Data Science

Teaching Python to Play Chess

August 17th, 2014

Jarret Petrillo, jarret.petrillo@gmail.com

If you can't solve a problem, there is an easier problem you can solve: find it - George Polya

How to get Python to play chess?

Solution 1: Statistical approach

- *Download sci-kit learn*
- *Categorize positions according to king safety, pawn structure, and development*
- *Develop a choice model*
- *Use historical games by grand masters to callibrate model parameters*
- *Too complicated*

Solution 2: Class-based approach

- *Define a chess board as an 8x8 grid*
- *Implement a generic piece class*
- *Subclass the piece class to include rules for how each moves*
- *A chess game would then be a board and a collection of pieces*
- *A move would take one piece and place it somewhere else*
- *Still too complicated*

The bully heurestic

Presentation overview

- *Develop ease of use as a worthy project goal*
- *Describe bully heurestic as one type of solution*
- *Walk through pystockfish as an example of this approach in action*

Has anyone published a package?

Pystockfish takes a package and makes doing one task easier

- *Pystockfish builds off a class from the subprocess package*
- *Subprocess allows python to work with the input/output from a subroutine*
- *Stockfish runs at the terminal level*

Good classes in action

```
from flask import Flask

app = Flask(__name__)

app.config['DEBUG']=True

if __name__ == '__main__':
    app.run()
```

More good classes in action

```
from webapp import db
from models import User

u = User(name="Jarret")

db.session.add(u)
db.session.commit()
```

Ease of use hides complexity

- *Underneath the hood the sqlalchemy session object interacts with SQL*
- *Stores variables in a temporary state*
- *Saves objects to permanent database*
- *Handles synchronous requests*

Ease of use is progress

Question of thinking depth motivated Pystockfish

Pystockfish problem statement

- *Take two instances of the stockfish engine*
- *Set them to different thinking depths*
- *Have them play*
- *Record result*
- *Repeat*

Programming problem statement

- *Create two instances of the stockfish engine*
- *Set parameters*
- *Translate parameters into the language of stockfish*
- *Make a match between the two engines*
- *Have the first engine make a move*
- *Translate move into the language of stockfish and have the second engine move*
- *Repeat until one side wins or game is drawn*
- *Repeat with multiple engines*

How to use stockfish from the terminal

```
>>stockfish
Stockfish 120212 by Tord Romstad, Marco Costalba, and Joona Kiiski
ucinewgame
position startpos moves e2e4 e7e5
go depth 15
info depth 1 seldepth 1 score cp 64 nodes 59 nps 2185 time 27 multipv 1 pv b1c3
info depth 2 seldepth 2 score cp 12 nodes 298 nps 11037 time 27 multipv 1 pv b1c3 b8c6
...
info depth 15 seldepth 22 score cp 56 nodes 1608006 nps 1472532 time 1092 multipv g1f3...
bestmove g1f3 ponder g8f6
```

End use case

```
>>from pystockfish import Engine
>>deepthinker = Engine(depth=15)
>>deepthinker.setposition(['e2e4','e7e5'])
>>deepthinker.bestmove()
'g1f3'
```

Challenges

- *Stockfish understands only UCI protocol - TRANSLATION*
- *Bestmove function is not instantenous - SYNCHRONY*
- *Commands submitted in python but processed in terminal - STATE EQUALITY*

Wrapping subprocess

```
import subprocess

class Engine(subprocess.Popen):

    def __init__(self, depth):
        subprocess.Popen.__init__(self,
            'stockfish',
            universal_newlines=True,
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE)

self.depth = depth
```


Put function

```
def put(self, command):
        self.stdin.write(command+'\n')
```

Isready function and the put/until model

```
def isready(self):
        self.put('isready')
        last_line=''
        while True:
                text=self.stdout.readline().strip()
                if text=='readyok':
                        return lastline
                lastline=text
```

Building more complex functions

```
def newgame(self):
        self.put("ucinewgame")
        self.isready()
```

```
def setposition(self, moves=[]):
        moveString=self._movelisttostr(moves)
        self.put("position startpos moves %s" % moveString)
        self.isready()
```


Bestmove function using the put/until model

```
def bestmove(self):
        last_line = ""
        self.put('go depth %s'%self.depth)
        while True:
                text = self.stdout.readline().strip()
                split_text = text.split(' ')
                if split_text[0]=='bestmove':
                        return {'move': split_text[1],
                        'ponder': split_text[3],
                        'info': last_line
                        }
        last_line = text
```

Publishing your work

If it's not easy to find it doesn't exist

- *PyPI is one-stop pacakge repository*
- *Makes distribution straightforward using setuptools*
- *Allows users to install software with pip command*
- *Creates a weblink that can be shared*

Innovation one small step at a time

- *A problem solved makes room for the next one*
- *Making it easier to do something is an achievement*
- *If it's public others can build off it*

Thank you