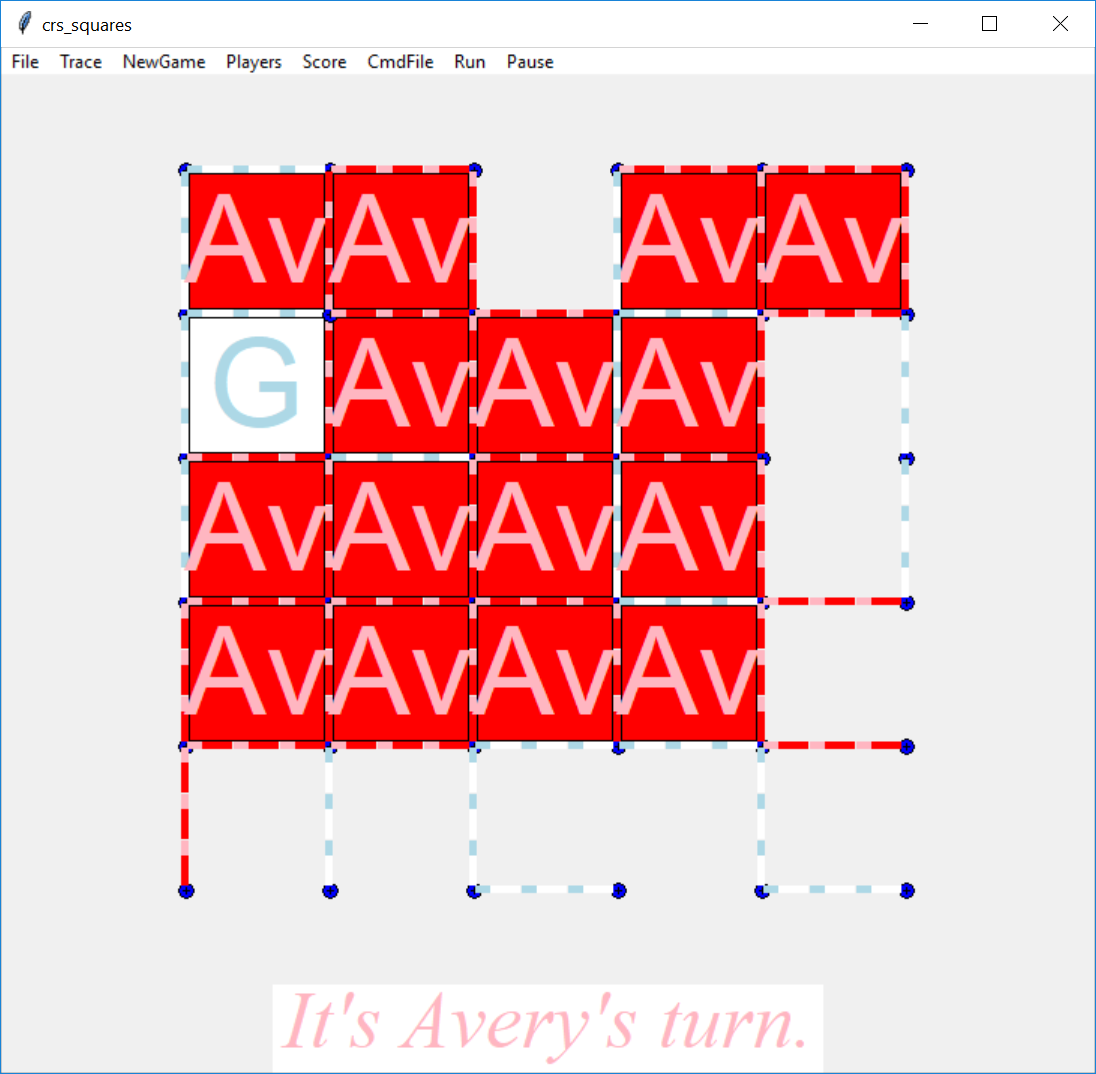
Python Memory Profiling

# The Setting:

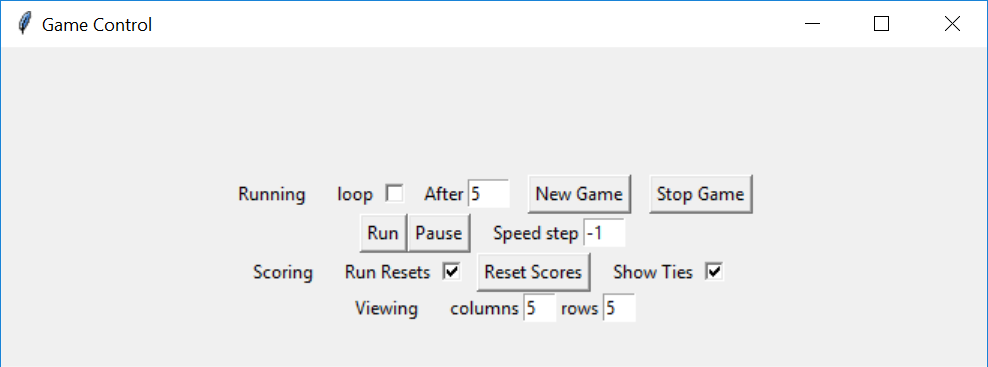
I am in the process of creating graphical games for young children of the ages 3 to 8 years old. While I have used a number of languages (C, C++, Perl, Java, …) over the years, currently I'm using Python. The current project is a graphic game called "Squares" – after creating a rectangle lattice of dots, each player has a turn to connect two adjacent dots creating a line. When such a move completes a square, this square is labeled with the mark of that player, who gets an additional tern. The game has been enhanced to provide automated opponents.

