SPYSpheres

AUTHOR

Version

07/23/2015

Table of Contents

Table of contents

Class Index

Class List

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

File Index

File List

Here is a list of all documented files with brief descriptions:

Class Documentation

ZeroRoboticsGame Class Reference

The class of the game object that you will use.

```
#include <ZRGame.h>
```

Public Member Functions

float **getFuelRemaining** ()
void **sendMessage** (unsigned char inputMsg)
unsigned char **receiveMessage** ()
bool **isFacingOther** ()

Check if the camera is pointed towards the other satellite.

float **takePic** ()

Attempts to take a picture in the current position.

float **getPicPoints** ()

Determines how many points a picture would give if taken immediately.

int **getMemoryFilled** () const

Returns how many memory slots are currently in use.

int **getMemorySize** ()

Returns the total number of memory slots available to the satellite.

float **uploadPics** (void)
Attempts to upload pictures taken to Earth.

bool **isCameraOn** ()
Makes sure the camera is on.

float **getEnergy** ()
Tells how much energy the player has.

float **getOtherEnergy** ()
Tells how much energy the opponent has.

bool **posInLight** (float pos[])
Returns true if the given coordinate is in the light zone.

bool **posInDark** (float pos[])
Returns true if the given coordinate is in the dark zone.

bool **posInGrey** (float pos[])
Returns true if the given coordinate is in a grey zone.

int **posInArea** (float pos[])
Returns 1 if the given coordinate is in the light, -1 if in the dark, and 0 otherwise.

float **getLightInterfacePosition** ()
Determines where the center of the grey zone at the tail end of the light zone is.

float **getDarkGreyBoundary** ()
Determines where the boundary between the dark zone and the grey zone is.

float **getLightGreyBoundary** ()
Determines where the boundary between the light zone and the grey zone is.

float **getLightSwitchTime** ()
Determines how long until the light and dark zones next switch (2D/3D).

int **getNumItem** ()
Returns the number of total items in play, whether they have been picked up yet or not.

bool **useMirror** ()
Uses a held mirror item.

int **getMirrorTimeRemaining** ()
Returns the amount of time left on your current mirror.

int **getNumMirrorsHeld** ()
Returns the number of mirrors currently held and available for use.

void **getItemLoc** (float pos[], int itemID)
Copies the location of a given item into the given array.

int **hasItem** (int itemID)
Tells who has a given item.

int **getItemType** (int itemID)
Returns what the item does.

float **getScore** ()
Returns the player's current score.

float **getOtherScore** ()
Returns the opponent's current score.

int **getCurrentTime** ()
Returns the time.

ZeroRoboticsGame (ZeroRoboticsGameImpl &impl, ZeroRoboticsAPIImpl &apiImpl)
Constructor for the game. The provided references should be singleton instances.

Static Public Member Functions

static **ZeroRoboticsGame** & **instance** ()

Detailed Description

The class of the game object that you will use.

Contains publicly available member functions.

Member Function Documentation

float ZeroRoboticsGame::getDarkGreyBoundary ()

Determines where the boundary between the dark zone and the grey zone is.

Returns:
The y-coordinate of the plane between the dark zone and the grey zone.

float ZeroRoboticsGame::getEnergy ()

Tells how much energy the player has.

Returns:
Amount of energy the player satellite currently has.

float ZeroRoboticsGame::getFuelRemaining ()

Tells the player how much fuel remains.

Returns:
float indicating how many seconds of fuel remain.

void ZeroRoboticsGame::getItemLoc (float *pos*[], int *itemID*)

Copies the location of a given item into the given array.

Parameters:

pos	A pointer to an array of size 3 which will be overwritten by the item location.
itemID	The integer identifier of a given item.

int ZeroRoboticsGame::getItemType (int *itemID*)

Returns what the item does.

