

README for *Pirates!* and *Pirate Fleet* API

Alan Dorin

TABLE OF CONTENTS

0.0	Capabilities specific to assignment 2
0.1	Ship heading
0.2	Ship signal flags
0.2	Tips for use
1.0	Controller naming convention
2.0	Controller computes next action
3.0	Controller queries ship's current state
4.0	Controller queries environment's current state
5.0	Compiling your controller
6.0	Framework parameters
7.0	Game controls
8.0	Game view explanation



0.0 CAPABILITIES SPECIFIC TO ASSIGNMENT 2

To complete your assignment 2 for FIT3094 AI For Gaming as detailed in the online submission instructions, you will need to write an AI controller class to pilot a fleet of ships around Rectangular bay in a *coordinated* fashion against a single powerful enemy ship. The enemy ship is depicted in a different colour to your fleet for you (the human) to see which ship is which. (The ships themselves can't see this colour.)

This file provides details of the API you will need to use to write your controller for assignments 1 & 2. Details of specific relevance to assignment 2 have been added at the front of this document in sections 0.0 to 0.3.

0.1 SHIP HEADING

The enemy ship has cannons that can fire to its port (left), starboard (right) *and* from its bow (front) simultaneously. The enemy ship cannot fire from its stern (back). It is therefore important for you to work out which direction the enemy ship is facing before you approach it!

Your ship can determine the direction of heading of any ship it has seen whether friend or foe, by calling a routine:

Globals::NeighbourDirection getShipAheadDirection (long relativeX, long relativeY)

If your ship is able to see into the requested location *and* there is a ship at the location specified, its heading direction will be returned. Otherwise, a random direction will be returned.

0.2 SHIP SIGNAL FLAGS

Every ship, including your own, has three coloured signal flags on board that may be raised or lowered by setting their visibility to `true` or `false`. These flags are:

`Globals::flagCyan`, `Globals::flagMagenta`, `Globals::flagYellow`

Your ship can determine the flags set on any ship it has seen by calling a routine:

bool getShipSignalFlag (long relativeX, long relativeY, Globals::SignalFlag flagColour)

If your ship is able to see into the requested location *and* there is a ship at the location specified, its flag setting for the specified colour will be returned. Otherwise, a random result will be returned.

You can determine the current setting for flags on your own ship:

bool getSignalFlag(Globals::SignalFlag flagColour)

You can set the flags on your own ship using the routine:

void setSignalFlag(Globals::SignalFlag flagColour, bool newValue)

0.3 TIPS FOR USE

I suggest, to avoid accidentally using random results in your algorithms, call the routines like this:

```
// First check the relX and relY are in visual range
// then...
if isShipAt(relX, relY)
{
    otherDirection = getShipAheadDirection(relX, relY);
    otherShowCyan   = getShipSignalFlag(relX, relY, Globals::flagCyan);
}
```

In this way you are sure you are getting a direction of a ship and whether or not it's flags are raised where there is actually a ship present. I.e. the two new methods are *not* a substitute for calling `isShipAt()`.

You can use the signal flags to communicate amongst your fleet¹. For instance, perhaps one combination of flags indicates to other ships that you are friendly. Perhaps a flag indicates to friendly ships that you have spotted an enemy ship or that you are badly damaged. It is up to you how you use these flags. Remember that the enemy ship can see your flags and has flags of its own. The enemy could possibly mimic your flag signals or anticipate your actions if it sees your flags.



¹ This is the only kind of inter-ship communication that is legal for assignment 2. Shared variables or data-structures (e.g. globals or static class members) are *not* permitted!

1.0 CONTROLLER NAMING CONVENTION

You need to write a controller class that is derived from the provided base class `AIController`.

Your controller class **MUST** be called `AIController##` where `##` is a two-digit code number allocated to you by the lecturer. Therefore you must write a class:

```
class AIController## : public AIController { ... };
```

and keep it in the files called `AIController##.h` and `AIController##.cpp`

An example controller class called `StudentAIControllerExample` is given for you in files `StudentAIControllerExample.h` and `StudentAIControllerExample.cpp`

For assignment 1, your controller pilots your single ship against multiple opponents piloted by controllers written by the other students in the class.

For assignment 2, your controller pilots your *fleet* of ships against a single powerful opponent ship that has (for instance) multiple guns and a greater ability to withstand attack. Each of your ships is operated by a different instance of the same controller. *Controllers are not allowed to share variables.*

2.0 CONTROLLER COMPUTES NEXT ACTION

The provided example `StudentAIControllerExample` gives a very simple controller and some sample behaviours coded to demonstrate how to write your own.

