ACQUIRE ONLINE
A Web Based Version of the Board Game Acquire

Marin Abernethy ’14
Inter. Computing & Economics
Advisor: John Ridgway
Professor of Computer Science

Abstract
Acquire Online cohesively incorporates the economic strategy of the classic board game, Acquire. In a web based multiplayer version of the game. As an authenticated user, one can enjoy as many games as desired. The objective of the game is to earn the most money by developing and merging hotel chains. This economic simulation game is consistently rated one of the best economics board games, however, people these days turn to their technological devices for entertainment rather than sitting around a game board. This projects aims to convert the fun of the board game to the Internet. This application is built in Ruby on Rails, a server-side web framework, alongside front end web languages such as HAML (HTML Abstracted Markup language), and SASS (CSS Compiler). Additionally, Ember.js, a client-side JavaScript framework helps to make the game dynamic, with its Handlebars templates that update automatically when the underlying data changes. The design is simple so that gameplay is easy and enjoyable.

1. Project Objectives
1.1. Web application people want to use
1.2. Replicate the gameplay and rules of Acquire (the board game) as close as possible
1.3. Simulate the board game experience and provide as much information as one would get from playing face-to-face
1.4. Design interface with intuitive features to provide enjoyment as simply and elegantly as possible
1.5. Efficient code for fast response time

2. Game Description
Objective: players earn money by developing and merging hotel chains
Turn:
1. Placing a tile
   - Orphan(7I), new chain(8I), chain extension(10H), merger(8G)
2. Buying stock
   - 0-3 shares in hotels that are on the board (founded)
   - share price depends on size of chain
3. Drawing replacement tile
Merger:
- Larger chain acquires smaller chain
- All players decide whether to hold, sell or trade their shares in acquire chain
Win:
- When all hotels have size >= 11 OR one hotel has size >= 41
- Liquidate all player’s shares in hotels
- The winner is the player with the most money

3. Architecture
- Ruby on Rails
  - Developed by David Heinemeier Hansson (2003)
  - Server-side web framework
- Model-View-Controller (MVC):
  - Model: Ruby classes. They talk to database, store and validate data, and perform business logic.
  - View: Describes presentation of application, it is what the user sees.
  - Controller: Acts as link between models and views to construct HTTP or HTTPS responses to send to the browser
- Ember
  - Developed by Yehuda Katz and Tom Dale (2011)
  - Client-side MVC framework (JavaScript)
  - Makes sending and receiving appropriate data to and from the server straightforward
  - Uses Handlebars templates that automatically update when underlying Ember model changes

4. Design
- Game Center
  - Notifications for when player’s turn in a game or when game ends (deletable, scrollable)
- List of other players online (scrollable)
- List of current user’s games and option to continue game (scrollable)
- Navigation bar with access to rules, home page, game center, etc.
- Start new game by naming game and choosing opponents from list

5. Implementation
- Database
  - A challenging portion of the project was setting up the structure and organization of the Rails database, and the relationship between the many models. For example, a user can have multiple games and each game can have multiple users (or players) associated with it. This relationship is accomplished with a has_many :through association. The join-table in the database in this many-to-many relationship between the Game model and User model is called GamePlayer.
- Many-to-Many Relationships:
  - Game and User
  - Game and Tile
  - Game and Stockcard
  - Tile and GamePlayer
  - Game and Hotel
  - GamePlayer and Notifications

- Game
  - id integer
  - name string
  - up_next string
- GamePlayer
  - id integer
  - game_id integer
- User
  - id integer
  - username string
  - password string
- Ember
  - Ember detects click event
    - Responds by sending asynchronous request to server
- Rails controller updates game state
  - Server processes request and dispatches request to games controller, which updates game state and sends response to browser (as JSON)
- Ember success handler updates models
  - Fulfillment handler updates game state in Ember models
- Handlebars templates auto updates
  - Once underlying Ember models change, information updated in view

6. Future Work
Always room for improvement
- Build more animated game board (e.g. draggable tiles)
- Chat functionality to mimic conversation around real game board
- Personal statistics in game center (e.g. games won, games lost)
- AI component, so players have option to have computer-generated opponent
- Refactor code to increase efficiency and speed
- General front-end work to make more aesthetically pleasing

Marin Abernethy ’14
Inter. Computing & Economics
Advisor: John Ridgway
Professor of Computer Science

