

World War Jump

v1.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

GameSettings	??
QGraphicsPixmapItem	
BackGround	??
Terrain	??
WorldObject	??
BattleUnit	??
Projectile	??
QGraphicsRectItem	
Input	??
QGraphicsScene	
GameplayInterface	??
QGraphicsView	
GameMenu	??
GameWorld	??
QMainWindow	
MainWindow	??
QObject	
Input	??
PhysicsCalc	??
SoundPlayer	??
WorldObject	??

Chapter 2

Class Index

2.1 Class List

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

BackGround	The background class covers the empty circle in the hollow circle in game. - Wang	??
BattleUnit	Subclass of WorldObject and represents the player's fighting units on the field	??
GameMenu	All necessary things for a functional game menu: Pages, buttons and images. It also has parameters which save custom options. - Wang	??
GameplayInterface	Displays the Terrain , and the players' multiple BattleUnit and Projectile	??
GameSettings	GameSettings saves the in-game setting... - Tomas and Wang	??
GameWorld	Container class for Terrain , Input and GameplayInterface . - Wang and ... who else?	??
Input	Receives the players' key hits	??
MainWindow	??
PhysicsCalc	Our own physics calculator engine and the core of the game. -Can, Tomas, Sebastian	??
Projectile	??
SoundPlayer	This is our sound system. - Wang and Can	??
Terrain	Terrain , the playground for our battle units in form of an inner circle. - WANG	??
WorldObject	??

Chapter 3

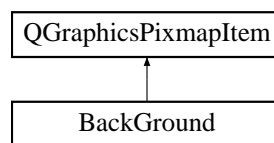
Class Documentation

3.1 BackGround Class Reference

The background class covers the empty circle in the hollow circle in game. - Wang.

```
#include <background.h>
```

Inheritance diagram for BackGround:



Public Member Functions

- **BackGround** ([GameSettings](#) *settings, QTimer *backGroundRotationTimer)

3.1.1 Detailed Description

The background class covers the empty circle in the hollow circle in game. - Wang.

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

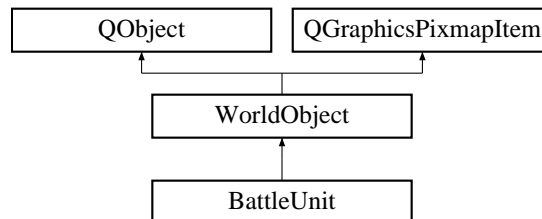
- WorldWarJump/WorldWarJump/background.h
- WorldWarJump/WorldWarJump/background.cpp

3.2 BattleUnit Class Reference

The [BattleUnit](#) class is a subclass of [WorldObject](#) and represents the player's fighting units on the field.

```
#include <battleunit.h>
```

Inheritance diagram for BattleUnit:



Public Slots

- void [shoot](#) ()
[BattleUnit::shoot](#) spawns projectile when the connected button is pressed. The unit chooses the projectile to choose based on how many times it has been shot before and plays corresponding sound. The ProjectileType is cycled and will be changed if the instance of the [BattleUnit](#) has hit an enemy.
- void [setShootAble](#) ()

Public Member Functions

- [BattleUnit](#) ([GameWorld](#) *parentView, [Player](#) player, [SoundPlayer](#) *soundplayer, unitType unittype)
[BattleUnit::BattleUnit](#) constructor. Initializes the center of mass dependent on unit type, initializes the unit parameters and connects the jump and shoot signals with respective slots.
- double [getFiredirection](#) ()
[BattleUnit::getFiredirection](#) returns the fire direction of a battleunit.
- void [setFiredirection](#) (double direction)
[BattleUnit::setFiredirection](#) sets the fire direction of a battleunit.
- unitType [getUnittype](#) ()
- void [calculateShootingPoint](#) (double *Point)
[BattleUnit::calculateShootingPoint](#) calculate the point where the projectile spawns in scene coordinates.

Public Attributes

- [SoundPlayer](#) * [soundpointer](#)

Additional Inherited Members

3.2.1 Detailed Description

The [BattleUnit](#) class is a subclass of [WorldObject](#) and represents the player's fighting units on the field.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 [BattleUnit::BattleUnit](#) ([GameWorld](#) * parentView, [Player](#) player, [SoundPlayer](#) * soundplayer, unitType unittype)

[BattleUnit::BattleUnit](#) constructor. Initializes the center of mass dependent on unit type, initializes the unit parameters and connects the jump and shoot signals with respective slots.

Parameters

<i>parentView</i>	pointer to connect() the BattleUnit to the player's input and the game's refresh rate.
<i>player</i>	the player controlling the unit
<i>soundplayer</i>	the pointer to the global sound player
<i>unittype</i>	the enum that gives the battle unit type

The [BattleUnit](#) is only allowed to shoot every certain milliseconds, set in [GameSettings](#).

3.2.3 Member Function Documentation

3.2.3.1 void BattleUnit::calculateShootingPoint (double * *Point*)

[BattleUnit::calculateShootingPoint](#) calculate the point where the projectile spawns in scene coordinates.

Parameters

<i>Point</i>	the array to be changed into shooting point
--------------	---

3.2.3.2 double BattleUnit::getFiredirection ()

[BattleUnit::getFiredirection](#) returns the fire direction of a battleunit.

Returns

return the fire direction in angles

3.2.3.3 void BattleUnit::setFiredirection (double *direction*)

[BattleUnit::setFiredirection](#) sets the fire direction of a battleunit.

Parameters

<i>direction</i>	the fire direction in angles
------------------	------------------------------

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

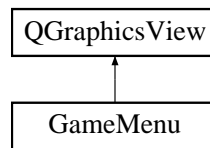
- WorldWarJump/WorldWarJump/battleunit.h
- WorldWarJump/WorldWarJump/battleunit.cpp

3.3 GameMenu Class Reference

The [GameMenu](#) class contains all necessary things for a functional game menu: Pages, buttons and images. It also has parameters which save custom options. - Wang.

```
#include <gamemenu.h>
```

Inheritance diagram for GameMenu:



Public Slots

- void [playeronewon](#) ()
[GameMenu::playeronewon\(\)](#) triggers once the physics calculator says the game is over and player red has won. It shows the end-game scene, then deletes the game scene with it's children.
- void [playertwowon](#) ()
[GameMenu::playertwowon\(\)](#) triggers once the physics calculator says the game is over and player blue has won. It shows the end-game scene, then deletes the game scene with it's children.
- void [changeBGMvolume](#) (int volume)
[GameMenu::changeBGMvolume\(int volume\)](#) sets the volume of the background music to the given number.
- void [changeSEvolume](#) (int volume)
[GameMenu::changeSEvolume\(int volume\)](#) sets the volume of sound effects to the given number.

