Purpose and Goals

Anagrams with Fiends is an online implementation of the word game Anagrams, in which players (called "fiends") draw letter tiles from a bag, and build words out of these tiles and previously-formed words. Fiends put the words they build into their stash, and they can swipe a word from another fiend stash if they add letters to that word, forming a new word. The winner is the fiend who has the most words in his stash when the bag runs out of tiles.

Anagrams is a fun game to play in person with physical tiles, but Anagrams with Fiends has several benefits over the physical version. Players don't have to be physically together to play it, and if the app becomes popular, it will be easy to find opponents to play with. Players will be spared from using physical tiles, which take effort to keep organized on the game table and to clean up. Anagrams with Fiends will also keep track of previous games played, potentially providing support for replays of old games, leaderboards, and assisted or automated matchmaking.

Revised (new goal statement): Our main goal is to provide a convenient way for people to play Anagrams, and perhaps to help them discover how much fun the game is. It would also be great if our app served as a proof-of-concept, inducing other game websites to add Anagrams to their online game suites.

Implementation-wise, we aim to provide users with rich meta-game features, such as a high-score leaderboard and ranking system, and a lobby system with challenge capability.

As success metrics, we’ll consider our app moderately successful if at some point a member of our team tries to use it to procrastinate, and fully successful if two people not associated with 6.170 voluntarily play a game with each other.

Key Concepts

Lobby: The set of currently active tables and currently online users which serves as the hub for users to find games. A user may challenge another user within the lobby. **Revised:** The lobby will also contain links to the leaderboard and to the user's replay log.

Challenge: From the lobby, a user may issue a challenge to another user. A challenge may be either accepted or rejected by the recipient. If the recipient accepts the challenge, a game is created and started with those users; if they reject, then the challenge is removed and no game is created. **Revised:** A player may only pose one challenge at a time.

Fiend: A fiend represents a single actor in a game of Anagrams; in each game, each user controls one fiend. Each fiend, during the game, owns a set of words, called their **stash**.

(Revised) Table: A game is played by fiends at a table, which contains the bag, pool, and stashes.

(Revised) Tile: A tile is a representation of a letter which can be moved around to form words. At any point during a game, a tile is in one of three locations: inside the *bag*, in the *pool*, or inside a *word*. Unlike certain other word games, Anagrams with Fiends has no concept of the point value of a tile. Any two tiles with the same letter are functionally identical. (Also note, Tile is only a user-facing concept, not one included in the data model; we discuss this in the notes on the data model.)

Bag: In each game, the bag is a set of tiles from which tiles are randomly drawn to form the pool. Each bag starts out with the same 144 tiles as in a game of Bananagrams. The ordering of tiles in the bag are not known to the fiends.

Pool: The pool is the set of free tiles which fiends can use to build new words, upgrade their own words, or swipe other fiends' words. A tile is added to the pool from the bag when one of three things happens: (1) when the pool is empty, after a short countdown; (2) after some fixed time has passed since the last tile was added to the pool, probably about 10 seconds, or (3) if all the players agree to add another tile to the pool.

(Revised) Flip: If all fiends agree, a new tile is *flipped*, removing it from the bag and placing it into the pool.

Word: A word is a list of tiles whose letters spell out an English word, as defined by a predetermined wordlist (specifically, SOWPODS). Words must be at least three letters long, and at any point in time during a game each word belongs to one fiend. Fiends can *build* words by choosing a subset of the tiles in the pool and specifying a word formed from those letters. Fiends can also *upgrade* one of their own words into another word, or *swipe* a word from another player by making another word. (More on upgrades and swipes below.)

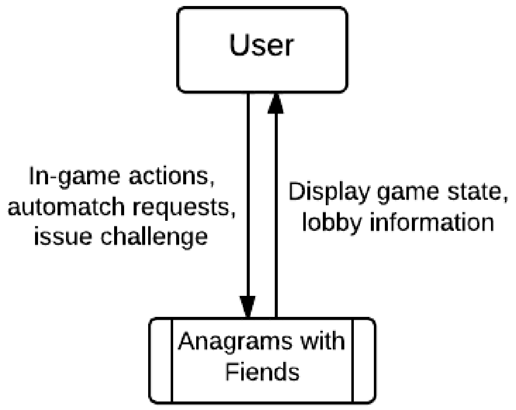
Building: A fiend may *build* a word from any subset of the tiles in the pool. Doing so removes the tiles from the pool and places the word in his stash.

