CS3270: Adv Web Programming **Due: Mon May 05**

Final Project **10:30AM**

**RE\_TimeLapse Project:**

**Dynamic Reconstructions** *(i.e. Time-Lapse Snapshots)* **of Completed 4-player RE Games**

For lo these many decades, serious students of Reef Encounter Theory have yearned for the functionality that would allow them to redisplay completed RE games on a turn-by-turn basis for their personal study, in order to enhance their mad RE skillz. And finally, in you and your newly-acquired PHP and DB abilities, they have found the long-awaited solution to all their problems!

The general solution described below may require two or three separate PHP pages.

In order to provide the clamoring horde with a satisfactory solution, you should accomplish the following:

The General Idea:

Your company wants a clickable browser app that allows the user to select a completed RE 4-Player game from a list. (**Part A**)

This will take the user to a page that displays a snapshot of the *State* of the game at the end of Player 1's Turn 1.

Clicking a link (or a button) will update the snapshot to the end of Player 2's turn. (**Part B**)

Until the end of the game is reached, when it will show the total Points and final game-state (**Part C**)

[ Sourced from: [**http://www.spielbyweb.com/games.php?list=cmp**](http://www.spielbyweb.com/games.php?list=cmp) ]

**A. Research downloading a live HTML page's source from the internet. Then autogenerate a table in PHP with the web-scraped contents.**

Find a PHP command that will download the HTML of a page-address you provide to it. You should then be able to process that string result line by line (e.g. PHP string methods) to extract the information you're looking for from among the embedded hypertext anchor tags.

Once you have found a good way to do this, write a PHP page that you can run to generate up to *THRESH* (i.e. threshold) completed RE game links (and other relevant details, such as ***gameID***) to create an HTML table of clickable entries in your browser, from the page: <http://www.spielbyweb.com/games.php?list=cmp>. Note: you may need to download more than one page's worth of links to accumulate at least THRESH table-entries.

Sample Table description:

ID : integer // row-Number (opt)

GameID : integer // SBW's numeric ID for the game

Name : varchar // Title of RE game

Players: integer // Number of Players

Names: varchar // handles of Players

However: clicking these links should not take you to SBW's official completed game-page for game #***gameID***, -- it should go to a *new* page you write (similar to REhack.php -- but perhaps named: *RElapse.php?gameID=****gameID***)

When a game-name on this page is clicked, it should load the ***initial*** view of the PHP page described in **part B**, which should display that game's *game-state* at the end of the first player's turn 1.

The RE\_Timelapse Project will proceed as described in **Part B**.

[ Sourced from: [**http://www.spielbyweb.com/gamelog.php?games\_id=99147**](http://www.spielbyweb.com/gamelog.php?games_id=99147) ]

**B. Page-by-page time-lapse of the selected RE game**

This page (**RElapse.php**) should make use of the ***gameID*** (and possibly the ***$states***-array index, for example) as ***$\_GET***-parameters in the address bar to track progress through the replay: for example *turn* ***T*** and *player* ***P*** .

The base page should be a game-page initialized for 4 players. (To acquire this, you can copy the source of a page from a live 4-player game in its early turns, and remove all tiles/pieces on each reef. )

[[**Here is a sample base-page**](https://cs.bemidjistate.edu/fneville/RE/RElapse.php) you may base your **RElapse.php** page upon]

The basic idea is this: before this RElapse page executes, it (or some other page) should process the source HTML of the GameLog page of the requested RE gameID ([Sample GameLog page](http://www.spielbyweb.com/gamelog.php?games_id=99147)). The entirety of this source may be processed into an array of state-objects, which RElapse.php can just step through.

Note: you need only display the game-states of the *end* of each player's turn: you are not required to display the result of *each* individual action/subaction taken during the turn.

*This section will be by far the most time-consuming part of this project -- be sure to set aside enough time for this step!!!*

This page should also have a link (or a button) for the user to click in order to reload the page to redisplay the game-state at the end of the *following* player's turn (*e.g. T=1, P=2*). ***You should be able to proceed in this manner all the way to the last player's last move*** (i.e. the end of the game!), and reach the same final board-state as those reached by the links on the site's "Completed Games" pages.

