sqrlab / **Threaded-Paws**
forked from luisarojas/threadedpaws

Serious game for learning about concurrency - thread interleavings, data races and deadlocks.
http://www.sqrlab.ca/software/threade...

#serious-game #concurrency #concurrent-programming #learning #education #computer-science

109 commits · **1** branch · **0** releases · **1** contributor · MPL-2.0

Branch: master · New pull request · Create new file · Upload files · Find File · Clone or download

This branch is 8 commits ahead, 35 commits behind luisarojas:master. Pull request · Compare

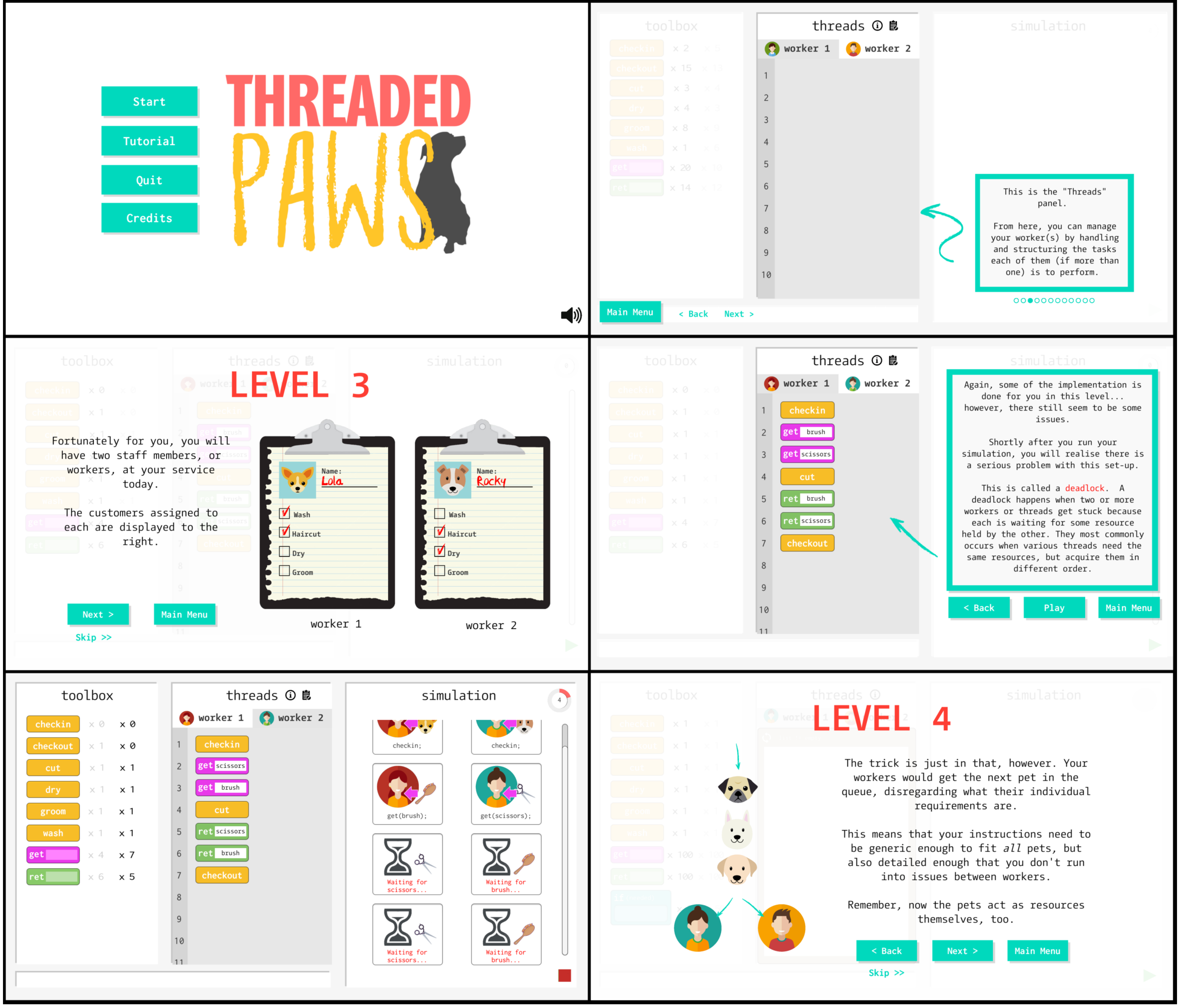
luisarojas Changed build settings — Latest commit `1b96540` on Nov 27, 2017

audio	Actions now take into account required resources.	2 years ago
images	Added new credits	2 years ago
readme_media	Added some screenshots, and updated the readme with the updated UI lo...	2 years ago
src	Changed build settings	a year ago
.gitignore	Updated gitignore file.	a year ago
License.txt	Create License.txt	2 years ago
README.md	Changed link.	a year ago

README.md

Threaded Paws is a serious game for learning about concurrency concepts including thread interleavings, data races and deadlocks.

Advances in multi-core processors continue to increase the need for concurrent programming. Unfortunately, writing concurrent programs remains difficult due to the many, possibly unexpected program executions. Furthermore, students learning concurrent programming need to comprehend and avoid common pitfalls such as data races and deadlocks. To address this need, we have developed **Threaded Paws**, a game-based learning tool that teaches students to identify and fix concurrency pitfalls and bugs.



This is the "Threads" panel.

From here, you can manage your worker(s) by handling and structuring the tasks each of them (if more than one) is to perform.

Fortunately for you, you will have two staff members, or workers, at your service today.

The customers assigned to each are displayed to the right.

Again, some of the implementation is done for you in this level... however, there still seem to be some issues.

Shortly after you run your simulation, you will realise there is a serious problem with this set-up.

This is called a deadlock. A deadlock happens when two or more workers or threads get stuck because each is waiting for some resource held by the other. They most commonly occurs when various threads need the same resources, but acquire them in different order.

The trick is just in that, however. Your workers would get the next pet in the queue, disregarding what their individual requirements are.

This means that your instructions need to be generic enough to fit *all* pets, but also detailed enough that you don't run into issues between workers.

Remember, now the pets act as resources themselves, too.