Initial Design for Banana Tag:

Group Members: Theodore Schneider, Matt Zhou, Noah Tervalon

a) **Description**: A game of tag where chaos reigns.

b) **Rules**: Normal tag. However, when you are tagged you are temporarily suspended from the game until the person that tagged you gets tagged. Thus, the only way to win is to tag everyone without being tagged once. Visit this link for more rulebook information, note that the video is not related to the rules on the page: https://ultimatecampresource.com/camp-games/tag-games/banana-tag/

c) **Our game**: We plan to make an interactive game that allows many people to play at once. Our area of play will be a finite map with multiple player characters all striving to become the champion.

Min Deliverable:

Our minimum goal is to create a visual game that is hosted on the Halligan servers. This game will be multi-player (minimum 4). Additionally, we want to have graphics for the game that allows all players to see what is currently happening in real-time. This includes player movement, players tagging each other, and other events happening on the map. For the game to work, we also plan to have the ability to accept input accurately from all players and avoid concurrency issues such as deadlocks, etc. On the whole, we aim to provide a smooth-gaming experience for the users through our input processing.

Max Deliverable:

Our maximum deliverable is a fully polished game that supports large-scale player bases (15+). We aim to have multiple maps that each have unique layouts (obstacles, dark mode, etc).

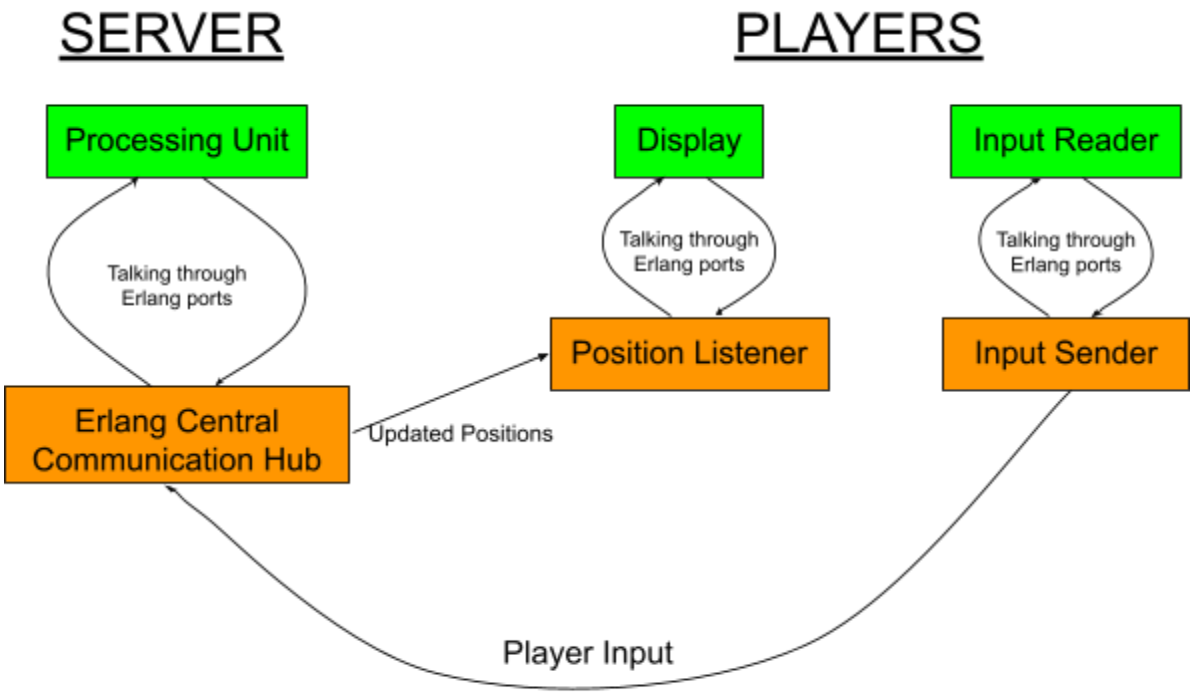
Design Decision Short Story:

Communication is essential to the success of any large-scale operation. In ancient China, they would send messages along the Great Wall by lighting fires to indicate that danger was coming. In our Banana Tag game, we had to decide between using Python solely or utilizing Erlang to communicate each player's game information back to the server (central unit).

