_WEAPON_3	
FORCE_WEAPON_4	
FORCE_WEAPON_5	

7.156.1.8 AnimationForceWeapon3

```
enum class AnimationForceWeapon3 : uint32_t [strong]
```

Enumerator

FORCE_WEAPON_DEFAULT	
FORCE_WEAPON_1	
FORCE_WEAPON_2	
FORCE_WEAPON_3	

7.156.1.9 AnimationShip

```
enum class 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.

7.156.2 Function Documentation

7.156.2.1 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.

7.157 animation_system.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** move_system
00006 */
00007
00008 #pragma once
00009
00010 #include "Systems/i_system.hpp"
00011 #include <entity_struct.hpp>
00012
00013 enum class AnimationShip : uint32_t
00014 {
00015     SHIP_DOWN,
00016     SHIP_FLIP_DOWN,
00017     SHIP_STRAIT,
00018     SHIP_FLIP_UP,
00019     SHIP_UP
00020 };
00021
```

```

00042 enum class AnimationBasicMonster : uint32_t
00043 {
00044     BASIC_MONSTER_DEFAULT,
00045     BASIC_MONSTER_1,
00046     BASIC_MONSTER_2,
00047     BASIC_MONSTER_3,
00048     BASIC_MONSTER_4,
00049     BASIC_MONSTER_5,
00050     BASIC_MONSTER_6,
00051     BASIC_MONSTER_7
00052 };
00053
00054 enum class AnimationForceWeapon1 : uint32_t
00055 {
00056     FORCE_WEAPON_DEFAULT,
00057     FORCE_WEAPON_1,
00058     FORCE_WEAPON_2,
00059     FORCE_WEAPON_3,
00060     FORCE_WEAPON_4,
00061     FORCE_WEAPON_5
00062 };
00063
00064 enum class AnimationForceWeapon2 : uint32_t
00065 {
00066     FORCE_WEAPON_DEFAULT,
00067     FORCE_WEAPON_1,
00068     FORCE_WEAPON_2,
00069     FORCE_WEAPON_3,
00070     FORCE_WEAPON_4,
00071     FORCE_WEAPON_5
00072 };
00073
00074 enum class AnimationForceWeapon3 : uint32_t
00075 {
00076     FORCE_WEAPON_DEFAULT,
00077     FORCE_WEAPON_1,
00078     FORCE_WEAPON_2,
00079     FORCE_WEAPON_3
00080 };
00081
00082 enum class AnimationForceMissile1 : uint32_t
00083 {
00084     FORCE_MISSILE_DEFAULT
00085 };
00086
00087 enum class AnimationForceMissile2 : uint32_t
00088 {
00089     FORCE_MISSILE_DEFAULT
00090 };
00091
00092 enum class AnimationForceMissile3 : uint32_t
00093 {
00094     FORCE_MISSILE_DEFAULT,
00095     FORCE_MISSILE_1,
00096     FORCE_MISSILE_2,
00097     FORCE_MISSILE_3,
00098     FORCE_MISSILE_4,
00099     FORCE_MISSILE_5,
00100     FORCE_MISSILE_6,
00101     FORCE_MISSILE_7
00102 };
00103
00104 enum class AnimationBoss : uint32_t
00105 {
00106     BOSS_DEFAULT,
00107     BOSS_1,
00108     BOSS_2,
00109     BOSS_3
00110 };
00111
00120 bool operator!=(AnimationComponent animation, AnimationComponent other);
00121
00148 class AnimationSystem : public ISystem {
00149 public:
00150     AnimationSystem(ComponentManager &componentManager, EntityManager &entityManager)
00151         : _componentManager(componentManager), _entityManager(entityManager){};
00152
00164     void AnimationEntities(ComponentManager &componentManager, EntityManager &entityManager,
00165         float deltaTime, bool &endOfLevel);
00166
00179     void animatePlayer(std::optional<VelocityComponent *> &velocity,
00180         std::optional<AnimationComponent *> &animation);
00181
00193     void animateBasicMonster(std::optional<AnimationComponent *> &animation);
00194
00204     void animateForceWeapon(std::optional<ForceWeaponComponent *> &forceWeapon,
00205         std::optional<AnimationComponent *> &animation, std::optional<HitboxComponent *> &hitbox);

```

```

00206
00221     void animateForceMissile(std::optional<ForceWeaponComponent *> &forceWeapon,
00222                             std::optional<AnimationComponent *> &animation, std::optional<HitboxComponent *> &hitbox);
00223
00224     void animateBoss(
00225         std::optional<BossComponent *> &boss, std::optional<AnimationComponent *> &animation);
00226
00227     private:
00235         ComponentManager &_componentManager;
00243         EntityManager &_entityManager;
00244 };

```

7.158 /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.hpp>
#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

- namespace [r_type](#)
- namespace [r_type::net](#)

7.159 level.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** level
00006  */
00007
00008 #pragma once
00009
00010 #include <Components/component_manager.hpp>
00011 #include <Components/components.hpp>
00012 #include <animation_system.hpp>
00013 #include <cmath>
00014 #include <game_struct.hpp>
00015
00016 #include <i_level.hpp>
00017
00018 namespace r_type {
00019     namespace net {
00020         template <typename T> class AServer;
00021     }
00022     template <typename T> class Level : virtual public ILevel<T> {
00023     public:
00024         Level() = default;
00025         ~Level() = default;
00026     };
00027 }

```

```

00036
00054 void Update(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00055             EntityManager &entityManager, std::chrono::system_clock::time_point newClock,
00056             bool *bUpdateEntities) override
00057 {
00058     if (server->_watingPlayersReady)
00059         return;
00060     while (std::chrono::duration_cast<std::chrono::milliseconds>(newClock - server->GetClock())
00061             .count() > 100) {
00062         *bUpdateEntities = true;
00063
00064         MoveUpdate(server, componentManager, entityManager, newClock);
00065         CollisionUpdate(server, componentManager, entityManager, newClock);
00066         AnimationUpdate(server, componentManager, entityManager, newClock);
00067         FireUpdate(server, componentManager, entityManager, newClock);
00068         if (server->_endOfLevel == false) {
00069             switch (_gameParameters.levelType) {
00070                 case GameState::LevelOne:
00071                     LevelOne(server, componentManager, entityManager, newClock);
00072                     break;
00073                 case GameState::LevelTwo:
00074                     LevelTwo(server, componentManager, entityManager, newClock);
00075                     break;
00076                 case GameState::LevelThree:
00077                     LevelThree(server, componentManager, entityManager, newClock);
00078                     break;
00079                 case GameState::Win:
00080                     EndOfGame(server, componentManager, entityManager);
00081                     break;
00082                 default:
00083                     break;
00084             }
00085         } else {
00086             if (server->_bossActive == false) {
00087                 SpawnEntity(server, entityManager, componentManager, 0,
00088                             EntityFactory::EnemyType::Boss);
00089             }
00090         }
00091         server->SetClock(server->GetClock() + std::chrono::milliseconds(500));
00092     }
00093 }
00094
00105 void SetSystem(ComponentManager &componentManager, EntityManager &entityManager) override
00106 {
00107     _moveSystem = std::make_shared<MoveSystem>(componentManager, entityManager);
00108     _collisionSystem = std::make_shared<CollisionSystem>(componentManager, entityManager);
00109     _animationSystem = std::make_shared<AnimationSystem>(componentManager, entityManager);
00110     _autoFireSystem = std::make_shared<AutoFireSystem>(componentManager, entityManager);
00111 }
00112
00127 void MoveUpdate(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00128                 EntityManager &entityManager, std::chrono::system_clock::time_point newClock) override
00129 {
00130     if (auto positionsBefore = componentManager.GetComponentMap<PositionComponent>()) {
00131         std::unordered_map<int, PositionComponent> previousPositions;
00132         // Save previous positions
00133         for (auto &pair : **positionsBefore) {
00134             int entityId = pair.first;
00135             auto positionComponent = pair.second;
00136             auto position = std::any_cast<PositionComponent>(&positionComponent);
00137             if (position) {
00138                 previousPositions.insert({entityId, *position});
00139             }
00140         }
00141         // Move entities
00142         _moveSystem.get()->moveEntities(componentManager, entityManager);
00143         // Compare new positions
00144         if (auto positionsAfter = componentManager.GetComponentMap<PositionComponent>()) {
00145             for (auto &pair : **positionsAfter) {
00146                 int entityId = pair.first;
00147                 auto &newPositionComponent = pair.second;
00148                 if (auto newPosition =
00149                     std::any_cast<PositionComponent>(&newPositionComponent)) {
00149                     auto it = previousPositions.find(entityId);
00150                     if (it != previousPositions.end()) {
00151                         auto &oldPosition = it->second;
00152                         if (oldPosition.x != newPosition->x ||
00153                             oldPosition.y != newPosition->y) {
00154                             if (auto spriteData =
00155                                 componentManager.GetComponent<SpriteDataComponent>(
00156                                     entityId)) {
00157                                 r_type::net::Message<TypeMessage> msg;
00158                                 vf2d newPos(newPosition->x, newPosition->y);
00159                                 msg.header.id = TypeMessage::MoveEntityMessage;
00160                                 msg « entityId « newPos;
00161                                 server->MessageAllClients(msg);
00162                             }
00163                         }
00164                     }
00165                 }
00166             }
00167         }
00168     }
00169 }

```



```

00164         }
00165     }
00166     }
00167     }
00168     }
00169 }
00170 }
00171
00190 static bool collisionAction(r_type::net::AServer<T> *server,
00191     ComponentManager &componentManager, EntityManager &entityManager,
00192     std::vector<int> &entitiesToRemove, std::vector<int> &entitiesToAdd, uint32_t entityId1,
00193     uint32_t entityId2)
00194 {
00195     // component of entity 1
00196     auto player1 = componentManager.GetComponent<PlayerComponent>(entityId1);
00197     auto playerMissile1 = componentManager.GetComponent<PlayerMissileComponent>(entityId1);
00198     auto enemyMissile1 = componentManager.GetComponent<EnemyMissileComponent>(entityId1);
00199     auto playerHealth1 = componentManager.GetComponent<HealthComponent>(entityId1);
00200     auto powerUp1 = componentManager.GetComponent<PowerUpComponent>(entityId1);
00201     auto enemy1 = componentManager.GetComponent<EnemyComponent>(entityId1);
00202     auto forceWeapon1 = componentManager.GetComponent<ForceWeaponComponent>(entityId1);
00203     auto forceMissile1 = componentManager.GetComponent<ForceMissileComponent>(entityId1);
00204     auto boss1 = componentManager.GetComponent<BossComponent>(entityId1);
00205     auto tail1 = componentManager.GetComponent<TailComponent>(entityId1);
00206
00207     // component of entity 2
00208     auto enemyMissile2 = componentManager.GetComponent<EnemyMissileComponent>(entityId2);
00209     auto player2 = componentManager.GetComponent<PlayerComponent>(entityId2);
00210     auto playerHealth2 = componentManager.GetComponent<HealthComponent>(entityId2);
00211     auto playerMissile2 = componentManager.GetComponent<PlayerMissileComponent>(entityId2);
00212     auto powerUp2 = componentManager.GetComponent<PowerUpComponent>(entityId2);
00213     auto enemy2 = componentManager.GetComponent<EnemyComponent>(entityId2);
00214     auto forceWeapon2 = componentManager.GetComponent<ForceWeaponComponent>(entityId2);
00215     auto forceMissile2 = componentManager.GetComponent<ForceMissileComponent>(entityId2);
00216     auto boss2 = componentManager.GetComponent<BossComponent>(entityId2);
00217     auto tail2 = componentManager.GetComponent<TailComponent>(entityId2);
00218
00219     // Handle collision
00220     if (player1) {
00221         if (playerHealth1) {
00222             if (enemy2 || enemyMissile2) {
00223                 if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId2) ==
00224                     entitiesToRemove.end()) {
00225                     entitiesToRemove.push_back(entityId2);
00226                     playerHealth1.value()->lives -= 1;
00227                 }
00228             }
00229             if (boss2) {
00230                 playerHealth1.value()->lives -= 1;
00231                 if (auto bossHealth =
00232                     componentManager.GetComponent<HealthComponent>(entityId2)) {
00233                     bossHealth.value()->lives -= 1;
00234                     if (bossHealth.value()->lives <= 0) {
00235                         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(),
00236                             entityId2) == entitiesToRemove.end()) {
00237                             entitiesToRemove.push_back(entityId2);
00238                         }
00239                     }
00240                 }
00241             }
00242             if (tail2) {
00243                 playerHealth1.value()->lives -= 1;
00244             }
00245             r_type::net::Message<TypeMessage> updLivesMsg;
00246             updLivesMsg.header.id = TypeMessage::UpdateInfoBar;
00247             updLivesMsg « server->UpdateInfoBar(entityId1);
00248             server->MessageClient(server->getClientById(server->_deqConnections,
00249                 server->GetPlayerClientId(entityId1)),
00250                 updLivesMsg);
00251             if (playerHealth1.value()->lives <= 0) {
00252                 if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId1) ==
00253                     entitiesToRemove.end()) {
00254                     entitiesToRemove.push_back(entityId1);
00255                 }
00256             }
00257         }
00258     }
00259     // when player collides with power up
00260     if (powerUp2) {
00261         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId2) ==
00262             entitiesToRemove.end()) {
00263             entitiesToRemove.push_back(entityId2);
00264         }
00265         auto linkForceComponent =
00266             componentManager.GetComponent<LinkForceComponent>(entityId1);
00267         if (linkForceComponent.value()->targetId != -1) {
00268             auto forceWeapon = componentManager.GetComponent<ForceWeaponComponent>(
00269                 linkForceComponent.value()->targetId);

```

```

00269         if (forceWeapon) {
00270             if (forceWeapon.value()->level < 3) {
00271                 forceWeapon.value()->level += 1;
00272             }
00273         }
00274     } else {
00275         Entity weapon = server->GetEntityFactory().createForceWeapon(
00276             entityManager, componentManager, entityId1);
00277         entitiesToAdd.push_back(weapon.getId());
00278         auto linkForceComponent =
00279             componentManager.GetComponent<LinkForceComponent>(entityId1);
00280         linkForceComponent.value()->targetId = weapon.getId();
00281     }
00282 }
00283 if (forceWeapon2) {
00284     auto frontComponent = componentManager.GetComponent<FrontComponent>(entityId1);
00285     if (frontComponent) {
00286         frontComponent.value()->targetId = entityId2;
00287         auto forceWeapon = componentManager.GetComponent<ForceWeaponComponent>(
00288             frontComponent.value()->targetId);
00289         if (forceWeapon) {
00290             forceWeapon.value()->attached = true;
00291         }
00292         auto forceWeaponMovementComponent =
00293             componentManager.GetComponent<MovementComponent>(
00294                 frontComponent.value()->targetId);
00295         if (forceWeaponMovementComponent) {
00296             forceWeaponMovementComponent.value()->move = false;
00297         }
00298     }
00299 }
00300 return true;
00301 } else if (playerMissile1) {
00302     if (enemy2 || enemyMissile2) {
00303         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId1) ==
00304             entitiesToRemove.end()) {
00305             entitiesToRemove.push_back(entityId1);
00306         }
00307         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId2) ==
00308             entitiesToRemove.end()) {
00309             if (rand() % 10 == 0) {
00310                 auto posEnemy =
00311                     componentManager.GetComponent<PositionComponent>(entityId2);
00312                 if (posEnemy) {
00313                     Entity weapon =
00314                         server->GetEntityFactory().createPowerUpBlueLaserCrystal(
00315                             entityManager, componentManager, posEnemy.value()->x,
00316                             posEnemy.value()->y);
00317                     entitiesToAdd.push_back(weapon.getId());
00318                 }
00319             }
00320             entitiesToRemove.push_back(entityId2);
00321         }
00322         int playerId = playerMissile1.value()->playerId;
00323         if (auto playerScore = componentManager.GetComponent<ScoreComponent>(playerId)) {
00324             playerScore.value()->score += 100;
00325         }
00326         r_type::net::Message<TypeMessage> updScoreMsg;
00327         updScoreMsg.header.id = TypeMessage::UpdateInfoBar;
00328         updScoreMsg « server->UpdateInfoBar(playerId);
00329         server->MessageClient(server->getClientById(server->_deqConnections,
00330             server->GetPlayerClientId(playerId)),
00331             updScoreMsg);
00332     }
00333 if (boss2) {
00334     if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId1) ==
00335         entitiesToRemove.end()) {
00336         entitiesToRemove.push_back(entityId1);
00337     }
00338     if (auto bossHealth = componentManager.GetComponent<HealthComponent>(entityId2)) {
00339         bossHealth.value()->lives -= 1;
00340         if (bossHealth.value()->lives <= 0) {
00341             if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(),
00342                 entityId2) == entitiesToRemove.end()) {
00343                 entitiesToRemove.push_back(entityId2);
00344             }
00345         }
00346     }
00347     int playerId = playerMissile1.value()->playerId;
00348     if (auto playerScore = componentManager.GetComponent<ScoreComponent>(playerId)) {
00349         playerScore.value()->score += 200;
00350     }
00351     r_type::net::Message<TypeMessage> updScoreMsg;
00352     updScoreMsg.header.id = TypeMessage::UpdateInfoBar;
00353     updScoreMsg « server->UpdateInfoBar(playerId);
00354     server->MessageClient(server->getClientById(server->_deqConnections,
00355         server->GetPlayerClientId(playerId)),

```

