

Developing Cognitive Tools for Self-Mastery

Requirements and Specification Document

02/18/16

Owen Chen

Project Abstract

Despite our best intentions, we often fail to act on our goals because of our cognitive limitations that make us short-sighted and prone to procrastination. Powerful artificial intelligence algorithms for planning and decision making could be applied to help people overcome these limitations. In this project we combine tools from artificial intelligence with models of human decision-making to develop a web-based to-do list gamification tool that helps people become more organized, stay motivated, procrastinate less, and achieve their goals more effectively.

Document Revision History

Rev. 1.0 2016-2-24 - initial version

Rev. 1.1 2016-3-4 -

Falk's revisions, completed API methods, revised models according to functional requirements. Implementation plan, timeline iteration 1.

Rev. 1.2 2016-3-20 - Added Todo Task list, Division of labor, Progress Section

Rev. 1.3 2016-4-5 -

Updated with revised Falk specifications for first prototype + diagram and specific technical todo

Functional Requirements

- 1) Graphically display a given to-do list as ordered list of boxes with *task names*, *checkbox or percent completed* text box. Point value. Task names with links
- 2) To-do lists can be displayed in three ways
 - a) no points
 - b) points = stars
 - c) points = money
- 3) Display users score and level.
- 4) Supports the following interactions:
 - a) Login and logout. Account creation
 - b) Random assignment: When a user creates an account they are randomly assigned to one of several experimental conditions. The condition determines the view that will be presented to the user.
 - c) View current version of their own to-do list.
 - d) When user finishes item or X% complete
 - i) Display updated values accordingly
 - ii) User score and level updated

- iii) Views with points and additional feedback
 - iv) Data points recorded in database
 - e) User leaderboard and rank
 - f) The user can generate a code that encrypts how many points they have earned. This feature is necessary so that we can pay participants in the procrastination experiment while protecting their privacy.
 - g) The experimenter can export user data into JSON or CSV file. The exported data should specify when which user completed which activity, and which experimental condition they were in, but it should not contain the user's e-mail address, username, or password.
- 5) Extensible to the following additional functions:
- a) Optimal incentives will be computer automatically and assigned to the activities on the user's to-do list.
 - b) Users can create their own to-do lists:
 - i) When creating a new to-do list, the user is asked to specify the goal that the activities on the list are for, how valuable that goal is, and by which time the goal should be accomplished.
 - ii) When the user adds a new activity to the list, the tool asks the user how much the activity contributes to the goal, how long it will take, and how difficult or unpleasant it is.
 - iii) User can create separate to-do lists for different goals
 - iv) User can navigate between different to-do lists
 - c) Display and creation of hierarchical To-Do Lists
 - d) User can delete to-do lists
 - e) User can provide names of the levels
 - f) User can opt-in to compete with other users. If they opt-in, the leader-board is made visible and their score will appear on other users' leaderboards.
- 6) Procrastination Experiment
- a) The first prototype will be used to test whether optimal incentives reduce procrastination.
- 7) Algorithms
- a) Different methods of analyzing the data or compute user scores

Technical Design Implementation Table of Contents

[1. System Architecture](#)

[1.1. Server-Side Architecture](#)

[1.2. Client-Side Architecture](#)

[1.3. Architecture Diagram](#)

[2. Design Details](#)

[2.1. Front End App Details](#)

[2.2. Back End Server Details](#)

[2.3. Database Details](#)

[2.3.1. Models](#)

[2.4. Invariants](#)

[2.5. Protocols](#)

[2.7. API](#)

[2.7.1. Creating A New User](#)

[2.7.2. User Login](#)

[2.7.3. Deleting A User](#)

[2.7.4. Changing User Password](#)

[2.8. View Details](#)

[2.9. Model Descriptions](#)

[2.10. Sequence Diagrams](#)

[2.11. Third Party APIs](#)

[3. Implementation Plan](#)

[3.1.1 Iteration 1](#)

[3.2. Risk Factors](#)

1. System Architecture

Tempo will be built using the client-server model.

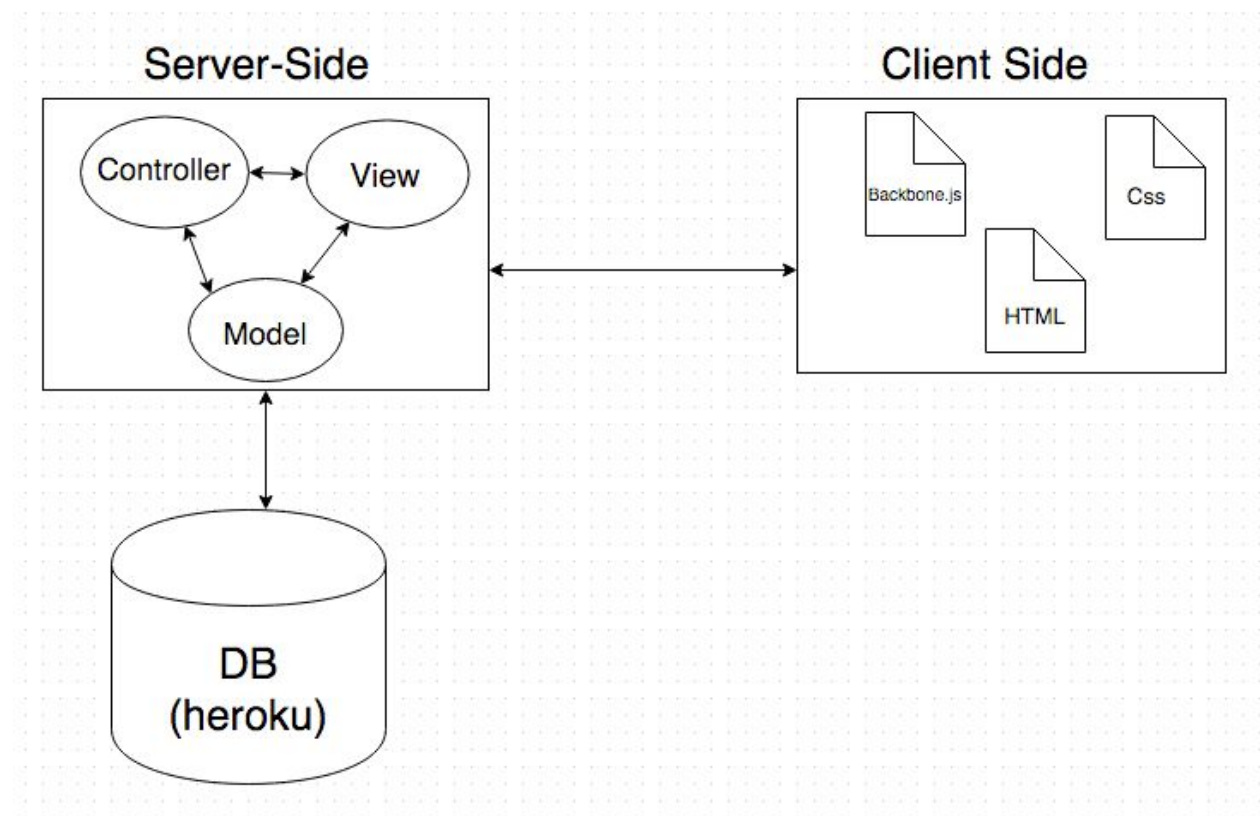
1.1. Server-Side Architecture

- **Framework** - The server-side of the project will implement the model-view-controller architecture through the Express framework.
- **Database** - The server-side database will store important application information, including user account credentials, user history, and activities.
- **Deployment** - The application will be deployed on Heroku.
- **API** - The server will implement a RESTful API to deal with storing and retrieving data. The API will use JSON as a data format.
- **Model-Controller** - The server will implement the model and controller portions of the model-view-controller architecture.

1.2. Client-Side Architecture

- **User Interface** - The client-side of the project will be comprised of JavaScript and HTML/CSS running in a web browser, packaged through the Backbone.js framework.
- **Communication** - The client will communicate with the server and database over HTTP using the server-side API. This communication will be initialized by users interacting with the interface.
- **Views** - The client will implement the view portion of the model-view-controller architecture.
- **Models** - Utilizing Backbone.js, we will create similar models as our Express application with some additions to handle sessions.

1.3. Architecture Diagram



2. Design Details

2.1. Front End App Details

The front end will be a dynamic web application. The structure of our front end application will be defined by HTML5 and styled by CSS3/SASS. Front end logic and animation will be handled by JavaScript, and utilizing Backbone.js framework.

The front-end components will be modularized MVC framework in backbone, with the interaction of the to-do gamification app done with client-side logic.

2.2. Back End Server Details

The back end of this application will be implemented using the Express framework. The server will handle all communication with the front end client via a comprehensive RESTful API.

The backend will be solely an API to retrieve, modify, and update database data for the application. Future algorithms for data analysis or score updating will also be computed on the server-side before send it to the frontend framework.

The back end will handle user authentication, permissions, persisting state through the database, updating, scraping, and analyzing data, etc. ExpressJS will have access to numerous helpful modules that will allow us to easily integrate complex features into our application (such as secure user authentication and permissions).

2.3. Database Details

2.3.1. Models

The models/tables we plan to use in our database include:

- Users
 - id (integer, primary key)
 - username (string)
 - password (string)
 - email (string)
 - name (string)
 - experimental_condition (string)
 - admin (boolean) [Admin's get access to all data for analysis]
 - all_activites (list of activity_ids)
 - completed_activities (list of activity_ids)
 - score (int)
 - level (int)
- Activities

- activity_id (integer, primary key)
- user_id(integer, foreign key)
- title (string)
- points (integer)
- points_gained (integer)
- link (string)
- time_Completed (DateTime)
- content (string) (optional)
- sub_activities (array of keys)
- SubActivities
 - subactivity_id (integer, primary key)
 - activity_id(integer, foreign key)
 - title (string)
 - points (integer)
 - time_Completed (DateTime)
 - link (string)
 - content (string) (optional)

2.4. Invariants

- User
 - The username must be unique, non-empty, and less than 20 characters.
 - The password must be non-empty and less than 30 characters.
 - The email must be unique and non-empty.
 - The name must be non-empty and less than 20 characters.
- Activity
 - The title must be non-empty and less than 128 characters.
 - Must have foreign key to reference User it belongs to
 - Point values of all sub-activities must add up to total_points of current Activity
- Sub-Activity
 - The title must be non-empty and less than 128 characters.
 - Must have foreign key to reference the Activity it belongs to

2.5. Protocols

We will be using HTTP protocols to communicate between the client and server through API calls. GET requests will be used to render views and gather information. POST requests will be used for User account creation. PUT requests will be used to update various information. DELETE requests will be used to delete User accounts.

