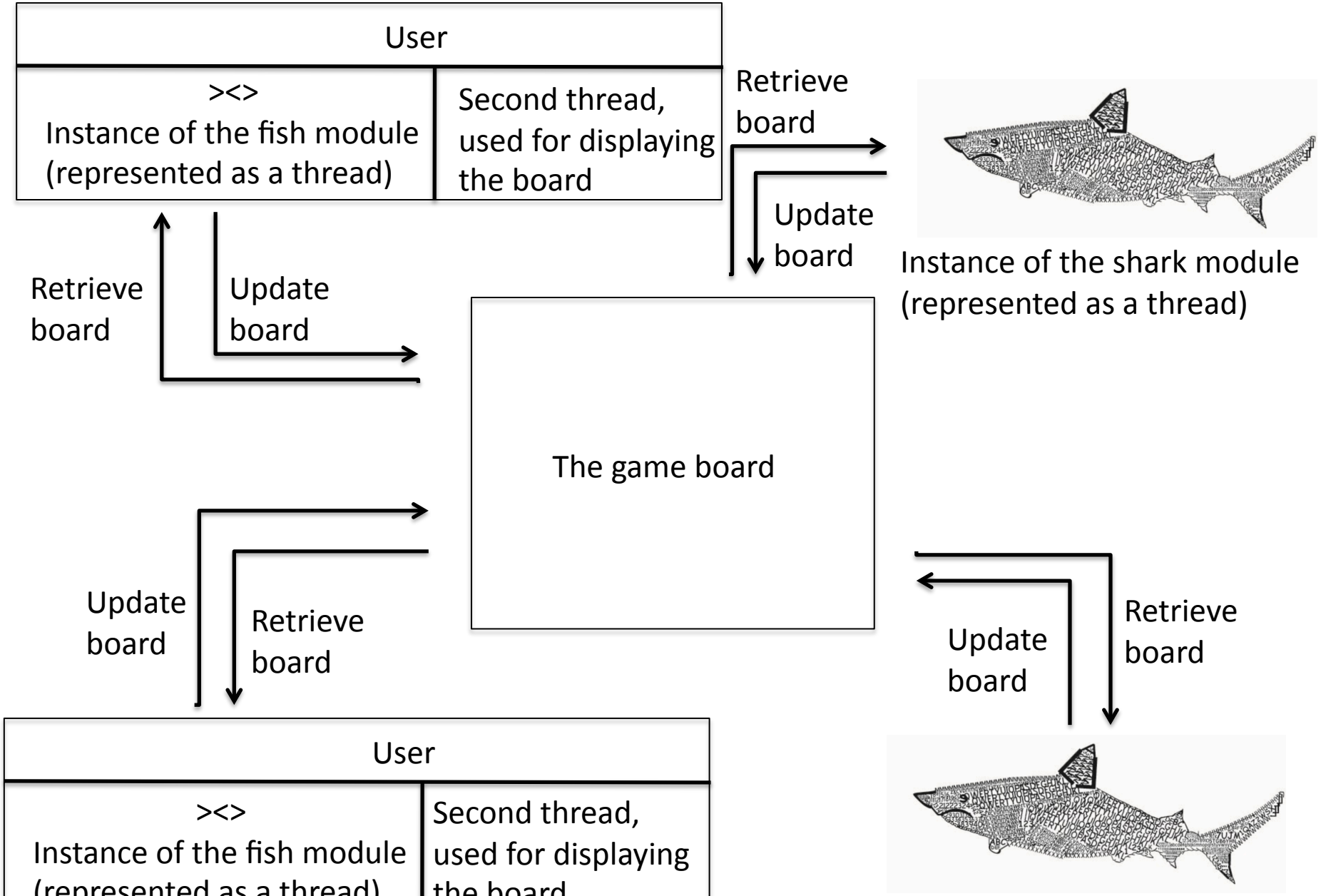
Development Plan
Team SharksAndMinnows
David Stern
Matthew Carrington-Fair
Matthew Epstein