```

00356         updScoreMsg);
00357     }
00358     if (tail2) {
00359         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId1) ==
00360             entitiesToRemove.end()) {
00361             entitiesToRemove.push_back(entityId1);
00362         }
00363     }
00364     return true;
00365 } else if (forceMissile1) {
00366     if (enemy2 || enemyMissile2) {
00367         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId1) ==
00368             entitiesToRemove.end()) {
00369             entitiesToRemove.push_back(entityId1);
00370         }
00371         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId2) ==
00372             entitiesToRemove.end()) {
00373             auto posEnemy = componentManager.GetComponent<PositionComponent>(entityId2);
00374             if (rand() % 10 == 0) {
00375                 auto posEnemy =
00376                     componentManager.GetComponent<PositionComponent>(entityId2);
00377                 if (posEnemy) {
00378                     Entity weapon =
00379                         server->GetEntityFactory().createPowerUpBlueLaserCrystal(
00380                             entityManager, componentManager, posEnemy.value()->x,
00381                             posEnemy.value()->y);
00382                     entitiesToAdd.push_back(weapon.getId());
00383                 }
00384             }
00385             entitiesToRemove.push_back(entityId2);
00386         }
00387     }
00388     auto weapon = componentManager.GetComponent<ForceWeaponComponent>(
00389         forceMissile1.value()->forceId);
00390     if (weapon) {
00391         int playerId = weapon.value()->playerId;
00392         if (auto playerScore = componentManager.GetComponent<ScoreComponent>(playerId)) {
00393             if (enemy2 || enemyMissile2) {
00394                 if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(),
00395                     entityId1) == entitiesToRemove.end()) {
00396                     entitiesToRemove.push_back(entityId1);
00397                 }
00398                 if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(),
00399                     entityId2) == entitiesToRemove.end()) {
00400                     auto posEnemy =
00401                         componentManager.GetComponent<PositionComponent>(entityId2);
00402                     if (rand() % 10 == 0) {
00403                         auto posEnemy =
00404                             componentManager.GetComponent<PositionComponent>(entityId2);
00405                         if (posEnemy) {
00406                             Entity weapon =
00407                                 server->GetEntityFactory().createPowerUpBlueLaserCrystal(
00408                                     entityManager, componentManager, posEnemy.value()->x,
00409                                     posEnemy.value()->y);
00410                             entitiesToAdd.push_back(weapon.getId());
00411                         }
00412                     }
00413                     entitiesToRemove.push_back(entityId2);
00414                 }
00415                 playerScore.value()->score += 100;
00416             }
00417             if (boss2) {
00418                 if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(),
00419                     entityId1) == entitiesToRemove.end()) {
00420                     entitiesToRemove.push_back(entityId1);
00421                 }
00422                 if (auto bossHealth =
00423                     componentManager.GetComponent<HealthComponent>(entityId2)) {
00424                     bossHealth.value()->lives -= 2;
00425                     if (bossHealth.value()->lives <= 0) {
00426                         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(),
00427                             entityId2) == entitiesToRemove.end()) {
00428                             entitiesToRemove.push_back(entityId2);
00429                         }
00430                     }
00431                 }
00432                 playerScore.value()->score += 200;
00433             }
00434         }
00435     }
00436     if (tail2) {
00437         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(),
00438             entityId1) == entitiesToRemove.end()) {
00439             entitiesToRemove.push_back(entityId1);
00440         }
00441     }
00442     r_type::net::Message<TypeMessage> updScoreMsg;

```

```

00443         updScoreMsg.header.id = TypeMessage::UpdateInfoBar;
00444         updScoreMsg « server->UpdateInfoBar(playerId);
00445         server->MessageClient(server->getClientById(server->_deqConnections,
00446             server->GetPlayerClientId(playerId)),
00447             updScoreMsg);
00448     }
00449     return true;
00450 } else if (forceWeapon1) {
00451     if (enemyMissile2) {
00452         if (std::find(entitiesToRemove.begin(), entitiesToRemove.end(), entityId2) ==
00453             entitiesToRemove.end()) {
00454             entitiesToRemove.push_back(entityId2);
00455         }
00456         std::cout « "force weapon collided with enemy missile" « std::endl;
00457     }
00458     return true;
00459 }
00460 return false;
00461 }
00462
00475 void CollisionUpdate(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00476     EntityManager &entityManager, std::chrono::system_clock::time_point newClock) override
00477 {
00478     std::vector<int> entitiesToRemove;
00479     std::vector<int> entitiesToAdd;
00480     auto beforeCollisionEntities = entityManager.getAllEntities();
00481
00482     for (size_t i = 0; i < beforeCollisionEntities.size(); ++i) {
00483         int entityId1 = beforeCollisionEntities[i].getId();
00484         for (size_t j = i + 1; j < beforeCollisionEntities.size(); ++j) {
00485             int entityId2 = beforeCollisionEntities[j].getId();
00486             // Check for collision
00487             if (_collisionSystem.get()->checkCollision(
00488                 componentManager, entityId1, entityId2)){
00489                 if (collisionAction(server, componentManager, entityManager, entitiesToRemove,
00490                     entitiesToAdd, entityId1, entityId2) == false) {
00491                     collisionAction(server, componentManager, entityManager, entitiesToRemove,
00492                         entitiesToAdd, entityId2, entityId1);
00493                 }
00494             }
00495         }
00496     }
00497
00498     // Add entities
00499     for (int entityId : entitiesToAdd) {
00500         r_type::net::Message<TypeMessage> msg;
00501         msg.header.id = TypeMessage::CreateEntityMessage;
00502         msg « server->InitiateWeaponForce(entityId);
00503         server->MessageAllClients(msg);
00504     }
00505     // Remove entities
00506     for (int entityId : entitiesToRemove) {
00507         r_type::net::Message<TypeMessage> msg;
00508         msg.header.id = TypeMessage::DestroyEntityMessage;
00509         msg « entityId;
00510         server->MessageAllClients(msg);
00511         if (auto playerComponent = componentManager.getComponent<PlayerComponent>(entityId)) {
00512             continue;
00513         }
00514         if (auto bossComponent = componentManager.getComponent<BossComponent>(entityId)) {
00515             server->RemoveBossTail(entityId);
00516         }
00517         componentManager.removeEntityFromAllComponents(entityId);
00518         entityManager.removeEntity(entityId);
00519     }
00520     // Remove entities when they go off-screen
00521     auto afterCollisionEntities = entityManager.getAllEntities();
00522     for (const auto &entity : afterCollisionEntities) {
00523         int entityId = entity.getId();
00524         if (_collisionSystem.get()->checkOffScreen(componentManager, entityId)) {
00525             r_type::net::Message<TypeMessage> msg;
00526             msg.header.id = TypeMessage::DestroyEntityMessage;
00527             msg « entityId;
00528             server->MessageAllClients(msg);
00529             componentManager.removeEntityFromAllComponents(entityId);
00530             entityManager.removeEntity(entityId);
00531         }
00532     }
00533 }
00534
00551 void AnimationUpdate(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00552     EntityManager &entityManager, std::chrono::system_clock::time_point newClock) override
00553 {
00554     if (auto animationsBefore = componentManager.getComponentMap<AnimationComponent>()) {
00555         std::unordered_map<int, AnimationComponent> previousAnimations;
00556     }
00557 }

```

```

00558         // Save previous animations
00559         for (const auto &pair : **animationsBefore) {
00560             int entityId = pair.first;
00561             const auto animationComponent = pair.second;
00562             auto animation = std::any_cast<AnimationComponent>(&animationComponent);
00563             if (animation) {
00564                 previousAnimations.insert({entityId, *animation});
00565             }
00566         }
00567         _animationSystem->AnimationEntities(
00568             componentManager, entityManager, 0.2, server->_endOfLevel);
00569         // Compare new Animations
00570         if (auto animationsAfter = componentManager.GetComponentMap<AnimationComponent>()) {
00571             for (const auto &pair : **animationsAfter) {
00572                 int entityId = pair.first;
00573                 const auto &newAnimationComponent = pair.second;
00574                 auto newAnimation = std::any_cast<AnimationComponent>(&newAnimationComponent);
00575                 if (newAnimation) {
00576                     auto it = previousAnimations.find(entityId);
00577                     if (it != previousAnimations.end()) {
00578                         const auto &oldAnimation = it->second;
00579                         if (oldAnimation != *newAnimation) {
00580                             r_type::net::Message<TypeMessage> msg;
00581                             msg.header.id = TypeMessage::AnimateEntityMessage;
00582                             msg << entityId << newAnimation->dimension << newAnimation->offset;
00583                             server->MessageAllClients(msg);
00584                         }
00585                     }
00586                 }
00587             }
00588         }
00589     }
00590 }
00591
00605 void FireUpdate(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00606               EntityManager &entityManager, std::chrono::system_clock::time_point newClock) override
00607 {
00608     // auto fire system
00609     _autoFireSystem->handleAutoFire(componentManager, entityManager);
00610     auto shootComponentMap = componentManager.GetComponentMap<ShootComponent>();
00611     if (shootComponentMap) {
00612         for (auto &pair : **shootComponentMap) {
00613             int entityId = pair.first;
00614             auto &shootComponent = pair.second;
00615             if (auto shootInfo = std::any_cast<ShootComponent>(&shootComponent)) {
00616                 if (shootInfo->canShoot) {
00617                     auto weapon =
00618                         componentManager.GetComponent<ForceWeaponComponent>(entityId);
00619                     if (weapon) {
00620                         if (!weapon.value()->attached) {
00621                             Entity missile = server->GetEntityFactory().createForceMissile(
00622                                 entityManager, componentManager, entityId);
00623                             r_type::net::Message<TypeMessage> weaponMissileMsg;
00624                             weaponMissileMsg.header.id = TypeMessage::CreateEntityMessage;
00625                             weaponMissileMsg << server->InitiatePlayerMissile(missile.getId());
00626                             server->MessageAllClients(weaponMissileMsg);
00627                         }
00628                         shootInfo->canShoot = false;
00629                     } else {
00630                         r_type::net::Message<TypeMessage> enemyMissileMsg;
00631                         enemyMissileMsg.header.id = TypeMessage::CreateEntityMessage;
00632                         enemyMissileMsg << server->InitiateEnemyMissile(entityId);
00633                         server->MessageAllClients(enemyMissileMsg);
00634                         shootInfo->canShoot = false;
00635                     }
00636                 }
00637             }
00638         }
00639     }
00640 }
00641
00655 void LevelOne(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00656              EntityManager &entityManager, std::chrono::system_clock::time_point newClock) override
00657 {
00658     if (std::chrono::duration_cast<std::chrono::seconds>(
00659         server->GetClock() - _basicMonsterSpawnTime)
00660         .count() > _gameParameters.spawnTimeBasicMonster) {
00661         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfBasicMonster,
00662                     EntityFactory::EnemyType::BasicMonster);
00663         _basicMonsterSpawnTime = server->GetClock();
00664     }
00665     if (std::chrono::duration_cast<std::chrono::seconds>(
00666         server->GetClock() - _shooterEnemySpawnTime)
00667         .count() > _gameParameters.spawnTimeShooterEnemy) {
00668         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfShooterEnemy,
00669                     EntityFactory::EnemyType::ShooterEnemy);
00670         _shooterEnemySpawnTime = server->GetClock();
00671     }
00672 }

```

```

00671     }
00672     if (std::chrono::duration_cast<std::chrono::seconds>(server->GetClock() - _WallSpawnTime)
00673         .count() > _gameParameters.spawnTimeWall) {
00674         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfWall,
00675             EntityFactory::EnemyType::Wall);
00676         _WallSpawnTime = server->GetClock();
00677     }
00678 }
00679
00693 void LevelTwo(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00694     EntityManager &entityManager, std::chrono::system_clock::time_point newClock) override
00695 {
00696     server->_bossActive = false;
00697     if (std::chrono::duration_cast<std::chrono::seconds>(
00698         server->GetClock() - _basicMonsterSpawnTime)
00699         .count() > _gameParameters.spawnTimeBasicMonster) {
00700         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfBasicMonster,
00701             EntityFactory::EnemyType::BasicMonster);
00702         _basicMonsterSpawnTime = server->GetClock();
00703     }
00704     if (std::chrono::duration_cast<std::chrono::seconds>(
00705         server->GetClock() - _shooterEnemySpawnTime)
00706         .count() > _gameParameters.spawnTimeShooterEnemy) {
00707         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfShooterEnemy,
00708             EntityFactory::EnemyType::ShooterEnemy);
00709         _shooterEnemySpawnTime = server->GetClock();
00710     }
00711     if (std::chrono::duration_cast<std::chrono::seconds>(server->GetClock() - _WallSpawnTime)
00712         .count() > _gameParameters.spawnTimeWall) {
00713         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfWall,
00714             EntityFactory::EnemyType::Wall);
00715         _WallSpawnTime = server->GetClock();
00716     }
00717 }
00718
00732 void LevelThree(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00733     EntityManager &entityManager, std::chrono::system_clock::time_point newClock) override
00734 {
00735     server->_bossActive = false;
00736     if (std::chrono::duration_cast<std::chrono::seconds>(
00737         server->GetClock() - _basicMonsterSpawnTime)
00738         .count() > _gameParameters.spawnTimeBasicMonster) {
00739         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfBasicMonster,
00740             EntityFactory::EnemyType::BasicMonster);
00741         _basicMonsterSpawnTime = server->GetClock();
00742     }
00743     if (std::chrono::duration_cast<std::chrono::seconds>(
00744         server->GetClock() - _shooterEnemySpawnTime)
00745         .count() > _gameParameters.spawnTimeShooterEnemy) {
00746         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfShooterEnemy,
00747             EntityFactory::EnemyType::ShooterEnemy);
00748         _shooterEnemySpawnTime = server->GetClock();
00749     }
00750     if (std::chrono::duration_cast<std::chrono::seconds>(server->GetClock() - _WallSpawnTime)
00751         .count() > _gameParameters.spawnTimeWall) {
00752         SpawnEntity(server, entityManager, componentManager, _gameParameters.nbrOfWall,
00753             EntityFactory::EnemyType::Wall);
00754         _WallSpawnTime = server->GetClock();
00755     }
00756 }
00757
00758 void EndOfGame(r_type::net::AServer<T> *server, ComponentManager &componentManager,
00759     EntityManager &entityManager) override
00760 {
00761     auto entities = entityManager.getAllEntities();
00762     for (const auto &entity : entities) {
00763         int entityId = entity.getId();
00764         if (auto playerComponent = componentManager.getComponent<PlayerComponent>(entityId)) {
00765             server->SavePlayerScore(entityId);
00766         }
00767     }
00768     r_type::net::Message<TypeMessage> msg;
00769     msg.header.id = TypeMessage::EndOfGame;
00770     server->MessageAllClients(msg);
00771     server->_endOfLevel = true;
00772     server->_bossActive = true;
00773 }
00774
00790 void SpawnEntity(r_type::net::AServer<T> *server, EntityManager &entityManager,
00791     ComponentManager &componentManager, int nbrOfEnemy, EntityFactory::EnemyType enemyType)
00792 {
00793     switch (enemyType) {
00794     case EntityFactory::EnemyType::BasicMonster: {
00795         int i = 0;
00796         int posX = 100;
00797         int posY = static_cast<int>((rand() % 70) + 10);
00798         while (i < nbrOfEnemy) {

```