3. API Documentation

All API calls will be made through the route: "<base_url>/api/<action>".

Authentication must be considered before allowing the client to make an HTTP request that could expose sensitive information or have destructive consequences (e.g. deletion or false data manipulation).

API Table of Contents

2. Users

- 2.1. User Creation
- 2.2. User Deletion
- 2.3. Resetting User Password
- 2.4. Forgot User Password
- 2.5. Forgot User Username
- 2.6. Get All Users (Admin access only)
- 2.7. Get User
- 2.8. Get User Details
- 2.9. Get User Activities
- 2.10. Get User SubActivites
- 2.11. Update User Activites
- 2.12. Update User Details
- 2.13 Add Activity
- 2.14 Update User Score
- 2.15 Return User Score, Completed Activities, Experimental Condition [Anonymous]

3. Activities

- 3.1. Activity Creation
- 3.2. Get All User Activites
- 3.4. Edit Activity details
- 3.5. Complete Activity
- 3.6. Delete Activity
- 3.7. Get Activity Score
- 3.8. Complete Activity
- 3.9. Add SubActivity
- 3.10. Complete SubActivity

4. SubActivities

- 4.1. SubActivity Creation
- 4.2. Get Current Activity's SubActivity details
- 4.4. Edit SubActivity

4.5. Complete SubActivity

4.6. Delete SubActivity

4.7. Complete SubActivity

Template

Description:

Request type:

Route:

Arguments:

Reply (if success, if failure):

Possible Error Messages:

2.7.1. Creating A New User

Request type: POST

Route: /api/users

Arguments:

- username - "<username>"
- password - "<password>"
- password_confirmation - "<password>" [optional, but recommended]
- email - "<email>"
- name - "<name>"

Reply:

- status (an integer response that changes based on the state)
 - 1 (success)
 - -1 (error)
- user (if status=1)
 - a hash containing id, name, username, email, and token
- errors (if status=-1)
 - a list of error messages, e.g.:
 - "Error: password can't be blank."
 - "Error: username can't be blank."
 - "Error: name is too long (maximum is 20 characters)."
 - "Error: password_confirmation doesn't match Password."

2.7.2. User Login

Request type: POST

Route: /api/login

Arguments:

- username OR email - "<username>" OR "<email>"
- password - "<password>"

Reply:

- status (an integer response that changes based on the state)
 - 1 (success)
 - -1 (error)

- user (if status=1)
 - a hash containing id, name, username, email, and token
- errors (if status=-1)
 - a list of error messages, e.g.:
 - "Error: you must provide a username or email address."
 - "Error: you must provide a password."
 - "Error: the provided username and password do not match."

2.7.3. Deleting A User

Request type: DELETE

Route: /api/users/:id

Arguments:

- token - "<token>"

Reply:

- status (an integer response that changes based on the state)
 - 1 (success)
 - -1 (error)
- errors (if status=-1)
 - a list of error messages, e.g.:
 - "Error: user#:id does not exist."

2.7.4. Changing User Password

Request type: PUT

Route: /api/users/:id/change_password

Arguments:

- token - "<token>"
- old_password - "<old_password>"
- new_password - "<new_password>"

Reply:

- status (an integer response that changes based on the state)
 - 1 (success)
 - -1 (error)
- errors (if status=-1)
 - a list of error messages, e.g.:
 - "Error: the old password is not correct.")
 - "Error: user #:id does not exist."

2.7.5. Get All Activities

Request type: GET

Route: /api/activities

Arguments:

- interests - a list of Interest.names [optional]
- time - an integer representing the upper bound time limit [optional]
- token - "<token>" for grabbing a User's Activities as well [optional]

- `user_id` - an integer mapping to a User id, needed with token [optional]

Reply:

- `status` (an integer response that changes based on the state)
 - 1 (success)
 - -1 (error)
- `activities` (if `status=1`)
 - a list of all activities that match the given arguments
- `errors` (if `status=-1`)
 - a list of error messages, e.g.:
 - "Error: no activities found with the given interest."
 - "Error: no activities found."

2.7.7. Get All Activities Of A Specified User

Request type: GET

Route: `/api/users/:id/activities`

Arguments:

- `id` - the integer of the `user_id` of the User to get Activities for

2.7.10 Add A New Activity For A User

Request type: POST

Route: `/api/users/:id/activity`

Arguments:

- `id` - an integer representing the `user_id` of the User
- `Activity` - a dictionary mapping to the parameters of the new Activity:
 - `title` - `<"title">`
 - `completion_time` - integer
 - `content` - `<"content">`

Reply:

- `status` (an integer response that changes based on the state)
 - 1 (success)
 - -1 (error)
- `custom_activity` (if `status=1`)
 - a mapping to the attributes of the newly created Activity, including:
 - `title` - `<"title">`
 - `completion_time` - integer
 - `content` - `<"content">`
 - `user_id` - integer
- `errors` (if `status=-1`)
 - a list of error messages, e.g.:
 - "Error: title can't be blank."

2.7.11 User Logout

Request type: DELETE

Route: /api/logout

Arguments:

- token - "<token>"

Reply:

- status (an integer response that changes based on the state)
 - 1 (success)
 - -1 (error)
- message
 - a string
 - if status=1
 - "Success: please remove the JWT token from the client side."
 - if status =-1
 - "Error: you don't have permission to log out this user."

2.8. View Details

- Login/Signup View
 - Displays a field for the username
 - Displays a field for the password
 - Displays a button to sign up if the User does not already have an account
 - Brings up username, password, and password confirmation fields with a submit button
 - Displays a link for a forgotten username/password
- Home View
 - Reached after successful login or if the User is already logged in
 - Menu button that slides out to show a menu consisting of:
 - Account
 - Settings
 - Leaderboard
- LeaderBoard View
 - Reached after hitting "leaderboard" menu option
 - Displays table of all users, sorted by score in descending order.
- Settings View
 - Contains options to change interests
 - Displays a link to change the User's password

2.9. Model Descriptions

Model Name	User
Variables	<p>id (integer, primary key)</p> <p>username (string)</p> <p>password (string)</p> <p>email (string)</p> <p>name (string)</p> <p>experimental_condition (string)</p> <p>admin (boolean) [Admin's get access to all data for analysis]</p> <p>todo_list (array of activity_ids)</p> <p>completed_activities (list of activity_ids)</p> <p>score (int)</p> <p>level (int)</p>
Methods	<ul style="list-style-type: none"> new User(String username, String password, String email, String name): <ul style="list-style-type: none"> Constructor to setup user details. Create empty todo_list, completed activities, and score. Set a random experimental_condition get_activities(): returns a list of all of the User's Activities add_activity(Activity): adds the Activity to the User's todo_list change_password(new_password): updates the User's password to new_password get_basic_info(): returns a hash of a User's basic info get_advanced_info(): returns a hash of a User's advanced info update_User_Score(Int points): update the current User's score after completing Activities return_User_Score(): Returns user's current score and completed activities return_User_Progress(): Returns details of user's completed activities and correlated experimental condition.

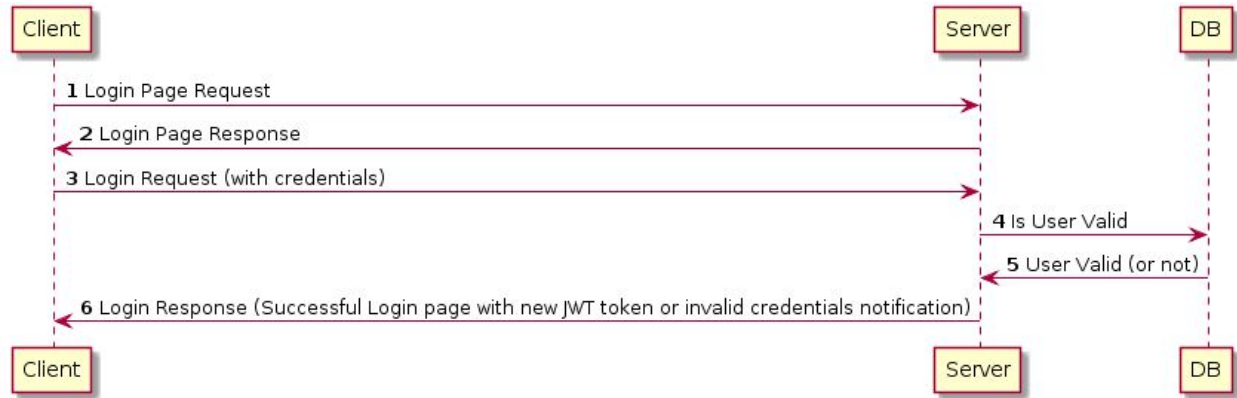
Model Name	Activity
Variables	<p>activity_id (integer, primary key)</p> <p>user_id(integer, foreign key)</p> <p>title (string)</p> <p>total_points (integer)</p> <p>points_gained (integer)</p> <p>time_completed (DateTime)</p> <p>link (string)</p> <p>content (string) (optional)</p> <p>sub_activities (array of SubActivity_ids)</p>
Methods	<ul style="list-style-type: none"> new Activity(Int user_id, String title, Int points, String link): <ul style="list-style-type: none"> Constructor for creating new activities. Instantiate

	<p>time_completed as null, unique activity_id, and sub_activities array</p> <ul style="list-style-type: none"> • add_subActivity(SubActivity): <ul style="list-style-type: none"> ◦ Add's a SubActivity to the current Activity's sub_activities • get_Activity_Score(): <ul style="list-style-type: none"> ◦ Returns this activity's total_points. • complete_Activity(): <ul style="list-style-type: none"> ◦ Create current DateTime object to finish time_completed. ◦ Call update_User_Score and pass in get_Activity_Score() • complete_SubActivity(Int, subActivity_id): <ul style="list-style-type: none"> ◦ updates points_gained of this activity based on points of given subactivity_id
--	--