Public Member Functions

- [GameMenu](#) ([SoundPlayer](#) *soundplayer)
*[GameMenu::GameMenu\(SoundPlayer *soundplayer\)](#) constructor instantiates the setting of the game, startScene, which is the main menu, and several buttons and pictures.*
- int [getGameMenuSize](#) () const
[GameMenu::getGameMenuSize\(\)](#) returns the set resolution of the game menu.
- void [setGameMenuSize](#) (int value)
[GameMenu::setGameMenuSize\(int value\)](#) sets the resolution of the menu.
- void [mousePressEvent](#) ([QMouseEvent](#) *event)
*[GameMenu::mousePressEvent\(QMouseEvent *event\)](#) secures the main functionality of the menu. It detects mouse clicks and compares the [QGraphicsItem](#) on which the mouse is currently positioned with buttons, which are inherited from [QGraphicsPixmapItem](#), then acts correspondingly.*
- int [getPlayer1UnitCount](#) () const
- int [getPlayer2UnitCount](#) () const
- int [getWhichStage](#) () const
[GameMenu::getWhichStage\(\)](#) returns the index of the currently selected stage.

Public Attributes

- [GameSettings](#) * **settings**
- [SoundPlayer](#) * **soundpointer**
because soundplayer is instantiated in main, we need this as a reference to gain access on it.
- [GameWorld](#) * **reference**
used to gain access on created game.
- [MainWindow](#) * **w** = new [MainWindow](#)
- [QGraphicsScene](#) * **startScene**

- is the main menu.*
- QGraphicsScene * [beforeGameScene](#)
- is the scene between main menu and game scene.*
- QGraphicsScene * [settingsScene](#)
- is the setting menu for sound.*
- QGraphicsScene * [aboutScene](#)
- is the about page.*
- QGraphicsScene * [endScene](#)
- is the scene after someone has won.*
- QGraphicsPixmapItem * [startSceneBackground](#)
- is the background picture for main menu.*
- QGraphicsPixmapItem * [beforeGameSceneBackground](#)
- is the background picture for pregame settings.*
- QGraphicsPixmapItem * [endSceneBackground](#)
- is the background picture for winning scene.*
- QGraphicsPixmapItem * **startButton**
- QGraphicsPixmapItem * **settingsButton**
- QGraphicsPixmapItem * **aboutButton**
- QGraphicsPixmapItem * **exitButton**
- QGraphicsPixmapItem * **addPlayer1UnitButton**
- QGraphicsPixmapItem * **addPlayer2UnitButton**
- QGraphicsPixmapItem * **addRedTankButton**
- QGraphicsPixmapItem * **addRedShipButton**
- QGraphicsPixmapItem * **removeRedTankButton**
- QGraphicsPixmapItem * **removeRedShipButton**
- QGraphicsPixmapItem * **addBlueTankButton**
- QGraphicsPixmapItem * **addBlueShipButton**
- QGraphicsPixmapItem * **removeBlueTankButton**
- QGraphicsPixmapItem * **removeBlueShipButton**
- QGraphicsPixmapItem * **removePlayer1UnitButton**
- QGraphicsPixmapItem * **removePlayer2UnitButton**
- QGraphicsPixmapItem * **changeStageButton**
- QGraphicsPixmapItem * **startBattleButton**
- QGraphicsPixmapItem * **backButton**
- QGraphicsPixmapItem * **friendlyFireButton**
- QGraphicsPixmapItem * **yesorno**
- QGraphicsPixmapItem * **player1UnitPicture**
- QGraphicsPixmapItem * **player2UnitPicture**
- QGraphicsPixmapItem * **stagePicture**
- QGraphicsPixmapItem * **titlePicture**
- QGraphicsPixmapItem * **player1UnitCountPicture**
- QGraphicsPixmapItem * **player2UnitCountPicture**
- QGraphicsPixmapItem * **playerRedShipCountPicture**
- QGraphicsPixmapItem * **playerRedTankCountPicture**
- QGraphicsPixmapItem * **playerBlueShipCountPicture**
- QGraphicsPixmapItem * **playerBlueTankCountPicture**
- QGraphicsPixmapItem * **redShipPicture**
- QGraphicsPixmapItem * **redTankPicture**
- QGraphicsPixmapItem * **blueShipPicture**
- QGraphicsPixmapItem * **blueTankPicture**
- QGraphicsPixmapItem * **thumbnail**
- QGraphicsPixmapItem * **muteBGMBButton**
- QGraphicsPixmapItem * **muteSEButton**

- QGraphicsPixmapItem * **bgmVolume**
- QGraphicsPixmapItem * **seVolume**
- QGraphicsPixmapItem * **volumeHint**
- QSlider * **BGMslider**
- QSlider * **SEslider**
- QGraphicsPixmapItem * **settingsBackground**
- QGraphicsPixmapItem * **aboutBackground**
- QGraphicsPixmapItem * **playeronewinsPic**
- QGraphicsPixmapItem * **playertwowinsPic**

3.3.1 Detailed Description

The [GameMenu](#) class contains all necessary things for a functional game menu: Pages, buttons and images. It also has parameters which save custom options. - Wang.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 GameMenu::GameMenu ([SoundPlayer](#) * *soundplayer*)

[GameMenu::GameMenu\(\[SoundPlayer\]\(#\) **soundplayer*\)](#) constructor instantiates the setting of the game, startScene, which is the main menu, and several buttons and pictures.

Parameters

<i>soundplayer</i>	Because the soundplayer is instantiated already in main.cpp, we need to pass it from main to GameMenu .
--------------------	---

3.3.3 Member Function Documentation

3.3.3.1 void GameMenu::changeBGMvolume (int *volume*) [slot]

[GameMenu::changeBGMvolume\(int *volume*\)](#) sets the volume of the background music to the given number.

Parameters

<i>volume</i>	wished volume.
---------------	----------------

3.3.3.2 void GameMenu::changeSEvolume (int *volume*) [slot]

[GameMenu::changeSEvolume\(int *volume*\)](#) sets the volume of sound effects to the given number.

