

## Tux vs Bloatware

Generated by Doxygen 1.14.0



---

<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List . . . . .	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy . . . . .	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List . . . . .	5
<b>4 File Index</b>	<b>7</b>
4.1 File List . . . . .	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 globalVars Namespace Reference . . . . .	9
5.1.1 Variable Documentation . . . . .	9
5.1.1.1 inInterLevel . . . . .	9
5.1.1.2 points . . . . .	9
5.1.1.3 usernameGlobal . . . . .	9
<b>6 Class Documentation</b>	<b>11</b>
6.1 AbstractObstacle Class Reference . . . . .	11
6.1.1 Detailed Description . . . . .	12
6.1.2 Constructor & Destructor Documentation . . . . .	12
6.1.2.1 ~AbstractObstacle() . . . . .	12
6.1.3 Member Function Documentation . . . . .	12
6.1.3.1 checkCollisionWithPlayer() . . . . .	12
6.1.3.2 draw() . . . . .	12
6.1.3.3 getSpeed() . . . . .	13
6.1.3.4 setSpeed() . . . . .	13
6.1.3.5 update() . . . . .	13
6.2 Background Class Reference . . . . .	13
6.2.1 Detailed Description . . . . .	14
6.2.2 Constructor & Destructor Documentation . . . . .	14
6.2.2.1 Background() . . . . .	14
6.2.3 Member Function Documentation . . . . .	14
6.2.3.1 renderBackground() . . . . .	14
6.3 Ball Class Reference . . . . .	14
6.3.1 Detailed Description . . . . .	14
6.4 BallShot Class Reference . . . . .	14
6.4.1 Constructor & Destructor Documentation . . . . .	16
6.4.1.1 BallShot() . . . . .	16
6.4.2 Member Function Documentation . . . . .	16
6.4.2.1 draw() . . . . .	16
6.4.2.2 isIsActive() . . . . .	17

---

6.4.2.3 shotCollidedWithBoss()	17
6.4.2.4 shotCollidedWithPlayer()	17
6.4.2.5 update()	18
6.5 Bootstrap Class Reference	18
6.5.1 Detailed Description	18
6.5.2 Member Function Documentation	18
6.5.2.1 cleanup_allegro()	18
6.5.2.2 file_exists()	18
6.5.2.3 init_allegro_libs()	19
6.5.2.4 initialize_allegro()	19
6.5.2.5 register_allegro_events()	19
6.5.2.6 start_font()	19
6.5.2.7 start_sprite()	19
6.6 BrokenShip Class Reference	20
6.6.1 Detailed Description	21
6.6.2 Constructor & Destructor Documentation	21
6.6.2.1 BrokenShip() [1/2]	21
6.6.2.2 BrokenShip() [2/2]	21
6.6.3 Member Function Documentation	22
6.6.3.1 draw()	22
6.6.3.2 get_radius()	22
6.6.3.3 restart()	22
6.6.3.4 set_radius()	22
6.6.3.5 update()	22
6.7 Button Class Reference	22
6.7.1 Detailed Description	23
6.7.2 Constructor & Destructor Documentation	23
6.7.2.1 Button()	23
6.7.3 Member Function Documentation	23
6.7.3.1 drawButton()	23
6.7.3.2 gotClicked()	24
6.7.3.3 setText()	24
6.8 CircleObstacle Class Reference	24
6.8.1 Detailed Description	26
6.8.2 Constructor & Destructor Documentation	26
6.8.2.1 CircleObstacle()	26
6.8.3 Member Function Documentation	26
6.8.3.1 checkCollisionWithPlayer()	26
6.8.3.2 draw()	27
6.8.3.3 get_radius()	27
6.8.3.4 update()	27
6.9 Coordinates Struct Reference	27

---

6.9.1 Detailed Description . . . . .	28
6.9.2 Constructor & Destructor Documentation . . . . .	28
6.9.2.1 Coordinates() [1/2] . . . . .	28
6.9.2.2 Coordinates() [2/2] . . . . .	28
6.9.3 Member Data Documentation . . . . .	28
6.9.3.1 _heigth . . . . .	28
6.9.3.2 _width . . . . .	28
6.9.3.3 _x . . . . .	29
6.9.3.4 _y . . . . .	29
6.10 DatabaseUsers Class Reference . . . . .	29
6.10.1 Detailed Description . . . . .	29
6.10.2 Constructor & Destructor Documentation . . . . .	30
6.10.2.1 DatabaseUsers() . . . . .	30
6.10.2.2 ~DatabaseUsers() . . . . .	30
6.10.3 Member Function Documentation . . . . .	30
6.10.3.1 addValuesGameOverScreen() . . . . .	30
6.10.3.2 authenticateUser() . . . . .	30
6.10.3.3 deleteUser() . . . . .	31
6.10.3.4 getUserByUsername() . . . . .	31
6.10.3.5 listUsers() . . . . .	31
6.10.3.6 registerUser() . . . . .	32
6.10.3.7 updateGamesNumber() . . . . .	32
6.10.3.8 updateScore() . . . . .	32
6.11 dotenv Class Reference . . . . .	33
6.11.1 Detailed Description . . . . .	33
6.11.2 Constructor & Destructor Documentation . . . . .	34
6.11.2.1 dotenv() . . . . .	34
6.11.2.2 ~dotenv() . . . . .	34
6.11.3 Member Function Documentation . . . . .	34
6.11.3.1 getenv() . . . . .	34
6.11.3.2 init() [1/2] . . . . .	35
6.11.3.3 init() [2/2] . . . . .	35
6.11.4 Member Data Documentation . . . . .	35
6.11.4.1 OptionsNone . . . . .	35
6.11.4.2 Preserve . . . . .	36
6.12 exitGame Class Reference . . . . .	36
6.12.1 Detailed Description . . . . .	36
6.12.2 Constructor & Destructor Documentation . . . . .	36
6.12.2.1 ~exitGame() . . . . .	36
6.12.3 Member Function Documentation . . . . .	37
6.12.3.1 execute() . . . . .	37
6.13 FixedShip Class Reference . . . . .	37

6.13.1 Detailed Description . . . . .	38
6.13.2 Constructor & Destructor Documentation . . . . .	38
6.13.2.1 FixedShip() [1/2] . . . . .	38
6.13.2.2 FixedShip() [2/2] . . . . .	38
6.13.3 Member Function Documentation . . . . .	39
6.13.3.1 draw() . . . . .	39
6.13.3.2 get_radius() . . . . .	39
6.13.3.3 moveShip() . . . . .	39
6.13.3.4 set_radius() . . . . .	39
6.13.3.5 setCanTakeDamage() . . . . .	39
6.13.3.6 takeDamage() . . . . .	39
6.14 FlappyMovement Class Reference . . . . .	40
6.14.1 Detailed Description . . . . .	41
6.14.2 Member Function Documentation . . . . .	41
6.14.2.1 apply_gravity() . . . . .	41
6.14.2.2 getMoveForce() . . . . .	41
6.14.2.3 move_flappy() . . . . .	41
6.15 GameObject Class Reference . . . . .	42
6.15.1 Detailed Description . . . . .	42
6.15.2 Constructor & Destructor Documentation . . . . .	43
6.15.2.1 GameObject() [1/2] . . . . .	43
6.15.2.2 GameObject() [2/2] . . . . .	43
6.15.2.3 ~GameObject() . . . . .	43
6.15.3 Member Function Documentation . . . . .	43
6.15.3.1 get_position() . . . . .	43
6.15.3.2 set_bitmap() . . . . .	43
6.15.3.3 set_position() . . . . .	43
6.15.4 Member Data Documentation . . . . .	44
6.15.4.1 _position . . . . .	44
6.15.4.2 objectSprite . . . . .	44
6.16 gameOverOption Class Reference . . . . .	44
6.16.1 Detailed Description . . . . .	44
6.16.2 Constructor & Destructor Documentation . . . . .	44
6.16.2.1 ~gameOverOption() . . . . .	44
6.16.3 Member Function Documentation . . . . .	45
6.16.3.1 execute() . . . . .	45
6.17 gameOverScreen Class Reference . . . . .	45
6.17.1 Detailed Description . . . . .	45
6.17.2 Constructor & Destructor Documentation . . . . .	46
6.17.2.1 gameOverScreen() . . . . .	46
6.17.3 Member Function Documentation . . . . .	46
6.17.3.1 draw() . . . . .	46

---

6.17.3.2 run()	46
6.17.3.3 setbestScore()	47
6.17.3.4 setCurrentScore()	47
6.17.3.5 setHighScore()	47
6.17.3.6 setnumGames()	47
6.18 Interface Class Reference	47
6.18.1 Detailed Description	48
6.18.2 Constructor & Destructor Documentation	48
6.18.2.1 Interface()	48
6.18.3 Member Function Documentation	48
6.18.3.1 drawOffGameInterface()	48
6.18.4 Member Data Documentation	48
6.18.4.1 exitGameButton	48
6.18.4.2 playButton	48
6.18.4.3 returnToMenuButton	48
6.18.4.4 stopSongButton	49
6.19 Level Class Reference	49
6.19.1 Detailed Description	49
6.19.2 Friends And Related Symbol Documentation	49
6.19.2.1 interLevelHandling	49
6.19.3 Member Data Documentation	50
6.19.3.1 _bg	50
6.19.3.2 _event	50
6.19.3.3 _player	50
6.20 LevelOne Class Reference	50
6.20.1 Detailed Description	51
6.20.2 Member Function Documentation	51
6.20.2.1 cleanLevel()	51
6.20.2.2 handleKeyPressEvents()	51
6.20.2.3 handleKeyReleaseEvents()	51
6.20.2.4 handleTimerEvents()	52
6.20.2.5 mainLoop()	53
6.20.2.6 setLevelOne()	53
6.21 LevelThree Class Reference	53
6.21.1 Detailed Description	54
6.21.2 Member Function Documentation	54
6.21.2.1 cleanLevel()	54
6.21.2.2 handleKeyPressEvents() [1/2]	54
6.21.2.3 handleKeyPressEvents() [2/2]	55
6.21.2.4 handleKeyReleaseEvents() [1/2]	55
6.21.2.5 handleKeyReleaseEvents() [2/2]	55
6.21.2.6 handleTimerEvents()	55

6.21.2.7 mainLoop()	55
6.21.2.8 setLevelThree()	55
6.21.2.9 updatePlayerPosition()	55
6.22 LevelTwo Class Reference	56
6.22.1 Detailed Description	56
6.22.2 Member Function Documentation	56
6.22.2.1 cleanLevel()	56
6.22.2.2 handleKeyPressEvents()	57
6.22.2.3 handleKeyReleaseEvents()	57
6.22.2.4 handleTimerEvents()	57
6.22.2.5 mainLoop()	57
6.22.2.6 setLevelTwo()	57
6.23 Line Class Reference	57
6.23.1 Detailed Description	58
6.24 LineShot Class Reference	58
6.24.1 Constructor & Destructor Documentation	60
6.24.1.1 LineShot()	60
6.24.2 Member Function Documentation	61
6.24.2.1 draw()	61
6.24.2.2 isIsActive()	61
6.24.2.3 shotCollidedWithBoss()	61
6.24.2.4 shotCollidedWithPlayer()	61
6.24.2.5 update()	62
6.25 Menu Class Reference	62
6.25.1 Detailed Description	62
6.25.2 Member Data Documentation	63
6.25.2.1 event	63
6.25.2.2 font	63
6.25.2.3 interface	63
6.26 Music Class Reference	63
6.26.1 Detailed Description	64
6.26.2 Constructor & Destructor Documentation	64
6.26.2.1 Music()	64
6.26.2.2 ~Music()	65
6.26.3 Member Function Documentation	65
6.26.3.1 muteMusic()	65
6.26.3.2 pause()	65
6.26.3.3 play()	65
6.26.3.4 unMuteMusic()	65
6.26.3.5 update_fade_in_fade_out()	65
6.27 ObstaclesList Class Reference	66
6.27.1 Detailed Description	66

---

6.27.2 Constructor & Destructor Documentation . . . . .	66
6.27.2.1 ~ObstaclesList() . . . . .	66
6.27.3 Member Function Documentation . . . . .	67
6.27.3.1 clear() . . . . .	67
6.27.3.2 drawAll() . . . . .	67
6.27.3.3 getList() . . . . .	67
6.27.3.4 setCircleObstaclesList() . . . . .	67
6.27.3.5 setPolygonsObstaclesList() . . . . .	67
6.27.3.6 updateAll() . . . . .	68
6.28 Pipe Class Reference . . . . .	68
6.28.1 Detailed Description . . . . .	69
6.28.2 Constructor & Destructor Documentation . . . . .	70
6.28.2.1 Pipe() . . . . .	70
6.28.3 Member Function Documentation . . . . .	70
6.28.3.1 checkCollisionWithPlayer() . . . . .	70
6.28.3.2 draw() . . . . .	70
6.28.3.3 update() . . . . .	71
6.29 PipeList Class Reference . . . . .	71
6.29.1 Constructor & Destructor Documentation . . . . .	71
6.29.1.1 PipeList() . . . . .	71
6.29.1.2 ~PipeList() . . . . .	71
6.29.2 Member Function Documentation . . . . .	72
6.29.2.1 clear() . . . . .	72
6.29.2.2 generatePipes() . . . . .	72
6.29.2.3 getList() . . . . .	72
6.30 playAgain Class Reference . . . . .	73
6.30.1 Detailed Description . . . . .	73
6.30.2 Constructor & Destructor Documentation . . . . .	73
6.30.2.1 ~playAgain() . . . . .	73
6.30.3 Member Function Documentation . . . . .	73
6.30.3.1 execute() . . . . .	73
6.31 PolygonObstacle Class Reference . . . . .	74
6.31.1 Detailed Description . . . . .	75
6.31.2 Constructor & Destructor Documentation . . . . .	75
6.31.2.1 PolygonObstacle() . . . . .	75
6.31.3 Member Function Documentation . . . . .	76
6.31.3.1 checkCollisionWithPlayer() . . . . .	76
6.31.3.2 draw() . . . . .	77
6.31.3.3 getVertices() . . . . .	77
6.31.3.4 update() . . . . .	77
6.32 RegisterInterface Class Reference . . . . .	78
6.32.1 Detailed Description . . . . .	78

6.32.2 Constructor & Destructor Documentation . . . . .	78
6.32.2.1 RegisterInterface() . . . . .	78
6.32.2.2 ~RegisterInterface() . . . . .	79
6.32.3 Member Function Documentation . . . . .	79
6.32.3.1 draw() . . . . .	79
6.32.3.2 getName() . . . . .	79
6.32.3.3 getPassword() . . . . .	79
6.32.3.4 getUsername() . . . . .	80
6.32.3.5 handleKeyInput() . . . . .	80
6.32.3.6 handleMouseClick() . . . . .	81
6.32.3.7 mainLoop() . . . . .	81
6.32.3.8 resetFields() . . . . .	81
6.33 Represents Class Reference . . . . .	82
6.33.1 Detailed Description . . . . .	82
6.34 returnMenu Class Reference . . . . .	82
6.34.1 Detailed Description . . . . .	82
6.34.2 Constructor & Destructor Documentation . . . . .	82
6.34.2.1 ~returnMenu() . . . . .	82
6.34.3 Member Function Documentation . . . . .	83
6.34.3.1 execute() . . . . .	83
6.35 Shot Class Reference . . . . .	83
6.35.1 Detailed Description . . . . .	84
6.35.2 Constructor & Destructor Documentation . . . . .	84
6.35.2.1 Shot() . . . . .	84
6.35.2.2 ~Shot() . . . . .	85
6.35.3 Member Function Documentation . . . . .	85
6.35.3.1 cleanShots() . . . . .	85
6.35.3.2 draw() . . . . .	85
6.35.3.3 drawShots() . . . . .	85
6.35.3.4 isIsActive() . . . . .	85
6.35.3.5 shotCollidedWithBoss() . . . . .	85
6.35.3.6 shotCollidedWithPlayer() . . . . .	86
6.35.3.7 update() . . . . .	86
6.35.3.8 updateShots() . . . . .	86
6.35.4 Member Data Documentation . . . . .	86
6.35.4.1 _direction . . . . .	86
6.35.4.2 _shotColor . . . . .	86
6.35.4.3 ShotsList . . . . .	86
6.36 Sound Class Reference . . . . .	87
6.36.1 Detailed Description . . . . .	87
6.36.2 Constructor & Destructor Documentation . . . . .	87
6.36.2.1 Sound() . . . . .	87

---

6.36.2.2 ~Sound()	88
6.36.3 Member Function Documentation	88
6.36.3.1 muteSounds()	88
6.36.3.2 play()	88
6.36.3.3 unmuteSounds()	88
6.36.4 Member Data Documentation	88
6.36.4.1 isSoundMuted	88
6.36.4.2 sound_sample	88
6.36.4.3 volumeMester	89
6.37 StartMenu Class Reference	89
6.37.1 Detailed Description	89
6.37.2 Member Function Documentation	89
6.37.2.1 mainLoopMenu()	89
6.38 User Struct Reference	90
6.38.1 Detailed Description	90
6.38.2 Member Data Documentation	90
6.38.2.1 games	90
6.38.2.2 id	90
6.38.2.3 name	90
6.38.2.4 score	90
6.38.2.5 username	90
6.39 Vector Class Reference	91
6.39.1 Detailed Description	91
6.39.2 Constructor & Destructor Documentation	91
6.39.2.1 Vector() [1/3]	91
6.39.2.2 Vector() [2/3]	92
6.39.2.3 Vector() [3/3]	92
6.39.3 Member Function Documentation	92
6.39.3.1 distance()	92
6.39.3.2 dot()	92
6.39.3.3 operator*()	93
6.39.3.4 operator+()	93
6.39.3.5 operator-()	93
6.39.3.6 shortestDistancePointToSegment()	93
6.39.4 Member Data Documentation	93
6.39.4.1 _x	93
6.39.4.2 _y	94
6.40 victoryInterface Class Reference	94
6.40.1 Detailed Description	94
6.40.2 Constructor & Destructor Documentation	94
6.40.2.1 victoryInterface()	94
6.40.3 Member Function Documentation	94

---

6.40.3.1 drawVictoryScreen() . . . . .	94
6.41 Windows Class Reference . . . . .	95
6.41.1 Detailed Description . . . . .	95
6.42 WindowsBoss Class Reference . . . . .	95
6.42.1 Constructor & Destructor Documentation . . . . .	96
6.42.1.1 WindowsBoss() . . . . .	96
6.42.2 Member Function Documentation . . . . .	96
6.42.2.1 draw() . . . . .	96
6.42.2.2 getHalfSide() . . . . .	97
6.42.2.3 isDead() . . . . .	97
6.42.2.4 takeDamage() . . . . .	97
6.42.2.5 update() . . . . .	97
<b>7 File Documentation</b> . . . . .	<b>99</b>
7.1 include/abstract_obstacle.hpp File Reference . . . . .	99
7.2 abstract_obstacle.hpp . . . . .	99
7.3 include/bootstrap.hpp File Reference . . . . .	100
7.3.1 Variable Documentation . . . . .	101
7.3.1.1 BACKGROUND_COLOR . . . . .	101
7.3.1.2 backgroundImage . . . . .	101
7.3.1.3 ballShotSprite . . . . .	101
7.3.1.4 BUTTON_H . . . . .	101
7.3.1.5 BUTTON_W . . . . .	101
7.3.1.6 death_sound . . . . .	101
7.3.1.7 defeat_music . . . . .	101
7.3.1.8 display . . . . .	101
7.3.1.9 event_queue . . . . .	102
7.3.1.10 FPS . . . . .	102
7.3.1.11 gameFont . . . . .	102
7.3.1.12 gameOverBackground . . . . .	102
7.3.1.13 gunshot_sound1 . . . . .	102
7.3.1.14 gunshot_sound2 . . . . .	102
7.3.1.15 gunshot_sound3 . . . . .	102
7.3.1.16 gunshot_sound4 . . . . .	102
7.3.1.17 level_one_music . . . . .	102
7.3.1.18 level_three_music . . . . .	102
7.3.1.19 level_two_music . . . . .	103
7.3.1.20 levelFont . . . . .	103
7.3.1.21 menu_music . . . . .	103
7.3.1.22 OBSTACLES_LIST_NUM . . . . .	103
7.3.1.23 pause_game_music . . . . .	103
7.3.1.24 pendrive . . . . .	103

---

7.3.1.25 pinguimBandido . . . . .	103
7.3.1.26 SCALE_ASTEROID . . . . .	103
7.3.1.27 SCALE_PIPES . . . . .	103
7.3.1.28 SCREEN_H . . . . .	103
7.3.1.29 SCREEN_W . . . . .	104
7.3.1.30 TAM_VECTOR_VELOCITY . . . . .	104
7.3.1.31 timer . . . . .	104
7.3.1.32 velocity . . . . .	104
7.3.1.33 victory_music . . . . .	104
7.4 bootstrap.hpp . . . . .	104
7.5 include/boss_states.hpp File Reference . . . . .	105
7.5.1 Enumeration Type Documentation . . . . .	105
7.5.1.1 AttackType . . . . .	105
7.5.1.2 BossStates . . . . .	106
7.6 boss_states.hpp . . . . .	106
7.7 include/circle_obstacle.hpp File Reference . . . . .	106
7.8 circle_obstacle.hpp . . . . .	107
7.9 include/collision.hpp File Reference . . . . .	107
7.9.1 Function Documentation . . . . .	108
7.9.1.1 checkCircleCollision() . . . . .	108
7.9.1.2 checkCollisionWithPlayer() . . . . .	108
7.9.1.3 circleCircleCollision() . . . . .	108
7.9.1.4 circleSquareCollision() . . . . .	108
7.9.1.5 distanceBetweenPoints() . . . . .	109
7.9.1.6 isCollidingEdge() . . . . .	109
7.9.1.7 newPositionAfterCollisionEdge() . . . . .	109
7.10 collision.hpp . . . . .	110
7.11 include/database_users.hpp File Reference . . . . .	110
7.12 database_users.hpp . . . . .	110
7.13 include/dotenv.h File Reference . . . . .	111
7.14 dotenv.h . . . . .	111
7.15 include/game_object.hpp File Reference . . . . .	115
7.16 game_object.hpp . . . . .	115
7.17 include/game_over.hpp File Reference . . . . .	116
7.18 game_over.hpp . . . . .	117
7.19 include/interface.hpp File Reference . . . . .	117
7.20 interface.hpp . . . . .	118
7.21 include/levels.hpp File Reference . . . . .	119
7.21.1 Macro Definition Documentation . . . . .	120
7.21.1.1 LEVEL_DURATION . . . . .	120
7.21.2 Function Documentation . . . . .	120
7.21.2.1 interLevelHandling() . . . . .	120

7.22 levels.hpp . . . . .	120
7.23 include/menu.hpp File Reference . . . . .	121
7.24 menu.hpp . . . . .	122
7.25 include/movement.hpp File Reference . . . . .	122
7.26 movement.hpp . . . . .	122
7.27 include/music.hpp File Reference . . . . .	123
7.28 music.hpp . . . . .	123
7.29 include/obstacles_list.hpp File Reference . . . . .	124
7.30 obstacles_list.hpp . . . . .	124
7.31 include/pipe.hpp File Reference . . . . .	124
7.32 pipe.hpp . . . . .	125
7.33 include/polygon_obstacle.hpp File Reference . . . . .	125
7.34 polygon_obstacle.hpp . . . . .	125
7.35 include/register_interface.hpp File Reference . . . . .	126
7.36 register_interface.hpp . . . . .	126
7.37 include/shapes_repository.hpp File Reference . . . . .	127
7.37.1 Variable Documentation . . . . .	127
7.37.1.1 shape_repository . . . . .	127
7.38 shapes_repository.hpp . . . . .	128
7.39 include/shots.hpp File Reference . . . . .	128
7.40 shots.hpp . . . . .	129
7.41 include/sound.hpp File Reference . . . . .	130
7.42 sound.hpp . . . . .	130
7.43 include/windows_boss.hpp File Reference . . . . .	130
7.44 windows_boss.hpp . . . . .	131
7.45 src/abstract_obstacle.cpp File Reference . . . . .	131
7.46 src/bootstrap.cpp File Reference . . . . .	131
7.46.1 Variable Documentation . . . . .	132
7.46.1.1 BACKGROUND_COLOR . . . . .	132
7.46.1.2 backgroundImage . . . . .	132
7.46.1.3 ballShotSprite . . . . .	132
7.46.1.4 death_sound . . . . .	133
7.46.1.5 defeat_music . . . . .	133
7.46.1.6 display . . . . .	133
7.46.1.7 event_queue . . . . .	133
7.46.1.8 gameFont . . . . .	133
7.46.1.9 gameOverBackground . . . . .	133
7.46.1.10 gunshot_sound1 . . . . .	133
7.46.1.11 gunshot_sound2 . . . . .	133
7.46.1.12 gunshot_sound3 . . . . .	133
7.46.1.13 gunshot_sound4 . . . . .	133
7.46.1.14 level_one_music . . . . .	134

---

7.46.1.15 level_three_music . . . . .	134
7.46.1.16 level_two_music . . . . .	134
7.46.1.17 levelFont . . . . .	134
7.46.1.18 menu_music . . . . .	134
7.46.1.19 pause_game_music . . . . .	134
7.46.1.20 pendrive . . . . .	134
7.46.1.21 pinguimBandido . . . . .	134
7.46.1.22 timer . . . . .	134
7.46.1.23 victory_music . . . . .	134
7.47 src/circle_obstacle.cpp File Reference . . . . .	135
7.48 src/collision.cpp File Reference . . . . .	135
7.48.1 Function Documentation . . . . .	135
7.48.1.1 circleCircleCollision() . . . . .	135
7.48.1.2 circleSquareCollision() . . . . .	136
7.48.1.3 distanceBetweenPoints() . . . . .	136
7.48.1.4 isCollidingEdge() . . . . .	136
7.48.1.5 newPositionAfterCollisionEdge() . . . . .	137
7.49 src/database_users.cpp File Reference . . . . .	137
7.50 src/game_object.cpp File Reference . . . . .	137
7.50.1 Variable Documentation . . . . .	137
7.50.1.1 PLAYER_RADIUS . . . . .	137
7.51 src/gameover.cpp File Reference . . . . .	138
7.51.1 Variable Documentation . . . . .	138
7.51.1.1 event_queue . . . . .	138
7.51.1.2 gameOverBackground . . . . .	138
7.51.1.3 timer . . . . .	138
7.52 src/interface.cpp File Reference . . . . .	138
7.53 src/levels.cpp File Reference . . . . .	139
7.53.1 Function Documentation . . . . .	139
7.53.1.1 interLevelHandling() . . . . .	139
7.54 src/main.cpp File Reference . . . . .	139
7.54.1 Function Documentation . . . . .	140
7.54.1.1 main() . . . . .	140
7.54.2 Variable Documentation . . . . .	140
7.54.2.1 event_queue . . . . .	140
7.54.2.2 gameFont . . . . .	140
7.54.2.3 timer . . . . .	140
7.55 src/menu.cpp File Reference . . . . .	140
7.56 src/movement.cpp File Reference . . . . .	141
7.57 src/music.cpp File Reference . . . . .	141
7.58 src/obstacles_list.cpp File Reference . . . . .	141
7.59 src/pipe.cpp File Reference . . . . .	141

7.60 src/polygon_obstacle.cpp File Reference . . . . .	141
7.61 src/register_interface.cpp File Reference . . . . .	141
7.62 src/shots.cpp File Reference . . . . .	142
7.63 src/sound.cpp File Reference . . . . .	142
7.64 src/windows_boss.cpp File Reference . . . . .	142
7.64.1 Variable Documentation . . . . .	142
7.64.1.1 cont . . . . .	142
<b>Index</b>	<b>143</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">globalVars</a> . . . . .	9
--------------------------------------	---



# Chapter 2

## Hierarchical Index

### 2.1 Class Hierarchy

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

Background . . . . .	13
Ball . . . . .	14
Bootstrap . . . . .	18
Button . . . . .	22
Coordinates . . . . .	27
DatabaseUsers . . . . .	29
dotenv . . . . .	33
GameObject . . . . .	42
AbstractObstacle . . . . .	11
CircleObstacle . . . . .	24
Pipe . . . . .	68
PolygonObstacle . . . . .	74
FixedShip . . . . .	37
FlappyMovement . . . . .	40
BrokenShip . . . . .	20
Shot . . . . .	83
BallShot . . . . .	14
LineShot . . . . .	58
WindowsBoss . . . . .	95
gameOverOption . . . . .	44
exitGame . . . . .	36
playAgain . . . . .	73
returnMenu . . . . .	82
gameOverScreen . . . . .	45
Interface . . . . .	47
Level . . . . .	49
LevelOne . . . . .	50
LevelThree . . . . .	53
LevelTwo . . . . .	56
Line . . . . .	57
Menu . . . . .	62
StartMenu . . . . .	89
ObstaclesList . . . . .	66
PipeList . . . . .	71

RegisterInterface . . . . .	78
Represents . . . . .	82
Sound . . . . .	87
Music . . . . .	63
User . . . . .	90
Vector . . . . .	91
victoryInterface . . . . .	94
Windows . . . . .	95

# Chapter 3

## Class Index

### 3.1 Class List

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

<a href="#">AbstractObstacle</a>	Classe base abstrata para todos os tipos de obstáculos no jogo	11
<a href="#">Background</a>	Moving background logic	13
<a href="#">Ball</a>	Ball-shaped shot	14
<a href="#">BallShot</a>		14
<a href="#">Bootstrap</a>	Static class that encapsulates initialization and cleanup of allegro components	18
<a href="#">BrokenShip</a>	Player of phases 1 and 2, implements Flappy Movement	20
<a href="#">Button</a>	Simple button logic with draw, click checking and setText functions	22
<a href="#">CircleObstacle</a>	Representa um obstáculo com uma forma de colisão perfeitamente circular	24
<a href="#">Coordinates</a>	Extremely basic coordinates struct. Takes x and y coordinates, width and height as paramenters. Mostly used to keep better logic in buttons	27
<a href="#">DatabaseUsers</a>	Gerencia todas as operações de banco de dados relacionadas a usuários	29
<a href="#">dotenv</a>		33
<a href="#">exitGame</a>	Ação para sair do jogo	36
<a href="#">FixedShip</a>	Player of phase 3, implements 2D movement	37
<a href="#">FlappyMovement</a>	Base class that implements Flappy-Bird-like movement	40
<a href="#">GameObject</a>	Base class to objects of the game	42
<a href="#">gameOverOption</a>	Classe base abstrata para as opções de ação na tela de Game Over	44
<a href="#">gameOverScreen</a>	Gerencia a exibição da mensagem de Game Over, scores, botões de ação e a interação do jogador. _currentScore: pontuação na partida _highScore: maior pontuação (recorde pessoal) _bestScore: Melhor pontuação (recorde geral) ALLEGRO_FONT* _font: fonte dos textos da tela _playAgainButton: botão jogar novamente _returnToMenuButton: botão para retornar para a tela inicial _exitGameButton: botão para sair do jogo	45