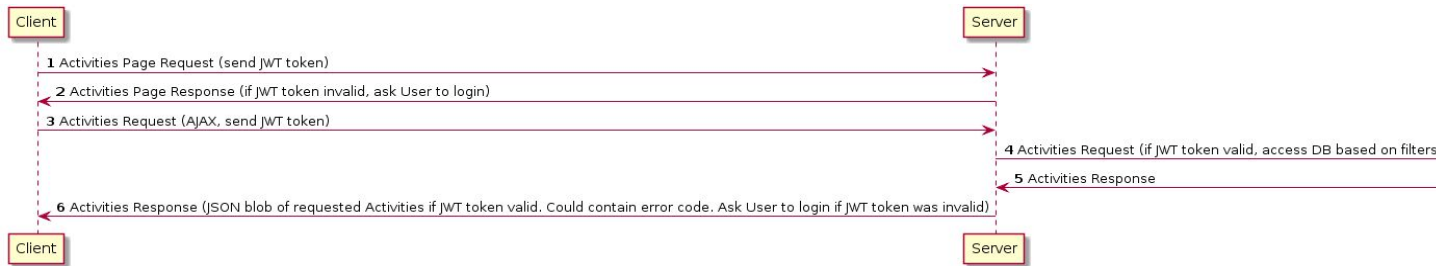
Model Name	SubActivity
Variables	<p>subactivity_id (integer, primary key)</p> <p>activity_id(integer, foreign key)</p> <p>title (string)</p> <p>points (integer)</p> <p>time_completed (DateTime)</p> <p>link (string)</p> <p>content (string) (optional)</p>
Methods	<ul style="list-style-type: none"> • new SubActivity(Int activity_id, String title, Int points, String link): <ul style="list-style-type: none"> ◦ Constructor for creating new SubActivities. Instantiate unique subactivity_id • get_SubActivity_Score(): <ul style="list-style-type: none"> ◦ Returns this SubActivity's score • complete_SubActivity(): <ul style="list-style-type: none"> ◦ Create DateTime object for current SubActivity, call activity_id's complete_Subactivity(this.subactivity_id)

Sequence Diagrams

Login Sequence:



Activities Request Sequence:



Review Board Material:

On Devise:

Devise is a gem, a software library for Ruby on Rails, that makes user authentication easy for programmers. Devise is a layer of abstraction on top of another gem called Warden. Warden acts as middleware and provides basic authentication functionality. With this gem, here is how the authentication process will go:

1. User details are salted and hashed using the bcrypt algorithm. Basic description of the algorithm: <https://en.wikipedia.org/wiki/Bcrypt>.
2. The middleware, Warden, will check if authentication details are correct. If they are, then Warden, which sits as an object in the code, will expose another “user” object that contains the information regarding the authenticated user. The webpage will then render itself based on this information.

The official rubygems page is here: <https://rubygems.org/gems/devise/versions/3.5.6>. As you can see, with 15 million total downloads in its lifetime, the library is extremely popular and used in many commercial web applications.

Data Export Schema:

We will be exporting the data in JSON format. For each user, we will assign some label that preserves their privacy, such as “user 1”, “user 2”, etc. Then, we will have already been recording their activity along with times. This will be included along with the current activity list and experimental state. The review board can be assured that this process is a rather trivial matter, and the privacy of the experimental subjects will most certainly be preserved.

Implementation plan:

Workload Distribution + Progress

[Eric's Contributions]

[Eric] - Wrote devise writeup and security review for review board. Wrote up Data export Schema Plan

[Eric] - Quit user functionality now working and records quit_time and user_id.

[Owen's Contributions]

[Owen] Design Doc (2/29) - 2 hours Functional Requirements, technical stack descriptions, database model schemas, API table of contents, model methods, sequence diagrams, Iteration 1 implementation plan.

[Owen] Rails Webapp - (3/7). 3 hours Basic todo app backend created with Activity model that contains just a content. Ability to create new activity, edit an activity, delete an activity, and show all activity in a list. Basic temporary styling. Next step to create user login authentication and account creation.

<https://www.youtube.com/watch?v=jNa1oNtB6c4&feature=youtu.be>

[Owen] Rails Webapp - (3/15). 6 hours Created User authentication and login accounts with Rails Devise library. User models created with password encryption for signing up users. Activities model updated with user_id fields to connect User_id with associated activity post. Validation to ensure user must be logged in to carry out actions.

- Database Setup (1/2 DD)
 - ⊖ Create User, Interest, and Activity models
 - ⊖ Put dummy data in the database
- Users (1 DD)
 - ⊖ Create the User class
 - ⊖ Dependent on Users being set up in the DB
- Activities (1 DD)
 - ⊖ Create the Activities class
 - ⊖ Dependent on Activities being set up in the DB
- API Controller (2 DD)
 - ⊖ Create function calls and responses for User login, getting all Activities, getting Activities of an Interest, getting Interests, and getting User's Interests
- Login/Signup View (1/2 DD)
 - ⊖ Pull data from the back end (with Backbone.js)
 - ⊖ Depends on the User DB setup, and the User class being created
 - ⊖ Basic styling

[Owen] 3/24 - 6 hours

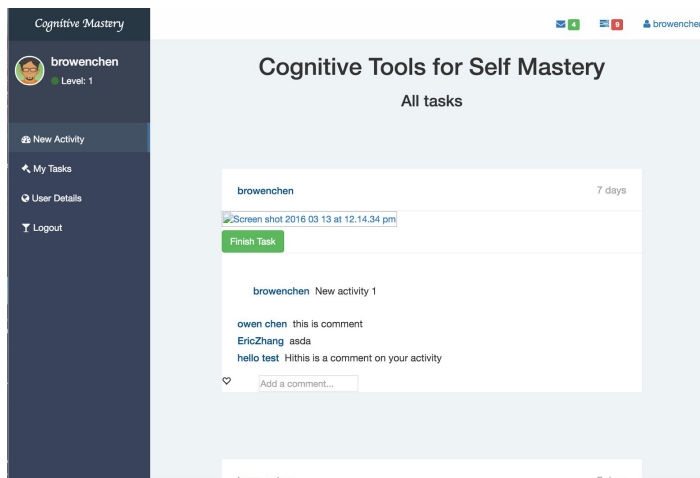
1. New Activities - build user authentication and account creation, tied user_id of each activity to the user_id of the user that is logged in session.
 - a. Different users can create their own activities tied to their user_id with foreign key. Added checks constraints to prevent other users from modifying other activities that are not theirs
2. Comments to each todo list activity (extensible for user's notes for each todo task)
 - a. Created comment model with association to specific activity_id with associated user
 - b. Each comment is created with association to the user that is logged in.
 - c. Creation, deletion functionality for each comment
 - d. Basic scss styling and formatting of forms for structure.
 - e. AJAX calls in adding and deleting comments (automatically reloading data without call to the server for entire page)
 - i. Routing partials/templates for activities, comments form
 - ii. Create.js + destroy.js for AJAX
3. Backend routing - display all activities/leaderboard page + only displaying personal tasks page for specified logged in user.
 - a. My Tasks page displays logged in user's todo task
4. User Details page
 - a. Shows user details, score, level, experimental condition
5. 2) User Model, create migrations to add in following
 - a. `--score (int)`
 - b. `--level (int) [with relevant level markers]`
 - c. `--Experimental condition string, randomly assigned upon user creation`

[Owen] 3/26 - 3 hours

1. Started building interface. Build grid system for navigation and main papp
 - a. New navbar.scss partial, built framework for left navbar
 - i. Avatar for user, displays name + current level
 - ii. Nav links on left sidebar + set up hover/nav transitions

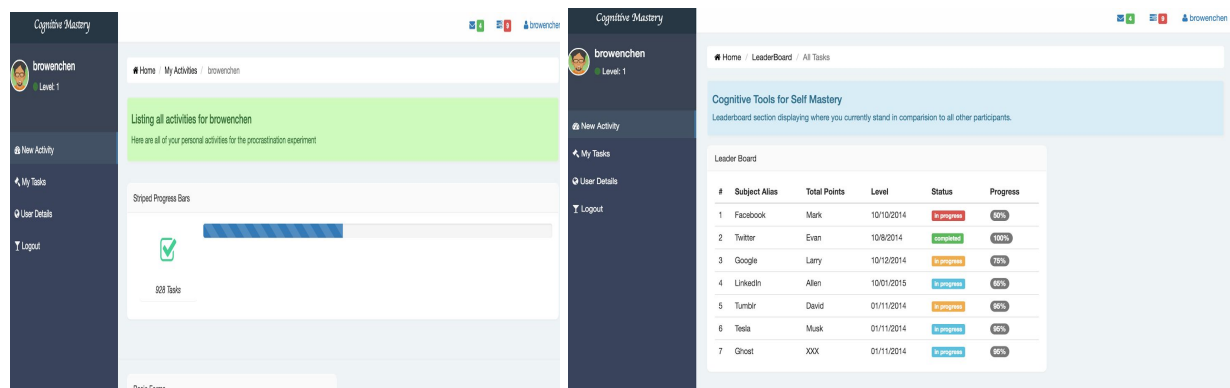
[Owen] 3/27 - 3 hours

- b. Imported font-awesome rails gem. Formatted navbar, needs bug fixes.
 - c. Top nav bar with user name and dropdowns. View all task dropdown with animation, tasks to pull in for user, and progress for each task.
 - d. Routed stylesheets + finished first round of styling both nav bars



2. Main app container styling - description paragraphs + headers/navigation display

[Owen] 3/29 - 2hr Main body for app - leaderboard table display, my tasks displays. Building containers and components for each activity details to be shown in the "My Tasks" View