Parameters

<i>volume</i>	wished volume
---------------	---------------

3.3.3.3 `int GameMenu::getGameMenuSize () const`

[GameMenu::getGameMenuSize\(\)](#) returns the set resolution of the game menu.

Returns

the set resolution of the menu

3.3.3.4 `int GameMenu::getPlayer1UnitCount () const`

[GameMenu::getPlayer1UnitCount\(\)](#) returns the unit count of player red (redundant).

3.3.3.5 `int GameMenu::getPlayer2UnitCount () const`

[GameMenu::getPlayer2UnitCount\(\)](#) returns the unit count of blue player (redundant).

3.3.3.6 `int GameMenu::getWhichStage () const`

[GameMenu::getWhichStage\(\)](#) returns the index of the currently selected stage.

Returns

index of the current stage.

3.3.3.7 `void GameMenu::setGameMenuSize (int value)`

[GameMenu::setGameMenuSize\(int value\)](#) sets the resolution of the menu.

Parameters

<i>value</i>	is the wished resolution
--------------	--------------------------

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

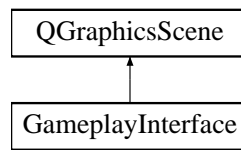
- WorldWarJump/WorldWarJump/gamemenu.h
- WorldWarJump/WorldWarJump/gamemenu.cpp

3.4 GameplayInterface Class Reference

The [GameplayInterface](#) class displays the [Terrain](#), and the players' multiple [BattleUnit](#) and [Projectile](#).

```
#include <GameplayInterface.h>
```

Inheritance diagram for GameplayInterface:



Public Member Functions

- [GameplayInterface](#) ([SoundPlayer](#) *soundplayer)
[GameplayInterface::GameplayInterface](#).

Public Attributes

- [PhysicsCalc](#) * **physicsCalulator**

3.4.1 Detailed Description

The [GameplayInterface](#) class displays the [Terrain](#), and the players' multiple [BattleUnit](#) and [Projectile](#).

Furthermore, the [GameplayInterface](#) contains our physical engine [PhysicsCalc](#).

3.4.2 Constructor & Destructor Documentation

3.4.2.1 [GameplayInterface::GameplayInterface](#) ([SoundPlayer](#) * *soundplayer*)

[GameplayInterface::GameplayInterface](#).

Parameters

<i>soundplayer</i>	static values for the scene's size are fetched from GameSettings
--------------------	--

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

- WorldWarJump/WorldWarJump/GameplayInterface.h
- WorldWarJump/WorldWarJump/GameplayInterface.cpp

3.5 GameSettings Class Reference

[GameSettings](#) saves the in-game setting... - Tomas and Wang.

```
#include <gamesettings.h>
```


Public Member Functions

- int **getPlayer1UnitCount** () const
- void **setPlayer1UnitCount** (int value)
- int **getPlayer2UnitCount** () const
- void **setPlayer2UnitCount** (int value)
- bool **getBeforeGameSceneAlreadyCreated** () const
- void **setBeforeGameSceneAlreadyCreated** (bool value)
- bool **getSettingsSceneAlreadyCreated** () const
- void **setSettingsSceneAlreadyCreated** (bool value)
- bool **getFrendlyFire** ()
- void **setFrendlyFire** (bool value)
- int **getMeeleDmg** ()
- void **setMeeleDmg** (int value)

Static Public Member Functions

- static int **getGameWorldSize** ()
 - static int **getWhichStage** ()
 - static void **setWhichStage** (int value)
 - static double **getGravity** ()
 - static void **setGravityFromMenu** (double value)
 - static double **getTimeStep** ()
 - static void **setTimeStep** (double value)
 - static int **getSecondsToChangeLevel** ()
 - static void **setSecondsToChangeLevel** (int value)
 - static bool **getBGMMuted** ()
 - static void **setBGMMuted** (bool value)
 - static bool **getSEMuted** ()
 - static void **setSEMuted** (bool value)
 - static int **getPlayerRedTankCount** ()
 - static void **setPlayerRedTankCount** (int value)
 - static int **getPlayerRedShipCount** ()
 - static void **setPlayerRedShipCount** (int value)
 - static int **getPlayerBlueShipCount** ()
 - static void **setPlayerBlueShipCount** (int value)
 - static bool **getUnitcollison** ()
 - static int **getPlayerBlueTankCount** ()
 - static void **setPlayerBlueTankCount** (int value)
 - static int **getJumpCountForDestruction** ()
 - static void **setJumpCountForDestruction** (int value)
 - static void **resetUnitCount** ()
- [GameSettings::resetUnitCount](#) sets all units count to 0.*
- static int **getBGMvolume** ()
 - static void **setBGMvolume** (int value)
 - static int **getSEvolume** ()
 - static void **setSEvolume** (int value)
 - static int **getMillisecondsBetweenBattleUnitShots** ()
 - static int **getRefreshRate** ()
 - static void **setRefreshRate** (int value)

Static Public Attributes

- static bool **BGMMuted** = false
- static bool **SEMuted** = false
- static int **BGMvolume** = 25
- static int **SEvolume** = 35
- static bool **gameCreated** = false

3.5.1 Detailed Description

[GameSettings](#) saves the in-game setting... - Tomas and Wang.

All the game's variables are accessible as static member for other classes that include [GameSettings](#) 's header.

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

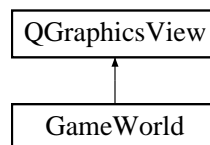
- WorldWarJump/WorldWarJump/gamesettings.h
- WorldWarJump/WorldWarJump/gamesettings.cpp

3.6 GameWorld Class Reference

container class for [Terrain](#), [Input](#) and [GameplayInterface](#). - Wang and ... who else?

```
#include <gameworld.h>
```

Inheritance diagram for GameWorld:



Public Slots

- void **playeronewins** ()
- void **playertwowins** ()
- void **rotateBackground** ()
[GameWorld::rotateBackground](#) This function rotates the Background.
- void **displayMelee** ()
[GameWorld::displayMelee](#) This function sets the Meleelabel to visible.
- void **hideMelee** ()
[GameWorld::hideMelee](#) This sets the Meleelabel to invisible.