```

00799         Entity Monster = server->GetEntityFactory().createBasicMonster(
00800             entityManager, componentManager, posX, posY);
00801         posY += (static_cast<int>(rand() % 10) - static_cast<int>(rand() % 10));
00802         if (posY > 90)
00803             posY = static_cast<int>((rand() % 70) + 10);
00804         posX += 5;
00805         r_type::net::Message<TypeMessage> msg;
00806         msg.header.id = TypeMessage::CreateEntityMessage;
00807         msg « server->FormatEntityInformation(Monster.getId());
00808         server->MessageAllClients(msg);
00809         i++;
00810     }
00811 } break;
00812 case EntityFactory::EnemyType::ShooterEnemy: {
00813     int i = 0;
00814     int posX = 99;
00815     int posY = static_cast<int>((rand() % 70) + 10);
00816     while (i < nbrOfEnemy) {
00817         Entity ShooterEnemy = server->GetEntityFactory().createShooterEnemy(
00818             entityManager, componentManager, posX, posY);
00819         posX += 5;
00820         posY += (static_cast<int>(rand() % 20) - static_cast<int>(rand() % 10));
00821
00822         if (posY > 90)
00823             posY = static_cast<int>((rand() % 70) + 10);
00824         r_type::net::Message<TypeMessage> msg;
00825         msg.header.id = TypeMessage::CreateEntityMessage;
00826         msg « server->FormatEntityInformation(ShooterEnemy.getId());
00827         server->MessageAllClients(msg);
00828         i++;
00829     }
00830 } break;
00831 case EntityFactory::EnemyType::Wall: {
00832     int i = 0;
00833     int posX = 99;
00834     int posY = static_cast<int>((rand() % 70) + 10);
00835
00836     while (i < nbrOfEnemy) {
00837         Entity wall = server->GetEntityFactory().createWall(
00838             entityManager, componentManager, posX, posY);
00839         posX += 5;
00840         posY += (static_cast<int>(rand() % 20) - static_cast<int>(rand() % 10));
00841
00842         if (posY > 90)
00843             posY = static_cast<int>((rand() % 70) + 10);
00844         r_type::net::Message<TypeMessage> msg;
00845         msg.header.id = TypeMessage::CreateEntityMessage;
00846         msg « server->FormatEntityInformation(wall.getId());
00847         server->MessageAllClients(msg);
00848         i++;
00849     }
00850 } break;
00851 case EntityFactory::EnemyType::Boss: {
00852     server->InitBoss(server);
00853 } break;
00854 default:
00855     break;
00856 }
00857 }
00858
00859 EntityInformation GetEntityBackGround(r_type::net::AServer<T> *server,
00860     EntityManager &entityManager, ComponentManager &componentManager) override
00861 {
00862     EntityInformation entityInfo;
00863     auto background = componentManager.getComponentMap<BackgroundComponent>();
00864     if (background) {
00865         for (auto &pair : **background) {
00866             int entityId = pair.first;
00867             auto &backgroundComponent = pair.second;
00868             if (auto backgroundInfo =
00869                 std::any_cast<BackgroundComponent>(&backgroundComponent)) {
00870                 return server->FormatEntityInformation(entityId);
00871             }
00872         }
00873     } else {
00874         return InitiateBackground(server, entityManager, componentManager);
00875     }
00876     return entityInfo;
00877 }
00878
00879 void ChangeBackground(r_type::net::AServer<T> *server, EntityManager &entityManager,
00880     ComponentManager &componentManager) override
00881 {
00882     r_type::net::Message<TypeMessage> msg;
00883     msg.header.id = TypeMessage::DestroyEntityMessage;
00884     auto background = componentManager.getComponentMap<BackgroundComponent>();
00885     if (background) {

```

```

00906         for (auto &pair : **background) {
00907             int entityId = pair.first;
00908             auto &backgroundComponent = pair.second;
00909             if (auto backgroundInfo =
00910                 std::any_cast<BackgroundComponent>(&backgroundComponent)) {
00911                 msg « entityId;
00912                 server->MessageAllClients(msg);
00913                 componentManager.removeEntityFromAllComponents(entityId);
00914                 entityManager.removeEntity(entityId);
00915                 break;
00916             }
00917         }
00918     }
00919     r_type::net::Message<TypeMessage> newMsg;
00920     newMsg.header.id = TypeMessage::CreateEntityMessage;
00921     EntityInformation entityInfo = InitiateBackground(server, entityManager, componentManager);
00922     newMsg « entityInfo;
00923     server->MessageAllClients(newMsg);
00924 }
00925
00926 GameState GetLevel() override { return _gameParameters.levelType; }
00927
00935 EntityInformation InitiateBackground(r_type::net::AServer<T> *server,
00936     EntityManager &entityManager, ComponentManager &componentManager) override
00937 {
00938     EntityInformation entityInfo;
00939     Entity background = server->GetEntityFactory().backgroundFactory(
00940         entityManager, componentManager, _gameParameters.levelType);
00941     entityInfo.uniqueID = background.getId();
00942     auto sprite = componentManager.getComponent<SpriteDataComponent>(entityInfo.uniqueID);
00943     auto backgroundPos = componentManager.getComponent<PositionComponent>(entityInfo.uniqueID);
00944     auto animation = componentManager.getComponent<AnimationComponent>(entityInfo.uniqueID);
00945     if (sprite) {
00946         entityInfo.spriteData = * (sprite.value());
00947         entityInfo.vPos.x = backgroundPos.value()->x;
00948         entityInfo.vPos.y = backgroundPos.value()->y;
00949         if (animation) {
00950             entityInfo.ratio.x =
00951                 (animation.value()->dimension.x * sprite.value()->scale.x) / SCREEN_WIDTH;
00952             entityInfo.ratio.y =
00953                 (animation.value()->dimension.y * sprite.value()->scale.y) / SCREEN_HEIGHT;
00954             entityInfo.animationComponent.dimension = animation.value()->dimension;
00955             entityInfo.animationComponent.offset = animation.value()->offset;
00956         }
00957     }
00958     return entityInfo;
00959 }
00960
00968 void SetGameParameters(GameParameters gameParameters) { _gameParameters = gameParameters; }
00969
00977 void ChangeLevel(GameState state) override { _gameParameters.levelType = state; }
00978
00979 protected:
00988     std::shared_ptr<MoveSystem> _moveSystem;
00997     std::shared_ptr<CollisionSystem> _collisionSystem;
01006     std::shared_ptr<AnimationSystem> _animationSystem;
01015     std::shared_ptr<AutoFireSystem> _autoFireSystem;
01016
01024     std::chrono::system_clock::time_point _basicMonsterSpawnTime =
01025         std::chrono::system_clock::now();
01033     std::chrono::system_clock::time_point _shooterEnemySpawnTime =
01034         std::chrono::system_clock::now();
01042     std::chrono::system_clock::time_point _WallSpawnTime = std::chrono::system_clock::now();
01050     std::chrono::system_clock::time_point _spawnTimeMonsterThree;
01051
01058     GameParameters _gameParameters;
01059 };
01060 } // namespace r_type

```

7.160 /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.hpp>
#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

- namespace [r_type](#)
- namespace [r_type::net](#)

7.161 a_server.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 ** EPITECH PROJECT, 2024
00003 ** R-Type
00004 ** File description:
00005 ** netServer
00006 */
00007
00008 #pragma once
00009
00010 #include <Components/component_manager.hpp>
00011 #include <Components/components.hpp>
00012 #include <Entities/entity_factory.hpp>
00013 #include <Entities/entity_manager.hpp>
00014 #include <Net/i_server.hpp>
00015 #include <Systems/systems.hpp>
00016 #include <cmath>
00017 #include <entity_struct.hpp>
00018 #include <error_handling.hpp>
00019 #include <filesystem>
00020 #include <fstream>
00021 #include <game_struct.hpp>
00022 #include <iostream>
00023 #include <level.hpp>
00024 #include <macros.hpp>
00025 #include <unordered_map>
00026
00027 namespace r_type {
00028     namespace net {
00029
00041         template <typename T> class AServer : virtual public r_type::net::IServer<T> {
00042             public:
00054                 AServer(uint16_t port)
00055                     : r_type::net::IServer<T>(),
00056                       _asioSocket(_asioContext, asio::ip::udp::endpoint(asio::ip::udp::v4(), port)),
00057                       _port(port)
00058                 {
00059                     srand(time(NULL));
00060                     _componentManager = ComponentManager();
00061                     _entityManager = EntityManager();
00062                     _entityFactory = EntityFactory();
00063                     _level = r_type::Level<T>();
00064                     _level.SetSystem(_componentManager, _entityManager);
00065                 }
00066             };
00067     }
00068 }
```

```

00065     }
00066
00073 ~AServer() { Stop(); }
00074
00085 bool Start()
00086 {
00087     try {
00088         WaitForClientMessage();
00089
00090         _threadContext = std::thread([this]() { _asioContext.run(); });
00091     } catch (std::exception &e) {
00092         std::cerr << "[SERVER] Exception: " << e.what() << std::endl;
00093         return false;
00094     }
00095     std::cout << "[SERVER] Started on port " << _port << std::endl;
00096     return true;
00097 }
00098
00105 void Stop()
00106 {
00107     _asioContext.stop();
00108
00109     if (_threadContext.joinable())
00110         _threadContext.join();
00111
00112     if (_asioSocket.is_open())
00113         _asioSocket.close();
00114
00115     std::cout << "[SERVER] Stopped!\n";
00116 }
00117
00134 void WaitForClientMessage()
00135 {
00136     _asioSocket.async_receive_from(asio::buffer(_tempBuffer.data(), _tempBuffer.size()),
00137     _clientEndpoint, [this](std::error_code ec, std::size_t bytes_recvd) {
00138         if (_clientEndpoint.protocol() != asio::ip::udp::v4())
00139             return WaitForClientMessage();
00140         if (!ec) {
00141             std::cout << "[SERVER] New Connection: " << _clientEndpoint << std::endl;
00142             // check if connection already exists
00143             std::cout << "Client endpoint connection: " << _clientEndpoint << std::endl;
00144             for (auto &conn : _deqConnections) {
00145                 std::cout << "Client endpoint: " << conn->getEndpoint() << std::endl;
00146                 if (conn->getEndpoint() == _clientEndpoint) {
00147                     std::cout << "[" << conn->GetID() << "]" Connection Approved"
00148                     << std::endl;
00149                     std::cout << "[SERVER] Connection already exists" << std::endl;
00150                     if (OnClientConnect(conn)) {
00151                         std::cout << "[SERVER] Connection already exists" << std::endl;
00152                     } else {
00153                         std::cout << "[-----] Connection Denied" << std::endl;
00154                     }
00155                     return;
00156                 }
00157             }
00158             asio::ip::udp::socket newSocket(
00159                 _asioContext, asio::ip::udp::endpoint(asio::ip::udp::v4(), 0));
00160
00161             // create client socket
00162             std::shared_ptr<Connection<T>> newConn =
00163                 std::make_shared<Connection<T>>(Connection<T>::owner::server, _asioContext,
00164                 std::move(newSocket), _clientEndpoint, _qMessagesIn);
00165
00166             if (OnClientConnect(newConn)) {
00167                 _deqConnections.push_back(std::move(newConn));
00168                 _deqConnections.back()->ConnectToClient(this, _nIDCounter++);
00169                 std::cout << "[" << _deqConnections.back()->GetID()
00170                 << "]" Connection Approved" << std::endl;
00171             } else {
00172                 std::cout << "[-----] Connection Denied" << std::endl;
00173             }
00174         } else {
00175             std::cout << "[SERVER] New Connection Error: " << ec.message() << std::endl;
00176         }
00177     }
00178     WaitForClientMessage();
00179 });
00180 }
00181
00190 void MessageClient(std::shared_ptr<Connection<T>> client, const Message<T> &msg)
00191 {
00192     if (client && client->IsConnected()) {
00193         client->Send(msg);
00194     } else {
00195         OnClientDisconnect(client);
00196
00197         client.reset();

```

```

00198
00199         _deqConnections.erase(
00200             std::remove(_deqConnections.begin(), _deqConnections.end(), client),
00201             _deqConnections.end());
00202     }
00203 }
00204
00218 void MessageAllClients(
00219     const Message<T> &msg, std::shared_ptr<Connection<T>> pIgnoreClient = nullptr)
00220 {
00221     bool bInvalidClientExists = false;
00222
00223     for (auto &client : _deqConnections) {
00224         if (client && client->IsConnected()) {
00225             if (client != pIgnoreClient) {
00226                 client->Send(msg);
00227                 if (msg.header.id == TypeMessage::DestroyEntityMessage) {
00228                     client->_lastMsg = msg;
00229                     client->SetStatus(ServerStatus::WAITING);
00230                 }
00231                 if (msg.header.id == TypeMessage::GameTransitionMode) {
00232                     client->_lastMsg = msg;
00233                     client->SetStatus(ServerStatus::TRANSITION);
00234                 }
00235             }
00236         } else {
00237             OnClientDisconnect(client);
00238             client.reset();
00239             bInvalidClientExists = true;
00240         }
00241     }
00242
00243     if (bInvalidClientExists)
00244         _deqConnections.erase(
00245             std::remove(_deqConnections.begin(), _deqConnections.end(), nullptr),
00246             _deqConnections.end());
00247 }
00248
00259 UIEntityInformation UpdateInfoBar(int playerId)
00260 {
00261     UIEntityInformation entity;
00262     int clientId = GetPlayerClientId(playerId);
00263     int infoBarId = GetClientInfoBarId(clientId);
00264     auto playerHealth = _componentManager.getComponent<HealthComponent>(playerId);
00265     auto playerScore = _componentManager.getComponent<ScoreComponent>(playerId);
00266     auto barSpriteData = _componentManager.getComponent<SpriteDataComponent>(infoBarId);
00267     auto barTextData = _componentManager.getComponent<TextDataComponent>(infoBarId);
00268     if (playerHealth && barSpriteData && barTextData) {
00269         entity.uniqueID = infoBarId;
00270         entity.spriteData = *(barSpriteData.value());
00271         entity.textData = *(barTextData.value());
00272         entity.lives = playerHealth.value()->lives;
00273         entity.score = playerScore.value()->score;
00274     }
00275     return entity;
00276 }
00277
00293 void Update(size_t nMaxMessages = -1, bool bWait = false)
00294 {
00295     if (_nbrOfPlayers == 0) {
00296         _entityManager.removeAllEntities();
00297         _componentManager.removeAllComponents();
00298         _playerConnected = false;
00299         _bossActive = false;
00300         _bossDefeated = false;
00301         _endOfLevel = false;
00302         return;
00303     }
00304     if (_nbrOfPlayers > 0 && !_playerConnected) {
00305         _playerConnected = true;
00306         _clock = std::chrono::system_clock::now();
00307     }
00308     _level.SetSystem(_componentManager, _entityManager);
00309
00310     bool bUpdateEntities = false;
00311     std::chrono::system_clock::time_point newClock = std::chrono::system_clock::now();
00312
00313     std::thread t_level([this, newClock, &bUpdateEntities]() {
00314         _level.Update(this, _componentManager, _entityManager, newClock, &bUpdateEntities);
00315     });
00316
00317     auto setToAllClients = [this](ServerStatus type) {
00318         for (auto &client : _deqConnections) {
00319             if (client && client->IsConnected()) {
00320                 client->SetStatus(type);
00321             }
00322         }
00323     }

