SU:DA2004_INDU_Projektrapport
Stockholms Universitet

2022–03–20

Forewords

This report is written in academic Swenglish and comments the attached codeproject. Quick info about the choosen INDU project below.
Choosen project: Tre i rad (Uppgift 1 & 2)
Python Modules used: unittest, mock, random, datetime, matplotlib

Contents

1 File structure

Within the working folder, you will find the following file structure (to the right). **To run the program**, run: *main.py*, this will execute the "Tre i rad" simulator program and start prompting for user input via the terminal. Then follow instructions given in the terminal to proceed.

The **main.py** is the program, while the other modules are just definitions of functions that won't run unless given input from main.

```
SU_DA2004_HT2021_INDU_Projekt
|
•– Code
|   |
|   •– Results
|   |   |
|   |   •– Program output:↦ PDF diagrams
|   |
|   •– game_sim.py
|   |
|   •– main.py
|   |
|   •– statistics_export.py
|   |
|   •– tests.py
|   |
|   •– user_prompts.py
|
•– Kursfiler
|   |
|   •– regler.pdf
|   |
|   •– tre_i_rad.pdf
|
•– Rapporten_DA2004_HT2021.pdf
```

2 Program flow and Modules

As seen in the file structure this program is seperated into five modules, each with a descriptive name. Here's a short review of each.

main.py

This is from where the program is run for normal execution. It triggers the user prompts and iterates feeding this input to the *game_sim.py* module for simulations. Then it consolidates this data for export, handing it to the *exportResults()* function in the *statistics_export.py* module.

user_prompts.py ⇒ up.*

All user prompts are contained within this module. Each prompt function preforms error handling, that iterates until the user gives acceptable input. Only predefined inputs are allowed, minimizing the risk of having forgot of some fringe example.

game_sim.py ⇒ gs.*

The most complex module of them all, this is where the *tic-tac-toe* games are simulated. While *main.py* is the main module of the program, the *game_tictactoe()* is the main function of the *game_sim.py* module. To best understand this module use that function as your starting point.

statistics_export.py ⇒ se.*

When all is said and done through the simulations the data of each run is saved within the *score_board* in *main.py*. This data is sent to *exportingResults()* where processing is done to create a matplotlib diagram and save it to the *Results/* folder. The module also contains the *output_summary()* that will print a statistics summary to the terminal before the program ends.

tests.py

Running this will preform unittests of the whole program. Each function within the whole project will be tested except two, who are "out-commented" because they have no following dependencies and only prints to terminal making them difficult to unittest anyway.

3 Code design, Algorithms & Data structures

Comments on some of the design decisions within the program.

Error handling

Prevention is the best cure. This program attempts to catch errors before they become troublesome. Loading of modules are included in "try–except" blocks. And each of the functions in the *user_prompts.py* module, contain error handling and enforced integer input.

Algorithms & Data structures

The data structures in this program are kept simple. Only in use are the most fundamenal: *integers*, *floating points*, *lists* and *strings*. The most complex combination of these occur in the *board* variable within the *game_tictactoe()* function from *game_sim.py*. The *board* variable is constructed by the *make_move()* function, seen below, and it's a list of integer-lists. Where each integer-lists stores a player ID together with coordinates for a move on the board, this represents the placement of a piece.

Listing 1: game_sim.py

```
def make_move(board:list,next_move:list,player:int):
    """Claiming the next_move,
    by attaching player ID before adding to board."""
    [x, y, z] = next_move
    board.append([player, x, y, z])

    return board
```

The last entry to the *board* is also always the last move. This is used when checking for the result of a new move. Using parametrisation, the function look for pieces along the lines where they ought to be and adds score for each finding. A score of three is a win! See the code for the full function.

Listing 2: from gs.check_result()

```
# Parameterizes a line in the direction of adjacent
# Then checks for three in a row along that line
    for i in adjacent:
        [dx,dy,dz] = [i[1]-x,i[2]-y,i[3]-z]
        if dx != 0 or dy != 0 or dz != 0:
            score = 0
            for j in [0,1,2]: #If from endpoint
                if [player,x+dx*j,y+dy*j,z+dz*j] in board:
                    score += 1
```