<a href="#">Interface</a>	Base game start interface using button class . . . . .	47
<a href="#">Level</a>	Base class for the game levels, encapsulates the bare minimum logic to maintain a level . . . . .	49
<a href="#">LevelOne</a>	First phase of the game, basic vertically-oriented flappy bird with satellites as pipes . . . . .	50
<a href="#">LevelThree</a>	Third phase of the game, a free-movement, shooter boss fight against <a href="#">Windows</a> (The biggest piece of bloatware in earth) . . . . .	53
<a href="#">LevelTwo</a>	Second phase of the game, vertically-oriented flappy bird with moving obstacles (asteroids) . . . . .	56
<a href="#">Line</a>	Line-shaped shot, like a laser . . . . .	57
<a href="#">LineShot</a>		58
<a href="#">Menu</a>	Base menu class . . . . .	62
<a href="#">Music</a>	<a href="#">Sound</a> in a more complex way, with pause and playback methods with fade-in and fade-out systems . . . . .	63
<a href="#">ObstaclesList</a>	Gerencia uma coleção polimórfica de múltiplos obstáculos . . . . .	66
<a href="#">Pipe</a>	Representa um par de obstáculos (canos) que se movem em conjunto . . . . .	68
<a href="#">PipeList</a>		71
<a href="#">playAgain</a>	Ação para reiniciar o jogo . . . . .	73
<a href="#">PolygonObstacle</a>	Representa um obstáculo com uma forma de colisão poligonal customizável . . . . .	74
<a href="#">RegisterInterface</a>	Gerencia a interface gráfica e a lógica para registro e login de usuários . . . . .	78
<a href="#">Represents</a>		82
<a href="#">returnMenu</a>	Ação para retornar ao menu principal do jogo . . . . .	82
<a href="#">Shot</a>	<a href="#">Shot</a> in a more abstract way, abstract class . . . . .	83
<a href="#">Sound</a>	Class that represents a simple sound, with a musical object, ALLEGRO_SAMPLE, and a play method . . . . .	87
<a href="#">StartMenu</a>	Main menu that inherits from the base menu class . . . . .	89
<a href="#">User</a>	Estrutura simples para armazenar os dados de um usuário . . . . .	90
<a href="#">Vector</a>	Implements 2D vectors that represent cartesian coordinates . . . . .	91
<a href="#">victoryInterface</a>	<a href="#">Interface</a> to the "victory screen" after defeating the final boss ( <a href="#">windows</a> ) . . . . .	94
<a href="#">Windows</a>	Game's boss, <a href="#">Windows</a> . . . . .	95
<a href="#">WindowsBoss</a>		95

# Chapter 4

## File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

include/abstract_obstacle.hpp	99
include/bootstrap.hpp	100
include/boss_states.hpp	105
include/circle_obstacle.hpp	106
include/collision.hpp	107
include/database_users.hpp	110
include/dotenv.h	111
include/game_object.hpp	115
include/game_over.hpp	116
include/interface.hpp	117
include/levels.hpp	119
include/menu.hpp	121
include/movement.hpp	122
include/music.hpp	123
include/obstacles_list.hpp	124
include/pipe.hpp	124
include/polygon_obstacle.hpp	125
include/register_interface.hpp	126
include/shapes_repository.hpp	127
include/shots.hpp	128
include/sound.hpp	130
include/windows_boss.hpp	130
src/abstract_obstacle.cpp	131
src/bootstrap.cpp	131
src/circle_obstacle.cpp	135
src/collision.cpp	135
src/database_users.cpp	137
src/game_object.cpp	137
src/gameover.cpp	138
src/interface.cpp	138
src/levels.cpp	139
src/main.cpp	139
src/menu.cpp	140
src/movement.cpp	141
src/music.cpp	141

src/ <a href="#">obstacles_list.cpp</a>	141
src/ <a href="#">pipe.cpp</a>	141
src/ <a href="#">polygon_obstacle.cpp</a>	141
src/ <a href="#">register_interface.cpp</a>	141
src/ <a href="#">shots.cpp</a>	142
src/ <a href="#">sound.cpp</a>	142
src/ <a href="#">windows_boss.cpp</a>	142

## Chapter 5

# Namespace Documentation

## 5.1 globalVars Namespace Reference

### Variables

- bool `inInterLevel` = false
- int `points` = 0
- std::string `usernameGlobal` = ""

### 5.1.1 Variable Documentation

#### 5.1.1.1 `inInterLevel`

```
bool globalVars::inInterLevel = false
```

#### 5.1.1.2 `points`

```
int globalVars::points = 0
```

#### 5.1.1.3 `usernameGlobal`

```
std::string globalVars::usernameGlobal = ""
```



# Chapter 6

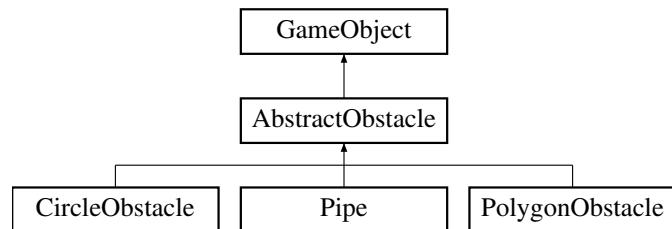
## Class Documentation

### 6.1 AbstractObstacle Class Reference

Classe base abstrata para todos os tipos de obstáculos no jogo.

```
#include <abstract_obstacle.hpp>
```

Inheritance diagram for AbstractObstacle:



#### Public Member Functions

- virtual void `draw ()=0`
- virtual void `update ()=0`
- virtual bool `checkCollisionWithPlayer (BrokenShip &player)=0`
- virtual `~AbstractObstacle ()=default`

#### Public Member Functions inherited from [GameObject](#)

- [`GameObject \(\)`](#)  
*Create a new default [GameObject](#).*
- [`GameObject \(Vector position\)`](#)  
*Create a new [GameObject](#) on given position.*
- [`Vector get\_position \(\)`](#)  
*Get the position of a [GameObject](#).*
- [`void set\_position \(const Vector &position\)`](#)  
*Sets the position of a [GameObject](#).*
- virtual `~GameObject ()=0`  
*Game Object empty destructor.*
- [`void set\_bitmap \(const char \*path\)`](#)  
*Sets the sprite of the [GameObject](#).*

## Protected Member Functions

- void [setSpeed \(Vector speed\)](#)  
*Define a velocidade do obstáculo.*
- [Vector getSpeed \(\)](#)  
*Obtém a velocidade atual do obstáculo.*

## Additional Inherited Members

### Public Attributes inherited from [GameObject](#)

- ALLEGRO\_BITMAP \* [objectSprite](#) = NULL

### Protected Attributes inherited from [GameObject](#)

- [Vector \\_position](#)

## 6.1.1 Detailed Description

Classe base abstrata para todos os tipos de obstáculos no jogo.

Esta classe define o contrato que todas as classes de obstáculo concretas devem seguir. Ela herda de [GameObject](#) e adiciona funcionalidades específicas de obstáculos, como velocidade e métodos virtuais puros para desenho, atualização e detecção de colisão. Não pode ser instanciada diretamente.

## 6.1.2 Constructor & Destructor Documentation

### 6.1.2.1 [~AbstractObstacle\(\)](#)

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

## 6.1.3 Member Function Documentation

### 6.1.3.1 [checkCollisionWithPlayer\(\)](#)

```
virtual bool AbstractObstacle::checkCollisionWithPlayer ( BrokenShip & player) [pure virtual]
```

Implemented in [CircleObstacle](#), [Pipe](#), and [PolygonObstacle](#).

### 6.1.3.2 [draw\(\)](#)

```
virtual void AbstractObstacle::draw () [pure virtual]
```

Implemented in [CircleObstacle](#), [Pipe](#), and [PolygonObstacle](#).

### 6.1.3.3 `getSpeed()`

```
Vector AbstractObstacle::getSpeed () [protected]
```

Obtém a velocidade atual do obstáculo.

#### Returns

O vetor de velocidade atual (`_speed`) do obstáculo.

### 6.1.3.4 `setSpeed()`

```
void AbstractObstacle::setSpeed (
    Vector speed) [protected]
```

Define a velocidade do obstáculo.

Este método atualiza o vetor de velocidade do objeto

#### Parameters

<code>speed</code>	O novo vetor de velocidade a ser atribuído ao obstáculo.
--------------------	--

### 6.1.3.5 `update()`

```
virtual void AbstractObstacle::update () [pure virtual]
```

Implemented in [CircleObstacle](#), [Pipe](#), and [PolygonObstacle](#).

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

- [include/abstract\\_obstacle.hpp](#)
- [src/abstract\\_obstacle.cpp](#)

## 6.2 Background Class Reference

moving background logic

```
#include <levels.hpp>
```

### Public Member Functions

- [Background \(\)](#)
- [void renderBackground \(\)](#)  
*renders the moving background*

## 6.2.1 Detailed Description

moving background logic

## 6.2.2 Constructor & Destructor Documentation

### 6.2.2.1 Background()

```
Background::Background ()
```

## 6.2.3 Member Function Documentation

### 6.2.3.1 renderBackground()

```
void Background::renderBackground ()
```

renders the moving background

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

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

## 6.3 Ball Class Reference

represents a ball-shaped shot.

### 6.3.1 Detailed Description

represents a ball-shaped shot.

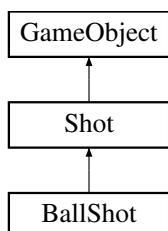
The documentation for this class was generated from the following file:

- [src/shots.cpp](#)

## 6.4 BallShot Class Reference

```
#include <shots.hpp>
```

Inheritance diagram for BallShot:



## Public Member Functions

- **BallShot** (`Vector` initialPosi, `Vector` direction, float radius, float speed=40)  
*Constructs a ball-shaped shot and inserts it into the shot list.*
- void **draw** () override  
*Draw a ball.*
- void **update** () override  
*Add to the position of the ball (shot), the direction vector multiplied by the velocity. (normally the direction vectors are unitary)*
- bool **isIsActive** () override  
*Checks if the ball (shot) positions are within the game region.*
- bool **shotCollidedWithBoss** (`WindowsBoss` &boss) override  
*Checks if the ball (shot) collided with the boss. (Circle-square collision.)*
- bool **shotCollidedWithPlayer** (`FixedShip` &player) override  
*Check if the ball (shot) collided with the player. (Circle-to-circle collision.)*

## Public Member Functions inherited from `Shot`

- **Shot** (`Vector` position, `Vector` direction, ALLEGRO\_COLOR shotColor)  
*Build a standard shot.*
- virtual ~**Shot** ()=default

## Public Member Functions inherited from `GameObject`

- **GameObject** ()  
*Create a new default `GameObject`.*
- **GameObject** (`Vector` position)  
*Create a new `GameObject` on given position.*
- **Vector get\_position** ()  
*Get the position of a `GameObject`.*
- void **set\_position** (const `Vector` &position)  
*Sets the position of a `GameObject`.*
- virtual ~**GameObject** ()=0  
*Game Object empty destructor.*
- void **set\_bitmap** (const char \*path)  
*Sets the sprite of the `GameObject`.*

## Additional Inherited Members

### Static Public Member Functions inherited from `Shot`

- static void **updateShots** (`FixedShip` \*player, `WindowsBoss` &boss, bool &playing)  
*For all shots, from the shot list, they are updated and then it is checked if the shot is active or if it collided with something (boss or the player)*
- static void **drawShots** ()  
*Calls the `draw()` function of all active shots.*
- static void **cleanShots** ()  
*Destroys all shots.*

## Public Attributes inherited from [GameObject](#)

- ALLEGRO\_BITMAP \* `objectSprite` = NULL

## Protected Attributes inherited from [Shot](#)

- `Vector _direction`
- ALLEGRO\_COLOR `_shotColor`

## Protected Attributes inherited from [GameObject](#)

- `Vector _position`

## Static Protected Attributes inherited from [Shot](#)

- static std::list< [Shot](#) \* > `ShotsList`

## 6.4.1 Constructor & Destructor Documentation

### 6.4.1.1 `BallShot()`

```
BallShot::BallShot (
    Vector initialPosi,
    Vector direction,
    float radius,
    float speed = 40)
```

Constructs a ball-shaped shot and inserts it into the shot list.

#### Parameters

<i>initialPosi</i>	The initial position of the shot.
<i>direction</i>	A vector in which the shot moves.
<i>radius</i>	Radius of the shot.
<i>speed</i>	Speed of the shot.

## 6.4.2 Member Function Documentation

### 6.4.2.1 `draw()`

```
void BallShot::draw () [override], [virtual]
```

Draw a ball.

Implements [Shot](#).

#### 6.4.2.2 `isItActive()`

```
bool BallShot::isItActive () [override], [virtual]
```

Checks if the ball (shot) positions are within the game region.

##### Returns

If the ball shot is still active.

Implements [Shot](#).

#### 6.4.2.3 `shotCollidedWithBoss()`

```
bool BallShot::shotCollidedWithBoss (
    WindowsBoss & boss) [override], [virtual]
```

Checks if the ball (shot) collided with the boss. (Circle-square collision.)

##### Parameters

<i>boss</i>	Boss address(windows).
-------------	------------------------

##### Returns

If the ball shot collided like the boss.

Implements [Shot](#).

#### 6.4.2.4 `shotCollidedWithPlayer()`

```
bool BallShot::shotCollidedWithPlayer (
    FixedShip & player) [override], [virtual]
```

Check if the ball (shot) collided with the player. (Circle-to-circle collision.)

##### Parameters

<i>player</i>	Player address.
---------------	-----------------

##### Returns

If the ball shot collided like the player.

Implements [Shot](#).

#### 6.4.2.5 update()

```
void BallShot::update () [override], [virtual]
```

Add to the position of the ball (shot), the direction vector multiplied by the velocity. (normally the direction vectors are unitary)

Implements [Shot](#).

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

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

## 6.5 Bootstrap Class Reference

Static class that encapsulates initialization and cleanup of allegro components.

```
#include <bootstrap.hpp>
```

### Static Public Member Functions

- static bool [initialize\\_allegro \(\)](#)  
*Fully initializes Allegro.*
- static bool [init\\_allegro\\_libs \(\)](#)  
*Initializes allegro libs one by one while checking if sucessfull, if not, print an error.*
- static void [register\\_allegro\\_events \(\)](#)  
*Regiester all allegro event sources used in the game.*
- static void [cleanup\\_allegro \(\)](#)  
*Frees the memory deleting all pointers and destroys allegro components.*
- static bool [file\\_exists \(const char \\*path\)](#)  
*utility class that checks if a file exists*
- static void [start\\_sprite \(ALLEGRO\\_BITMAP \\*&bitm, const char \\*path\)](#)  
*utility member function to check if the sprite is loadable*
- static void [start\\_font \(ALLEGRO\\_FONT \\*&font, const char \\*path, int size\)](#)  
*utility member function to check if the font is loadable*

### 6.5.1 Detailed Description

Static class that encapsulates initialization and cleanup of allegro components.

## 6.5.2 Member Function Documentation

### 6.5.2.1 cleanup\_allegro()

```
void Bootstrap::cleanup_allegro () [static]
```

Frees the memory deleting all pointers and destroys allegro components.

### 6.5.2.2 file\_exists()

```
bool Bootstrap::file_exists (
    const char * path) [static]
```

utility class that checks if a file exists

**Parameters**

<i>path</i>	Path to the file
-------------	------------------

**Returns**

boolean indicating if the file exists

**6.5.2.3 init\_allegro\_libs()**

```
bool Bootstrap::init_allegro_libs () [static]
```

Initializes allegro libs one by one while checking if sucessfull, if not, print an error.

**Returns**

true if all initializes correctly, false if not

**6.5.2.4 initialize\_allegro()**

```
bool Bootstrap::initialize_allegro () [static]
```

Fully initializes Allegro.

**6.5.2.5 register\_allegro\_events()**

```
void Bootstrap::register_allegro_events () [static]
```

Regiester all allegro event sources used in the game.

**6.5.2.6 start\_font()**

```
void Bootstrap::start_font (
    ALLEGRO_FONT *& font,
    const char * path,
    int size) [static]
```

utility member function to check if the font is loadable

**Parameters**

<i>font</i>	reference to a ALLEGRO_FONT pointer that will be loaded
<i>path</i>	path to the ttf font file

**6.5.2.7 start\_sprite()**

```
void Bootstrap::start_sprite (
    ALLEGRO_BITMAP *& bitm,
    const char * path) [static]
```

utility member function to check if the sprite is loadable

### Parameters

<i>bitm</i>	reference to a ALLEGRO_BITMAP pointer that will be loaded
<i>path</i>	path to the image file

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

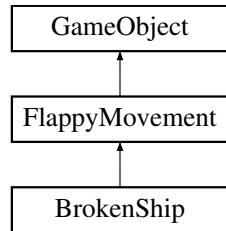
- include/bootstrap.hpp
- src/bootstrap.cpp

## 6.6 BrokenShip Class Reference

Player of phases 1 and 2, implements Flappy Movement.

```
#include <game_object.hpp>
```

Inheritance diagram for BrokenShip:



### Public Member Functions

- **BrokenShip ()**
- **BrokenShip (const Vector &pos)**

*Create a new `BrokenShip` at given position.*
- float **get\_radius () const**

*Get the radius of the `BrokenShip`.*
- void **set\_radius (float r)**

*Sets the radius of the `BrokenShip` Hitbox's.*
- void **update ()**

*Updates the physics of the `BrokenShip` by gravity and then re-draw it on the screen.*
- void **draw ()**

*Draw the `BrokenShip` on the screen.*
- void **restart ()**

*Moves the `BrokenShip` back to default position.*

### Public Member Functions inherited from FlappyMovement

- void **apply\_gravity ()**

*Apply gravity physics on the object.*
- void **move\_flappy ()**

*Apply movement physics on the object.*
- **Vector getMoveForce ()**

*Get the Move Force of a flappy movement.*

## Public Member Functions inherited from [GameObject](#)

- [GameObject \(\)](#)  
*Create a new default [GameObject](#).*
- [GameObject \(Vector position\)](#)  
*Create a new [GameObject](#) on given position.*
- [Vector get\\_position \(\)](#)  
*Get the position of a [GameObject](#).*
- [void set\\_position \(const Vector &position\)](#)  
*Sets the position of a [GameObject](#).*
- [virtual ~GameObject \(\)=0](#)  
*Game Object empty destructor.*
- [void set\\_bitmap \(const char \\*path\)](#)  
*Sets the sprite of the [GameObject](#).*

## Additional Inherited Members

### Public Attributes inherited from [GameObject](#)

- [ALLEGRO\\_BITMAP \\* objectSprite = NULL](#)

### Protected Attributes inherited from [GameObject](#)

- [Vector \\_position](#)

## 6.6.1 Detailed Description

Player of phases 1 and 2, implements Flappy Movement.

- [\\_radius: Object's Hitbox radius](#)

## 6.6.2 Constructor & Destructor Documentation

### 6.6.2.1 [BrokenShip\(\) \[1/2\]](#)

```
BrokenShip::BrokenShip ()
```

### 6.6.2.2 [BrokenShip\(\) \[2/2\]](#)

```
BrokenShip::BrokenShip (const Vector & pos)
```

Create a new [BrokenShip](#) at given position.

## 6.6.3 Member Function Documentation

### 6.6.3.1 draw()

```
void BrokenShip::draw ()
```

Draw the [BrokenShip](#) on the screen.

### 6.6.3.2 get\_radius()

```
float BrokenShip::get_radius () const
```

Get the radius of the [BrokenShip](#).

#### Returns

Float that represents the [BrokenShip](#)'s Hitbox radius

### 6.6.3.3 restart()

```
void BrokenShip::restart ()
```

Moves the [BrokenShip](#) back to default position.

### 6.6.3.4 set\_radius()

```
void BrokenShip::set_radius (
    float r)
```

Sets the radius of the [BrokenShip](#) Hitbox's.

### 6.6.3.5 update()

```
void BrokenShip::update ()
```

Updates the physics of the [BrokenShip](#) by gravity and then re-draw it on the screen.

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

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

## 6.7 Button Class Reference

simple button logic with draw, click checking and setText functions

```
#include <interface.hpp>
```

## Public Member Functions

- `Button (Coordinates coords, ALLEGRO_COLOR color, std::string text, ALLEGRO_FONT *font, bool drawBackground=true)`  
*Button constructor.*
- `void drawButton ()`  
*Draws the button using its attributes.*
- `bool gotClicked (int mx, int my)`  
*checks if the mouse coordinates (when click occurs) matches the button region*
- `void setText (const char *txt)`  
*set a new text to the button*

### 6.7.1 Detailed Description

simple button logic with draw, click checking and setText functions

- `_coords`: x and y coordinates, width and height of the created button
- `_text`: button text
- `_font`: pointer to an allegro font
- `_drawBackground`: option to draw or not to draw the background

### 6.7.2 Constructor & Destructor Documentation

#### 6.7.2.1 `Button()`

```
Button::Button (
    Coordinates coords,
    ALLEGRO_COLOR color,
    std::string text,
    ALLEGRO_FONT * font,
    bool drawBackground = true)
```

`Button` constructor.

##### Parameters

<code>coords</code>	button dimensions using the <code>Coordinates</code> class
<code>color</code>	allegro-typed rgb color
<code>text</code>	button center text
<code>font</code>	center text font
<code>drawBackground</code>	bool that sets if the background will be displayed

### 6.7.3 Member Function Documentation

#### 6.7.3.1 `drawButton()`

```
void Button::drawButton ()
```

Draws the button using its attributes.

### 6.7.3.2 gotClicked()

```
bool Button::gotClicked (
    int mx,
    int my)
```

checks if the mouse coordinates (when click occurs) matches the button region

#### Parameters

<i>mx</i>	mouse x coordinates
<i>my</i>	mouse y coordinates

### 6.7.3.3 setText()

```
void Button::setText (
    const char * txt)
```

set a new text to the button

#### Parameters

<i>txt</i>	new text to substitute the old one
------------	------------------------------------

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

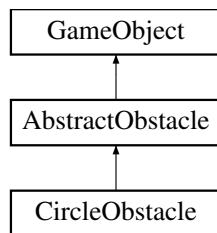
- include/interface.hpp
- src/interface.cpp

## 6.8 CircleObstacle Class Reference

Representa um obstáculo com uma forma de colisão perfeitamente circular.

```
#include <circle_obstacle.hpp>
```

Inheritance diagram for CircleObstacle:



## Public Member Functions

- `CircleObstacle` (const `Vector` &pos, const char \*path)  
*Construtor da classe `CircleObstacle`.*
- float `get_radius` () const  
*Obtém o raio do obstáculo circular.*
- void `draw` () override  
*Desenha o obstáculo na tela.*
- void `update` () override  
*Atualiza a posição do obstáculo.*
- bool `checkCollisionWithPlayer` (`BrokenShip` &player) override  
*Verifica se há colisão entre este obstáculo e a nave do player.*

## Public Member Functions inherited from `AbstractObstacle`

- virtual ~`AbstractObstacle` ()=default

## Public Member Functions inherited from `GameObject`

- `GameObject` ()  
*Create a new default `GameObject`.*
- `GameObject` (`Vector` position)  
*Create a new `GameObject` on given position.*
- `Vector get_position` ()  
*Get the position of a `GameObject`.*
- void `set_position` (const `Vector` &position)  
*Sets the position of a `GameObject`.*
- virtual ~`GameObject` ()=0  
*Game Object empty destructor.*
- void `set_bitmap` (const char \*path)  
*Sets the sprite of the `GameObject`.*

## Additional Inherited Members

### Public Attributes inherited from `GameObject`

- ALLEGRO\_BITMAP \* `objectSprite` = NULL

### Protected Member Functions inherited from `AbstractObstacle`

- void `setSpeed` (`Vector` speed)  
*Define a velocidade do obstáculo.*
- `Vector getSpeed` ()  
*Obtém a velocidade atual do obstáculo.*

### Protected Attributes inherited from `GameObject`

- `Vector _position`

### 6.8.1 Detailed Description

Representa um obstáculo com uma forma de colisão perfeitamente circular.

Esta classe herda de [AbstractObstacle](#) e implementa a lógica para um obstáculo simples cuja detecção de colisão é baseada em um raio.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 CircleObstacle()

```
CircleObstacle::CircleObstacle (
    const Vector & pos,
    const char * path)
```

Construtor da classe [CircleObstacle](#).

Inicializa um obstáculo circular com uma posição, uma imagem (bitmap) e uma velocidade vertical aleatória, selecionada a partir de um vetor de velocidades predefinido, com velocidades plausíveis para o jogo.

#### Parameters

<i>pos</i>	A posição inicial (Vetor x, y) do obstáculo.
<i>path</i>	O caminho para o arquivo de imagem (sprite) do obstáculo.

### 6.8.3 Member Function Documentation

#### 6.8.3.1 checkCollisionWithPlayer()

```
bool CircleObstacle::checkCollisionWithPlayer (
    BrokenShip & player) [override], [virtual]
```

Verifica se há colisão entre este obstáculo e a nave do player.

A detecção é feita comparando a distância entre os centros do obstáculo e do jogador com a soma de seus raios.

#### Parameters

<i>player</i>	Uma referência ao objeto do jogador ( <a href="#">BrokenShip</a> ) para verificar a colisão.
---------------	--

#### Returns

Retorna 'true' se a distância for menor que a soma dos raios (houve colisão), e 'false' caso contrário.

Implements [AbstractObstacle](#).

### 6.8.3.2 draw()

```
void CircleObstacle::draw () [override], [virtual]
```

Desenha o obstáculo na tela.

Calcula as coordenadas de desenho com base na posição central e no raio, ajustando sua escala para corresponder ao diâmetro do obstáculo.

Implements [AbstractObstacle](#).

### 6.8.3.3 get\_radius()

```
float CircleObstacle::get_radius () const
```

Obtém o raio do obstáculo circular.

#### Returns

O valor do raio (`_radius`) do obstáculo.

### 6.8.3.4 update()

```
void CircleObstacle::update () [override], [virtual]
```

Atualiza a posição do obstáculo.

Move o obstáculo de acordo com sua velocidade. Se ele sair da parte inferior da tela é reposicionado na parte superior com uma coordenada na largura aleatória, criando um efeito de loop contínuo.

Implements [AbstractObstacle](#).

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

- [include/circle\\_obstacle.hpp](#)
- [src/circle\\_obstacle.cpp](#)

## 6.9 Coordinates Struct Reference

Extremely basic coordinates struct. Takes x and y coordinates, width and height as parameters. Mostly used to keep better logic in buttons.

```
#include <interface.hpp>
```

### Public Member Functions

- [Coordinates](#) (double x, double y, double width, double height)  
*Coordinates constructor.*
- [Coordinates](#) ()

## Public Attributes

- double `_x`
- double `_y`
- double `_width`
- double `_height`

### 6.9.1 Detailed Description

Extremely basic coordinates struct. Takes x and y coordinates, width and height as parameters. Mostly used to keep better logic in buttons.

### 6.9.2 Constructor & Destructor Documentation

#### 6.9.2.1 Coordinates() [1/2]

```
Coordinates::Coordinates (
    double x,
    double y,
    double width,
    double height)
```

[Coordinates](#) constructor.

##### Parameters

<code>x</code>	x coordinates in the display
<code>y</code>	y coordinates in the display
<code>width</code>	region width
<code>height</code>	region height

#### 6.9.2.2 Coordinates() [2/2]

```
Coordinates::Coordinates () [inline]
```

### 6.9.3 Member Data Documentation

#### 6.9.3.1 `_height`

```
double Coordinates::_height
```

#### 6.9.3.2 `_width`

```
double Coordinates::_width
```

### 6.9.3.3 `_x`

```
double Coordinates::_x
```

### 6.9.3.4 `_y`

```
double Coordinates::_y
```

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

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

## 6.10 DatabaseUsers Class Reference

Gerencia todas as operações de banco de dados relacionadas a usuários.

```
#include <database_users.hpp>
```

### Public Member Functions

- [`DatabaseUsers \(\)`](#)  
*Construtor da classe `DatabaseUsers`.*
- [`~DatabaseUsers \(\)`](#)  
*Destrutor da classe `DatabaseUsers`.*
- [`bool registerUser \(const std::string &name, const std::string &username, const std::string &password, int initialScore=0, int initialGames=0\)`](#)  
*Registra um novo usuário no banco de dados.*
- [`bool deleteUser \(const std::string &username\)`](#)  
*Exclui um usuário do banco de dados com base no nome de usuário.*
- [`std::vector< User > listUsers \(\)`](#)  
*Retorna uma lista de todos os usuários cadastrados.*
- [`bool updateScore \(const std::string &username, int new\_score\)`](#)  
*Atualiza a pontuação de um usuário específico.*
- [`bool updateGamesNumber \(const std::string &username, int new\_games\)`](#)  
*Atualiza o número total de jogos de um usuário.*
- [`std::unique\_ptr< User > getUserByUsername \(const std::string &username\)`](#)  
*Busca e retorna os dados completos de um usuário pelo seu nome de usuário.*
- [`bool authenticateUser \(const std::string &username, const std::string &password\)`](#)  
*Autentica um usuário com base no nome de usuário e senha.*
- [`void addValuesGameOverScreen \(std::string &username, gameOverScreen &game\_over\_screen\)`](#)  
*Processa e atualiza os dados de fim de um dos mini-jogos para um usuário.*

### 6.10.1 Detailed Description

Gerencia todas as operações de banco de dados relacionadas a usuários.

Encapsula a conexão com o banco de dados PostgreSQL (usando pqxx) e implementa uma API para criar, autenticar, atualizar, deletar e listar usuários.

## 6.10.2 Constructor & Destructor Documentation

### 6.10.2.1 DatabaseUsers()

```
DatabaseUsers::DatabaseUsers ()
```

Construtor da classe [DatabaseUsers](#).

Inicializa a conexão com o banco de dados PostgreSQL. Lê a string de conexão a partir de um arquivo .env, estabelece a conexão e lança uma exceção se a conexão não puder ser estabelecida com sucesso.

#### Exceptions

<code>std::runtime_error</code>	Se a conexão com o banco de dados falhar por um motivo genérico.
<code>pqxx::sql_error</code>	Se ocorrer um erro específico do SQL durante a tentativa de conexão.

### 6.10.2.2 ~DatabaseUsers()

```
DatabaseUsers::~DatabaseUsers ()
```

Destrutor da classe [DatabaseUsers](#).

Fechá a conexão com o banco de dados, se ela estiver ativa, para liberar os recursos de forma segura.