## Demo:

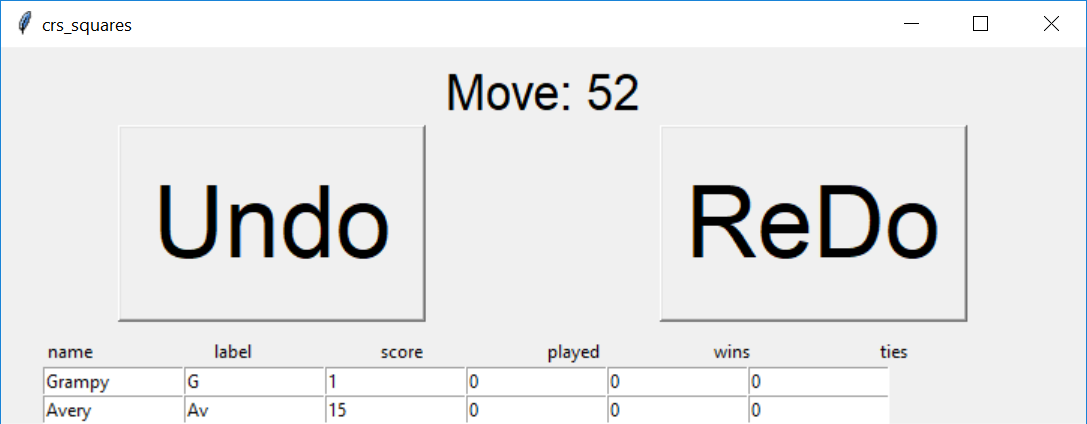
…crs\_squares.py…



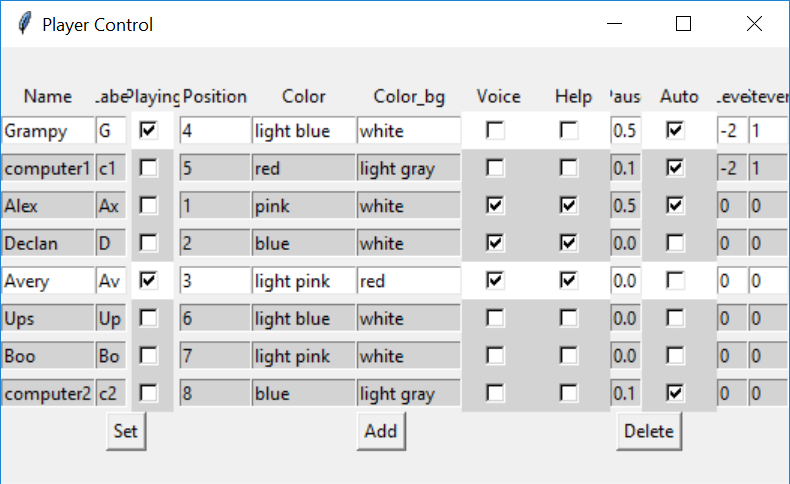
Squares Game In Progress – Main Window



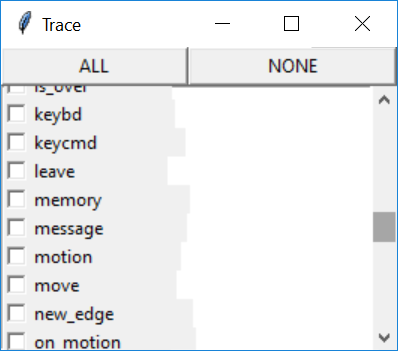
Optional Game Control Window



Optional Scoring Window, with Undo/Redo



Optional Player Selection / Control



Optional Trace/Logging Control - for Debugging

# The Problem Although the program, with some faults, continues to operate without catastrophic error, after some time it "crashes". After some investigation, I discovered this was caused by memory leaks – program grows in size because some resources are not properly released when no longer needed. At the lowest levels the resource is computer memory. In some languages such as C or C++ memory management is mostly the problem of the programmer. While in languages such as Java and Python, the language and the Run-Time-System take a bigger part. But in almost all complicated cases, if not careful, the programmer can get into some problems.

# Beginning to Look

## What to do…

Find out if there is a problem.

Is there a leak?

How big?

## What Tools Are Available?

### With No Program Changes

Windows: Task Manager

Unix/Linux: "ps aux" command in shell loop

Run the program and simultaneously run the memory measurement checking on the memory size.

## Benchmarking Memory Usage / Leaking

Before searching and correcting memory problems we wanted to benchmark the actual memory usage / leakage so that we might be able to better determine and measure any improvements or changes. We added some code to print out the current memory usage.

