

Conquer the Seas Implementation Document

Benson Perry, Matt Dannenberg, Brian Shaginaw

Peer Review Process

Assembled team with roles report

Module	Programmer Name	Due Date	Actual Date Completed
server.py	Benson	3/03/12	- changed to networking.py, incomplete
bg_waves.py	Brian	16/04/12	12/03/12
client.py	Brian	3/03/12	- changed to networking.py, incomplete
message_panel.py	Brian	3/03/12	03/03/12
mousehitbox.py	Brian	3/03/12	01/03/12
upgrade.py	Brian	16/03/12	16/03/12
creditsscreen.py	Brian	16/04/12	10/03/12
introscreen.py	Brian	16/04/12	10/03/12
joinscreen.py	Brian	16/04/12	12/03/12
lobbyscreen.py	Brian	3/03/12	03/03/12
mainscreen.py	Brian	16/04/12	incomplete
screen.py	Brian	3/03/12	28/02/12
action.py	Brian, Matt	3/03/12	26/02/12
unit.py	Matt, Brian	20/03/12	16/03/12
gamescreen.py	Matt, Brian	16/04/12	incomplete
shopscreen.py	Matt, Brian	10/03/12	10/03/12
upgradescreen.py	Matt, Brian	16/03/12	14/03/12
board.py	Matt	20/03/12	18/03/12
constants.py	Matt	10/04/12	incomplete - constantly being updated
defense.py	Matt	20/03/12	23/03/12
offense_panel.py	Matt	4/03/12	15/03/12

Inspection Error List

System: Networking

Subsystem: networking.py

Module Name: networking.py

Moderator: Benson

Inspection Type: Code

Location	Error Description	Error Type	Error Class	Severity
	No ability to send network data (for example, messages) longer than 255 characters	Functionality	M	Minor
20	TODO message remains even after it has been done	Documentation	E	Minor
38	Closed connection could be polled for info which could cause an I/O error	Logic	W	Major
77-82	Bad way to parse for text, can be cleaned up and made more efficient	Standards	W	Minor
83-85	Old code is commented out rather than removed	Documentation	E	Minor
121	Can set a slot to AI or closed even if it has a player in it, instead of kicking player and closing connection first	Functionality	W	Minor
143	Unindented statement allows anyone to start game	Syntax	W	Major
143	Unindented statement only checks to see if everyone is ready if that player is not the host	Syntax	W	Major
36,107, 113, 140	has_key is used although "in" is used in the rest of the code	Standards	W	Minor

Inspection Summary

System: Networking

Subsystem: networking.py

Module Name: networking.py

Moderator: Benson

Inspection Type: Code

MINOR ERRORS				
Error	Missing	Extra	Wrong	Total
Interface	0	0	0	0
Data	0	0	0	0
Logic	0	0	1	1
Input/Output	0	0	0	0
Syntax	0	0	0	0
Performance	0	0	0	0
Human Factors	0	0	0	0
Standards	0	0	2	2
Documentation	0	2	0	2
Other	0	0	0	0

MAJOR ERRORS				
Error	Missing	Extra	Wrong	Total
Interface	0	0	0	0
Data	0	0	0	0
Logic	0	0	1	1
Input/Output	0	0	0	0
Syntax	0	0	2	2
Performance	0	0	0	0
Human Factors	0	0	0	0
Standards	0	0	0	0
Documentation	0	0	0	0
Other	0	0	0	0

Inspection Report

System: Networking

Subsystem: networking.py

Module Name: networking.py

Moderator: Benson

Size of material: 220 lines

Inspection Meeting Number: 1

Inspection Duration: 1hr

Number of Inspectors: 3

Total Preparation Time: 20 minutes

Unit Disposition: Meet

Estimated rework effort: 2 hours

Rework to be completed by: Matt

Reinspection scheduled for: 1 week

Other inspectors: Matt, Brian

Date: March 8, 2012

Additional Comments: There are probably errors that aren't visible just by looking at the code. These will manifest themselves upon testing.

Test Case: mousehitboxes

Mousehitboxes is an interesting test case as it clearly shows the automated testing process of python's unittest package. We use mousehitboxes for any clickable space on the screen - buttons, units, menu options, etc. We needed to test that these boxes would behave as expected when putting them down in all sorts of combinations. Using unittest this is fairly easy to do, as we explained in the Design Document (Section 5.3).