## 6.10.3 Member Function Documentation

### 6.10.3.1 addValuesGameOverScreen()

```
void DatabaseUsers::addValuesGameOverScreen (
    std::string & username,
    gameOverScreen & game_over_screen)
```

Processa e atualiza os dados de fim de um dos mini-jogos para um usuário.

Esta função é chamada ao final de uma partida. Ela atualiza o número de jogos do usuário, verifica se a pontuação da partida é um novo recorde pessoal (e atualiza se for), e preenche um objeto de tela de game over com todas as informações relevantes (pontuação obtida nesse exato momento, recorde pessoal, recorde geral do jogo).

#### Parameters

<code>username</code>	O nome de usuário do jogador.
<code>game_over_screen</code>	Referência ao objeto da tela de "Game Over" que conterá os dados.

### 6.10.3.2 authenticateUser()

```
bool DatabaseUsers::authenticateUser (
    const std::string & username,
    const std::string & password)
```

Autentica um usuário com base no nome de usuário e senha.

**Parameters**

<i>username</i>	O nome de usuário para autenticação.
<i>password</i>	A senha fornecida pelo usuário.

**Returns**

'true' se o nome de usuário e a senha corresponderem, 'false' caso contrário.

**6.10.3.3 deleteUser()**

```
bool DatabaseUsers::deleteUser (
    const std::string & username)
```

Exclui um usuário do banco de dados com base no nome de usuário.

**Parameters**

<i>username</i>	O nome de usuário a ser excluído.
-----------------	-----------------------------------

**Returns**

'true' se o usuário foi encontrado e excluído com sucesso, 'false' caso contrário.

**6.10.3.4 getUserByUsername()**

```
std::unique_ptr< User > DatabaseUsers::getUserByUsername (
    const std::string & username)
```

Busca e retorna os dados completos de um usuário pelo seu nome de usuário.

**Parameters**

<i>username</i>	O nome de usuário a ser buscado.
-----------------	----------------------------------

**Returns**

Um ponteiro único (`std::unique_ptr`) para um objeto `User` se o usuário for encontrado. Retorna `nullptr` se o usuário não for encontrado ou em caso de erro.

**6.10.3.5 listUsers()**

```
std::vector< User > DatabaseUsers::listUsers ()
```

Retorna uma lista de todos os usuários cadastrados.

Executa uma consulta para buscar todos os usuários, ordenando o resultado pela pontuação (score) em ordem decrescente, o que facilita a obter a maior pontuação geral depois.

**Returns**

Um vetor de objetos '`User`'. Retorna um vetor vazio se não houver usuários ou em caso de erro.

### 6.10.3.6 registerUser()

```
bool DatabaseUsers::registerUser (
    const std::string & name,
    const std::string & username,
    const std::string & password,
    int initialScore = 0,
    int initialGames = 0)
```

Registra um novo usuário no banco de dados.

Insere um novo registro na tabela 'users'.

#### Parameters

<i>name</i>	O nome do usuário.
<i>username</i>	O nome de login do usuário (deve ser único).
<i>password</i>	A senha do usuário.
<i>initial_score</i>	A pontuação inicial a ser definida para o usuário, 0 de default.
<i>initial_games</i>	O número inicial de jogos a ser definido para o usuário, 0 de default.

#### Returns

'true' se o usuário for registrado com sucesso, 'false' se ocorrer um erro (ex: username já existe).

### 6.10.3.7 updateGamesNumber()

```
bool DatabaseUsers::updateGamesNumber (
    const std::string & username,
    int new_games)
```

Atualiza o número total de jogos de um usuário.

#### Parameters

<i>username</i>	O nome de usuário cujo número de jogos será atualizado.
<i>new_games</i>	O novo número total de jogos.

#### Returns

'true' se a atualização for bem-sucedida, 'false' se o usuário não for encontrado ou em caso de erro.

### 6.10.3.8 updateScore()

```
bool DatabaseUsers::updateScore (
    const std::string & username,
    int new_score)
```

Atualiza a pontuação de um usuário específico.

**Parameters**

<i>username</i>	O nome de usuário cujo score será atualizado.
<i>new_score</i>	A nova pontuação a ser registrada.

**Returns**

'true' se a atualização for bem-sucedida, 'false' se o usuário não for encontrado ou em caso de erro.

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

- include/[database\\_users.hpp](#)
- src/[database\\_users.cpp](#)

## 6.11 dotenv Class Reference

```
#include <dotenv.h>
```

### Public Member Functions

- [dotenv \(\)=delete](#)
- [~dotenv \(\)=delete](#)

### Static Public Member Functions

- static void [init \(const char \\*filename=".env"\)](#)
- static void [init \(int flags, const char \\*filename=".env"\)](#)
- static std::string [getenv \(const char \\*name, const std::string &def=""\)](#)

### Static Public Attributes

- static const unsigned char [Preserve](#) = 1 << 0
- static const int [OptionsNone](#) = 0

### 6.11.1 Detailed Description

Utility class for loading environment variables from a file.

### 6.11.1.0.1 Typical use

Given a file .env

```
DATABASE_HOST=localhost
DATABASE_USERNAME=user
DATABASE_PASSWORD="antipasto"
```

and a program example.cpp

```
// example.cpp
#include <iostream>
#include <dotenv.h>

int main()
{
    dotenv::init();

    std::cout << std::getenv("DATABASE_USERNAME") << std::endl;
    std::cout << std::getenv("DATABASE_PASSWORD") << std::endl;

    return 0;
}
```

Compile and run the program, e.g. using,

```
c++ example.cpp -o example -I/usr/local/include/laserpants/dotenv-0.9.3 && ./example
```

and the output is:

```
user
antipasto
```

See also

<https://github.com/laserpants/dotenv-cpp>

## 6.11.2 Constructor & Destructor Documentation

### 6.11.2.1 dotenv()

```
dotenv::dotenv () [delete]
```

### 6.11.2.2 ~dotenv()

```
dotenv::~dotenv () [delete]
```

## 6.11.3 Member Function Documentation

### 6.11.3.1 getenv()

```
std::string dotenv::getenv (
    const char * name,
    const std::string & def = "") [inline], [static]
```

Wrapper for std::getenv() which also takes a default value, in case the variable turns out to be empty.

**Parameters**

<i>name</i>	the name of the variable to look up
<i>def</i>	a default value

**Returns**

the value of the environment variable *name*, or *def* if the variable is not set

**6.11.3.2 init() [1/2]**

```
void dotenv::init (
    const char * filename = ".env") [inline], [static]
```

Read and initialize environment variables from the `.env` file, or a file specified by the *filename* argument.

**Parameters**

<i>filename</i>	a file to read environment variables from
-----------------	---

**6.11.3.3 init() [2/2]**

```
void dotenv::init (
    int flags,
    const char * filename = ".env") [inline], [static]
```

Read and initialize environment variables using the provided configuration flags.

By default, if a name is already present in the environment, `dotenv::init()` will replace it with the new value. To preserve existing variables, you must pass the `Preserve` flag.

```
dotenv::init(dotenv::Preserve);
```

**Parameters**

<i>flags</i>	configuration flags
<i>filename</i>	a file to read environment variables from

**6.11.4 Member Data Documentation****6.11.4.1 OptionsNone**

```
const int dotenv::OptionsNone = 0 [static]
```

#### 6.11.4.2 Preserve

```
const unsigned char dotenv::Preserve = 1 << 0 [static]
```

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

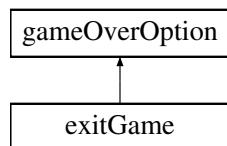
- [include/dotenv.h](#)

## 6.12 exitGame Class Reference

Ação para sair do jogo.

```
#include <game_over.hpp>
```

Inheritance diagram for exitGame:



### Public Member Functions

- [~exitGame \(\) override=default](#)
- [void execute \(\) override](#)

*Função virtual pura que será sobrescrita pelas classes derivadas.*

### Public Member Functions inherited from [gameOverOption](#)

- [virtual ~gameOverOption \(\)=default](#)

*Destruitor virtual padrão para que a memória seja liberada corretamente nas classes derivadas.*

### 6.12.1 Detailed Description

Ação para sair do jogo.

## 6.12.2 Constructor & Destructor Documentation

### 6.12.2.1 [~exitGame\(\)](#)

```
exitGame::~exitGame () [override], [default]
```

### 6.12.3 Member Function Documentation

#### 6.12.3.1 execute()

```
void exitGame::execute () [override], [virtual]
```

Função virtual pura que será sobreescrita pelas classes derivadas.

Implements [gameOverOption](#).

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

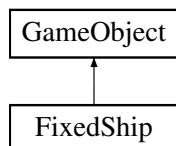
- [include/game\\_over.hpp](#)
- [src/gameover.cpp](#)

## 6.13 FixedShip Class Reference

Player of phase 3, implements 2D movement.

```
#include <game_object.hpp>
```

Inheritance diagram for FixedShip:



### Public Member Functions

- [FixedShip \(\)](#)
- [FixedShip \(const Vector &pos\)](#)

*Creates a new `FixedShip`.*
- [float get\\_radius \(\) const](#)

*Get `FixedShip(player)`'s Hitbox radius.*
- [void set\\_radius \(float r\)](#)

*Sets a radius to the player.*
- [void moveShip \(char direction\)](#)

*Implements 2D movement on `FixedShip`.*
- [void draw \(\)](#)

*Draw the player on the screen.*
- [void setCanTakeDamage \(bool canTakeDamage\)](#)

*Method of setting whether or not the player can take damage.*
- [void takeDamage \(bool &playing, int damage=1\)](#)

*Removes a life from the player, if the life is zero, the game is over.*

## Public Member Functions inherited from [GameObject](#)

- [GameObject \(\)](#)  
*Create a new default [GameObject](#).*
- [GameObject \(Vector position\)](#)  
*Create a new [GameObject](#) on given position.*
- [Vector get\\_position \(\)](#)  
*Get the position of a [GameObject](#).*
- [void set\\_position \(const Vector &position\)](#)  
*Sets the position of a [GameObject](#).*
- [virtual ~GameObject \(\)=0](#)  
*Game Object empty destructor.*
- [void set\\_bitmap \(const char \\*path\)](#)  
*Sets the sprite of the [GameObject](#).*

## Additional Inherited Members

### Public Attributes inherited from [GameObject](#)

- ALLEGRO\_BITMAP \* [objectSprite](#) = NULL

### Protected Attributes inherited from [GameObject](#)

- [Vector \\_position](#)

## 6.13.1 Detailed Description

Player of phase 3, implements 2D movement.

- move\_force: [Vector](#) that represents the force object can exerce on himself
- \_radius: Object's Hitbox radius

## 6.13.2 Constructor & Destructor Documentation

### 6.13.2.1 [FixedShip\(\) \[1/2\]](#)

```
FixedShip::FixedShip ()
```

### 6.13.2.2 [FixedShip\(\) \[2/2\]](#)

```
FixedShip::FixedShip (
    const Vector & pos)
```

Creates a new [FixedShip](#).

### 6.13.3 Member Function Documentation

#### 6.13.3.1 draw()

```
void FixedShip::draw ()
```

Draw the player on the screen.

#### 6.13.3.2 get\_radius()

```
float FixedShip::get_radius () const
```

Get [FixedShip\(player\)](#)'s Hitbox radius.

##### Returns

A float that indicates the player's Hitbox Radius

#### 6.13.3.3 moveShip()

```
void FixedShip::moveShip (
    char direction)
```

Implements 2D movement on [FixedShip](#).

#### 6.13.3.4 set\_radius()

```
void FixedShip::set_radius (
    float r)
```

Sets a radius to the player.

#### 6.13.3.5 setCanTakeDamage()

```
void FixedShip::setCanTakeDamage (
    bool canTakeDamage)
```

Method of setting whether or not the player can take damage.

##### Parameters

<code>canTakeDamage</code>	Boolean using to set.
----------------------------	-----------------------

#### 6.13.3.6 takeDamage()

```
void FixedShip::takeDamage (
    bool & playing,
    int damage = 1)
```

Removes a life from the player, if the life is zero, the game is over.

### Parameters

<i>playing</i>	Variable that controls whether the game is active.
<i>damage</i>	Integer that informs the damage taken.

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

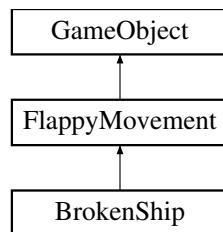
- include/game\_object.hpp
- src/game\_object.cpp

## 6.14 FlappyMovement Class Reference

Base class that implements Flappy-Bird-like movement.

```
#include <game_object.hpp>
```

Inheritance diagram for FlappyMovement:



### Public Member Functions

- void [apply\\_gravity \(\)](#)  
*Apply gravity physics on the object.*
- void [move\\_flappy \(\)](#)  
*Apply movement physics on the object.*
- [Vector getMoveForce \(\)](#)  
*Get the Move Force of a flappy movement.*

### Public Member Functions inherited from [GameObject](#)

- [GameObject \(\)](#)  
*Create a new default [GameObject](#).*
- [GameObject \(Vector position\)](#)  
*Create a new [GameObject](#) on given position.*
- [Vector get\\_position \(\)](#)  
*Get the position of a [GameObject](#).*
- void [set\\_position \(const Vector &position\)](#)  
*Sets the position of a [GameObject](#).*
- virtual [~GameObject \(\)=0](#)  
*Game Object empty destructor.*
- void [set\\_bitmap \(const char \\*path\)](#)  
*Sets the sprite of the [GameObject](#).*

## Additional Inherited Members

### Public Attributes inherited from [GameObject](#)

- ALLEGRO\_BITMAP \* [objectSprite](#) = NULL

### Protected Attributes inherited from [GameObject](#)

- [Vector \\_position](#)

## 6.14.1 Detailed Description

Base class that implements Flappy-Bird-like movement.

- gravity: [Vector](#) that represents the force that pulls the main character
- move\_force: [Vector](#) that represents the force object applies on itself to move

## 6.14.2 Member Function Documentation

### 6.14.2.1 [apply\\_gravity\(\)](#)

```
void FlappyMovement::apply_gravity ()
```

Apply gravity physics on the object.

### 6.14.2.2 [getMoveForce\(\)](#)

```
Vector FlappyMovement::getMoveForce ()
```

Get the Move Force of a flappy movement.

#### Returns

A [Vector](#) that represents the mount of force the object applies on itself in order to move

### 6.14.2.3 [move\\_flappy\(\)](#)

```
void FlappyMovement::move_flappy ()
```

Apply movement physics on the object.

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

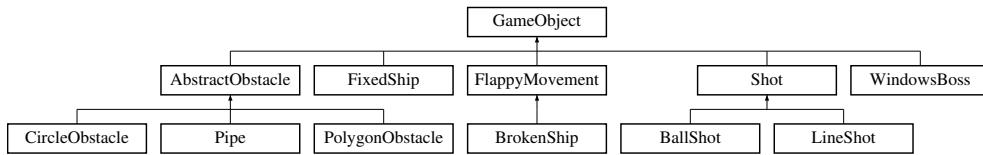
- [include/game\\_object.hpp](#)
- [src/game\\_object.cpp](#)

## 6.15 GameObject Class Reference

Base class to objects of the game.

```
#include <game_object.hpp>
```

Inheritance diagram for GameObject:



### Public Member Functions

- [GameObject \(\)](#)  
*Create a new default GameObject.*
- [GameObject \(Vector position\)](#)  
*Create a new GameObject on given position.*
- [Vector get\\_position \(\)](#)  
*Get the position of a GameObject.*
- [void set\\_position \(const Vector &position\)](#)  
*Sets the position of a GameObject.*
- [virtual ~GameObject \(\)=0](#)  
*Game Object empty destructor.*
- [void set\\_bitmap \(const char \\*path\)](#)  
*Sets the sprite of the GameObject.*

### Public Attributes

- `ALLEGRO_BITMAP * objectSprite = NULL`

### Protected Attributes

- `Vector _position`

#### 6.15.1 Detailed Description

Base class to objects of the game.

- `_position`: [Vector](#) that represents object's current position on screen

## 6.15.2 Constructor & Destructor Documentation

### 6.15.2.1 GameObject() [1/2]

```
GameObject::GameObject ()
```

Create a new default [GameObject](#).

### 6.15.2.2 GameObject() [2/2]

```
GameObject::GameObject (
    Vector position)
```

Create a new [GameObject](#) on given position.

### 6.15.2.3 ~GameObject()

```
GameObject::~GameObject () [pure virtual]
```

Game Object empty destructor.

## 6.15.3 Member Function Documentation

### 6.15.3.1 get\_position()

```
Vector GameObject::get_position ()
```

Get the position of a [GameObject](#).

#### Returns

A vector that represents the [GameObject](#) position

### 6.15.3.2 set\_bitmap()

```
void GameObject::set_bitmap (
    const char * path)
```

Sets the sprite of the [GameObject](#).

### 6.15.3.3 set\_position()

```
void GameObject::set_position (
    const Vector & position)
```

Sets the position of a [GameObject](#).

## 6.15.4 Member Data Documentation

### 6.15.4.1 `_position`

```
Vector GameObject::_position [protected]
```

### 6.15.4.2 `objectSprite`

```
ALLEGRO_BITMAP* GameObject::objectSprite = NULL
```

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

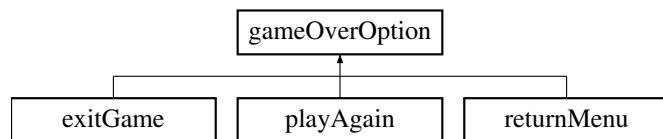
- [include/game\\_object.hpp](#)
- [src/game\\_object.cpp](#)

## 6.16 gameOverOption Class Reference

Classe base abstrata para as opções de ação na tela de Game Over.

```
#include <game_over.hpp>
```

Inheritance diagram for gameOverOption:



### Public Member Functions

- virtual `~gameOverOption ()=default`  
*Destruitor virtual padrão para que a memória seja liberada corretamente nas classes derivadas.*
- virtual void `execute ()=0`  
*Função virtual pura que será sobrescrita pelas classes derivadas.*

### 6.16.1 Detailed Description

Classe base abstrata para as opções de ação na tela de Game Over.

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 `~gameOverOption()`

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

Destruitor virtual padrão para que a memória seja liberada corretamente nas classes derivadas.

### 6.16.3 Member Function Documentation

#### 6.16.3.1 execute()

```
virtual void gameOverOption::execute () [pure virtual]
```

Função virtual pura que será sobreescrita pelas classes derivadas.

Implemented in [exitGame](#), [playAgain](#), and [returnMenu](#).

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

- [include/game\\_over.hpp](#)

## 6.17 gameOverScreen Class Reference

Gerencia a exibição da mensagem de Game Over, scores, botões de ação e a interação do jogador. `_currentScore`: pontuação na partida `_highScore`: maior pontuação (recorde pessoal) `_bestScore`: Melhor pontuação (recorde geral) `ALLEGRO_FONT* _font`: fonte dos textos da tela `_playAgainButton`: botão jogar novamente ← `_returnToMenuButton`: botão para retornar para a tela inicial `_exitGameButton`: botão para sair do jogo.

```
#include <game_over.hpp>
```

### Public Member Functions

- **[gameOverScreen \(ALLEGRO\\_FONT \\*font\)](#)**  
*Construtor da classe [gameOverScreen](#).*
- **[void setCurrentScore \(int score\)](#)**  
*metodos para acessar e modificar as pontuações do jogo e desenhar os elementos visuais na tela.*
- **[void setHighScore \(int score\)](#)**  
*Define o recorde (maior pontuação) a ser exibido na tela.*
- **[void setbestScore \(int score\)](#)**  
*Define o recorde global a ser exibido na tela.*
- **[void setnumGames \(int games\)](#)**  
*Define o numero de games jogados pelo usuario.*
- **[void draw \(\)](#)**  
*Desenha todos os elementos visuais na tela de Game Over.*
- **[gameOverOption \\* run \(ALLEGRO\\_EVENT\\_QUEUE \\*event\\_queue, ALLEGRO\\_TIMER \\*timer\)](#)**  
*Gerencia o loop de eventos da tela de Game Over e a interação do jogador.*

### 6.17.1 Detailed Description

Gerencia a exibição da mensagem de Game Over, scores, botões de ação e a interação do jogador. `_currentScore`: pontuação na partida `_highScore`: maior pontuação (recorde pessoal) `_bestScore`: Melhor pontuação (recorde geral) `ALLEGRO_FONT* _font`: fonte dos textos da tela `_playAgainButton`: botão jogar novamente ← `_returnToMenuButton`: botão para retornar para a tela inicial `_exitGameButton`: botão para sair do jogo.

## 6.17.2 Constructor & Destructor Documentation

### 6.17.2.1 gameOverScreen()

```
gameOverScreen::gameOverScreen (
    ALLEGRO_FONT * font)
```

Construtor da classe [gameOverScreen](#).

Construtor da tela de Game Over. Inicializa os membros da classe, incluindo a fonte e os botões com suas posições e textos.

## 6.17.3 Member Function Documentation

### 6.17.3.1 draw()

```
void gameOverScreen::draw ()
```

Desenha todos os elementos visuais na tela de Game Over.

Inclui a imagem de fundo, textos informativos e os botões.

### 6.17.3.2 run()

```
gameOverOption * gameOverScreen::run (
    ALLEGRO_EVENT_QUEUE * event_queue,
    ALLEGRO_TIMER * timer)
```

Gerencia o loop de eventos da tela de Game Over e a interação do jogador.

A tela permanece ativa, desenhando e capturando cliques, até que o jogador

#### Parameters

<i>event_queue</i>	Fila de eventos Allegro para capturar input.
<i>timer</i>	Timer Allegro para controlar o redesenho da tela.

#### Returns

Um ponteiro para o objeto '[gameOverOption](#)' que representa a ação escolhida pelo jogador.

#### Parameters

<i>event_queue</i>	Ponteiro para a fila de eventos Allegro do jogo.
<i>timer</i>	Ponteiro para o timer Allegro do jogo.

#### Returns

Ponteiro para o objeto [gameOverOption](#) que representa a ação escolhida pelo jogador.

### 6.17.3.3 setbestScore()

```
void gameOverScreen::setbestScore (
    int score)
```

Define o recorde global a ser exibido na tela.

### 6.17.3.4 setCurrentScore()

```
void gameOverScreen::setCurrentScore (
    int score)
```

metodos para acessar e modificar as pontuações do jogo e desenhar os elementos visuais na tela.

Define a pontuação da partida atual a ser exibida na tela..

### 6.17.3.5 setHighScore()

```
void gameOverScreen::setHighScore (
    int score)
```

Define o recorde (maior pontuação) a ser exibido na tela.

### 6.17.3.6 setnumGames()

```
void gameOverScreen::setnumGames (
    int games)
```

Define o numero de games jogados pelo usuario.

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

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

## 6.18 Interface Class Reference

Base game start interface using button class.

```
#include <interface.hpp>
```

### Public Member Functions

- [Interface](#) (ALLEGRO\_FONT \*font)  
*Construct the main menu interface using pre-setted buttons.*
- void [drawOffGameInterface](#) ()  
*Draws the main menu interface.*

## Public Attributes

- `Button playButton`
- `Button stopSongButton`
- `Button returnToMenuButton`
- `Button exitGameButton`

### 6.18.1 Detailed Description

Base game start interface using button class.

### 6.18.2 Constructor & Destructor Documentation

#### 6.18.2.1 Interface()

```
Interface::Interface (
    ALLEGRO_FONT * font)
```

Construct the main menu interface using pre-setted buttons.

### 6.18.3 Member Function Documentation

#### 6.18.3.1 drawOffGameInterface()

```
void Interface::drawOffGameInterface ()
```

Draws the main menu interface.

### 6.18.4 Member Data Documentation

#### 6.18.4.1 exitGameButton

```
Button Interface::exitGameButton
```

#### 6.18.4.2 playButton

```
Button Interface::playButton
```

#### 6.18.4.3 returnToMenuButton

```
Button Interface::returnToMenuButton
```

#### 6.18.4.4 stopSongButton

`Button Interface::stopSongButton`

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

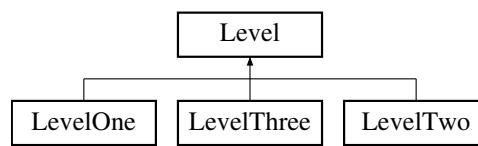
- [include/interface.hpp](#)
- [src/interface.cpp](#)

## 6.19 Level Class Reference

Base class for the game levels, encapsulates the bare minimum logic to maintain a level.

`#include <levels.hpp>`

Inheritance diagram for Level:



### Static Protected Attributes

- static `GameObject * _player = nullptr`
- static `Background _bg`
- static `ALLEGRO_EVENT _event`

### Friends

- void `interLevelHandling (std::vector< AbstractObstacle * > &obstacles, ALLEGRO_BITMAP *sprite, const char *message, float bitmapScale)`

*Clears the obstacles, draw a selected sprite with a user controlled scale and a message.*

### 6.19.1 Detailed Description

Base class for the game levels, encapsulates the bare minimum logic to maintain a level.

- `_music`: pointer to the level's music
- `_player`: pointer to the level's player (either `BrokenShip` or `FixedShip`)
- `_bg`: Parallax background
- `_event`: level's Allegro event

### 6.19.2 Friends And Related Symbol Documentation

#### 6.19.2.1 interLevelHandling

```
void interLevelHandling (
    std::vector< AbstractObstacle * > & obstacles,
    ALLEGRO_BITMAP * sprite,
    const char * message,
    float bitmapScale) [friend]
```

*Clears the obstacles, draw a selected sprite with a user controlled scale and a message.*

**Parameters**

<i>obstacles</i>	vector of abstract obstacles to be cleared
<i>sprite</i>	allegro-typed bitmap of sprite
<i>message</i>	message in the screen
<i>bitmapScale</i>	new width and height scale

**6.19.3 Member Data Documentation****6.19.3.1 `_bg`**

`Background` `Level::_bg` [static], [protected]

**6.19.3.2 `_event`**

`ALLEGRO_EVENT` `Level::_event` [static], [protected]

**6.19.3.3 `_player`**

`GameObject` \* `Level::_player` = `nullptr` [static], [protected]

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

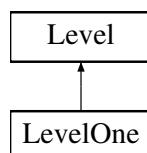
- [include/levels.hpp](#)
- [src/levels.cpp](#)

**6.20 LevelOne Class Reference**

First phase of the game, basic vertically-oriented flappy bird with satellites as pipes.

```
#include <levels.hpp>
```

Inheritance diagram for LevelOne:



## Static Public Member Functions

- static `BrokenShip * setLevelOne ()`  
*Initializes `Level One`.*
- static void `cleanLevel ()`  
*Frees memory used on `Level Two`.*
- static void `mainLoop (bool &playing, bool &isAlive)`  
*MainLoop of game's second phase.*
- static void `handleTimerEvents (bool &playing, BrokenShip *player, std::vector< AbstractObstacle * > &obstacles, bool &isAlive)`  
*Controls automatic events and events triggered by time, implements `DeltaTime`.*
- static void `handleKeyPressEvents (bool &playing, BrokenShip *player, bool &isAlive)`
- static void `handleKeyReleaseEvents ()`  
*Simple method for debug and logging.*

## Additional Inherited Members

### Static Protected Attributes inherited from `Level`

- static `GameObject * _player = nullptr`
- static `Background _bg`
- static `ALLEGRO_EVENT _event`

## 6.20.1 Detailed Description

First phase of the game, basic vertically-oriented flappy bird with satellites as pipes.

## 6.20.2 Member Function Documentation

### 6.20.2.1 `cleanLevel()`

```
void LevelOne::cleanLevel () [static]
```

Frees memory used on `Level Two`.

### 6.20.2.2 `handleKeyPressEvents()`

```
void LevelOne::handleKeyPressEvents (
    bool & playing,
    BrokenShip * player,
    bool & isAlive) [static]
```

### 6.20.2.3 `handleKeyReleaseEvents()`

```
void LevelOne::handleKeyReleaseEvents () [static]
```

Simple method for debug and logging.

#### 6.20.2.4 handleTimerEvents()

```
void LevelOne::handleTimerEvents (
    bool & playing,
    BrokenShip * player,
    std::vector< AbstractObstacle * > & obstacles,
    bool & isAlive) [static]
```

Controls automatic events and events triggered by time, implements DeltaTime.

**Parameters**

<i>playing</i>	loop control variable to close the game on collision
<i>player</i>	the player object of this phase (BrokenShip class)
<i>obstacles</i>	Reference to vector of AbstractObstacles of this phase

**6.20.2.5 mainLoop()**

```
void LevelOne::mainLoop (
    bool & playing,
    bool & isAlive) [static]
```

MainLoop of game's second phase.

**Parameters**

<i>playing</i>	Loop control variable to finish the level on collision or quit
----------------	--

**6.20.2.6 setLevelOne()**

```
BrokenShip * LevelOne::setLevelOne () [static]
```

Initializes [Level](#) One.

**Returns**

Pointer to the player object, in this case [BrokenShip](#)

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

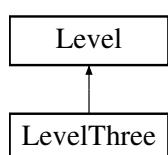
- [include/levels.hpp](#)
- [src/levels.cpp](#)

**6.21 LevelThree Class Reference**

Third phase of the game, a free-movement, shooter boss fight against [Windows](#) (The biggest piece of bloatware in earth)

```
#include <levels.hpp>
```

Inheritance diagram for LevelThree:



## Static Public Member Functions

- static `FixedShip * setLevelThree ()`  
*Initializes Level Three.*
- static void `cleanLevel ()`  
*Delete the player and clear the shots.*
- static void `mainLoop (bool &playing, bool &isAlive)`
- static void `updatePlayerPosition (FixedShip *player)`
- static void `handleTimerEvents (bool &playing, FixedShip *player, WindowsBoss &windows, bool &isAlive)`
- static void `handleKeyPressEvents (bool &playing, FixedShip *player, WindowsBoss &boss, bool &isAlive)`
- static void `handleKeyReleaseEvents (bool &playing)`
- static void `handleKeyPressEvents (bool &playing, FixedShip *player)`
- static void `handleKeyReleaseEvents ()`

## Additional Inherited Members

### Static Protected Attributes inherited from `Level`

- static `GameObject * _player = nullptr`
- static `Background _bg`
- static `ALLEGRO_EVENT _event`

## 6.21.1 Detailed Description

Third phase of the game, a free-movement, shooter boss fight against `Windows` (The biggest piece of bloatware in earth)

- Key\_pressed:

## 6.21.2 Member Function Documentation

### 6.21.2.1 `cleanLevel()`

```
void LevelThree::cleanLevel () [static]
```

Delete the player and clear the shots.