Upgrading/swiping: A fiend may combine an existing word with tiles from the pool and rearrange those tiles to create a new word, which then goes in her stash. If the word in question originally belonged to her, this is called *upgrading*; if it belonged to another player, it is called *swiping*. **Revised:** Internally, we refer to any upgrade or swipe as a *morph*. However, we don't expose this terminology to the user anywhere, and it's not necessary for understanding the game rules.

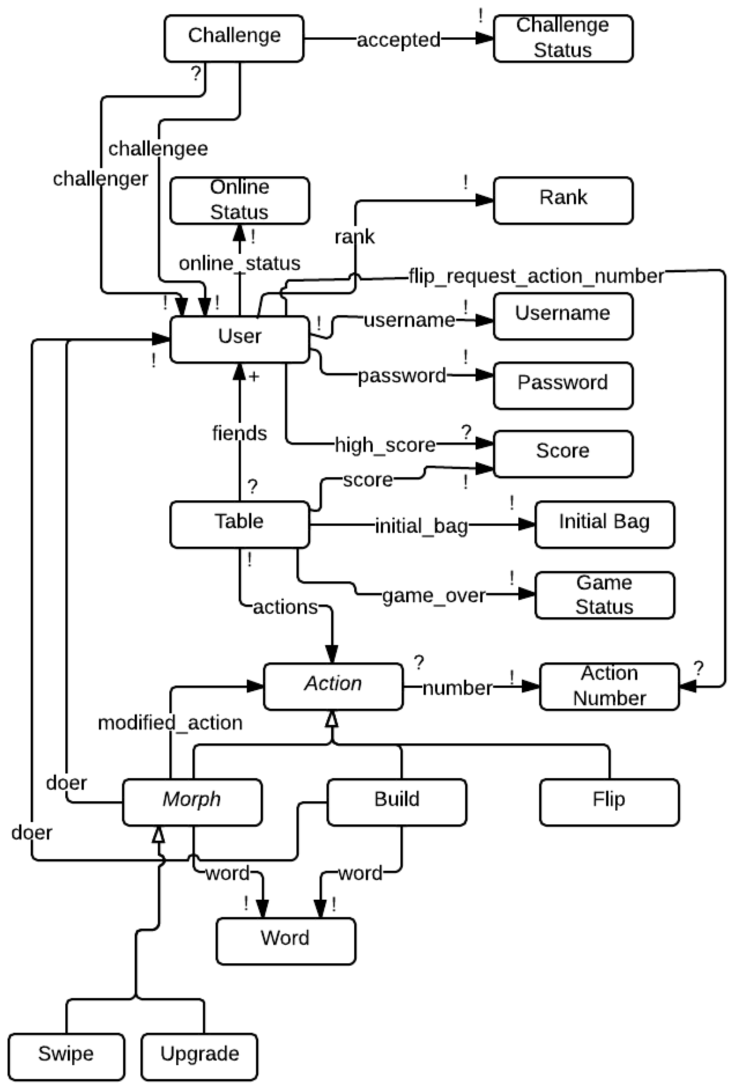
Feature Descriptions

Play Anagrams with anyone, or just by yourself. Just sign up, and you can play Anagrams in single- or multi-player mode!
Scores and leaderboard. Aim for a high score in single-player mode! Compare yourself to the world on a global leaderboard.
Match history. Keep track of your performance in both game modes.
Ranking and matchmaking. Start games in multi-player mode by challenging other players in the lobby, or by being automatically matched to a player of similar skill.
Recorded games. Relive and publicly share your past anagramming glory.

Context diagram


User
In-game actions, automatch requests, issue challenge
Display game state, lobby information
Anagrams with Fiends

Data Model



Extra constraints:
In the relation fiends: Table -> User, each Table maps to either 1 or 2 Users (there are no "empty tables").