Signals

- void **playerOneWinsSignal** ()
- void **playerTwoWinsSignal** ()

Public Member Functions

- [GameWorld](#) ([SoundPlayer](#) *soundplayer)
GameWorld Constructor.
- void **setGameWorldSize** (int value)
- void **pause** ()
- void **resume** ()

Public Attributes

- [Terrain](#) * **terrain**
- [Input](#) * **input**
- [GameplayInterface](#) * **scene**
- QTimer * **backGroundRotationTimer**
- QGraphicsPixmapItem * **background**
- [SoundPlayer](#) * **soundpointer**

3.6.1 Detailed Description

container class for [Terrain](#), [Input](#) and [GameplayInterface](#). - Wang and ... who else?

Details: [GameWorld](#) contains classes that need to communicate with each other and enables connect() functions between them. It also contains a QTimer to make the background rotate.

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

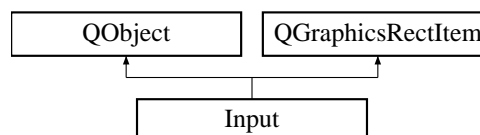
- WorldWarJump/WorldWarJump/gameworld.h
- WorldWarJump/WorldWarJump/gameworld.cpp

3.7 Input Class Reference

The [Input](#) class receives the players' key hits.

```
#include <input.h>
```

Inheritance diagram for Input:



Signals

- void **playerOneJump** ()
- void **playerOneShoot** ()
- void **playerTwoJump** ()
- void **playerTwoShoot** ()

Public Member Functions

- [Input \(\)](#)
[Input::Input](#) This function is the constructor of the [Input](#) class.
- void [keyPressEvent](#) (QKeyEvent *k)
[Input::keyPressEvent](#) This function reacts on any keyboard input and emit a signal if the control buttons for player one or player two had been pressed.
- [~Input \(\)](#)
[Input::~~Input](#) Destructor of the [Input](#) class.

Public Attributes

- QTimer * [refreshRateTimer](#)
Gameplay refresh rate.

3.7.1 Detailed Description

The [Input](#) class receives the players' key hits.

Detailed: it is the focused QGraphicsPixmapItem in our [GameplayInterface](#) scene, and therefore able to receive keyboard input. It receives the input for both players and sends according SIGNALs.

Also because of early architecture decisions, it has the QTimer refreshRateTimer which is connected() with every unit and projectile in our game. This QTimer triggers the move() function in [WorldObject](#) every certain amount of milliseconds set in [GameSettings](#). -Tomas

3.7.2 Member Function Documentation

3.7.2.1 void Input::keyPressEvent (QKeyEvent * k)

[Input::keyPressEvent](#) This function reacts on any keyboard input and emit a signal if the control buttons for player one or player two had been pressed.

Parameters

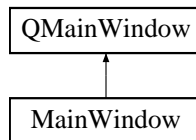
<i>k</i>	is the keyboard button which has been pressed.
----------	--

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

- WorldWarJump/WorldWarJump/input.h
- WorldWarJump/WorldWarJump/input.cpp

3.8 MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Member Functions

- **MainWindow** (QWidget *parent=0)

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

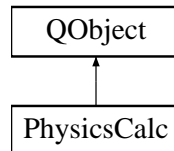
- WorldWarJump/WorldWarJump/mainwindow.h
- WorldWarJump/WorldWarJump/mainwindow.cpp

3.9 PhysicsCalc Class Reference

Our own physics calculator engine and the core of the game. -Can, Tomas, Sebastian.

```
#include <physicscalc.h>
```

Inheritance diagram for PhysicsCalc:



Signals

- void **playeronewins** ()
- void **playertwowins** ()
- void **meeleDmg** ()

Public Member Functions

- **PhysicsCalc** (SoundPlayer *soundplayer)
PhysicsCalc::PhysicsCalc. JumpFrameLimit determines how many timesteps the unit is allowed to not collide with the ground before it is able to jump again.
- void **calculateNewRotValues** (WorldObject *worldObject)
PhysicsCalc::calculateNewRotValues calculates the next orientation of the given *WorldObject* based on it's current orientation and its current angular velocity. Angular array stores in the following order, the angle and angular velocity. Different calculations on projectiles and battleunits. The projectiles "head" is made to always point the speed vector. The Battleunits are made to slowly stand perpendicular to the gravity vector in stabilization module. The closer they get to the center, the less they are stabilized.
- void **updateRotValues** (WorldObject *worldObject, double *angular)
PhysicsCalc::updateRotValues sets the objects new orientation and new angular velocity.

- void **gravVec** ([WorldObject](#) *worldObject, double *gravityVector)
PhysicsCalc::gravityVector gives the gravity vector effecting an objects center of mass at a certain time. First element gives the x and the second gives the y coordinate.
- void **getTopRight** ([WorldObject](#) *worldObject, double *topRight)
- void **getTopLeft** ([WorldObject](#) *worldObject, double *topLeft)
- void **getBottomRight** ([WorldObject](#) *worldObject, double *bottomRight)
- void **getBottomLeft** ([WorldObject](#) *worldObject, double *bottomLeft)
- void **getImpactPoint** ([WorldObject](#) *worldObject, double *impactPoint)
PhysicsCalc::getImpactPoint calculates the impact point with the cornerpoints of the given [WorldObject](#).
- double **gravityAngleDifference** (double rotation, double *gravityVector)
PhysicsCalc::gravityAngleDifference calculates the angle from the gravity vector to the current orientation. The positive direction is clockwise.
- double **roundDown** (double numberToRound, int digit)
PhysicsCalc::roundDown calculates the floor of a number from the given digit.
- void **calculateNewValues** ([WorldObject](#) *)
PhysicsCalc::calculateNewValues calculates the next position of the given [WorldObject](#) based on it's current position and its current speed.
- double **vectorsAbsoluteValue** (double *vector)
PhysicsCalc::vectorsAbsoluteValue calculates the absolute value for a vector in R^2 .
- void **velocityEulerToRadialCoordinates** (double *eulInputPosition, double *inputVelVector, double *outputVelVector, bool eulerToRadial)
PhysicsCalc::velocityEulerToRadialCoordinates transforms the velocity of a [WorldObject](#). -Tomas.
- void **eulToPol** (double *eul, double *pol, char type)
PhysicsCalc::eulToPol translates the given cartesian coordinate system to a polar coordinate system and saves them into a given output pointer.
- void **polToEul** (double *pol, double *eul, char type)
PhysicsCalc::polToEul This function transforms polar coordinates into cartesian coordinates.
- QGraphicsItem * **CollideWithUnit** ([WorldObject](#) *object)
PhysicsCalc::CollideWithUnit checks, if an object collides with an other object of the type [BattleUnit](#) or [Projectile](#) and returns that object.
- void **hitUnit** ([WorldObject](#) *worldObject)
PhysicsCalc::hitUnit calculates the damage, between two colliding objects and checks one of the [WorldObject](#) gets destroyed.
- void **impuls** ([WorldObject](#) *obj1, [WorldObject](#) *obj2)
PhysicsCalc::impuls executes the conservation of the linear momentum for the two colliding objects obj1 and obj2.
- void **checkHealth** ([WorldObject](#) *obj)
PhysicsCalc::checkHealth checks if the given object has healthpoint lower or equal to zero and destroys that unit. The unitcounter of the owning player will be decreased too.
- void **checkWinCondition** ()
PhysicsCalc::checkWinCondition checks if one of the players are out of units and then emit a winning signal.
- void **inverseSpeed** ([WorldObject](#) *colliding1, [WorldObject](#) *colliding2)
PhysicsCalc::inverseSpeed inverts the speed of the first given Worldobject.
- void **meeleDamage** ([WorldObject](#) *colliding1, [WorldObject](#) *colliding2)
PhysicsCalc::meeleDamage calculates the Meele Damage between two Objects. The unit which has a 10 values higher speed than the other deals the damage.
- bool **collideWithAny** ([WorldObject](#) *object)
PhysicsCalc::collideWithAny checks if the given object collides with either an unit or the terrain.
- void **unitUnitCollisionFunc** ([WorldObject](#) *bat1, [WorldObject](#) *bat2)
PhysicsCalc::unitUnitCollisionFunc calculates the collision between two objects and changes the speed of the units. This function is called with BattleUnits.
- bool **CollideWithTerrain** ([WorldObject](#) *object)
CollideWithTerrain checks if one touches the ground and returns a boolean argument. - WANG.