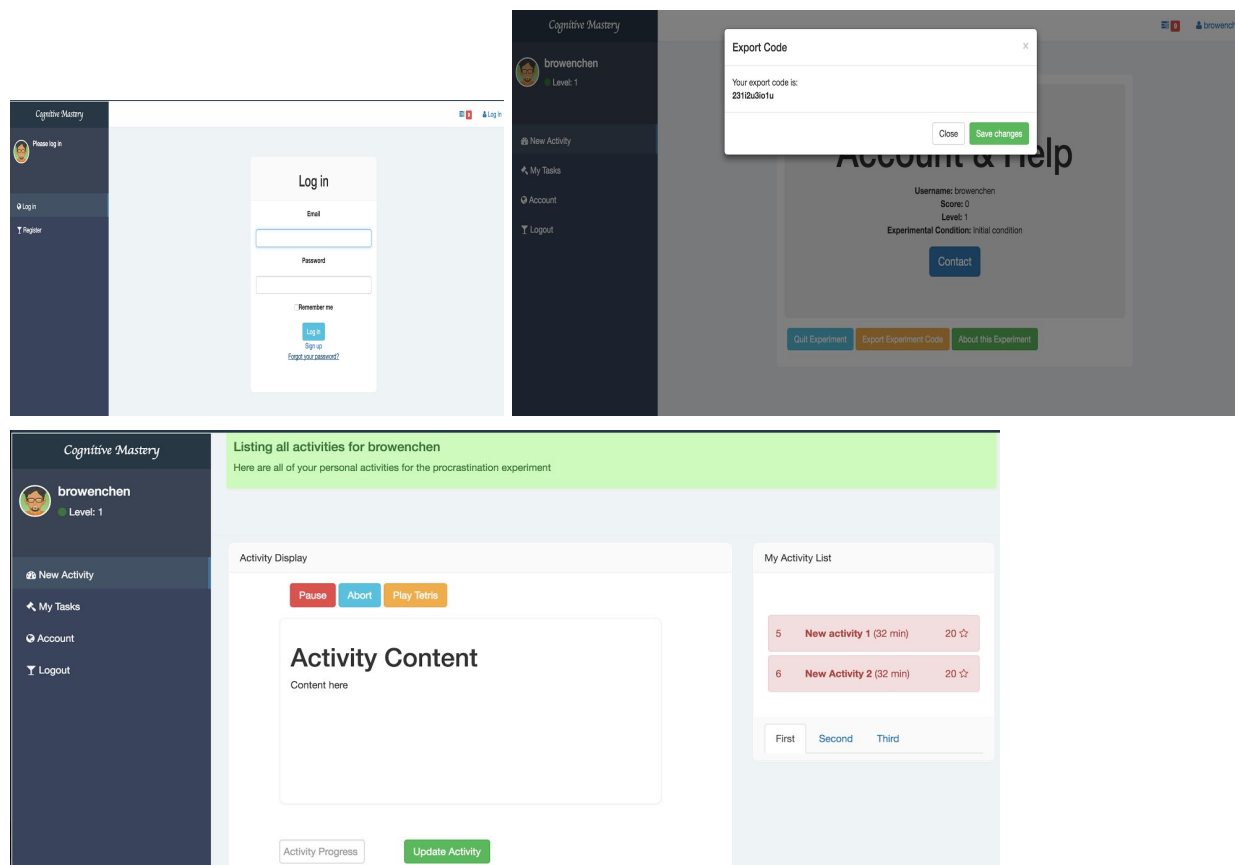


[Owen] 4/1 - 3hrs Created updated my activities view, two containers with left one displaying current activity and right container as a list to display all activities for that user. Left container displays with selected activity

[Owen] 4/3 - 2 hours Changed main prototype UI to Falk's specifications (activity list and container).

[Owen] April 5th, 2016 - 3 hours Changed account page to show Falk's specifications of User details, about experiment page, contact button,

Images: Log in/Account creation flow, Activity main prototype, Account page,



[Owen] 4/7 - Changed layout of activity container panel into a partial that contains the pause abort play tetris buttons + update activity button. Selecting a different activity will now change that entire container. Now has a container to host the experiment as well

[Owen] 4/11 - 3.5 hrs Hooked up activity panel tasks with linking to relevant container task and auto generating id's of each activity in the list for a tag element.

-Finished routing leaderboard and displaying users on a leaderboard.

[Owen] 4/12 30 min Linked play tetris button and embedded tetris game into container when clicked.

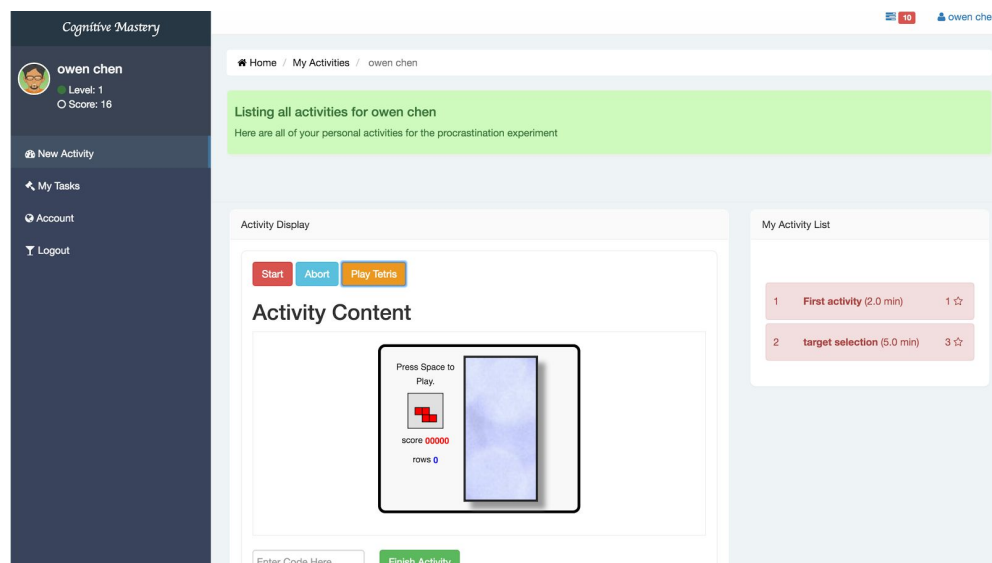
- How to do activity finish and update user score function
 - Click a button that takes current activity_id, get that activity_id's points and the current user. Update user score with activity_id's points.

[Owen] 4/13 -30 min Hooked up activity finish button in the activity panel view with controller, view, and routes. Needs to take current activity's id, call it from the db in the backend, and update current user with that activity's points.

- Hooked up update score function
 - Need to hook up activity id and userid look up
 - Automatically refresh user detail portion when user score updates

2 hr - Couldn't figure out client-side/server side communication for passing activity id of selected activity to finish activity button where it queries the database with that activity id to update current user's score with that activity's score

[Owen] 4/13 - 45 minutes - Meeting with Falk - Adjusted embedded Tetris game to fit container. Bug with Vigilance Experiment and not automatically resizing to fit our container (so a scroll bar appears). Maybe resize original vigilance experiment?



- Added activity's code parameter in model schema in db. Rake db:migrate

[Owen] 4/17 - 1.5 hr - met with Eric. Went over tasks to do. Figured out workaround for activity selection issue by tailoring to first prototype and hard-coding the href links with activity id and implementing unique activity id's in the database. Original issue was, on client-side, href dynamically links to the activity id's that are generated in the partials list. But once a user clicks on a href, (activity in the todo list), the chosen activity is passed around in the front-end, where the relevant container is displayed. But, there was no easy way to query the backend with the selected activity id in the front end to get that specific activity's details for the backend data. This issue relates to client-side compiling of javascript variables, and AJAX calls to the backend, linking them up.

Code encryption and decryption for user scores and compensation was also discussed, it's trickier than it seems. Eric is almost done with priority 3. Merged into master branch

1.5hr - Implemented new backend API GET route for specific activity details. Start button takes currently selected activity ID and hits `/get_activity_detail/:id` route to query the database for that activity's details. Returns JSON response for the client to then query the vigilance game with the `?duration` and `?code` parameters.

.5 hr - Implemented Code checking validation for finished activity and updating user score. Checks input with activity's completion code, and enables a button to update user with a new score once the correct code is input. Automatic page refresh is run when score is updated to reflect change.

4/22 [Owen] .5 hrs - Debugging Eric's `Start_tetris` function and `Quit_experiemment` functions to accurately update db records of a user quitting the experiment or starting tetris.

4/25 [Owen] 2hr - Enabled admin. The admin account will have username "Admin", and the experimenter will create an account, then enter `"/enable_admin/[code]"` into address bar. The code currently is set to 9128, which will enable the username "Admin" to have admin privileges (for adding experiment activities and exporting the data"

4/27 [Owen] 2 hr - (commit logs)

added admin capabilities and routing. Changed leaderboard to show only to admins, changed new activity creation and view to only admins, added admin dashboard under account settings only visible to admins, with buttons for export data and set default activities buttons

added controller and routes for functions 'set default activities' and 'export data'. admin can insert his experiments with add new activity, and hit set default activities button to set his activities as the default activities for all experimenters'

Flow for admins creating activities:

- 1) Click create new activity page
- 2) Enter activity details (name, score, duration, completion code, activity id <- this has to be between 1-10)
- 3) Go to account page, click set default all activities. This is intended to be the way experimenter sets the default activities for all experimenters when they create an account. Still have to figure out a way to do this in the backend
- 4) For granting admin access, enter `"/enable_admin/[code]"`, with the code temporarily set as 9128.

Currently, only the username Admin, can be the admin

STEPS FOR CREATING ADMIN ACCOUNT

1. Create new account, set user name to be "Admin".
2. After account creation. Hit `"/enable_admin/9128"`. This will set `is_admin` to that account as true.

4/28 - [Owen] 2-3 hrs. Enabled setting default activities functionality. Admin creates up to 10 activities, each new experimenter creates an account, hits "Account" page, and hit "Set Activities" button. This queries the backend to find the admin's activities to populate the experimentee's activity task list with the Admin's list.

20min. Finished activities now turn green bar with a checkmark.

4/29 [Owen]. Fixed activity completion bug, querying the correct activity in the database.