In the relation modified_action: Morph -> Action, the target Action cannot be a Flip.

Notes:
Rankings will be handled by the Trueskill gem. From the perspective of our app, "Rank" will be a black-box object which takes in the winners and losers of games and produces numerical rankings.

Tables are never "reused" -- a Table is created for each game instance, and remains attached to that game instance even after the game finishes.

The pool is determined by the sequence of Actions which have occurred at a given Table. The letter on a given flipped tile is determined by the initial bag and the sequence of Actions.

There is no separate data field to say whether a Table is one- or two-player. We can check this by counting the number of fiends at the Table.

The concept of "Tile" is strictly a user-facing concept (drawing on the analogy to Scrabble/Bananagrams). The places in which this concept shows up are the bag, the pool, and stashes, all of which are *ordered* collections of letters. So the underlying data type is really an ordered collection of letters, i.e. a string. Attempting to incorporate "Tile" into the data model would not be helpful; it would amount to saying strings contain letters, and would not capture the notion of order.

Revisions to Data Model since preliminary design doc

- There is now a "Challenge" object type which represents a challenge from one user to another. (As mentioned above, this is how two-player tables are initiated.)
- "Turn" (which sounds more like a user-facing concept) was renamed to "Action." Actions are an internal representation of game history that the user does not directly see all at once.
- For clarity, "Swipe" and "Upgrade" have been added to the diagram as subtypes of Morph.

- A Table may now have a Score associated with it (namely, finished one-player games will have scores). The idea is that storing the score will make it easier to show high-score lists and allow users to view highest-scoring games.
- Table now has a boolean field “game_over” indicating whether the game has finished. This makes sense because the question of whether a given game is finished should be conceptually separate from the end-condition itself, and will also make it easier to query the database for finished games when implementing replay view.
- User now has a data field telling the turn number at which he has most recently requested a flip. If all Users at a given Table have the current turn number for this field, a Flip is created.

Security

Authentication: Users must log in to join a table and play. When a game starts at a table, no other users may join the table. On the other hand, results and replays from past games are publicly available, though their URLs are obscured by hashes; that way, users can easily share past games with their friends.

Privacy: Only usernames will be visible in the lobby, in a table, and in past game results and replays.

Risks:

- Players may cheat by using a computer program to identify new words to build, upgrade, or swipe.
- In multi-player mode, players may conspire to artificially change their rank by purposely losing to one another.
- Players may attempt to cheat by sending illegal moves to the server.
- Players may try to masquerade as another user with a higher ranking

Threat model:

