1) **Team name:** SharksAndMinnows

2) **Group Members:**
   David Stern
   Matthew Carrington-Fair
   Matthew Epstein

3) **Project:** Our goal is to create a concurrent, multiplayer sharks and minnows game. Players will take control of small fish in a tank and will attempt to avoid sharks of increasing count and speeds with each wave, and the last player alive will be declared the victor.

4) **Minimum:**
   At minimum, we expect the following. At least two players from different machines should be able to join a lobby. One of the players should be able to host this lobby. We expect that each player should be represented by a thread, and each shark to be represented by one as well. There would be a queue for threads to be launched and a Semaphore that allows new sharks to be spawned once old sharks have left the screen. There will also be a maximum number of sharks, so that we do not create infinite threads. We also expect that players will lose upon coming in contact with a shark, and the last player alive will win. After research into the conventions of modern games, we have decided that a tie should be determined by chance; if the last two players are hit by a shark at the same time, the winner will depend on which fish the shark thread sensed a collision with first; collisions between different minnows, however, will be prohibited. Lastly, we expect the visual representation of the game to be ASCII.

   Maximum:
   At maximum, we would like to implement the game with sprites, as opposed to ASCII art. We would also like to introduce gameplay above the tank, with the ability to jump out of the water to avoid sharks. This would introduce new enemies as well: seagulls. Another goal would be randomized terrain, which fish must go around to avoid sharks, making it inaccessible space in the tank. We would also like to implement a leaderboard. This would be represented as a server, which the host would send requests to after a game to update it. This would be accessible to view from the login screen.

5) **First Step:** We believe our first step should be drawing a tank and a fish on the screen and allowing for a user to control the movements of the fish. This would be quickly followed up by allowing two fish (different players) to currently be added to the tank, and move on their own.

6) **Biggest Initial Challenge:** We foresee our biggest initial challenge to be networking issues. We do not have any experience setting up lobbies with people hosting and joining. There seems to be a lot of support online, however, so we do not think it will be too difficult to overcome.