Implementation:

This test case was implemented using the automated unittest package in python. This package allows for writing classes that run whenever the python script they are contained in is run. This means that we wrote a class, TestMouseHitboxes, inside mousehitbox.py. This class is run every time mousehitboxes.py is used (which is frequently throughout the rest of the code of Conquer the Seas).

This is a black-box test case: it does not take into account the code of mousehitboxes.py. It simply tests that hitboxes are correctly applied in all possible real-world scenarios. It positions hitboxes on top of each other, near each other, and in every other possible scenario. For example, the following code tests what happens when two boxes are partially inside one another on the same layer:

```
def test_half_inside(self):  
    with self.assertRaises(AttributeError):  
        self.mh.append((0, 1, 2, 2), self.function(0))  
        self.mh.append((1, 0, 2, 4), self.function(1))
```

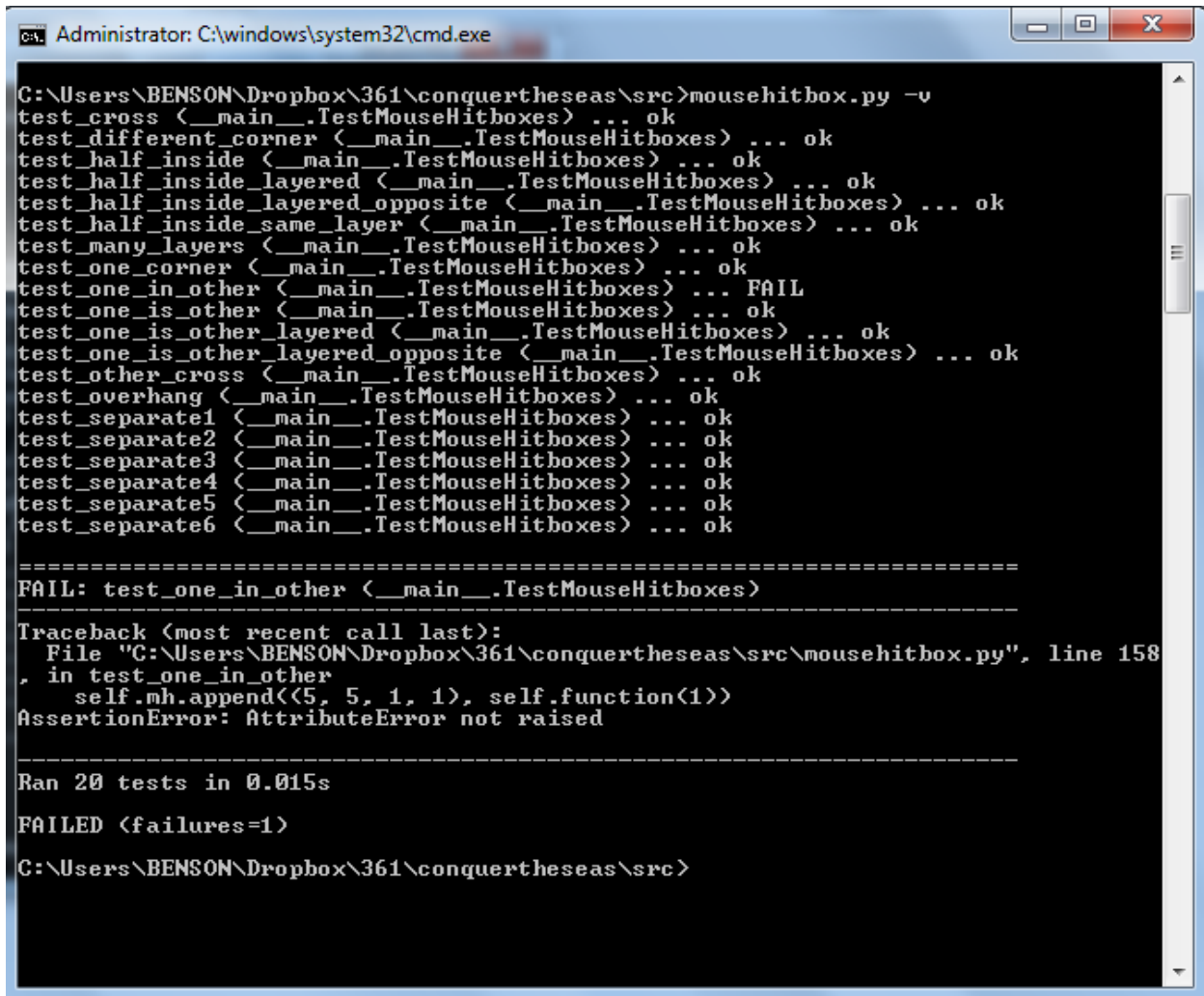
When mousehitboxes is run, this method is called, along with all other methods that begin with "test_". If it raises AttributeError, it will return true, if it does not, it will return false, and the test will fail. This is just one example of inputs and expected outputs. The following table lists all the inputs and expected outputs.