Optionally, you might implement a second link to take one step *back* to the *preceding* state.

More Optionally, you might implement a separate link/button to advance the game-state by an entire turn (*e.g. from: T=2, P=3 to: T=3, P=3*).

Below the displayed board-state on your page, the current player ***P***'s just-completed actions should be displayed in tabular format, similar to their display in the GameLog page.

Implementation Hint: much of Part C above relies on *Game State* being maintained as the GameLog page is traversed. You already have four **$rf[]** arrays in a **$bord** array maintaining the state of the board (which you used Extract and Re-order action-strings from log into $acts array:

• Extract Final Round actions

• Extract Numbered Round actions

o IF in\_Round1: create list of Player-names in player-order

• Extract Initial Round actions

Process $acts array:

ForEach $act-string:

• Get Player-name

• Derive Player-Index

• IF is\_actionX:

Do actionX stuff

ELIF is\_actionY:

ELIF

ELIF …

• Add current state to $states

Save $states to $\_SESSION array

Redirect browser to Relapse rendering page

in HW8 specifically to render the state of the board). You may want to implement in addition: 1) a 10-length Dominance array; 2) a 4-length PlayerName array; 3) a 4-length ConsumedPolyps array; 4) a 4-length array of EatenPolyps, 5) a 4-length array of "Larva" Cubes; 6) a 4-length Board[] array (and if you implement the first E.C. below: 6e) a 4-length array of PolypTilesInHand). All these could be saved into an associative array: **$state[]**.

As each line in the GameLog is processed, you would normally want to modify the appropriate portion of this set of GameState variables, before saving the final **$state** variable back into the **$\_SESSION** array. Then, you would render from it the page-display in the browser much as you did in HW8 (when all you had to worry about was **$bord**!)

But -- there's a small problem: the GameLog presents events in *reverse* order -- most-recent event first! You, of course, might prefer to step through the game from *least*-recent to *most*-recent. How to do this? My suggestion: pre-process the page by loading each HTML-block representing a player-turn into an (temporary) array of strings (e.g. **$plyrTurns[]**)) -- and then proceed to pre-process this *entire* array into individual game-state instances, saved into an array **$states[]**. I'd save this array into the **$\_SESSION** array too -- and use it for successive page-reloads.

You know what suddenly strikes me would be *really* convenient here, is a DB with a ***States***table where each row consists of ***T***, ***P*** and ***State*** value!!! But this is optional. You can weigh down the $\_SESSION variable with a ghastly $states[] array if you prefer…!

Include a writeup of the necessary PHP pages in your ***hw.htm*** report.

Some Example (Project Savepoint) links shown in class:

