Università della Svizzera italiana

Facoltà di scienze informatiche

Programming Fundamentals 2 - Smalltalk **2004 - 2005**

Prof. Michele Lanza Paolo Bonzini, Mircea Lungu and Romain Robbes

Assigment 3 : First steps with Seaside Due 23/03/2005 @ 10:00

This assignment will let you practice the basics of Seaside before the next lecture. For this, we will rely on the work you have been on hands-on 2, and we will use the FMPlayer and FMPlayerPool classes as the model of our application. We will define separate components to manipulate them.

Exercise 1 - Getting started

Seaside is already loaded in the Smalltalk image you have installed. To test it, point your browser to `http://localhost:8008/seaside/go/counter`. Remember this guy ? ;-)

Then, create and register your new application. Since you are practicing, we will build a fairly simple application, with a simple control flow, so we will directly use a Component, and not a Task.

- Create the WAPlayerPool component, as a subclass of WAComponent (in the FootballManager package).
- Override the `#canBeRoot` method.
- Override the `#renderContentOn:` method, to display a simple message, such as “hello, world”.
- Go to the class side, and override the `#initialize` method. Paste the following code in it: `self registerAsApplica`
  Then in a Workspace, evaluate `WAPlayerPool initialize`.
- Hopefully, clicking `http://localhost:8008/seaside/go/handson2` will greet you and a few other people.

Now, we can do the interesting stuff . . .

Exercise 2 - Working with a single component

The first thing that we want to do is to have a component able to display a list of players. To do this, we need to add an instance variable to WAPlayerPool containing the player pool that needs to be displayed. This player pool will be initialized randomly at creation time (ie in the instance-side `#initialize` method).

Now:

- Rewrite `#renderContentOn:` to display all the players in the pool, and their statistics. Display them as you see fit, possibly using tables. Use the tic-tac-toe as an example of how to use the WAHTMLRenderer. Look at the implementation of WAHTMLRenderer and its hierarchy. To keep things simple, proceed step-by-step, checking the result as often as possible. *Hint :* FTDR. It is probably smart to define a helper method such as `#renderPlayer:on:`.
- Now add on top of the page some of the simple statistics that you know about players, such as number of attackers, or if there are enough players to play a game. . . . Again, I trust you to have a decent display of the data.

- Add another instance variable in WAPlayerPool, which is a collection of *currently selected* players. By default, it will be equivalent to the entire player list. Use this variable to display all the players, instead of querying the FMPlayerPool.

- Use the functions you defined returning a subcollection of the players, such as `FMPlayerPool>>#potentialAttacker` to display some actions you can do on the web interface (reminder: use anchorWithAction:text:). These actions will filter the list displayed. For example, clicking on the "potential attackers" link will modify the currently selected players to be only the one returned by `#potentialAttackers`. Define also `#potentialGoalKeepers`, ... and provide an action to view the unfiltered list of players (resetting the currently selected players instance variable). *Hint :* Do not forget to split `#renderContentOn:` with helper methods!

Exercise 3 - Working with several components

Now we will define one component per player. We will define the WAPlayer class, again a subclass of WAComponent. Add an instance variable containing the subcomponents of WAPlayerPool. This variable will be initialized right after the pool and the selected player variables are.

Your WAPlayer should have its own `#renderContentOn:` method, containing what was previously related to players, and a player instance variable.

- Rewrite `WAPlayerPool>>#renderContentOn:` to use the WAPlayer subcomponents.

- Override the `#children` method, to return the subcomponent collection.

- Now test if the actions such as potential attackers are still displaying filtered results. Normally, they should not – not yet. To fix this, you have to *selectively render* the subcomponents. That means that you should check if a Component's player is member of the list of currently selected players of WAPlayerPool, before rendering it.

- Add an action to display a WAPlayer "folded". WAPlayer should have an extra instance variable telling if they are folded or not. If a WAPlayer is folded, it will only display its name when it is rendered. Otherwise, it will display the entire statistics.

- Make the statistics of each player modifiable. That is, rather than just displaying the text for a statistic, you should provide a link allowing to edit it. You can use `#request:` to receive input from the user.

- Try to modify the statistics of a player such that, for example, he is no longer a potential attacker, when the filtered list displays only potential attackers. Find a way to update the filtered list automatically. *Hint :* You may need some extra state variables for this: on WAPlayerPool, a symbol telling which filtering is actually occuring (`#attackers`, `#defenders`, `#middleField`, `#none`). On WAPlayer, you may need an instance variable pool, containing a link to the WAPlayerPool, so that when the player is modified, you have a pointer to the object that needs to be updated.

- Add an action on each player, "fire", which removes the player from the pool. The list of players should be updated automatically.