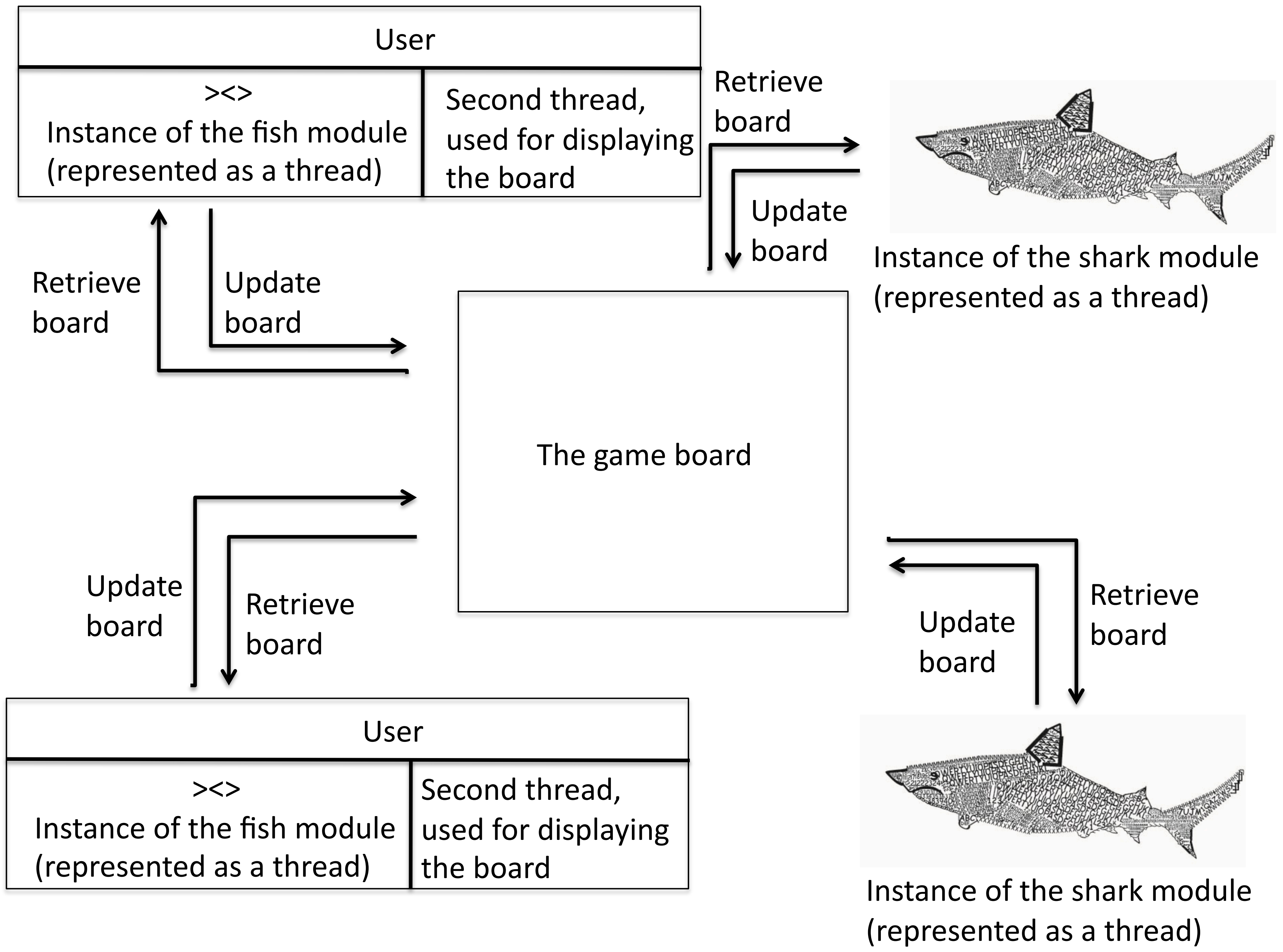
Development Plan
Team SharksAndMinnows
David Stern
Matthew Carrington-Fair
Matthew Epstein