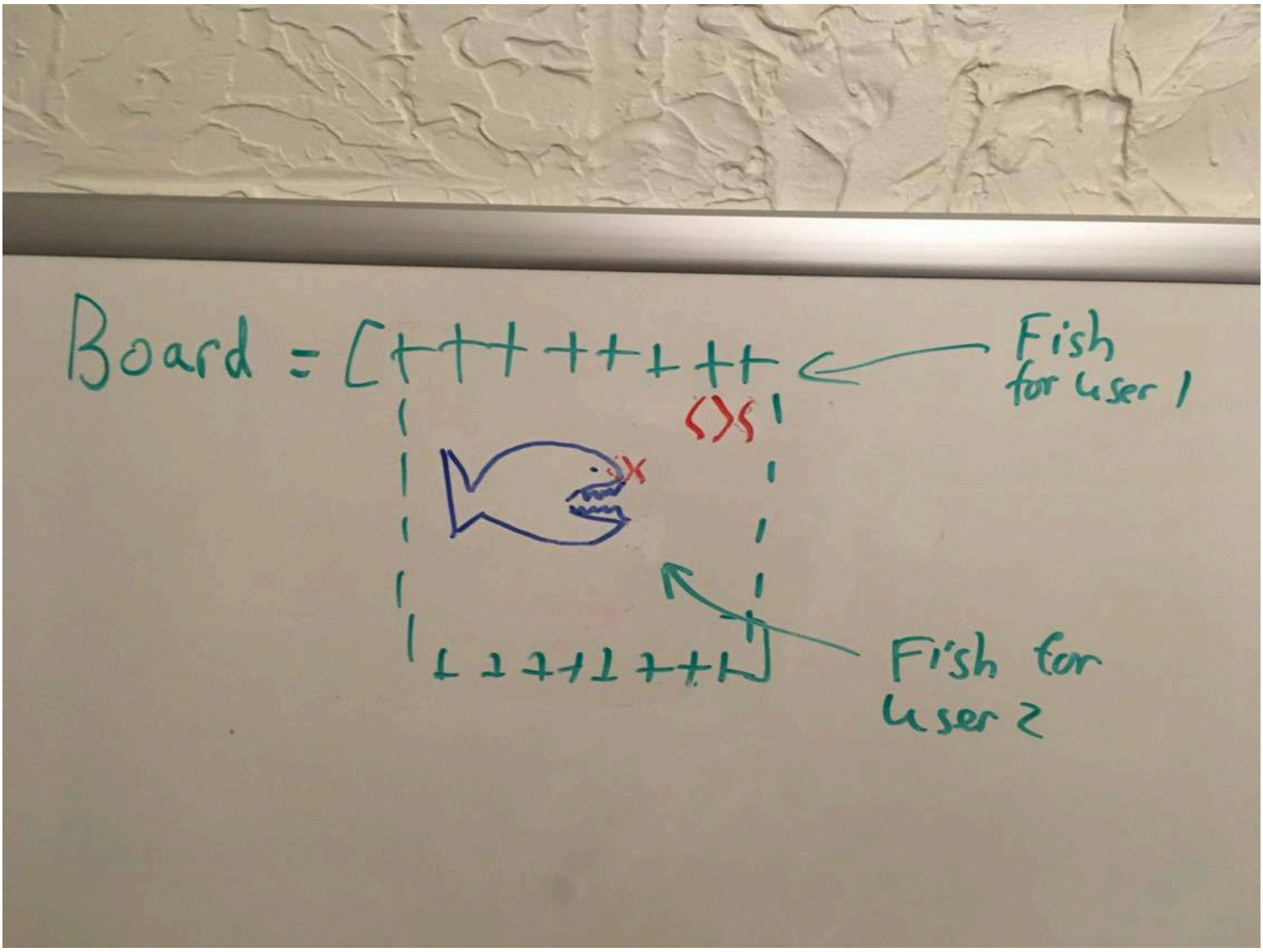
So far, we have used Pyro to fully implement a board object, which can be minimally interacted with by other users. We also have models for the fish and shark images, and have a script that can read them in and create objects. We believe that the next step is to allow fish and sharks to interact with the board. Currently, we have not been able to print fish and sharks onto the board, and we believe this to be the first step. This is then followed up with movement. We believe that the sharks should be able to move on their own, as it is more straight forward. Then, we will have to tackle the problem of user movement.