Public Attributes

- int **JumpFrameLimit**
- int **bounceB4Destruction** = settings->getJumpCountForDestruction()
- [SoundPlayer](#) * **soundpointer**
- [GameSettings](#) * **settings**
- double **gravity** = settings->getGravity()
- double **timeStep** = settings->getTimeStep()

3.9.1 Detailed Description

Our own physics calculator engine and the core of the game. -Can, Tomas, Sebastian.

Detailed: It checks for collisions between units and follows a collision protocol. it checks if any player has won and emits according SIGNALs. Furthermore it calculates and triggers sounds accordingly for:

1. Rotation of [WorldObject](#) s
2. Translation of [WorldObject](#) s
3. Gravity effects
4. Momentum conservation at collision
5. Recoil triggering at [BattleUnit](#) shoot()

3.9.2 Constructor & Destructor Documentation

3.9.2.1 PhysicsCalc::PhysicsCalc ([SoundPlayer](#) * *soundplayer*)

[PhysicsCalc::PhysicsCalc](#). JumpFrameLimit determines how many timesteps the unit is allowed to not collide with the ground before it is able to jump again.

Parameters

<i>soundplayer</i>	the global soundplayer pointer
--------------------	--------------------------------

3.9.3 Member Function Documentation

3.9.3.1 void PhysicsCalc::calculateNewRotValues ([WorldObject](#) * *worldObject*)

[PhysicsCalc::calculateNewRotValues](#) calculates the next orientation of the given [WorldObject](#) based on it's current orientation and its current angular velocity. Angular array stores in the following order, the angle and angular velocity. Different calculations on projectiles and battleunits. The projectiles "head" is made to always point the speed vector. The Battleunits are made to slowly stand perpendicular to the gravity vector in stabilization module. The closer they get to the center, the less they are stabilized.

Parameters

<i>worldObject</i>	the worldobject to be calculated
--------------------	----------------------------------

The stabilization module only activates when the object is close to the ground -Can

3.9.3.2 void PhysicsCalc::calculateNewValues (WorldObject * worldObject)

[PhysicsCalc::calculateNewValues](#) calculates the next position of the given [WorldObject](#) based on it's current position and its current speed.

When the [WorldObject](#) moves below the ground (collision) the movement speed of the [WorldObject](#) in radial direction is set in the direction of the center. Then it sets the object's new position and new speed. -Tomas

Parameters

<i>worldObject</i>	the WorldObject instance for which new position is to be calculated and set. If it is a WorldObject of the type Projectile ,then the Projectile bounce counter is increased.
--------------------	--

3.9.3.3 void PhysicsCalc::checkHealth (WorldObject * obj)

[PhysicsCalc::checkHealth](#) checks if the given object has healthpoint lower or equal to zero and destroys that unit. The unitcounter of the owning player will be decreased too.

Parameters

<i>obj</i>	is the WorldObject which should be checked
------------	--

3.9.3.4 bool PhysicsCalc::collideWithAny (WorldObject * object)

[PhysicsCalc::collideWithAny](#) checks if the given object collides with either an unit or the terrain.

Parameters

<i>object</i>	is the object, which will be checked.
---------------	---------------------------------------

Returns

true if it collides, false if it does not.

3.9.3.5 bool PhysicsCalc::CollideWithTerrain (WorldObject * object)

[CollideWithTerrain](#) checks if one touches the ground and returns a boolean argument. - WANG.

[PhysicsCalc::CollideWithTerrain](#) checks if the given object collides with the terrain and returns true or false.

Parameters

<i>object</i>	is the WorldObject , which will be checked.
---------------	---

Returns

true if it collides, false if it does not.

3.9.3.6 QGraphicsItem * PhysicsCalc::CollideWithUnit (WorldObject * *object*)

[PhysicsCalc::CollideWithUnit](#) checks, if an object collides with an other object of the type [BattleUnit](#) or [Projectile](#) and returns that object.

Parameters

<i>object</i>	is the object, which will be checked.
---------------	---------------------------------------

Returns

is a pointer to the object, the object collides with

3.9.3.7 void PhysicsCalc::eulToPol (double * *eul*, double * *pol*, char *type*)

[PhysicsCalc::eulToPol](#) translates the given cartesian coordinate system to a polar coordinate system and saves them into a given output pointer.

Parameters

<i>eul</i>	inputpointer in cartesian coordinates, [0] -> x, [1] -> y.
<i>pol</i>	outputpointer in polar coordinates, [0] -> r, [1] -> phi.
<i>type</i>	type of the translation, v -> velocity, p -> position

3.9.3.8 void PhysicsCalc::getImpactPoint (WorldObject * *worldObject*, double * *impactPoint*)

[PhysicsCalc::getImpactPoint](#) calculates the impact point with the cornerpoints of the given [WorldObject](#).

