Twisty Puzzle Analysis version 2.1

by ***Sebastian Jost*** and ***Thomas Bartscher***

mid August 2020

What can it do?

This version focusses on the 3d animation using `vpython`

It also implements a console interface to control the animation. Overall it can do the following things:

- import .ggb files into vpython 3d points
- save a puzzle imported like that in a new .xml file
- visually define moves by clicking on the points
- automatically calculate the rotation axis for every move and animate it
- load a previously saved puzzle with it's moves from a .xml file

Additions in version 2.1

- scramble the puzzle
- reset the puzzle to solved state
- train an AI using Q-Learning to learn to automatically solve any sufficiently easy puzzle. harder puzzles can be partially solved
- plot the success of the AI during training (basic plot, no legend, acis labels or any explanation)
- solve watch the trained AI solve the puzzle
- control animation speed
- start of an implementation for neural networks to solve more difficult puzzles. However this is not yet connected to the console interface.

fixed issues:

2. the ugly `history_dict` was replaced with a `Twisty_Puzzle` class, which seperates animation and console interface much more cleanly, making the class reusable for future gui implementations.

How does it do that?

Q-Learning

In each episode we start with a solved puzzle and scramble it with `n` random moves. This `n` always starts as 1 and only gets increased if the puzzle was solved in all of the last `30` episodes. Every time the difficulty `n` is increased, ε and `x` get reset to their initial values.

Between each training episode we evaluate whether or not the puzzle was successfully solved. If it was, we *decrease* a variable `x` by `0.2`$\cdot \varepsilon$, otherwise it is *increased* by `0.2`$\cdot (\varepsilon_0-\varepsilon)$. The scaling factors ensure that we don't get too far away from `x=0`.

Once `x` is updated, we calculate the new $\varepsilon = \varepsilon_0 \cdot \sigma($`x`$)$. That way ε either approaches 0 or ε_0. So if the AI is good at solving the puzzle, the

exploration rate gets decreased so it's more likely to reliable solve it.

This two-way adjustment of ε apparently can result in a equilibrium where neither the success of the AI nor the value of ε changes much.

In Addition to the rather complex adjustment of ε between episodes, we also use exponential decay of ε during episodes. That way for new difficulties we explore the first few moves much more than the final moves before a solved state.

Issues

1. Consider a 2-cycle with center of mass `CoM = Puzzle_CoM`. Then the rotation axis described before is not clearly defined. As a real-world example where this happens, consider a 'slice move' on a floppy cube. The same issue may occur with other cuboids.

   It seems like there is no obvious or easy solution to this problem. However it can be solved by defining **one** rotation axis for the entire move. This is tricky with geared puzzles though as there one rotation axis is *not* sufficient to describe all the movement in one move.

2. The .ggb file import still leaves behind ugly unnecessary files: the `.ggb` file renamed to `.zip` (geogebra files are just renamed zips) and the definition file `geogebra.xml`. Both of these are probably not needed or should at least be deleted after the import is done.

new issues in version 2.1

3. The `.txt` files containing the Q-tables can get very large. For example the complete Q-table for a normal skewb - not a very difficult puzzle - would already be 2.5 GB. This size can be halved by deleting all inverse moves, which also speeds up the training.

   This really is an issue because of the way we load an existing Q-table: with the `eval()` command. This quickly leads to a `MemoryError`, crashing the program.

4. There seems to be a new bug introduced for the cycle input, which is really annoying. (most likely not clearing the list of arrows and cycles)

5. Several paramters of the Q-Learning algorithm are not adjustable yet:

   - `reward_dict` - rewards for solving the puzzle and penalties for moves or timeout
   - number of victories before increasing difficulty (currently `30`)
   - starting value for `x` for sigmoid function (currently `0.2`)
   - Δ`x` for winning or losing (currently $\mp$`0.2`$\cdot$`fact`)

Remark: Some puzzles can be reorientated in space using normal moves (like a 2x2 rubiks cube). Therefor they can technically have multiple solved states. However the program can handle only one. This is not really an issue, more like a choice we made.

Dependecies

non-standard library modules:

- vpython

- colored
- lxml

Conclusion

The basic elements used for the file import, export and the animation are sufficiently seperated into files and will most likely be reused for future versions.

Also the interaction methods for the different console commands are a nice baseline and establish what basic functions are required.