Mechmania Reference Manual

Generated by Doxygen 1.3.9.1

Sat Oct 21 12:47:24 2006

Contents

Chapter 1

Mechmania Main Page

Each team starts out with Pirate Boats, Rum Runners, and a pier. Your objective is to have the most software at then end of the game. Games last for 300 turns.

Pirate Boats are the main type of boat. They can attack with cannons, board and take over ships, and steal software. To steal software you go to the enemy pier, load up as much as you can carry, and then take it back to your own pier.

Rum Runners cannot attack or pick up software, but they can heal other boats that they are next to. A Rum Runner cannot heal itself and they cannot be boarded. They are faster than Pirate Boats.

If you collide with an enemy boat you will take damage. The angle of the collision affects the damage done. You do not take damage if you collide with your own boats.

Chapter 2

Mechmania Hierarchical Index

2.1 Mechmania Class Hierarchy

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

Chapter 3

Mechmania Class Index

3.1 Mechmania Class List

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

Chapter 4

Mechmania File Index

4.1 Mechmania File List

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

Chapter 5

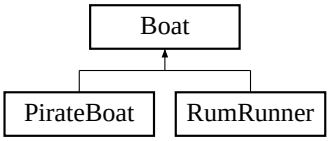
Mechmania Class Documentation

5.1 Boat Class Reference

The "abstract" boat class.

`#include <Model.h>`

Inheritance diagram for Boat::



Public Member Functions

- void **setOrdersQueue** (OrdersQueue *oq)

  Ignore this function.

- void **turnAbsolute** (double **goalHeading**)

  Sets a new heading for the boat.

- void **stopTurning** ()

  Sets the boats goal heading to the current heading.

- void **velocity** (double goalVelocity)

  Sets the current velocity.

Public Attributes

- int **player**

  The id of the player who owns this boat.

- int **id**

The id of the boat. When storing boat information across ticks, in a std::map for example, this is the key you should use as pointers will not be valid across function calls.

- **Coord location**

  The current location of the boat.

- double **heading**

  A degree, 0 to 360, containing the boats current heading.

- double **goalHeading**

  A degree, 0 to 360, containing the direction the boat. is moving towards.

- double **health**

  The boats current health.

- double **speed**

  The boats speed.

- std::vector< int > **collidesWith**

  The id of all the boats that this boat is currently colliding with.

Protected Member Functions

- virtual void **forceRTTI** ()

Protected Attributes

- OrdersQueue * **ordersQueue**

5.1.1 Detailed Description

The "abstract" boat class.

The base class that PirateBoats and RumRunners inherit from. Though it doesn't have any pure virtual functions you should not actually have any instances of this clas.

5.1.2 Member Function Documentation

5.1.2.1 void Boat::turnAbsolute (double *goalHeading*) [inline]

Sets a new heading for the boat.

Parameters:

goalHeading the new heading in degrees.

5.1.2.2 void Boat::velocity (double *goalVelocity*) [inline]

Sets the current velocity.

Parameters:
goalVelocity the target velocity.

5.1.3 Member Data Documentation

5.1.3.1 OrdersQueue* Boat::ordersQueue [protected]

The order queue associated with this boat. This is for communication with the server and you shouldn't need to use it

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

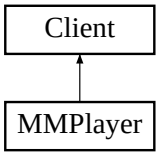
- Model.h

5.2 Client Class Reference

This is the class you will fill in.

`#include <Client.h>`

Inheritance diagram for Client::



Public Member Functions

- **Client** (int pNumber)
- virtual void **init** ()

  This function will be called at the beginning of the game.

- virtual void **tick** ()=0

  This function is called every 'frame' of the game.

- void **updateState** (const **State** &newState)

Protected Attributes

- std::vector< **Boat** * > **allBoats**
- std::vector< **Boat** * > **myBoats**
- std::vector< **Boat** * > **enemyBoats**
- std::vector< **PirateBoat** * > **allPBoats**
- std::vector< **PirateBoat** * > **myPBoats**
- std::vector< **PirateBoat** * > **enemyPBoats**
- **RumRunner** * **myRumRunner**
- **RumRunner** * **enemyRumRunner**
- std::vector< **Player** * > **players**
- **Player** * **me**
- **Player** * **enemy**
- **Game** * **game**

5.2.1 Detailed Description

This is the class you will fill in.