Input and Output

Input	Output
Two hitboxes in the same place, same layer, click simulated in center	AttributeError is raised, true is returned
Two hitboxes in the same place, on different layers, click simulated in center	Top box is clicked on, true is returned
Two hitboxes in same place, layered in different order, click simulated in center	Top box is clicked on, true is returned
Two hitboxes, one partially inside the other, default layer (same layer)	AttributeError is raised, true is returned
Two hitboxes, one partially inside the other, set to same layer	AttributeError is raised, true is returned
Two hitboxes, one partially inside the other, on different layers. Clicks simulated inside each, and on overlapping area	Correct box is clicked on for each case, true is returned each time
Two hitboxes, one partially inside the other, on different layers, opposite order as previous. Clicks simulated inside each, and on overlapping area	Correct box is clicked on for each case, true is returned each time
Two hitboxes, one with its corner inside another on the default level	AttributeError is raised, true is returned
Two hitboxes, one with a different corner inside another on the default level	AttributeError is raised, true is returned
Two hitboxes, one contained entirely within another on the same default layer	AttributeError is raised, true is returned
Four hitboxes, all layered on top of each other, in decreasing size	Correct box is clicked on for each case, true is returned each time
Three hitboxes, on three different layers, all slightly overhanging each other	Correct box is clicked on for each case, true is returned each time
Two hitboxes in a cross formation on the same layer	AttributeError is raised, true is returned
Two hitboxes in a cross formation on the same layer, cross in opposite direction	AttributeError is raised, true is returned
Two hitboxes, close but not touching, six different orientations - this is to check overlap math	No errors raised, true is returned

