Just Snek Things

Anne Oursler, Lexi Galantino, Matt Turner

Our project can best be described as distributed Snake. We use erlang gen_servers to allow users to create and join multiplayer games of Snake. Actual calculation for a given game is done in a python program, which is in regular contact with the game server via erlport. The server then communicates with every client gen_server attached to the given game to send updated boards and receive moves from each player. The board is then represented on users' screens via erlport translation to a python gui. The gui additionally handles keyboard input.

The concurrency in our project comes from the various players' ability to move and function at the same time across a shared domain (the board), which is handled via erlang messaging.

As we've dived into the project, our largest issue has been handling keypresses. Both erlang and python are line buffered at the terminal. This means that every key press has to be associated with a return. There's a workaround for python, using tty and termios modules, but the workaround is limited to the use of an actual terminal, and so it's incompatible with erlport and an erlang-managed python VM. Similarly, erlang has something called the wx library, but as Mark noted in class, it's a brittle package, and its implementation protocols are slightly convoluted. We've found some potential success by shifting our GUI development to tkinter, which allows us to live-track keyboard presses and we hope to have it linked up this week.

Originally, we assumed our biggest issue with this project was finding a way to handle so many updates on the shared board, consistently, and quickly. Our board as a data structure is a large 2d array, passed across the erlang connection as a tuple. This is because list passing between Python and Erlang is a slow and painful process. However, tuple translation is native to the package and python casting to and from tuple/list is fast for our array sizes. Our idea is to limit interactions per player to 1 per .~5 seconds. Even with that limit, latency and computation time are huge considerations. We could program everything in erlang, but as all objects in erlang are immutable, the sheer space constraints of so many list updates on a large 2d array seems unfeasible. So, instead, we've relied on erlang as a simple way to deal with multiple user requests, and pushed off all actual computation to

a python VM running a class MySnek which contains all state necessary to represent the game.

So far, we've spent a considerable amount of time porting over our erlang gen_server code, as well as reviewing possible solutions for the client-side GUI. Once the GUI's done and linked, we'll have largely met our minimum deliverable. However, the next step will be to try to carve out some of the stretch goals, to have a more complex product.