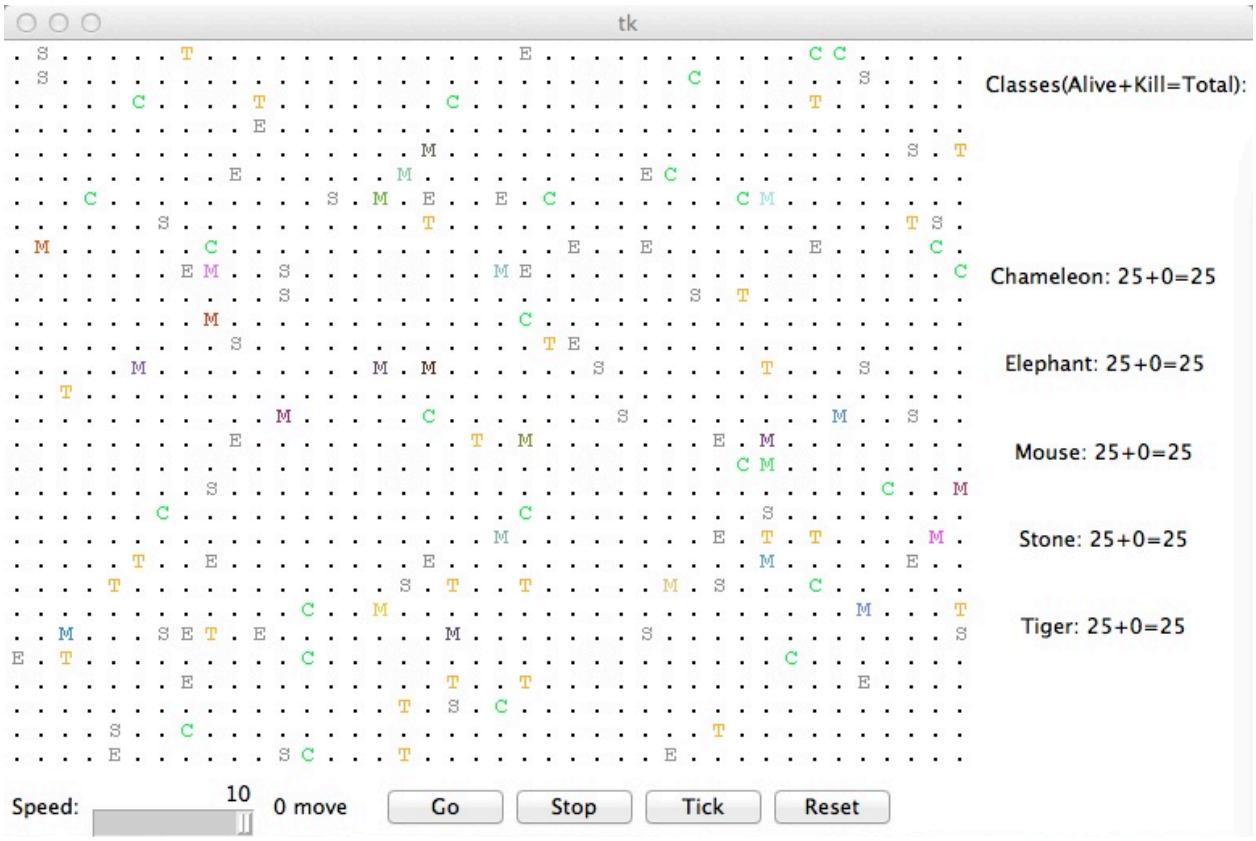
Assignment #1: Critters! 2.0 (Party Edition)
CSCI 364 Spring 2019 Oberlin College
Due: Monday February 18 at 1:30 PM

Background

Our first assignment this semester has two goals:

1. Have some fun experimenting with creating intelligent agents that interact in an environment, updating a favorite lab from CSCI 150
2. Practice using GitHub for source code management

In particular, we will be creating critters that play a modified version of the critter lab. The original details for the critter lab can be found at:
http://www.cs.oberlin.edu/~aeck/Fall2018/CSCI150/Labs/Lab10/index.html



Here, the environment is a square grid, where critters can move around. If two critters try to occupy the same space on the grid, they will interact with one another. New to this assignment, we have additional types of interactions. Each interaction, critters choose one of five actions:

1. Three fighting actions (from the original lab): ROAR, SCRATCH, and POUNCE
2. A PARTY action, and
3. A HEAL action

The goal of the critters is to gain as many **karma** points as they can from their interactions with other critters.

The rules of interactions are as follows:

1. If two critters choose to fight, then one is a winner and one loses, based on their chosen actions (similar in structure to Ro-Sham-Bo or Paper-Rock-Scissors): ROAR beats SCRATCH beats POUNCE beats ROAR. The losing critter loses 25 **health** (starting with a maximum of 100), and the winning critter gains one karma point. The losing critter's karma is unchanged.