Log Results

Because one of our coding practices is to only commit and push new changes if all test cases are passed, there was no point in time where committed code failed tests. As such, to show what failed test results look like, a trivial change was made to cause a test to fail. Here is what the results of the test look like when a test is failed:



```
Administrator: C:\windows\system32\cmd.exe

C:\Users\BENSON\Dropbox\361\conquertheseas\src>mousehitbox.py -v
test_cross (<__main__.TestMouseHitboxes>) ... ok
test_different_corner (<__main__.TestMouseHitboxes>) ... ok
test_half_inside (<__main__.TestMouseHitboxes>) ... ok
test_half_inside_layered (<__main__.TestMouseHitboxes>) ... ok
test_half_inside_layered_opposite (<__main__.TestMouseHitboxes>) ... ok
test_half_inside_same_layer (<__main__.TestMouseHitboxes>) ... ok
test_many_layers (<__main__.TestMouseHitboxes>) ... ok
test_one_corner (<__main__.TestMouseHitboxes>) ... ok
test_one_in_other (<__main__.TestMouseHitboxes>) ... FAIL
test_one_is_other (<__main__.TestMouseHitboxes>) ... ok
test_one_is_other_layered (<__main__.TestMouseHitboxes>) ... ok
test_one_is_other_layered_opposite (<__main__.TestMouseHitboxes>) ... ok
test_other_cross (<__main__.TestMouseHitboxes>) ... ok
test_overhang (<__main__.TestMouseHitboxes>) ... ok
test_separate1 (<__main__.TestMouseHitboxes>) ... ok
test_separate2 (<__main__.TestMouseHitboxes>) ... ok
test_separate3 (<__main__.TestMouseHitboxes>) ... ok
test_separate4 (<__main__.TestMouseHitboxes>) ... ok
test_separate5 (<__main__.TestMouseHitboxes>) ... ok
test_separate6 (<__main__.TestMouseHitboxes>) ... ok

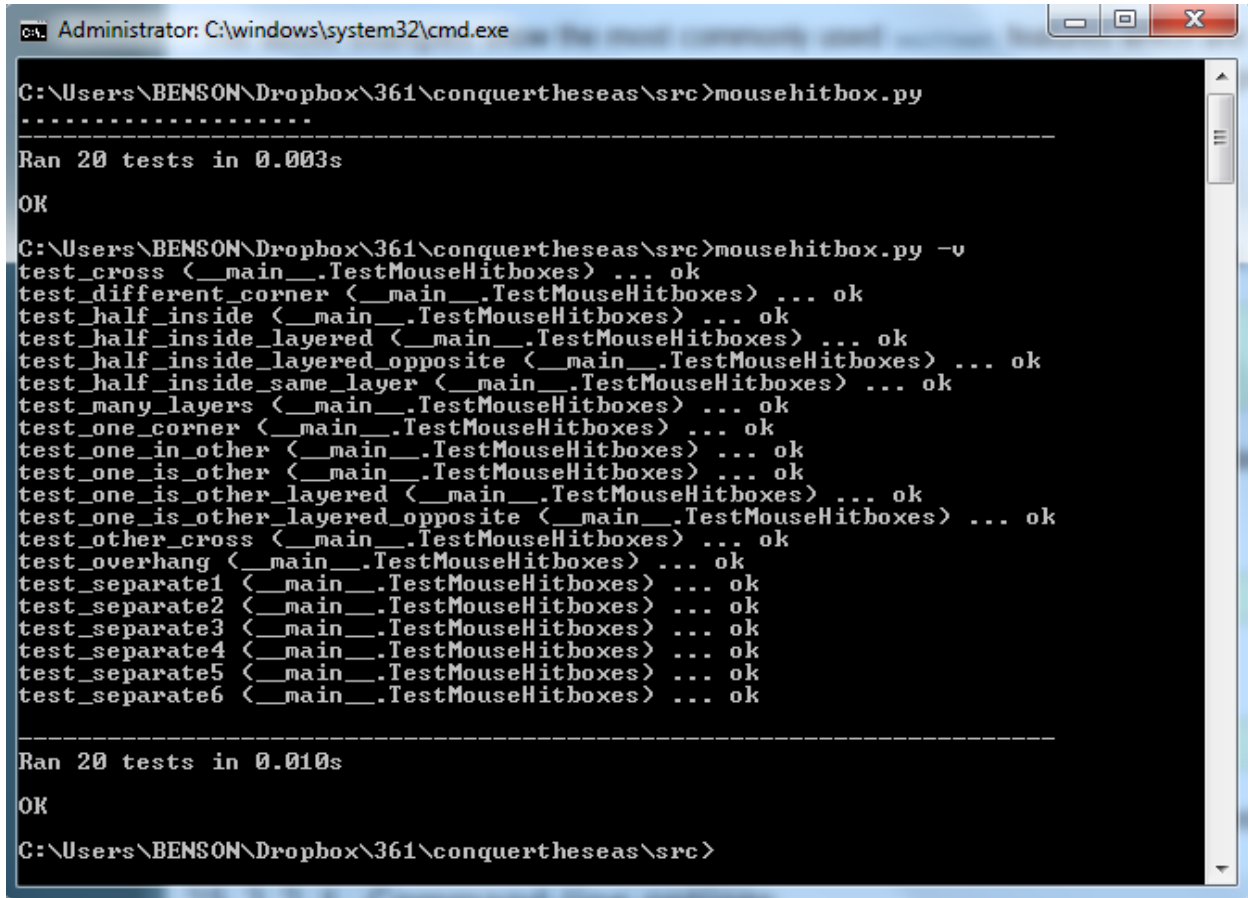
=====
FAIL: test_one_in_other (<__main__.TestMouseHitboxes>)
=====
Traceback (most recent call last):
  File "C:\Users\BENSON\Dropbox\361\conquertheseas\src\mousehitbox.py", line 158
, in test_one_in_other
    self.mh.append(<5, 5, 1, 1>, self.function(<1>>)
AssertionError: AttributeError not raised

=====
Ran 20 tests in 0.015s

FAILED (failures=1)
C:\Users\BENSON\Dropbox\361\conquertheseas\src>
```

Each test is listed along with its result. As seen in this image, `test_one_in_other` has failed. The line which fails is shown, and the `AssertionError` is given - in this case, `AttributeError` was not raised when it was expected to. This means that two boxes were allowed to be placed in an overlapping formation that was not allowed. We would have to go back to where we raise `AttributeErrors` and make sure that we're raising them properly for box overlaps that should not be allowed.