This is the class you will be subclassing to create your AI.

5.2.2 Member Function Documentation

5.2.2.1 virtual void Client::init () [inline, virtual]

This function will be called at the beginning of the game.
You can implement this function if you want to do any sort of pre-game setup.

5.2.2.2 virtual void Client::tick () [pure virtual]

This function is called every 'frame' of the game.
This is where you should be writing all of your code.

5.2.2.3 void Client::updateState (const State & *newState*)

Update the current state of the game

5.2.3 Member Data Documentation

5.2.3.1 std::vector<Boat*> Client::allBoats [protected]

Contains every boat in the game

5.2.3.2 std::vector<PirateBoat*> Client::allPBoats [protected]

Contains every **PirateBoat**(p. 17) in the game

5.2.3.3 std::vector<Boat*> Client::enemyBoats [protected]

Contains all boats owned by the enemy

5.2.3.4 std::vector<PirateBoat*> Client::enemyPBoats [protected]

Contains all PirateBoats owned by the enemy

5.2.3.5 RumRunner* Client::enemyRumRunner [protected]

Points to the **RumRunner**(p. 21) owned by the enemy

5.2.3.6 std::vector<Boat*> Client::myBoats [protected]

Contains all boats owned by the player

5.2.3.7 std::vector<PirateBoat*> Client::myPBoats [protected]

Contains all PirateBoats owned by the player

5.2.3.8 RumRunner* Client::myRumRunner [protected]

Points to the **RumRunner**(p. 21) owned by the player

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

- Client.h
- Client.cc

5.3 Coord Class Reference

Stores a 2D double coordinate.

`#include <Model.h>`

Public Member Functions

- **Coord** (double x, double y)

Public Attributes

- double **x**
- double **y**

5.3.1 Detailed Description

Stores a 2D double coordinate.

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

- Model.h

5.4 Game Class Reference

Stores some game information.

`#include <Model.h>`

Public Attributes

- int **tm**

  Current time.

- int **maxTm**

  Time when the game ends.

- int **winner**

  The player id of the winner.

- int **status**

5.4.1 Detailed Description

Stores some game information.

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

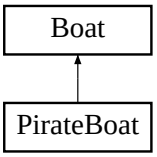
- Model.h

5.5 PirateBoat Class Reference

The main boat type.

`#include <Model.h>`

Inheritance diagram for PirateBoat::



Public Member Functions

- void **load** (double amount)
  Loads software onto your boat. You can only do this when you are at a pier.
- void **unload** (double amount)
  Unloads software from your boat. You can only do this when you are at a pier.
- void **board** (int bid)
  Attempts to board another ship. You can only do this when the ship has no health left.
- void **fireLeft** ()
  Fires the left cannon.
- void **fireRight** ()
  Fires the right cannon.

Public Attributes

- double **software**
  The amount of software the boat currently has.
- std::vector< double > **fireAngle**
  The direction your cannons are pointed at.
- std::vector< double > **fireDamage**
  The amount of damage your cannons do.
- std::vector< double > **fireRange**
  The range of your cannons.
- std::vector< double > **fireWait**
  How many turns you have to wait until you can fire the cannon again.
- std::vector< bool > **fire**
  True if the cannon has just fired.

Static Public Attributes

- const unsigned int **LEFT_CANNON** = 0

  The vector index of the left cannon.

- const unsigned int **RIGHT_CANNON** = 1

  The vector index of the right cannon.

- const double **MAX_SPEED** = 2.0

  The maximum possible speed.

- const double **MAX_HEALTH** = 100.0

  The maximum possible health.

5.5.1 Detailed Description

The main boat type.

A PirateBoat is the main boat in your fleet. It has a cannon it can attack with, and has the capability to board and take over another ship. You can steal software from the opposing team by loading it into a PirateBoat and taking it back to your base.

5.5.2 Member Function Documentation

5.5.2.1 void PirateBoat::board (int *bid*) [inline]

Attempts to board another ship. You can only do this when the ship has no health left.

Parameters:

bid the id of the boat to board.

5.5.2.2 void PirateBoat::load (double *amount*) [inline]

Loads software onto your boat. You can only do this when you are at a pier.

