PaSciRo Documentation

Lisa Luff

Contents

Overview

Purpose

My program is a simulation of a section of the universe named PaScirRo.

PaSciRo is populated by Humans, Vulcans, and Klingons, species of aliens (and one non-alien) based on the popular comic, book, TV, and movie series Star Trek. In our simulation we look at the movement, breeding and fighting patterns of these three species within a select area over a given period.

The simulation is designed to make use as easy as possible, while allowing enough user specified options to allow for a wide variety of simulation possibilities.

Features

Each species has unique values assigned to them based on my personal beliefs about their likely attributes, meaning that each species is completely unique.

As there are several steps that allow for an element of random choice, re-running the simulation repeatedly with the same inputs can lead to a wide variety of outputs. This variety is expanded even further by allowing for so much control over the initial grid by the user.

Observe the movement, breeding and fighting patterns with an interactive plot with a playable time slider, and a printed csv file with a summary of their interactions.

User Guide

Requirements

- The latest version of Python
- The following python packages:
- random
- sys
- pandas
- time
- plotly

Getting started

All elements of the program have been brought together so that you only need to give a single command to run the entire simulation. The file to run is assignment_main.py. As it is a python program, your command will start with python or python3 depending on your set up.

- Six arguments need to be given when you run the program as part of the command. The arguments, in the order they are to be input are as follows:
  - *n_humans*: Start number of Human's
  - *n_vulcans*: Start number of Vulcan's
  - *n_klingons*: Start number of Klingon's
  - *grid_height*: How many grid rows
  - *grid_length*: How many grid columns
  - *years*: How many years to simulate
    These inputs should be given as numbers separated by a space

To run

The command to run the program should look like this:
python assignment_main.py *n_humans n_vulcans n_klingons grid_height grid_length years*
Eg.

```
$ python assignment_main.py 10 10 10 20 20 100
```

If the input given is incorrect, the simulation will not run and an error message will be given. All you need to do is run it again with the correct input.

Code

assignment_grids.py

This program was made to be used for any grid purposes as a standalone. So, all the methods are non-specific to the content of this specific simulation.

- Classes:
  - Grid()

Note: The grid is made of nested lists. Every internal list holds a 0 once the grid is made, so any objects will be starting from 1 in the coordinates it's held in if using this by itself. Do not remove the 0 to maintain consistency.

Grid

- Grid class variables:
- NORTH [0, -1]
- SOUTH [0, 1]
- EAST [1, 0]
- WEST [-1, 0]
- COMPASS: [NORTH, SOUTH, EAST, WEST]
  Direction references to be used for movement

- Grid methods:
  - ___init___(*self, height, width*)
    - *height*: The number of grid rows
    - *width*: The number of grid columns
    - Initialise the list to be used to create the grid
  - initialise_grid(*self*)
    - Add specified number of rows and columns of lists
  - check_barriers(*self, row, column*)
    - Used to check if the given coordinates are a valid location in the grid
    - *row*: Row index in the grid
    - *column*: Column index in the grid
  - add_object(*self, obj, row, column*)
    - Add an object to a specified location in the grid
    - *obj*: The object to be appended to the list in the given location
    - *row*: Row index in the grid
    - *column*: Column index in the grid
    - Run check_barriers
  - get_objects(*self, row, column*)
    - Gives you the list at the specified location
    - *row*: Row index in the grid
    - *column*: Column index in the grid
  - remove_objects(*self, obj*)
    - Removes a specified object
    - *obj*: The object to be removed
    - Object needs to hold the following information:
    - *self.obj_row*: Current row index
    - *self.obj_column*: Current column index
  - object_movement(*self, obj, p_change*)
    - Moves an object one space in a weighted random direction
    - *obj*: The object to be moved
    - *p_change*: The probability of changing directions 0-1
    - Object needs to hold the following information:
    - *self.obj_row*: Current row index
    - *self.obj_column*: Current column index
    - *self.direction*: Current facing direction
    - Run add_object, and remove_object
  - print_grid(*self*)
    - Print the grid nicely

assignment_species.py

This program was made to hold all the class information for the three alien species.

- Classes:
  - Species()
    - *name*: name of the instance
  - Human(Species)
  - Vulcan(Species)
  - Klingon(Species)

Species

- Species methods:
  - ___init___(*self, name*)
    - Set instance *name*
    - Initiate gender, lifespan, age, row, column and direction
  - decide_lifespan(*self*)
    - Randomly gives each instance a max age between 70 and 100
  - decide_gender(*self*)
    - Random assigns each instance a gender 'm' or 'f'
- Human, Vulcan and Klingon methods:
  - All the same with different class values
  - ___repr___(*self*)
    - Represents each instance using their *species*: *name*

Human, Vulcan and Klingon

- Human, Vulcan and Klingon class variables
  - *SPECIES_TYPE*: The name of the species
  - *SPEED*: Steps they take per movement cycle
  - *IMPULSIVENESS*: Probability of changing direction
  - *BREED_SKILL*: Likelyhood to breed
  - *START_BREED*: Holds original value
  - *AGGRESSIVENESS*: Likelyhood to fight
  - *START_AGGRESSIVENESS*: Holds original value
  - *FIGHT_SKILL*: Likelyhood to win a fight

assignment_species_grid.py

This program links assignment_grids.py and assignment_species.py. Main program for simulation.