### 6.21.2.2 `handleKeyPressEvents() [1/2]`

```
void LevelThree::handleKeyPressEvents (
    bool & playing,
    FixedShip * player) [static]
```

**6.21.2.3 handleKeyPressEvents() [2/2]**

```
void LevelThree::handleKeyPressEvents (
    bool & playing,
    FixedShip * player,
    WindowsBoss & boss,
    bool & isAlive) [static]
```

**6.21.2.4 handleKeyReleaseEvents() [1/2]**

```
void LevelThree::handleKeyReleaseEvents () [static]
```

**6.21.2.5 handleKeyReleaseEvents() [2/2]**

```
void LevelThree::handleKeyReleaseEvents (
    bool & playing) [static]
```

**6.21.2.6 handleTimerEvents()**

```
void LevelThree::handleTimerEvents (
    bool & playing,
    FixedShip * player,
    WindowsBoss & windows,
    bool & isAlive) [static]
```

**6.21.2.7 mainLoop()**

```
void LevelThree::mainLoop (
    bool & playing,
    bool & isAlive) [static]
```

**6.21.2.8 setLevelThree()**

```
FixedShip * LevelThree::setLevelThree () [static]
```

Initializes [Level](#) Three.

**Returns**

Pointer to the player object, in this case [FixedShip](#)

**6.21.2.9 updatePlayerPosition()**

```
void LevelThree::updatePlayerPosition (
    FixedShip * player) [static]
```

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

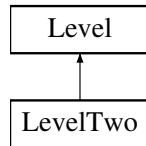
- [include/levels.hpp](#)
- [src/levels.cpp](#)

## 6.22 LevelTwo Class Reference

Second phase of the game, vertically-oriented flappy bird with moving obstacles (asteroids)

```
#include <levels.hpp>
```

Inheritance diagram for LevelTwo:



### Static Public Member Functions

- static `BrokenShip * setLevelTwo ()`  
*Initializes Level One.*
- static void `cleanLevel ()`  
*Frees memory used on Level two.*
- static void `mainLoop (bool &playing, bool &isAlive)`
- static void `handleTimerEvents (bool &playing, BrokenShip *player, std::vector< AbstractObstacle * > &obstacles, bool &isAlive)`
- static void `handleKeyPressEvents (bool &playing, BrokenShip *player, bool &isAlive)`
- static void `handleKeyReleaseEvents ()`

### Additional Inherited Members

#### Static Protected Attributes inherited from `Level`

- static `GameObject * _player = nullptr`
- static `Background _bg`
- static `ALLEGRO_EVENT _event`

### 6.22.1 Detailed Description

Second phase of the game, vertically-oriented flappy bird with moving obstacles (asteroids)

- `_obstaclesList`: level's list of `Obstacles` objects

### 6.22.2 Member Function Documentation

#### 6.22.2.1 `cleanLevel()`

```
void LevelTwo::cleanLevel () [static]
```

*Frees memory used on Level two.*

### 6.22.2.2 handleKeyPressEvents()

```
void LevelTwo::handleKeyPressEvents (
    bool & playing,
    BrokenShip * player,
    bool & isAlive) [static]
```

### 6.22.2.3 handleKeyReleaseEvents()

```
void LevelTwo::handleKeyReleaseEvents () [static]
```

### 6.22.2.4 handleTimerEvents()

```
void LevelTwo::handleTimerEvents (
    bool & playing,
    BrokenShip * player,
    std::vector< AbstractObstacle * > & obstacles,
    bool & isAlive) [static]
```

### 6.22.2.5 mainLoop()

```
void LevelTwo::mainLoop (
    bool & playing,
    bool & isAlive) [static]
```

### 6.22.2.6 setLevelTwo()

```
BrokenShip * LevelTwo::setLevelTwo () [static]
```

Initializes [Level One](#).

#### Returns

Pointer to the player object, in this case [BrokenShip](#)

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

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

## 6.23 Line Class Reference

represents a line-shaped shot, like a laser.

### 6.23.1 Detailed Description

represents a line-shaped shot, like a laser.

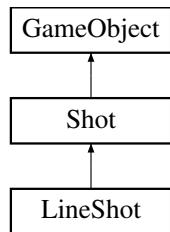
The documentation for this class was generated from the following file:

- [src/shots.cpp](#)

## 6.24 LineShot Class Reference

```
#include <shots.hpp>
```

Inheritance diagram for LineShot:



### Public Member Functions

- **LineShot (Vector initialPosi, Vector direction, float espessura, float comprimento, double tempoAtivacao)**  
*Build a shot in the form of a line, initializing the values, placing its address in the list of shots and making the variable \_direction store the end point of the half-line.*
- void **draw ()** override  
*Draw a line.*
- void **update ()** override  
*It makes the line thinner over time, and when the activation time ends, the line turns red and starts to deal damage.*
- bool **isIsActive ()** override  
*Checks if line thickness is greater than 0.4.*
- bool **shotCollidedWithBoss (WindowsBoss &boss)** override  
*We haven't implemented the line shot for the player, so it doesn't make sense to implement this function.*
- bool **shotCollidedWithPlayer (FixedShip &player)** override  
*Check if the ball (line) collided with the player. (Circle-to-line collision.)*

### Public Member Functions inherited from Shot

- **Shot (Vector position, Vector direction, ALLEGRO\_COLOR shotColor)**  
*Build a standard shot.*
- virtual **~Shot ()=default**

## Public Member Functions inherited from [GameObject](#)

- [GameObject \(\)](#)  
*Create a new default [GameObject](#).*
- [GameObject \(Vector position\)](#)  
*Create a new [GameObject](#) on given position.*
- [Vector get\\_position \(\)](#)  
*Get the position of a [GameObject](#).*
- [void set\\_position \(const Vector &position\)](#)  
*Sets the position of a [GameObject](#).*
- [virtual ~GameObject \(\)=0](#)  
*Game Object empty destructor.*
- [void set\\_bitmap \(const char \\*path\)](#)  
*Sets the sprite of the [GameObject](#).*

## Additional Inherited Members

### Static Public Member Functions inherited from [Shot](#)

- [static void updateShots \(FixedShip \\*player, WindowsBoss &boss, bool &playing\)](#)  
*For all shots, from the shot list, they are updated and then it is checked if the shot is active or if it collided with something (boss or the player)*
- [static void drawShots \(\)](#)  
*Calls the [draw\(\)](#) function of all active shots.*
- [static void cleanShots \(\)](#)  
*Destroys all shots.*

### Public Attributes inherited from [GameObject](#)

- ALLEGRO\_BITMAP \* [objectSprite](#) = NULL

### Protected Attributes inherited from [Shot](#)

- [Vector \\_direction](#)
- ALLEGRO\_COLOR [\\_shotColor](#)

### Protected Attributes inherited from [GameObject](#)

- [Vector \\_position](#)

### Static Protected Attributes inherited from [Shot](#)

- [static std::list< Shot \\* > ShotsList](#)

## 6.24.1 Constructor & Destructor Documentation

### 6.24.1.1 LineShot()

```
LineShot::LineShot (
    Vector initialPosi,
    Vector direction,
    float thickness,
    float length,
    double activationTime)
```

Build a shot in the form of a line, initializing the values, placing its address in the list of shots and making the variable *\_direction* store the end point of the half-line.

**Parameters**

<i>initialPosi</i>	Starting point of the line segment.
<i>direction</i>	<a href="#">Vector</a> that indicates the direction of the line.
<i>thickness</i>	line thickness. (Unit vector normally.)
<i>length</i>	Length of the semi-straight.
<i>activationTime</i>	Time for the shot to cause damage.

**6.24.2 Member Function Documentation****6.24.2.1 draw()**

```
void LineShot::draw () [override], [virtual]
```

Draw a line.

Implements [Shot](#).

**6.24.2.2 isItActive()**

```
bool LineShot::isItActive () [override], [virtual]
```

Checks if line thickness is greater than 0.4.

**Returns**

If the ball shot is still active.

Implements [Shot](#).

**6.24.2.3 shotCollidedWithBoss()**

```
bool LineShot::shotCollidedWithBoss (
    WindowsBoss & boss) [override], [virtual]
```

We haven't implemented the line shot for the player, so it doesn't make sense to implement this function.

**Parameters**

<i>boss</i>	Boss address(windows).
-------------	------------------------

**Returns**

If the line shot collided like the boss.

Implements [Shot](#).

**6.24.2.4 shotCollidedWithPlayer()**

```
bool LineShot::shotCollidedWithPlayer (
    FixedShip & player) [override], [virtual]
```

Check if the ball (line) collided with the player. (Circle-to-line collision.)

**Parameters**

<i>player</i>	Player address.
---------------	-----------------

**Returns**

If the line shot collided like the player.

Implements [Shot](#).

**6.24.2.5 update()**

```
void LineShot::update () [override], [virtual]
```

It makes the line thinner over time, and when the activation time ends, the line turns red and starts to deal damage.

Implements [Shot](#).

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

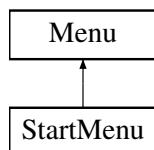
- [include/shots.hpp](#)
- [src/shots.cpp](#)

**6.25 Menu Class Reference**

base menu class

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

Inheritance diagram for Menu:

**Static Protected Attributes**

- static ALLEGRO\_EVENT *event*
- static ALLEGRO\_FONT \* *font* = nullptr
- static [Interface](#) \* *interface* = nullptr

**6.25.1 Detailed Description**

base menu class

## 6.25.2 Member Data Documentation

### 6.25.2.1 event

```
ALLEGRO_EVENT Menu::event [static], [protected]
```

### 6.25.2.2 font

```
ALLEGRO_FONT * Menu::font = nullptr [static], [protected]
```

### 6.25.2.3 interface

```
Interface * Menu::interface = nullptr [static], [protected]
```

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

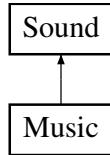
- include/menu.hpp
- src/menu.cpp

## 6.26 Music Class Reference

represents a sound in a more complex way, with pause and playback methods with fade-in and fade-out systems.

```
#include <music.hpp>
```

Inheritance diagram for Music:



### Public Member Functions

- **Music** (const char \*sound\_address, float volume\_parameter=0.6f, float fade\_speed\_parameter=2.0f)
 

*Construct a music.*
- **~Music ()** override
 

*Destroy the music. (If the destroyer of music is called, the destroyer of sound will not be called, because ALLEGRO SAMPLE INSTANCE is just a more robust ALLEGRO SAMPLE, when we destroy ALLEGRO SAMPLE INSTANCE the ALLEGRO SAMPLE is destroyed together.)*
- void **play ()**

*Play the music, starting where it left off.*
- void **pause ()**

*Pause the music, setting the volume to zero.*

## Public Member Functions inherited from Sound

- `Sound (const char *sound_address)`  
*Construct a sound.*
- `virtual ~Sound ()`  
*Destroys the sound.*
- `void play (float volume=1.0)`  
*Create and play a new sound.*

## Static Public Member Functions

- `static void update_fade_in_fade_out ()`  
*Performs fade-in and fade-out (this method must be called cyclically to work correctly)*
- `static void muteMusic ()`  
*Pauses the currently playing music. (By definition it should only play one song at a time, that's why this works.)*
- `static void unMuteMusic ()`  
*Play the last song that tried to be played.*

## Static Public Member Functions inherited from Sound

- `static void muteSounds ()`  
*Disables sounds and mutes music.*
- `static void unmuteSounds ()`  
*Enables sounds and plays music.*

## Additional Inherited Members

### Static Public Attributes inherited from Sound

- `static float volumeMester = 1.0f`

### Protected Attributes inherited from Sound

- `ALLEGRO_SAMPLE * sound_sample = nullptr`

### Static Protected Attributes inherited from Sound

- `static bool isSoundMuted = false`

## 6.26.1 Detailed Description

represents a sound in a more complex way, with pause and playback methods with fade-in and fade-out systems.

## 6.26.2 Constructor & Destructor Documentation

### 6.26.2.1 Music()

```
Music::Music (
    const char * sound_address,
    float volume_parameter = 0.6f,
    float fade_speed_parameter = 2.0f)
```

Construct a music.

### Parameters

<i>sound_address</i>	The address of the music.
<i>ballast_volume_parameter</i>	Initializes the ballast_volume of the object.
<i>fade_speed_parameter</i>	Initializes the fade_speed of the object.

### 6.26.2.2 ~Music()

```
Music::~Music () [override]
```

Destroy the music. (If the destroyer of music is called, the destroyer of sound will not be called, because ALLEGRO SAMPLE INSTANCE is just a more robust ALLEGRO SAMPLE, when we destroy ALLEGRO SAMPLE INSTANCE the ALLEGRO SAMPLE is destroyed together.)

## 6.26.3 Member Function Documentation

### 6.26.3.1 muteMusic()

```
void Music::muteMusic () [static]
```

Pauses the currently playing music. (By definition it should only play one song at a time, that's why this works.)

### 6.26.3.2 pause()

```
void Music::pause ()
```

Pause the music, setting the volume to zero.

### 6.26.3.3 play()

```
void Music::play ()
```

Play the music, starting where it left off.

### 6.26.3.4 unMuteMusic()

```
void Music::unMuteMusic () [static]
```

Play the last song that tried to be played.

### 6.26.3.5 update\_fade\_in\_fade\_out()

```
void Music::update_fade_in_fade_out () [static]
```

Performs fade-in and fade-out (this method must be called cyclically to work correctly)

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

- include/music.hpp
- src/music.cpp

## 6.27 ObstaclesList Class Reference

Gerencia uma coleção polimórfica de múltiplos obstáculos.

```
#include <obstacles_list.hpp>
```

### Public Member Functions

- void [setPolygonsObstaclesList](#) (const std::vector< [Vector](#) > &verts, const char \*path)  
*Adiciona novos obstáculos a uma lista de obstáculos poligonais.*
- void [setCircleObstaclesList](#) (const char \*path)  
*Popula a lista com novos obstáculos do tipo obstáculos circulares.*
- std::vector< [AbstractObstacle](#) \* > [getList](#) ()  
*Obtém uma referência para a lista de obstáculos.*
- void [updateAll](#) (std::vector< [AbstractObstacle](#) \* > obstaclesList)  
*Atualiza o estado de todos os obstáculos em uma lista fornecida.*
- void [drawAll](#) (std::vector< [AbstractObstacle](#) \* > obstaclesList)  
*Desenha todos os obstáculos de uma lista fornecida na tela.*
- void [clear](#) ()  
*Limpia a lista de obstáculos, liberando toda a memória alocada.*
- [~ObstaclesList](#) ()  
*Destruitor da classe [ObstaclesList](#).*

### 6.27.1 Detailed Description

Gerencia uma coleção polimórfica de múltiplos obstáculos.

Atua como uma lista para diferentes tipos de obstáculos herdados de [AbstractObstacle](#). É responsável por criar, destruir, atualizar e desenhar todos os obstáculos contidos em sua lista interna.

#### Warning

Esta classe gerencia a memória dos obstáculos. A posse dos ponteiros pertence à lista, que os deletará em seu destrutor ou no método [clear\(\)](#).

### 6.27.2 Constructor & Destructor Documentation

#### 6.27.2.1 [~ObstaclesList\(\)](#)

```
ObstaclesList::~ObstaclesList ()
```

Destruitor da classe [ObstaclesList](#).

Responsável por liberar a memória de todos os obstáculos que foram alocados dinamicamente.

### 6.27.3 Member Function Documentation

#### 6.27.3.1 clear()

```
void ObstaclesList::clear ()
```

Limpa a lista de obstáculos, liberando toda a memória alocada.

deleta cada ponteiro de obstáculo e, em seguida, limpa o vetor. É útil para reiniciar o estado da lista.

#### 6.27.3.2 drawAll()

```
void ObstaclesList::drawAll (
    std::vector< AbstractObstacle * > obstaclesList)
```

Desenha todos os obstáculos de uma lista fornecida na tela.

##### Parameters

<i>obstaclesList</i>	A lista de obstáculos a ser desenhada.
----------------------	--

#### 6.27.3.3 getList()

```
std::vector< AbstractObstacle * > & ObstaclesList::getList ()
```

Obtém uma referência para a lista de obstáculos.

##### Returns

Uma referência para o vetor *\_obstaclesList*.

#### 6.27.3.4 setCircleObstaclesList()

```
void ObstaclesList::setCircleObstaclesList (
    const char * path)
```

Popula a lista com novos obstáculos do tipo obstáculos circulares.

Limpa a lista atual e a preenche com uma quantidade determinada de novos obstáculos circulares, posicionados aleatoriamente na parte superior da tela.

##### Parameters

<i>path</i>	Caminho para o arquivo de imagem/sprite dos obstáculos.
-------------	---

#### 6.27.3.5 setPolygonsObstaclesList()

```
void ObstaclesList::setPolygonsObstaclesList (
    const std::vector< Vector > & verts,
    const char * path)
```

Adiciona novos obstáculos a uma lista de obstáculos poligonais.

Limpa a lista atual e a preenche com uma quantidade determinada de novos obstáculos poligonais, posicionados aleatoriamente na parte superior da tela.

**Parameters**

<i>verts</i>	Vetor de Pontos que define a forma do polígono do obstáculo.
<i>path</i>	Caminho para o arquivo de imagem/sprite dos obstáculos.

**6.27.3.6 updateAll()**

```
void ObstaclesList::updateAll (
    std::vector< AbstractObstacle * > obstaclesList)
```

Atualiza o estado de todos os obstáculos em uma lista fornecida.

**Parameters**

<i>obstaclesList</i>	A lista de obstáculos a ser atualizada.
----------------------	---

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

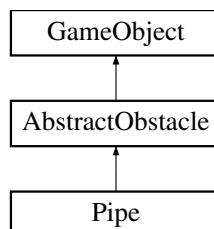
- include/obstacles\_list.hpp
- src/obstacles\_list.cpp

**6.28 Pipe Class Reference**

Representa um par de obstáculos (canos) que se movem em conjunto.

```
#include <pipe.hpp>
```

Inheritance diagram for Pipe:

**Public Member Functions**

- **Pipe** (const `Vector` &startPosition, const `std::vector< Vector >` &shapeLeft, const `std::vector< Vector >` &shapeRight, const char \*imagePathLeft, const char \*imagePathRight)
   
*Construtor da classe Pipe.*
- void **update** () override
   
*Atualiza a posição do par de canos.*
- void **draw** () override
   
*Desenha ambos os canos (esquerdo e direito) na tela.*
- bool **checkCollisionWithPlayer** (`BrokenShip` &player) override
   
*Verifica se o jogador colidiu com algum dos canos.*

## Public Member Functions inherited from [AbstractObstacle](#)

- virtual `~AbstractObstacle ()=default`

## Public Member Functions inherited from [GameObject](#)

- `GameObject ()`  
*Create a new default [GameObject](#).*
- `GameObject (Vector position)`  
*Create a new [GameObject](#) on given position.*
- `Vector get_position ()`  
*Get the position of a [GameObject](#).*
- void `set_position (const Vector &position)`  
*Sets the position of a [GameObject](#).*
- virtual `~GameObject ()=0`  
*Game Object empty destructor.*
- void `set_bitmap (const char *path)`  
*Sets the sprite of the [GameObject](#).*

## Additional Inherited Members

### Public Attributes inherited from [GameObject](#)

- ALLEGRO\_BITMAP \* `objectSprite` = NULL

### Protected Member Functions inherited from [AbstractObstacle](#)

- void `setSpeed (Vector speed)`  
*Define a velocidade do obstáculo.*
- `Vector getSpeed ()`  
*Obtém a velocidade atual do obstáculo.*

### Protected Attributes inherited from [GameObject](#)

- `Vector _position`

## 6.28.1 Detailed Description

Representa um par de obstáculos (canos) que se movem em conjunto.

Esta classe encapsula dois objetos [PolygonObstacle](#) (`leftPipe` e `rightPipe`) para criar um único obstáculo composto, como os canos em jogos no estilo Flappy Bird. Ela gerencia a lógica de atualização e desenho de ambos os canos como uma unidade, mantendo um gap fixo entre eles.

## 6.28.2 Constructor & Destructor Documentation

### 6.28.2.1 Pipe()

```
Pipe::Pipe (
    const Vector & startPosition,
    const std::vector<Vector> & shapeLeft,
    const std::vector<Vector> & shapeRight,
    const char * imagePathLeft,
    const char * imagePathRight)
```

Construtor da classe [Pipe](#).

Cria um par de obstáculos (cano esquerdo e direito) que se movem em conjunto. A posição do cano direito é calculada com base na posição inicial do esquerdo. Uma velocidade vertical de 5 unidaes para baixo é definida como padrão para o par.

#### Parameters

<i>startPosition</i>	A posição inicial do cano esquerdo.
<i>shapeLeft</i>	Os vértices que definem a forma do polígono do cano esquerdo.
<i>shapeRight</i>	Os vértices que definem a forma do polígono do cano direito.
<i>imagePathLeft</i>	O caminho para o arquivo de imagem do cano esquerdo.
<i>imagePathRight</i>	O caminho para o arquivo de imagem do cano direito.

## 6.28.3 Member Function Documentation

### 6.28.3.1 checkCollisionWithPlayer()

```
bool Pipe::checkCollisionWithPlayer (
    BrokenShip & player) [override], [virtual]
```

Verifica se o jogador colidiu com algum dos canos.

#### Parameters

<i>player</i>	Uma referência ao objeto da nave do jogador a ser verificado.
---------------	---

#### Returns

'true' se houver colisão com o cano esquerdo OU o direito, 'false' caso contrário.

Implements [AbstractObstacle](#).

### 6.28.3.2 draw()

```
void Pipe::draw () [override], [virtual]
```

Desenha ambos os canos (esquerdo e direito) na tela.

Implements [AbstractObstacle](#).

### 6.28.3.3 update()

```
void Pipe::update () [override], [virtual]
```

Atualiza a posição do par de canos.

Move o par de canos verticalmente para baixo. Se os canos ultrapassam o limite inferior da tela, eles são reposicionados no topo com uma nova posição horizontal aleatória. A lógica de reposicionamento garante que o gap entre eles seja mantido e que permaneçam dentro dos limites da tela.

Implements [AbstractObstacle](#).

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

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

## 6.29 PipeList Class Reference

```
#include <obstacles_list.hpp>
```

### Public Member Functions

- [PipeList \(\)=default](#)
- [~PipeList \(\)](#)  
*Destrutor da classe PipeList.*
- [void generatePipes \(const std::vector< Vector > &shapeLeft, const std::vector< Vector > &shapeRight, const char \\*imagePathLeft, const char \\*imagePathRight\)](#)  
*Gera e adiciona um par de canos a uma lista de pipes.*
- [std::vector< AbstractObstacle \\* > & getList \(\)](#)  
*Obtém uma referência para a lista de canos.*
- [void clear \(\)](#)  
*Limpa a lista de canos, liberando a memória.*

### 6.29.1 Constructor & Destructor Documentation

#### 6.29.1.1 PipeList()

```
PipeList::PipeList () [default]
```

#### 6.29.1.2 ~PipeList()

```
PipeList::~PipeList ()
```

Destrutor da classe PipeList.

Libera a memória de todos os objetos Canos armazenados na lista.

## 6.29.2 Member Function Documentation

### 6.29.2.1 clear()

```
void PipeList::clear ()
```

Limpa a lista de canos, liberando a memória.

### 6.29.2.2 generatePipes()

```
void PipeList::generatePipes (
    const std::vector< Vector > & shapeLeft,
    const std::vector< Vector > & shapeRight,
    const char * imagePathLeft,
    const char * imagePathRight)
```

Gera e adiciona um par de canos a uma lista de pipes.

Cria e insere dois novos objetos na lista, esses dois preimeiros canos, possuem posições pré-definidas, para formar os obstáculos do jogo.

#### Parameters

<i>shapeLeft</i>	O vetor de Pontos do cano esquerdo.
<i>shapeRight</i>	O vetor de Pontos do cano direito.
<i>imagePathLeft</i>	O caminho da imagem para o cano esquerdo.
<i>imagePathRight</i>	O caminho da imagem para o cano direito.

### 6.29.2.3 getList()

```
std::vector< AbstractObstacle * > & PipeList::getList ()
```

Obtém uma referência para a lista de canos.

#### Returns

Uma referência para o vetor pipes contendo ponteiros [AbstractObstacle](#), permitindo polimorfismo.

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

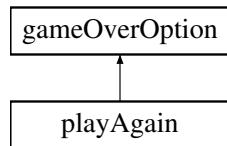
- include/[obstacles\\_list.hpp](#)
- src/[obstacles\\_list.cpp](#)

## 6.30 playAgain Class Reference

Ação para reiniciar o jogo.

```
#include <game_over.hpp>
```

Inheritance diagram for playAgain:



### Public Member Functions

- `~playAgain () override=default`
- `void execute () override`

*Função virtual pura que será sobrescrita pelas classes derivadas.*

### Public Member Functions inherited from [gameOverOption](#)

- `virtual ~gameOverOption ()=default`

*Destruitor virtual padrão para que a memória seja liberada corretamente nas classes derivadas.*

### 6.30.1 Detailed Description

Ação para reiniciar o jogo.

### 6.30.2 Constructor & Destructor Documentation

#### 6.30.2.1 `~playAgain()`

```
playAgain::~playAgain () [override], [default]
```

### 6.30.3 Member Function Documentation

#### 6.30.3.1 `execute()`

```
void playAgain::execute () [override], [virtual]
```

Função virtual pura que será sobrescrita pelas classes derivadas.

Implements [gameOverOption](#).

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

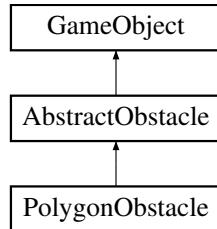
- `include/game_over.hpp`
- `src/gameover.cpp`

## 6.31 PolygonObstacle Class Reference

Represents a obstacle with a customizable polygonal collision shape.

```
#include <polygon_obstacle.hpp>
```

Inheritance diagram for PolygonObstacle:



### Public Member Functions

- `PolygonObstacle (const Vector &pos, const std::vector< Vector > &verts, float scale, const char *imagePath)`  
*Construtor da classe `PolygonObstacle`.*
- `void draw () override`  
*Desenha o obstáculo na tela.*
- `std::vector< Vector > getVertices ()`  
*Obtém os vértices que definem a forma do polígono.*
- `void update () override`  
*Atualiza a posição do obstáculo.*
- `bool checkCollisionWithPlayer (BrokenShip &player) override`  
*Verifica a colisão entre o obstáculo poligonal e a nave do jogador.*

### Public Member Functions inherited from `AbstractObstacle`

- `virtual ~AbstractObstacle ()=default`

### Public Member Functions inherited from `GameObject`

- `GameObject ()`  
*Create a new default `GameObject`.*
- `GameObject (Vector position)`  
*Create a new `GameObject` on given position.*
- `Vector get_position ()`  
*Get the position of a `GameObject`.*
- `void set_position (const Vector &position)`  
*Sets the position of a `GameObject`.*
- `virtual ~GameObject ()=0`  
*Game Object empty destructor.*
- `void set_bitmap (const char *path)`  
*Sets the sprite of the `GameObject`.*

## Additional Inherited Members

### Public Attributes inherited from [GameObject](#)

- ALLEGRO\_BITMAP \* [objectSprite](#) = NULL

### Protected Member Functions inherited from [AbstractObstacle](#)

- void [setSpeed](#) ([Vector](#) speed)  
*Define a velocidade do obstáculo.*
- [Vector getSpeed](#) ()  
*Obtém a velocidade atual do obstáculo.*

### Protected Attributes inherited from [GameObject](#)

- [Vector \\_position](#)

## 6.31.1 Detailed Description

Representa um obstáculo com uma forma de colisão poligonal customizável.

Esta classe herda de [AbstractObstacle](#) e é usada para criar obstáculos cujos limites de colisão são definidos por um conjunto de vértices. Foi utilizado para formas complexas como asteroides ou os pipes.

## 6.31.2 Constructor & Destructor Documentation

### 6.31.2.1 [PolygonObstacle\(\)](#)

```
PolygonObstacle::PolygonObstacle (
    const Vector & pos,
    const std::vector< Vector > & verts,
    float scale,
    const char * imagePath) [inline]
```

Construtor da classe [PolygonObstacle](#).

Inicializa o obstáculo, definindo sua posição, forma, escala e imagem. A escala fornecida é aplicada a cada um dos vértices no momento da construção.

#### Parameters

<i>pos</i>	A posição inicial do obstáculo.
<i>verts</i>	Um vetor de Vetores definindo a forma do polígono antes da escala.
<i>scale</i>	O fator de escala a ser aplicado nos vértices e no sprite.
<i>imagePath</i>	O caminho para o arquivo de imagem (sprite) do obstáculo.

### 6.31.3 Member Function Documentation

#### 6.31.3.1 checkCollisionWithPlayer()

```
bool PolygonObstacle::checkCollisionWithPlayer (
    BrokenShip & player) [override], [virtual]
```

Verifica a colisão entre o obstáculo poligonal e a nave do jogador.

Este método implementa a detecção de colisão entre um polígono e um círculo. Ele itera sobre cada aresta do polígono e calcula a menor distância do centro da nave até essa aresta. Se a distância for menor ou igual ao raio da nave para qualquer uma das arestas, considera-se que houve colisão.

**Parameters**

<i>player</i>	Uma referência ao objeto da nave do jogador, que é tratado como um círculo.
---------------	---

**Returns**

'true' se houver colisão, 'false' caso contrário.

Implements [AbstractObstacle](#).

### 6.31.3.2 draw()

```
void PolygonObstacle::draw () [override], [virtual]
```

Desenha o obstáculo na tela.

Renderiza o sprite associado ao obstáculo. O sprite é desenhado de forma centralizada na posição do obstáculo e sua escala é ajustada de acordo com o membro `_scale`, para evitar que a imagem fique muito grande.

Implements [AbstractObstacle](#).

### 6.31.3.3 getVertices()

```
std::vector< Vector > PolygonObstacle::getVertices ()
```

Obtém os vértices que definem a forma do polígono.

**Returns**

Um vetor de Vetores(coordenadas X e Y) contendo os vértices do polígono.

### 6.31.3.4 update()

```
void PolygonObstacle::update () [override], [virtual]
```

Atualiza a posição do obstáculo.

Move o obstáculo verticalmente para baixo, de acordo com sua velocidade. Se o obstáculo ultrapassar o limite inferior da tela, ele é reposicionado no topo com uma nova coordenada X aleatória.

Implements [AbstractObstacle](#).