```

```

00323     };
00324
00325     if (_bossDefeated) {
00326         if (!_watingPlayersReady) {
00327             switch (_level.GetLevel()) {
00328                 case GameState::LevelOne: {
00329                     _level.ChangeLevel(GameState::LevelTwo);
00330                 } break;
00331                 case GameState::LevelTwo: {
00332                     _level.ChangeLevel(GameState::LevelThree);
00333                 } break;
00334                 case GameState::LevelThree: {
00335                     _level.ChangeLevel(GameState::Win);
00336                 } break;
00337
00338                 default:
00339                     break;
00340             }
00341             r_type::net::Message<T> msg;
00342             _watingPlayersReady = true;
00343             msg.header.id = TypeMessage::GameTransitionMode;
00344             MessageAllClients(msg);
00345         }
00346         if (_playerReady == _nbrOfPlayers) {
00347             _endOfLevel = false;
00348             _bossDefeated = false;
00349             _clock = std::chrono::system_clock::now();
00350             r_type::net::Message<T> msg;
00351             msg.header.id = TypeMessage::FinishInitialization;
00352             MessageAllClients(msg);
00353             setToAllClients(ServerStatus::RUNNING);
00354             _watingPlayersReady = false;
00355             _playerReady = 0;
00356             _level.ChangeBackground(this, _entityManager, _componentManager);
00357         }
00358     }
00359
00360     size_t nMessageCount = 0;
00361     while (nMessageCount < nMaxMessages && !_qMessagesIn.empty()) {
00362         auto msg = _qMessagesIn.pop_front();
00363
00364         OnMessage(msg.remote, msg.msg);
00365
00366         nMessageCount++;
00367     }
00368     t_level.join();
00369     if (bUpdateEntities)
00370         _clock = newClock;
00371 }
00372
00373 void UpdatePlayerPosition(PlayerMovement direction, uint32_t entityId) override
00374 {
00375     auto entitySpriteData = _componentManager.GetComponent<SpriteDataComponent>(entityId);
00376     EntityInformation entity;
00377
00378     auto hitbox = _componentManager.GetComponent<HitboxComponent>(entityId);
00379     auto pos = _componentManager.GetComponent<PositionComponent>(entityId);
00380
00381     vf2d newPos = {pos.value()->x, pos.value()->y};
00382     switch (direction) {
00383     case PlayerMovement::UP: {
00384         newPos.y -= 2;
00385     } break;
00386     case PlayerMovement::DOWN: {
00387         newPos.y += 2;
00388     } break;
00389     case PlayerMovement::LEFT: {
00390         newPos.x -= 2;
00391     } break;
00392     case PlayerMovement::RIGHT: {
00393         newPos.x += 2;
00394     } break;
00395     }
00396
00397     if (hitbox && pos) {
00398         float halfWidth = hitbox.value()->w / 2;
00399         float halfHeight = hitbox.value()->h / 2;
00400         float minX, maxX, minY, maxY;
00401
00402         maxX = ((newPos.x / 100) * SCREEN_WIDTH) + halfWidth;
00403         minX = ((newPos.x / 100) * SCREEN_WIDTH) - halfWidth;
00404         maxY = ((newPos.y / 100) * SCREEN_HEIGHT) + halfHeight;
00405         minY = ((newPos.y / 100) * SCREEN_HEIGHT) - halfHeight;
00406
00407         if (maxX > SCREEN_WIDTH || minX < 0 || maxY > (SCREEN_HEIGHT - 30) || minY < 0) {
00408             return;
00409         }
00410     }

```

```

00423         auto vel = _componentManager.getComponent<VelocityComponent>(entityId);
00424         if (pos && vel) {
00425             // player go down
00426             if (pos.value()->y < newPos.y) {
00427                 vel.value()->y -= 0.1;
00428                 if (vel.value()->y < -1) {
00429                     vel.value()->y = -1;
00430                 }
00431             } else if (pos.value()->y > newPos.y) {
00432                 vel.value()->y += 0.1;
00433                 if (vel.value()->y > 1) {
00434                     vel.value()->y = 1;
00435                 }
00436             }
00437             pos.value()->x = newPos.x;
00438             pos.value()->y = newPos.y;
00439
00440             auto frontComponent = _componentManager.getComponent<FrontComponent>(entityId);
00441             if (frontComponent.value()->targetId != -1) {
00442                 std::cout << "frontComponent.targetId: " << frontComponent.value()->targetId
00443                     << std::endl;
00444                 auto posFront = _componentManager.getComponent<PositionComponent>(
00445                     frontComponent.value()->targetId);
00446                 posFront.value()->x = newPos.x + 2;
00447                 posFront.value()->y = newPos.y;
00448                 vf2d newPosFront = {posFront.value()->x, posFront.value()->y};
00449                 uint32_t entityIdFront = frontComponent.value()->targetId;
00450                 r_type::net::Message<TypeMessage> moveMsg;
00451                 moveMsg.header.id = TypeMessage::MoveEntityMessage;
00452                 moveMsg << entityIdFront << newPosFront;
00453                 MessageAllClients(moveMsg);
00454             }
00455         }
00456
00457         // std::cout << "Pos: " << newPos.x << " " << newPos.y << std::endl;
00458         // //////////////////////////////////////
00459
00460         // Update entity information and send to all clients
00461         r_type::net::Message<TypeMessage> moveMsg;
00462         moveMsg.header.id = TypeMessage::MoveEntityMessage;
00463         moveMsg << entityId << newPos;
00464         MessageAllClients(moveMsg);
00465     }
00466 }
00467
00481 void SavePlayerScore(uint32_t playerId)
00482 {
00483     std::string directory = "GameScores";
00484     std::string filePath = directory + "/scores.txt";
00485     if (!std::filesystem::exists(directory)) {
00486         std::filesystem::create_directory(directory);
00487     }
00488     if (!std::filesystem::exists(filePath)) {
00489         std::ofstream outFile(filePath); // Creating the file
00490         if (!outFile) {
00491             throw failedToCreateFile();
00492         }
00493         outFile.close();
00494     }
00495     std::ofstream outFile(filePath, std::ios_base::app); // Open in append mode
00496     if (!outFile) {
00497         throw failedToOpenFile();
00498     }
00499     if (auto scoreComponent = _componentManager.getComponent<ScoreComponent>(playerId)) {
00500         std::string playerId = "Player " + std::to_string(playerId);
00501         outFile << playerId << ": " << scoreComponent.value()->score << "\n";
00502         outFile.close();
00503     }
00504 }
00505
00512 uint32_t GetClientPlayerId(uint32_t id) { return _clientPlayerID[id]; }
00513
00526 uint32_t GetPlayerClientId(uint32_t id)
00527 {
00528     for (const auto &pair : _clientPlayerID) {
00529         if (pair.second == id) {
00530             return pair.first;
00531         }
00532     }
00533     throw playerIdNotFound();
00534 }
00535
00542 uint32_t GetClientInfoBarId(uint32_t id) { return _clientInfoBarID[id]; }
00543
00552 void RemovePlayer(uint32_t id) { _clientPlayerID.erase(id); }
00553
00564 void RemoveEntity(uint32_t id)

```

```

00565     {
00566         if (auto entity = _entityManager.getEntity(id)) {
00567             _componentManager.removeEntityFromAllComponents(id);
00568             _entityManager.removeEntity(id);
00569         }
00570     }
00571
00572 void RemoveInfoBar(uint32_t infoBarId)
00573 {
00574     if (auto textDataComponent =
00575         _componentManager.getComponent<TextDataComponent>(infoBarId)) {
00576         for (uint32_t categoryId : textDataComponent.value()->categoryIds) {
00577             if (auto entity = _entityManager.getEntity(categoryId)) {
00578                 _entityManager.removeEntity(categoryId);
00579             }
00580         }
00581         RemoveEntity(infoBarId);
00582     }
00583     _clientInfoBarID.erase(infoBarId);
00584 }
00585
00586 void RemoveBossTail(int bossId)
00587 {
00588     if (auto bossComp = _componentManager.getComponent<BossComponent>(bossId)) {
00589         for (size_t i = 0; i < bossComp.value()->tailSegmentIds.size(); i++) {
00590             int tailSegId = bossComp.value()->tailSegmentIds[i];
00591             if (auto entity = _entityManager.getEntity(tailSegId)) {
00592                 r_type::net::Message<TypeMessage> msg;
00593                 msg.header.id = TypeMessage::DestroyEntityMessage;
00594                 msg << tailSegId;
00595                 MessageAllClients(msg);
00596                 RemoveEntity(tailSegId);
00597                 // std::cout << "Tail segment removed" << std::endl;
00598                 // //////////////////////////////////////
00599             }
00600         }
00601     }
00602     _bossDefeated = true;
00603 }
00604
00605 EntityInformation InitiatePlayer(int clientId)
00606 {
00607     EntityInformation entityInfo;
00608     Entity player =
00609         _entityFactory.createPlayer(_entityManager, _componentManager, _nbrOfPlayers);
00610     entityInfo.uniqueID = player.getId();
00611     auto playerSprite =
00612         _componentManager.getComponent<SpriteDataComponent>(entityInfo.uniqueID);
00613     auto playerPos = _componentManager.getComponent<PositionComponent>(entityInfo.uniqueID);
00614     auto animation = _componentManager.getComponent<AnimationComponent>(entityInfo.uniqueID);
00615     if (playerSprite && playerPos && animation) {
00616         entityInfo.ratio.x =
00617             (animation.value()->dimension.x * playerSprite.value()->scale.x) / SCREEN_WIDTH;
00618         entityInfo.ratio.y =
00619             (animation.value()->dimension.y * playerSprite.value()->scale.y) / SCREEN_HEIGHT;
00620         entityInfo.spriteData = *(playerSprite.value());
00621         entityInfo.vPos.x = playerPos.value()->x;
00622         entityInfo.vPos.y = playerPos.value()->y;
00623         entityInfo.animationComponent.dimension = animation.value()->dimension;
00624         entityInfo.animationComponent.offset = animation.value()->offset;
00625     }
00626     _clientPlayerID.insert_or_assign(clientId, entityInfo.uniqueID);
00627     return entityInfo;
00628 }
00629
00630 UIEntityInformation InitInfoBar(int clientId)
00631 {
00632     UIEntityInformation entityInfo;
00633     Entity infoBar = _entityFactory.createInfoBar(_entityManager, _componentManager);
00634     entityInfo.uniqueID = infoBar.getId();
00635     std::cout << "GameBarInformation" << std::endl;
00636     std::cout << "Entity ID: " << entityInfo.uniqueID << std::endl;
00637     std::cout << "Info: " << entityInfo.lives << ", " << entityInfo.score << std::endl;
00638     std::cout << "SpriteData: " << entityInfo.spriteData.spritePath << ", "
00639         << entityInfo.spriteData.scale.x << ", " << entityInfo.spriteData.scale.y << ", "
00640         << std::endl;
00641     auto spriteData = _componentManager.getComponent<SpriteDataComponent>(entityInfo.uniqueID);
00642     auto textData = _componentManager.getComponent<TextDataComponent>(entityInfo.uniqueID);
00643     auto health = _componentManager.getComponent<HealthComponent>(GetClientPlayerId(clientId));
00644     if (spriteData && textData && health) {
00645         entityInfo.spriteData = *(spriteData.value());
00646         entityInfo.textData = *(textData.value());
00647         entityInfo.textData.categorySize = textData.value()->categorySize;
00648         entityInfo.lives = health.value()->lives;
00649     }
00650     _clientInfoBarID.insert_or_assign(clientId, entityInfo.uniqueID);
00651     return entityInfo;
00652 }

```

```

00672     }
00673
00686 EntityInformation FormatEntityInformation(uint32_t entityId)
00687 {
00688     EntityInformation entityInfo;
00689     auto entityPos = _componentManager.getComponent<PositionComponent>(entityId);
00690     auto sprite = _componentManager.getComponent<SpriteDataComponent>(entityId);
00691     auto animation = _componentManager.getComponent<AnimationComponent>(entityId);
00692     if (entityPos && sprite) {
00693         entityInfo.uniqueID = entityId;
00694         entityInfo.vPos.x = entityPos.value()->x;
00695         entityInfo.vPos.y = entityPos.value()->y;
00696         entityInfo.spriteData = *(sprite.value());
00697         if (animation) {
00698             entityInfo.ratio.x =
00699                 (animation.value()->dimension.x * sprite.value()->scale.x) / SCREEN_WIDTH;
00700             entityInfo.ratio.y =
00701                 (animation.value()->dimension.y * sprite.value()->scale.y) / SCREEN_HEIGHT;
00702             entityInfo.animationComponent.dimension = animation.value()->dimension;
00703             entityInfo.animationComponent.offset = animation.value()->offset;
00704         }
00705     }
00706     return entityInfo;
00707 }
00708
00718 EntityInformation InitiatePlayerMissile(int entityId)
00719 {
00720     EntityInformation entityInfo;
00721     entityInfo.uniqueID = entityId;
00722     auto missilePos = _componentManager.getComponent<PositionComponent>(entityInfo.uniqueID);
00723     auto sprite = _componentManager.getComponent<SpriteDataComponent>(entityInfo.uniqueID);
00724     auto animation = _componentManager.getComponent<AnimationComponent>(entityInfo.uniqueID);
00725     if (missilePos && sprite) {
00726         entityInfo.vPos.x = missilePos.value()->x;
00727         entityInfo.vPos.y = missilePos.value()->y;
00728         entityInfo.spriteData = *(sprite.value());
00729         if (animation) {
00730             entityInfo.ratio.x =
00731                 (animation.value()->dimension.x * sprite.value()->scale.x) / SCREEN_WIDTH;
00732             entityInfo.ratio.y =
00733                 (animation.value()->dimension.y * sprite.value()->scale.y) / SCREEN_HEIGHT;
00734             entityInfo.animationComponent.dimension = animation.value()->dimension;
00735             entityInfo.animationComponent.offset = animation.value()->offset;
00736         }
00737     }
00738     return entityInfo;
00739 }
00740
00741 EntityInformation InitiateEnemyMissile(int enemyId)
00742 {
00743     EntityInformation entityInfo;
00744     Entity missile =
00745         _entityFactory.createEnemyMissile(_entityManager, _componentManager, enemyId);
00746     entityInfo.uniqueID = missile.getId();
00747     auto missilePos = _componentManager.getComponent<PositionComponent>(entityInfo.uniqueID);
00748     auto sprite = _componentManager.getComponent<SpriteDataComponent>(entityInfo.uniqueID);
00749     auto animation = _componentManager.getComponent<AnimationComponent>(entityInfo.uniqueID);
00750     if (missilePos && sprite) {
00751         entityInfo.vPos.x = missilePos.value()->x;
00752         entityInfo.vPos.y = missilePos.value()->y;
00753         if (animation) {
00754             entityInfo.ratio.x =
00755                 (animation.value()->dimension.x * sprite.value()->scale.x) / SCREEN_WIDTH;
00756             entityInfo.ratio.y =
00757                 (animation.value()->dimension.y * sprite.value()->scale.y) / SCREEN_HEIGHT;
00758             entityInfo.animationComponent.dimension = animation.value()->dimension;
00759             entityInfo.animationComponent.offset = animation.value()->offset;
00760         }
00761         entityInfo.spriteData = *(sprite.value());
00762     }
00763     return entityInfo;
00764 }
00765
00766 EntityInformation InitiateWeaponForce(int entityId)
00767 {
00768     EntityInformation entityInfo;
00769     entityInfo.uniqueID = entityId;
00770     auto position = _componentManager.getComponent<PositionComponent>(entityInfo.uniqueID);
00771     auto sprite = _componentManager.getComponent<SpriteDataComponent>(entityInfo.uniqueID);
00772     auto animation = _componentManager.getComponent<AnimationComponent>(entityInfo.uniqueID);
00773     if (position && sprite) {
00774         entityInfo.vPos.x = position.value()->x;
00775         entityInfo.vPos.y = position.value()->y;
00776         entityInfo.spriteData = *(sprite.value());
00777         if (animation) {
00778             entityInfo.ratio.x =
00779                 (animation.value()->dimension.x * sprite.value()->scale.x) / SCREEN_WIDTH;