Possible Item Types:

1 ITEM_TYPE_ADD_SCORE
2 ITEM_TYPE_ADD_ENERGY
3 ITEM_TYPE_ADD_MEMORY

Parameters:

itemID	The integer identifier of a given item.

Returns:

The corresponding item type to the given identifier.

float ZeroRoboticsGame::getLightGreyBoundary ()

Determines where the boundary between the light zone and the grey zone is.

Returns:

The y-coordinate of the plane between the light zone and the grey zone.

float ZeroRoboticsGame::getLightInterfacePosition ()

Determines where the center of the grey zone at the tail end of the light zone is.

The tail end is at the lower Y-coordinate of the light zone, disregarding any portion that has wrapped around.

Returns:

The y-coordinate of the light interface plane.

float ZeroRoboticsGame::getLightSwitchTime ()

Determines how long until the light and dark zones next switch (2D/3D).

Returns:

Number of seconds until the light switches.

int ZeroRoboticsGame::getMemoryFilled () const

Returns how many memory slots are currently in use.

Returns:

The number of memory slots used.

int ZeroRoboticsGame::getMemorySize ()

Returns the total number of memory slots available to the satellite.

This includes both used and unused slots.

Returns:

Number of memory slots available.

int ZeroRoboticsGame::getMirrorTimeRemaining ()

Returns the amount of time left on your current mirror.

Returns:

remaining time with a mirror up, zero if no mirror is up.

int ZeroRoboticsGame::getNumItem ()

Returns the number of total items in play, whether they have been picked up yet or not.

Returns:

Number of total items.

int ZeroRoboticsGame::getNumMirrorsHeld ()

Returns the number of mirrors currently held and available for use.

Returns:

number of mirrors held by the player.

float ZeroRoboticsGame::getOtherEnergy ()

Tells how much energy the opponent has.

Returns:

Amount of energy the opponent satellite currently has.

float ZeroRoboticsGame::getPicPoints ()

Determines how many points a picture would give if taken immediately.

Does not actually take a picture. Costs 0.1 energy.

Returns:

The amount of points that the picture is worth.

float ZeroRoboticsGame::getScore ()

Returns the player's current score.

Returns:

Player satellite score.

int ZeroRoboticsGame::hasItem (int *itemID*)

Tells who has a given item.

Parameters:

itemID	The integer identifier of a given item.

Returns:

0 if you have picked up the specified item, 1 if the other player has, or -1 if no one has.

static ZeroRoboticsGame& ZeroRoboticsGame::instance () `[static]`

Retrieves the singleton instance of the game API. Users are not allowed to construct a game instance, so the API must be retrieved through this interface.

Returns:

singleton of the game API

bool ZeroRoboticsGame::isCameraOn ()

Makes sure the camera is on.

Returns:

true if the camera is usable, false if not.

bool ZeroRoboticsGame::isFacingOther ()

Check if the camera is pointed towards the other satellite.

Returns:

true if the camera is facing the other satellite, false otherwise.

int ZeroRoboticsGame::posInArea (float *pos*[])

Returns 1 if the given coordinate is in the light, -1 if in the dark, and 0 otherwise.

Parameters:

pos	An array of three floats in (x, y, z) order.

Returns:

1 if the given coordinate is in the light, -1 if in the dark, and 0 else.

bool ZeroRoboticsGame::posInDark (float *pos*[])

Returns true if the given coordinate is in the dark zone.

Parameters:

pos	An array of three floats in (x, y, z) order.

Returns:

true if the coordinate is in dark, false else.

bool ZeroRoboticsGame::posInGrey (float *pos*[])

Returns true if the given coordinate is in a grey zone.

Parameters:

pos	An array of three floats in (x, y, z) order.

Returns:

true if the coordinate is in grey, false else.

bool ZeroRoboticsGame::posInLight (float *pos*[])

Returns true if the given coordinate is in the light zone.

Parameters:

pos	An array of three floats in (x, y, z) order.

