Model

Field Array: 20 x 20 grid

-20pc army for red

-20pc army for blue

Red Tray

Blue Tray

Text Message List

-Identity red or blue

Methods

Initialize

-fill trays with respective pieces

-empty playing field

Set Pieces

-Allow a piece to be moved from a tray and placed on the field

Move

-Check if destination square is within bounds and not occupied by another piece of own side

-Check if destination square is occupied by enemy

-Check if destination is within the allowed number of spaces

-Log identity of each piece

Attack

-Compare pieces

-Determine attack winner based on rules

-Remove losing piece to respective tray

-Determine if move results in a win

Insert Message

Rules

-Spy beats Admiral

-Minesweeper beats bomb

-Any piece captures a flag

Otherwise

-Rank prevails

Other features

-If possible, allow multiple games simultaneously

-Code is clearly separated and identified so that rules can be modified for FUN

-

Ruby Hash for field

{

     ‘game\_id’ => 1

        {

          ‘pieces’ => [

          {

            ‘col’ => 3

            ‘row’ => 2

           ’team’ => ‘Federation'

           ‘rank’ => '5’

           }

     }

}

{

  "currentPage":1,

  "numberOfPages":951,

  "totalResults":47523,

  "data":[

    {

      "id":"cBLTUw",

      "name":"\"18\" Imperial IPA 2",

      "nameDisplay":"\"18\" Imperial IPA 2",

      "description":"Hop Heads this one's for you!  Checking in with 143 IBU's this ale punches you in the mouth with extreme bitterness then rounds out with toffee flavors and finishes with a citrus aroma.  Made with tons of US 2 Row Barley to get this to ABV 11.1%.",

      "abv":"11.1",

Logic for computer to play

Level: Basic

Follow the rules of taking turns, legal moves and melee,

Move a random piece forward

Level 2: Some Strategy

Move the closest piece, favor the direction of the most opponents.

Level 3: Assign scores to possible moves.

High value is an attack where the opponent has already been revealed.

Low-move toward distant piece

High-move toward probable flag when guarded only by pieces that haven't moved

What questions do I ask about each piece?

How far am I from an attack?

How important is my survival?

How important is my current position?

Do I have a good target in sight?

What enemy pieces are left?

What pieces do I have left?

Easily measured:

Dist to nearest enemy

Proximity to own flag

Process:

In the array of pieces, have columns for original\_location, has\_been\_revealed, spaces\_moved.

Tables are:

Gameboard: col\_1, col\_2, col\_3, ...

One record for each row, contents are token\_id

Token: color, rank, img, spaces\_can\_move, has\_been\_revealed, spaces\_moved\_this\_game, cum\_adj\_open\_spaces

Blue tray col\_1

Red tray col\_2

Red\_discard

Blue\_discard

Also track turn\_count

So there needs to be a method that iterates over the pieces.

If row is not zero (indicating discarded)

Iterate row-1 to row

In plain English, strategy of playing Warp Attack:

Possible motives for moving my next piece.

1. General advance: make an available move to get a piece closer to the enemy. (semi-random)
2. Protect the flag: move closer to the flag because enemy is near and protection is getting thin (requires observation, quantifying flag exposure & scoring relative value of moving closer)
3. Attack a known piece
4. Attack a piece in hopes of defeating it
5. Attack a piece to identify it
6. Move out the way of a piece I need: For example, two spaces away, a Rank 3 has taken my Rank 5 and I want to get my Rank 2 in position to attack
7. Retreat from a known or suspected superior (only need to know proximity, but it’s somewhat complex to know if the retreat is of value. A higher rank compatriot has to be in position to attack)
8. Move toward a known/suspected piece
9. Move toward the flag

Generalized motivations for piece movement

1.     Attack/Pursuit/Scout

2.     Evade/Defend

How about this approach?

My models include the things that I think would be useful to an artificial player. I somehow modularize the logic for programmitically choosing a move. The minimum requirements for an AI player is that it

Follow the rules of taking turns, legal moves and melee,

By playing a game, one could observe how useful additional logic components would be. A simple one is: favor a move that is an attack. What would be even more interesting than having a supersmart computer player, I think is devising a straightforward method for adding additional logic.

So the question is, is there a way, that I could just insert another ruby file such that it’s detected and run after every player move and it’s return value is added to the score of the move.

Object of moves is generated

piece, to\_row, to\_col, score

The project description:

A 40x40 grid board game based on Stratego, but with icons of starpships. It’s a two-player game with each player interacting with a different web page/Backbone.js view.

Logic Modules

NewGame

-Respond to selecting new game

Choose Red/Blue

-Respond to selecting new game

Choose Cancel

-Respond to Cancel

Choose Reset to Tray

-Respond to Reset to Tray

MovePiece

-Switch: Legal move? Possible Move? Attack? Win?

-Result is one or two pieces in new position(s). Updated status

User interface:

Firebase provides for login with facebook, twitter, google, github. This may be a bit more than neccesary for the purpose of the game. However, sessions is a much better way to track multiple games than url.

So, I need to see an example and how I would plug into that.

Display fun:

UI modes:

Sign in (user must have a Google/Facebook/GitHub/Twitter)

Prefenece for an empty field with a dialog box over it.

Buttons for Pick a side (first-to-choose gets choice) or sign out

Preparing field

Opponent’s field is dimmed (CSS on opponent’s tokens)

My available spaces have an inner border/shadow/something.

Buttons for resetting pieces to tray, signing out.

Some sort of indication that you can’t put that token there. (I think just a snap back is okay.

All my pieces set

Button to indicate all ready

Game in Progress

Pieces lost display

Button for sign-out

Win detected

Display to announce winner

Everything is object and event-based

What do I need?

Need to study tic tac toe example.

Question: The example used seems to be angular and firebase. Firebase can be on the server or on the client. I think that the authentication feature should drive the choice of the rest of the technology. I want to use Firebase for authentication and for hosting.