```

```

00780         entityInfo.ratio.y =
00781             (animation.value()->dimension.y * sprite.value()->scale.y) / SCREEN_HEIGHT;
00782         entityInfo.animationComponent.dimension = animation.value()->dimension;
00783         entityInfo.animationComponent.offset = animation.value()->offset;
00784     }
00785 }
00786 return entityInfo;
00787 }
00788
00789 void InitBoss(r_type::net::AServer<T> *server)
00790 {
00791     server->_bossActive = true;
00792     // r_type::net::Message<TypeMessage> msg;
00793     // msg.header.id = TypeMessage::ChangeBackgroundMusic;
00794     // msg < 1;
00795     // server->MessageAllClients(msg);
00796     Entity boss = _entityFactory.createBoss(_entityManager, _componentManager, _entityFactory);
00797     int bossId = boss.getId();
00798     float segmentOffsetX = -2.0f;
00799     float segmentOffsetY = -2.0f;
00800     auto bossComp = _componentManager.getComponent<BossComponent>(bossId);
00801     auto bossPos = _componentManager.getComponent<PositionComponent>(bossId);
00802     auto bossSpriteData = _componentManager.getComponent<SpriteDataComponent>(bossId);
00803     auto bossAnimation = _componentManager.getComponent<AnimationComponent>(bossId);
00804
00805     if (bossComp && bossPos && bossSpriteData && bossAnimation) {
00806         int firstTailSegId = bossComp.value()->tailSegmentIds[0];
00807         if (auto firstTailSegPos =
00808             _componentManager.getComponent<PositionComponent>(firstTailSegId)) {
00809             firstTailSegPos.value()->x = 76;
00810             firstTailSegPos.value()->y = 88;
00811         }
00812
00813         for (size_t i = 1; i < bossComp.value()->tailSegmentIds.size(); i++) {
00814             int tailSegId = bossComp.value()->tailSegmentIds[i];
00815             if (auto tailSegPos =
00816                 _componentManager.getComponent<PositionComponent>(tailSegId)) {
00817                 int precedingTailSegId = bossComp.value()->tailSegmentIds[i - 1];
00818                 if (auto precedingTailSegPos =
00819                     _componentManager.getComponent<PositionComponent>(
00820                         precedingTailSegId)) {
00821                     tailSegPos.value()->x = precedingTailSegPos.value()->x + segmentOffsetX;
00822                     tailSegPos.value()->y = precedingTailSegPos.value()->y + segmentOffsetY;
00823                 }
00824             }
00825         }
00826
00827         EntityInformation bossEntityInfo;
00828         bossEntityInfo.uniqueID = bossId;
00829
00830         bossEntityInfo.vPos.x = bossPos.value()->x;
00831         bossEntityInfo.vPos.y = bossPos.value()->y;
00832         bossEntityInfo.spriteData = *(bossSpriteData.value());
00833         bossEntityInfo.ratio.x =
00834             (bossAnimation.value()->dimension.x * bossSpriteData.value()->scale.x) /
00835             SCREEN_WIDTH;
00836         bossEntityInfo.ratio.y =
00837             (bossAnimation.value()->dimension.y * bossSpriteData.value()->scale.y) /
00838             SCREEN_HEIGHT;
00839
00840         bossEntityInfo.animationComponent.dimension = bossAnimation.value()->dimension;
00841         bossEntityInfo.animationComponent.offset = bossAnimation.value()->offset;
00842
00843         r_type::net::Message<TypeMessage> bossMsg;
00844         bossMsg.header.id = TypeMessage::CreateEntityMessage;
00845         bossMsg < bossEntityInfo;
00846         server->MessageAllClients(bossMsg);
00847
00848         for (int i = 0; i != bossComp.value()->tailSegmentIds.size(); i++) {
00849             EntityInformation tailSegEntityInfo;
00850             tailSegEntityInfo.uniqueID = bossComp.value()->tailSegmentIds[i];
00851             auto tailSpriteData = _componentManager.getComponent<SpriteDataComponent>(
00852                 tailSegEntityInfo.uniqueID);
00853             auto tailPos =
00854                 _componentManager.getComponent<PositionComponent>(tailSegEntityInfo.uniqueID);
00855             auto tailAnimation =
00856                 _componentManager.getComponent<AnimationComponent>(tailSegEntityInfo.uniqueID);
00857             if (tailSpriteData && tailPos && tailAnimation) {
00858                 tailSegEntityInfo.vPos.x = tailPos.value()->x;
00859                 tailSegEntityInfo.vPos.y = tailPos.value()->y;
00860                 tailSegEntityInfo.spriteData = *(tailSpriteData.value());
00861                 tailSegEntityInfo.ratio.x =
00862                     (tailAnimation.value()->dimension.x * tailSpriteData.value()->scale.x) /
00863                     SCREEN_WIDTH;
00864                 tailSegEntityInfo.ratio.y =
00865                     (tailAnimation.value()->dimension.y * tailSpriteData.value()->scale.y) /
00866                     SCREEN_HEIGHT;

```



```

00867         tailSegEntityInfo.animationComponent.dimension =
00868             tailAnimation.value()->dimension;
00869         tailSegEntityInfo.animationComponent.offset = tailAnimation.value()->offset;
00870     }
00871     r_type::net::Message<TypeMessage> tailMsg;
00872     tailMsg.header.id = TypeMessage::CreateEntityMessage;
00873     tailMsg « tailSegEntityInfo;
00874     server->MessageAllClients(tailMsg);
00875 }
00876 }
00877 }
00878
00879 std::shared_ptr<Connection<T>> getClientById(
00880     const std::deque<std::shared_ptr<Connection<T>>> &connections, uint32_t clientId)
00881 {
00882     for (const auto &client : connections) {
00883         if (client && client->GetID() == clientId) {
00884             return client;
00885         }
00886     }
00887     return nullptr;
00888 }
00889
00890 virtual void OnClientValidated(std::shared_ptr<Connection<T>> client) {}
00900
00910 ComponentManager GetComponentManager() override { return _componentManager; }
00911
00921 EntityManager &GetEntityManager() override { return _entityManager; }
00922
00933 EntityFactory &GetEntityFactory() override { return _entityFactory; }
00934
00944 std::chrono::system_clock::time_point GetClock() override { return _clock; }
00945
00951 void SetClock(std::chrono::system_clock::time_point clock) { _clock = clock; }
00952
00953 protected:
00961 virtual bool OnClientConnect(std::shared_ptr<Connection<T>> client) { return false; }
00962
00968 virtual void OnClientDisconnect(std::shared_ptr<Connection<T>> client) {}
00969
00976 virtual void OnMessage(std::shared_ptr<Connection<T>> client, Message<T> &msg) {}
00977
00978 public:
00986 ThreadSafeQueue<OwnedMessage<T>> _qMessagesIn;
00987
00997 std::deque<std::shared_ptr<Connection<T>>> _deqConnections;
00998
01006 asio::io_context _asioContext;
01015 std::thread _threadContext;
01016
01023 asio::ip::udp::socket _asioSocket;
01030 asio::ip::udp::endpoint _clientEndpoint;
01031
01038 std::array<uint8_t, 1024> _tempBuffer;
01039
01047 uint32_t _nIDCounter = 10000;
01048
01056 ComponentManager _componentManager;
01064 EntityManager _entityManager;
01068 EntityFactory _entityFactory;
01069
01070 bool _endOfLevel = false;
01071 bool _bossActive = false;
01072 bool _bossDefeated = false;
01073 bool _waitingPlayersReady = false;
01074
01085 std::unordered_map<uint32_t, uint32_t> _clientPlayerID;
01086
01087 std::unordered_map<uint32_t, uint32_t> _clientInfoBarID;
01088
01092 int _nbrOfPlayers = 0;
01093
01101 std::chrono::system_clock::time_point _clock = std::chrono::system_clock::now();
01102 bool _playerConnected = false;
01103
01111 EntityInformation _background;
01112
01113 int _port;
01114
01115 r_type::Level<T> _level;
01116
01117 int _playerReady = 0;
01118 };
01119 } // namespace net
01120 } // namespace r_type

```

7.162 /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

- namespace [r_type](#)
- namespace [r_type::net](#)

7.163 server.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002  ** EPITECH PROJECT, 2024
00003  ** R-Type
00004  ** File description:
00005  ** main
00006  */
00007
00008 #pragma once
00009
00010 #include "a_server.hpp"
00011
00012 namespace r_type {
00013     namespace net {
00024         class Server : virtual public r_type::net::AServer<TypeMessage> {
00025             public:
00035                 Server(uint16_t nPort)
00036                     : r_type::net::IServer<TypeMessage>(), r_type::net::AServer<TypeMessage>(nPort)
00037                 {
00038                 }
00039
00047                 ~Server() {}
00048
00049             protected:
00056                 bool OnClientConnect(std::shared_ptr<r_type::net::Connection<TypeMessage> client);
00057
00064                 void OnClientDisconnect(std::shared_ptr<r_type::net::Connection<TypeMessage> client,
00065                                         r_type::net::Message<TypeMessage> &msg);
00066
00077                 void OnMessage(std::shared_ptr<r_type::net::Connection<TypeMessage> client,
00078                                r_type::net::Message<TypeMessage> &msg);
00079         };
00080     } // namespace net
00081 } // namespace r_type
```

7.164 /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)
- static `vf2d` `animationShipFactory` (`AnimationShip` animation)
Generates a vector representing the animation state of a ship.
- static `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)
- `vf2d` `animationBossFactory` (`AnimationBoss` animation)

7.164.1 Function Documentation

7.164.1.1 `animateForceMissileLevel1()`

```
static void animateForceMissileLevel1 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.164.1.2 `animateForceMissileLevel2()`

```
static void animateForceMissileLevel2 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.164.1.3 `animateForceMissileLevel3()`

```
static void animateForceMissileLevel3 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.164.1.4 `animateForceWeaponLevel1()`

```
static void animateForceWeaponLevel1 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.164.1.5 `animateForceWeaponLevel2()`

```
static void animateForceWeaponLevel2 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.164.1.6 animateForceWeaponLevel3()

```
static void animateForceWeaponLevel3 (
    std::optional< AnimationComponent * > & animation ) [static]
```

7.164.1.7 animationBasicMonsterFactory()

```
static vf2d animationBasicMonsterFactory (
    AnimationBasicMonster animation ) [static]
```

7.164.1.8 animationBossFactory()

```
vf2d animationBossFactory (
    AnimationBoss animation )
```

7.164.1.9 animationForceMissile1Factory()

```
static vf2d animationForceMissile1Factory (
    AnimationForceMissile1 animation ) [static]
```

7.164.1.10 animationForceMissile2Factory()

```
static vf2d animationForceMissile2Factory (
    AnimationForceMissile2 animation ) [static]
```

7.164.1.11 animationForceMissile3Factory()

```
static vf2d animationForceMissile3Factory (
    AnimationForceMissile3 animation ) [static]
```

7.164.1.12 animationForceWeapon1Factory()

```
static vf2d animationForceWeapon1Factory (
    AnimationForceWeapon1 animation ) [static]
```

7.164.1.13 animationForceWeapon2Factory()

```
static vf2d animationForceWeapon2Factory (
    AnimationForceWeapon2 animation ) [static]
```

7.164.1.14 animationForceWeapon3Factory()

```
static vf2d animationForceWeapon3Factory (
    AnimationForceWeapon3 animation ) [static]
```

7.164.1.15 animationShipFactory()

```
static vf2d animationShipFactory (
    AnimationShip animation ) [static]
```

Generates a vector representing the animation state of a ship.

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}.

7.164.1.16 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.

7.164.1.17 operator=="()

```
bool operator== (
    const vf2d & lhs,
    const vf2d & rhs )
```

7.165 /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](#), 189, 200
[174](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/label_component.hpp](#), 200
[/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/client.hpp](#), 200, 201
[175, 176](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/link_component.hpp](#), 201
[/home/runner/work/R-Type/R-Type/Client/Interface/Include/Net/i_client.hpp](#), 201
[178](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/motion_component.hpp](#), 201
[/home/runner/work/R-Type/R-Type/Client/Interface/Include/mainmenu.hpp](#), 201, 202
[173](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/offscreen_component.hpp](#), 203
[/home/runner/work/R-Type/R-Type/Client/Interface/Include/scenes.hpp](#), 203
[179, 180](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/on_screen_component.hpp](#), 203
[/home/runner/work/R-Type/R-Type/Client/Src/keyToString.cpp](#), 203, 204
[180](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/platform_component.hpp](#), 204
[/home/runner/work/R-Type/R-Type/Client/Src/main.cpp](#), 204
[181](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/platformer_component.hpp](#), 204
[/home/runner/work/R-Type/R-Type/Client/Src/scenes.cpp](#), 204, 205
[184](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/pool_component.hpp](#), 205
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ProjectileComponent.hpp](#), 205
[188](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/pool_component.hpp](#), 205
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ProjectileComponent.hpp](#), 205
[189](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/recoil_component.hpp](#), 206
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/RotationComponent.hpp](#), 206
[189, 190](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/score_component.hpp](#), 207
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ScoreComponent.hpp](#), 207
[190, 191](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_component.hpp](#), 207
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ShootComponent.hpp](#), 207
[191](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/shoot_monster_component.hpp](#), 208
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/ShootMonsterComponent.hpp](#), 208
[192](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_component.hpp](#), 209
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/SpriteComponent.hpp](#), 209
[192](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_component.hpp](#), 209
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/SpriteComponent.hpp](#), 209
[193](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_manager.hpp](#), 210
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/sprite_manager.hpp](#), 210
[194, 195](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp](#), 211
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/TextComponent.hpp](#), 211
[195, 196](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/text_component.hpp](#), 211
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/TextComponent.hpp](#), 211
[196](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/updown_component.hpp](#), 212
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/UpdownComponent.hpp](#), 212
[197](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/velocity_component.hpp](#), 213
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/VelComponent.hpp](#), 213
[197](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/weapon_component.hpp](#), 214
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/WeaponComponent.hpp](#), 214
[198](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity.hpp](#), 215
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/HealthComponent.hpp](#), 215
[198](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_factory.hpp](#), 216
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/HitComponent.hpp](#), 216
[199](#) [/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Entities/entity_manager.hpp](#), 217
[/home/runner/work/R-Type/R-Type/ECS/Interface/Include/Components/HitComponent.hpp](#), 217