At the very least, you need to provide your controller with a method specified using this header:

```
Globals::ShipAction computeNextAction(void);
```

This method is called by the provided framework to ask your ship's controller what it wants to do at each time step.

Available actions for this method to return are:

`Globals::actionMoveAhead` - move the ship one square in the direction it is facing

`Globals::actionTurnLeft` - turn the ship to the left by 90 degrees

`Globals::actionTurnRight` - turn the ship to the right by 90 degrees

`Globals::actionCollectGold` - attempt to collect gold from the current location

`Globals::actionFireCannonAhead` - fire the cannon straight ahead

`Globals::actionPass` - do nothing (wait out this time step)

There are no other actions you can return. You must choose from this list at each time step based on what your ship (i) knows about itself (see section 3.0) and what it has learned about its environment (see section 4.0) by querying the framework. Your ship may build up a lot of knowledge about the environment as the game is played. For instance, it may build a map of the bay, the location of gold markers and searched grid cells so that it can be sure to cover the area properly.

3.0 CONTROLLER QUERIES SHIP'S CURRENT STATE

Here are the methods your pirate ship controller can call to learn about itself:

long getShipPositionX(void)

long getShipPositionY(void)

These methods get the current ship's X and Y position in the Sea grid. Grid labels are drawn for you around the edges of the world. (Run the game to see them.) The bottom left hand corner grid cell, a rock, is (0,0)

Globals::NeighbourDirection getShipAheadDirection(void)

The ship heading is obtained by calling this method which returns either `Globals::neighbourUp`, `Globals::neighbourRight`, `Globals::neighbourDown`, `Globals::neighbourLeft` referring to the direction the ship is pointing on the screen.

long getShipVisionRange(void)

The distance in grid cells that the ship can see along the x and y axes from its current location is given by this method.

long getShipCannonRange(void)

The distance in grid cells that the ship can shoot its cannon straight ahead along the x and y axes from its current location is given by this method.

bool justDamagedShip(void)

If the ship was just damaged in its previous time step, this flag will be true, otherwise it will be false. A ship can be damaged by:

- a hit from a cannon ball fired by another ship
- running aground on rocks
- running into another ship

Note: If a `shipA` is rammed by another `shipB` and `shipA` attempted to move out of the way but accidentally sailed in a direction towards rocks, `shipA` sinks *immediately*! The combined force of sailing onto rocks and being shoved onto rocks by `shipB` ensures there is no chance of surviving this encounter. Beware!

long getShipMaxDamage(void)

The maximum amount of damage that this ship can sustain before sinking.

long getShipDamage(void)

The amount of damage that this ship has so far sustained.

(*NB) Ramming immediately increases the sustained damage to above the maximum sustainable damage (see note under `justDamagedShip()`)

bool getShipLastActionSuccess(void)

This method reports whether or not the last action was successful as follows:

`Globals::actionMoveAhead`

Successful if destination cell is not occupied by a rock or ship.

`Globals::actionTurnLeft` and `Globals::actionTurnRight`

Turning is always successful.

`Globals::actionCollectGold`

successful if there is a gold marker here *and* the marker has some gold left beneath it.

`Globals::actionFireCannonAhead`

Successful if another ship was hit. A hit occurs if another ship is (i) straight ahead *and* (ii) within cannon range *and* (iii) the trajectory was not interrupted by rocks. If the ship attempts to fire off the edge of the screen, usually this will be blocked by rocks. In cases where rocks do not obstruct the cannonball, the ball will reappear on the far side of the sea/screen. I.e the sea wraps around like a torus.

`Globals::actionPass`

Doing nothing is always successful — but it won't help you avoid cannonballs or collect gold ;-)

`long getShipRewardStored(void)`

This method reports on the amount of gold that has been stored onboard the ship so far.

4.0 CONTROLLER QUERIES ENVIRONMENT'S CURRENT STATE

Here are the methods your pirate ship controller can call to learn about its local environment:

`bool isGoldMarkerHere(void)`

Method reports whether or not a gold marker is at the ship's current location. Gold markers are left in place after their gold is collected so a marker is *not* guaranteed to have gold beneath it. However, gold is never found at any location other than one where there is a gold marker.

`bool isShipAt(long relativeX, long relativeY)`

Method reports if there is another ship at the relative grid location (relativeX, relativeY) where relativeX/Y specifies the distance from the current ship's position along the x and y axes in grid cells. For instance, if the ship is at (2,2) then (1, -2) refers to location (3, 0).

If the ship is requesting to see to a position over the edge of the sea it can look past the edge and back onto the other side of the board... i.e the sea wraps around like a torus.

If a request is made to see beyond the visual range of the ship reported by `getShipVisionRange()` then the result will always be false, regardless of whether or not a ship is present at the requested location.