Parameters

<i>worldObject</i>	the object, which impact point should be calculated.
<i>impactPoint</i>	is pointer to the array where the point will be saved.

3.9.3.9 double PhysicsCalc::gravityAngleDifference (double *rotation*, double * *gravityVector*)

[PhysicsCalc::gravityAngleDifference](#) calculates the angle from the gravity vector to the current orientation. The positive direction is clockwise.

Parameters

<i>rotation</i>	the rotation of the unit
<i>gravityVector</i>	the gravity vector of the unit

Returns

the difference between the units bottom and the gravity vector

3.9.3.10 void PhysicsCalc::gravVec (WorldObject * worldObject, double * gravityVector)

PhysicsCalc::gravityVector gives the gravity vector effecting an objects center of mass at a certain time. First element gives the x and the second gives the y coordinate.

Parameters

<i>worldObject</i>	
--------------------	--

3.9.3.11 void PhysicsCalc::hitUnit (WorldObject * worldObject)

PhysicsCalc::hitUnit calculates the damage, between two colliding objects and checks one of the WorldObject gets destroyed.

Parameters

<i>worldObject</i>	is the WorldObject for which the collision will be calculated.
--------------------	--

3.9.3.12 void PhysicsCalc::impuls (WorldObject * obj1, WorldObject * obj2)

PhysicsCalc::impuls executes the conservation of the linear momentum for the two colliding objects obj1 and obj2.

Parameters

<i>obj1</i>	is the first object which collides.
<i>obj2</i>	is the second object which collides.

3.9.3.13 void PhysicsCalc::inverseSpeed (WorldObject * colliding1, WorldObject * colliding2)

PhysicsCalc::inverseSpeed inverts the speed of the first given Worldobject.

Parameters

<i>colliding1</i>	is the first WorldObject which speed gets inverted.
<i>colliding2</i>	is the second WorldObject, which speed remains unchanged.

3.9.3.14 void PhysicsCalc::meeleDamage (WorldObject * colliding1, WorldObject * colliding2)

PhysicsCalc::meeleDamage calculates the Meele Damage between two Objects. The unit which has a 10 values higher speed than the other deals the damage.

Parameters

<i>colliding1</i>	is the first colliding object.
<i>colliding2</i>	is the second colliding object.

3.9.3.15 void PhysicsCalc::polToEul (double * *pol*, double * *eul*, char *type*)

[PhysicsCalc::polToEul](#) This function transforms polar coordinates into cartesian coordinates.

Parameters

<i>pol</i>	is the inputpointer for polar coordinates, [0] -> x, [1] -> y.
<i>eul</i>	is the outputpointer for the cartesian coordinates, [0] -> r, [1] -> phi.
<i>type</i>	type of the translation, v -> velocity, p -> position.

3.9.3.16 double PhysicsCalc::roundDown (double *numberToRound*, int *digit*)

[PhysicsCalc::roundDown](#) calculates the floor of a number from the given digit.

Parameters

<i>numberToRound</i>	the number to be rounded down
<i>digit</i>	the digit after which will be set to zero

Returns

the rounded number

3.9.3.17 void PhysicsCalc::unitUnitCollisionFunc (WorldObject * *bat1*, WorldObject * *bat2*)

[PhysicsCalc::unitUnitCollisionFunc](#) calculates the collision between two objects and changes the speed of the units. This function is called with BattleUnits.

Parameters

<i>bat1</i>	is the first WorldObject which collides.
<i>bat2</i>	is the second WorldObject which collides.

3.9.3.18 void PhysicsCalc::updateRotValues (WorldObject * *worldObject*, double * *angular*)

[PhysicsCalc::updateRotValues](#) sets the objects new orientation and new angular velocity.

Parameters

<i>worldObject</i>	the worldobject to be updated
<i>angular</i>	the angle and angular speed to be set

3.9.3.19 double PhysicsCalc::vectorsAbsoluteValue (double * *vector*)

[PhysicsCalc::vectorsAbsoluteValue](#) calculates the absolute value for a vector in R^2 .

Parameters

<i>vector</i>	
---------------	--

Returns

absolute value of a vector in R^2 .

3.9.3.20 void PhysicsCalc::velocityEulerToRadialCoordinates (double * *eullInputPosition*, double * *inputVelVector*, double * *outputVelVector*, bool *eulerToRadial*)

[PhysicsCalc::velocityEulerToRadialCoordinates](#) transforms the velocity of a [WorldObject](#). -Tomas.

Detailed: the new velocity vector is in a coordinate system which always points with the first coordinate from the center of the world through the position of the unit outwards radially. The second coordinate points facing int the same direction to the left.

Parameters

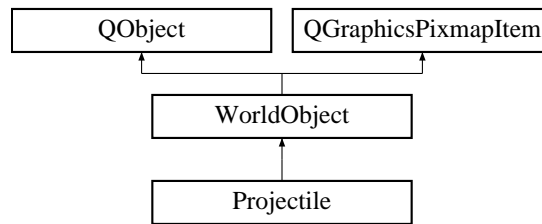
<i>eullInputPosition</i>	objects position to determine what direction is outward.
<i>eullInputVelocity</i>	objects velocity to transform
<i>radialOutput</i>	first coordinate radial, second coordinate is tangential to the Terrain 's circle.
<i>eulerToRadial</i>	true if transforming from Euler coordinates, or false if transforming back to Euler coordinates.

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

- WorldWarJump/WorldWarJump/physicscalc.h
- WorldWarJump/WorldWarJump/physicscalc.cpp

3.10 Projectile Class Reference

Inheritance diagram for Projectile:



Public Member Functions

- [Projectile](#) ([GameWorld](#) *parentView, [BattleUnit](#) *shootingUnit, ProjectileType p, [SoundPlayer](#) *soundplayer, double *shootingPoint)
Projectile::Projectile constructor. Initializes the position, the initial angle , the initial speed ,the projectile type , the weight and the damage and connects the timer It sets the picture and damage depending on the enum Player and ProjectileType.
- [~Projectile](#) ()
Projectile::~~Projectile This function is the destructor of the [Projectile](#) class.
- void [recoil](#) ([WorldObject](#) *obj1, [WorldObject](#) *obj2)
Projectile::recoil This function, creates a recoil on the shooting [BattleUnit](#) by using the conservation of the linear momentum.
- void [polToEul](#) (double *pol, double *eul, char type)
- [WorldObject](#) * [getshootingUnit](#) ()
Projectile::getshootingUnit This function returns the shootingUnit.