Returns:

true if the coordinate is in light, false else.

unsigned char ZeroRoboticsGame::receiveMessage ()

Recieve value from 0-255 from other satellite.

Returns:

An unsigned char containing a value from 0-255.

void ZeroRoboticsGame::sendMessage (unsigned char *inputMsg*)

Send a value from 0-255 to the other satellite.

Parameters:

inputMsg	Unsigned Char to be sent to other satellite.

float ZeroRoboticsGame::takePic ()

Attempts to take a picture in the current position.

The camera will be disabled for 3 seconds after an attempt, whether successful or not. Costs 1.0 energy.

Returns:

The amount of points that the picture taken is worth.

float ZeroRoboticsGame::uploadPics (void)

Attempts to upload pictures taken to Earth.

Will fail if not facing Earth (3D/Alliance). Disables camera for three seconds upon sucessful upload. Costs 1.0 energy.

Returns:

The total score over the course of the game so far.

bool ZeroRoboticsGame::useMirror ()

Uses a held mirror item.

Returns:

true if the item existed and was used, false otherwise.

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

4 **ZRGame.h**

File Documentation

Constants.h File Reference

A list of constants used in the ZR program.

```
#include "math_matrix.h"
```

Defines

#define **GAME_TIME** 0

The time at game start.

#define **VEL_X** 3

The index for the beginning of the velocity array inside of ZRState.

#define **MAX_GAME_TIME** 180

Length of the whole game in seconds.

#define **MAX_FACING_ANGLE** 0.968912f

Cosine of the angle at which pictures may be taken/uploaded.

#define **ITEM_TYPE_ADD_SCORE** 0

The type identifier for a score item.

#define **ITEM_TYPE_ADD_ENERGY** 1

The type identifier for an energy item.

#define **ITEM_TYPE_MIRROR** 2

The type identifier for a mirror.

#define **ITEM_SCORE** 1.5f

The added score given by a score item.

#define **ITEM_ENERGY** 5.0f

The added energy given by an energy item.

#define **ITEM_MIRROR_DURATION** 15

The length a mirror lasts once activated.

#define **NUM_ITEMS** 10

The number of items in the game.

#define **MP_SPEED** 0.01f

The maximum speed at which an item may be picked up.

#define **MP_RADIUS** 0.05f

The maximum distance from which an item may be picked up.

#define **MP_ROTATION_ANGLE** 0.707106f

(rad) Rotation of satellite needed to pick up item (cos(90/2))

#define **LIGHT_SWITCH_PERIOD** 60

The light switches this number of seconds after the first flip in the 2D/3D versions of the game.

#define **LIGHT_SPEED** .025f

The light moves at this speed (in m/s) during the Alliance portion of the game.

#define **LIGHT_WIDTH** .8

The width of the area that is not dark. Note that this includes the grey zone.

#define **LIGHT_GREY_WIDTH** .2

The width of the grey zone in the 2D/3D versions. The width of each grey zone in Alliance is LIGHT_GREY_WIDTH/2.

#define **DISABLE_CAMERA_TIME** 3

The camera is disabled for this many seconds after taking and uploading pictures.

#define **CAMERA_DEFAULT_MEMORY** 2

The number of memory slots an unmodified camera has.

#define **CAMERA_MAX_MEMORY** 4

The number of memory slots the camera may have at a maximum.

#define **PHOTO_MIN_DISTANCE** 0.5

The minimum distance the sphere may be from the target of its photograph.

#define **PROP_ALLOWED_SECONDS** 60.0f

Total time in thruster-seconds allowed per user. Full tank ~500 seconds.

#define **MAX_ENERGY** 5.0f

Energy capacity.

#define **STARTING_ENERGY MAX_ENERGY**

Starting energy.

#define **ENERGY_GAIN_RATE** 0.5f

Energy gained per second.

#define **ENERGY_COST_TAKE_PICTURE** 1.0f

The energy cost to take a picture.