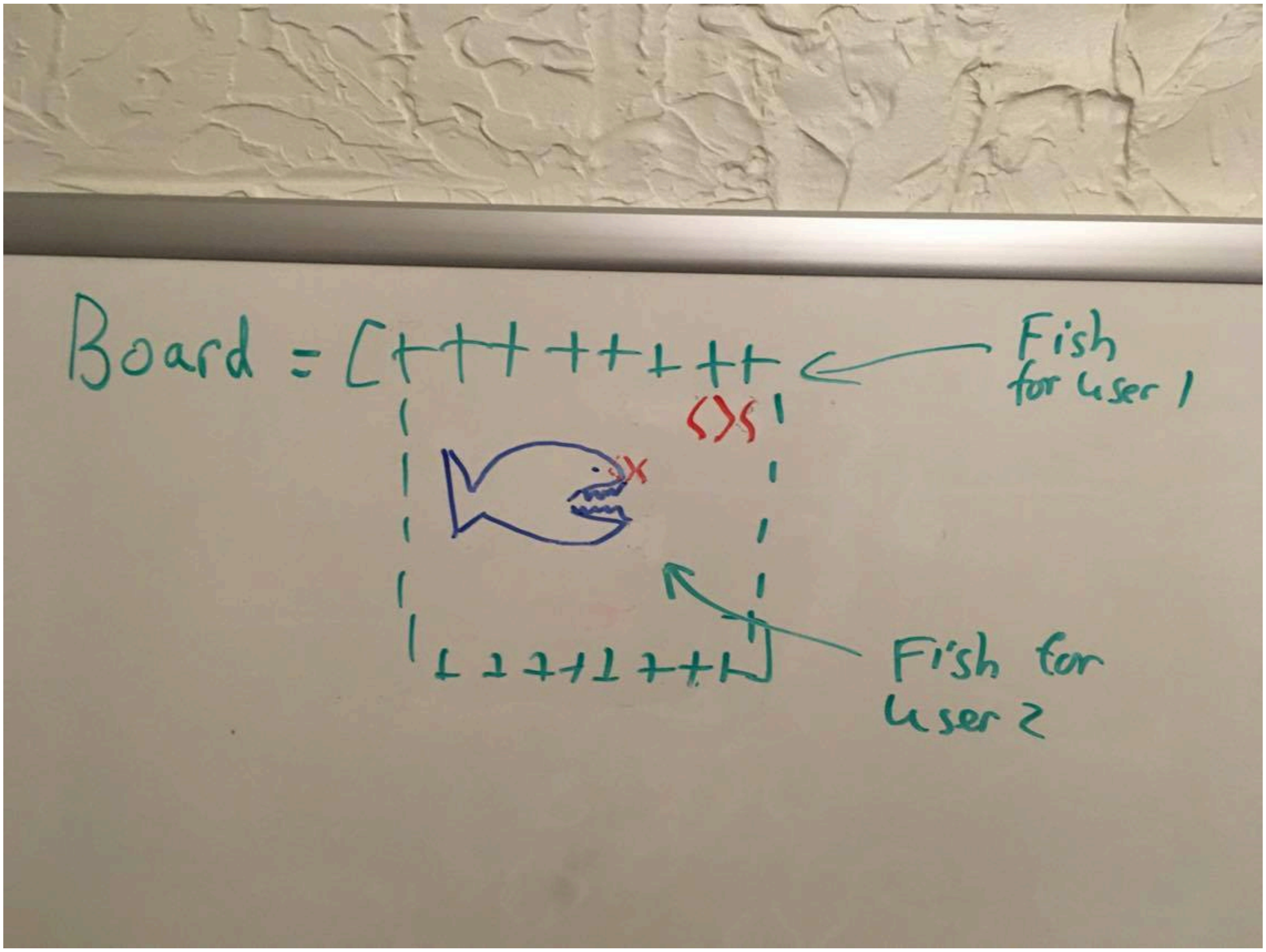
So far, we have used Pyro to fully implement a board object, which can be minimally interacted with by other users. We also have models for the fish and shark images, and have a script that can read them in and create objects. We believe that the next step is to allow fish and sharks to interact with 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
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.

Refined Design – 11/19/2017

Since our previous design submission, we have been able to implement a solid prototype of our board using Pyro, allowing distributed access to the same board object that can be updated and read exactly in the way we want. Currently, we host the nameserver locally on one of our systems, but we hope to eventually move it to a remote server that we'll have running. We hope to be able to re-use that server for our "extra" additions (high-score board, allowing gamer-tags for players, etc.). We managed to get past one of our initial hurdles of automating the nameserver start-up & initialization of our board object upon a host starting a game.

Besides setting up our board object with Pyro, we have also been able to simulate movements and updates to the board by testing sharks. One major implementation obstacle we faced was the process of updating and rendering the board to emulate real-time movement of the sharks on the board. We discovered that the issue originated from our algorithm used to update the board given a new current position of an object. Every time a shark moves, it first read the board before writing its position, and then re-read the board after it wrote to it, which slowed down overall performance considerably. To address this issue, we changed it so that to update the board, the object sends its current location and the board itself will write the shark onto itself.

For the rest of our minimal deliverable, we still need to create a clear distinction between the two user threads, one for controlling the fish and writing to the board, and one thread for simply reading the board. We have begun working to make sharks entirely independent of the user, treating them almost as an extension of the board (except with no influence from the user). Once we clean up the user control of the fish, it is simply a matter of connecting all the pieces into a polished product (i.e. hosting nameserver not locally, creating a game lobby to act as a barrier for players waiting to play, handling how the game terminates).