Trinity College Hartford Connecticut

© 2013-2014 Marin Abernethy

Name: Marin Abernethy
Major: BS Econ
Advisor: John Ridgway
Class: ’14

Project: ACQUIRE ONLINE
A Web Based Version of the Board Game Acquire

Abstract
Acquire Online cohesively incorporates the economic strategy of the classic board game, Acquire. In a web based multiplayer version of the game. As an authenticated user, one can enjoy as many games as desired. The objective of the game is to earn the most money by developing and merging hotel chains. This economic simulation game is consistently rated one of the best economics board games, however, people these days turn to their technological devices for entertainment rather than sitting around a game board. This projects aims to convert the fun of the board game to the Internet. This application is built in Ruby on Rails, a server-side web framework, alongside front end web languages such as HAML (HTML Abstracted Markup language), and SASS (CSS Compiler). Additionally, Ember.js, a client-side JavaScript framework helps to make the game dynamic, with its Handlebars templates that update automatically when the underlying data changes. The design is simple so that gameplay is easy and enjoyable.

1. Project Objectives
1.1. Web application people want to use
1.2. Replicate the gameplay and rules of Acquire (the board game) as close as possible
1.3. Simulate the board game experience and provide as much information as one would get from playing face-to-face
1.4. Design interface with intuitive features to provide enjoyment as simply and elegantly as possible
1.5. Efficient code for fast response time

2. Game Description
Objective: players earn money by developing and merging hotel chains
Turn:
1. Placing a tile
   - Orphan(7I), new chain(8I), chain extension(10H), merger(8G)
2. Buying stock
   - 0-3 shares in hotels that are on the board (founded)
   - share price depends on size of chain
3. Drawing replacement tile
Merger:
- Larger chain acquires smaller chain
- All players decide whether to hold, sell or trade their shares in acquire chain
Win:
- When all hotels have size >= 11 OR one hotel has size >= 41
- Liquidate all player’s shares in hotels
- The winner is the player with the most money

3. Architecture
- Ruby on Rails
  - Developed by David Heinemeier Hansson (2003)
  - Server-side web framework
- Model-View-Controller (MVC):
  - Model: Ruby classes. They talk to database, store and validate data, and perform business logic.
  - View: Describes presentation of application, it is what the user sees.
  - Controller: Acts as link between models and views to construct HTTP or HTTPS responses to send to the browser
- Ember
  - Developed by Yehuda Katz and Tom Dale (2011)
  - Client-side MVC framework (JavaScript)
  - Makes sending and receiving appropriate data to and from the server straightforward
  - Uses Handlebars templates that automatically update when underlying Ember model changes

4. Design
- Game Center
  - Notifications for when player’s turn in a game or when game ends (deletable, scrollable)
- List of other players online (scrollable)
- List of current user’s games and option to continue game (scrollable)
- Navigation bar with access to rules, home page, game center, etc.
- Start new game by naming game and choosing opponents from list

5. Implementation
- Database
  - A challenging portion of the project was setting up the structure and organization of the Rails database, and the relationship between the many models. For example, a user can have multiple games and each game can have multiple users (or players) associated with it. This relationship is accomplished with a has_many :through association. The join-table in the database in this many-to-many relationship between the Game model and User model is called GamePlayer.
- Many-to-Many Relationships:
  - Game and User
  - Game and Tile
  - Game and Stockcard
  - Tile and GamePlayer
  - Game and Hotel
  - GamePlayer and Notifications

- Game
  - id integer
  - name string
  - up_next string
- GamePlayer
  - id integer
  - game_id integer
- User
  - id integer
  - username string
  - password string
- Ember
  - Ember detects click event
    - Responds by sending asynchronous request to server
- Rails controller updates game state
  - Server processes request and dispatches request to games controller, which updates game state and sends response to browser (as JSON)
- Ember success handler updates models
  - Fulfillment handler updates game state in Ember models
- Handlebars templates auto updates
  - Once underlying Ember models change, information updated in view

6. Future Work
Always room for improvement
- Build more animated game board (e.g. draggable tiles)
- Chat functionality to mimic conversation around real game board
- Personal statistics in game center (e.g. games won, games lost)
- AI component, so players have option to have computer-generated opponent
- Refactor code to increase efficiency and speed
- General front-end work to make more aesthetically pleasing