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

- [include/polygon\\_obstacle.hpp](#)
- [src/polygon\\_obstacle.cpp](#)

## 6.32 RegisterInterface Class Reference

Gerencia a interface gráfica e a lógica para registro e login de usuários.

```
#include <register_interface.hpp>
```

### Public Member Functions

- `RegisterInterface (ALLEGRO_FONT *font)`  
*Construtor da classe RegisterInterface.*
- `~RegisterInterface ()`  
*Destrutor da classe RegisterInterface.*
- `void draw ()`  
*Desenha todos os elementos da interface na tela.*
- `void handleKeyInput (int keycode, unsigned char unicode)`  
*Processa a entrada de teclado para os campos de texto.*
- `void handleMouseClick (int mx, int my)`  
*Gerencia cliques do mouse para ativar os campos de texto.*
- `std::string getUsername () const`  
*Obtém o nome de usuário inserido.*
- `std::string getName () const`  
*Obtém o nome inserido.*
- `std::string getPassword () const`  
*Obtém a senha inserida.*
- `void resetFields ()`  
*Limpa todos os campos de texto e redefine o estado de digitação.*
- `int mainLoop (bool &inRegister, bool &playing, DatabaseUsers &db)`  
*Executa o loop principal da interface de registro e login.*

### 6.32.1 Detailed Description

Gerencia a interface gráfica e a lógica para registro e login de usuários.

Esta classe encapsula todos os elementos da interface (campos de texto, botões), processa a entrada do usuário (teclado e mouse) e interage com o banco de dados para autenticar ou criar novos usuários.

### 6.32.2 Constructor & Destructor Documentation

#### 6.32.2.1 RegisterInterface()

```
RegisterInterface::RegisterInterface (
    ALLEGRO_FONT * font)
```

Construtor da classe RegisterInterface.

Inicializa a interface de registro e login, definindo a fonte, as áreas dos campos de texto (nome, username, senha) e os botões de interação (Login, Register, Sair).

**Parameters**

<i>font</i>	Um ponteiro para o ALLEGRO_FONT que será usado para renderizar os textos na interface.
-------------	--

**6.32.2.2 ~RegisterInterface()**

```
RegisterInterface::~RegisterInterface ()
```

Destruitor da classe [RegisterInterface](#).

Libera a memoria alocada para o BitMap da Imagem

**6.32.3 Member Function Documentation****6.32.3.1 draw()**

```
void RegisterInterface::draw ()
```

Desenha todos os elementos da interface na tela.

Renderiza as caixas de texto para nome, username e senha, e cada botão da interface.

**6.32.3.2 getName()**

```
std::string RegisterInterface::getName () const
```

Obtém o nome inserido.

**Returns**

A string do nome.

**6.32.3.3 getPassword()**

```
std::string RegisterInterface::getPassword () const
```

Obtém a senha inserida.

**Returns**

A string da senha.

#### 6.32.3.4 getUsername()

```
std::string RegisterInterface::getUsername () const
```

Obtém o nome de usuário inserido.

##### Returns

A string do nome de usuário.

#### 6.32.3.5 handleKeyInput()

```
void RegisterInterface::handleKeyInput (
    int keycode,
    unsigned char unicode)
```

Processa a entrada de teclado para os campos de texto.

Verifica qual campo de texto está ativo e modifica a string correspondente. Suporta a adição de caracteres ASCII imprimíveis e a remoção com a tecla Backspace.

**Parameters**

<i>keycode</i>	O código da tecla pressionada, usado para detectar teclas especiais como Backspace.
<i>unicode</i>	O caractere unicode correspondente à tecla, usado para adicionar texto.

**6.32.3.6 handleMouseClick()**

```
void RegisterInterface::handleMouseClick (
    int mx,
    int my)
```

Gerencia cliques do mouse para ativar os campos de texto.

**Parameters**

<i>mx</i>	A coordenada X do clique do mouse.
<i>my</i>	A coordenada Y do clique do mouse.

**6.32.3.7 mainLoop()**

```
int RegisterInterface::mainLoop (
    bool & inRegister,
    bool & playing,
    DatabaseUsers & db)
```

Executa o loop principal da interface de registro e login.

Gerencia o fluxo de eventos da tela, como entrada de teclado, cliques de mouse e fechamento da janela. Além disso, interage com a classe de banco de dados para logar ou registrar usuários. Ademais, controla a lógica de redesenho da tela e a animação do plano de fundo.

**Parameters**

<i>inRegister</i>	controla a permanência neste loop. É setado para false ao logar o usuário.
<i>playing</i>	Referência a um booleano que controla o loop principal do jogo. Ou seja, é false se o usuário fechar o jogo
<i>db</i>	Referência ao objeto <a href="#">DatabaseUsers</a> para realizar operações de banco de dados.

**Returns**

Retorna 0 na conclusão normal ou ao fechar a janela.

**6.32.3.8 resetFields()**

```
void RegisterInterface::resetFields ()
```

Limpa todos os campos de texto e redefine o estado de digitação.

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

- [include/register\\_interface.hpp](#)
- [src/register\\_interface.cpp](#)

## 6.33 Represents Class Reference

### 6.33.1 Detailed Description

states that the boss, windows, can have.

types of attacks the boss can have.

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

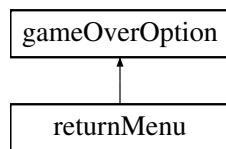
- [include/boss\\_states.hpp](#)

## 6.34 returnMenu Class Reference

Ação para retornar ao menu principal do jogo.

```
#include <game_over.hpp>
```

Inheritance diagram for returnMenu:



### Public Member Functions

- [~returnMenu \(\) override=default](#)
- [void execute \(\) override](#)

*Função virtual pura que será sobreescrita pelas classes derivadas.*

### Public Member Functions inherited from [gameOverOption](#)

- [virtual ~gameOverOption \(\)=default](#)

*Destruitor virtual padrão para que a memória seja liberada corretamente nas classes derivadas.*

### 6.34.1 Detailed Description

Ação para retornar ao menu principal do jogo.

### 6.34.2 Constructor & Destructor Documentation

#### 6.34.2.1 [~returnMenu\(\)](#)

```
returnMenu::~returnMenu () [override], [default]
```

### 6.34.3 Member Function Documentation

#### 6.34.3.1 execute()

```
void returnMenu::execute () [override], [virtual]
```

Função virtual pura que será sobreescrita pelas classes derivadas.

Implements [gameOverOption](#).

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

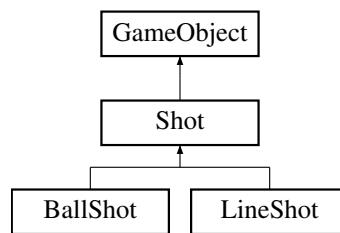
- [include/game\\_over.hpp](#)
- [src/gameover.cpp](#)

## 6.35 Shot Class Reference

represents a shot in a more abstract way, abstract class.

```
#include <shots.hpp>
```

Inheritance diagram for Shot:



### Public Member Functions

- [Shot \(Vector position, Vector direction, ALLEGRO\\_COLOR shotColor\)](#)  
*Build a standard shot.*
- [virtual ~Shot \(\)=default](#)
- [virtual bool isActive \(\)=0](#)
- [virtual bool shotCollidedWithPlayer \(FixedShip &player\)=0](#)
- [virtual bool shotCollidedWithBoss \(WindowsBoss &boss\)=0](#)
- [virtual void draw \(\)=0](#)
- [virtual void update \(\)=0](#)

### Public Member Functions inherited from [GameObject](#)

- [GameObject \(\)](#)  
*Create a new default [GameObject](#).*
- [GameObject \(Vector position\)](#)  
*Create a new [GameObject](#) on given position.*
- [Vector get\\_position \(\)](#)  
*Get the position of a [GameObject](#).*
- [void set\\_position \(const Vector &position\)](#)  
*Sets the position of a [GameObject](#).*
- [virtual ~GameObject \(\)=0](#)  
*Game Object empty destructor.*
- [void set\\_bitmap \(const char \\*path\)](#)  
*Sets the sprite of the [GameObject](#).*

## Static Public Member Functions

- static void `updateShots` (`FixedShip` \*player, `WindowsBoss` &boss, `bool` &playing)
 

*For all shots, from the shot list, they are updated and then it is checked if the shot is active or if it collided with something (boss or the player)*
- static void `drawShots` ()
 

*Calls the `draw()` function of all active shots.*
- static void `cleanShots` ()
 

*Destroys all shots.*

## Protected Attributes

- `Vector _direction`
- `ALLEGRO_COLOR _shotColor`

## Protected Attributes inherited from `GameObject`

- `Vector _position`

## Static Protected Attributes

- static `std::list< Shot * > ShotsList`

## Additional Inherited Members

## Public Attributes inherited from `GameObject`

- `ALLEGRO_BITMAP * objectSprite` = NULL

### 6.35.1 Detailed Description

represents a shot in a more abstract way, abstract class.

### 6.35.2 Constructor & Destructor Documentation

#### 6.35.2.1 `Shot()`

```
Shot::Shot (
    Vector position,
    Vector direction,
    ALLEGRO_COLOR shotColor)
```

Build a standard shot.

**Parameters**

<i>position</i>	The initial position of the shot.
<i>direction</i>	A vector in which the shot moves or points.
<i>shotColor</i>	A shot color.

**6.35.2.2 ~Shot()**

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

**6.35.3 Member Function Documentation****6.35.3.1 cleanShots()**

```
void Shot::cleanShots () [static]
```

Destroys all shots.

**6.35.3.2 draw()**

```
virtual void Shot::draw () [pure virtual]
```

Implemented in [BallShot](#), and [LineShot](#).

**6.35.3.3 drawShots()**

```
void Shot::drawShots () [static]
```

Calls the [draw\(\)](#) function of all active shots.

**6.35.3.4 isItActive()**

```
virtual bool Shot::isItActive () [pure virtual]
```

Implemented in [BallShot](#), and [LineShot](#).

**6.35.3.5 shotCollidedWithBoss()**

```
virtual bool Shot::shotCollidedWithBoss (
    WindowsBoss & boss) [pure virtual]
```

Implemented in [BallShot](#), and [LineShot](#).

### 6.35.3.6 shotCollidedWithPlayer()

```
virtual bool Shot::shotCollidedWithPlayer (
    FixedShip & player) [pure virtual]
```

Implemented in [BallShot](#), and [LineShot](#).

### 6.35.3.7 update()

```
virtual void Shot::update () [pure virtual]
```

Implemented in [BallShot](#), and [LineShot](#).

### 6.35.3.8 updateShots()

```
void Shot::updateShots (
    FixedShip * player,
    WindowsBoss & boss,
    bool & playing) [static]
```

For all shots, from the shot list, they are updated and then it is checked if the shot is active or if it collided with something (boss or the player)

#### Parameters

<i>player</i>	Player pointer.
<i>boss</i>	Boss address(windows).
<i>playing</i>	Address of the variable that indicates whether the game is still active.

## 6.35.4 Member Data Documentation

### 6.35.4.1 \_direction

```
Vector Shot::_direction [protected]
```

### 6.35.4.2 \_shotColor

```
ALLEGRO_COLOR Shot::_shotColor [protected]
```

### 6.35.4.3 ShotsList

```
std::list< Shot * > Shot::ShotsList [static], [protected]
```

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

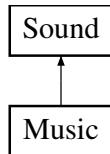
- [include/shots.hpp](#)
- [src/shots.cpp](#)

## 6.36 Sound Class Reference

class that represents a simple sound, with a musical object, ALLEGRO\_SAMPLE, and a play method.

```
#include <sound.hpp>
```

Inheritance diagram for Sound:



### Public Member Functions

- `Sound (const char *sound_address)`  
*Construct a sound.*
- `virtual ~Sound ()`  
*Destroys the sound.*
- `void play (float volume=1.0)`  
*Create and play a new sound.*

### Static Public Member Functions

- `static void muteSounds ()`  
*Disables sounds and mutes music.*
- `static void unmuteSounds ()`  
*Enables sounds and plays music.*

### Static Public Attributes

- `static float volumeMester = 1.0f`

### Protected Attributes

- `ALLEGRO_SAMPLE * sound_sample = nullptr`

### Static Protected Attributes

- `static bool isSoundMuted = false`

## 6.36.1 Detailed Description

class that represents a simple sound, with a musical object, ALLEGRO\_SAMPLE, and a play method.

## 6.36.2 Constructor & Destructor Documentation

### 6.36.2.1 Sound()

```
Sound::Sound (
    const char * sound_address)
```

Construct a sound.

**Parameters**

<code>sound_address</code>	The address of the sound.
----------------------------	---------------------------

**6.36.2.2 ~Sound()**

```
Sound::~Sound () [virtual]
```

Destroys the sound.

**6.36.3 Member Function Documentation****6.36.3.1 muteSounds()**

```
void Sound::muteSounds () [static]
```

Disables sounds and mutes music.

**6.36.3.2 play()**

```
void Sound::play (
    float volume = 1.0)
```

Create and play a new sound.

**Parameters**

<code>volume</code>	Volume at which the new sound will play.
---------------------	--

**6.36.3.3 unmuteSounds()**

```
void Sound::unmuteSounds () [static]
```

Enables sounds and plays music.

**6.36.4 Member Data Documentation****6.36.4.1 isSoundMuted**

```
bool Sound::isSoundMuted = false [static], [protected]
```

**6.36.4.2 sound\_sample**

```
ALLEGRO_SAMPLE* Sound::sound_sample = nullptr [protected]
```

### 6.36.4.3 volumeMester

```
float Sound::volumeMester = 1.0f [static]
```

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

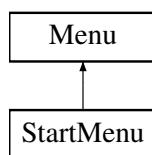
- include/sound.hpp
- src/sound.cpp

## 6.37 StartMenu Class Reference

main menu that inherits from the base menu class

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

Inheritance diagram for StartMenu:



### Static Public Member Functions

- static void [mainLoopMenu](#) (bool &playing)  
*starts needed variables and handles needed events*

### Additional Inherited Members

#### Static Protected Attributes inherited from [Menu](#)

- static ALLEGRO\_EVENT [event](#)
- static ALLEGRO\_FONT \* [font](#) = nullptr
- static [Interface](#) \* [interface](#) = nullptr

### 6.37.1 Detailed Description

main menu that inherits from the base menu class

### 6.37.2 Member Function Documentation

#### 6.37.2.1 [mainLoopMenu\(\)](#)

```
void StartMenu::mainLoopMenu (
    bool & playing) [static]
```

starts needed variables and handles needed events

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

- include/menu.hpp
- src/menu.cpp

## 6.38 User Struct Reference

Estrutura simples para armazenar os dados de um usuário.

```
#include <database_users.hpp>
```

### Public Attributes

- int `id`
- std::string `username`
- int `score`
- std::string `name`
- int `games`

### 6.38.1 Detailed Description

Estrutura simples para armazenar os dados de um usuário.

### 6.38.2 Member Data Documentation

#### 6.38.2.1 `games`

```
int User::games
```

#### 6.38.2.2 `id`

```
int User::id
```

#### 6.38.2.3 `name`

```
std::string User::name
```

#### 6.38.2.4 `score`

```
int User::score
```

#### 6.38.2.5 `username`

```
std::string User::username
```

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

- [include/database\\_users.hpp](#)

## 6.39 Vector Class Reference

Implements 2D vectors that represent cartesian coordinates.

```
#include <movement.hpp>
```

### Public Member Functions

- `Vector ()`  
*Creates a new default vector.*
- `Vector (float _xy)`  
*Creates a new vector based on one value.*
- `Vector (float _x, float _y)`  
*Creates a new vector based on x and y values.*
- `Vector operator+ (const Vector &other) const`  
*Implements vector sum using operator overload.*
- `Vector operator- (const Vector &other) const`  
*Implements vector subtraction using operator overload.*
- `Vector operator* (float value) const`  
*Implements vector multiplication by scalar using operator overload.*

### Static Public Member Functions

- `static float dot (const Vector &a, const Vector &b)`  
*Calcula o produto escalar entre dois vetores.*
- `static float distance (const Vector &a, const Vector &b)`  
*Calcula a distância entre dois pontos (representados como vetores).*
- `static float shortestDistancePointToSegment (const Vector &p, const Vector &a, const Vector &b)`  
*Calcula a menor distância de um ponto a um segmento de reta.*

### Public Attributes

- `float _x`
- `float _y`

### 6.39.1 Detailed Description

Implements 2D vectors that represent cartesian coordinates.

### 6.39.2 Constructor & Destructor Documentation

#### 6.39.2.1 `Vector()` [1/3]

```
Vector::Vector ()
```

Creates a new default vector.

### 6.39.2.2 Vector() [2/3]

```
Vector::Vector (
    float _xy)
```

Creates a new vector based on one value.

### 6.39.2.3 Vector() [3/3]

```
Vector::Vector (
    float _x,
    float _y)
```

Creates a new vector based on x and y values.

## 6.39.3 Member Function Documentation

### 6.39.3.1 distance()

```
float Vector::distance (
    const Vector & a,
    const Vector & b) [static]
```

Calcula a distância entre dois pontos (representados como vetores).

Mede o comprimento do segmento de reta que conecta os dois pontos. A implementação calcula o vetor diferença  $d = a - b$  e então retorna seu módulo, que é a raiz quadrada do produto escalar de  $d$  por ele mesmo.

#### Parameters

<i>a</i>	O primeiro vetor.
<i>b</i>	O segundo vetor.

#### Returns

A distância em linha reta entre os pontos a e b.

### 6.39.3.2 dot()

```
float Vector::dot (
    const Vector & a,
    const Vector & b) [static]
```

Calcula o produto escalar entre dois vetores.

#### Parameters

<i>a</i>	O primeiro vetor.
<i>b</i>	O segundo vetor.

#### Returns

O valor do produto escalar como um número de ponto flutuante.

### 6.39.3.3 operator\*()

```
Vector Vector::operator* (
    float value) const
```

Implements vector multiplication by scalar using operator overload.

### 6.39.3.4 operator+()

```
Vector Vector::operator+ (
    const Vector & other) const
```

Implements vector sum using operator overload.

### 6.39.3.5 operator-()

```
Vector Vector::operator- (
    const Vector & other) const
```

Implements vector subtraction using operator overload.

### 6.39.3.6 shortestDistancePointToSegment()

```
float Vector::shortestDistancePointToSegment (
    const Vector & p,
    const Vector & a,
    const Vector & b) [static]
```

Calcula a menor distância de um ponto a um segmento de reta.

Este método encontra o ponto em um segmento de reta (definido por a e b) que está mais próximo do ponto p e retorna a distância entre eles. A lógica consiste em projetar o vetor ap sobre o vetor ab e obter um triângulo retângulo.

#### Parameters

<i>p</i>	O ponto do qual se deseja calcular a distância.
<i>a</i>	O ponto inicial do segmento de reta.
<i>b</i>	O ponto final do segmento de reta.

#### Returns

A menor distância entre o ponto p e o segmento ab.

## 6.39.4 Member Data Documentation

### 6.39.4.1 \_x

```
float Vector::_x
```

### 6.39.4.2 `_y`

```
float Vector::_y
```

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

- include/movement.hpp
- src/movement.cpp

## 6.40 victoryInterface Class Reference

interface to the "victory screen" after defeating the final boss (windows)

```
#include <interface.hpp>
```

### Public Member Functions

- `victoryInterface (ALLEGRO_FONT *font)`  
*victoryInterface constructor*
- void `drawVictoryScreen ()`  
*draws the victory screen*

### 6.40.1 Detailed Description

interface to the "victory screen" after defeating the final boss (windows)

### 6.40.2 Constructor & Destructor Documentation

#### 6.40.2.1 `victoryInterface()`

```
victoryInterface::victoryInterface (
```

	ALLEGRO_FONT * font)
--	----------------------

`victoryInterface` constructor

##### Parameters

<code>font</code>	pointer to an ALLEGRO_FONT variable for the text to be drawn
-------------------	--

### 6.40.3 Member Function Documentation

#### 6.40.3.1 `drawVictoryScreen()`

```
void victoryInterface::drawVictoryScreen ()
```

`draws the victory screen`

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

- include/interface.hpp
- src/interface.cpp

## 6.41 Windows Class Reference

represents the game's boss, [Windows](#).

### 6.41.1 Detailed Description

represents the game's boss, [Windows](#).

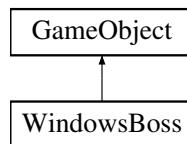
The documentation for this class was generated from the following file:

- src/[windows\\_boss.cpp](#)

## 6.42 WindowsBoss Class Reference

```
#include <windows_boss.hpp>
```

Inheritance diagram for WindowsBoss:



### Public Member Functions

- [WindowsBoss](#) (float halfSide, float life)  
*Build a windows boss and define the variables.*
- void [draw](#) ()  
*Draw the boss, first he draws the big square and then the 4 smaller ones inside and they are all made from the center point of the square.*
- void [update](#) (FixedShip \*player, bool &playing)  
*Implements the boss states and checks if the player collided with it. The boss's states are, Descending, when he enters the screen, beginning of the stage, Attacking, main mode where he switches between different types of attack, Rising, when the boss's life is zero.*
- float [getHalfSide](#) ()  
*Returns the measurement of half the side of the square.*
- bool [isDead](#) ()
- void [takeDamage](#) (FixedShip \*player)  
*Applies damage to the boss.*

## Public Member Functions inherited from [GameObject](#)

- [GameObject \(\)](#)  
*Create a new default [GameObject](#).*
- [GameObject \(Vector position\)](#)  
*Create a new [GameObject](#) on given position.*
- [Vector get\\_position \(\)](#)  
*Get the position of a [GameObject](#).*
- [void set\\_position \(const Vector &position\)](#)  
*Sets the position of a [GameObject](#).*
- [virtual ~GameObject \(\)=0](#)  
*Game Object empty destructor.*
- [void set\\_bitmap \(const char \\*path\)](#)  
*Sets the sprite of the [GameObject](#).*

## Additional Inherited Members

### Public Attributes inherited from [GameObject](#)

- [ALLEGRO\\_BITMAP \\* objectSprite](#) = NULL

### Protected Attributes inherited from [GameObject](#)

- [Vector \\_position](#)

## 6.42.1 Constructor & Destructor Documentation

### 6.42.1.1 WindowsBoss()

```
WindowsBoss::WindowsBoss (
    float halfSide,
    float life)
```

Build a windows boss and define the variables.

## 6.42.2 Member Function Documentation

### 6.42.2.1 draw()

```
void WindowsBoss::draw ()
```

Draw the boss, first he draws the big square and then the 4 smaller ones inside and they are all made from the center point of the square.

### 6.42.2.2 getHalfSide()

```
float WindowsBoss::getHalfSide ()
```

Returns the measurement of half the side of the square.

#### Returns

Return half side of square.

### 6.42.2.3 isDead()

```
bool WindowsBoss::isDead ()
```

### 6.42.2.4 takeDamage()

```
void WindowsBoss::takeDamage (
    FixedShip * player)
```

Applies damage to the boss.

#### Parameters

<i>player</i>	The player's address, to disable player damage when necessary.
---------------	--

### 6.42.2.5 update()

```
void WindowsBoss::update (
    FixedShip * player,
    bool & playing)
```

Implements the boss states and checks if the player collided with it. The boss's states are, Descending, when he enters the screen, beginning of the stage, Attacking, main mode where he switches between different types of attack, Rising, when the boss's life is zero.

#### Parameters

<i>player</i>	Player memory address.
<i>playing</i>	Variable that controls whether the game is active or not.

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

- include/windows\_boss.hpp
- src/windows\_boss.cpp



# Chapter 7

## File Documentation

### 7.1 include/abstract\_obstacle.hpp File Reference

```
#include "game_object.hpp"
```

#### Classes

- class [AbstractObstacle](#)  
*Classe base abstrata para todos os tipos de obstáculos no jogo.*

### 7.2 abstract\_obstacle.hpp

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

```
00001 #pragma once
00002 #include "game_object.hpp"
00003
00012
00013 class AbstractObstacle : public GameObject {
00014     private:
00015         Vector _speed;
00016     protected:
00022     void setSpeed(Vector speed);
00023
00028     Vector getSpeed();
00029 public:
00030     virtual void draw() = 0;
00031     virtual void update() = 0;
00032     virtual bool checkCollisionWithPlayer(BrokenShip& player) = 0;
00033     virtual ~AbstractObstacle() = default;
00034
00035 };
```

## 7.3 include/bootstrap.hpp File Reference

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_primitives.h>
#include <allegro5/allegro_ttf.h>
#include <allegro5/mouse.h>
#include <allegro5/events.h>
#include <allegro5/timer.h>
#include <allegro5/color.h>
#include <allegro5/allegro_image.h>
#include "sound.hpp"
#include "music.hpp"
```

### Classes

- class [Bootstrap](#)  
*Static class that encapsulates initialization and cleanup of allegro components.*

### Namespaces

- namespace [globalVars](#)

### Variables

- constexpr float [FPS](#) = 30
- constexpr int [SCREEN\\_W](#) = 800
- constexpr int [SCREEN\\_H](#) = 600
- constexpr int [BUTTON\\_W](#) = 400
- constexpr int [BUTTON\\_H](#) = 60
- constexpr int [OBSTACLES\\_LIST\\_NUM](#) = 4
- constexpr float [SCALE\\_PIPES](#) = 0.33f
- constexpr float [SCALE\\_ASTEROID](#) = 0.19f
- const float [velocity](#) [] = {9.0f, 9.3f, 9.7f, 10.2f, 10.8f, 11.5f, 12.0f, 12.8f, 13.5f}
- const int [TAM\\_VECTOR\\_VELOCITY](#) = sizeof([velocity](#))/sizeof([velocity](#)[0])
- const ALLEGRO\_COLOR [BACKGROUND\\_COLOR](#)
- ALLEGRO\_DISPLAY \* [display](#)
- ALLEGRO\_EVENT\_QUEUE \* [event\\_queue](#)
- ALLEGRO\_TIMER \* [timer](#)
- ALLEGRO\_FONT \* [gameFont](#)
- ALLEGRO\_FONT \* [levelFont](#)
- ALLEGRO\_BITMAP \* [gameOverBackground](#)
- ALLEGRO\_BITMAP \* [pinguinBandido](#)
- ALLEGRO\_BITMAP \* [backgroundImage](#)
- ALLEGRO\_BITMAP \* [pendrive](#)
- ALLEGRO\_BITMAP \* [ballShotSprite](#)
- Sound \* [death\\_sound](#)
- Sound \* [gunshot\\_sound1](#)
- Sound \* [gunshot\\_sound2](#)
- Sound \* [gunshot\\_sound3](#)
- Sound \* [gunshot\\_sound4](#)

- `Music * menu_music`
- `Music * pause_game_music`
- `Music * level_one_music`
- `Music * level_two_music`
- `Music * level_three_music`
- `Music * defeat_music`
- `Music * victory_music`

### 7.3.1 Variable Documentation

#### 7.3.1.1 BACKGROUND\_COLOR

```
const ALLEGRO_COLOR BACKGROUND_COLOR [extern]
```

#### 7.3.1.2 backgroundImage

```
ALLEGRO_BITMAP* backgroundImage [extern]
```

#### 7.3.1.3 ballShotSprite

```
ALLEGRO_BITMAP* ballShotSprite [extern]
```

#### 7.3.1.4 BUTTON\_H

```
int BUTTON_H = 60 [constexpr]
```

#### 7.3.1.5 BUTTON\_W

```
int BUTTON_W = 400 [constexpr]
```

#### 7.3.1.6 death\_sound

```
Sound* death_sound [extern]
```

#### 7.3.1.7 defeat\_music

```
Music* defeat_music [extern]
```

#### 7.3.1.8 display

```
ALLEGRO_DISPLAY* display [extern]
```

### 7.3.1.9 event\_queue

```
ALLEGRO_EVENT_QUEUE* event_queue [extern]
```

### 7.3.1.10 FPS

```
float FPS = 30 [constexpr]
```

### 7.3.1.11 gameFont

```
ALLEGRO_FONT* gameFont [extern]
```

### 7.3.1.12 gameOverBackground

```
ALLEGRO_BITMAP* gameOverBackground [extern]
```

### 7.3.1.13 gunshot\_sound1

```
Sound* gunshot_sound1 [extern]
```

### 7.3.1.14 gunshot\_sound2

```
Sound* gunshot_sound2 [extern]
```

### 7.3.1.15 gunshot\_sound3

```
Sound* gunshot_sound3 [extern]
```

### 7.3.1.16 gunshot\_sound4

```
Sound* gunshot_sound4 [extern]
```

### 7.3.1.17 level\_one\_music

```
Music* level_one_music [extern]
```

### 7.3.1.18 level\_three\_music

```
Music* level_three_music [extern]
```

**7.3.1.19 level\_two\_music**

```
Music* level_two_music [extern]
```

**7.3.1.20 levelFont**

```
ALLEGRO_FONT* levelFont [extern]
```

**7.3.1.21 menu\_music**

```
Music* menu_music [extern]
```

**7.3.1.22 OBSTACLES\_LIST\_NUM**

```
int OBSTACLES_LIST_NUM = 4 [constexpr]
```

**7.3.1.23 pause\_game\_music**

```
Music* pause_game_music [extern]
```

**7.3.1.24 pendrive**

```
ALLEGRO_BITMAP* pendrive [extern]
```

**7.3.1.25 pinguimBandido**

```
ALLEGRO_BITMAP* pinguimBandido [extern]
```

**7.3.1.26 SCALE\_ASTEROID**

```
float SCALE_ASTEROID = 0.19f [constexpr]
```

**7.3.1.27 SCALE\_PIPES**

```
float SCALE_PIPES = 0.33f [constexpr]
```

**7.3.1.28 SCREEN\_H**

```
int SCREEN_H = 600 [constexpr]
```

### 7.3.1.29 SCREEN\_W

```
int SCREEN_W = 800 [constexpr]
```

### 7.3.1.30 TAM\_VECTOR\_VELOCITY

```
const int TAM_VECTOR_VELOCITY = sizeof(velocity)/sizeof(velocity[0])
```

### 7.3.1.31 timer

```
ALLEGRO_TIMER* timer [extern]
```

### 7.3.1.32 velocity

```
const float velocity[] = {9.0f, 9.3f, 9.7f, 10.2f, 10.8f, 11.5f, 12.0f, 12.8f, 13.5f}
```

### 7.3.1.33 victory\_music

```
Music* victory_music [extern]
```

## 7.4 bootstrap.hpp

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