`bool isRockAt(long relativeX, long relativeY)`

Method reports if there is a rock at the relative grid location (relativeX, relativeY) where relativeX/Y specifies the distance from the current ship's position along the x and y axes in grid cells. For instance, if the ship is at (2,2) then (1, -2) refers to location (3, 0).

If the ship is requesting to see to a position over the edge of the sea it can look past the edge and back onto the other side of the board... i.e the sea wraps around like a torus.

If a request is made to see beyond the visual range of the ship reported by `getShipVisionRange()` then the result will always be false, regardless of whether or not a rock is present at the requested location.

5.0 COMPILING YOUR CONTROLLER

To try out your controller you will need to compile it with all of the code that the lecturer has provided. You will also need to link in the libraries for OpenGL (-lGL) and GLUT (-lglut) and locate the headers for these libraries.

A sample `Makefile` has been included for you to use. The `Makefile` coordinates the compilation of multiple files under UNIX.

Where you see the text `StudentAIControllerExample` in the `Makefile`, replace it with your own controller class name, `AIController##` **Do NOT change any other parts of this file if you don't know what you are doing! You could easily delete all of your work if you change something incorrectly. Ask the lecturer for help if you need it and be sure to keep a BACKUP of your code in a safe location away from the `Makefile` when you compile, especially when you type “`make clean`” (see below) as this deletes files.**

In the file `Sea.cpp` provided in the framework, do a search for the term `StudentAIControllerExample`. This appears twice. Once as a `#include`, and once in the `Sea` constructor. Replace both instances of the text with the name of your own controller, `AIController##`

To build the project under UNIX be sure the `Makefile` is in the same directory as all of the project source code (but not your back up code), then just type `make` at the UNIX prompt. You may need to specify the paths to the OpenGL and GLUT libraries and header files based on their locations on your own machine. You can do this in the `Makefile` where you see the paths specified.

If you change some files and want to do a recompilation from scratch, type `Make clean` This will remove all the `.o` files from the directory, allowing you then to do a standard `Make` from scratch.

6.0 FRAMEWORK PARAMETERS

Your controller writing will *not* involve changing the framework code provided by the lecturer in any way apart from that discussed in section 5.0. When your code is assessed and played against the other class members' code the standard, unedited framework will be used... so don't change this or your ship may not work at all in the original framework. If this happens you will not receive any marks at all!

At the top of file `main.cpp` is a constructor call for class `gGlobal`. This sets up some basic parameters for the game. You can edit these parameters whilst you are testing your controller by typing different numbers into the constructor call. In order, the parameters are:

- gold:** The number of gold deposits placed around the sea.
- ships:** The initial number of ships placed in the sea. (All ships have the same controller specified in `StudentAIControllerExample` in this version of the code).



grid-size: The dimensions of the (square) sea counted in grid cells.

rocks: The number of rocks placed in the middle of the sea. Rocks always surround the boundary of the sea. Be sure also not to specify in this parameter more rocks than there are locations inside your sea or the code will report an error when it is executed.

steps: The length of a game in time steps (updates).

repeats: The number of games you want to run automatically in a row.

scenario: The type of game, either `Globals::manyAgainstMany` (assignment 1) or `Globals::oneAgainstMany` (assignment 2).