- `_collisionSystem`
 - `r_type::Level< T >`, 130
- `_componentManager`
 - `AnimationSystem`, 25
 - `AutoFireSystem`, 62
 - `CollisionSystem`, 69
 - `MoveSystem`, 136
 - `r_type::net::AServer< T >`, 53
 - `RenderSystem`, 143
 - `UpdateSystem`, 169
- `_currentDaltonismMode`
 - `AScenes`, 35
- `_currentGameMode`
 - `AScenes`, 35
- `_currentMusicFilePath`
 - `AudioSystem`, 60
- `_currentScene`
 - `AScenes`, 35
- `_deqConnections`
 - `r_type::net::AServer< T >`, 53
- `_displayDaltonismChoice`
 - `AScenes`, 35
- `_displayGameModeChoice`
 - `AScenes`, 36
- `_displayKeyBindsChoice`
 - `AScenes`, 36
- `_endOfLevel`
 - `r_type::net::AServer< T >`, 53
- `_entityFactory`
 - `r_type::net::AServer< T >`, 53
- `_entityManager`
 - `AnimationSystem`, 25
 - `AutoFireSystem`, 62
 - `CollisionSystem`, 69
 - `MoveSystem`, 136
 - `r_type::net::AServer< T >`, 53
 - `UpdateSystem`, 169
- `_font`
 - `RenderSystem`, 143
- `_gameParameters`
 - `r_type::Level< T >`, 130
- `_id`
 - `Entity`, 75
- `_ip`
 - `AScenes`, 36
- `_level`
 - `r_type::net::AServer< T >`, 54
- `_moveSystem`
 - `r_type::Level< T >`, 130
- `_nIDCounter`
 - `r_type::net::AServer< T >`, 54
- `_nbrOfPlayers`
 - `r_type::net::AServer< T >`, 54
- `_networkClient`
 - `Scenes`, 151
- `_playerConnected`
 - `r_type::net::AServer< T >`, 54
- `_playerReady`
 - `AScenes`, 36
 - `r_type::net::AServer< T >`, 54
- `_port`
 - `AScenes`, 36
 - `r_type::net::AServer< T >`, 54
- `_previousScene`
 - `AScenes`, 36
- `_qMessagesIn`
 - `r_type::net::AServer< T >`, 54
- `_shooterEnemySpawnTime`
 - `r_type::Level< T >`, 130
- `_soundEffect`
 - `AudioSystem`, 60
- `_spawnTimeMonsterThree`
 - `r_type::Level< T >`, 131
- `_tempBuffer`
 - `r_type::net::AServer< T >`, 55
- `_threadContext`
 - `r_type::net::AServer< T >`, 55
- `_watingPlayersReady`
 - `r_type::net::AServer< T >`, 55
- `_window`
 - `RenderSystem`, 144
 - `Scenes`, 151
 - `UpdateSystem`, 170
- `~AClient`
 - `r_type::net::AClient< T >`, 16
- `~AScenes`
 - `AScenes`, 30
- `~AServer`
 - `r_type::net::AServer< T >`, 41
- `~IClient`
 - `r_type::net::IClient< T >`, 102
- `~IEntityFactory`
 - `IEntityFactory`, 106
- `~IScenes`
 - `IScenes`, 116
- `~ISystem`
 - `ISystem`, 118
- `~Level`
 - `r_type::Level< T >`, 121
- `~Scenes`
 - `Scenes`, 147
- `~Server`
 - `r_type::net::Server`, 155
- `AbstractScenes`, 15
- `AClient`
 - `r_type::net::AClient< T >`, 16
- `Actions`
 - `AScenes`, 28
- `ActionType`
 - `sound_path.hpp`, 230
- `addComponent`
 - `ComponentManager`, 70
- `addEntity`
 - `r_type::net::Client`, 66
- `ALLY`
 - `AScenes`, 30

- AllyComponent, [20](#)
- AllyMissileComponent, [20](#)
- animateBasicMonster
 - AnimationSystem, [22](#)
- animateBoss
 - AnimationSystem, [23](#)
- animateEntity
 - r_type::net::Client, [66](#)
- animateForceMissile
 - AnimationSystem, [23](#)
- animateForceMissileLevel1
 - animation_system.cpp, [273](#)
- animateForceMissileLevel2
 - animation_system.cpp, [273](#)
- animateForceMissileLevel3
 - animation_system.cpp, [273](#)
- animateForceWeapon
 - AnimationSystem, [23](#)
- animateForceWeaponLevel1
 - animation_system.cpp, [273](#)
- animateForceWeaponLevel2
 - animation_system.cpp, [273](#)
- animateForceWeaponLevel3
 - animation_system.cpp, [273](#)
- animatePlayer
 - AnimationSystem, [23](#)
- animation_component.hpp
 - operator!=, [189](#)
- animation_system.cpp
 - animateForceMissileLevel1, [273](#)
 - animateForceMissileLevel2, [273](#)
 - animateForceMissileLevel3, [273](#)
 - animateForceWeaponLevel1, [273](#)
 - animateForceWeaponLevel2, [273](#)
 - animateForceWeaponLevel3, [273](#)
 - animationBasicMonsterFactory, [274](#)
 - animationBossFactory, [274](#)
 - animationForceMissile1Factory, [274](#)
 - animationForceMissile2Factory, [274](#)
 - animationForceMissile3Factory, [274](#)
 - animationForceWeapon1Factory, [274](#)
 - animationForceWeapon2Factory, [274](#)
 - animationForceWeapon3Factory, [274](#)
 - animationShipFactory, [274](#)
 - operator!=, [275](#)
 - operator==, [275](#)
- animation_system.hpp
 - AnimationBasicMonster, [248](#)
 - AnimationBoss, [248](#)
 - AnimationForceMissile1, [249](#)
 - AnimationForceMissile2, [249](#)
 - AnimationForceMissile3, [249](#)
 - AnimationForceWeapon1, [249](#)
 - AnimationForceWeapon2, [250](#)
 - AnimationForceWeapon3, [250](#)
 - AnimationShip, [250](#)
 - BASIC_MONSTER_1, [248](#)
 - BASIC_MONSTER_2, [248](#)
 - BASIC_MONSTER_3, [248](#)
 - BASIC_MONSTER_4, [248](#)
 - BASIC_MONSTER_5, [248](#)
 - BASIC_MONSTER_6, [248](#)
 - BASIC_MONSTER_7, [248](#)
 - BASIC_MONSTER_DEFAULT, [248](#)
 - BOSS_1, [249](#)
 - BOSS_2, [249](#)
 - BOSS_3, [249](#)
 - BOSS_DEFAULT, [249](#)
 - FORCE_MISSILE_1, [249](#)
 - FORCE_MISSILE_2, [249](#)
 - FORCE_MISSILE_3, [249](#)
 - FORCE_MISSILE_4, [249](#)
 - FORCE_MISSILE_5, [249](#)
 - FORCE_MISSILE_6, [249](#)
 - FORCE_MISSILE_7, [249](#)
 - FORCE_MISSILE_DEFAULT, [249](#)
 - FORCE_WEAPON_1, [250](#)
 - FORCE_WEAPON_2, [250](#)
 - FORCE_WEAPON_3, [250](#)
 - FORCE_WEAPON_4, [250](#)
 - FORCE_WEAPON_5, [250](#)
 - FORCE_WEAPON_DEFAULT, [249](#), [250](#)
 - operator!=, [251](#)
 - SHIP_DOWN, [250](#)
 - SHIP_FLIP_DOWN, [250](#)
 - SHIP_FLIP_UP, [250](#)
 - SHIP_STRAIT, [250](#)
 - SHIP_UP, [250](#)
- AnimationBasicMonster
 - animation_system.hpp, [248](#)
- animationBasicMonsterFactory
 - animation_system.cpp, [274](#)
- AnimationBoss
 - animation_system.hpp, [248](#)
- animationBossFactory
 - animation_system.cpp, [274](#)
- AnimationComponent, [20](#)
 - AnimationComponent, [20](#)
 - dimension, [21](#)
 - offset, [21](#)
- animationComponent
 - EntityInformation, [87](#)
- AnimationEntities
 - AnimationSystem, [24](#)
- AnimationForceMissile1
 - animation_system.hpp, [249](#)
- animationForceMissile1Factory
 - animation_system.cpp, [274](#)
- AnimationForceMissile2
 - animation_system.hpp, [249](#)
- animationForceMissile2Factory
 - animation_system.cpp, [274](#)
- AnimationForceMissile3
 - animation_system.hpp, [249](#)
- animationForceMissile3Factory
 - animation_system.cpp, [274](#)

AnimationForceWeapon1
 animation_system.hpp, 249
animationForceWeapon1Factory
 animation_system.cpp, 274
AnimationForceWeapon2
 animation_system.hpp, 250
animationForceWeapon2Factory
 animation_system.cpp, 274
AnimationForceWeapon3
 animation_system.hpp, 250
animationForceWeapon3Factory
 animation_system.cpp, 274
AnimationShip
 animation_system.hpp, 250
animationShipFactory
 animation_system.cpp, 274
AnimationSystem, 21
 _componentManager, 25
 _entityManager, 25
 animateBasicMonster, 22
 animateBoss, 23
 animateForceMissile, 23
 animateForceWeapon, 23
 animatePlayer, 23
 AnimationEntities, 24
 AnimationSystem, 22
AnimationUpdate
 r_type::Level< T >, 122
AScenes, 25
 _currentDaltonismMode, 35
 _currentGameMode, 35
 _currentScene, 35
 _displayDaltonismChoice, 35
 _displayGameModeChoice, 36
 _displayKeyBindsChoice, 36
 _ip, 36
 _playerReady, 36
 _port, 36
 _previousScene, 36
 ~AScenes, 30
Actions, 28
ALLY, 30
AScenes, 30
BACKGROUND, 30
buttons, 36
CHOOSE_DIFFICULTY, 29
CUSTOM_DIFFICULTY, 29
DaltonismMode, 28
DEUTERANOPIA, 28
DOWN, 28
EASY, 29
ENEMY, 30
EXIT, 29
FILTER, 30
filter, 37
FIRE, 28
GAME_LOOP, 29
GameMode, 28
getDaltonism, 30
getDisplayDaltonismChoice, 30
getDisplayGameModeChoice, 31
getDisplayKeyBindsChoice, 31
getGameMode, 31
getIp, 31
GetPlayerReady, 32
getPort, 32
getPreviousScene, 32
HARD, 29
IN_GAME_MENU, 29
keyBinds, 37
LEFT, 28
MAIN_MENU, 29
MEDIUM, 29
NORMAL, 28
OTHER, 30
PAUSE, 28
PLAYER, 30
POWER_UP, 30
PROTANOPIA, 28
QUIT, 28
RIGHT, 28
Scene, 29
setDaltonism, 32
setDisplayDaltonismChoice, 33
setDisplayGameModeChoice, 33
setDisplayKeyBindsChoice, 33
setGameMode, 33
setIp, 34
SetPlayerReady, 34
setPort, 34
setScene, 35
SETTINGS_MENU, 29
SpriteType, 29
TRANSITION_LEVEL, 29
TRITANOPIA, 28
UI, 30
UP, 28
WEAPON, 30
AServer
 r_type::net::AServer< T >, 40
attached
 ForceWeaponComponent, 99
AudioManager, 55
 getSoundBuffer, 56
 soundBuffers, 57
AudioSystem, 57
 _audioManager, 59
 _backgroundMusic, 59
 _currentMusicFilePath, 60
 _soundEffect, 60
 AudioSystem, 58
 playBackgroundMusic, 59
 playSoundEffect, 59
 stopBackgroundMusic, 59
AutoFireSystem, 60
 _componentManager, 62

- [_entityManager, 62](#)
 - [AutoFireSystem, 61](#)
 - [handleAutoFire, 61](#)
- BACKGROUND
 - [AScenes, 30](#)
- Background
 - [sound_path.hpp, 231](#)
- Background1
 - [sprite_path.hpp, 233](#)
- Background2
 - [sprite_path.hpp, 233](#)
- Background3
 - [sprite_path.hpp, 233](#)
- BackgroundComponent, [62](#)
- backgroundFactory
 - [EntityFactory, 77](#)
- Bar
 - [sprite_path.hpp, 233](#)
- BASIC_MONSTER_1
 - [animation_system.hpp, 248](#)
- BASIC_MONSTER_2
 - [animation_system.hpp, 248](#)
- BASIC_MONSTER_3
 - [animation_system.hpp, 248](#)
- BASIC_MONSTER_4
 - [animation_system.hpp, 248](#)
- BASIC_MONSTER_5
 - [animation_system.hpp, 248](#)
- BASIC_MONSTER_6
 - [animation_system.hpp, 248](#)
- BASIC_MONSTER_7
 - [animation_system.hpp, 248](#)
- BASIC_MONSTER_DEFAULT
 - [animation_system.hpp, 248](#)
- BasicMonster
 - [IEntityFactory, 106](#)
- BasicMonsterComponent, [62](#)
- bind
 - [BindComponent, 63](#)
- BindComponent, [63](#)
 - [bind, 63](#)
 - [BindComponent, 63](#)
 - [isHovered, 63](#)
- BlueLaserCrystal
 - [sprite_path.hpp, 233](#)
- Boss
 - [IEntityFactory, 106](#)
 - [sound_path.hpp, 231](#)
 - [sprite_path.hpp, 233](#)
- BOSS_1
 - [animation_system.hpp, 249](#)
- BOSS_2
 - [animation_system.hpp, 249](#)
- BOSS_3
 - [animation_system.hpp, 249](#)
- BOSS_DEFAULT
 - [animation_system.hpp, 249](#)
- BossBullet
 - [sprite_path.hpp, 233](#)
- BossComponent, [64](#)
 - [tailSegmentIds, 64](#)
- BossDeath
 - [sound_path.hpp, 231](#)
- buttons
 - [AScenes, 36](#)
- canShoot
 - [ShootComponent, 159](#)
- categoryIds
 - [TextDataComponent, 164](#)
- categorySize
 - [TextDataComponent, 164](#)
- categoryTexts
 - [TextDataComponent, 164](#)
- ChangeBackground
 - [r_type::Level< T >, 122](#)
- ChangeLevel
 - [r_type::Level< T >, 123](#)
- charSize
 - [TextDataComponent, 164](#)
- checkCollision
 - [CollisionSystem, 68](#)
- CheckCollisionLogic
 - [hitbox_tmp.cpp, 244](#)
- CheckEntityMovement
 - [hitbox_tmp.cpp, 244](#)
 - [hitbox_tmp.hpp, 227](#)
- CheckEntityPosition
 - [hitbox_tmp.cpp, 244](#)
 - [hitbox_tmp.hpp, 227](#)
- checkOffScreen
 - [CollisionSystem, 69](#)
- CHOOSE_DIFFICULTY
 - [AScenes, 29](#)
- CIRCLE
 - [movement_component.hpp, 202](#)
- collisionAction
 - [r_type::Level< T >, 123](#)
- CollisionSystem, [67](#)
 - [_componentManager, 69](#)
 - [_entityManager, 69](#)
 - [checkCollision, 68](#)
 - [checkOffScreen, 69](#)
 - [CollisionSystem, 68](#)
- CollisionUpdate
 - [r_type::Level< T >, 124](#)
- ComponentManager, [70](#)
 - [addComponent, 70](#)
 - [components, 72](#)
 - [getComponent, 71](#)
 - [getComponentMap, 71](#)
 - [removeAllComponents, 71](#)
 - [removeEntityFromAllComponents, 72](#)
 - [removeEntityFromComponent, 72](#)
- componentNotFound, [73](#)
 - [what, 73](#)
- components