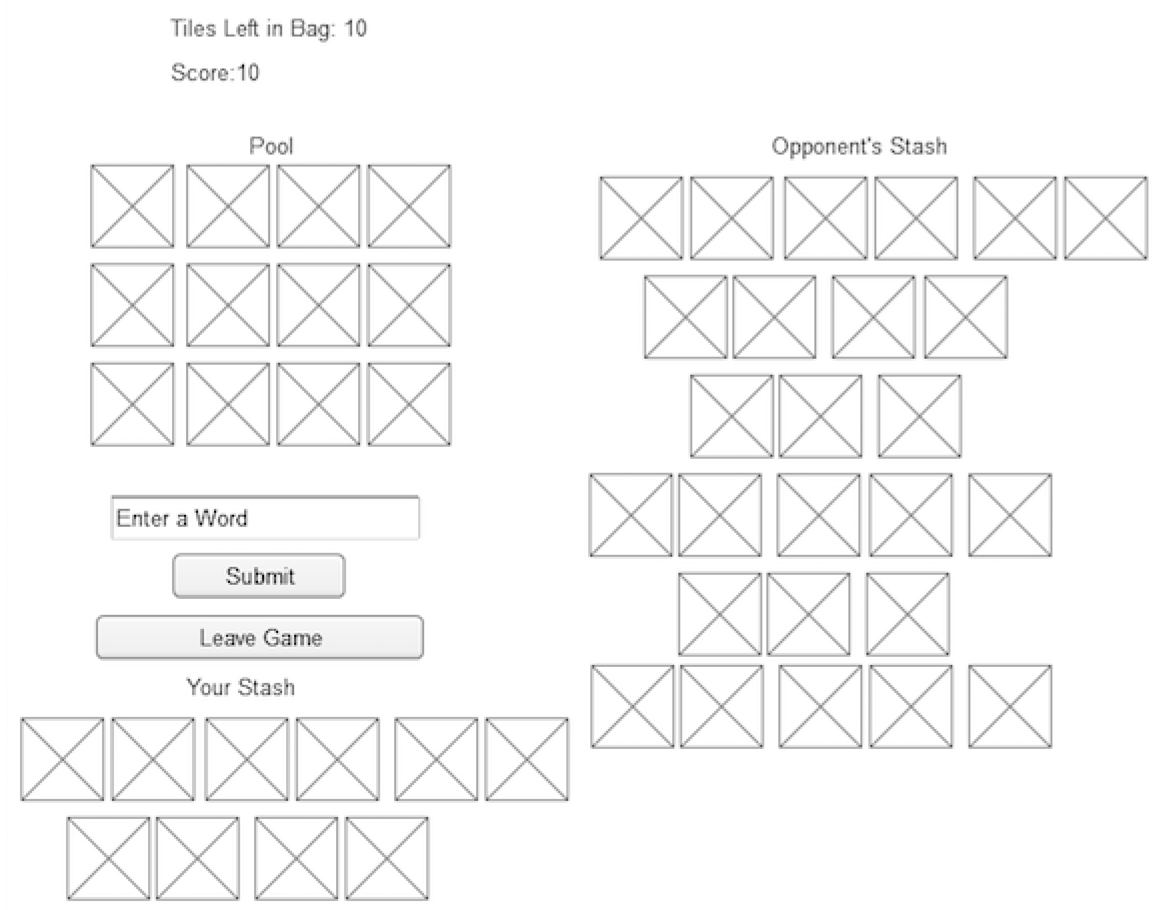
- We assume that users of our application are interested in playing Anagrams. There is nothing to gain from using the site for any other purpose, except:
  - To steal passwords
  - To feign reputation (by inflating their ranking or deflating another player's ranking)

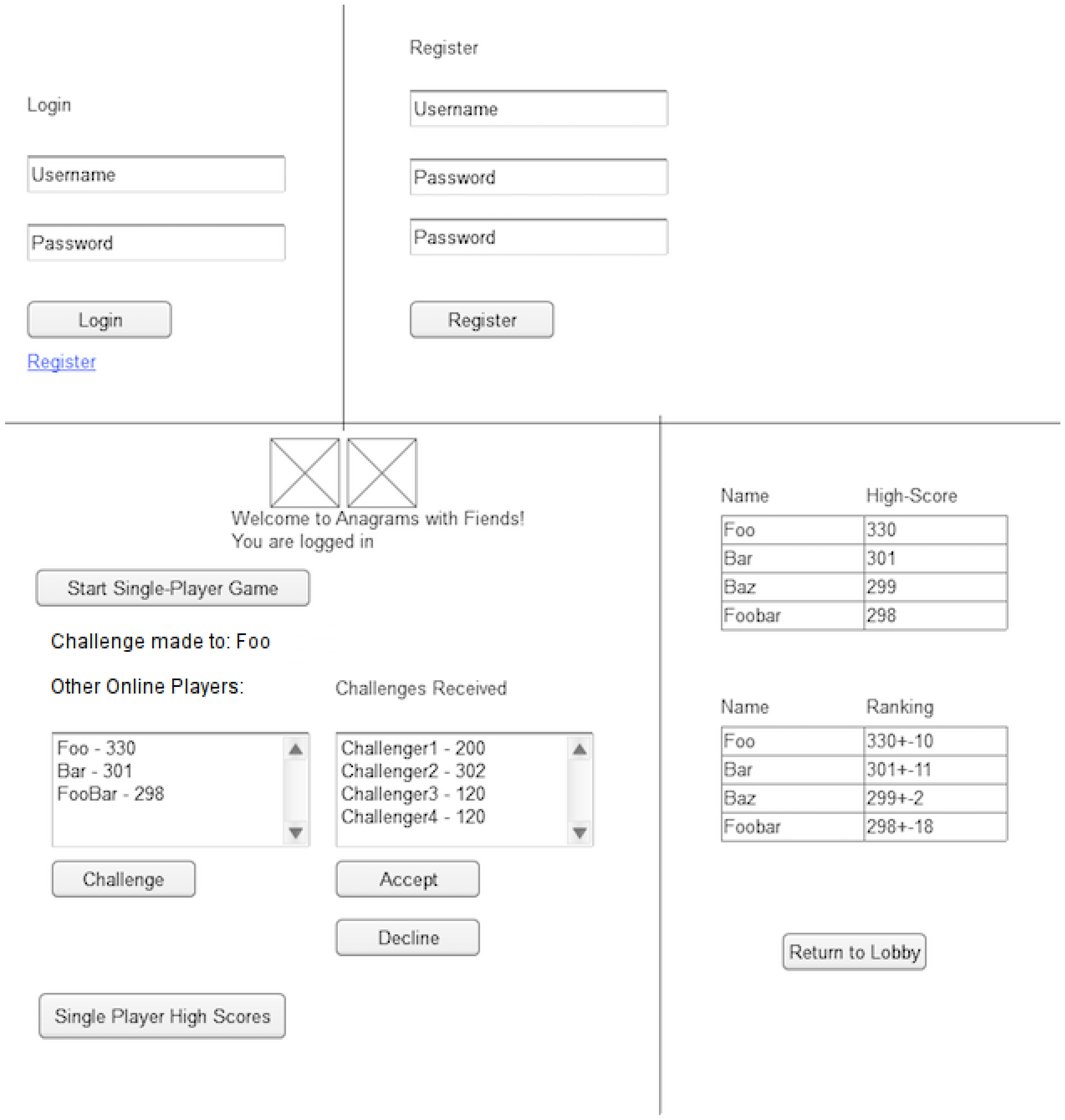
Mitigations:

