Design

# Needs (optional extras in *italics*)

* Software for running laboratory experiments *and online experiments with e.g. mturk*
* Easy administrative control from the command line *and via web interface*
  + Start, stop and pause the experiment
  + Observe who is connected
  + Send messages to individual clients
* Ability to push data to clients *instantly via RPC?*
  + *Fine-grained timing data available*
* Control over which clients can connect. Authentication by different methods.
* Control flow: server, not client, controls what is displayed at any time (short of leaving experiment)
  + *Ability to push clients on manually from command line*
  + *Different clients go through different stage orders?*
* Testing. Easy testing with multiple “users”. Ability to replay sessions. Simple creation of fake users.
  + *Simple creation of test periods with fake opponents for real subjects.*
* Debugging using standard tools.
* Easy standard HTML forms for quick creation.
  + *With data validation automatically built in? (So, autobuild “check” stage)*

# Implementation

## Experiment class.

Contains a description of the order of the experiment and the running state.

* What stages go when. Stages list. addStage method.
* Controls entry
  + How many subjects, who can enter (“fill up” to N, versus allow at any time)
  + Authentication methods
* Records session information, data received from clients (in transport-neutral way – allows for test clients to interact).
* Experiment control (start/stop/pause/*rewind/replay at different speeds*)
  + If the experiment is deterministic given an initial state and a set of commands, then we could rewind easily.
* Experiment info (probably straight to command line, perhaps with single method that could be overridden). Sometimes pushed, e.g. when all subjects are close to end of stage.
* Experiment states.
  + STOPPED. Server not running experiment. When admin calls start(), moves to
  + WAITING. Accepting clients but not yet ready to start. Start can be automatic when enough clients login, or require admin push. start\_method field.

## Session class?

* Would it make sense to say an experiment runs multiple sessions?

## Stage class.

* Describes who may enter and “waiting rule” for entry. (Individual, SubjectGroup or All – are these all really SubjectGroup objects? But should be easy to write for user.)
* Contains code to run as users enter stage. before function, passed in by developer
* Describes the HTML shown to the user. html function
  + HTML can refer to the stage’s SubjectGroup (including e.g. their history of play)
* Checks client’s response and determines if can move forward. check function
  + Again, can refer to stage’s SubjectGroup. These may be TestSubjects.
  + When response passes it is stored in the Subject.
  + **If nothing in the stage has a side effect, except by passing data to Subject, then this becomes rewindable.**
    - **Is this a good reason to not have “special cases” where global variables record “first entry?”**
  + **Example: two subjects enter a stage. On first entry, one of them is chosen to receive $50. To replay, do we want to rerun the random decision? Generally, yes, otherwise subject experiences become predictable!**
* Contains code to run as users leave stage. after function, per-user. finished() method which checks if all SubjectGroup have completed.
* **Stage states:**
  + **Subject is READY (has not yet seen HTML. When HTML sent, becomes:)**
  + **RUNNING (has seen HTML, awaiting valid response. When response passes check, Subject becomes:)**
  + **FINISHED (given legit response. Tell Experiment Subject has finished.)**

## Server class.

* runs the experiment; receives HTTP requests and translates them to Experiment commands; sends HTML to clients
* Maintains connection with clients, decides if connections are down?
  + Maintains a table of clients, links client IDs to server-specific connection methods
* Authenticates clients

## SubjectGroup class

* List of subjects
* Associated with a Stage. (So, can be created at start of experiment or stage by stage.)
* Convenience functions to allocate subjects into groups randomly, blocking etc.

## Subject class

* stores the responses made by each subject. Ultimately, one response to each stage? (Failed responses discarded?)
* Receives data from the Stage after it has been checked.
* Links to a table where data is stored per subject/stage.

## RealSubject class

* Linked via ID with clients table in Server.

## TestSubject class

* Automates sending commands to experiment. (Bypassing server? Probably not.)
* **Example: guessing game.**
  + The real experiment has two clients trying to guess the same number. Both enter a number and it is displayed. They win a prize if it is the same.
  + If the same Stage was set up with 2 real clients, each paired with a testclient, then the testclient data would instantly be available and the clients would have a test stage.
  + If the same Stage had two test clients, we’d have instant testing.
  + So: a Stage can be populated with (data from) Clients.
  + But it can also be populated with (data from) Testclients.