- Changed up aborting activities to be logged every time an activity is aborted
- Format in DB
- [#<Quitter id: 15, created_at: "2016-04-28 20:23:47", updated_at: "2016-04-28 20:23:47", user_id: 5, time_quit: nil, tetris_time: nil, **activityAbortTime: "2016-04-28 13:23:47 -0700", activity_id: "1", activity_start_time: nil, activity_finish_time: nil>]>**

Fixed up logging actions, bugs and edge cases

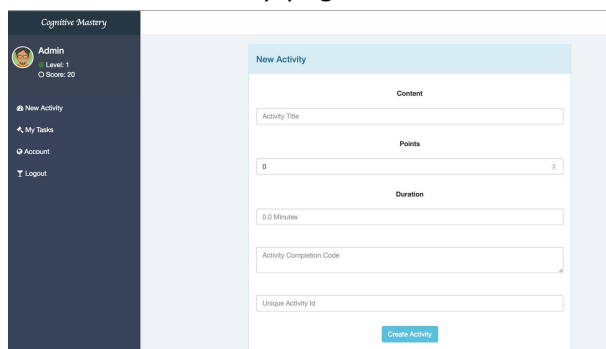
```
=> #<ActiveRecord::Relation [#<Quitter id: 53, created_at: "2016-04-28 20:47:52",  
updated_at: "2016-04-28 20:47:56", user_id: 5, time_quit: nil, tetris_time: nil,  
activityAbortTime: nil, activity_id: "1", activity_start_time: "2016-04-28 13:47:52 -0700",  
activity_finish_time: "2016-04-28 13:47:56 -0700">, #<Quitter id: 54, created_at:  
"2016-04-28 20:48:04", updated_at: "2016-04-28 20:48:07", user_id: 5, time_quit: nil,  
tetris_time: nil, activityAbortTime: "2016-04-28 13:48:07 -0700", activity_id: "1",  
activity_start_time: "2016-04-28 13:48:04 -0700", activity_finish_time: nil>, #<Quitter id:  
55, created_at: "2016-04-28 20:49:16", updated_at: "2016-04-28 20:49:24", user_id: 5,  
time_quit: nil, tetris_time: nil, activityAbortTime: nil, activity_id: "1", activity_start_time:  
"2016-04-28 13:49:16 -0700", activity_finish_time: "2016-04-28 13:49:24 -0700">]>
```

Meeting:

Claim: Needed admin access to complete code generation function (Priority 3).
False.

What the Admin function does:

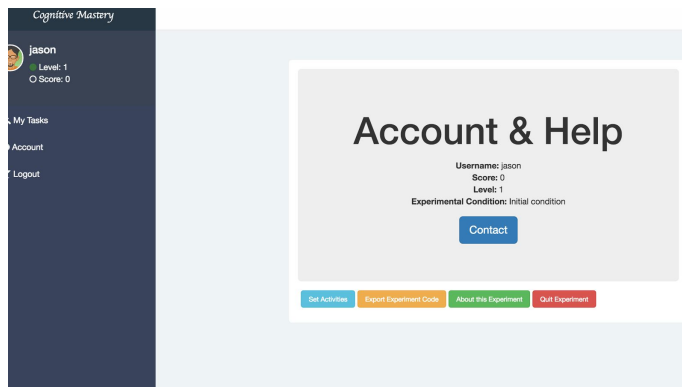
Admin account creates 10 activities. Changed access so that Admin is able to access the "Create new Activity page".



Before - All users could create activities, for purposes of finishing other tasks for the application. Created admin privileges last because it is the least priority in developing the app to work.

How it works:

New experimentees "Sign up" for an account. Then they go to their "Account" page to "Set all default activities" ← This is viewable only for non-admins. (Button on far left)



How this works in the backend.

```
def set_default_activities
  @admin_id = User.where(:user_name => "Admin")
  @activities = Activity.where(:user_id => @admin_id)
  @activities.each do |record|
    puts record
    Activity.new(content: record.content, user_id: params[:current_user], points:
record.points, duration: record.duration, code: record.code, a_id: record.a_id).save
  end
  redirect_to root_path
end
```

Takes the admin's id. Takes the activities of the admin. For each of those admin activities, populate this current user's activities with the same activities, but with his own user id. So now this user has the admin's 10 listed activities.

This is the same as if the experiment user just manually created the 10 activities by himself.

What is required for code generation (specifications below):

The data in the database is the same, regardless if "Set activities" from admin button was enabled or not.

Code generation takes that users activities, which ones are completed, and which ones have not. Then computes a code to display on client side when a button is pressed. **This is not dependent on whether an admin creates the 10 activities, or if, for the purposes of using stub data for finishing the priority, you create the 10 activities yourself. Either way, you must populate the database with activities yourself, for the purpose of testing and making sure the function works. It is irrelevant if you add 10 activities from the admin page, or manually.**

4/29 [Owen] - 2hrs. Worked on Psuedorewards versus displaying no points for the interface.

Currently - There is a points, and no points version. "Initial condition" is the first condition for displaying points and score + level.

Finished export data in json format. User data and activity data

Use this to format JSON response to view data:

<https://jsonformatter.curiousconcept.com/>

An example response for stats log:

```
{
  "id":53,
  "created_at":"2016-04-28T20:47:52.658Z",
  "updated_at":"2016-04-28T20:47:56.394Z",
  "user_id":5,
  "time_quit":null,
  "tetris_time":null,
  "activityAbortTime":null,
  "activity_id":"1",
  "activity_start_time":"2016-04-28 13:47:52 -0700",
  "activity_finish_time":"2016-04-28 13:47:56 -0700"
}
```

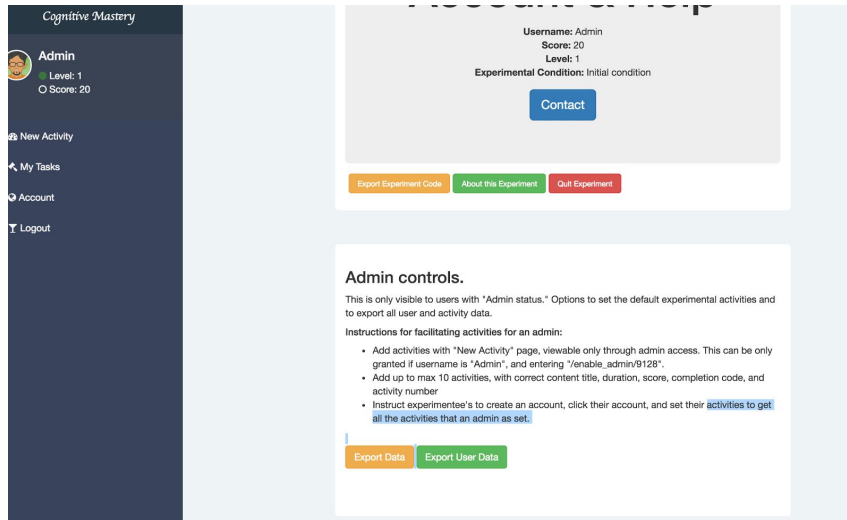
Indicates activity_id of 1, user 5, start time and when the activity was finished. That specific activity was not aborted.

Example of user data

```
{
  "id":1,
  "email":"owenchen@berkeley.edu",
  "created_at":"2016-04-06T20:30:10.846Z",
  "updated_at":"2016-04-28T18:39:58.353Z",
  "user_name":"owen chen",
  "score":278,
  "level":1,
  "experimental_condition":"Initial condition",
  "is_active":false,
}
```

```
"is_admin":false
}
```

Sample Admin Panel (viewable only for admin):



Export data gives JSON of the activities stats. Export user data gives JSON of all users in the database.

4/29 - [Owen]. 1hr 10 minutes. Finished Eric's task. Code generation upon completion.

FORMAT: [activitiescompleted] "letter" [User Id] "letter" [User quit] "letter" [user Experimental condition]

[Activities completed]:: For each activity in that user, 1 is completed, 0 is not.

So if a user has 10 activities

010010101. Means activities #2, 5, 7 and 9 are completed. The rest are not.

For User Quit: 1 = user is active, 0 is User quit

For User experimental condition: 1 = User sees points, 0 = User does not see points or levels

Example

b0i4d1q1e

Decodes into:

User currently only has 1 activity, and it's not completed

User id is user number 4.

User is active

User sees points and levels.

This had absolutely no relevance to whether we needed admin privileges or not.

Rails logic


```

# Code generation for when users claim their payment
def generate_code
  @user_id = current_user.id
  @activities = Activity.where(:user_id => @user_id)
  @user_experimental_condition = current_user.experimental_condition
  @user_score = current_user.score
  @user_quit = current_user.is_active
  @score = "b"
  @activities.each do |record|
    if record.is_completed
      @score = @score + "1"
    else
      @score = @score + "0"
    end
  end
  @score = @score + "i"
  # User ID
  @score += @user_id.to_s
  @score = @score + "d"
  #user quit
  if @user_quit
    @score += "1"
  else
    @score += "0"
  end
  @score = @score + "q"
  #User experimental Condition
  if @user_experimental_condition == "Initial condition"
    @score += "1"
  else
    @score += "0"
  end
  @score += "e"
  render json: @score.to_json
end

```

5/2 - [Owen] .5 hrs. When user quits experiment, popup to explain what happens and gives them their generated code for rewards for the experiment.

5/4 - [Owen] 2 ½ hrs. Fixed UI issues + display none for smaller window sizes b/c app is not web responsive. Deployed to heroku, but encountered issue with sqlite3 and heroku compatability. Bugs ensued, migration errors from previous migrations conflicted with postgres database. Resolved. App deployed at <https://young-citadel-21432.herokuapp.com/>

5/5 - [Owen] - 30 min meeting with Falk + 45 min added remove activity button for the backend and frontend for admins only. Gives Falk the ability to reset the activities in the experiment

5% - [Owen] -1.5 hr. Went through bug fixes + updated descriptions and fixes that falk listed. Fixed the bug where if an admin "quits" an experiment and is unable to log in, u cant do anything anymore.

- Fixed overpopulation of activities when setting activities bug.
- Does this pose a problem if user clicks set activity multiple times?