```
00001 #ifndef BOOTSTRAP_HPP
00002 #define BOOTSTRAP_HPP
00003
00004 #include <allegro5/allegro.h>
00005 #include <allegro5/allegro_font.h>
00006 #include <allegro5/allegro_primitives.h>
00007 #include <allegro5/allegro_ttf.h>
00008 #include <allegro5/mouse.h>
00009 #include <allegro5/events.h>
00010 #include <allegro5/timer.h>
00011 #include <allegro5/color.h>
00012 #include <allegro5/events.h>
00013 #include <allegro5/allegro_image.h>
00014 #include <allegro5/allegro_image.h>
00015 #include <allegro5/allegro_ttf.h>
00016 #include <allegro5/allegro_font.h>
00017 #include "sound.hpp"
00018 #include "music.hpp"
00019
00020 /* -- Globals -- */
00021
00022 /* Consts */
00023 constexpr float FPS = 30;
00024 constexpr int SCREEN_W = 800;
00025 constexpr int SCREEN_H = 600;
00026 constexpr int BUTTON_W = 400;
00027 constexpr int BUTTON_H = 60;
00028 constexpr int OBSTACLES_LIST_NUM = 4;
00029 constexpr float SCALE_PIPES = 0.33f;
00030 constexpr float SCALE_ASTEROID = 0.19f;
00031 const float velocity[] = {9.0f, 9.3f, 9.7f, 10.2f, 10.8f, 11.5f, 12.0f, 12.8f, 13.5f};
00032 const int TAM_VECTOR_VELOCITY = sizeof(velocity)/sizeof(velocity[0]);
00033
00034 extern const ALLEGRO_COLOR BACKGROUND_COLOR;
00035 /* Allegro Components */
00036 extern ALLEGRO_DISPLAY* display;
00037 extern ALLEGRO_EVENT_QUEUE* event_queue;
```

```

00038 extern ALLEGRO_TIMER* timer;
00039
00040 /* Fonts */
00041 extern ALLEGRO_FONT* gameFont;
00042 extern ALLEGRO_FONT* levelFont;
00043
00044 namespace globalVars {
00045     extern bool inInterLevel;
00046     extern int points;
00047     extern std::string usernameGlobal;
00048 }
00049
00050 /* Assets */
00051 extern ALLEGRO_BITMAP* gameOverBackground;
00052 extern ALLEGRO_BITMAP* pinguimBandido;
00053 extern ALLEGRO_BITMAP* backgroundImage;
00054 extern ALLEGRO_BITMAP* pendrive;
00055 extern ALLEGRO_BITMAP* ballShotSprite;
00056
00057 /* Sound FX */
00058 extern Sound* death_sound;
00059 extern Sound* gunshot_sound1;
00060 extern Sound* gunshot_sound2;
00061 extern Sound* gunshot_sound3;
00062 extern Sound* gunshot_sound4;
00063
00064 /* Game Music */
00065 extern Music* menu_music;
00066 extern Music* pause_game_music;
00067 extern Music* level_one_music;
00068 extern Music* level_two_music;
00069 extern Music* level_three_music;
00070 extern Music* defeat_music;
00071 extern Music* victory_music;
00072
00073
00074 class Bootstrap {
00075     private:
00076         static bool initialize_sys_sound();
00077         static void initialize_sounds();
00078     public:
00079         static bool initialize_allegro();
00080         static bool init_allegro_libs();
00081         static void register_allegro_events();
00082         static void cleanup_allegro();
00083         static bool file_exists(const char* path);
00084         static void start_sprite(ALLEGRO_BITMAP *&bitm, const char* path);
00085         static void start_font(ALLEGRO_FONT *&font, const char* path, int size);
00086     };
00087
00088 #endif

```

## 7.5 include/boss\_states.hpp File Reference

### Enumerations

- enum struct **BossStates** { descending , attacking , ascending }
- enum struct **AttackType** {
 ballShots1 , ballShots2 , ballShots3 , lineShotsRight ,
 lineShotsLeft , lineShotsDown }

### 7.5.1 Enumeration Type Documentation

#### 7.5.1.1 AttackType

```
enum struct AttackType [strong]
```

**Enumerator**

ballShots1	
ballShots2	
ballShots3	
lineShotsRight	
lineShotsLeft	
lineShotsDown	

**7.5.1.2 BossStates**

```
enum struct BossStates [strong]
```

**Enumerator**

descending	
attacking	
ascending	

**7.6 boss\_states.hpp**

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

```
00001 #ifndef BOSS_STATES
00002 #define BOSS_STATES
00003
00007 enum struct BossStates {
00008     descending,
00009     attacking,
00010     ascending
00011 };
00012
00016 enum struct AttackType {
00017     ballShots1,
00018     ballShots2,
00019     ballShots3,
00020     lineShotsRight,
00021     lineShotsLeft,
00022     lineShotsDown
00023 };
00024
00025 #endif
```

**7.7 include/circle\_obstacle.hpp File Reference**

```
#include "movement.hpp"
#include "abstract_obstacle.hpp"
#include <vector>
#include <math.h>
```

**Classes**

- class [CircleObstacle](#)

*Representa um obstáculo com uma forma de colisão perfeitamente circular.*

## 7.8 circle\_obstacle.hpp

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

```

00001 #ifndef OBSTACLE_HPP
00002 #define OBSTACLE_HPP
00003
00004 #include "movement.hpp"
00005 #include "abstract_obstacle.hpp"
00006 #include <vector>
00007 #include <math.h>
00008
00015
00016 class CircleObstacle : public AbstractObstacle{
00017 private:
00018
00019     float _radius = 20;
00020
00021 public:
00030     CircleObstacle(const Vector &pos, const char* path);
00031
00036
00037     float get_radius() const;
00043
00044     void draw() override;
00051
00052     void update() override;
00053
00062
00063     bool checkCollisionWithPlayer(BrokenShip& player) override;
00064
00065 };
00066
00067 #endif
00068
00069

```

## 7.9 include/collision.hpp File Reference

```

#include "game_object.hpp"
#include "bootstrap.hpp"
#include "polygon_obstacle.hpp"
#include "movement.hpp"

```

### Functions

- bool **checkCircleCollision** (GameObject &a, float radius\_a, GameObject &b, float radius\_b)  
 Verifica se o jogador está colidindo com a borda direita da tela.
- bool **isCollidingEdge** (Vector &new\_position, BrokenShip \*player)  
 Corrigue a posição do jogador após uma colisão com a borda direita da tela.
- void **newPositionAfterCollisionEdge** (Vector &new\_position, BrokenShip \*player)  
 Corrigue a posição do jogador após uma colisão com a borda direita da tela.
- bool **checkCollisionWithPlayer** (PolygonObstacle &polygon, BrokenShip &player)
- bool **circleSquareCollision** (Vector circlePoint, float radius, Vector squarePoint, float halfSide)  
 Checks whether a circle and a square collide on the plane.
- bool **circleCircleCollision** (Vector circleA, float radiusA, Vector circleB, float radiusB)  
 Check if two circles collided.
- float **distanceBetweenPoints** (Vector pointA, Vector pointB)  
 Calculates the distance between two points on the plane.

## 7.9.1 Function Documentation

### 7.9.1.1 checkCircleCollision()

```
bool checkCircleCollision (
    GameObject & a,
    float radius_a,
    GameObject & b,
    float radius_b)
```

### 7.9.1.2 checkCollisionWithPlayer()

```
bool checkCollisionWithPlayer (
    PolygonObstacle & polygon,
    BrokenShip & player)
```

### 7.9.1.3 circleCircleCollision()

```
bool circleCircleCollision (
    Vector circleA,
    float radiusA,
    Vector circleB,
    float radiusB)
```

Check if two circles collided.

#### Parameters

<i>circleA</i>	Center point of circle A.
<i>radiusA</i>	Radius of circle A.
<i>CircleB</i>	Center point of circle B.
<i>radiusB</i>	Radius of circle B.

#### Returns

There was a collision.

### 7.9.1.4 circleSquareCollision()

```
bool circleSquareCollision (
    Vector circlePoint,
    float radius,
    Vector squarePoint,
    float halfSide)
```

Checks whether a circle and a square collide on the plane.

**Parameters**

<i>circlePoint</i>	Center point of the circle.
<i>radius</i>	Radius of circle.
<i>squarePoint</i>	Center point of the square.
<i>halfSide</i>	Half the side of the square.

**Returns**

There was a collision.

**7.9.1.5 distanceBetweenPoints()**

```
float distanceBetweenPoints (
    Vector pointA,
    Vector pointB)
```

Calculates the distance between two points on the plane.

**Returns**

The distance.

**7.9.1.6 isCollidingEdge()**

```
bool isCollidingEdge (
    Vector & new_position,
    BrokenShip * player)
```

Verifica se o jogador está colidindo com a borda direita da tela.

Testa se a coordenada X da posição futura do jogador, somada ao seu raio, ultrapassa a largura da tela (SCREEN←\_W).

**Parameters**

<i>new_position</i>	A posição futura do jogador a ser testada.
<i>player</i>	Um ponteiro para o objeto do jogador, usado para obter seu raio.

**Returns**

true se estiver colidindo com a borda direita, false caso contrário.

**7.9.1.7 newPositionAfterCollisionEdge()**

```
void newPositionAfterCollisionEdge (
    Vector & new_position,
    BrokenShip * player)
```

Corrigie a posição do jogador após uma colisão com a borda direita da tela.

Reposiciona o jogador de forma que sua borda direita fique exatamente alinhada com a borda direita da tela SCREEN\_W, impedindo que ele saia da área visível.

**Parameters**

<code>new_position</code>	A posição do jogador, que será modificada por referência.
<code>player</code>	Um ponteiro para o objeto do jogador, usado para obter seu raio.

## 7.10 collision.hpp

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

```
00001 #ifndef COLLISION_HPP
00002 #define COLLISION_HPP
00003
00004 #include "game_object.hpp"
00005 #include "bootstrap.hpp"
00006 #include "polygon_obstacle.hpp"
00007 #include "movement.hpp"
00008
00009 bool checkCircleCollision(GameObject& a, float radius_a, GameObject& b, float radius_b);
00010 bool isCollidingEdge(Vector& new_position, BrokenShip* player);
00011 void newPositionAfterCollisionEdge(Vector& new_position, BrokenShip* player);
00012 bool checkCollisionWithPlayer(PolygonObstacle& polygon, BrokenShip& player);
00013
00014 bool circleSquareCollision(Vector circlePoint, float radius, Vector squarePoint, float halfSide);
00015 bool circleCircleCollision(Vector circleA, float radiusA, Vector circleB, float radiusB);
00016 float distanceBetweenPoints(Vector pointA, Vector pointB);
00017
00018 #endif
```

## 7.11 include/database\_users.hpp File Reference

```
#include <string>
#include <vector>
#include "game_over.hpp"
#include <pqxx/pqxx>
```

**Classes**

- struct **User**  
*Estrutura simples para armazenar os dados de um usuário.*
- class **DatabaseUsers**  
*Gerencia todas as operações de banco de dados relacionadas a usuários.*

## 7.12 database\_users.hpp

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

```
00001 #ifndef DATABASE_USER_H
00002 #define DATABASE_USER_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include "game_over.hpp"
00007 #include <pqxx/pqxx>
00008
00013
00014 struct User {
00015     int id;
```

```

00016     std::string username;
00017     int score;
00018     std::string name;
00019     int games;
00020 };
00021
00022
00023 class DatabaseUsers{
00024 private:
00025     pqxx::connection* _connect;
00026 public:
00027
00028     DatabaseUsers();
00029
00030     ~DatabaseUsers();
00031     bool registerUser(const std::string& name, const std::string& username, const std::string&
00032         password, int initialScore = 0, int initialGames = 0);
00033
00034     bool deleteUser(const std::string& username);
00035
00036     std::vector<User> listUsers();
00037
00038
00039     bool updateScore(const std::string& username, int new_score);
00040
00041     bool updateGamesNumber(const std::string& username, int new_games);
00042     std::unique_ptr<User> getUserByUsername(const std::string& username);
00043
00044     bool authenticateUser(const std::string& username, const std::string& password);
00045
00046     void addValuesGameOverScreen(std::string& username, gameOverScreen &game_over_screen);
00047 };
00048
00049 #endif

```

## 7.13 include/dotenv.h File Reference

```

#include <string>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <functional>
#include <cctype>

```

### Classes

- class [dotenv](#)

## 7.14 dotenv.h

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

```

00001 // Copyright (c) 2018 Heikki Johannes Hildén <hildenjohannes@gmail.com>
00002 //
00003 // All rights reserved.
00004 //
00005 // Redistribution and use in source and binary forms, with or without
00006 // modification, are permitted provided that the following conditions are met:
00007 //
00008 //     * Redistributions of source code must retain the above copyright
00009 //       notice, this list of conditions and the following disclaimer.
00010 //
00011 //     * Redistributions in binary form must reproduce the above
00012 //       copyright notice, this list of conditions and the following
00013 //       disclaimer in the documentation and/or other materials provided

```

```

00014 //      with the distribution.
00015 //
00016 //      * Neither the name of copyright holder nor the names of other
00017 //      contributors may be used to endorse or promote products derived
00018 //      from this software without specific prior written permission.
00019 //
00020 // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00021 // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00022 // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00023 // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00024 // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00026 // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00027 // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00028 // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00029 // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00030 // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00031
00035 #pragma once
00036
00037 #include <string>
00038 #include <cstdlib>
00039 #include <fstream>
00040 #include <iostream>
00041 #include <algorithm>
00042 #include <functional>
00043 #include <cctype>
00044
00091 class dotenv
00092 {
00093 public:
00094     dotenv() = delete;
00095     ~dotenv() = delete;
00096
00097     static const unsigned char Preserve = 1 << 0;
00098
00099     static const int OptionsNone = 0;
00100
00101     static void init(const char* filename = ".env");
00102     static void init(int flags, const char* filename = ".env");
00103
00104     static std::string getenv(const char* name, const std::string& def = "");
00105
00106 private:
00107     static void do_init(int flags, const char* filename);
00108     static std::string strip_quotes(const std::string& str);
00109
00110     static std::pair<std::string, bool> resolve_vars(size_t iline, const std::string& str);
00111     static void ltrim(std::string& s);
00112     static void rtrim(std::string& s);
00113     static void trim(std::string& s);
00114     static std::string trim_copy(std::string s);
00115     static size_t find_var_start(const std::string& str, size_t pos, std::string& start_tag);
00116     static size_t find_var_end(const std::string& str, size_t pos, const std::string& start_tag);
00117 };
00118
00125 inline void dotenv::init(const char* filename)
00126 {
00127     dotenv::do_init(OptionsNone, filename);
00128 }
00129
00145 inline void dotenv::init(int flags, const char* filename)
00146 {
00147     dotenv::do_init(flags, filename);
00148 }
00149
00160 inline std::string dotenv::getenv(const char* name, const std::string& def)
00161 {
00162     const char* str = std::getenv(name);
00163     return str ? std::string(str) : def;
00164 }
00165
00166 #if defined(_MSC_VER) || defined(__MINGW32__)
00167
00168 // https://stackoverflow.com/questions/17258029/c-setenv-undefined-identifier-in-visual-studio
00169 inline int setenv(const char *name, const char *value, int overwrite)
00170 {
00171     int errcode = 0;
00172
00173     if (!overwrite)
00174     {
00175         size_t envsize = 0;
00176         errcode = getenv_s(&envsize, NULL, 0, name);
00177         if (errcode || envsize) return errcode;
00178     }
00179     return _putenv_s(name, value);
00180 }

```

```

00181
00182 #endif // _MSC_VER
00183
00194 inline size_t dotenv::find_var_start(const std::string& str, size_t pos, std::string& start_tag)
00195 {
00196     size_t p1      = str.find('$', pos);
00197     size_t p2      = str.find("${", pos);
00198     size_t pos_var = (std::min)(p1, p2);
00199     if(pos_var != std::string::npos) start_tag = (pos_var == p2) ? "${":"$";
00200     return pos_var;
00201 }
00202
00213 inline size_t dotenv::find_var_end(const std::string& str, size_t pos, const std::string& start_tag)
00214 {
00215     char end_tag    = (start_tag == "${")? '}' : ' ';
00216     size_t pos_end  = str.find(end_tag, pos);
00217     // special case when $VARIABLE is at end of str with no trailing whitespace
00218     if(pos_end == std::string::npos && end_tag==' ') pos_end = str.length();
00219     return pos_end;
00220 }
00221
00222 // trim whitespace from left (in place)
00223 inline void dotenv::ltrim(std::string& s) {
00224     s.erase(s.begin(), std::find_if(s.begin(), s.end(), [](int c) {return !std::isspace(c); }));
00225 }
00226
00227 // trim whitespace from right (in place)
00228 inline void dotenv::rtrim(std::string& s) {
00229     s.erase(std::find_if(s.rbegin(), s.rend(), [](int c) {return !std::isspace(c); }).base(),
00230             s.end());
00231
00232 // trim both ends (in place)
00233 inline void dotenv::trim(std::string& s) {
00234     ltrim(s);
00235     rtrim(s);
00236 }
00237
00238 // trim from both ends (copying)
00239 inline std::string dotenv::trim_copy(std::string s) {
00240     trim(s);
00241     return s;
00242 }
00243
00253 inline std::pair<std::string, bool> dotenv::resolve_vars(size_t iline, const std::string& str)
00254 {
00255     std::string resolved;
00256
00257     size_t pos = 0;
00258     size_t pre_pos = pos;
00259     size_t nvar = 0;
00260
00261     bool finished=false;
00262     while(!finished)
00263     {
00264         // look for start of variable expression after pos
00265         std::string start_tag;
00266         pos = find_var_start(str, pos, start_tag);
00267         if(pos != std::string::npos)
00268         {
00269             // a variable definition detected
00270             nvar++;
00271
00272             // keep start of variable expression
00273             size_t pos_start = pos;
00274
00275             size_t lstart = start_tag.length(); // length of start tag
00276             size_t lend   = (lstart>1)? 1 : 0; // length of end tag
00277
00278             // add substring since last variable
00279             resolved += str.substr(pre_pos, pos-pre_pos);
00280
00281             // look for end of variable expression
00282             pos = find_var_end(str, pos, start_tag);
00283             if(pos != std::string::npos)
00284             {
00285                 // variable name with decoration
00286                 std::string var = str.substr(pos_start, pos-pos_start+1);
00287
00288                 // variable name without decoration
00289                 std::string env_var = var.substr(lstart, var.length()-lstart-lend);
00290
00291                 // remove possible whitespace at the end
00292                 rtrim(env_var);
00293
00294                 // evaluate environment variable
00295                 if(const char* env_str = std::getenv(env_var.c_str())))

```

```

00296         {
00297             resolved += env_str;
00298             nvar--; // decrement to indicate variable resolved
00299         }
00300     else
00301     {
00302         // could not resolve the variable, so don't decrement
00303         std::cout << "dotenv: Variable " << var << " is not defined on line " << iline << std::endl;
00304     }
00305
00306     // skip end tag
00307     pre_pos = pos+lend;
00308 }
00309 }
00310 else {
00311     // no more variables
00312     finished = true;
00313 }
00314 }
00315
00316 // add possible trailing non-whitespace after last variable
00317 if(pre_pos < str.length())
00318 {
00319     resolved += str.substr(pre_pos);
00320 }
00321
00322 // nvar must be 0, or else we have an error
00323 return std::make_pair(resolved,(nvar==0));
00324 }
00325
00326 inline void dotenv::do_init(int flags, const char* filename)
00327 {
00328     std::ifstream file;
00329     std::string line;
00330
00331     file.open(filename);
00332
00333     if (file)
00334     {
00335         unsigned int i = 1;
00336
00337         while (getline(file, line))
00338         {
00339             const auto len = line.length();
00340             if (len == 0 || line[0] == '#') {
00341                 continue;
00342             }
00343
00344             const auto pos = line.find("=");
00345
00346             if (pos == std::string::npos) {
00347                 std::cout << "dotenv: Ignoring ill-formed assignment on line "
00348                     << i << ":" << line << std::endl;
00349             } else {
00350                 auto name = trim_copy(line.substr(0, pos));
00351                 auto line_stripped = strip_quotes(trim_copy(line.substr(pos + 1)));
00352
00353                 // resolve any contained variable expressions in 'line_stripped'
00354                 auto p = resolve_vars(i, line_stripped);
00355                 bool ok = p.second;
00356                 if(!ok) {
00357                     std::cout << "dotenv: Ignoring ill-formed assignment on line "
00358                         << i << ":" << line << std::endl;
00359                 }
00360             }
00361
00362             // variable resolved ok, set as environment variable
00363             const auto& val = p.first;
00364             setenv(name.c_str(), val.c_str(), ~flags & dotenv::Preserve);
00365         }
00366     }
00367     ++i;
00368 }
00369 }
00370 }
00371
00372 inline std::string dotenv::strip_quotes(const std::string& str)
00373 {
00374     const std::size_t len = str.length();
00375
00376     if (len < 2)
00377         return str;
00378
00379     const char first = str[0];
00380     const char last = str[len - 1];
00381
00382     if (first == last && ("\" == first || '\\\" == first))

```

```
00383     return str.substr(1, len - 2);
00384
00385     return str;
00386 }
```

## 7.15 include/game\_object.hpp File Reference

```
#include "movement.hpp"
#include "boss_states.hpp"
#include "bootstrap.hpp"
#include <allegro5/allegro.h>
#include <allegro5/allegro_primitives.h>
#include <allegro5(bitmap.h>
#include <allegro5(bitmap_draw.h>
#include <allegro5(bitmap_io.h>
```

### Classes

- class [GameObject](#)  
*Base class to objects of the game.*
- class [FlappyMovement](#)  
*Base class that implements Flappy-Bird-like movement.*
- class [FixedShip](#)  
*Player of phase 3, implements 2D movement.*
- class [BrokenShip](#)  
*Player of phases 1 and 2, implements Flappy Movement.*

## 7.16 game\_object.hpp

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

```
00001 #ifndef GAME_OBJECT_HPP
00002 #define GAME_OBJECT_HPP
00003
00004 #include "movement.hpp"
00005 #include "boss_states.hpp"
00006 #include "bootstrap.hpp"
00007 #include <allegro5/allegro.h>
00008 #include <allegro5/allegro_primitives.h>
00009 #include <allegro5(bitmap.h>
00010 #include <allegro5(bitmap_draw.h>
00011 #include <allegro5(bitmap_io.h>
00012 #include "movement.hpp"
00013
00019
00020 class GameObject {
00021 protected:
00022     Vector _position;
00023 public:
00024     GameObject();
00025     GameObject(Vector position);
00026     ALLEGRO_BITMAP *objectSprite = NULL;
00027     Vector get_position();
00028     void set_position(const Vector &position);
00029     virtual ~GameObject() = 0;
00030     void set_bitmap(const char* path);
00031 };
00032
00039
00040 class FlappyMovement : public GameObject {
00041 private:
00042     static Vector gravity;
```

```

00043     static Vector move_force;
00044 public:
00045     void apply_gravity();
00046     void move_flappy();
00047     Vector getMoveForce();
00048 };
00049
00056 class FixedShip : public GameObject{
00057 private:
00058     static float move_force;
00059     static float _radius;
00060     bool _applyDamage = false; // Indicates whether or not the player can take damage
00061     int _life = 3;
00062
00063 public:
00064     FixedShip();
00065     FixedShip(const Vector &pos);
00066     float get_radius() const;
00067     void set_radius(float r);
00068     void moveShip(char direction);
00069     void draw();
00070
00071     void setCanTakeDamage(bool canTakeDamage);
00072     void takeDamage(bool &playing, int damage=1);
00073 };
00079 class BrokenShip : public FlappyMovement {
00080 private:
00081     float _radius = 12;
00082 public:
00083     BrokenShip();
00084     BrokenShip(const Vector &pos);
00085     float get_radius() const;
00086     void set_radius(float r);
00087     void update();
00088     void draw();
00089     void restart();
00090 };
00091
00092 #endif

```

## 7.17 include/game\_over.hpp File Reference

```

#include "interface.hpp"
#include <string>
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_color.h>
#include <allegro5/allegro_primitives.h>

```

### Classes

- class **gameOverOption**

*Classe base abstrata para as opções de ação na tela de Game Over.*

- class **playAgain**

*Ação para reiniciar o jogo.*

- class **returnMenu**

*Ação para retornar ao menu principal do jogo.*

- class **exitGame**

*Ação para sair do jogo.*

- class **gameOverScreen**

*Gerencia a exibição da mensagem de Game Over, scores, botões de ação e a interação do jogador. \_currentScore: pontuação na partida \_highScore: maior pontuação (recorde pessoal) \_bestScore: Melhor pontuação (recorde geral) ALLEGRO\_FONT\* \_font: fonte dos textos da tela \_playAgainButton: botão jogar novamente \_returnToMenuButton: botão para retornar para a tela inicial \_exitGameButton: botão para sair do jogo.*

## 7.18 game\_over.hpp

Go to the documentation of this file.

```
00001 #ifndef GAME_OVER_HPP
00002 #define GAME_OVER_HPP
00003
00004 #include "interface.hpp"
00005 #include <string>
00006
00007 #include <allegro5/allegro.h>
00008 #include <allegro5/allegro_font.h>
00009 #include <allegro5/allegro_color.h>
00010 #include <allegro5/allegro_primitives.h>
00011
00012
00013     class gameOverOption {
00014 public:
00015     virtual ~gameOverOption() = default;
00016     virtual void execute () = 0;
00017 };
00018
00019
00020
00021
00022
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034 class playAgain : public gameOverOption {
00035 public:
00036     ~playAgain() override = default;
00037     void execute() override;
00038 };
00039
00040
00041
00042
00043
00044 class returnMenu : public gameOverOption {
00045 public:
00046     ~returnMenu() override = default;
00047     void execute() override;
00048 };
00049
00050
00051
00052
00053
00054 class exitGame : public gameOverOption {
00055 public:
00056     ~exitGame() override = default;
00057     void execute() override;
00058 };
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072 class gameOverScreen {
00073 private:
00074
00075     ALLEGRO_FONT* _font;
00076
00077     int _currentScore;
00078     int _highScore;
00079     int _bestScore;
00080     int _numGames;
00081
00082     Button _playAgainButton;
00083     Button _returnToMenuItem;
00084     Button _exitGameButton;
00085
00086
00087 public:
00088     gameOverScreen (ALLEGRO_FONT* font);
00089
00090     void setCurrentScore(int score);
00091     void setHighScore(int score);
00092     void setBestScore(int score);
00093     void setNumGames(int games);
00094     void draw();
00095
00096     gameOverOption* run (ALLEGRO_EVENT_QUEUE* event_queue, ALLEGRO_TIMER* timer);
00097
00098 };
00099
00100 #endif
```

## 7.19 include/interface.hpp File Reference

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_ttf.h>
#include <allegro5/allegro_primitives.h>
```

```
#include <allegro5/color.h>
#include <allegro5/transformations.h>
#include <string>
```

## Classes

- struct [Coordinates](#)  
*Extremely basic coordinates struct. Takes x and y coordinates, width and height as parameters. Mostly used to keep better logic in buttons.*
- class [Button](#)  
*simple button logic with draw, click checking and setText functions*
- class [Interface](#)  
*Base game start interface using button class.*
- class [victoryInterface](#)  
*interface to the "victory screen" after defeating the final boss (windows)*

## 7.20 interface.hpp

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

```
00001 #pragma once
00002
00003 #include <allegro5/allegro.h>
00004 #include <allegro5/allegro_font.h>
00005 #include <allegro5/allegro_ttf.h>
00006 #include <allegro5/allegro_primitives.h>
00007 #include <allegro5/color.h>
00008 #include <allegro5/transformations.h>
00009 #include <string>
00010
00011
00020
00021 struct Coordinates {
00022     double _x, _y, _width, _height;
00023
00024     Coordinates(double x, double y, double width, double height);
00025     Coordinates() : _x(0), _y(0), _width(0), _height(0) {}
00026 };
00027
00039
00040 class Button {
00041     private:
00042     Coordinates _coords;
00043     ALLEGRO_COLOR _color;
00044     std::string _text;
00045     ALLEGRO_FONT* _font;
00046
00047     bool _drawBackground;
00048
00049     public:
00050         Button(Coordinates coords, ALLEGRO_COLOR color, std::string text,
00051                 ALLEGRO_FONT* font, bool drawBackground = true);
00052         void drawButton();
00053         bool gotClicked(int mx, int my);
00054         void setText(const char* txt);
00055
00056 };
00057
00064 class Interface {
00065
00066     ALLEGRO_FONT* _font;
00067     public:
00068     Button playButton;
00069     Button stopSongButton;
00070     Button returnToMenuButton;
00071     Button exitGameButton;
00072
00073     Interface(ALLEGRO_FONT* font);
00074
00075     void drawOffGameInterface();
```

```

00076
00077 };
00078
00079
00086 class victoryInterface {
00087     ALLEGRO_FONT* _font;
00088     public:
00089
00090     victoryInterface(ALLEGRO_FONT* font);
00091     void drawVictoryScreen();
00092
00093
00094 };
00095

```

## 7.21 include/levels.hpp File Reference

```

#include <allegro5/allegro_font.h>
#include <allegro5/bitmap.h>
#include <allegro5/bitmap_draw.h>
#include <allegro5/bitmap_io.h>
#include <allegro5/color.h>
#include <allegro5/display.h>
#include <allegro5/drawing.h>
#include <allegro5/keycodes.h>
#include <allegro5/timer.h>
#include <unistd.h>
#include "abstract_obstacle.hpp"
#include "obstacles_list.hpp"

```

### Classes

- class [Background](#)  
*moving background logic*
- class [Level](#)  
*Base class for the game levels, encapsulates the bare minimum logic to maintain a level.*
- class [LevelOne](#)  
*First phase of the game, basic vertically-oriented flappy bird with satellites as pipes.*
- class [LevelTwo](#)  
*Second phase of the game, vertically-oriented flappy bird with moving obstacles (asteroids)*
- class [LevelThree](#)  
*Third phase of the game, a free-movement, shooter boss fight against Windows (The biggest piece of bloatware in earth)*

### Macros

- #define [LEVEL\\_DURATION](#) 15  
*Duração padrão das fases 1 e 2 (em segundos)*

### Functions

- void [interLevelHandling](#) (std::vector< [AbstractObstacle](#) \* > &obstacles, ALLEGRO\_BITMAP \*sprite, const char \*message, float bitmapScale)  
*Clears the obstacles, draw a selected sprite with a user controlled scale and a message.*

## 7.21.1 Macro Definition Documentation

### 7.21.1.1 LEVEL\_DURATION

```
#define LEVEL_DURATION 15
```

Duração padrão das fases 1 e 2 (em segundos)

## 7.21.2 Function Documentation

### 7.21.2.1 interLevelHandling()

```
void interLevelHandling (
    std::vector< AbstractObstacle * > & obstacles,
    ALLEGRO_BITMAP * sprite,
    const char * message,
    float bitmapScale)
```

Clears the obstacles, draw a selected sprite with a user controlled scale and a message.