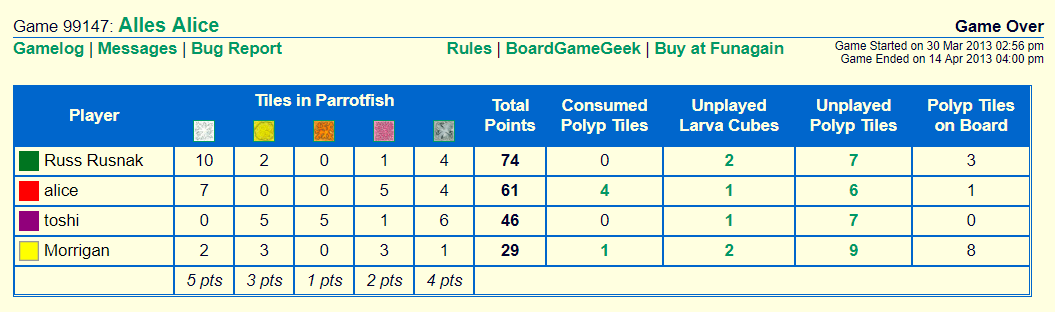
* Extract/Print all action-strings from gamelog: <https://cs.bemidjistate.edu/fneville/RE/RElapse1.php?games_id=99147>
* Save only useful action-links, display growing actions-array, and initialize ***$state*** object at page-bottom: <https://cs.bemidjistate.edu/fneville/RE/RElapse2.php?games_id=99147>
* Implement actions 0: CHOOSE\_CUBES and 4: INTRODUCE\_SHRIMP: <https://cs.bemidjistate.edu/fneville/RE/RElapse2b.php?games_id=99147>
* Implement actions 10: CHOOSE CUBE/TILES, 8: EXCHANGE CUBE\_FOR\_TILE, and 5: MOVE SHRIMP : <https://cs.bemidjistate.edu/fneville/RE/RElapse2c.php?games_id=99147>
* Implement actions 2,3: PLACE TILES, and: INTERRUPT PLACEMENT W/ SHRIMP-MOVE : <https://cs.bemidjistate.edu/fneville/RE/RElapse2d.php?games_id=99147>
* Implement action 1: EAT SHRIMP, CORAL (see new eatPolyps() fct in Free Code below): <https://cs.bemidjistate.edu/fneville/RE/RElapse2e.php?games_id=99147>
* Final PRElapse page does all the processing above, but when finished only prints a link to the RElapse starting-point (state 0: all players have selected cubes; all crimson text and reef-tiles are retrieved from the $states[0] object). The link below the reef-boards in RElapse.php renders subsequent snapshots by processing subsequent state-objs in $states:

<https://cs.bemidjistate.edu/fneville/RE/PRElapse.php?games_id=99147>

**C:** **Final Result Page**

Finally, the final page in this timelapse series should display essentially the same final page and score-table that the SpielByWeb site displays for the completed game with that gameID -- that is, a summary of eaten polyps, points and cubes/tiles remaining in tie-break order.

*Example:*



But, your final state-numbers may not perfectly match the numbers shown on the official webpage, so **here is what I'd like you to do**: use your final ***State*** object to print your own resource values *(i.e. cube #s, polyp #s, etc)* out in parentheses **()** immediately next\_to/after each of the values in the above image. In a perfect world, your parenthesized values should be exactly the same but… a game is a complicated program! Lots of room for a mistake or ten to occur in those 500 lines of code!!!

Show me how close you came by this project's due-date! *(Note: inaccurate values will not necessarily sink your grade, just as perfect values necessarily deliver a perfect grade -- it all depends on your explanations in your writeup/presentation! Make an effort to find the code where you went wrong, and/or explain how/what you fixed to make things better -- that's what I'm most interested to read in your write-up!)*

Note: there is some *missing* information in the GameLog that you need to know to reach the correct game-state at the end of the game! Each board starts with five hard-coded polyp-tiles at the start of the game. These are printed directly onto the physical boards, an image of which is available at: <https://www.boardgamegeek.com/image/196666/reef-encounter> (Notice that the boards in the picture are not in the same order or orientation as in your RE interface!)

Make sure these 20 tiles are taken into account when you initialize a new game-state in Part C

(As mentioned above, the GameLog doesn't record what the OpenSea slots refill with, so do not worry about that)

**Final Presentation & Writeup:**

Plan for a 10-15 minute team presentation during the scheduled Final period for this course. This presentation should be about 5-10 slides long, describing your approach to solving the solution, the code you wrote to detect/fix your errors, and your results. It may include code-snippets, and a live demonstration of your favorite or most-useful code implementations.

**Deliverables**:

Your deliverables for this project will be:

* a GoogleDocs Report: this should contain:
  + a writeup of your observations and experiences in completing this project. If this was a team effort, list major contributions by each team-member.
  + an (wireframe) illustration of the pages comprising this web-app and their relationships to each other
  + an appendix containing the code written for this project
* a link to the first PHP page in your "RElapse" webApp
* the Final Presentation document presented in class during the final exam period
  + It can be either a {MSPowerpoint saved as a PDF) file, or a GoogleSlides presentation link.
    - If your presentation is in GoogleSlides format, remember to set "Sharing" to: "Anyone with the link can view"

Provide a link to each of these from your **hw.htm**.

