Selection and Insertion Sort

Animal Sorting Lab

# Main Objectives

* Watch and study the algorithms of the selection and insertion sorts
* Write the selection sort algorithm including swapping values in a List.
* Develop and write an algorithm which shuffles a List.
* Write the insertion sort algorithm.

# Prior Knowledge

* Familiarity with List methods. (The lab could be modified to work with arrays).
* Familiarity with loop and control structures.

# The Lack of Greenfoot Background

Although this lab is written with Greenfoot to perform the visualizations, students do not need to use any of the Greenfoot API. All of the visual movements of objects around on the screen has been provided in helper methods. Students can thus write a regular selection sort and a regular insertion sort without having pre-learned Greenfoot.

# Getting Started

Install the Greenfoot IDE: <https://www.greenfoot.org/download>

* Greenfoot went through many changes beginning with the 3.0.0 release. Version 2.4.2, although older, tends to be one of the most stable versions but is missing a few of the newer additions.

In your Animal Sorting folder run the project file with the Greenfoot icon to begin.

# Demonstration

View a sample demonstration of what the Animal Sorting lab could look like here:

<https://www.greenfoot.org/scenarios/23655>

# Instructions

The instructions are written directly in the MyWorld class as comments. Students will be writing four methods: selectionSort, swap, insertionSort, and shuffle.

# Extensions and Modifications

* The Animal class could be written by students. It has a few getters and setters and implements Comparable versus other Animals. Comparable is no longer on the APCSA exam, but teachers who still use Comparable should feel comfortable with having students write an Animal class implementing Comparable.
* Writing Merge sort is not part of the APCSA exam needs. However, an extension for faster students would be to have them attempt to write merge sort. Visual movement and swapping of Animals is not required, visualLineUp can take care of that, but strong students could attempt to make the individual merges be seen visually for a challenge.
* There is minor benefit to removing the three InsertionSort/Randomize/SelectionSort classes. Students who are familiar with calling methods from a specialized World could easily write these classes themselves.
* All the movement helper methods end up using magic numbers multiple times. Students could be tasked with turning these into static fields so allow the program to be resized as needed or make the numbers auto-adjust if the World’s size changes. A good extension for strong students is to be able to adjust the size of the world and have all of the visuals work perfectly no matter which numbers are given for the width and height.