Our first way to do this was having each user have some Python process that converts player key press information into Erlang form. Then using the Erlang node connected to each user's Python process, we would send the information to an Erlang node connected to the central Python server that would disseminate the updated game information to each player using this same method but in reverse. The second way would be to only use Python and have each player's process update their location in a shared data structure that the central unit consistently removes from to update the game state.

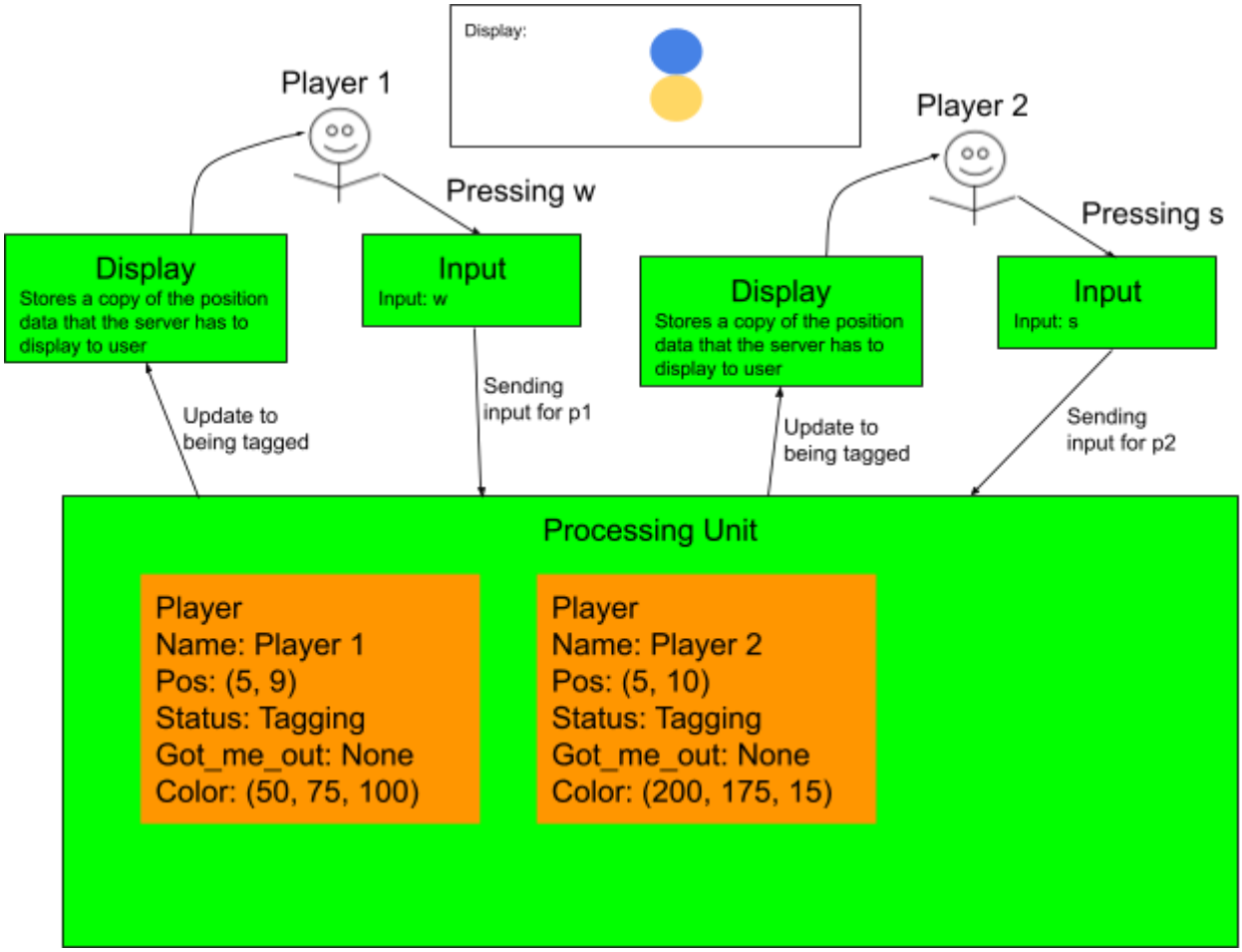
With this hard decision in mind, Matt, Terv, and Teddy took a spring break trip to a Tibetan Buddhist Temple to receive clarity of mind through rigorous meditation. During this trip, we decided to use the first method to support communication in our program. Despite the time needed to convert messages from Python to Erlang and back, we believe that message-passing in Erlang is so clear and easy that it will help our program significantly. In short, Erlang's native support of message-passing is our primary reason for choosing this method of communication.