Some suggested Extra-Credit projects follow, but they are optional.

**D. Dynamic RE Reconstruction Extra Credit:**

Extend the Player data table to include each player's *Behind\_Your\_Screen* holdings comprised of cube/tile draws from the OpenSea (i.e. excluding the player's initial starting-hand and parrotfish-belly tile), so far as they can be determined. You can expect these to be pretty inaccurate for the first four turns or so (since the GameLog understandably does not reveal players' starting-hand of tiles), but by turns 7-8, they should be quite accurate…!

**==========================**

**E. Statistical Meta-RE Extra Credit:**

**==========================**

Experienced Reef Encounter players know that each RE game will begin with one of exactly two setup-arrangements of Dominance tiles, thereby potentially providing an initial advantage/disadvantage to certain polyp-colors in each game.

The question is: is there any evidence that any given color is permanently advantaged/disadvantaged overall due to these two possible initial setup arrangements over all four-player games, or does it balance out over time? Write a PHP program that determines how many times each of the polyp-colors were *maximally-dominant* (i.e. worth the most points ***in that game*** -- count all tie-participants) by the end of all four-player RE games saved in your database, as well as how many times each color was *minimally-dominant*.

For example, if X games of RE were played and no particular coral advantage was present, we might expect that each color ended up maximally-dominant about X/5 times, and minimally-dominant about X/5 times. Does this actually seem to be the case? Feel free to use any additional statistics knowledge you may have available in your group to help make your case.

Display and explain a table of each of these results in your group's final presentation, include this code in your report-appendix, and attach a link running this PHP program to your hw.htm page

Definitions:

***maximally-dominant*** means worth the most points (or dominant over the most *other* corals) in the just-completed game. A coral need *not* be worth 5 points to be maximally-dominant in its game! If two (or more) colors are maximally-dominant, count each one.

***minimimally-dominant*** is similar to the above, except that it is tied to *least* point-value in the just-completed game. A coral need not necessarily be worth 1 point to be minimally-dominant in its game. Again, count all ties.

**Note:** while lectures will be few and infrequent in this lab in the next few weeks, you are still expected to attend each lab-period to assist your group-mates, benefit from the help and knowledge of your peers, and allow them the opportunity to benefit from your own.

**FYI: Notes for Next Class:** below are some concepts useful to this project I recently ran across that I didn't get to earlier in the semester, placed here as a reminder to myself to present to you in the next class period -- but available to you should you run across it earlier!

* Adding modified **objects** to an array:

e.g. { $s = clone $state; array\_push($states, $s); }

When we want to instantiate a PHP object, save it to an array, then modify it and save the new modified instance to the array, we have to use the PHP **clone** keyword to make a *deep copy* of the object. By default, PHP only stores *pointers* to object instances (i.e. a memory-address; or a *shallow copy*) -- so pushing a series of modified state-objects to an array without using clone would result in an array where all slots will contain the **same**, **final** **state** of the object (since they would all be pointers to the same address)!!!

* **function eatPolyps** : selecting orthogonal elements from a 2D (or 1D) array:

When executing Action1 of the RE game, you need to pick up all same-color coral tiles that are north/south/east/west-adjacent (or *orthogonally*-adjacent) to the shrimp you are eating.

To do this, I use an array variable as a *Queue*; I initialize it with the shrimp-location, erase the contents of that location in my boards-array, then look at the locations directly N/S/E/W of shrimp-location to see if they contain corals of the same color (as well as ensuring the new location is not already-present in array, to avoid infinite loops!): if so, those locations are added to the array. I process in this manner each location in the array. When I reach the array's end, I have processed each orthogonally-adjacent same-color coral to the shrimp-tile. See **function eatPolyps($cell, $state)** in the Free Code below for PHP details.

* When RElapse.php is loaded after the $states[] array has been saved in $\_SESSION[], your class definition should precede the session\_start() call for the array of objects to be correctly interpreted…

include("REstate\_cls.php");

/\* REstate class definition should *precede* session\_start call \*/

session\_start();

* if (strpos($act, "chose action 1,") !== FALSE) -- verifies that value is 0 (i.e. FALSE) *and* boolean. Because 0 might just mean the string was found at position 0 of the string!