Parameters:

amount the amount of software to load.

5.5.2.3 void PirateBoat::unload (double *amount*) [inline]

Unloads software from your boat. You can only do this when you are at a pier.

Parameters:

amount the amount of software to unload.

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

- Model.h

5.6 Player Class Reference

Stores information pertaining to the player.

`#include <Model.h>`

Public Member Functions

- void **setOrdersQueue** (OrdersQueue *oq)

  Ignore this function.

- void **trashTalk** (const char *text)

  Ask the server to display a chat message.

- void **setName** (const char ***name**)

  Set the team name.

Public Attributes

- int **id**

  The id of the player, used to identify their boats.

- double **software**

  The amount of software the player has obtained.

- **Coord pier**

  The coordinates of the player pier.

- std::vector< int > **color**

  The team colors.

- std::string **name**

  The team name.

- OrdersQueue * **ordersQueue**

5.6.1 Detailed Description

Stores information pertaining to the player.

Stores information pertaining to the player.

5.6.2 Member Function Documentation

5.6.2.1 void Player::setName (const char * *name*) `[inline]`

Set the team name.

You should call this once, in your init function.

5.6.3 Member Data Documentation

5.6.3.1 OrdersQueue* Player::ordersQueue

The order queue associated with this player. This is for communication with the server and you shouldn't need to use it.

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

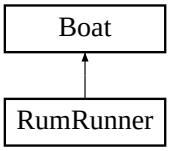
- Model.h

5.7 RumRunner Class Reference

A medic boat.

`#include <Model.h>`

Inheritance diagram for RumRunner::



Public Member Functions

- void **heal** (int bid)

  Heals a friendly boat. You can only do this when you are next to the boat.

Static Public Attributes

- const double **MAX_SPEED** = 3.0

  The maximum possible speed.

- const double **MAX_HEALTH** = 100.0

  The maximum possible health.

5.7.1 Detailed Description

A medic boat.

The RumRunner is the "medic" of your fleet. It has the capability to heal other boats. It is also faster than the PirateBoats however it does not have a cannon and cannot steal software.

5.7.2 Member Function Documentation

5.7.2.1 void RumRunner::heal (int *bid*) `[inline]`

Heals a friendly boat. You can only do this when you are next to the boat.

Parameters:

bid The id of the boat you want to heal

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

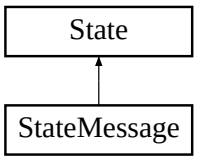
- Model.h

5.8 State Class Reference

Stores the current state of the game.

`#include <Model.h>`

Inheritance diagram for State::



Public Attributes

- **Game** * **game**
- std::vector< **Player** * > **players**
- std::vector< **Boat** * > **boats**

5.8.1 Detailed Description

Stores the current state of the game.

Stores the current state of the game, including players, boats, time remaining, and current score.

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

- Model.h

Chapter 6

Mechmania File Documentation

6.1 Geom.h File Reference

This contains some helper functions for basic geometry.

```
#include "Model.h"
#include <cmath>
```

Defines

- #define **PI** 3.14159

Functions

- float **rad2deg** (float radVal)

  converts a radian value to a degree

- float **deg2rad** (float degVal)

  converts a degree to its radian value

- float **angleBetween** (const **Coord** &src, const **Coord** &dest)

  finds the angle between two coordinates

- float **distance** (const **Coord** &p1, const **Coord** &p2)

  finds the distance between two points

6.1.1 Detailed Description

This contains some helper functions for basic geometry.

6.1.2 Function Documentation

6.1.2.1 float angleBetween (const Coord & *src*, const Coord & *dest*)

finds the angle between two coordinates

Parameters:

src

dest

Returns:

the angle between src and dest

6.1.2.2 float deg2rad (float *degVal*)

converts a degree to its radian value

Parameters:

degVal a degree

Returns:

a radian

6.1.2.3 float distance (const Coord & *p1*, const Coord & *p2*)

finds the distance between two points

Parameters:

p1

p2

Returns:

the distance between p1 and p2

6.1.2.4 float rad2deg (float *radVal*)

converts a radian value to a degree

Parameters:

radVal a radian

Returns:

a degree

Index