- Dependencies:
  - random
  - assignment_grids.py
  - assignment_species.py

- Classes:
  - SpeciesSimulation()
  - SpeciesGrid(SpeciesSimulation)

SpeciesSimulation

- SpeciesSimulation methods:
  - ___init___(*self*)
    - Initiate lists of species instances
      - humans, vulcans and klingons
    - Initiate and hold counters for output
      - human, vulcan and klingon _total
      - current_total and total
      - baby_counts
      - fight_counts
  - new_object(*self, gridi, obj_type, name, pos*)
    - Creates a new species instance and add it to the grid
    - *gridi*: The grid instance
    - *obj_type*: Species type
    - *name*: Species instance name
    - *pos*: Current position [row, column]
    - Run decide_gender and decide_lifespan
    - Run add_object
  - make_baby(*self, gridi, obj1, obj2*)
    - Makes a baby if interaction leads to baby
    - *gridi*: The grid instance
    - *obj1*: First interacting object
    - *obj2*: Second interacting object
    - If different species, random assign baby species one of parents
    - Run new_object
  - have_fight(*self, gridi, obj1, obj2*)
    - Randomly (weighted) decides who dies if interaction leads to fight
    - *gridi*: The grid instance
    - *obj1*: First interacting object
    - *obj2*: Second interacting object
    - Run remove_object
  - obj_interaction(*self, gridi, obj1, obj2*)
    - If there is an interaction, decides what to do
    - *gridi*: The grid instance
    - *obj1*: First interacting object
    - *obj2*: Second interacting object
    - baby_prob
      - Nested function, finds probability of interaction leading to a baby outcome
    - Randomly (weighted) decides if interaction leads to baby, fight or nothing

SpeciesGrid

- SpeciesGrid methods:
  - ___init___(*self, n_list, grid_height, grid_length*)
    - *n_list*: List with start numbers for each species
    - *grid_height*: Number of grid rows
    - *grid_length*: Number of columns
    - Gets parent ___init___
    - Initiates grid
  - grid_setup(*self*)
    - Creates Grid instance
    - Runs initialise_grid
  - initial_objects(*self*)
    - Creates species instances and places them in grid based on input numbers
    - get_pos()
      - Nested function randomly decides grid location
    - Run new_object
  - obj_movement(*self, obj*)
    - Makes each species object move their specified number of times
    - Object needs to hold the following information:
      - *self.IMPULSIVENESS*: Probability to change direction
      - *self.SPEED*: Number of steps to take
      - *self.age*: How many simulation years they've been alive
      - *self.lifespan*: How many years they can live
    - Run object_movement
    - Check age with lifespan, and use remove_object if reached lifespan
  - find_interaction(*self*)
    - Finds where there are two or more objects in a given grid location
    - If more than one, runs obj_interaction

- grid_fullness(*self*)
  - Varies breeding up and aggressiveness down if population numbers low
  - Varies oppositely if population numbers high
  - Object needs to hold the following information:
    - *self.START_AGGRESSIVENESS*: Hold start value
    - *self.START_BREED*: Hold start value
    - *self.AGGRESSIVENESS*: Likelyhood to fight
    - *SELF.BREED_SKILL*: Likelyhood to have babies
  - skill_current(*obj, skill, value*)
    - Nested function to find new skill level
    - *obj*: Species object
    - *skill*: Breed skill ('b') or aggressiveness ('a')
    - *value*: 0 - 0%, 1 - 100%, 2 - -2/3%, 3 - +2/3%
  - apply_skill(*skill, value*)
    - Nested function to change skill levels for all living species instances
    - *skill*: Breed skill ('b') or aggressiveness ('a')
    - *value*: 0 - 0%, 1 - 100%, 2 - -2/3%, 3 - +2/3%
    - Run skill_current
  - End program and prints message if population reaches 0, or equals number of grid elements
- print_grid(*self*)
  - Prints grid nicely using grids print_grid

assignment_data.py

Collects the data from the current simulation, does some basic analysis, then prints it nicely and stores it as a dataframe in assignment_data.csv

- Dependencies:
  - sys
  - pandas
  - assignment_data.csv
- Function:
  - data_collector(*usergrid*)
    - *usergrid*: The grid instance from species_grid
    - Specific to grids, species, and species_grid, so cannot be repurposed without meaningful changes

assignment_data.csv

Holds dataframe of data from assignment_data.csv

Note: File is overwritten every simulation, if you want to save your data, copy the file before running another simulation

assignment_main.py

Combines everything else as the interface to run the simulation

- Dependencies:
  - sys
  - time
  - pandas
  - plotly
  - assignment_species_grid.py
  - assignment_data.py
- Program input:
  - *n_humans*: Start number of Human's
  - *n_vulcans*: Start number of Vulcan's
  - *n_klingons*: Start number of Klingon's
  - *grid_height*: How many grid rows
  - *grid_length*: How many grid columns
  - *years*: How many years to simulate
- Collect inputs
- Prints welcome message
- Creates SpeciesGrid
- Runs grid_setup, initial_objects
- Initiates date, s_type, s_name, s_row and s_column data collectors
- Function:
  - run_simulation()
    - Runs obj_movement for each living species object for one year
    - Sends data to data collectors
    - Runs find_interactions
    - Runs grid_fullness
- While loop to stop run based on grid_fullness
  - Nested for loop for number of years user requested, to run run_simulation that many times
- Run data_collector
- Create graph with plotly
- Print goodbye message and end program