**FREE CODE**:

Functions and snippets of code I have found to be useful. Of course, some assumptions from my own code may be baked-in, so… use at your own risk! ;)

**<?php**

**include("REstate\_cls.php"); // REstate class definition**

**session\_start();**

**/\* Uses $\_GET array to load selected gamepage -- or a default if none provided! \*/**

**if (isset($\_GET["games\_id"])) {**

**$gid = $\_GET["games\_id"];**

**} else {**

**$gid = (string) 99147; //115839;**

**}**

**{ array\_push($states, clone $state); }**

**function ignoreEntry($s) {**

**// signals whether to ignore log entry based on contents**

**$res = FALSE;**

**$ignorestrs = array("triggered the end", "picked a polyp", "picked two starting");**

**foreach ($ignorestrs as $str) {**

**if (strpos($s, $str) !== FALSE) {**

**return TRUE;**

**}**

**}**

**return $res;**

**}**

**function eatPolyps($cell, $state) {**

**/\* This function 1) collects the indexes of all orthogonally-adjacent cells containing the same color polyp-tile as the one under the shrimp being eaten ($cell), 2) erases them in the $state object, and 3) returns the number of cells affected**

**\*/**

**$b = $state->cell2board($cell); // Board #**

**$c = $state->cell2slot($cell); // 1-D reef-array slot-number**

**$ps = array(1=>$c); // temporary array var (to be "used as a Queue")**

**$p = $state->bord[$b][$c][0]; // get 0th character in string-value (polyp color)**

**$i = 0; // # of valid coral-tiles processed**

**while ($i < sizeof($ps)) {**

**$i++;**

**$c = $ps[$i];**

**if (($c % 7 < 6) && ($state->bord[$b][$c+1] == $p) && !in\_array(($c+1), $ps))**

**{$ps[sizeof($ps)+1] = $c+1;}**

**if (($c % 7 > 0) && ($state->bord[$b][$c-1] == $p) && !in\_array(($c-1), $ps))**

**{$ps[sizeof($ps)+1] = $c-1;}**

**if (($c > 6) && ($state->bord[$b][$c-7] == $p) && !in\_array(($c-7), $ps))**

**{$ps[sizeof($ps)+1] = $c-7;}**

**if (($c < 35) && ($state->bord[$b][$c+7] == $p) && !in\_array(($c+7), $ps))**

**{$ps[sizeof($ps)+1] = $c+7;}**

**$state->bord[$b][$c] = ''; // erase cell contents**

**}**

**return $i; // optional; just checking my number matches the Log's**

**}**

**function xtractstr($ss, $begstr, $endstr) {**

**// Note: $begstr and $endstr should directly abut string to be extracted**

**$beg = strpos($ss, addslashes($begstr));**

**$end = strpos($ss, addslashes($endstr), $beg);**

**if ($end===FALSE) {$end = strlen($ss)+1;} // extract from $beg to end\_of\_string**

**if ($beg !== FALSE && $end !== FALSE) {**

**$beg = $beg + strlen($begstr);**

**$s = trim(substr($ss, $beg, $end-$beg));**

**return $s;**

**} else {**

**print("<b>xtractERROR: $begstr -- " . htmlentities($endstr) . "</b>beg: $beg; end: $end<br>" . PHP\_EOL);**

**return("xtractERROR: $begstr -- " . htmlentities($endstr) );**

**}**

**}**

**////////////////////////////////////////////////////////////////////////**

**///////////////////////// STATE Class //////////////////////////////////**

**class REstate {**

**public $bord; // 4-length array of Reef arrays**

**public $games\_id; // id for SBW RE game whose state this object maintains**

**public $doms; // 10-length array of dom-tiles?**

**public $players; // 4-length player-names (+ shrimp + cubes?)