- We do not entrust clients with the handling of rules. There will be legality checks on both client side and server side.
- We use the standard "technical" mitigations, e.g. Rails's has_secure_password to store passwords safely in case of database compromise, and signed cookies to prevent session spoofing.
- We will use an existing ranking system (Trueskill) which has stood the test of time. We cannot completely eliminate the possibility of collusion between players to manipulate rankings, but a good ranking system will minimize it.
- **(Added in revision)** In order to prevent spamming of game invites, a user may only have one outgoing challenge at a time (though there may be multiple incoming challenges from other users).
- **(Added in revision)** More on technical mitigations:
  - CSRF: protect_from_forgery (with: :exception)
  - XSS: only consider whitelisted params, escape HTML/Erb/javascript.

Wireframes

(updated to reflect design revisions)



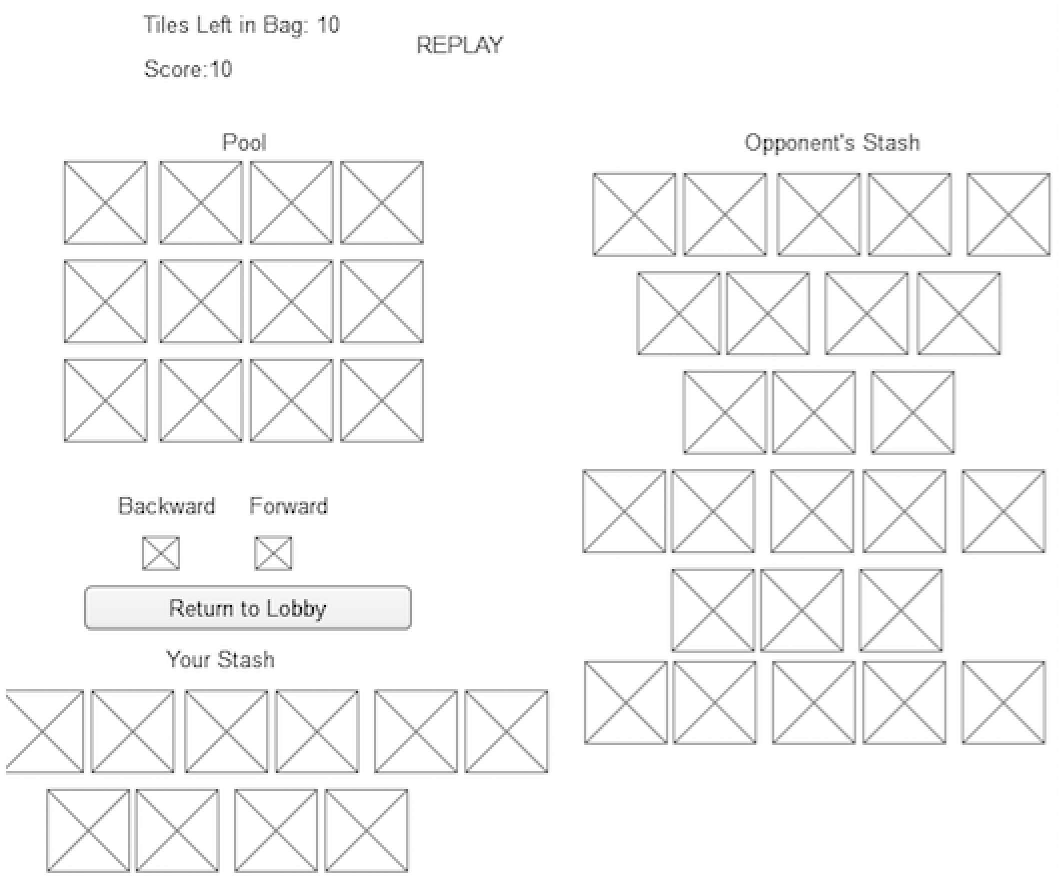


Foo - 330
Bar - 301
FooBar - 298

Challenger1 - 200
Challenger2 - 302
Challenger3 - 120
Challenger4 - 120

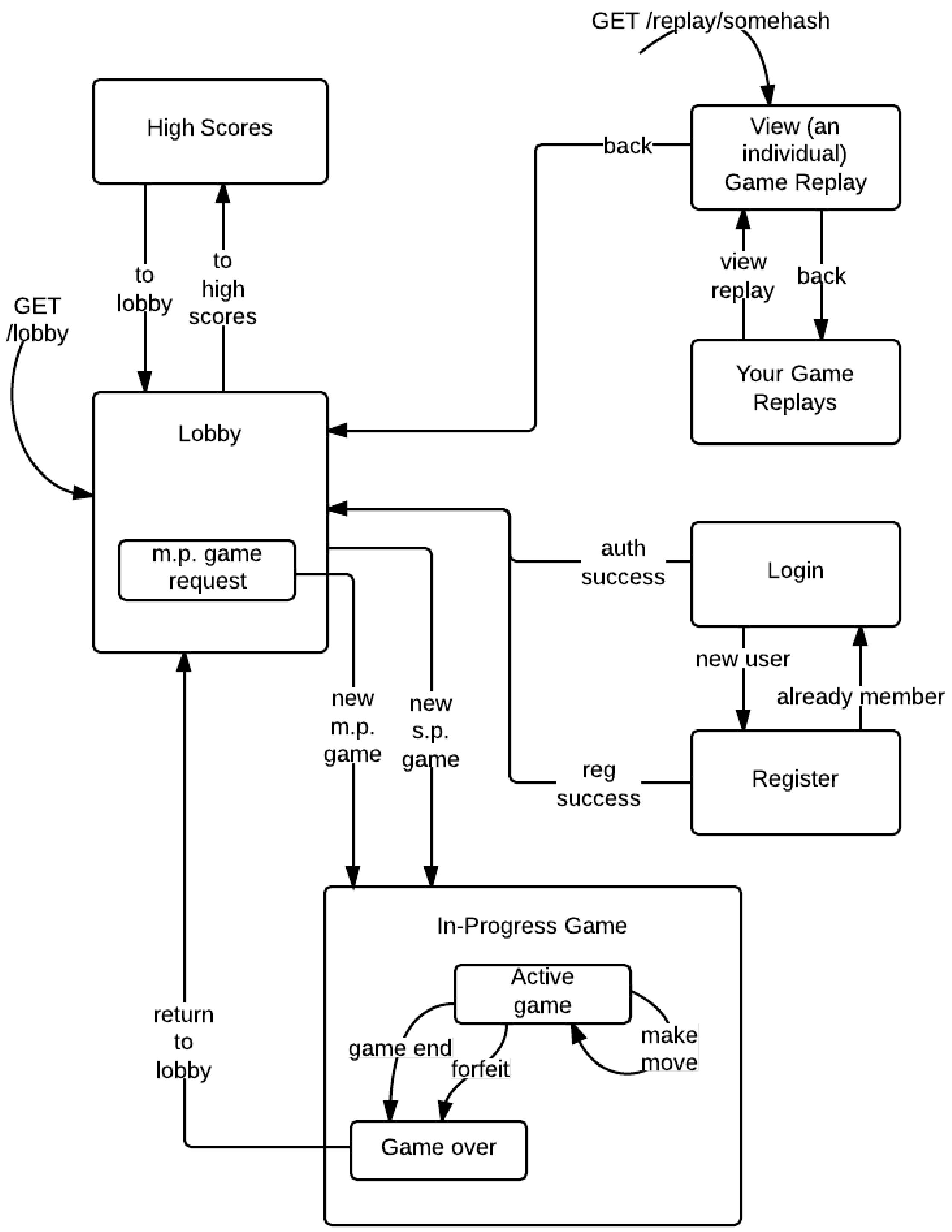
Name	High-Score
Foo	330
Bar	301
Baz	299
Foobar	298