- ComponentManager, 72
- Connect
 - r_type::net::AClient< T >, 17
 - r_type::net::IClient< T >, 102
- cooldownTime
 - ShootComponent, 159
- creatable_client_object.hpp
 - CreatableClientObject, 214
 - NONE, 214
 - PLAYERMISSILE, 214
- CreatableClientObject
 - creatable_client_object.hpp, 214
- createBackgroundLevelOne
 - EntityFactory, 77
 - IEntityFactory, 106
- createBackgroundLevelThree
 - EntityFactory, 78
 - IEntityFactory, 106
- createBackgroundLevelTwo
 - EntityFactory, 78
 - IEntityFactory, 107
- createBackgroundMenu
 - EntityFactory, 78
 - IEntityFactory, 107
- createBasicMonster
 - EntityFactory, 79
 - IEntityFactory, 108
- createBoss
 - EntityFactory, 79
- createButton
 - EntityFactory, 80
 - IEntityFactory, 108
- createDaltonismChoiceButtons
 - scenes.cpp, 184
- createEnemyMissile
 - EntityFactory, 80
 - IEntityFactory, 109
- createEntity
 - EntityManager, 88
- createFilter
 - EntityFactory, 81
- createForceMissile
 - EntityFactory, 81
 - IEntityFactory, 109
- createForceWeapon
 - EntityFactory, 82
 - IEntityFactory, 110
- createInfoBar
 - EntityFactory, 82
 - IEntityFactory, 110
- createKeyBindingButtons
 - scenes.cpp, 184
- createPlayer
 - EntityFactory, 83
 - IEntityFactory, 111
- createPlayerMissile
 - EntityFactory, 83
 - IEntityFactory, 111
- createPowerUpBlueLaserCrystal
 - EntityFactory, 84
 - IEntityFactory, 111
- createShooterEnemy
 - EntityFactory, 84
 - IEntityFactory, 112
- createSmallButton
 - EntityFactory, 85
 - IEntityFactory, 112
- createTailEnd
 - EntityFactory, 85
 - IEntityFactory, 113
- createTailSegment
 - EntityFactory, 85
 - IEntityFactory, 113
- createUpdateButton
 - EntityFactory, 85
 - IEntityFactory, 113
- createWall
 - EntityFactory, 86
 - IEntityFactory, 113
- CUSTOM_DIFFICULTY
 - AScenes, 29
- DaltonismMode
 - AScenes, 28
- DEUTERANOPIA
 - AScenes, 28
- DIAGONAL
 - movement_component.hpp, 202
- difficultyChoices
 - IScenes, 116
 - Scenes, 148
- difficultyChoicesCustomization
 - Scenes, 148
- dimension
 - AnimationComponent, 21
- Disconnect
 - r_type::net::AClient< T >, 17
 - r_type::net::IClient< T >, 102
- displayEndOfGame
 - r_type::net::Client, 66
- DOWN
 - AScenes, 28
 - input_component.hpp, 200
- EASY
 - AScenes, 29
- EndOfGame
 - r_type::Level< T >, 124
- ENEMY
 - AScenes, 30
- Enemy1
 - sprite_path.hpp, 232
- Enemy2
 - sprite_path.hpp, 232
- Enemy3
 - sprite_path.hpp, 232
- EnemyComponent, 74

- EnemyMissileComponent, 74
- EnemyType
 - IEntityFactory, 106
- entities
 - EntityManager, 90
- Entity, 74
 - _id, 75
 - Entity, 74
 - getId, 75
- entity_factory.cpp
 - operator<<, 241, 242
- EntityFactory, 75
 - backgroundFactory, 77
 - createBackgroundLevelOne, 77
 - createBackgroundLevelThree, 78
 - createBackgroundLevelTwo, 78
 - createBackgroundMenu, 78
 - createBasicMonster, 79
 - createBoss, 79
 - createButton, 80
 - createEnemyMissile, 80
 - createFilter, 81
 - createForceMissile, 81
 - createForceWeapon, 82
 - createInfoBar, 82
 - createPlayer, 83
 - createPlayerMissile, 83
 - createPowerUpBlueLaserCrystal, 84
 - createShooterEnemy, 84
 - createSmallButton, 85
 - createTailEnd, 85
 - createTailSegment, 85
 - createUpdateButton, 85
 - createWall, 86
- EntityInformation, 86
 - animationComponent, 87
 - ratio, 87
 - spriteData, 87
 - uniqueID, 87
 - vPos, 87
- EntityManager, 87
 - createEntity, 88
 - entities, 90
 - entityNb, 90
 - getAllEntities, 88
 - getEntity, 88
 - removeAllEntities, 90
 - removeEntity, 90
- entityNb
 - EntityManager, 90
- entityNotFound, 91
 - what, 91
- EXIT
 - AScenes, 29
- Explosion
 - sound_path.hpp, 231
- failedToCreateFile, 92
 - what, 92
- failedToLoadFont, 92
 - what, 93
- failedToLoadSound, 93
 - what, 94
- failedToLoadTexture, 94
 - what, 94
- failedToOpenFile, 95
 - what, 95
- FILTER
 - AScenes, 30
- filter
 - AScenes, 37
- FIRE
 - AScenes, 28
- FireUpdate
 - r_type::Level< T >, 124
- font_path.cpp
 - FontFactory, 242
- font_path.hpp
 - FontFactory, 223
 - FontPath, 223
 - MAIN, 223
 - NONE, 223
 - operator<<, 224
- FontFactory
 - font_path.cpp, 242
 - font_path.hpp, 223
- FontManager, 95
 - fonts, 97
 - getFont, 96
 - releaseFont, 96
- FontPath
 - font_path.hpp, 223
- fontPath
 - TextDataComponent, 164
- fonts
 - FontManager, 97
- FORCE_MISSILE_1
 - animation_system.hpp, 249
- FORCE_MISSILE_2
 - animation_system.hpp, 249
- FORCE_MISSILE_3
 - animation_system.hpp, 249
- FORCE_MISSILE_4
 - animation_system.hpp, 249
- FORCE_MISSILE_5
 - animation_system.hpp, 249
- FORCE_MISSILE_6
 - animation_system.hpp, 249
- FORCE_MISSILE_7
 - animation_system.hpp, 249
- FORCE_MISSILE_DEFAULT
 - animation_system.hpp, 249
- FORCE_WEAPON_1
 - animation_system.hpp, 250
- FORCE_WEAPON_2
 - animation_system.hpp, 250
- FORCE_WEAPON_3

- animation_system.hpp, 250
- FORCE_WEAPON_4
 - animation_system.hpp, 250
- FORCE_WEAPON_5
 - animation_system.hpp, 250
- FORCE_WEAPON_DEFAULT
 - animation_system.hpp, 249, 250
- forceld
 - ForceMissileComponent, 97
- ForceMissile
 - sprite_path.hpp, 233
- ForceMissileComponent, 97
 - forceld, 97
- ForceWeapon
 - sprite_path.hpp, 233
- ForceWeaponComponent, 98
 - attached, 99
 - ForceWeaponComponent, 98
 - level, 99
 - playerId, 99
- FormatEntityInformation
 - r_type::net::AServer< T >, 41
- FrontComponent, 99
 - FrontComponent, 100
 - targetId, 100
- GAME_LOOP
 - AScenes, 29
- game_text.cpp
 - GameTextFactory, 243
- game_text.hpp
 - GameText, 225
 - GameTextFactory, 225
 - Lives, 225
 - NONE, 225
 - operator<<, 225
 - Score, 225
- gameLoop
 - IScenes, 116
 - Scenes, 148
- GameMode
 - AScenes, 28
- GameOver
 - sound_path.hpp, 231
- GameText
 - game_text.hpp, 225
- GameTextFactory
 - game_text.cpp, 243
 - game_text.hpp, 225
- getAllEntities
 - EntityManager, 88
- getClientById
 - r_type::net::AServer< T >, 41
- GetClientInfoBarId
 - r_type::net::AServer< T >, 41
- GetClientPlayerId
 - r_type::net::AServer< T >, 42
- GetClock
 - r_type::net::AServer< T >, 42
- getComponent
 - ComponentManager, 71
- GetComponentManager
 - r_type::net::AServer< T >, 42
- GetComponentMap
 - ComponentManager, 71
- getConnection
 - r_type::net::AClient< T >, 17
- getDaltonism
 - AScenes, 30
- getDisplayDaltonismChoice
 - AScenes, 30
- getDisplayGameModeChoice
 - AScenes, 31
- getDisplayKeyBindsChoice
 - AScenes, 31
- getEntity
 - EntityManager, 88
- GetEntityBackGround
 - r_type::Level< T >, 125
- GetEntityFactory
 - r_type::net::AServer< T >, 43
- GetEntityManager
 - r_type::net::AServer< T >, 43
- getFont
 - FontManager, 96
- getGameMode
 - AScenes, 31
- getId
 - Entity, 75
- getIp
 - AScenes, 31
- GetLevel
 - r_type::Level< T >, 125
- GetPlayerClientId
 - r_type::net::AServer< T >, 43
- getPlayerId
 - r_type::net::AClient< T >, 17
- GetPlayerReady
 - AScenes, 32
- getPort
 - AScenes, 32
- getPreviousScene
 - AScenes, 32
- getRenderWindow
 - IScenes, 116
 - Scenes, 148
- getSoundBuffer
 - AudioManager, 56
- getTexture
 - TextureManager, 165
- getWindowSize
 - r_type::net::AClient< T >, 17
- h
 - HitboxComponent, 101
- handleAutoFire
 - AutoFireSystem, 61
- handleEvents

- scenes.cpp, 184
- HandleMessage
 - Scenes, 148
- HandleTransitionLevelMessage
 - Scenes, 148
- HARD
 - AScenes, 29
- HealthComponent, 100
 - lives, 100
- hitbox_tmp.cpp
 - CheckCollisionLogic, 244
 - CheckEntityMovement, 244
 - CheckEntityPosition, 244
- hitbox_tmp.hpp
 - CheckEntityMovement, 227
 - CheckEntityPosition, 227
- HitboxComponent, 101
 - h, 101
 - w, 101
- hitboxX
 - SpriteComponent, 160
- hitboxY
 - SpriteComponent, 160
- IClient
 - r_type::net::IClient< T >, 102
- IEntityFactory, 104
 - ~IEntityFactory, 106
 - BasicMonster, 106
 - Boss, 106
 - createBackgroundLevelOne, 106
 - createBackgroundLevelThree, 106
 - createBackgroundLevelTwo, 107
 - createBackgroundMenu, 107
 - createBasicMonster, 108
 - createButton, 108
 - createEnemyMissile, 109
 - createForceMissile, 109
 - createForceWeapon, 110
 - createInfoBar, 110
 - createPlayer, 111
 - createPlayerMissile, 111
 - createPowerUpBlueLaserCrystal, 111
 - createShooterEnemy, 112
 - createSmallButton, 112
 - createTailEnd, 113
 - createTailSegment, 113
 - createUpdateButton, 113
 - createWall, 113
 - EnemyType, 106
 - ShooterEnemy, 106
 - Wall, 106
- IN_GAME_MENU
 - AScenes, 29
- Incoming
 - r_type::net::AClient< T >, 17
 - r_type::net::IClient< T >, 103
- index
 - MovementComponent, 134
- inGameMenu
 - IScenes, 116
 - Scenes, 148
- InitBoss
 - r_type::net::AServer< T >, 44
- InitiateBackground
 - r_type::Level< T >, 125
- InitiateEnemyMissile
 - r_type::net::AServer< T >, 44
- InitiatePlayer
 - r_type::net::AServer< T >, 44
- InitiatePlayerMissile
 - r_type::net::AServer< T >, 44
- InitiateWeaponForce
 - r_type::net::AServer< T >, 45
- InitInfoBar
 - r_type::net::AServer< T >, 45
- initInfoBar
 - r_type::net::Client, 66
- input
 - InputComponent, 115
- input_component.hpp
 - DOWN, 200
 - InputType, 199
 - LEFT, 200
 - NONE, 200
 - QUIT, 200
 - RIGHT, 200
 - SHOOT, 200
 - UP, 200
- InputComponent, 114
 - input, 115
- InputType
 - input_component.hpp, 199
- IScenes, 115
 - ~IScenes, 116
 - difficultyChoices, 116
 - gameLoop, 116
 - getRenderWindow, 116
 - inGameMenu, 116
 - mainMenu, 116
 - render, 117
 - settingsMenu, 117
 - shouldQuit, 117
- isClicked
 - OnClickComponent, 138
- IsConnected
 - r_type::net::AClient< T >, 18
 - r_type::net::IClient< T >, 103
- isHovered
 - BindComponent, 63
- isValidIPv4
 - main.cpp, 181
- isValidPort
 - main.cpp, 181, 182
- ISystem, 117
 - ~ISystem, 118
 - ISystem, 118

- keyBinds
 - AScenes, 37
- keyToString
 - keyToString.cpp, 181
 - scenes.hpp, 179
- keyToString.cpp
 - keyToString, 181
- labelComponent, 118
 - name, 119
 - x, 119
 - y, 119
- LEFT
 - AScenes, 28
 - input_component.hpp, 200
- Level
 - r_type::Level< T >, 121
- level
 - ForceWeaponComponent, 99
- LevelOne
 - r_type::Level< T >, 126
- LevelThree
 - r_type::Level< T >, 126
- LevelTwo
 - r_type::Level< T >, 126
- LinkForceComponent, 131
 - LinkForceComponent, 132
 - targetId, 132
- Lives
 - game_text.hpp, 225
- lives
 - HealthComponent, 100
 - UIEntityInformation, 167
- loopRunning
 - main.cpp, 183
- 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, 229
 - SCREEN_WIDTH, 229
- MAIN
 - font_path.hpp, 223
- main
 - main.cpp, 181, 183
- main.cpp
 - isValidIPv4, 181
 - isValidPort, 181, 182
 - loopRunning, 183
 - main, 181, 183
 - signal_handler, 183
- MAIN_MENU
 - AScenes, 29
- MainMenu
 - mainmenu.hpp, 173
- mainMenu
 - IScenes, 116
 - Scenes, 149
- mainmenu.hpp
 - MainMenu, 173
- MEDIUM
 - AScenes, 29
- MessageAll
 - r_type::net::Client, 66
- MessageAllClients
 - r_type::net::AServer< T >, 45
- MessageClient
 - r_type::net::AServer< T >, 46
- Missile
 - sprite_path.hpp, 233
- move
 - MovementComponent, 134
- moveEntities
 - MoveSystem, 135
- moveEntity
 - MoveSystem, 136
 - r_type::net::Client, 66
- movement_component.hpp
 - CIRCLE, 202
 - DIAGONAL, 202
 - MovementType, 202
 - NONE, 202
 - STRAIGHT, 202
 - SWEEPING, 202
 - WIGGLE, 202
- MovementComponent, 132
 - index, 134
 - move, 134
 - MovementComponent, 133
 - movementType, 134
- MovementType
 - movement_component.hpp, 202
- movementType
 - MovementComponent, 134
- MoveSystem, 134
 - _componentManager, 136
 - _entityManager, 136
 - moveEntities, 135
 - moveEntity, 136
 - MoveSystem, 135
- MoveUpdate
 - r_type::Level< T >, 127
- name
 - labelComponent, 119
- nextShootTime
 - ShootComponent, 159
- NONE
 - creatable_client_object.hpp, 214
 - font_path.hpp, 223
 - game_text.hpp, 225
 - input_component.hpp, 200
 - movement_component.hpp, 202
 - sound_path.hpp, 231