#### Parameters

<i>obstacles</i>	vector of abstract obstacles to be cleared
<i>sprite</i>	allegro-typed bitmap of sprite
<i>message</i>	message in the screen
<i>bitmapScale</i>	new width and height scale

## 7.22 levels.hpp

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

```
00001 #ifndef LEVEL_HPP
00002 #define LEVEL_HPP
00003
00004 #include <allegro5/allegro_font.h>
00005 #include <allegro5/bitmap.h>
00006 #include <allegro5/bitmap_draw.h>
00007 #include <allegro5/bitmap_io.h>
00008 #include <allegro5/color.h>
00009 #include <allegro5/display.h>
00010 #include <allegro5/drawing.h>
00011 #include <allegro5/keycodes.h>
00012 #include <allegro5/timer.h>
00013 #include <unistd.h>
00014 #include <allegro5/bitmap.h>
00015 #include "abstract_obstacle.hpp"
00016 #include "obstacles_list.hpp"
00017
00021 #define LEVEL_DURATION 15
00022
00023
00033
00034 void interLevelHandling(std::vector<AbstractObstacle*> & obstacles, ALLEGRO_BITMAP* sprite, const char*
    message, float bitmapScale);
00035
00042
00043 class Background {
00044     float scrollSpeed;
00045     float bgy;
00046     public:
```

```

00048     Background();
00049
00050     void renderBackground();
00051
00052 };
00053
00054
00055
00056 class Level{
00057 protected:
00058     static GameObject* _player;
00059     static Background _bg;
00060     static ALLEGRO_EVENT _event;
00061
00062 public:
00063     friend void interLevelHandling(std::vector<AbstractObstacle*>& obstacles, ALLEGRO_BITMAP* sprite,
00064     const char* message, float bitmapScale);
00065 };
00066
00067 void interLevelHandling(std::vector<AbstractObstacle*>& obstacles, ALLEGRO_BITMAP* sprite, const char*
00068     message, float bitmapScale);
00069
00070
00071 class LevelOne : public Level{
00072 private:
00073     static PipeList _pipesList;
00074
00075 public:
00076     static BrokenShip* setLevelOne();
00077     static void cleanLevel();
00078     static void mainLoop(bool &playing, bool &isAlive);
00079     static void handleTimerEvents(bool &playing, BrokenShip* player, std::vector<AbstractObstacle*>&
00080     obstacles, bool &isAlive);
00081     static void handleKeyPressEvents(bool &playing, BrokenShip* player, bool &isAlive);
00082     static void handleKeyReleaseEvents();
00083 };
00084
00085
00086 class LevelTwo : public Level{
00087 private:
00088     static ObstaclesList _obstaclesList;
00089
00090 public:
00091     static BrokenShip* setLevelTwo();
00092     static void cleanLevel();
00093     static void mainLoop(bool &playing, bool &isAlive);
00094     static void handleTimerEvents(bool &playing, BrokenShip* player, std::vector<AbstractObstacle*>&
00095     obstacles, bool &isAlive);
00096     static void handleKeyPressEvents(bool &playing, BrokenShip* player, bool &isAlive);
00097     static void handleKeyReleaseEvents();
00098 };
00099
00100
00101 class LevelThree : public Level{
00102 private:
00103     static bool key_pressed[ALLEGRO_KEY_MAX];
00104
00105 public:
00106     static FixedShip* setLevelThree();
00107     static void cleanLevel();
00108     static void mainLoop(bool &playing, bool &isAlive);
00109     static void updatePlayerPosition(FixedShip* player);
00110
00111     static void handleTimerEvents(bool &playing, FixedShip* player,
00112     WindowsBoss &windows, bool &isAlive);
00113
00114     static void handleKeyPressEvents(bool &playing, FixedShip* player, WindowsBoss &boss, bool
00115     &isAlive);
00116     static void handleKeyReleaseEvents(bool &playing);
00117     static void handleKeyPressEvents(bool &playing, FixedShip* player);
00118     static void handleKeyReleaseEvents();
00119 };
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135 };
00136
00137
00138 #endif

```

## 7.23 include/menu.hpp File Reference

```
#include "interface.hpp"
#include "music.hpp"
```

## Classes

- class [Menu](#)  
*base menu class*
- class [StartMenu](#)  
*main menu that inherits from the base menu class*

## 7.24 menu.hpp

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

```
00001 #ifndef MENU_HPP
00002 #define MENU_HPP
00003 #include "interface.hpp"
00004 #include "music.hpp"
00005
00011 class Menu {
00012 protected:
00013     static ALLEGRO_EVENT event;
00014     static ALLEGRO_FONT* font;
00015     static Interface* interface;
00016 };
00017
00023 class StartMenu : public Menu {
00024 private:
00025     static void handleTimerEvents();
00026     static void handleMouseEvents(bool &playing, bool &displayInterface);
00027     static void cleanMenu();
00028     static void drawBackground();
00029
00030 public:
00031     static void mainLoopMenu(bool &playing);
00032 };
00033
00034
00035 #endif
```

## 7.25 include/movement.hpp File Reference

### Classes

- class [Vector](#)  
*Implements 2D vectors that represent cartesian coordinates.*

## 7.26 movement.hpp

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

```
00001 #ifndef MOVEMENT_HPP
00002 #define MOVEMENT_HPP
00003
00008 struct Vector{
00009     public:
00010         float _x, _y;
00011
00015
00016     Vector();
00020     Vector(float _xy);
00024     Vector(float _x, float _y);
00028     Vector operator+(const Vector& other) const;
00029
00033     Vector operator-(const Vector& other) const;
00037     Vector operator*(float value) const;
00044     static float dot(const Vector& a, const Vector& b);
00054
00055     static float distance(const Vector& a, const Vector& b);
00066     static float shortestDistancePointToSegment(const Vector& p, const Vector& a, const Vector& b);
00067 };
00068 #endif
```

## 7.27 include/music.hpp File Reference

```
#include <list>
#include <string>
#include "sound.hpp"
#include <allegro5/allegro.h>
#include <allegro5/allegro_audio.h>
#include <allegro5/allegro_acodec.h>
```

### Classes

- class [Music](#)

*represents a sound in a more complex way, with pause and playback methods with fade-in and fade-out systems.*

## 7.28 music.hpp

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

```
00001 #ifndef MUSIC_HPP
00002 #define MUSIC_HPP
00003
00004 #include <list>
00005 #include <string>
00006 #include "sound.hpp"
00007 #include <allegro5/allegro.h>
00008 #include <allegro5/allegro_audio.h>
00009 #include <allegro5/allegro_acodec.h>
00010
00017
00018 class Music : public Sound {
00019 private:
00020     ALLEGRO_SAMPLE_INSTANCE* music_sample = nullptr;           // Music object (more complex than the
sample)
00021
00022     float ballast_volume;                                     // Save the default volume
00023     float volume = 0;                                         // Reference volume at which the system follows
00024     float current_volume = 0;                                  // Real-time volume
00025
00026     unsigned int current_music_position = 0;                  // Position at which the music was paused
00027
00028     float fade_speed;                                       // Fade-in and fade-out speed
00029
00030     static std::list<Music*> music_address;                // Music addresses to have their volumes updated, with
fade-in and fade-out
00031
00032     static Music* lastMusicPlayed;
00033
00034 public:
00035     Music(const char* sound_address, float volume_parameter=0.6f, float fade_speed_parameter=2.0f);
00036
00037     ~Music() override;                                      // If the destroyer of music is called, the destroyer
of sound will not be called
00038
00039     void play();                                           // Method of playing music
00040     void pause();                                          // Method to pause music
00041
00042     static void update_fade_in_fade_out();                 // Method to do fade-in and fade-out
00043     static void muteMusic();
00044     static void unMuteMusic();
00045
00046 #endif
```

## 7.29 include/obstacles\_list.hpp File Reference

```
#include <vector>
#include "abstract_obstacle.hpp"
#include "bootstrap.hpp"
#include "polygon_obstacle.hpp"
#include "circle_obstacle.hpp"
#include "pipe.hpp"
#include <memory>
```

### Classes

- class [ObstaclesList](#)  
*Gerencia uma coleção polimórfica de múltiplos obstáculos.*
- class [PipeList](#)

## 7.30 obstacles\_list.hpp

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

```
00001 #pragma once
00002 #include <vector>
00003 #include "abstract_obstacle.hpp"
00004 #include "bootstrap.hpp"
00005 #include "polygon_obstacle.hpp"
00006 #include "circle_obstacle.hpp"
00007 #include "pipe.hpp"
00008 #include <memory>
00009
00019
00020 class ObstaclesList{
00021
00022 private:
00023     std::vector<AbstractObstacle*> _obstaclesList;
00024
00025 public:
00033     void setPolygonsObstaclesList(const std::vector<Vector>& verts, const char* path);
00040     void setCircleObstaclesList(const char* path);
00045
00046     std::vector<AbstractObstacle*>& getList();
00051     void updateAll(std::vector<AbstractObstacle*> obstaclesList);
00056     void drawAll(std::vector<AbstractObstacle*> obstaclesList);
00062     void clear();
00068     ~ObstaclesList();
00069 };
00070
00071 class PipeList {
00072 private:
00073     std::vector<AbstractObstacle*> pipes;
00074
00075 public:
00076
00077     PipeList() = default;
00082
00083     ~PipeList();
00093     void generatePipes(const std::vector<Vector>& shapeLeft,
00094                         const std::vector<Vector>& shapeRight, const char* imagePathLeft, const char* imagePathRight);
00100     std::vector<AbstractObstacle*>& getList();
00104     void clear();
00105 };
```

## 7.31 include/pipe.hpp File Reference

```
#include "polygon_obstacle.hpp"
```

**Classes**

- class Pipe

*Representa um par de obstáculos (canos) que se movem em conjunto.*

## 7.32 pipe.hpp

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

```
00001 #pragma once
00002 #include "polygon_obstacle.hpp"
00003
00004
00005
00006
00007
00008
00009
00010
00011
00012
00013 class Pipe : public AbstractObstacle{
00014
00015     private:
00016         PolygonObstacle leftPipe;
00017         PolygonObstacle rightPipe;
00018         float gap = 150;
00019
00020     public:
00021         Pipe(const Vector& startPosition, const std::vector<Vector>& shapeLeft,
00022             const std::vector<Vector>& shapeRight, const char* imagePathLeft, const char* imagePathRight);
00023         void update() override;
00024         void draw() override;
00025         bool checkCollisionWithPlayer(BrokenShip& player) override;
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053 };
```

## 7.33 include/polygon\_obstacle.hpp File Reference

```
#include "abstract_obstacle.hpp"
#include "game_object.hpp"
#include "windows_boss.hpp"
#include "boss_states.hpp"
#include <vector>
```

**Classes**

- class PolygonObstacle

*Representa um obstáculo com uma forma de colisão poligonal customizável.*

## 7.34 polygon\_obstacle.hpp

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

```
00001 #pragma once
00002
00003 #include "abstract_obstacle.hpp"
00004 #include "game_object.hpp"
00005 #include "windows_boss.hpp"
00006 #include "boss_states.hpp"
00007 #include <vector>
00008
00009
00010
00011
00012
00013
00014
00015
00016
00017 class PolygonObstacle : public AbstractObstacle{
00018
00019     private:
00020         std::vector<Vector> vertices;
00021         float _scale = 1.00f;
```

```

00022
00023 public:
00024     PolygonObstacle(const Vector &pos, const std::vector<Vector>& verts, float scale, const char*
00025     imagePath)
00026     : vertices(verts), _scale(scale) {
00027
00028     this->set_position(pos);
00029
00030     // aplica a escala em todo o vetor
00031     for (auto& v : vertices) {
00032         v = v * _scale;
00033     }
00034
00035     this->set_bitmap(imagePath);
00036 }
00037
00038 void draw() override;
00039 std::vector<Vector> getVertices();
00040
00041 void update() override;
00042 bool checkCollisionWithPlayer(BrokenShip& player) override;
00043 }
00044
00045 
```

## 7.35 include/register\_interface.hpp File Reference

```
#include "interface.hpp"
#include "database_users.hpp"
```

### Classes

- class **RegisterInterface**  
*Gerencia a interface gráfica e a lógica para registro e login de usuários.*

## 7.36 register\_interface.hpp

Go to the documentation of this file.

```

00001 #pragma once
00002 #include "interface.hpp"
00003 #include "database_users.hpp"
00004
00012
00013 class RegisterInterface {
00014
00015     private:
00016
00017     ALLEGRO_FONT* _font;
00018     ALLEGRO_BITMAP* _titleImage;
00019     bool typingUsername;
00020     bool typingName;
00021     bool typingPassword;
00022     std::string username;
00023     std::string name;
00024     std::string password;
00025     Coordinates nameBox;
00026     Coordinates usernameBox;
00027     Coordinates passwordBox;
00028     Button loginButton;
00029     Button registerButton;
00030     Button exitButton;
00031
00032     public:
00033
00040
00041     RegisterInterface(ALLEGRO_FONT* font);
00042
00047     ~RegisterInterface();
00048
00054     void draw();
```

```

00063     void handleKeyInput(int keycode, unsigned char unicode);
00069     void handleMouseClick(int mx, int my);
00074     std::string getUsername() const;
00079     std::string getName() const;
00084     std::string getPassword() const;
00088     void resetFields();
00099     int mainLoop(bool &inRegister, bool &playing, DatabaseUsers &db);
00100 };

```

## 7.37 include/shapes\_repository.hpp File Reference

```

#include <map>
#include <vector>
#include <string>
#include "movement.hpp"

```

### Variables

- std::map< std::string, const std::vector< Vector > > shape\_repository

### 7.37.1 Variable Documentation

#### 7.37.1.1 shape\_repository

```
std::map<std::string, const std::vector<Vector> > shape_repository
```

##### Initial value:

```

= {
    {"asteroid2", {
        {33.071f, -151}, {72.071f, -160}, {78.071f, -154}, {100.071f, -153},
        {107.071f, -156}, {111.071f, -128}, {141.071f, -102}, {165.071f, -71},
        {160.071f, -50}, {149.071f, -32}, {149.071f, -20}, {140.071f, -10},
        {137.071f, 3}, {142.071f, 19}, {143.071f, 42}, {130.071f, 64},
        {130.071f, 70}, {117.071f, 85}, {94.071f, 95}, {80.071f, 112},
        {68.071f, 118}, {59.071f, 129}, {51.071f, 148}, {22.071f, 149},
        {7.071f, 160}, {-8.929f, 162}, {-23.929f, 158}, {-32.929f, 164},
        {-45.929f, 166}, {-71.929f, 158}, {-80.929f, 140}, {-96.929f, 134},
        {-129.929f, 109}, {-136.929f, 85}, {-154.929f, 65}, {-165.929f, 26},
        {-171.929f, 14}, {-178.929f, 9}, {-181.929f, -1}, {-179.929f, -35},
        {-162.929f, -59}, {-136.929f, -76}, {-125.929f, -105}, {-99.929f, -129},
        {-67.929f, -139}, {-47.929f, -137}, {-37.929f, -144}, {-20.929f, -148},
        {-0.929f, -145}, {7.071f, -151}
    }},
    {"pipe", {
        {-559, 217}, {55, 216}, {-555, 214}, {-555, 187}, {-557, 214}, {-556, 187},
        {-619, 185}, {-766, 199}, {-768, -196}, {-754, -188}, {-762, -191}, {-558, -188},
        {-558, -215}, {-557, -187}, {-555, -214}, {336, -214}, {349, -244}, {365, -245},
        {352, -246}, {364, -247}, {364, -259}, {528, -261}, {530, -247}, {558, -232},
        {560, -207}, {574, -201}, {575, -159}, {586, -159}, {590, 145}, {588, 159},
        {574, 158}, {574, 198}, {560, 229}, {527, 258}, {365, 260}, {364, 246},
        {354, 246}, {351, 231}, {350, 244}, {335, 217}
    }},
}

```

## 7.38 shapes\_repository.hpp

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

```

00001 #pragma once
00002 #include <map>
00003 #include <vector>
00004 #include <string>
00005 #include "movement.hpp"
00006
00007 std::map<std::string, const std::vector<Vector>> shape_repository = {
00008     {"asteroid2", {
00009         {33.071f, -151}, {72.071f, -160}, {78.071f, -154}, {100.071f, -153},
00010         {107.071f, -156}, {111.071f, -128}, {141.071f, -102}, {165.071f, -71},
00011         {160.071f, -50}, {149.071f, -32}, {149.071f, -20}, {140.071f, -10},
00012         {137.071f, 3}, {142.071f, 19}, {143.071f, 42}, {130.071f, 64},
00013         {130.071f, 70}, {117.071f, 85}, {94.071f, 95}, {80.071f, 112},
00014         {68.071f, 118}, {59.071f, 129}, {51.071f, 148}, {22.071f, 149},
00015         {7.071f, 160}, {-8.929f, 162}, {-23.929f, 158}, {-32.929f, 164},
00016         {-45.929f, 166}, {-71.929f, 158}, {-80.929f, 140}, {-96.929f, 134},
00017         {-129.929f, 109}, {-136.929f, 85}, {-154.929f, 65}, {-165.929f, 26},
00018         {-171.929f, 14}, {-178.929f, 9}, {-181.929f, -1}, {-179.929f, -35},
00019         {-162.929f, -59}, {-136.929f, -76}, {-125.929f, -105}, {-99.929f, -129},
00020         {-67.929f, -139}, {-47.929f, -137}, {-37.929f, -144}, {-20.929f, -148},
00021         {-0.929f, -145}, {7.071f, -151}
00022     }},
00023     {"pipe", {
00024         {-559, 217}, {55, 216}, {-555, 214}, {-555, 187}, {-557, 214}, {-556, 187},
00025         {-619, 185}, {-766, 199}, {-768, -196}, {-754, -188}, {-762, -191}, {-558, -188},
00026         {-558, -215}, {-557, -187}, {-555, -214}, {336, -214}, {349, -244}, {365, -245},
00027         {352, -246}, {364, -247}, {364, -259}, {528, -261}, {530, -247}, {558, -232},
00028         {560, -207}, {574, -201}, {575, -159}, {586, -159}, {590, 145}, {588, 159},
00029         {574, 158}, {574, 198}, {560, 229}, {527, 258}, {365, 260}, {364, 246},
00030         {354, 246}, {351, 231}, {350, 244}, {335, 217}
00031     }},
00032 };

```

## 7.39 include/shots.hpp File Reference

```

#include "bootstrap.hpp"
#include "movement.hpp"
#include "game_object.hpp"
#include "levels.hpp"
#include "abstract_obstacle.hpp"
#include "windows_boss.hpp"
#include <string>
#include <list>
#include <iostream>
#include <vector>
#include <allegro5/allegro.h>
#include <allegro5/allegro_primitives.h>
#include <allegro5(bitmap.h>
#include <allegro5(bitmap_draw.h>
#include <allegro5(bitmap_io.h>

```

### Classes

- class [Shot](#)  
*represents a shot in a more abstract way, abstract class.*
- class [BallShot](#)
- class [LineShot](#)

## 7.40 shots.hpp

Go to the documentation of this file.

```
00001 #ifndef SHOTS_HPP
00002 #define SHOTS_HPP
00003
00004 #include "bootstrap.hpp"
00005 #include "movement.hpp"
00006 #include "game_object.hpp"
00007 #include "levels.hpp"
00008 #include "bootstrap.hpp"
00009 #include "abstract_obstacle.hpp"
00010 #include "windows_boss.hpp"
00011
00012 #include <string>
00013 #include <list>
00014 #include <iostream>
00015 #include <vector>
00016
00017 #include <allegro5/allegro.h>
00018 #include <allegro5/allegro_primitives.h>
00019 #include <allegro5/allegro_primitives.h>
00020 #include <allegro5/bitmap.h>
00021 #include <allegro5/bitmap_draw.h>
00022 #include <allegro5/bitmap_io.h>
00023
00024
00030 class Shot : public GameObject {
00031 private:
00032     static void removeInactiveShots(); // Clears the list of inactive shots
00033     static std::vector<Shot*> inactiveShotsList; // List of inactive shots
00034
00035 protected:
00036     Vector _direction;
00037     ALLEGRO_COLOR _shotColor;
00038     static std::list<Shot*> ShotsList; // List of active shots
00039
00040
00041 public:
00042     Shot(Vector position, Vector direction, ALLEGRO_COLOR shotColor);
00043     virtual ~Shot() = default;
00044
00045     bool virtual isActive() = 0;
00046     bool virtual shotCollidedWithPlayer(FixedShip& player) = 0;
00047     bool virtual shotCollidedWithBoss(WindowsBoss& boss) = 0;
00048     void virtual draw() = 0;
00049     void virtual update() = 0;
00050
00051     static void updateShots(FixedShip* player, WindowsBoss& boss, bool &playing); // Updates all shots
00052     static void drawShots(); // Draws all the shots
00053     static void cleanShots();
00054 };
00055
00061 class BallShot : public Shot {
00062 private:
00063     float _speed;
00064     float _radius;
00065
00066 public:
00067     BallShot(Vector initialPosi, Vector direction, float radius, float speed=40);
00068
00069     void draw() override;
00070     void update() override;
00071     bool isActive() override;
00072     bool shotCollidedWithBoss(WindowsBoss& boss) override;
00073     bool shotCollidedWithPlayer(FixedShip& player) override;
00074 };
00075
00081 class LineShot : public Shot {
00082 private:
00083     float _thickness;
00084     float _length;
00085     double _activationTime; // Time for the line shot to cause damage
00086     bool _activated=false;
00087
00088 public:
00089     LineShot(Vector initialPosi, Vector direction, float espessura, float comprimento, double tempoAtivacao);
00090
00091     void draw() override;
00092     void update() override;
00093     bool isActive() override;
00094     bool shotCollidedWithBoss(WindowsBoss& boss) override;
00095     bool shotCollidedWithPlayer(FixedShip& player) override;
00096
```

```
00097 };
00098 #endif
00099 #endif
```

## 7.41 include/sound.hpp File Reference

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_audio.h>
#include <allegro5/allegro_acodec.h>
```

### Classes

- class [Sound](#)

*class that represents a simple sound, with a musical object, ALLEGRO\_SAMPLE, and a play method.*

## 7.42 sound.hpp

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

```
00001 #ifndef SOUND_HPP
00002 #define SOUND_HPP
00003
00004 #include <allegro5/allegro.h>
00005 #include <allegro5/allegro_audio.h>
00006 #include <allegro5/allegro_acodec.h>
00007
00014
00015 class Sound {
00016 protected:
00017     ALLEGRO_SAMPLE* sound_sample = nullptr;      // Simple music object
00018     static bool isSoundMuted;
00019
00020 public:
00021     Sound(const char* sound_address);           // Build the sound
00022     virtual ~Sound();                          // Destroy the sound
00023     void play(float volume=1.0);                // Play the sound
00024
00025     static void muteSounds();
00026     static void unmuteSounds();
00027
00028     static float volumeMester;
00029 };
00030
00031
00032 #endif
```

## 7.43 include/windows\_boss.hpp File Reference

```
#include "game_object.hpp"
```

### Classes

- class [WindowsBoss](#)

## 7.44 windows\_boss.hpp

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

```

00001 #ifndef WINDOWS_BOSS_HPP
00002 #define WINDOWS_BOSS_HPP
00003
00004 #include "game_object.hpp"
00005
00011 class WindowsBoss : public GameObject {
00012 private:
00013     ALLEGRO_COLOR _color = al_map_rgb(255, 255, 255);
00014
00015     BossStates _bossState = BossStates::descending; // Initial
00016     AttackType _attacktype = AttackType::ballShots1; // Initial
00017
00018     bool _applyDamage = false; // Indicates whether or not the boss can take damage
00019     int timeBetweenAttacks;
00020
00021     float _halfSide;
00022     float _life;
00023
00024     bool upBoss(float yStop, float speed);
00025     bool downBoss(float yStop, float speed);
00026
00027     void bossAttack();
00028     static void makeBallShots1();
00029     static void makeBallShots2();
00030     static void makeBallShots3();
00031     static void makeLineShotsRight();
00032     static void makeLineShotsLeft();
00033     static void makeLineShotsDown();
00034
00035     Vector _relativeDistanceToCenterSquare[4];
00036     float _sideOfTheMiniSquare;
00037     ALLEGRO_COLOR miniSquaresColor;
00038     void calculateMiniSquarePositions(); // Calculates the 2 variables above for draw() to work
00039
00040 public:
00041     WindowsBoss(float halfSide, float life);
00042
00043     void draw();
00044     void update(FixedShip* player, bool &playing);
00045     float getHalfSide();
00046     bool isDead();
00047     void takeDamage(FixedShip* player);
00048
00049
00050 };
00051 #endif

```

## 7.45 src/abstract\_obstacle.cpp File Reference

```
#include "abstract_obstacle.hpp"
```

## 7.46 src/bootstrap.cpp File Reference

```

#include <allegro5/allegro_font.h>
#include <allegro5/bitmap_io.h>
#include <allegro5/display.h>
#include <allegro5/events.h>
#include <exception>
#include <iostream>
#include <stdexcept>
#include "bootstrap.hpp"
#include "sound.hpp"
#include "music.hpp"
#include "shots.hpp"

```

## Namespaces

- namespace `globalVars`

## Variables

- const ALLEGRO\_COLOR `BACKGROUND_COLOR` = `al_map_rgb(0, 0, 0)`
- ALLEGRO\_DISPLAY \* `display` = `nullptr`
- ALLEGRO\_EVENT\_QUEUE \* `event_queue` = `nullptr`
- ALLEGRO\_TIMER \* `timer` = `nullptr`
- ALLEGRO\_FONT \* `gameFont` = `nullptr`
- ALLEGRO\_FONT \* `levelFont` = `nullptr`
- ALLEGRO\_BITMAP \* `gameOverBackground` = `nullptr`
- ALLEGRO\_BITMAP \* `pinguimBandido` = `nullptr`
- ALLEGRO\_BITMAP \* `pendrive` = `nullptr`
- ALLEGRO\_BITMAP \* `backgroundImage` = `nullptr`
- ALLEGRO\_BITMAP \* `ballShotSprite` = `nullptr`
- Sound \* `death_sound` = `nullptr`
- Sound \* `gunshot_sound1` = `nullptr`
- Sound \* `gunshot_sound2` = `nullptr`
- Sound \* `gunshot_sound3` = `nullptr`
- Sound \* `gunshot_sound4` = `nullptr`
- bool `globalVars::inInterLevel` = `false`
- int `globalVars::points` = `0`
- std::string `globalVars::usernameGlobal` = `""`
- Music \* `menu_music` = `nullptr`
- Music \* `pause_game_music` = `nullptr`
- Music \* `level_one_music` = `nullptr`
- Music \* `level_two_music` = `nullptr`
- Music \* `level_three_music` = `nullptr`
- Music \* `defeat_music` = `nullptr`
- Music \* `victory_music` = `nullptr`

## 7.46.1 Variable Documentation

### 7.46.1.1 BACKGROUND\_COLOR

```
const ALLEGRO_COLOR BACKGROUND_COLOR = al_map_rgb(0, 0, 0)
```

### 7.46.1.2 backgroundImage

```
ALLEGRO_BITMAP* backgroundImage = nullptr
```

### 7.46.1.3 ballShotSprite

```
ALLEGRO_BITMAP* ballShotSprite = nullptr
```

#### 7.46.1.4 death\_sound

```
Sound* death_sound = nullptr
```

#### 7.46.1.5 defeat\_music

```
Music* defeat_music = nullptr
```

#### 7.46.1.6 display

```
ALLEGRO_DISPLAY* display = nullptr
```

#### 7.46.1.7 event\_queue

```
ALLEGRO_EVENT_QUEUE* event_queue = nullptr
```

#### 7.46.1.8 gameFont

```
ALLEGRO_FONT* gameFont = nullptr
```

#### 7.46.1.9 gameOverBackground

```
ALLEGRO_BITMAP* gameOverBackground = nullptr
```

#### 7.46.1.10 gunshot\_sound1

```
Sound* gunshot_sound1 = nullptr
```

#### 7.46.1.11 gunshot\_sound2

```
Sound* gunshot_sound2 = nullptr
```

#### 7.46.1.12 gunshot\_sound3

```
Sound* gunshot_sound3 = nullptr
```

#### 7.46.1.13 gunshot\_sound4

```
Sound* gunshot_sound4 = nullptr
```

#### 7.46.1.14 **level\_one\_music**

```
Music* level_one_music = nullptr
```

#### 7.46.1.15 **level\_three\_music**

```
Music* level_three_music = nullptr
```

#### 7.46.1.16 **level\_two\_music**

```
Music* level_two_music = nullptr
```

#### 7.46.1.17 **levelFont**

```
ALLEGRO_FONT* levelFont = nullptr
```

#### 7.46.1.18 **menu\_music**

```
Music* menu_music = nullptr
```

#### 7.46.1.19 **pause\_game\_music**

```
Music* pause_game_music = nullptr
```

#### 7.46.1.20 **pendrive**

```
ALLEGRO_BITMAP* pendrive = nullptr
```

#### 7.46.1.21 **pinguimBandido**

```
ALLEGRO_BITMAP* pinguimBandido = nullptr
```

#### 7.46.1.22 **timer**

```
ALLEGRO_TIMER* timer = nullptr
```

#### 7.46.1.23 **victory\_music**

```
Music* victory_music = nullptr
```

## 7.47 src/circle\_obstacle.cpp File Reference

```
#include "circle_obstacle.hpp"
#include "bootstrap.hpp"
```

## 7.48 src/collision.cpp File Reference

```
#include "collision.hpp"
#include <cmath>
```

### Functions

- bool `isCollidingEdge` (`Vector` &`new_position`, `BrokenShip` \*`player`)  
*Verifica se o jogador está colidindo com a borda direita da tela.*
- void `newPositionAfterCollisionEdge` (`Vector` &`new_position`, `BrokenShip` \*`player`)  
*Corrigue a posição do jogador após uma colisão com a borda direita da tela.*
- bool `circleSquareCollision` (`Vector` `circlePoint`, float `radius`, `Vector` `squarePoint`, float `halfSide`)  
*Checks whether a circle and a square collide on the plane.*
- bool `circleCircleCollision` (`Vector` `circleA`, float `radiusA`, `Vector` `circleB`, float `radiusB`)  
*Check if two circles collided.*
- float `distanceBetweenPoints` (`Vector` `pointA`, `Vector` `pointB`)  
*Calculates the distance between two points on the plane.*

### 7.48.1 Function Documentation

#### 7.48.1.1 `circleCircleCollision()`

```
bool circleCircleCollision (
    Vector circleA,
    float radiusA,
    Vector circleB,
    float radiusB)
```

Check if two circles collided.

##### Parameters

<code>circleA</code>	Center point of circle A.
<code>radiusA</code>	Radius of circle A.
<code>circleB</code>	Center point of circle B.
<code>radiusB</code>	Radius of circle B.

##### Returns

There was a collision.

### 7.48.1.2 circleSquareCollision()