Additional Inherited Members

3.10.1 Constructor & Destructor Documentation

- 3.10.1.1 **Projectile::Projectile** ([GameWorld](#) * *parentView*, [BattleUnit](#) * *shootingUnit*, ProjectileType *p*, [SoundPlayer](#) * *soundplayer*, double * *shootingPoint*)

[Projectile::Projectile](#) constructor. Initializes the position, the initial angle , the initial speed ,the projectile type , the weight and the damage and connects the timer It sets the picture and damage depending on the enum Player and ProjectileType.

Parameters

<i>parentView</i>	pointer to connect() the BattleUnit to the player's input and the game's refresh rate.
<i>shootingUnit</i>	the battle unit shooting the projectile
<i>p</i>	the enum that gives the projectile type
<i>soundplayer</i>	the pointer to the global sound player
<i>shootingPoint</i>	the point in scene coordinates where the projectile should spawn

3.10.2 Member Function Documentation

- 3.10.2.1 **WorldObject** * [Projectile::getshootingUnit](#) ()

[Projectile::getshootingUnit](#) This function returns the shootingUnit.

Returns

the shooting Unit

3.10.2.2 void Projectile::recoil (WorldObject * obj1, WorldObject * obj2)

Projectile::recoil This function, creates a recoil on the shooting [BattleUnit](#) by using the conservation of the linear momentum.

Parameters

<i>obj1</i>	is the Shooting BattleUnit
<i>obj2</i>	is Projectile

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

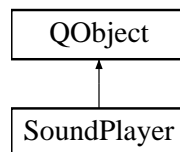
- WorldWarJump/WorldWarJump/projectile.h
- WorldWarJump/WorldWarJump/projectile.cpp

3.11 SoundPlayer Class Reference

This is our sound system. - Wang and Can.

```
#include <soundplayer.h>
```

Inheritance diagram for SoundPlayer:

**Public Member Functions**

- [SoundPlayer](#) ()
SoundPlayer::SoundPlayer initializes the sound players and playlists.
- void [playProjectileTypeShoot](#) (int type)
SoundPlayer::playProjectileTypeShoot plays the correct shooting sound queue to the corresponding projectile type. The projectile sounds cut each other if there is one previously playing. It also plays a taunt voice randomly, with diminishing possibility each time. The taunt line is not cut.
- void [playMenuBGM](#) ()
SoundPlayer::playMenuBGM play menu music.
- void [playGameBGM](#) ()
SoundPlayer::playGameBGM play game music.
- void [playJump](#) ()
SoundPlayer::playJump plays the jump sound when a unit jumps.
- void [playHit](#) ()
SoundPlayer::playHit plays the hit sound when a unit gets hit.

Public Attributes

- QMediaPlayer * **BGMplayer**
- QMediaPlayer * **Jumpplayer**
- QMediaPlayer * **ShootProjectilePlayer**
- QMediaPlaylist * **Projectileplaylist**
- QMediaPlayer * **ShootTauntplayer**
- QMediaPlaylist * **BGMplaylist**
- QMediaPlayer * **Hitplayer**
- int **randomIndex**

3.11.1 Detailed Description

This is our sound system. - Wang and Can.

3.11.2 Member Function Documentation

3.11.2.1 void SoundPlayer::playProjectileTypeShoot (int *type*)

[SoundPlayer::playProjectileTypeShoot](#) plays the correct shooting sound queue to the corresponding projectile type. The projectile sounds cut each other if there is one previously playing. It also plays a taunt voice randomly, with diminishing possibily each time. The taunt line is not cut.

Parameters

<i>type</i>	
-------------	--

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

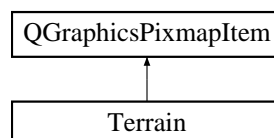
- WorldWarJump/WorldWarJump/soundplayer.h
- WorldWarJump/WorldWarJump/soundplayer.cpp

3.12 Terrain Class Reference

[Terrain](#), the playground for our battle units in form of an inner circle. - WANG.

```
#include <terrain.h>
```

Inheritance diagram for Terrain:



Public Member Functions

- **Terrain** ([GameSettings](#) *settings, [GameplayInterface](#) *scene)

3.12.1 Detailed Description

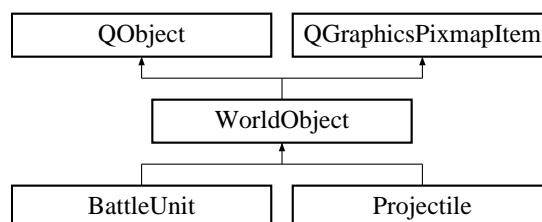
[Terrain](#), the playground for our battle units in form of an inner circle. - WANG.

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

- WorldWarJump/WorldWarJump/terrain.h
- WorldWarJump/WorldWarJump/terrain.cpp

3.13 WorldObject Class Reference

Inheritance diagram for WorldObject:



Public Slots

- void **move** ()
[WorldObject::move](#) This function is called every timestep and gets the new position and speed values for the [WorldObject](#) from the physicscalc.
- void **jump** ()
[WorldObject::jump](#) makes the unit jump in the direction of its head and introduces random rotation. The unit is able to jump in certain proximity to the ground, or when it is colliding with an other unit. The rotation has constant magnitude, but the direction is random.
- void **hit** ()
[WorldObject::hit](#) This function is called every timestep by every [Projectile](#) subclass to check if itself hit any [WorldObject](#).