#define **ENERGY_COST_GET_OTHER_ENERGY** 0.1f

The energy cost to determine how much energy your opponent has.

#define **ENERGY_COST_GET_PIC_POINTS** 0.1f

The energy cost to determine how many points taking a picture right now would be worth, should you choose to take it.

#define **ENERGY_COST_UPLOAD_PICTURES** 1.0f

The energy cost to upload pictures.

#define **ENERGY_COST_THRUSTERS** (0.0005f)*(.01f)*(PROP_ALLOWED_SECONDS)

The energy cost to use one second of fuel.

#define **START_SCORE** 0.0f

Your score upon starting the game.

#define **ZONE_pX** 0.64f

The highest X coordinate in bounds.

#define **ZONE_pY** 0.80f

The highest Y coordinate in bounds.

#define **ZONE_pZ** 0.64f

The highest Z coordinate in bounds.

#define **ZONE_nX -ZONE_pX**

The lowest X coordinate in bounds.

#define **ZONE_nY -ZONE_pY**

The lowest Y coordinate in bounds.

#define **ZONE_nZ -ZONE_pZ**

The lowest Z coordinate in bounds.

Variables

const float **EARTH** [3] = {0.0f, 0.0f, 1.0f}
  Contains the attitude towards Earth.

const float **ITEM_LOC** [**NUM_ITEMS**][3]
  Array that outlines the locations of each item.

const int **ITEM_TYPES** [**NUM_ITEMS**]
  Array that outlines the types of each item.

const float **limits** [3] = {**ZONE_pX**,**ZONE_pY**,**ZONE_pZ**}
  The limits of the interaction zone.

Detailed Description

A list of constants used in the ZR program.

Variable Documentation

const float EARTH[3] = {0.0f, 0.0f, 1.0f}

Contains the attitude towards Earth.

The satellite's attidude must be within MAX_FACING_ANGLE to this attitude

const float ITEM_LOC[NUM_ITEMS][3]

Initial value:

```
{
  { 0.3, 0.0, 0.0},
  {-0.3, 0.0, 0.0},
  { 0.0, 0.3, 0.0},
  { 0.0,-0.3, 0.0},
  { 0.6, 0.4, 0.6},
  { 0.6, 0.4,-0.6},
  {-0.6, 0.4, 0.6},
  {-0.6, 0.4,-0.6},
  {-0.6, 0.0,-0.6},
  { 0.6, 0.0, 0.6}
}
```

Array that outlines the locations of each item.

Usage: ITEM_LOC[int ItemID] Each element is an array of three floats for the XYZ coordinates.

const int ITEM_TYPES[NUM_ITEMS]

Initial value:

```
{
  ITEM_TYPE_ADD_ENERGY,
  ITEM_TYPE_ADD_ENERGY,
  ITEM_TYPE_ADD_ENERGY,
  ITEM_TYPE_ADD_ENERGY,
  ITEM_TYPE_ADD_SCORE,
  ITEM_TYPE_ADD_SCORE,
  ITEM_TYPE_ADD_SCORE,
  ITEM_TYPE_ADD_SCORE,
  ITEM_TYPE_MIRROR,
  ITEM_TYPE_MIRROR
}
```

Array that outlines the types of each item.

Usage: ITEM_TYPES[int ItemID] Each element is an integer that refers to one of the previously defined item types.

ZRGame.h File Reference

Contains documentation of functions specific to the player side of the game.

```
#include "pads.h"
#include "Constants.h"
#include "spheres_constants.h"
#include "ZR_API.h"
#include "ZRGameInternal.h"
```

Classes

class **ZeroRoboticsGame**

The class of the game object that you will use.

Detailed Description

Contains documentation of functions specific to the player side of the game.

Use this documentation to learn about using the API functions available in the challenge. General API operations for Zero Robotics are available under ZR_API.h.

Access members of this file by using the syntax "game.functionname(inputs);"

Index

INDEX