- NORMAL
 - AScenes, 28
- offset
 - AnimationComponent, 21
 - OffsetComponent, 137
- OffsetComponent, 137
 - offset, 137
- onClick
 - OnClickComponent, 138
- OnClickComponent, 137
 - isClicked, 138
 - onClick, 138
 - OnClickComponent, 138
- OnClientConnect
 - r_type::net::AServer< T >, 46
 - r_type::net::Server, 155
- OnClientDisconnect
 - r_type::net::AServer< T >, 46
 - r_type::net::Server, 156
- OnClientValidated
 - r_type::net::AServer< T >, 47
- OnMessage
 - r_type::net::AServer< T >, 47
 - r_type::net::Server, 156
- operator!=
 - animation_component.hpp, 189
 - animation_system.cpp, 275
 - animation_system.hpp, 251
- operator<<
 - entity_factory.cpp, 241, 242
 - font_path.hpp, 224
 - game_text.hpp, 225
 - sprite_data_component.hpp, 210
 - sprite_path.hpp, 233
- operator==
 - animation_system.cpp, 275
- OTHER
 - AScenes, 30
- PAUSE
 - AScenes, 28
- PingServer
 - r_type::net::Client, 66
- playBackgroundMusic
 - AudioSystem, 59
- PLAYER
 - AScenes, 30
- PlayerComponent, 138
- playerId
 - ForceWeaponComponent, 99
 - PlayerMissileComponent, 140
 - r_type::net::AClient< T >, 19
- playerIdNotFound, 139
 - what, 139
- PLAYERMISSILE
 - creatable_client_object.hpp, 214
- PlayerMissileComponent, 139
 - playerId, 140
- playSoundEffect
 - AudioSystem, 59
- PositionComponent, 140
 - PositionComponent, 140
 - x, 141
 - y, 141
- POWER_UP
 - AScenes, 30
- PowerUp
 - sound_path.hpp, 231
- PowerUpComponent, 141
- PROTANOPIA
 - AScenes, 28
- QUIT
 - AScenes, 28
 - input_component.hpp, 200
- r_type, 13
- r_type::Level< T >, 119
 - _WallSpawnTime, 131
 - _animationSystem, 129
 - _autoFireSystem, 129
 - _basicMonsterSpawnTime, 130
 - _collisionSystem, 130
 - _gameParameters, 130
 - _moveSystem, 130
 - _shooterEnemySpawnTime, 130
 - _spawnTimeMonsterThree, 131
 - ~Level, 121
 - AnimationUpdate, 122
 - ChangeBackground, 122
 - ChangeLevel, 123
 - collisionAction, 123
 - CollisionUpdate, 124
 - EndOfGame, 124
 - FireUpdate, 124
 - GetEntityBackGround, 125
 - GetLevel, 125
 - InitiateBackground, 125
 - Level, 121
 - LevelOne, 126
 - LevelThree, 126
 - LevelTwo, 126
 - MoveUpdate, 127
 - SetGameParameters, 127
 - SetSystem, 128
 - SpawnEntity, 128
 - Update, 129
- r_type::net, 13
- r_type::net::AClient< T >, 15
 - ~AClient, 16
 - AClient, 16
 - Connect, 17
 - Disconnect, 17
 - getConnection, 17
 - getPlayerId, 17
 - getWindowSize, 17
 - Incoming, 17

- IsConnected, 18
- m_connection, 19
- m_context, 19
- m_qMessagesIn, 19
- playerId, 19
- Send, 18
- setPlayerId, 18
- setWindowSize, 18
- thrContext, 19
- windowSize, 19
- r_type::net::AServer< T >, 38
 - _asioContext, 51
 - _asioSocket, 51
 - _background, 51
 - _bossActive, 52
 - _bossDefeated, 52
 - _clientEndpoint, 52
 - _clientInfoBarID, 52
 - _clientPlayerID, 52
 - _clock, 52
 - _componentManager, 53
 - _deqConnections, 53
 - _endOfLevel, 53
 - _entityFactory, 53
 - _entityManager, 53
 - _level, 54
 - _nIDCounter, 54
 - _nbrOfPlayers, 54
 - _playerConnected, 54
 - _playerReady, 54
 - _port, 54
 - _qMessagesIn, 54
 - _tempBuffer, 55
 - _threadContext, 55
 - _waitingPlayersReady, 55
 - ~AServer, 41
 - AServer, 40
 - FormatEntityInformation, 41
 - getClientById, 41
 - getClientInfoBarId, 41
 - getClientPlayerId, 42
 - GetClock, 42
 - GetComponentManager, 42
 - GetEntityFactory, 43
 - GetEntityManager, 43
 - GetPlayerClientId, 43
 - InitBoss, 44
 - InitiateEnemyMissile, 44
 - InitiatePlayer, 44
 - InitiatePlayerMissile, 44
 - InitiateWeaponForce, 45
 - InitInfoBar, 45
 - MessageAllClients, 45
 - MessageClient, 46
 - OnClientConnect, 46
 - OnClientDisconnect, 46
 - OnClientValidated, 47
 - OnMessage, 47
 - RemoveBossTail, 47
 - RemoveEntity, 47
 - RemoveInfoBar, 48
 - RemovePlayer, 48
 - SavePlayerScore, 48
 - SetClock, 49
 - Start, 49
 - Stop, 49
 - Update, 49
 - UpdateInfoBar, 50
 - UpdatePlayerPosition, 50
 - WaitForClientMessage, 51
- r_type::net::Client, 64
 - addEntity, 66
 - animateEntity, 66
 - displayEndOfGame, 66
 - initInfoBar, 66
 - MessageAll, 66
 - moveEntity, 66
 - PingServer, 66
 - removeEntity, 67
 - updateInfoBar, 67
- r_type::net::IClient< T >, 101
 - ~IClient, 102
 - Connect, 102
 - Disconnect, 102
 - IClient, 102
 - Incoming, 103
 - IsConnected, 103
 - Send, 103
- r_type::net::Server, 152
 - ~Server, 155
 - OnClientConnect, 155
 - OnClientDisconnect, 156
 - OnMessage, 156
 - Server, 155
- ratio
 - EntityInformation, 87
- rectangleShape
 - RectangleShapeComponent, 142
- RectangleShapeComponent, 141
 - rectangleShape, 142
 - RectangleShapeComponent, 141
- releaseFont
 - FontManager, 96
- releaseTexture
 - TextureManager, 166
- reloadFilter
 - scenes.cpp, 185
- removeAllComponents
 - ComponentManager, 71
- removeAllEntities
 - EntityManager, 90
- RemoveBossTail
 - r_type::net::AServer< T >, 47
- RemoveEntity
 - r_type::net::AServer< T >, 47
- removeEntity

- EntityManager, 90
- r_type::net::Client, 67
- removeEntityFromAllComponents
 - ComponentManager, 72
- removeEntityFromComponent
 - ComponentManager, 72
- RemoveInfoBar
 - r_type::net::AServer< T >, 48
- RemovePlayer
 - r_type::net::AServer< T >, 48
- render
 - IScenes, 117
 - RenderSystem, 143
 - Scenes, 149
- RenderSystem, 142
 - _componentManager, 143
 - _font, 143
 - _window, 144
 - render, 143
 - RenderSystem, 143
- RIGHT
 - AScenes, 28
 - input_component.hpp, 200
- run
 - Scenes, 150
- SavePlayerScore
 - r_type::net::AServer< T >, 48
- scale
 - SpriteDataComponent, 162
- Scene
 - AScenes, 29
- Scenes, 144
 - _networkClient, 151
 - _window, 151
 - ~Scenes, 147
 - difficultyChoices, 148
 - difficultyChoicesCustomization, 148
 - gameLoop, 148
 - getRenderWindow, 148
 - HandleMessage, 148
 - HandleTransitionLevelMessage, 148
 - inGameMenu, 148
 - mainMenu, 149
 - render, 149
 - run, 150
 - Scenes, 147
 - settingsMenu, 150
 - shouldQuit, 150
 - StopGameLoop, 150
 - TransitionLevel, 150
- scenes.cpp
 - createDaltonismChoiceButtons, 184
 - createKeyBindingButtons, 184
 - handleEvents, 184
 - reloadFilter, 185
 - waitForKey, 185
- scenes.hpp
 - keyToString, 179
- Score
 - game_text.hpp, 225
- score
 - ScoreComponent, 151
 - UIEntityInformation, 167
- ScoreComponent, 151
 - score, 151
- SCREEN_HEIGHT
 - macros.hpp, 229
- SCREEN_WIDTH
 - macros.hpp, 229
- Send
 - r_type::net::AClient< T >, 18
 - r_type::net::IClient< T >, 103
- Server
 - r_type::net::Server, 155
- SetClock
 - r_type::net::AServer< T >, 49
- setDaltonism
 - AScenes, 32
- setDisplayDaltonismChoice
 - AScenes, 33
- setDisplayGameModeChoice
 - AScenes, 33
- setDisplayKeyBindsChoice
 - AScenes, 33
- setGameMode
 - AScenes, 33
- SetGameParameters
 - r_type::Level< T >, 127
- setIp
 - AScenes, 34
- setPlayerId
 - r_type::net::AClient< T >, 18
- SetPlayerReady
 - AScenes, 34
- setPort
 - AScenes, 34
- setScene
 - AScenes, 35
- SetSystem
 - r_type::Level< T >, 128
- SETTINGS_MENU
 - AScenes, 29
- settingsMenu
 - IScenes, 117
 - Scenes, 150
- setWindowSize
 - r_type::net::AClient< T >, 18
- shader
 - ShaderComponent, 158
- ShaderComponent, 157
 - shader, 158
 - ShaderComponent, 157
- Ship1
 - sprite_path.hpp, 232
- Ship2
 - sprite_path.hpp, 232

- Ship3
 - sprite_path.hpp, 232
- Ship4
 - sprite_path.hpp, 232
- SHIP_DOWN
 - animation_system.hpp, 250
- SHIP_FLIP_DOWN
 - animation_system.hpp, 250
- SHIP_FLIP_UP
 - animation_system.hpp, 250
- SHIP_STRAIT
 - animation_system.hpp, 250
- SHIP_UP
 - animation_system.hpp, 250
- SHOOT
 - input_component.hpp, 200
- ShootComponent, 158
 - canShoot, 159
 - cooldownTime, 159
 - nextShootTime, 159
 - ShootComponent, 158
- ShooterEnemy
 - IEntityFactory, 106
- Shot
 - sound_path.hpp, 231
- shouldQuit
 - IScenes, 117
 - Scenes, 150
- signal_handler
 - main.cpp, 183
- sound_path.cpp
 - SoundFactory, 245
- sound_path.hpp
 - ActionType, 230
 - Background, 231
 - Boss, 231
 - BossDeath, 231
 - Explosion, 231
 - GameOver, 231
 - NONE, 231
 - PowerUp, 231
 - Shot, 231
 - SoundFactory, 231
 - Win, 231
- soundBuffers
 - AudioManager, 57
- SoundFactory
 - sound_path.cpp, 245
 - sound_path.hpp, 231
- SpawnEntity
 - r_type::Level< T >, 128
- sprite
 - SpriteComponent, 161
- sprite_data_component.hpp
 - operator<<, 210
- sprite_path.cpp
 - SpriteFactory, 246
- sprite_path.hpp
 - Background1, 233
 - Background2, 233
 - Background3, 233
 - Bar, 233
 - BlueLaserCrystal, 233
 - Boss, 233
 - BossBullet, 233
 - Enemy1, 232
 - Enemy2, 232
 - Enemy3, 232
 - ForceMissile, 233
 - ForceWeapon, 233
 - Missile, 233
 - operator<<, 233
 - Ship1, 232
 - Ship2, 232
 - Ship3, 232
 - Ship4, 232
 - SpriteFactory, 233
 - SpritePath, 232
 - Wall, 233
- SpriteComponent, 159
 - hitboxX, 160
 - hitboxY, 160
 - sprite, 161
 - SpriteComponent, 160
 - type, 161
- spriteData
 - EntityInformation, 87
 - UIEntityInformation, 167
- SpriteDataComponent, 161
 - scale, 162
 - spritePath, 162
 - type, 162
- SpriteFactory
 - sprite_path.cpp, 246
 - sprite_path.hpp, 233
- SpritePath
 - sprite_path.hpp, 232
- spritePath
 - SpriteDataComponent, 162
- SpriteType
 - AScenes, 29
- Start
 - r_type::net::AServer< T >, 49
- Stop
 - r_type::net::AServer< T >, 49
- stopBackgroundMusic
 - AudioSystem, 59
- StopGameLoop
 - Scenes, 150
- STRAIGHT
 - movement_component.hpp, 202
- SWEEPING
 - movement_component.hpp, 202
- TailComponent, 162
- tailSegmentIds
 - BossComponent, 64

- targetId
 - FrontComponent, 100
 - LinkForceComponent, 132
- text
 - TextComponent, 163
- TextComponent, 162
 - text, 163
 - TextComponent, 163
- textData
 - UIEntityInformation, 167
- TextDataComponent, 163
 - categoryIds, 164
 - categorySize, 164
 - categoryTexts, 164
 - charSize, 164
 - fontPath, 164
- TextureManager, 165
 - getTexture, 165
 - releaseTexture, 166
 - textures, 166
- textures
 - TextureManager, 166
- thrContext
 - r_type::net::AClient< T >, 19
- TRANSITION_LEVEL
 - AScenes, 29
- TransitionLevel
 - Scenes, 150
- TRITANOPIA
 - AScenes, 28
- type
 - SpriteComponent, 161
 - SpriteDataComponent, 162
- UI
 - AScenes, 30
- UIEntityInformation, 166
 - lives, 167
 - score, 167
 - spriteData, 167
 - textData, 167
 - uniqueID, 167
- uniqueID
 - EntityInformation, 87
 - UIEntityInformation, 167
- UP
 - AScenes, 28
 - input_component.hpp, 200
- Update
 - r_type::Level< T >, 129
 - r_type::net::AServer< T >, 49
- UpdateInfoBar
 - r_type::net::AServer< T >, 50
- updateInfoBar
 - r_type::net::Client, 67
- UpdatePlayerPosition
 - r_type::net::AServer< T >, 50
- updateSpritePositions
 - UpdateSystem, 169
- UpdateSystem, 168
 - _componentManager, 169
 - _entityManager, 169
 - _window, 170
 - updateSpritePositions, 169
 - UpdateSystem, 169
- updateText
 - UpdateTextComponent, 170
- UpdateTextComponent, 170
 - updateText, 170
 - UpdateTextComponent, 170
- VelocityComponent, 171
 - x, 171
 - y, 171
- vf2d, 171
 - x, 172
 - y, 172
- vPos
 - EntityInformation, 87
- w
 - HitboxComponent, 101
- WaitForClientMessage
 - r_type::net::AServer< T >, 51
- waitForKey
 - scenes.cpp, 185
- Wall
 - IEntityFactory, 106
 - sprite_path.hpp, 233
- WallComponent, 172
- WEAPON
 - AScenes, 30
- what
 - componentNotFound, 73
 - entityNotFound, 91
 - failedToCreateFile, 92
 - failedToLoadFont, 93
 - failedToLoadSound, 94
 - failedToLoadTexture, 94
 - failedToOpenFile, 95
 - playerIdNotFound, 139
- WIGGLE
 - movement_component.hpp, 202
- Win
 - sound_path.hpp, 231
- windowSize
 - r_type::net::AClient< T >, 19
- x
 - labelComponent, 119
 - PositionComponent, 141
 - VelocityComponent, 171
 - vf2d, 172
- y
 - labelComponent, 119
 - PositionComponent, 141
 - VelocityComponent, 171
 - vf2d, 172