1hr - **Major bug.** Completing activities in sequence doesnt work for some reason.

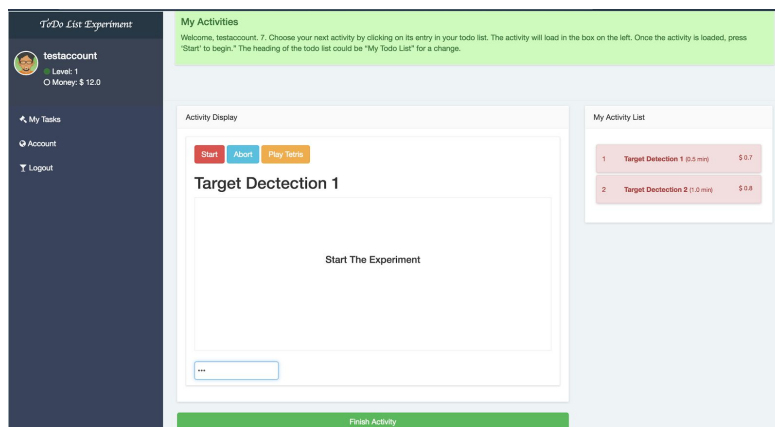
Activity disable finish button is not working

You can still click finish button if the CSS is disabled

Got dollar display feature working. 3 experimental conditions set at random by clicking "Set Activities".

1hr - Fixed the above bug. After much frustrations. Built a workaround. Finish activity button will now only display/shown if 1) activity has not been completed in the database, and 2) if the code is entered correctly.

Finish activity button is now a full button outside of main containers, + build an additional GET request for the finish action for the controller. Something about switching the inner partials was messing up the GET requests for the finish activity buttons.



.5r [Owen] - changed the flow of start. Welcome screen -> "set activities" function can be called only once (if the user is in initial condition). -> the my activities page will then display the instructions on points and money if the user is assigned to points or money condition

***** when a user is in "Initial condition" - no interface is displayed until user hits the "Initialize to-do list" button in the welcome screen. Only then will he see the relevant features of his experiment

Todo Tasks List

Updated Falk first prototype UI and functionalities (April 5th, 2016)



TODO QUESTIONS (April 5th)

- Update predefined Activities list from Falk
- Update Activity Model. Needs Points, progress percentage, is_completed
- Figure out how to hold activity content information. What will activities be, will they have images attached to them or just text? How is the play tetris function implemented?
- What do the pause and abort button specifically do? How to implement activities in this fashion. How do we update activity progress? Time? Manual update on progress? How will this be implemented in the backend
- How to generate code of how much participants will be paid?
- Disabled accounts? Will accounts now have a disabled and active parameter?
- Dynamic updating of activity data, constant communication between DB server and client, AJAX.

General TODO of first prototype

Activities

- Highlight active activities + highlight hovered activity
- Link activities in the task bar to display relevant activity in the activity container

User Account features

- Generating code for how much user is paid. Take user score from DB and translate to code
- Quit experiment, log user out and deactivate user account in backend

Feedback elements

- Updating points, show notice on user's updated points
- Currently do so in lefthand sidebar
- Dynamic responsive app, lots of AJAX but currently not routed with a front-end javascript framework that has easier AJAX abilities

Proposed TimeLine

2/25/16: Write up Design Doc.

3/3/16: Finish up Design Doc modifications, API routes, methods. Set up Express backend stack.

3/10/16: Iteration 1 - Database setup, Users, Activities Setup, maybe get started on API REST calls.

3/17/16: Get User Authentication and account creations set up. Create ability to comment [create sub-activities] on individual activities and relate activities to users. Look into implementing more methods and expand model schemas according to doc.

3/20/16: Finish up backend features by end of March, Start building front-end interface by April

4/12/16: Finish Falk's finalized to-do list by end of month of April

Intended Stages of Development

1. Get backend structure and API running. User account creation and basic todo app functionalities for users. [Tentative week after spring break]
2. Work on a front-end interface and code up front-end app framework [Tentative Spring break start working on it]
3. Go back to functional requirements and add in fine-details to the backend.
 - a. Support exporting scores, leaderboards, and subactivity functionalities [Tentative early-mid April?]

Meeting Notes | Finalized Todo list.

(4/8)

- ~~**Priority (1) Activity box**~~ The container will be a black box. Pause, abort, play tetris.
 - Hosted experiment with html string.
 - **How to communicate back to the task?**
 - <http://cocosci.berkeley.edu/mturk/falk/taskRating/Vigilance/index.html>
 - Parameters of query string—duration
 - Log tetris time and aborting in Quitter class (only one datetime for last time accessing tetris)
 - Each activity also has time of abort parameter
 - **Pause button**—have a pause count and a time of pause.
 - What happens when the activity is paused? There is no way to stop interactions with the embedded experimental game.
 - **Abort Button**—have a time of abort task button.
 - What does this do? Does it also remove the experimental game or tetris game view when abort is clicked? Points don't update?
 - Need a list of all times an activity is aborted, not just the last one.

- This might require additional routing in the backend for a new class to contain the list of all experiments quit.
- tetris
- Activity complete function
 - As a start, task completion can be communicated by the user copying the code generated by the task into a textbox.
 - This would require an input submission button that sends a request to the backend to validate the code generated by the textbox. Or, we connect the input to the "finish activity" button which updates the users score only if the code generated by the task is the correct code.
 - When activity is complete, user hits complete button and update user's score with that activities amount of points.
 - AJAX request in rails
- **Priority (1.5)** — Set admin capabilities for Falk to input the activities to all experimenters
 - Admin user account for falk:
 - He puts in the (max 10) activities that experimenters will do
 - All these activities are then inserted in all user's task list
 - Need to figure out how to automatically give all experimenters all the activities that are predetermined by Falk for the experiment.
- **Priority (2)** Code Generation — Code based on score and user
 - Unique code that encodes information in non-transparent way
 - Don't want users to be sharing codes for the money
 - Unique Score
- **Priority (3)** When they Quit
 - We want to know when and how often users quit.
 - Quit experiment button logs user out and notes the time when a user quits the experiment.
 - Prevent log in if account is deactive?
- **Priority (4)** Dynamic updating of scores and user progress
- **Priority (4)** Leaderboard (not first prototype, but for later)
 - Currently Displays list of users in the database with relevant information
- **Priority** when a user is in "Initial condition" — no interface is displayed until user hits the "Initialize to do list" button in the welcome screen. Only then will he see the relevant features of his experiment
 - User can only set his activities and experimental condition by selecting the button once.

Bugs List

- Responsive sizing issues
 - Temporary fix, disable app when window size gets too small.
- Quit experiment prevents users from logging in again — FIXED
- Trying to finish activity crashes app (undefined method `points' for nil:NilClass')

-
- Fix randomization of experimental conditions upon user creation
- ~~(FIXED) Deployment to Heroku "SQLite3" not compatible. Switch to Postgres on production~~
Major Priority ~~(fixed database migration configuration for heroku after 1.5hr)~~
- ~~Set Default Activities for Experimentees must only be able to be activated once (to avoid overpopulation of activities)~~ **PRIORITY 2**
- ~~(FIXED) Deployed app not showing up tetris and experiments at all. Works locally but not on the heroku server. CORS issue? (Create an account, click on an activity and try to click play tetris. Doesn't show up)~~ **High Priority**
 - ~~HTTPS vs HTTP issue~~
- ~~(FIXED) Allow deletion of activities for admin.:~~
 - Change email access for admin (privacy issues)
- ~~(FIXED - 2 hrs) Major bug. Completing activities in sequence doesn't work for some reason.~~
 - **Activity disable finish button is not working**
 - **You can still click finish button if the CSS is disabled.**
-

Web app link: <https://young-citadel-21432.herokuapp.com/>

Admin account: falklieder@gmail.com

Password: password

4/28/

- Will receive from Falk
 - Text about experiment and instructions
 - List mapping points to levels including levels names
 - Points scheme - when activities finish, points for other activities change.
- Mainly want measurements of user actions
 - ~~tetris time, abort activity, quitting experiment, start activity, finish activity~~
- Bug check why is completed sometimes doesn't work
- * **Activity points change when activity is completed**
- * **3rd condition. Allow for heuristics. Same display as points condition but different points.**
 - **Make experimental conditions uniformly distributed**
 - Putting this function under setting default activities in controller

CFor next Meeting

- Owen exporting data function call, priority (5).

Deploy app for testing for Falk Let's change the text on the button from "Export Experiment Code" to "Generate Bonus Code".

About this Experiment: "This HIT is a psychology experiment being conducted by Falk Lieder, a graduate student in Professor Tom Griffiths's Computational Cognitive Science Lab at the University of California, Berkeley.

This study is designed to investigate the effects of different types of digital to-do list tools on productivity and motivation.

Your participation in this research is voluntary. You may refrain from answering any questions that make you uncomfortable and may quit the study at any time without penalty.

If you complete all of these tasks by 3pm on Sunday <Month> <Day> 2016, you will receive a bonus of \$20. If you don't, then you will receive a much smaller bonus of about \$3 per hour that you worked on the tasks we assigned you."

Quit Experiment: "Are you sure you want to quit the experiment for good? If you quit now, there will be no return. You will not be able to resume the experiment and you will not be able to earn the \$20 bonus for completing the experiment. Instead, you will receive a much smaller bonus of \$3 per hour that you worked on the tasks. If you choose to quit, please make sure to copy and paste or write down the following bonus code: <CODE>.

To collect your bonus, please log in to Amazon Mechanical Turk, find the HIT called "Reimbursement for Todo List Experiment" by the requester "CoCoSei Lab at UC Berkeley" and paste this code."

Let's add a "Cancel" button in the quit screen.

We would like to run the experiment with two different kinds of bonuses: immediate monetary rewards and points. Thus, there should be a third display condition, in which the points are displayed as dollars rather than stars.