After having fixed this error, we run mousehitbox.py again, and we see that all tests have been passed.



```
C:\Users\BENSON\Dropbox\361\conquertheseas\src>mousehitbox.py
.....
Ran 20 tests in 0.003s
OK

C:\Users\BENSON\Dropbox\361\conquertheseas\src>mousehitbox.py -v
test_cross (<__main__.TestMouseHitboxes>) ... ok
test_different_corner (<__main__.TestMouseHitboxes>) ... ok
test_half_inside (<__main__.TestMouseHitboxes>) ... ok
test_half_inside_layered (<__main__.TestMouseHitboxes>) ... ok
test_half_inside_layered_opposite (<__main__.TestMouseHitboxes>) ... ok
test_half_inside_same_layer (<__main__.TestMouseHitboxes>) ... ok
test_many_layers (<__main__.TestMouseHitboxes>) ... ok
test_one_corner (<__main__.TestMouseHitboxes>) ... ok
test_one_in_other (<__main__.TestMouseHitboxes>) ... ok
test_one_is_other (<__main__.TestMouseHitboxes>) ... ok
test_one_is_other_layered (<__main__.TestMouseHitboxes>) ... ok
test_one_is_other_layered_opposite (<__main__.TestMouseHitboxes>) ... ok
test_other_cross (<__main__.TestMouseHitboxes>) ... ok
test_overhang (<__main__.TestMouseHitboxes>) ... ok
test_separate1 (<__main__.TestMouseHitboxes>) ... ok
test_separate2 (<__main__.TestMouseHitboxes>) ... ok
test_separate3 (<__main__.TestMouseHitboxes>) ... ok
test_separate4 (<__main__.TestMouseHitboxes>) ... ok
test_separate5 (<__main__.TestMouseHitboxes>) ... ok
test_separate6 (<__main__.TestMouseHitboxes>) ... ok

Ran 20 tests in 0.010s
OK

C:\Users\BENSON\Dropbox\361\conquertheseas\src>
```

Now that all test cases have been passed, we can be sure that any and all changes to mousehitbox are acceptable and no functionality has been compromised.

Log Corrections and Regression Changes

Again, because of our practice of not committing code that failed tests, we do not have a place we can revert back to where we had errors in this code that caused it to fail tests. However, we can list the changes and revisions that took place to mousehitbox.py.

Date	Change
11/12/11	modulized, credits added, clickbox deletion added
11/12/11	fixed unittests
11/12/11	more hitbox test fixes
09/01/12	moved to a different location
19/01/12	hitbox reworking begins - tests for one inside the other added
20/01/12	one inside another is no longer allowed
01/02/12	multi-level clickboxes are allowed, test cases for them added, and all test cases still pass
02/02/12	tests for different boxes inside one another added
02/02/12	slicing makes new indices - test cases still all pass
02/02/12	minor comment change
16/02/12	added a way to clear a set of mousehitboxes
18/03/12	mouse functions now only take mpos, not scr - test cases all still pass

Each of these changes was only committed once all mousehitbox tests passed.

Modification Log

This table shows the changes that have been made since earlier documents.

Title	Description	Date	Responsible	Results/Changes
Client/Server.py	Client.py and server.py merged into networking.py	02/03/12	Matt	More compact networking code, all in one place
Due dates	Most important features were given earliest due dates, despite being the most difficult to complete	--	Benson, Matt, Brian	Many deadlines were missed as code that was intended to be finished was not complete.
Networking completed before single-player AI	Networking code was completed before single-player AI.	--	Benson, Matt, Brian	Single-player AI was given a lower priority
Test Driven Development	Many areas of the code (such as networking and GUI) are very difficult to write tests for before writing any code.	--	Benson, Matt, Brian	Testing for some classes will take place manually after code has been written.
UpgradeFactory removed	UpgradeFactory has been abandoned in favor of using a hard-coded dictionary that can generate upgrade trees.	16/03/12	Brian	A hard-coded dictionary of upgrades needs to be created

Programmer's Journal

Benson's Journal

March 1st

Wrote up the deliverables document and the programmer responsibility table to turn in. Committed it, but I don't think we'll need it later.

March 2nd

Worked on a new server in python. learned a lot about servers, i've never done server stuff in python before. this way seems much easier than the previous way my teammates wrote.

March 9th

Fixed some hitbox issues on the main screen. Also, made it so you can change your multiplayer name. Because I spent so much time writing documents the first half of the year, I'm behind on the code. I don't think I'd like working somewhere that needed so many documents, I much prefer a more code-first fix-later style.

March 10th

Made name changing for multiplayer work. Also, you can now host and join games without race conditions ever being an issue. Networking stuff is pretty easy to have problems with.

March 11th

Names can now have spaces - before, they were just regarded as invalid. Names also will scroll appropriately if you enter a name that's too long. Also changed some mousehitbox issues on the new game submenu.

March 13th

Made upgrade screen have a back button that works. Drew lines to partition the buying and info display section of the upgrade screen.