Name	Ranking
Foo	330+-10
Bar	301+-11
Baz	299+-2
Foobar	298+-18


Tiles Left in Bag: 10
Score:10
REPLAY
Pool
Opponent's Stash
Backward
Forward
Return to Lobby
Your Stash

State Machine Diagram

(updated to reflect design revisions



Design challenges

1. **Race conditions.** If Alice tries to make a move, the app will check the validity of her move, and will accept the move if valid. However, during this check, another player (Bob) may have already moved, and such a move may render Alice's move illegal! So we will need to prevent the occurrence of two moves at the same time.
    a. We could try to arrange for some sort of semaphore-based solution. (Not sure how that would work with Rails, though.)
    b. We could try to arrange for each game to run in a single thread, so that check/accept cycles are done in sequence. This would be difficult though, as multiple web sockets will be open simultaneously.
    c. *Decision taken*: Each action has a "turn number." When Alice tries to make her move, the Action object created has a action number associated with it (say n), and the database will enforce that each Action within a Table has a unique action number. Thus, if Bob has already moved, the action number n will be taken when we try to save Alice's move to database. The nice thing about this solution is that we can then see whether Alice's move would be legal as action number n+1, and accept if so, giving the appearance of essentially "simultaneous" moves if such simultaneity is possible.
    Revision: The only type of action likely to cause a player to suddenly decide not to take the action he was previously planning on, is flip requests. That is, finding out that the pool has changed may cause him to see new words. So, only flip requests are tied to a specific turn number (and therefore expire if new actions have since occurred). For other types of actions, the turn number is set automatically when the action is created (so the action will automatically be created despite any moves which have occurred since the user sent it, as long as it still remains legal). Even for non-Flip actions, turn number is useful in that it enforces a canonical ordering of turns on the database level.
2. **Availability of past games and replays.** Who can view results and replays of past games?
    a. Only a user that participated in the game. This would require the user to be logged in, and would prevent players from sharing past games with others. However, this allows the user to hide potentially embarrassing replays.
    b. Only a logged-in user. This allows sharing past games, and ensures that only someone who actually plays the game can see replays. However, forcing a user to log in may be somewhat inconvenient if the user has not yet registered an account.
    c. *Decision taken*: Anyone who has a link to the replay or result of the past game. This makes sharing past games trivial. To try to maintain some privacy for the participants in the game, we can hash the id of the past game, so that one can only view games that he has a link to.
3. **Simultaneous games.** Can a user participate in multiple games simultaneously?

    a. Allowing a user to participate in multiple games simultaneously doesn't make much sense, because the game fundamentally depends on quick real-time reactions to tiles being added to the pool.
    b. *Decision taken*: A user may only play a single game at a time. If a user disconnects or leaves the page while in a game, logging back on will redirect to the game (if it has not yet ended.)