In the star conditions, the "Generate Bonus Code" button should be disabled until the participant has completed all of the tasks. In this condition, the text on the code screen should say "Congratulations, you have earned the \$20 bonus. Your code is <CODE>. To collect your bonus, please log in to Amazon Mechanical Turk, find the HIT called "Reimbursement for Todo List Experiment" by the requester "CoCoSei Lab at UC Berkeley" and paste the code there."

In the dollar, condition the "Generate Bonus Code" button should always be enabled. In this condition, the text on the bonus screen should say "You have earned \$X so far. Your code is <CODE>. If this is the first time you collect a bonus, you will be paid \$X. If you have collected part of this bonus before, you will receive the additional bonus that you have earned since then. To collect your bonus, please log in to Amazon Mechanical Turk, find the HIT called "Reimbursement for Todo List Experiment" by the requester "CoCoSei Lab at UC Berkeley" and paste the code there."

The title "Cognitive Mastery" is nice and I like the font. But for the first experiment let's make the title "Todo List Experiment". We can return to "Cognitive Mastery" for the public version.

~~I have tested the functions that are implemented so far and they seem to work well. I only have a few minor suggestions. See below:~~

~~I have e-mailed you the list of levels and the point schemes are finalized. Currently, the level does not advance as points are accumulated:~~

~~Let's make the points and level more salient. For instance, we could move the display of levels and points into the user's field of view, increase the font size, make them bold, and highlight them in green. Likewise, the points on the todo list should be the largest and most salient element of each item.~~

~~For money, let's display the amounts in the format "\$X.XX".~~

~~Let's change "Start the Experiment" to "Press 'Start' to start the task."~~

~~When performing one of the task, I automatically use the space key to advance the task. But the space key scrolls down and does not advance the task even though the task says "Press any key."~~

~~The task completion code should be generated at random such that it is different for every task and for every user. Otherwise users might not have to do the task to guess the code correctly.~~

~~When the user finishes the last task, it would be nice to update the information above the todo list to something like: "Congratulations! You have completed all of the tasks. Please navigate to 'Account', click 'Generate Bonus Code' and redeem your bonus on Amazon Mechanical Turk."~~

~~When I say that I forgot my password and request to be e-mailed resetting instructions, then I get an error: "We're sorry, but something went wrong." I think we could reword to say that this is also for forgetting your user name. If the bug is hard to fix or the feature is difficult to implement we could leave it out.~~

~~When I tried to sign up with a taken e-mail address, the log-in button was pushed down so far it was almost invisible.~~

~~When the deadline passes, the tasks should be disabled and the text above the todo list should say something like "The deadline has passed and the tasks have been disabled. Please collect your bonus for the tasks that you have already completed. Completing additional tasks is no longer possible."~~

Steps to create local development version

1. Install Rails 4.16: `rvm gemset create rails-4.0`
2. Install PostgreSQL
3. `ARCHFLAGS="-arch x86_64"` bundle install
4. `rake db:migrate`
5. `rails s`
6. localhost:3000
7. Sign up as "Admin"
8. localhost:3000/enable_admin/9128

TODOs:

1. Learn Rails and Ruby
2. Add control condition with constant number of points:
 - a. Add “control_condition” to experimental_condition in activities_controller.rb
 - b. Changes points argument of the Activity.new command in line 286
3. Populate todo list as admin
4. Test the different conditions
5. When everything seems to work nicely:
 - a. Deploy on Heroku
 - i. Get lab’s heroku credentials
 - ii. Follow instructions in <https://devcenter.heroku.com/articles/getting-started-with-rails4>
6. If we want to recruit student developers: try to contact students who took CS169 (maybe ask prof. teaching that course for recommendations).

Questions

1. What remains to be done before we can start running the first procrastination experiment? How will we do it?
2. Which conditions are currently implemented?
3. What would I do to add another condition with a different points scheme? (e.g., control condition where each task receives the same number of points)

4. Can you recommend a tutorial for learning Ruby on Rails for somebody who does not know Ruby yet?
5. To which extent do you want to stay involved with the todo list gamification project?
Can I test changes that I make to the code locally? If so, which web-server package should I install to create a local development environment?
6. What should I and/or future developers who know Ruby on Rails know about the codebase to be able to develop it further? Is everything we need to know already in the design document?
7. Can we contact you with questions about the code in the future?
8. The files in app/models all appear to be empty. So where do you define the user model and the activity model? I think I found it in the db folder.
9. What would I have to do to change the points assigned to the activities?
10. How would I change the list of activities presented to new users?
11. What would it take to allow users to create their own todos?
12. The files for the rewards_controller.rb and todos_controller.rb appear to be empty.
13. Can you explain the instructions in the README.rdoc? Are they for setting up a local test/development environment?
14. A brief explanation of the folder/file structure would help.

7/30 Final Iteration Bug Fixes.

Logs

[Start] 7/30 - 3:26p.m

New Additions since last iteration

- Control condition w/ constant points
- Modified schema.rb file directly ←--- potential build issues here
- Added csv file and reader
- Created points model
- Deduction of points when user aborts activity
- Change how admin_id is determined from default activities

Pulled fresh repo. Created admin account. Initialize Todo-list bug
"NoMethodError (undefined method `constant_point_value' for
#<Rails::Application::Configuration:0x007ffd7d075120>):
app/controllers/activities_controller.rb:345:in `set_default_activities'
"

****Bug***

Config global variables not initialized. bonus, deadline, etc

Logic flow -

- 1) Create admin account
- 2) Enable admin

Bugs - constant point value is calculated using nr_activities, but there are no activities.

Bug - "Set_default values" is called twice in enable admin.

[Bug fixed] - Tested flow - created admin, enabled admin, made activities, made new user, initialized their todo-list

Issue

Potential bug may occur in the flow if 1) create new app and admin account, and the admin clicks initialize to-do list before enabling himself as admin. Default global variables are set under the "enable_admin" function, which may cause future bugs down the road.

****App crashes if Admin clicks initialize to do list before enabling admin****

Bug Deploying test to heroku. Application failure.

Crashes because heroku database is not migrated yet

Use command

```
heroku run rake db:migrate
```

Bug App crashes when enabling admin.

Debugging. Enabling heroku rails logs

Heroku logs --source app -t for live logs in heroku

4:37p.m

Getting rid of set_global in enable admin lets app work

.

[Bug Fixed] - Now enable_admin set global and initialize todo works

Bug. Crashes when creating new activity

<https://blooming-tundra-76896.herokuapp.com/>

[End] 4:41p.m. Total time 1hr 15 minutes

[Start] 8:16 p.m

Bug. Admin_id is causing infinite loop and app to crash. [Nevermind]

“

```
if Activity.where(:user_id => params[:current_user]).exists? && params[:current_user] !=  
  @admin_id
```

“

Bug. Logic flow bug, *enable_admin must be called every time before set_default activities or else app will crash.* This is why the app crashes and cannot find constant global values.

Moved all config global variables under production.rb and development.rb

Bug. Infinite loop

SQL (0.0ms) DELETE FROM "points" WHERE "points"."id" = ? [["id", 63074]]

[End] 8:42p.m

Bug

```
nr_points = Point.where(activity_id: record.a_id, state: 0, condition:  
condition)[0].point_value
```

Bug

```
Activity.new(content: record.content, user_id: params[:current_user], points:  
record.points, duration: record.duration, code: @unique_code, a_id: record.a_id).save
```

Created new points model, was activity model schema updated correctly?

[Start] 8:56 p.m

Bug

```
nr_points = Point.where(activity_id: activity.a_id, state: get_state_id, condition:  
condition,time_left: remaining_time_steps)  
Nothing is returned from the SQL query and trying to access .point_value from null  
object
```

Traced the bug to get_state_id

```
nr_points = Point.where(activity_id: activity.a_id, state: get_state_id, condition:  
condition,time_left: remaining_time_steps)
```

Passing in this function into the points SQL query resulted in an error

[End] 9:46p.m

[Start] 11:47a.m 7/31/16

Checking flow of app, when is Points table created + initilized. Potential bug here for SQL points query crash

Bug issue - Points SQL called before points are initialized in the database

#Bug, calling POINTS Sql when these values do not exists yet
nr_points = Point.where(activity_id: activity.a_id, state: @state_id, condition:
condition,time_left: remaining_time_steps)

Where is create_points_table, function that parses csv file for points, called?

****Testing Create_points_table**** infinite loop inserting into database
Points table bug. There were 100,000 records of points. Why are these needed?

----- Debugging creation of points flow. This may need to be changed ---

80 entries for points.csv.

403200 / 2 entries for all_points.csv

80x2 + 403200 points records in the DB. This is much really needed?

Running create_points_table very slow for creating all records. Don't think points should be created as models in the db. How is this implemented + used in the app?

“

```
Point.create(activity_id: row["activity_id"].to_i, state: row["state_id"].to_i, point_value:  
row["point_value"].to_i, time_left: @total_time-row["time_step"].to_i+1, condition: "points  
condition" )
```

```
Point.create(activity_id: row["activity_id"].to_i, state: row["state_id"].to_i, point_value:  
row["point_value"].to_i, time_left: @total_time-row["time_step"].to_i+1, condition: "monetary  
condition")
```

“

Running these very identical db record creation is extremely inefficient, if the only difference is condition of each record. Database lookup, deletion, insertion very slow with these records. This in turn will create the entire app extremely computationally heavy and slow, if database queires for points will be frequent.

[Break 12:22p.m]

[Start 8/7 10:16 a.m]

Clicking “My Activites” from admin account each time re-populates ~100,00 - 400,000 new db records, taking >10 minutes.

[End 8/7 10:40 a.m]

[Start 8/14 11:19am]

Restarting with repo. Git reset --hard to owen's last working commit. Plan to manually rebase Falk's changes and new csv files while testing app

11:51am. Rebasing Falk's additions while testing. Added constant points feature

Rebased another commit.

11:56am Rebasing csv reader function

12:13am. Almost finished the csv importing function + testing. Ending now
[End 12:13am]

[Start 8/14 4:13p.m]

Imported csv files, CSV reader works. Enable admin correctly removes previous todo list and reads todo_list csv for new data to insert into activities.