Later on, made a shop button that works for upgrade screen. The ship 1, 2, and 3 buttons also change between the different upgrade trees (one set for each ship). Right now that just means switching the background color, because we don't actually have different trees.

March 16th

Made upgradIdeas.txt - a list of ideas for how to do our upgrades and generate the trees. Hopefully this will be used to discuss this, maybe not, we'll see.

March 17th

Made the purchase button on the shop gray out if you can't afford the thing you've selected. I also made upgrades purchasable. They don't actually do anything yet, but the trees make sure you have enough experience and the prerequisite upgrades purchased before allowing you to buy them. This is actually pretty fun to code.

March 23rd

Sent some emails about the implementation document. Made terrain generation. It's really bad and random right now, but the ideas are all down. Also made coins that float through the water that the defensive units can collect, right now they just kill the defensive units.

March 30th

Finishing up the implementation document as it's due tomorrow. I still really don't like writing documents, I feel like they just take away from the coding. Maybe just planning out some general structure early on?

Matt Journal

February 21

Worked on server: readiness and nick commands implemented. Struggling with clean disconnecting.

February 25

Added kicking and made disconnecting cleaner.

March 2

Worked on linking lobby screen to server with Brian: created dropdown menus for player slots.

March 6

Met with an external friend and figured out how to fix bad disconnects. Cleaned up networking code I wrote with benson on the 2nd.

March 8

Worked with Brian on lobbyscreen. Fixed client creation and disconnects in conjunction with closing the game (learning to hate networking).

March 9

Clean up lobby a bit more with Brian; should finally be perfect. Worked on shop and linked it to offense panel. Changed some magic numbers to constants.

March 10

Expanded networking to allow large messages.

March 13

Changed defensive units so that they can only preform movespeed actions and added undo functionality.

March 20

Fixed bug in undo. Fixed connecting to an improper source.

March 21

Fixed server issues created in my absence.

March 23

Started the gaming portion of networking with Brian (huge awful pain). Changed game creation to take list of players and whether or not it's an online game.

March 30

Did more multiplayer with Brian. Fixed unit coloring bug, board not resetting after submit, and defensive units knowing who they are. Helped Benson with implementation doc.

Brian's Journal

February 22, 2012:

Started work on the join screen, including handling keypresses (such as holding a key down), and ensuring correct IPs are entered. Added some code to lobbyscreen to write messages to a collective message surface. Also created a single class to handle the waves in the background of many screens, to avoid code repetition.

February 24, 2012:

Today was spent drawing some of the units in our game. This includes a mermaid with a knife, a squid-creature, a crab, a floating mine, and Cthulhu. I also added an FPS counter to the game to test our FPS, which is unarguably bad on some screens (dropping to 14 fps on the gamescreen itself). This is due in part to our naïve coding when we started, and we have since learned some tricks that I'm sure we can use to improve our performance drastically.

March 02, 2012:

I added on to the networking code my teammates had started. First I made the client/server code we had been testing much easier to test, and we can now run it from the command line. I also added some code to lobbyscreen in preparation of the networking code, and to the networking code itself added the ability to set non-human slots to opened, closed, or AI. Also added code for validating nicknames.

March 06, 2012

The Join Screen will now connect to the given IP and place you in the lobby, if there is one open. However, if the IP you have given does not have a lobby with an open slot, it will still crash hard. Added a dropdown to the lobbyscreen so the host can set non-humans to AI, closed, or open. Also added tab-completion to chat, to autocomplete names that you have started typing.

March 09, 2012

Today I cleaned up the code and added comments to a bunch of classes, and added the ability to send more than just a tadpole. Also added a great deal of missing functionality to shop screen.

March 13, 2012

Small changes, more images added to shop. Finally changed our terrible arrow-drawing for the thoroughly tested version. Helped Benson with upgrade screen code.

March 14, 2012

Added more images, fixed a small glitch, will work more tomorrow.

March 15, 2012

Ships now start underwater, singleplayer players no longer share their inventories with the AI.

March 16, 2012

Wrote a way to add new upgrades easily to upgrade screen as we think of them: they will draw to the screen in the correct positions and highlight as needed.

March 18, 2012

Removed moustache from mermaid. Fixed some legacy code related to mousehitboxes.

March 20, 2012

Save/load screen started, fixed more bugs, closing slots

March 23, 2012

Client will send its moves to the server, saving and loading is going to be much harder than I had anticipated, will finish later.

March 30, 2012

Completed multiplayer: two players can make moves and each see the moves changed on their screens.