**

**public $polyps; // 3D array: polyps[PLYR][""][CLR]**

**public $cubes; // 4-length array of larva cube arrays (2D): cubes[PLYR][CLR]**

**public $shrimp; // 4-length array of in-hand/onboard/eaten shrimp (2D): shrimp[PLYR][""]**

**public $name\_map; // assoc array of names => player-order ints**

**public $shrimp\_map; // numeric array of players' shrimp characters**

**function initState($nms, $gid) {**

**$this->bord = $this->initBord();**

**$this->games\_id = $gid;**

**$this->polyps = array\_fill(0, 4, array("eaten"=>array\_fill(0, 5, 0),**

**"inhand"=>array\_fill(0, 5, 0),**

**"consumed"=>array\_fill(0, 5, 0)) ); // polyps[PLYR][""][CLR]**

**$this->players = $nms;**

**$this->shrimp = array\_fill(0, 4, array("inhand"=>4,**

**"onboard"=>0,**

**"eaten"=>0) ); // shrimp[PLYR][""]**

**$this->cubes = array\_fill(0, 4, array\_fill(0, 5, 0)); // cubes[PLYR][CLR]**

**$this->name\_map = array($nms[0]=>0, $nms[1]=>1, $nms[2]=>2, $nms[3]=>3);**

**$this->shrimp\_map = array(0=>'P', 'G', 'R', 'Y');**

**$this->polyp\_map = array(0=>'w', 'y', 'o', 'p', 'g');**

**}**

**function initBord() {**

**// lists contents of each reef-board (includes invalid placement spaces)**

**$rf0 = array(0=>'x','x','x','','','','x','x','','','y','','','x','','','x','','w','','','','g','','','','x','','','','p','x','o','','','','','','','','','');**

**$rf1 = array('x','x','x','','','','x','','','','g','','','x','','','y','','x','','','','x','',' ','','w','','','','o','','p','','','x','','','','','','x');**

**$rf2 = array('x','','','x','','','','x','','','','o','','x','','','w','x','','y','','','','',' ','','x','','x','','p','','g','','','x','','','','x','','');**

**$rf3 = array(0=>'x','','','','','','x','','','','o','x','','','','','x','','g','','','','w','','','','','','x','','p','','y','','x','x','','','x','','','x');**

**$bord[0] = $rf0;**

**$bord[1] = $rf1;**

**$bord[2] = $rf2;**

**$bord[3] = $rf3;**

**return $bord;**

**}**

**function clrstr2int($c) {**

**$c = lcfirst(ltrim($c))[0];**

**if ($c == "w") {**

**$res = 0;**

**} else if ($c == 'y') {**

**$res = 1;**

**} else if ($c == 'o') {**

**$res = 2;**

**} else if ($c == 'p') {**

**$res = 3;**

**} else {**

**$res = 4;**

**}**

**return $res;**

**} //end clrstr2int() method**

**function cell2slot($cell) {**

**/\* Given a cell address, determines which slot in a 42-element (0-based) array represents it \*/**

**$let = lcfirst($cell[0]);**

**$num = (integer)(substr($cell, 1));**

**$num--; // 0-based array**

**$let = ord($let) - 97;**

**if ($let < 7) {**

**if ($num < 6) {**

**$slot = $num \* 7 + $let;**

**} else {**

**$slot = ($num-6) \* 7 + $let;**

**}**

**} else { // $let >= 7**

**if ($num < 6) {**

**$slot = $num \* 7 + $let - 7;**

**} else {**

**$slot = ($num-6) \* 7 + $let - 7;**

**}**

**}**

**return $slot;**

**} //end cell2slot() method**

**function cell2board($cell) {**

**/\* Given a cell-address, determines which board-array contains it; outputs INT \*/**

**$let = lcfirst($cell[0]);**

**$num = (integer)substr($cell, 1);**

**$let = ord($let) - 97;**

**if ($let < 7) {**

**if ($num < 7) {**

**return 0;**

**} else {**

**return 2;**

**}**

**} else { // $let >= 7**

**if ($num < 7) {**

**return 1;**

**} else {**

**return 3;**

**}**

**}**

**} //end cell2board() method**

**}**