Class Diagram:



Note that green boxes are python and orange boxes are erlang

Object Diagram:



The display at the top shows the game state before the players input their movements.
Player 1 started at position (5, 9).
Player 2 started at position (5, 10).
Player 1 sees a displayed game board and presses "W", this information is relayed to the server.
Player 2 sees a displayed game board and presses "S", this information is relayed to the server.
The server receives the messages, calculates the positions of the players, and recognizes that Player 1 and Player 2 can tag each other.
The server prompts the displays of the two players to show them that they can tag each other.

Roles/ Division of Work:

-Matt - Message Passing - node architecture

-Teddy - Keyboard input (each user's node collects keyboard input and sends its updated coordinates to the central node)

-Terve - graphical display + game state and player organization

Finally, your proposal should include a development plan. What have you done so far, and how do you plan to proceed? Have you had to change your weekly meeting schedule? How will the work be divided among the members of the team? If you are planning to split up the work, how will you manage the interfaces between the parts of the project? If you are planning to specialize, I suggest that you make some effort to ensure that all members of the team understand all parts of the design at an appropriate level of abstraction.

Development Plan:

All team members plan to meet every Sunday at 1 pm for varying hours to work on the project according to their roles and help each other out.

Status Update: We are on schedule. As of right now, we have a very basic simulation of the game that does not take user inputs, a way to collect user key presses, and code that spins up an Erlang node and connects it with a Python process through a port

During our weekly meetings, we plan to synchronize any work done during the week. We make a strong effort to ensure that all group members understand each part of the project and how it interacts with all other parts.

1) Week 0 (March 11):
    a) Familiarize ourselves with useful packages and come up with a more concrete design
    b) This includes writing small programs to test the usage of these packages
2) Week 1 (March 25):
    a) Finish object and class diagram for the initial design proposal
    b) Create a basic working model for message passing and work out the overall architecture. This includes creating a central node that's able to communicate with client nodes. Also, nail down the way to take inputs and display basic info on the client side.
3) Week 2 (April 01)
    a) Connect the different components. Mesh the client input and display code with the overall architecture. Finalize the communication protocol between the client and server.
    b) Work on the refined design
4) Week 3 (April 08)
    a) Implement the communication protocol and have basic functionality

    b) Test the connected features and communication protocol and work on bugs.
5) Week 4 (April 15)
    a) Implement Additional features:
        i) Multiple games being able to be played at once.
        ii) Fog of war.
        iii) Multiple maps.
        iv) Objects in maps.
6) Week 5 (April 22)
    a) Test advanced features.
    b) Test features even more. And even more.
    c) Write final report
7) Due Date: Apr 29, 2024

Additional Information:

** This is just our ideas in Word format. This is what we used to explain to ourselves how we want to design this project. **

General parts of the project (effectively code files):

- Python central unit which does the following
    - Lets users "log in" with their name and spins up a new thread/node for them
        - Erlang node (through Pyrlang) which will handle message passing
        - Python module which will show the graphic of what is happening
    - Listens to users moving around
        - Continuously sends updates about positioning to every node
        - Alerts nodes as to when they are getting tagged
        - Handles interaction between the players
    - Controls overall flow, and movement between phases of the game (waiting screen, playing screen)
        - Alerts players to changes in game state (game over, game start)
- Python unit which controls graphics display on the user side
    - Given the locations of players, display them as moving circles
- Erlang (embedded within Python) to handle the message-passing
    - Have a node at the central node, and one on each user
        - The central node listens and reports back up to the Python central node about what input the players are giving
        - Player nodes listen for keyboard input and send that to the central node if valid (i.e. pressing random buttons doesn't clog up the message-sending space)