Introduction

This is the api reference. All methods and fields listed here are public. Notice that *antenna* refers to the same thing as *radar*.

Contents

Interfaces

AsteroidTracker – Interface for your program, gives assignments to radar units based on asteroid locations and radar statuses.

Objects

Asteroid – The unchanging properties of an asteroid.

ElectromagneticField – Electric and magentic fields at a point.

Radar – The unchanging properties of a radar unit.

RadarAssignment – The current assignment given to a radar unit.

RadarStatus – The current physical status of the radar unit, is calculated from RadarAssignment.

Reception –Contains signal and noise values for a radar meassurement.

SphericalAngle – 3d direction given in spherical coordinates.

Static methods

BeamModel – Contains method for calculations conserning electromagnetic fields.

Scoring – calculates the score based on meassurements and power consumption.

AsteroidTrackerVisV1 – An example program test run.

Interfaces

AsteroidTracker

Back to contents

Methods

void *initialize*(List<Radar> radars, List<Asteroid> asteroids)
This method should setup your program making it ready to deliver assignments to the radar units.

List<RadarAssignment> *assignAntennas*();
This method should return the radar assignments for the next second. This will be called once for each second during the entire time period. Must return one RadarAssignment per radar.

Objects

Asteroid

Back to contents

This contains all you need to know about this asteroid, including exact travel path, how much score you get from it (s*cientificValue*) and how easy it is to get score from it (*reflectionArea* and *reflectionCoefficient*).

Fields

List<Vector3d> *trajectory*
List of positions for this asteroid, second for second. Null menas that the asteroid is not visible at that time.

double *scientificValue*
Scoring multiplier for hitting this asteroid.

double *reflectionArea*
Size of the asteroid, multiplier for received signal strength.

double *reflectionCoefficient*
The reflectivity of the asteroid, multiplier for received signal strength.

ElectromagneticField

Back to contents

This is a set of two vectors representing the electric and magnetic fields at a point.

Fields

Vector3d *electricField*
The electric field at this location.

Vector3d *magneticField*
The magnetic field at this location.

Constructors

***ElectromagneticField*()**
Creates a ElectromagneticField with values zero.

***ElectromagneticField*(Vector3d electricField,Vector3d magneticField)**
Creates a ElectromagneticField object with the given values

Methods

void *add*(ElectromagneticField)
Adds another electromagnetic field to this one.

double *power*()
Returns the power density of the electromagnetic field at this location. The power is proportional to the square of the electromagnetic field density, meaning that using more radar units for the same

asteroid will grant more power output for the same power input. In essence, 2 combined units will effectively transmit as 4 individual units, 3 as 9 etc.

Radar

Back to contents

Contains the specifications of a radar unit. Tells us where it is, how vulnerable it is, the size of its main beam, how long it takes for it to change targets and its maximum power output.

Fields

Vector3d *position*
3 dimensional position of radar unit

double *safeZoneRadius*
How close other radar beams can get before we start risking damage to this radar.

double *beamRadius*
The beam is modelled as a cylinder with this radius, shouldn't overlap with safeZone radius of other units or we risk damaging the radars. AsteroidTrackerVisV1 has an already built check for this.

double *reallocationTime*
The downtime incurred each time you need a radar unit to target another asteroid. The radar also requires more energy during this time.

double *maxPower*
The maximum power output of this radar unit.

RadarAssignment

Contains orders given to a radar unit. Specifically it has information on where to aim and how much power to use.

Back to contents

Fields

double *power*
Poweroutput of the radar. Is directly proportional to signal and noise strength created by this radar unit. Must be set to 0 when sleeping or reallocating.

int *asteroidIndex*
The asteroid this radar unit targets, must be set to -1 when sleeping or reallocating.

Constructors

***RadarAssignment*()**
Defaults power to 0 and target to -1.

RadarStatus

Contains information about the current state of the radar unit.

Back to contents

Fields

Vector3d *position*
Radar position at this time, since our radars don't move you can ignore this field.

double *phase*
The relative phase of this radar unit. Currently AsteroidTrackerVisV1 always sets it to align at the targeted asteroid to maximize power density.

double *power*
The amount of power you assigned this radar. Increases signal strength and energy consumption.

SphericalAngle *direction*
The direction the radar is pointing at. AsteroidTrackerVisV1 always points it directly at the targeted asteroid.

Reception

Back to contents

double *signal*
The power of the reflected field from the targeted asteroid. Increases score.

double *noise*
The power of the noise from other radars affecting this meassurement. Reduces score.

SphericalAngle

Back to contents

A spherical angle is a set of two angles representing a direction in three dimensional space. Read more

Fields

double *elevation*
The angle between the direction and the ground, so 0 means parallel to the ground, π/2 means straight up and -π/2 means straight down. Anything below 0 means that you target the ground.

double *azimuth*
This angle is the normal horizonthal angle, 0 is north, π/2 is west, π is south and π*3/2 is east.

Constructors

***SphericalAngle*()**
Creates a SphericalAngle pointing straight up.

***SphericalAngle*(double elevation,double azimuth)**
Creates a SphericalAngle with the given initial values.

Static Methods

AsteroidTrackerVisV1

Back to contents

double *evaluateTestcase*(List<Radar> radars, List<Asteroid> asteroids, AsteroidTracker asteroidTracker)
This is a sample method to test the performance of your AsteroidTracker program. Returns the total score for this run.

BeamModel

Back to contents

Here we have all the methods which models the physics of the system.

Constants

final double *WAVELENGTH*
The wavelength of the transmitted electromagnetic fields, set to 0.01 meters.

Methods

double *relativeDistancePhaseShift*(Vector3d antennaPosition, Vector3d position)
Calculates the relative phase shift incurred due to radar unit offset.

ElectromagneticField *normalizedPolarization*(Vector3d propagationDirection)
Calculates the polarization of the electromagnetic field in a given direction.

ElectromagneticField *calculateElectromagneticField*(RadarStatus radar, Vector3d position)
Calculates the electromagnetic field from a radar unit at a given position, use this to calculate the field strength at asteroids and other radars. Notice that the program uses different models for calculations depending on distance, which means that the formula for calculating noise will be drastically different from the formula for calculating signal strength.

ElectromagneticField *calculateArrayField*(List<RadarStatus> radars, Vector3d position)
Calculates the electromagnetic field contributes for the entire group of radar units at a specific position.

Vector3d *matrixVectorMul*(Matrix3d rotMat, Vector3d v)
Multiplies a matrix with a vector, useful for rotating vectors. You probably don't need this.

boolean *doesRadarBeamIntersect*(Radar transmitter, SphericalAngle transmitterLookingAngle, Radar receiver)
Checks if a radar beam is too close to another radar. Since we don't want to risk any damage this should return false for any position.

Vector3d *sphericalAngleToDirection*(SphericalAngle sphericalAngle)
Returns the unit vector corresponding a given spherical angle.

SphericalAngle *directionToSphericalAngle*(Vector3d direction)
Returns the spherical angle corresponding to a given direction.

Scoring

Back to contents

double *radarWallPower*(boolean isMoving, double transmitterPower)
Returns the power consumption for a radar dish at this moment.

double *energyScorePenalty*(double totalEnergy)
Returns the total energy costs for a given amount of energy.

double *knowledgeValue*(List< List<Reception> > receptions)
Returns the total value of a inputed list of data.