```
bool circleSquareCollision (
    Vector circlePoint,
    float radius,
    Vector squarePoint,
    float halfSide)
```

Checks whether a circle and a square collide on the plane.

#### Parameters

<i>circlePoint</i>	Center point of the circle.
<i>radius</i>	Radius of circle.
<i>squarePoint</i>	Center point of the square.
<i>halfSide</i>	Half the side of the square.

#### Returns

There was a collision.

### 7.48.1.3 distanceBetweenPoints()

```
float distanceBetweenPoints (
    Vector pointA,
    Vector pointB)
```

Calculates the distance between two points on the plane.

#### Returns

The distance.

### 7.48.1.4 isCollidingEdge()

```
bool isCollidingEdge (
    Vector & new_position,
    BrokenShip * player)
```

Verifica se o jogador está colidindo com a borda direita da tela.

Testa se a coordenada X da posição futura do jogador, somada ao seu raio, ultrapassa a largura da tela (SCREEN←\_W).

#### Parameters

<i>new_position</i>	A posição futura do jogador a ser testada.
<i>player</i>	Um ponteiro para o objeto do jogador, usado para obter seu raio.

#### Returns

true se estiver colidindo com a borda direita, false caso contrário.

### 7.48.1.5 newPositionAfterCollisionEdge()

```
void newPositionAfterCollisionEdge (
    Vector & new_position,
    BrokenShip * player)
```

Corriga a posição do jogador após uma colisão com a borda direita da tela.

Reposiciona o jogador de forma que sua borda direita fique exatamente alinhada com a borda direita da tela SCREEN\_W, impedindo que ele saia da área visível.

#### Parameters

<i>new_position</i>	A posição do jogador, que será modificada por referência.
<i>player</i>	Um ponteiro para o objeto do jogador, usado para obter seu raio.

## 7.49 src/database\_users.cpp File Reference

```
#include "database_users.hpp"
#include "bootstrap.hpp"
#include "dotenv.h"
#include <iostream>
#include <memory>
```

## 7.50 src/game\_object.cpp File Reference

```
#include <iostream>
#include "game_object.hpp"
#include "shots.hpp"
#include "bootstrap.hpp"
```

#### Variables

- const int **PLAYER\_RADIUS** = 50

### 7.50.1 Variable Documentation

#### 7.50.1.1 **PLAYER\_RADIUS**

```
const int PLAYER_RADIUS = 50
```

## 7.51 src/gameover.cpp File Reference

```
#include "game_over.hpp"
#include <iostream>
#include <string>
#include <allegro5/allegro_primitives.h>
#include <allegro5/allegro_font.h>
#include "music.hpp"
#include "bootstrap.hpp"
```

### Variables

- ALLEGRO\_BITMAP \* `gameOverBackground`
- ALLEGRO\_EVENT\_QUEUE \* `event_queue`
- ALLEGRO\_TIMER \* `timer`

### 7.51.1 Variable Documentation

#### 7.51.1.1 `event_queue`

```
ALLEGRO_EVENT_QUEUE* event_queue [extern]
```

#### 7.51.1.2 `gameOverBackground`

```
ALLEGRO_BITMAP* gameOverBackground [extern]
```

#### 7.51.1.3 `timer`

```
ALLEGRO_TIMER* timer [extern]
```

## 7.52 src/interface.cpp File Reference

```
#include "interface.hpp"
#include "bootstrap.hpp"
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_primitives.h>
#include <allegro5/color.h>
```

## 7.53 src/levels.cpp File Reference

```
#include "levels.hpp"
#include "interface.hpp"
#include "shots.hpp"
#include "shapes_repository.hpp"
#include "bootstrap.hpp"
#include "game_object.hpp"
#include "allegro5/events.h"
#include "music.hpp"
#include "abstract_obstacle.hpp"
#include "obstacles_list.hpp"
#include "collision.hpp"
#include "sound.hpp"
#include <allegro5/timer.h>
```

### Functions

- void **interLevelHandling** (vector< **AbstractObstacle** \* > &obstacles, ALLEGRO\_BITMAP \*sprite, const char \*message, float bitmapScale)

### 7.53.1 Function Documentation

#### 7.53.1.1 interLevelHandling()

```
void interLevelHandling (
    vector< AbstractObstacle * > & obstacles,
    ALLEGRO_BITMAP * sprite,
    const char * message,
    float bitmapScale)
```

## 7.54 src/main.cpp File Reference

```
#include <memory>
#include <iostream>
#include <allegro5/allegro.h>
#include "game_over.hpp"
#include "menu.hpp"
#include "levels.hpp"
#include "bootstrap.hpp"
#include "music.hpp"
#include "database_users.hpp"
#include "register_interface.hpp"
```

### Functions

- int **main** (int argc, char \*\*argv)

## Variables

- ALLEGRO\_EVENT\_QUEUE \* `event_queue`
- ALLEGRO\_TIMER \* `timer`
- ALLEGRO\_FONT \* `gameFont`

## 7.54.1 Function Documentation

### 7.54.1.1 main()

```
int main (
    int argc,
    char ** argv)
```

## 7.54.2 Variable Documentation

### 7.54.2.1 event\_queue

```
ALLEGRO_EVENT_QUEUE* event_queue [extern]
```

### 7.54.2.2 gameFont

```
ALLEGRO_FONT* gameFont [extern]
```

### 7.54.2.3 timer

```
ALLEGRO_TIMER* timer [extern]
```

## 7.55 src/menu.cpp File Reference

```
#include <allegro5/bitmap.h>
#include <allegro5(bitmap_draw.h>
#include <allegro5/color.h>
#include <allegro5/events.h>
#include <iostream>
#include <allegro5/allegro.h>
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_primitives.h>
#include <allegro5/allegro_ttf.h>
#include "menu.hpp"
#include "bootstrap.hpp"
#include "music.hpp"
```

## 7.56 src/movement.cpp File Reference

```
#include "movement.hpp"
#include <cmath>
```

## 7.57 src/music.cpp File Reference

```
#include "music.hpp"
#include <iostream>
```

## 7.58 src/obstacles\_list.cpp File Reference

```
#include "obstacles_list.hpp"
```

## 7.59 src/pipe.cpp File Reference

```
#include "pipe.hpp"
#include <iostream>
#include "bootstrap.hpp"
```

## 7.60 src/polygon\_obstacle.cpp File Reference

```
#include "polygon_obstacle.hpp"
```

## 7.61 src/register\_interface.cpp File Reference

```
#include "register_interface.hpp"
#include "bootstrap.hpp"
#include <allegro5/allegro_primitives.h>
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_ttf.h>
#include <allegro5/allegro.h>
#include <iostream>
```

## 7.62 src/shots.cpp File Reference

```
#include "shots.hpp"
#include "windows_boss.hpp"
#include "bootstrap.hpp"
#include "collision.hpp"
```

## 7.63 src/sound.cpp File Reference

```
#include "sound.hpp"
#include "bootstrap.hpp"
#include <iostream>
```

## 7.64 src/windows\_boss.cpp File Reference

```
#include "windows_boss.hpp"
#include "collision.hpp"
#include "interface.hpp"
#include "shots.hpp"
#include "levels.hpp"
#include "bootstrap.hpp"
#include <iostream>
```

### Variables

- int [cont](#) =0

### 7.64.1 Variable Documentation

#### 7.64.1.1 cont

```
int cont =0
```

# Index

\_bg  
  Level, 50  
\_direction  
  Shot, 86  
\_event  
  Level, 50  
\_height  
  Coordinates, 28  
\_player  
  Level, 50  
\_position  
  GameObject, 44  
\_shotColor  
  Shot, 86  
\_width  
  Coordinates, 28  
\_x  
  Coordinates, 28  
  Vector, 93  
\_y  
  Coordinates, 29  
  Vector, 93  
~AbstractObstacle  
  AbstractObstacle, 12  
~DatabaseUsers  
  DatabaseUsers, 30  
~GameObject  
  GameObject, 43  
~Music  
  Music, 65  
~ObstaclesList  
  ObstaclesList, 66  
~PipeList  
  PipeList, 71  
~RegisterInterface  
  RegisterInterface, 79  
~Shot  
  Shot, 85  
~Sound  
  Sound, 88  
~dotenv  
  dotenv, 34  
~exitGame  
  exitGame, 36  
~gameOverOption  
  gameOverOption, 44  
~playAgain  
  playAgain, 73  
~returnMenu  
  returnMenu, 82  
  
AbstractObstacle, 11  
  ~AbstractObstacle, 12  
  checkCollisionWithPlayer, 12  
  draw, 12  
  getSpeed, 12  
  setSpeed, 13  
  update, 13  
addValuesGameOverScreen  
  DatabaseUsers, 30  
apply\_gravity  
  FlappyMovement, 41  
ascending  
  boss\_states.hpp, 106  
attacking  
  boss\_states.hpp, 106  
AttackType  
  boss\_states.hpp, 105  
authenticateUser  
  DatabaseUsers, 30  
  
Background, 13  
  Background, 14  
  renderBackground, 14  
BACKGROUND\_COLOR  
  bootstrap.cpp, 132  
  bootstrap.hpp, 101  
backgroundImage  
  bootstrap.cpp, 132  
  bootstrap.hpp, 101  
Ball, 14  
BallShot, 14  
  BallShot, 16  
  draw, 16  
  isInactive, 16  
  shotCollidedWithBoss, 17  
  shotCollidedWithPlayer, 17  
  update, 17  
ballShots1  
  boss\_states.hpp, 106  
ballShots2  
  boss\_states.hpp, 106  
ballShots3  
  boss\_states.hpp, 106  
ballShotSprite  
  bootstrap.cpp, 132  
  bootstrap.hpp, 101  
Bootstrap, 18  
  cleanup\_allegro, 18

file\_exists, 18  
 init\_allegro\_libs, 19  
 initialize\_allegro, 19  
 register\_allegro\_events, 19  
 start\_font, 19  
 start\_sprite, 19  
**bootstrap.cpp**  
     BACKGROUND\_COLOR, 132  
     backgroundImage, 132  
     ballShotSprite, 132  
     death\_sound, 132  
     defeat\_music, 133  
     display, 133  
     event\_queue, 133  
     gameFont, 133  
     gameOverBackground, 133  
     gunshot\_sound1, 133  
     gunshot\_sound2, 133  
     gunshot\_sound3, 133  
     gunshot\_sound4, 133  
     level\_one\_music, 133  
     level\_three\_music, 134  
     level\_two\_music, 134  
     levelFont, 134  
     menu\_music, 134  
     pause\_game\_music, 134  
     pendrive, 134  
     pinguimBandido, 134  
     timer, 134  
     victory\_music, 134  
**bootstrap.hpp**  
     BACKGROUND\_COLOR, 101  
     backgroundImage, 101  
     ballShotSprite, 101  
     BUTTON\_H, 101  
     BUTTON\_W, 101  
     death\_sound, 101  
     defeat\_music, 101  
     display, 101  
     event\_queue, 101  
     FPS, 102  
     gameFont, 102  
     gameOverBackground, 102  
     gunshot\_sound1, 102  
     gunshot\_sound2, 102  
     gunshot\_sound3, 102  
     gunshot\_sound4, 102  
     level\_one\_music, 102  
     level\_three\_music, 102  
     level\_two\_music, 102  
     levelFont, 103  
     menu\_music, 103  
     OBSTACLES\_LIST\_NUM, 103  
     pause\_game\_music, 103  
     pendrive, 103  
     pinguimBandido, 103  
     SCALE\_ASTEROID, 103  
     SCALE\_PIPES, 103  
 SCREEN\_H, 103  
 SCREEN\_W, 103  
 TAM\_VECTOR\_VELOCITY, 104  
 timer, 104  
 velocity, 104  
 victory\_music, 104  
**boss\_states.hpp**  
     ascending, 106  
     attacking, 106  
     AttackType, 105  
     ballShots1, 106  
     ballShots2, 106  
     ballShots3, 106  
     BossStates, 106  
     descending, 106  
     lineShotsDown, 106  
     lineShotsLeft, 106  
     lineShotsRight, 106  
**BossStates**  
     boss\_states.hpp, 106  
**BrokenShip**, 20  
     BrokenShip, 21  
     draw, 22  
     get\_radius, 22  
     restart, 22  
     set\_radius, 22  
     update, 22  
**Button**, 22  
     Button, 23  
     drawButton, 23  
     gotClicked, 23  
     setText, 24  
**BUTTON\_H**  
     bootstrap.hpp, 101  
**BUTTON\_W**  
     bootstrap.hpp, 101  
 checkCircleCollision  
     collision.hpp, 108  
 checkCollisionWithPlayer  
     AbstractObstacle, 12  
     CircleObstacle, 26  
     collision.hpp, 108  
     Pipe, 70  
     PolygonObstacle, 76  
 circleCircleCollision  
     collision.cpp, 135  
     collision.hpp, 108  
**CircleObstacle**, 24  
     checkCollisionWithPlayer, 26  
     CircleObstacle, 26  
     draw, 26  
     get\_radius, 27  
     update, 27  
 circleSquareCollision  
     collision.cpp, 135  
     collision.hpp, 108  
 cleanLevel  
     LevelOne, 51

LevelThree, 54  
LevelTwo, 56  
cleanShots  
    Shot, 85  
cleanup\_allegro  
    Bootstrap, 18  
clear  
    ObstaclesList, 67  
    PipeList, 72  
collision.cpp  
    circleCircleCollision, 135  
    circleSquareCollision, 135  
    distanceBetweenPoints, 136  
    isCollidingEdge, 136  
    newPositionAfterCollisionEdge, 136  
collision.hpp  
    checkCircleCollision, 108  
    checkCollisionWithPlayer, 108  
    circleCircleCollision, 108  
    circleSquareCollision, 108  
    distanceBetweenPoints, 109  
    isCollidingEdge, 109  
    newPositionAfterCollisionEdge, 109  
cont  
    windows\_boss.cpp, 142  
Coordinates, 27  
    \_heighth, 28  
    \_width, 28  
    \_x, 28  
    \_y, 29  
    Coordinates, 28  
DatabaseUsers, 29  
    ~DatabaseUsers, 30  
    addValuesGameOverScreen, 30  
    authenticateUser, 30  
    DatabaseUsers, 30  
    deleteUser, 31  
    getUserByUsername, 31  
    listUsers, 31  
    registerUser, 31  
    updateGamesNumber, 32  
    updateScore, 32  
death\_sound  
    bootstrap.cpp, 132  
    bootstrap.hpp, 101  
defeat\_music  
    bootstrap.cpp, 133  
    bootstrap.hpp, 101  
deleteUser  
    DatabaseUsers, 31  
descending  
    boss\_states.hpp, 106  
display  
    bootstrap.cpp, 133  
    bootstrap.hpp, 101  
distance  
    Vector, 92  
distanceBetweenPoints  
    collision.cpp, 136  
    collision.hpp, 109  
dot  
    Vector, 92  
dotenv, 33  
    ~dotenv, 34  
    dotenv, 34  
    getenv, 34  
    init, 35  
    OptionsNone, 35  
    Preserve, 35  
draw  
    AbstractObstacle, 12  
    BallShot, 16  
    BrokenShip, 22  
    CircleObstacle, 26  
    FixedShip, 39  
    gameOverScreen, 46  
    LineShot, 61  
    Pipe, 70  
    PolygonObstacle, 77  
    RegisterInterface, 79  
    Shot, 85  
    WindowsBoss, 96  
drawAll  
    ObstaclesList, 67  
drawButton  
    Button, 23  
drawOffGameInterface  
    Interface, 48  
drawShots  
    Shot, 85  
drawVictoryScreen  
    victoryInterface, 94  
event  
    Menu, 63  
event\_queue  
    bootstrap.cpp, 133  
    bootstrap.hpp, 101  
    gameover.cpp, 138  
    main.cpp, 140  
execute  
    exitGame, 37  
    gameOverOption, 45  
    playAgain, 73  
    returnMenu, 83  
exitGame, 36  
    ~exitGame, 36  
    execute, 37  
exitGameButton  
    Interface, 48  
file\_exists  
    Bootstrap, 18  
FixedShip, 37  
    draw, 39  
    FixedShip, 38  
    get\_radius, 39

moveShip, 39  
 set\_radius, 39  
 setCanTakeDamage, 39  
 takeDamage, 39  
**FlappyMovement**, 40  
 apply\_gravity, 41  
 getMoveForce, 41  
 move\_flappy, 41  
**font**  
 Menu, 63  
**FPS**  
 bootstrap.hpp, 102  
**game\_object.cpp**  
 PLAYER\_RADIUS, 137  
**gameFont**  
 bootstrap.cpp, 133  
 bootstrap.hpp, 102  
 main.cpp, 140  
**GameObject**, 42  
 \_position, 44  
 ~GameObject, 43  
 GameObject, 43  
 get\_position, 43  
 objectSprite, 44  
 set\_bitmap, 43  
 set\_position, 43  
**gameover.cpp**  
 event\_queue, 138  
 gameOverBackground, 138  
 timer, 138  
**gameOverBackground**  
 bootstrap.cpp, 133  
 bootstrap.hpp, 102  
 gameover.cpp, 138  
**gameOverOption**, 44  
 ~gameOverOption, 44  
 execute, 45  
**gameOverScreen**, 45  
 draw, 46  
 gameOverScreen, 46  
 run, 46  
 setbestScore, 46  
 setCurrentScore, 47  
 setHighScore, 47  
 setnumGames, 47  
**games**  
 User, 90  
**generatePipes**  
 PipeList, 72  
**get\_position**  
 GameObject, 43  
**get\_radius**  
 BrokenShip, 22  
 CircleObstacle, 27  
 FixedShip, 39  
**getenv**  
 dotenv, 34  
**getHalfSide**  
 WindowsBoss, 96  
**getList**  
 ObstaclesList, 67  
 PipeList, 72  
**getMoveForce**  
 FlappyMovement, 41  
**getName**  
 RegisterInterface, 79  
**getPassword**  
 RegisterInterface, 79  
**getSpeed**  
 AbstractObstacle, 12  
**getUserByUsername**  
 DatabaseUsers, 31  
**getUsername**  
 RegisterInterface, 79  
**getVertices**  
 PolygonObstacle, 77  
**globalVars**, 9  
 inInterLevel, 9  
 points, 9  
 usernameGlobal, 9  
**gotClicked**  
 Button, 23  
**gunshot\_sound1**  
 bootstrap.cpp, 133  
 bootstrap.hpp, 102  
**gunshot\_sound2**  
 bootstrap.cpp, 133  
 bootstrap.hpp, 102  
**gunshot\_sound3**  
 bootstrap.cpp, 133  
 bootstrap.hpp, 102  
**gunshot\_sound4**  
 bootstrap.cpp, 133  
 bootstrap.hpp, 102  
**handleKeyInput**  
 RegisterInterface, 80  
**handleKeyPressEvents**  
 LevelOne, 51  
 LevelThree, 54  
 LevelTwo, 56  
**handleKeyReleaseEvents**  
 LevelOne, 51  
 LevelThree, 55  
 LevelTwo, 57  
**handleMouseClick**  
 RegisterInterface, 81  
**handleTimerEvents**  
 LevelOne, 51  
 LevelThree, 55  
 LevelTwo, 57  
**id**  
 User, 90  
**include/abstract\_obstacle.hpp**, 99  
**include/bootstrap.hpp**, 100, 104  
**include/boss\_states.hpp**, 105, 106

include/circle\_obstacle.hpp, 106, 107  
include/collision.hpp, 107, 110  
include/database\_users.hpp, 110  
include/dotenv.h, 111  
include/game\_object.hpp, 115  
include/game\_over.hpp, 116, 117  
include/interface.hpp, 117, 118  
include/levels.hpp, 119, 120  
include/menu.hpp, 121, 122  
include/movement.hpp, 122  
include/music.hpp, 123  
include/obstacles\_list.hpp, 124  
include/pipe.hpp, 124, 125  
include/polygon\_obstacle.hpp, 125  
include/register\_interface.hpp, 126  
include/shapes\_repository.hpp, 127, 128  
include/shots.hpp, 128, 129  
include/sound.hpp, 130  
include/windows\_boss.hpp, 130, 131  
inInterLevel  
    globalVars, 9  
init  
    dotenv, 35  
init\_allegro\_libs  
    Bootstrap, 19  
initialize\_allegro  
    Bootstrap, 19  
Interface, 47  
    drawOffGameInterface, 48  
    exitGameButton, 48  
    Interface, 48  
    playButton, 48  
    returnToMenuButton, 48  
    stopSongButton, 48  
interface  
    Menu, 63  
interLevelHandling  
    Level, 49  
    levels.cpp, 139  
    levels.hpp, 120  
isCollidingEdge  
    collision.cpp, 136  
    collision.hpp, 109  
isDead  
    WindowsBoss, 97  
isIsActive  
    BallShot, 16  
    LineShot, 61  
    Shot, 85  
isSoundMuted  
    Sound, 88  
Level, 49  
    \_bg, 50  
    \_event, 50  
    \_player, 50  
    interLevelHandling, 49  
LEVEL\_DURATION  
    levels.hpp, 120  
level\_one\_music  
    bootstrap.cpp, 133  
    bootstrap.hpp, 102  
level\_three\_music  
    bootstrap.cpp, 134  
    bootstrap.hpp, 102  
level\_two\_music  
    bootstrap.cpp, 134  
    bootstrap.hpp, 102  
levelFont  
    bootstrap.cpp, 134  
    bootstrap.hpp, 103  
LevelOne, 50  
    cleanLevel, 51  
    handleKeyPressEvents, 51  
    handleKeyReleaseEvents, 51  
    handleTimerEvents, 51  
    mainLoop, 53  
    setLevelOne, 53  
levels.cpp  
    interLevelHandling, 139  
levels.hpp  
    interLevelHandling, 120  
    LEVEL\_DURATION, 120  
LevelThree, 53  
    cleanLevel, 54  
    handleKeyPressEvents, 54  
    handleKeyReleaseEvents, 55  
    handleTimerEvents, 55  
    mainLoop, 55  
    setLevelThree, 55  
    updatePlayerPosition, 55  
LevelTwo, 56  
    cleanLevel, 56  
    handleKeyPressEvents, 56  
    handleKeyReleaseEvents, 57  
    handleTimerEvents, 57  
    mainLoop, 57  
    setLevelTwo, 57  
Line, 57  
LineShot, 58  
    draw, 61  
    isIsActive, 61  
    LineShot, 60  
    shotCollidedWithBoss, 61  
    shotCollidedWithPlayer, 61  
    update, 62  
lineShotsDown  
    boss\_states.hpp, 106  
lineShotsLeft  
    boss\_states.hpp, 106  
lineShotsRight  
    boss\_states.hpp, 106  
listUsers  
    DatabaseUsers, 31  
main  
    main.cpp, 140  
main.cpp

event\_queue, 140  
 gameFont, 140  
 main, 140  
 timer, 140  
**mainLoop**  
 LevelOne, 53  
 LevelThree, 55  
 LevelTwo, 57  
 RegisterInterface, 81  
**mainLoopMenu**  
 StartMenu, 89  
**Menu**, 62  
 event, 63  
 font, 63  
 interface, 63  
**menu\_music**  
 bootstrap.cpp, 134  
 bootstrap.hpp, 103  
**move\_flappy**  
 FlappyMovement, 41  
**moveShip**  
 FixedShip, 39  
**Music**, 63  
 ~Music, 65  
 Music, 64  
 muteMusic, 65  
 pause, 65  
 play, 65  
 unMuteMusic, 65  
 update\_fade\_in\_fade\_out, 65  
**muteMusic**  
 Music, 65  
**muteSounds**  
 Sound, 88  
**name**  
 User, 90  
**newPositionAfterCollisionEdge**  
 collision.cpp, 136  
 collision.hpp, 109  
**objectSprite**  
 GameObject, 44  
**OBSTACLES\_LIST\_NUM**  
 bootstrap.hpp, 103  
**ObstaclesList**, 66  
 ~ObstaclesList, 66  
 clear, 67  
 drawAll, 67  
 getList, 67  
 setCircleObstaclesList, 67  
 setPolygonsObstaclesList, 67  
 updateAll, 68  
**operator+**  
 Vector, 93  
**operator-**  
 Vector, 93  
**operator\***  
 Vector, 92

OptionsNone  
 dotenv, 35  
**pause**  
 Music, 65  
**pause\_game\_music**  
 bootstrap.cpp, 134  
 bootstrap.hpp, 103  
**pendrive**  
 bootstrap.cpp, 134  
 bootstrap.hpp, 103  
**pinguimBandido**  
 bootstrap.cpp, 134  
 bootstrap.hpp, 103  
**Pipe**, 68  
 checkCollisionWithPlayer, 70  
 draw, 70  
 Pipe, 70  
 update, 70  
**PipeList**, 71  
 ~PipeList, 71  
 clear, 72  
 generatePipes, 72  
 getList, 72  
 PipeList, 71  
**play**  
 Music, 65  
 Sound, 88  
**playAgain**, 73  
 ~playAgain, 73  
 execute, 73  
**playButton**  
 Interface, 48  
**PLAYER\_RADIUS**  
 game\_object.cpp, 137  
**points**  
 globalVars, 9  
**PolygonObstacle**, 74  
 checkCollisionWithPlayer, 76  
 draw, 77  
 getVertices, 77  
 PolygonObstacle, 75  
 update, 77  
**Preserve**  
 dotenv, 35  
**register\_allegro\_events**  
 Bootstrap, 19  
**RegisterInterface**, 78  
 ~RegisterInterface, 79  
 draw, 79  
 getName, 79  
 getPassword, 79  
 getUsername, 79  
 handleKeyInput, 80  
 handleMouseClick, 81  
 mainLoop, 81  
 RegisterInterface, 78  
 resetFields, 81

registerUser  
    DatabaseUsers, 31  
renderBackground  
    Background, 14  
Represents, 82  
resetFields  
    RegisterInterface, 81  
restart  
    BrokenShip, 22  
returnMenu, 82  
    ~returnMenu, 82  
    execute, 83  
returnToMenuItem  
    Interface, 48  
run  
    gameOverScreen, 46  
  
SCALE\_ASTEROID  
    bootstrap.hpp, 103  
SCALE\_PIPES  
    bootstrap.hpp, 103  
score  
    User, 90  
SCREEN\_H  
    bootstrap.hpp, 103  
SCREEN\_W  
    bootstrap.hpp, 103  
set\_bitmap  
    GameObject, 43  
set\_position  
    GameObject, 43  
set\_radius  
    BrokenShip, 22  
    FixedShip, 39  
setbestScore  
    gameOverScreen, 46  
setCanTakeDamage  
    FixedShip, 39  
setCircleObstaclesList  
    ObstaclesList, 67  
setCurrentScore  
    gameOverScreen, 47  
setHighScore  
    gameOverScreen, 47  
setLevelOne  
    LevelOne, 53  
setLevelThree  
    LevelThree, 55  
setLevelTwo  
    LevelTwo, 57  
setnumGames  
    gameOverScreen, 47  
setPolygonsObstaclesList  
    ObstaclesList, 67  
setSpeed  
    AbstractObstacle, 13  
setText  
    Button, 24  
shape\_repository  
    shapes\_repository.hpp, 127  
    shapes\_repository.hpp  
        shape\_repository, 127  
shortestDistancePointToSegment  
    Vector, 93  
Shot, 83  
    \_direction, 86  
    \_shotColor, 86  
    ~Shot, 85  
    cleanShots, 85  
    draw, 85  
    drawShots, 85  
    isInactive, 85  
    Shot, 84  
    shotCollidedWithBoss, 85  
    shotCollidedWithPlayer, 85  
    ShotsList, 86  
    update, 86  
    updateShots, 86  
shotCollidedWithBoss  
    BallShot, 17  
    LineShot, 61  
    Shot, 85  
shotCollidedWithPlayer  
    BallShot, 17  
    LineShot, 61  
    Shot, 85  
ShotsList  
    Shot, 86  
Sound, 87  
    ~Sound, 88  
    isSoundMuted, 88  
    muteSounds, 88  
    play, 88  
    Sound, 87  
    sound\_sample, 88  
    unmuteSounds, 88  
    volumeMester, 88  
sound\_sample  
    Sound, 88  
src/abstract\_obstacle.cpp, 131  
src/bootstrap.cpp, 131  
src/circle\_obstacle.cpp, 135  
src/collision.cpp, 135  
src/database\_users.cpp, 137  
src/game\_object.cpp, 137  
src/gameover.cpp, 138  
src/interface.cpp, 138  
src/levels.cpp, 139  
src/main.cpp, 139  
src/menu.cpp, 140  
src/movement.cpp, 141  
src/music.cpp, 141  
src/obstacles\_list.cpp, 141  
src/pipe.cpp, 141  
src/polygon\_obstacle.cpp, 141  
src/register\_interface.cpp, 141  
src/shots.cpp, 142

src/sound.cpp, 142  
src/windows\_boss.cpp, 142  
start\_font  
    Bootstrap, 19  
start\_sprite  
    Bootstrap, 19  
StartMenu, 89  
    mainLoopMenu, 89  
stopSongButton  
    Interface, 48  
  
takeDamage  
    FixedShip, 39  
    WindowsBoss, 97  
TAM\_VECTOR\_VELOCITY  
    bootstrap.hpp, 104  
timer  
    bootstrap.cpp, 134  
    bootstrap.hpp, 104  
    gameover.cpp, 138  
    main.cpp, 140  
  
unMuteMusic  
    Music, 65  
unmuteSounds  
    Sound, 88  
update  
    AbstractObstacle, 13  
    BallShot, 17  
    BrokenShip, 22  
    CircleObstacle, 27  
    LineShot, 62  
    Pipe, 70  
    PolygonObstacle, 77  
    Shot, 86  
    WindowsBoss, 97  
update\_fade\_in\_fade\_out  
    Music, 65  
updateAll  
    ObstaclesList, 68  
updateGamesNumber  
    DatabaseUsers, 32  
updatePlayerPosition  
    LevelThree, 55  
updateScore  
    DatabaseUsers, 32  
updateShots  
    Shot, 86  
User, 90  
    games, 90  
    id, 90  
    name, 90  
    score, 90  
    username, 90  
username  
    User, 90  
usernameGlobal  
    globalVars, 9

Vector, 91  
    \_x, 93  
    \_y, 93  
distance, 92  
dot, 92  
operator+, 93  
operator-, 93  
operator\*, 92  
shortestDistancePointToSegment, 93  
Vector, 91, 92  
  
velocity  
    bootstrap.hpp, 104  
victory\_music  
    bootstrap.cpp, 134  
    bootstrap.hpp, 104  
victoryInterface, 94  
    drawVictoryScreen, 94  
    victoryInterface, 94  
volumeMester  
    Sound, 88  
  
Windows, 95  
windows\_boss.cpp  
    cont, 142  
WindowsBoss, 95  
    draw, 96  
    getHalfSide, 96  
    isDead, 97  
    takeDamage, 97  
    update, 97  
    WindowsBoss, 96