So far, we have worked together on different aspects of the project; that is, we do not specifically break up tasks for each person, but rather we work together and split up as needed. For example, we worked together to create the design and make the decisions on big design choices, but we broke up so that one person set up basic file I/O and implement board/fish/shark objects, and others worked towards writing a Pyro tutorial and implementing it into the existing architecture. We are primarily using Github as a working interface (we are using a private repository, to avoid academic integrity issues).

Gather round fam, come, have a seat
We have stories to tell, and this one's a treat.
Consider a minnow, it wants to survive,
But in shark-fested waters it can be hard to strive.
But these little minnows, eaten they weren't,
They're all here together, in a game that's concurrent.
So the problem here that we wish to address:
So many fishies, not even one less.
And all of these guys, all of them stored
Each trying to access the same very board.
An issue like this, two ways we can go:
The first is called Erlport; the second, Pyro.
But with Erlport you see,
Things get quite tricky,
This intermediate step
Which we cannot forget
We've got all these fishies, all swimming around
Need to move quick; let's not be sharkbound.
And these little fishies, they're in such a fervor
Don't wanna take time to talk with the server.
But fish to gen server to board is the sport,
If we decided to go with Erlport.
But now listen closely: for Pyro you'll hear,
Straight from fishies to board? The choice here is clear!
There can be no question, there can be no debate,
We've resolved this argument, it need not get late.
For Pyro is simple, it's nice, it will work,
Our fishies will swim, the sharks they can lurk,
Our implementation, Mark will admire.
The reason: well, it's cuz Pyro is fire.

For our project, we are building a concurrent version of the classic Sharks and Minnows game. Each user controls a different minnow on a single board. The goal is to not get eaten and thus stay alive the longest. The main issue we needed to tackle was how to allow multiple users to simultaneously access the same board. The two ways we considered addressing this were with Erlport and Pyro. Ultimately, we chose to use Pyro since we thought it would be simpler. Whereas Erlport would add in an extra layer to the implementation, since the users would have to communicate with the gen server, which in turn would have to communicate with the board, Pyro allows the various users to communicate with the board directly. In addition, Pyro will be fire.


User
><>
Instance of the fish module
(represented as a thread)
Second thread,
used for displaying
the board
Retrieve
board
Update
board
Instance of the shark module
(represented as a thread)
Retrieve
board
Update
board
The game board
Update
board
Retrieve
board
Update
board
Retrieve
board
User
><>
Instance of the fish module
(represented as a thread)
Second thread,
used for displaying
the board
Instance of the shark module
(represented as a thread)


Board = [+++++++++
Fish for user 1
<><
X
1 2 3 4 5 6 7 8 9
Fish for user 2

Description of Diagram Two

In this diagram, we illustrate the game board when a collision occurs between a player fish and a shark.

The game may have reached this state in either of two cases.

Case one: The player moved the fish into the position already occupied by a shark

Case two: The shark thread updated its position moving it into the area occupied by the fish.

In either case, once the board receives the input from either the fish in the first case or the shark in the second, it will first determine whether there is a collision before returning the updated board. It would see that portions of both a fish and a shark are overlapping, therefore it will send a message to the player client indicating that there was a collision and update the all the player's display with the fish removed and the player's loss displayed afterwards.