4. **Added in revision: Challenges and lobby.** How do multi-player games get started?
    a. *Decision taken:* One player may challenge another player to a game from the lobby. If she does so, then the latter player receives a notification that he has been challenged and can either accept or decline the challenge. If he accepts, then a game starts; if not, the challenge disappears. This struck us as the simplest method to implement, and the one which gives users the most control over whom they get to play. This decision may lead to issues if the maximum number of concurrent users grows dramatically, but should be acceptable up to at least several dozen (as evidenced by the success of Dominion on Isotropic).
    b. A player may create a “table”, which is displayed in the lobby as having an empty seat. Another player can click a button to sit down at the table, after which the first player can start the game. This paradigm is potentially more extensible, as we can later implement things like private tables and invitations, but is also more asymmetric and potentially more difficult to implement.
5. **Timing out of a game.** A game should certainly end after the bag is empty and sufficiently much time passes without a move being made (i.e. no one builds, swipes, or upgrades a word for a sufficient period of time). What if a fiend disconnects or quits when there are tiles remaining in the bag?
    a. Single-player:
        i. The game times out and ends automatically after a fixed length of time without a move, so that users can't continue single-player games after an extended period away.
        ii. *Decision taken*: The game "pauses" until the user returns. This allows users to leave a single-player and continue it later.
    b. Multi-player:
        i. If a user rejoins the table within a short period of time, he may rejoin. If a player is away for an extended period of time, he is dropped from the game and cannot rejoin. While away or dropped, a player's words may still be swiped. The dropped player is still scored at the end of the game, and is ranked accordingly.
        ii. Players are still dropped, as above, but receive a score of 0.
        iii. *Decision taken*: (**Revised**) The game continues normally (regardless of the length of time a user is away), allowing disconnected players to rejoin the table if they reconnect and the game is still running. A fiend may also choose to forfeit the game; if so, the remaining player wins.
6. **Single-player scoring.** How do we compute a fiend's score in a single-player game?

   a. A fiend's score could be the number of words in his stash. In the physical, multi-player version of Anagrams, this is the standard. However, in single-player, this would not provide much differentiation between scores, and would reward the uninteresting strategy of forming as many short words as possible.
   b. We could use a Scrabble-like scoring system, where tiles are assigned point values; when a user builds or upgrades a word, his score increases by the sum of the point values of the letters in the word. This would encourage forming more uncommon words.
   c. We could use a Scrabble-like scoring system with bonuses for longer words; this is similar to the previous scoring method, but encourages forming long words.
   d. We could use a variation of one of the previous two methods, where the score is not incremented every time a user builds or upgrades a word, but is computed at the end.
   e. *Decision taken:* (**Revised**) At the end of the game, each word is worth a number of points which scales quadratically with the length of the word. In addition, a player loses points if he flips tiles when there are a large number of tiles in the pool. This rewards players who are able to think of words with only a limited number of tiles in the pool, and the quadratic scaling is similar to other word games like Boggle.

7. **Multi-player victory condition.** How do we determine the winner of a multi-player game?
   a. *Decision taken:* The winner is the player with the most words in his stash at the end of the game. This is standard in the physical version of Anagrams.
   b. Use the same scoring system as in single-player, where swiping and upgrading words are treated identically. This would be intuitive, but perhaps swiping should be rewarded more greatly.
   c. Use a modified version of the single-player scoring system, but giving a bonus for swiping words. This could introduce a new element of strategy, where fiends prioritize taking other fiends' words.
8. **Ranking.** How is a user's multiplayer ranking determined? There are a number of existing ranking algorithms, such as Elo and TrueSkill, with varying degrees of complexity and flexibility. Many also have implementations as Ruby gems.
   a. *Decision taken:* We'll use the TrueSkill rating system, which has a notion of the "variance" of a player's rating. It's more suited to games which are inherently random.
   b. We could use Elo rating instead. Elo is better suited for games which are non-random, like chess.