2. If one critter choses to fight and the other does not, the fighter automatically wins. Once again, the losing critter loses 25 health, but the winning critter *loses* karma for violently attacking a friendly critter. The amount of karma lost depends on the action chosen by the friendly critter: 3 karma lost for a PARTY action and 5 karma lost for a HEAL action. Again, the losing critter's karma is unchanged.

3. If both critters choose not to fight, both gain karma and neither loses health. In this case, a PARTY action earns 3 karma. If a critter chooses the HEAL action and the other critter has less than 100 health, then the other critter's health increases by up to 50 (for a maximum of 100 health), and the critter performing the HEAL action gains 5 karma. On the other hand, if a critter chooses HEAL and the other critter has full health, no healing is done and the healer gains no karma.

If a critter's health reaches 0 (or below), it is removed from the environment and it can no longer accumulate karma points. After each environment is simulated (for a maximum number of iterations), the species of critter that accumulated the most karma points (across all individuals of that species, including those who ran out of health) will be declared the ultimate victor.

Gitting Started

To begin this assignment, you will need to create a GitHub account if you don't already have one, as well as download the code for this assignment. Instructions can be found in the **Introduction to GitHub** document on the class webpage (under Handouts).

Once you have a GitHub account, you can get started on the assignment by following this link:

https://classroom.github.com/a/aH6HeVaI

After you have the source code, your assignment is to **implement five species of critters** that will interact in the critter environment. Your critters should be in python files named name1.py,

name2.py, …, name5.py, where "name" is replaced by your first initial and last name. So for me, my critters would be named aeck1.py, aeck2.py, …, aeck5.py. Each file should contain a different Python class, where the name of the class is the same as the name of the file, except with your first initial capitalized (for me, Aeck1, Aeck2, …, Aeck5).

To implement a critter, you should inherit from the Critter class in critter.py. In particular, you need to implement 5 functions:

```
# decides what the critter should do when interacting with
# another critter
#
# oppInfo describes the environment
def interact(self, oppInfo):

# determines what direction the critter should move in from the set
# critter.NORTH, critter.EAST, critter.SOUTH, critter.WEST
#
# info describes the environment
def getMove(self, info):

# returns a String that should be used to display the critter on the
# environment GUI
def getChar(self):

# returns a color constant (defined in colors.py) that should be used
# to display the critter on the environment GUI
def getColor(self):

# alerts the critter that an interaction happened so that it can
# update its knowledge about the environment
#
# won is a Boolean indicating whether or not the critter won its
# interaction
# oppFight is the chosen action of the opponent
def interactionOver(self, won, oppFight):
```

You should implement your critter's intelligent decisions of how to behave in the environment in the `interact` and `getMove` functions, whereas `interactionOver` will help the critter gain information about interactions that it can use to decide how to act in the future. Please be creative and have fun when giving your critters the ability to reason about how to act!

At least three of your critters should have interesting, (semi-)intelligent behavior. That is, do not *only* create really dumb critters that (1) pick the same action every time, (2) pick purely random actions each time, etc. This is your chance to have some fun, be creative, and start thinking about how an agent (i.e., a critter) might make its own decisions to achieve a goal (i.e., maximize its karma, possibly by needing to survive as long as possible).

Of note: critters have internal fields `health` and `karma` storing how much health they have (from 0 to 100) and how much karma they have earned (starting at 0). These fields should **NOT**

be updated by your code, but you can read them if it will help your critter make wise decisions of how to act.

The `oppInfo` and `info` parameters to `interact` and `getMove`, respectively, are `CritterInfo` objects containing the following fields and functions:

```
x # the current critter's x coordinate
y # the current critter's y coordinate
width # the width of the simulation world
height # the height of the simulation world
char # the current critter's display character
color # the current critter's display color
getNeighbor(direction) # a function that, when called with a parameter
   # representing one of the direction constants, returns the name of
   # the class (NOT the display character) of the critter in that
   # location (i.e. the location that is one space in the given
   # direction of the current critter.) and "." if there is no critter
   # there. locations include critter.NORTH, critter.NORTHEAST, etc.
getNeighborHealth(direction) # a function that, when called with a
    # parameter representing one of the direction constants, return
    # the health of the critter in that location and 0 if there is
    # no critter there
```

As you implement your five critters, please remember to commit them to your repository on GitHub. You do not need to wait until you are done with the assignment; it is good practice to do so not only after each coding session, but maybe after hitting important milestones or solving bugs during a coding session.

Make sure to document your critter class files, explaining what they do and how you implemented them. We might come back to this later in the semester… so good commenting will help us remember how they work!