Created a new account and activities were correctly initialized

Created revert commits to rollback to previous working repo without changes. Will merge into master, and merge new changes branch into the reverted master afterwards

TODO:

- create the Points model and refactor previous Activity handling. Not sure if adding Level's model yet.
- Setting deadline
- Adding global config variables
- Set current point values method. Taking into account deadline calculations
- Automatic deduction of points when user aborts activity (rufus scheduler)
- Load _break

[End 4:28p.m]

[Start 8/16 9:58p.m]

Starting Points model

Generated migration file, function for reading from points csv.

[End 10:18pm]

[Start 8/12 11:32am]

Working on this diff

<https://github.com/BrowenChen/Cognitive-Tools-for-Self-Mastery/commit/9cb6f60cf71f758e28b64e1ba8e82964b94a5147>

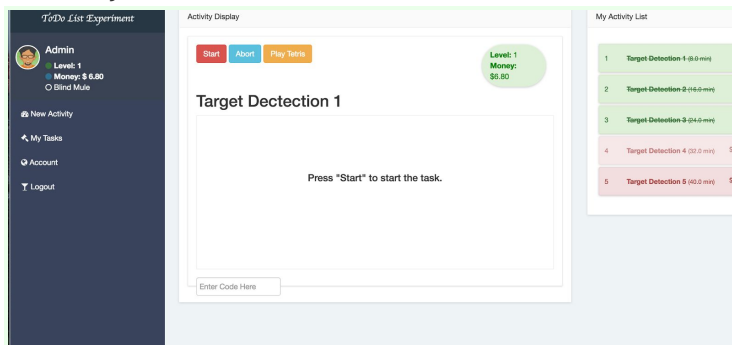
Debugging set default activities with the points system.

[End 1:14pm]

[Start 8/27 1:58p.m]

Debugging app with new points model.

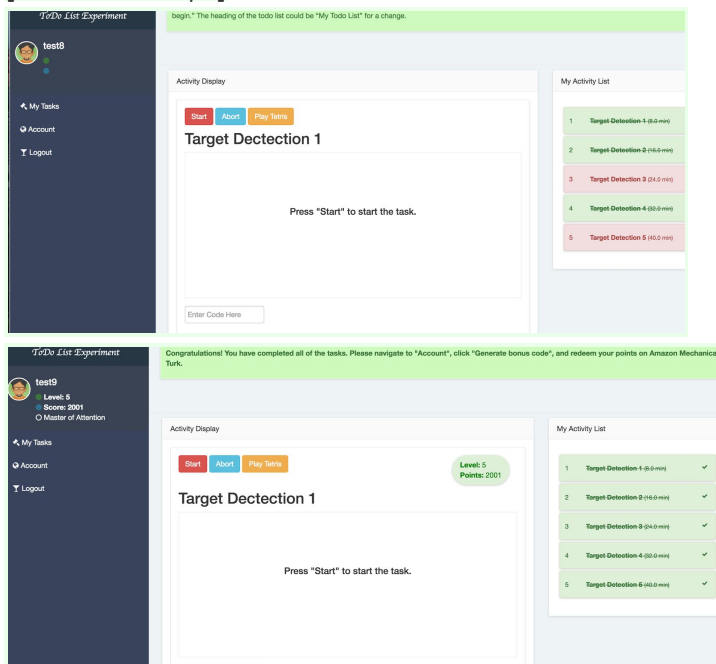
Currently works



Bug.test8@gmail is control condition with no points. But now app crashes because there is no checking for setting and calculating points when the user has no points

Fixed with guard condition when finishing current activity

[End 8/27 3:04pm]



-----End of work-----

Additional testing, refactoring, unpaid

[8/02 S 1:39A.M]

Created global class variables. Fixed up to June 9th commit.

TESTING

- 1) Dropped all database. Created admin account. Created test user. Completed all activities + UI updated to show user is finished with correct guides
- 2) Generate Bonus Code works and is enabled when user finishes all activities.
- 3) Constant Point condition -> Correctly creates activities with constant "5" Point values
- 4) Put global class variables in activities_controller. Constant_point_value based on bonus * 100 / nr_tasks. Points = 400?

[End 2:17 A.M]

09/16/16

[Start 12:07p] New macbook.

Cleanup and refractoring. Debugging bugs. Nr_tasks should be moved to enable_admin function in case admin is not initialized and causes a crash

Adding rufus scheduler for deduction of points.

[Lunch 12:38p]

[Start 1:37p] Adding load_break_points function

Break_points.csv > 80k entries? X2 Point records per entry = 160k point records?

Added compressed_break_points and compressed_points.csv.

Points loaded from compressed_points is negative

"

y_id" = ? AND "points"."state" = ? AND "points"."condition" = ? [["activity_id", 5],
["state", 0], ["condition", "points condition"]]

Point value is

Number points is not nil

-519

Current Point values

10

-51

-166

-324

-519

“

[End 2:16p]

[Start 8:31p 09/17/16]

Debugging, why are points showing up as negative when pulling from full points.csv

Compressed_points.csv has negative point entries. Error?

Refractor + cleanup + debug

Added update_score_and_points function

[End 9:21p]

[Start 10/02/16 2:23p.m]

Implementing deduction of points if user aborts.

Def remaining_time_steps

Debugging set_current_point_values

```
@nr_points = Point.where(activity_id: activity.a_id, state: get_state_id,  
condition: condition, time_left: remaining_time_steps)[-1].point_value
```

May want to change the way Point records are held in database and used for time_left parametr. Nilclass bug happening when trying to do lookups for point records with specific time_left values that might not exist in the database. Can be cleaner to avoid duplication of Point records and do point updates dynamically by calculating time_left in the controller and changing the Point record, having one Point record tied to each unique Activity.

[End 3:31p]

Start 4:19p

Debugging Rufus scheduler updating points

Log

```
Quitter Load (0.5ms) SELECT "quitters".* FROM "quitters" WHERE "quitters"."user_id" = ?  
[["user_id", 11]]  
{ 70215838143840 rufus-scheduler intercepted an error: (0.1ms) SELECT COUNT(*) FROM  
"activities" WHERE "activities"."user_id" = ? [["user_id", 11]]
```

```
70215838143840 job:
```

```
70215838143840 Rufus::Scheduler::EveryJob "30s" {}
```

```
70215838143840 error:
```

```

70215838143840      70215838143840
in scheduler 70215838143840      NoMethodError

5
70215838143840      undefined method `point_value' for nil:NilClassUser was slacking
70215838143840
/Users/owchen/Code/CognitiveTodo/Cognitive-Tools-for-Self-Mastery/app/controllers/activities_c
ontroller.rb:538:in `block (2 levels) in update_score_and_points'
70215838143840
/Users/owchen/.rvm/gems/ruby-2.3.0/gems/activerecord-4.2.4/lib/active_record/connection_adapte
rs/abstract/connection_pool.rb:292:in `with_connection'
70215838143840
/Users/owchen/Code/CognitiveTodo/Cognitive-Tools-for-Self-Mastery/app/controllers/activities_c
ontroller.rb:510:in `block in update_score_and_points'
70215838143840
/Users/owchen/.rvm/gems/ruby-2.3.0/gems/rufus-scheduler-3.2.1/lib/rufus/scheduler/jobs.rb:234:
in `do_call'
70215838143840
/Users/owchen/.rvm/gems/ruby-2.3.0/gems/rufus-scheduler-3.2.1/lib/rufus/scheduler/jobs.rb:258:
in `do_trigger'
Updating score according to break points
In get state id

70215838143840
/Users/owchen/.rvm/gems/ruby-2.3.0/gems/rufus-scheduler-3.2.1/lib/rufus/scheduler/jobs.rb:300:
in `block (3 levels) in start_work_thread'
70215838143840
/Users/owchen/.rvm/gems/ruby-2.3.0/gems/rufus-scheduler-3.2.1/lib/rufus/scheduler/jobs.rb:303:
in `block (2 levels) in start_work_thread'
70215838143840
/Users/owchen/.rvm/gems/ruby-2.3.0/gems/rufus-scheduler-3.2.1/lib/rufus/scheduler/jobs.rb:289:
in `loop'
70215838143840
/Users/owchen/.rvm/gems/ruby-2.3.0/gems/rufus-scheduler-3.2.1/lib/rufus/scheduler/jobs.rb:289:
in `block in start_work_thread'
70215838143840      tz:
70215838143840      ENV['TZ']:
70215838143840      Time.now: 2016-10-02 16:43:00 -0700
70215838143840      scheduler: Activity Load (0.2ms) SELECT "activities".* FROM "activities"
WHERE "activities"."user_id" = ? AND "activities"."is_completed" = ? [["user_id", 11],
["is_completed", "t"]]

70215838143840      object_id: 70215866095860
(0.2ms) SELECT COUNT(*) FROM "activities" WHERE "activities"."user_id" = ? [["user_id",
11]]
70215838143840      opts:
completed activities is not nil
70215838143840      {}
70215838143840      frequency: 0.3
70215838143840      scheduler_lock: #<Rufus::Scheduler::NullLock:0x007fb8cf819108>

```

```
70215838143840 trigger_lock: #<Rufus::Scheduler::NullLock:0x007fb8cf8190e0>
```

End 4:57p

[10/09/16 Start 10:14am]

Starting on fixing rufus scheduler.

If the points in csv and breakpoints are correct, it should work with the updates.

Deploying to heroku

Scaling up web dynos,

[End 10:36a.m]

[10/16/16 Start 6:55pm]

Updated deliverable date, 10/31. No more requirement of updating points dynamically via time, as it was difficult to implement successfully. Instead, points are just updated via state for the deliverable.

Added documentation above methods.

[End 7:15pm.]

Fixed CSV import issue

```
File.join(Rails.root, '/path-to.csv')
```

10/30

Deployed at

<https://ancient-river-53152.herokuapp.com/>

Admin account:

Email: falklieder@gmail.com

Password: password

Username: Admin

Open the repo.

Deploy to heroku.

Heroku run rake db:migrate.

heroku ps:scale web=1 (Scale a web dyno)

Create an account with "Admin" as the username (!Important)

Once logged in as Admin, hit the /enable_admin/9128 route first.

Subsequent users can now initialize their to-do lists via "Initialize todo list" Button.