7.0 GAME CONTROLS

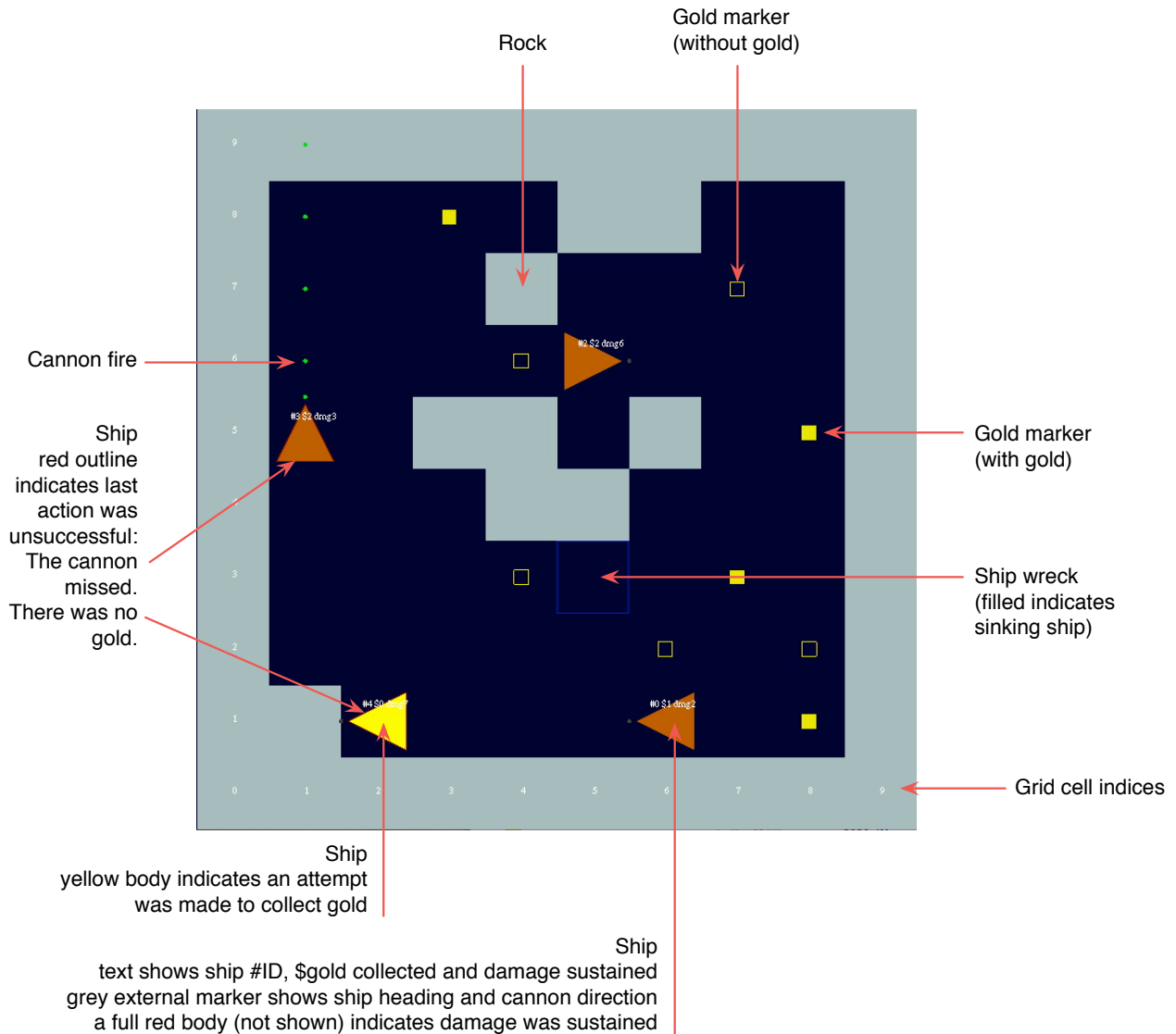
This game runs without human-player input. However, there are a some step-controls provided to assist you in debugging and watching the game unfold slowly. You can operate these by pressing keys on the keyboard as follows:

ESC or q: Exit the game

p: Pause / un-pause the game (the game is in pause mode at start-up by default).

o: One-step the game (increment the game by one update of all ships).

8.0 GAME VIEW EXPLANATION



The console prints out various game statistics during a game at 1000 frame intervals² and at Game Over. Game Over occurs when the number of time steps specified in the global game parameter (section 6.0) is reached, or when all ships have sunk. Here is a sample of the statistics for the first short game out of three in a series:

GAME PARAMETERS

```
sea grid dimensions: 10
number of ships: 4
number of gold boxes: 10
```

Pirates! will run 3 times for 2000 time steps.

```
----- time = 0
id=0, gold=0 damage=0/10
id=1, gold=0 damage=0/10
id=2, gold=0 damage=0/10
id=3, gold=0 damage=0/10
```

² If, for debugging, you want to change this, there is a variable in `GL_routines.cpp` / `animate()` called `frameInterval` that you can set to anything down to 1.

Total reward stored on ships=0, out of 10 possible gold boxes. Mean reward stored=0

----- time = 247

id=0, gold=1 damage=10/10 --- sunk!

id=1, gold=0 damage=10/10 --- sunk!

id=2, gold=2 damage=10/10 --- sunk!

id=3, gold=2 damage=10/10 --- sunk!

Total reward stored on ships=5, out of 10 possible gold boxes. Mean reward stored=1

----- GAME OVER -----

id=0, gold=1 damage=10/10 --- sunk!

id=1, gold=0 damage=10/10 --- sunk!

id=2, gold=2 damage=10/10 --- sunk!

id=3, gold=2 damage=10/10 --- sunk!

--- Ships with 2 gold boxes each ---

id=2, gold=2 damage=10/10 --- sunk!

id=3, gold=2 damage=10/10 --- sunk!

--- Congratulations to the 2 winning ships! ---

Total reward stored on ships=5, out of 10 possible gold boxes.

Mean reward stored=1