Signals

- void **sendHealth** (int health)

Public Member Functions

- [WorldObject](#) ([GameWorld](#) *parentView, [Player](#) p, [SoundPlayer](#) *soundplayer)
[WorldObject::WorldObject](#) constructor.
- void [setSpeed](#) (double *newSpeed)
[WorldObject::setSpeed](#) set the speed of the unit and limit to a max speed.
- void [getPosition](#) (double *outputPointer)
- double * [getSpeed](#) ()
[WorldObject::getSpeed](#) returns the speed of the unit.
- void [setOrientation](#) (double newOrientation)
[WorldObject::setOrientation](#) set the turning angle of the unit in degrees.
- double [getOrientation](#) () const
[WorldObject::getOrientation](#) get the turning angle of the unit in degrees.
- void [setRotVel](#) (double newRotVel)
[WorldObject::setRotVel](#) set the rotational velocity in degrees and limit it.
- double [getRotVel](#) () const
[WorldObject::getRotVel](#) returns the rotational velocity in degrees.
- void [setCenterOfMass](#) (double *newCenterOfMass)
[WorldObject::setCenterOfMass](#) sets the position of units center of mass in scene coordinates.
- double * [getCenterOfMass](#) ()
[WorldObject::getCenterOfMass](#) gets the position of units center of mass in scene coordinates.
- void [setHitCounter](#) (int hit)
[WorldObject::setHitCounter](#) set how many times the unit has hit the ground.
- int [getHitCounter](#) ()
[WorldObject::getHitCounter](#) get how many times the unit has hit the ground.
- [Player](#) [getPlayer](#) () const
[WorldObject::getPlayer](#) returns the player controlling the unit.
- int [getWeight](#) ()
[WorldObject::getWeight](#) returns the weight value of the unit.
- void [setWeight](#) (int w)
[WorldObject::setWeight](#) sets the weight value of the unit.
- int [getHealthpoints](#) ()
- int [getDamage](#) ()
- void [setDamage](#) (int d)
- void [setHealthpoints](#) (int points)
[WorldObject::setHealthpoints](#) This function sets the Healthpoints and emit a signal with the healthpoints to the healthpointsbar.
- void [setProjectile](#) (int proj)
- int [getProjectile](#) ()
- char [getChar](#) ()
[WorldObject::getChar](#) returns the character indicating the unit type. If it is a battle unit, the character is 'b' If it is a projectile, the character is 'p' If it is neither, the character is 'o'.
- bool [getBounced](#) () const
[WorldObject::getBounced](#) returns if the object has bounced before.
- void [setBounced](#) (bool value)
[WorldObject::setBounced](#) sets if the object has bounced before.
- bool [getFirstcollide](#) () const
- void [setFirstcollide](#) (bool col)

Public Attributes

- [SoundPlayer](#) * **soundpointer**
- [GameWorld](#) * **parentView**
- bool **collidedBefore**
- bool **okToJump**
- int **jumpCounter**
- bool **orientationChanged**
- int **orientationChangeCount**

Protected Attributes

- Player **p**
- char **ObjectType**

3.13.1 Constructor & Destructor Documentation

3.13.1.1 `WorldObject::WorldObject (GameWorld * parentView, Player p, SoundPlayer * soundplayer)`

[WorldObject::WorldObject](#) constructor.

Parameters

<i>parentView</i>	pointer to connect() the BattleUnit to the player's input and the game's refresh rate.
<i>p</i>	the player controlling the unit
<i>soundplayer</i>	the pointer to the global sound player

3.13.2 Member Function Documentation

3.13.2.1 `bool WorldObject::getBounced () const`

[WorldObject::getBounced](#) returns if the object has bounced before.

Returns

if the object has bounced before

3.13.2.2 `double * WorldObject::getCenterOfMass ()`

[WorldObject::getCenterOfMass](#) gets the position of units center of mass in scene coordinates.

Returns

the center of mass position

3.13.2.3 char WorldObject::getChar ()

[WorldObject::getChar](#) returns the character indicating the unit type. If it is a battle unit, the character is 'b' If it is a projectile, the character is 'p' If it is neither, the character is 'o'.

Returns

the units type

3.13.2.4 int WorldObject::getHitCounter ()

[WorldObject::getHitCounter](#) get how many times the unit has hit the ground.

Returns

the number of collisions with ground

3.13.2.5 double WorldObject::getOrientation () const

[WorldObject::getOrientation](#) get the turning angle of the unit in degrees.

Returns

the turning angle in degrees

3.13.2.6 Player WorldObject::getPlayer () const

[WorldObject::getPlayer](#) returns the player controlling the unit.

Returns

the player controlling the unit

3.13.2.7 double WorldObject::getRotVel () const

[WorldObject::getRotVel](#) returns the rotational velocity in degrees.

Returns

the rotational velocity in degrees

3.13.2.8 `double * WorldObject::getSpeed ()`

[WorldObject::getSpeed](#) returns the speed of the unit.

Returns

the pointer to the speed array

3.13.2.9 `int WorldObject::getWeight ()`

[WorldObject::getWeight](#) returns the weight value of the unit.

Returns

the weight

3.13.2.10 `void WorldObject::setBounced (bool value)`

[WorldObject::setBounced](#) sets if the object has bounced before.

Parameters

<i>value</i>	the bool value indicating if the object has bounced before or not
--------------	---

3.13.2.11 `void WorldObject::setCenterOfMass (double * newCenterOfMass)`

[WorldObject::setCenterOfMass](#) sets the position of units center of mass in scene coordinates.

Parameters

<i>newCenterOfMass</i>	the new center of mass position
------------------------	---------------------------------

3.13.2.12 `void WorldObject::setHealthpoints (int points)`

[WorldObject::setHealthpoints](#) This function sets the Healthpoints and emit a signal with the healthpoints to the healthpointsbar.

Parameters

<i>points</i>	are the lifepoints
---------------	--------------------

3.13.2.13 void WorldObject::setHitCounter (int *hit*)

[WorldObject::setHitCounter](#) set how many times the unit has hit the ground.

Parameters

<i>hit</i>	the number of collisions with ground
------------	--------------------------------------

3.13.2.14 void WorldObject::setOrientation (double *newOrientation*)

[WorldObject::setOrientation](#) set the turning angle of the unit in degrees.

Parameters

<i>newOrientation</i>	the new angle in degrees
-----------------------	--------------------------

3.13.2.15 void WorldObject::setRotVel (double *newRotVel*)

[WorldObject::setRotVel](#) set the rotational velocity in degrees and limit it.

Parameters

<i>newRotVel</i>	the new rotational velocity in degrees
------------------	--

3.13.2.16 void WorldObject::setSpeed (double * *newSpeed*)

[WorldObject::setSpeed](#) set the speed of the unit and limit to a max speed.

Parameters

<i>newSpeed</i>	the pointer to the new speed array
-----------------	------------------------------------

3.13.2.17 void WorldObject::setWeight (int *w*)

[WorldObject::setWeight](#) sets the weight value of the unit.

Parameters

<i>w</i>	the new weight value
----------	----------------------

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

- WorldWarJump/WorldWarJump/worldobject.h

- `WorldWarJump/WorldWarJump/worldobject.cpp`