### Memory Usage Measurement

We used code found in an answer on Stackoverflow.com to determine the current memory usage. The following script is a standalone example.

# memory\_size.py

import os

import psutil

process = psutil.Process(os.getpid())

print(process.memory\_info().rss) # in bytes

To track memory growth we recorded the memory usage change between game loops. The following is an abbreviated sample of the game program log:

20190318\_101744 Creating Log File Name: C:\users\raysm\workspace\python\imageSelect\log\crs\_squares\_20190318\_101744.sllog

…

20190318\_101744 crs\_squares.py

…

20190318\_101744 args: … nx=5, ny=5,

Loop 1

20190318\_101744 Memory Used: 78 MB, Change: 78.49 MB

…

Loop 2

20190318\_101746 Memory Used: 83 MB, Change: 4.58 MB

…

Loop 3

20190318\_101748 Memory Used: 84 MB, Change: 1.36 MB

20190318\_101748 delete board\_canvas

20190318\_101748 PlayerControl

20190318\_101750 end of game

20190318\_101750 Restarting game after 0 seconds

Looking at the effect of grid size on memory growth:

Scanning the augmented game program's log file

Using perl from the Cygwin Unix-like tool kit available for Windows.

raysm@DESKTOP-NCFAM48 /cygdrive/c/users/raysm/workspace/python/imageSelect/log

$ perl -ane 'if (/Memory Used/) { print;} if (/nx=(\d+)/) {print;print("\n$1\n")}' $(ls -rt| tail -4)

20190317\_162039 args: Namespace(btmove=1.0, ew\_display=3, ew\_select=5, ew\_standoff=5, height=600, loop=True, loop\_after=0.0, min\_xlen=10, nx=2, ny=2, run\_game=True, show\_id=False, show\_players=True, show\_score=True, speed\_step=-1, stroke\_move=True, trace='memory', width=600)

2

20190317\_162039 Memory Used: 111 MB, Change: 111.11 MB

20190317\_162050 Memory Used: 260 MB, Change: 148.92 MB

20190317\_162106 Memory Used: 247 MB, Change: -13.25 MB

20190317\_162123 Memory Used: 258 MB, Change: 11.52 MB

20190317\_162139 Memory Used: 268 MB, Change: 9.96 MB

20190317\_162154 Memory Used: 274 MB, Change: 5.91 MB

20190317\_162211 Memory Used: 281 MB, Change: 6.97 MB

20190317\_162227 Memory Used: 291 MB, Change: 9.78 MB

20190317\_162242 Memory Used: 295 MB, Change: 3.99 MB

20190317\_162257 Memory Used: 300 MB, Change: 4.66 MB

20190317\_162312 Memory Used: 307 MB, Change: 7.31 MB

20190317\_162327 Memory Used: 314 MB, Change: 7.01 MB

20190317\_162343 Memory Used: 323 MB, Change: 9.36 MB

20190317\_162358 Memory Used: 328 MB, Change: 5.21 MB

20190317\_162414 Memory Used: 337 MB, Change: 8.75 MB

20190317\_162429 Memory Used: 339 MB, Change: 1.56 MB

20190317\_162447 Memory Used: 344 MB, Change: 5.40 MB

20190317\_162502 Memory Used: 350 MB, Change: 5.51 MB

20190317\_162517 Memory Used: 354 MB, Change: 4.67 MB

20190317\_162533 Memory Used: 357 MB, Change: 2.84 MB

20190317\_162549 Memory Used: 366 MB, Change: 8.61 MB

20190317\_162604 Memory Used: 366 MB, Change: 0.70 MB

20190317\_162621 Memory Used: 375 MB, Change: 8.90 MB

20190317\_162638 Memory Used: 376 MB, Change: 0.74 MB

20190317\_162838 args: Namespace(btmove=1.0, ew\_display=3, ew\_select=5, ew\_standoff=5, height=600, loop=True, loop\_after=0.0, min\_xlen=10, nx=5, ny=5, run\_game=True, show\_id=False, show\_players=True, show\_score=True, speed\_step=-1, stroke\_move=True, trace='memory', width=600)

5

20190317\_162838 Memory Used: 111 MB, Change: 110.99 MB

20190317\_162848 Memory Used: 263 MB, Change: 151.54 MB

20190317\_162905 Memory Used: 252 MB, Change: -10.44 MB

20190317\_162922 Memory Used: 264 MB, Change: 11.41 MB

20190317\_162938 Memory Used: 283 MB, Change: 19.47 MB

20190317\_163233 args: Namespace(btmove=1.0, ew\_display=3, ew\_select=5, ew\_standoff=5, height=600, loop=True, loop\_after=0.0, min\_xlen=10, nx=5, ny=5, run\_game=True, show\_id=False, show\_players=True, show\_score=True, speed\_step=-1, stroke\_move=True, trace='memory', width=600)

5

20190317\_163233 Memory Used: 111 MB, Change: 110.86 MB

20190317\_163243 Memory Used: 262 MB, Change: 151.41 MB

20190317\_163259 Memory Used: 252 MB, Change: -10.75 MB

20190317\_163316 Memory Used: 266 MB, Change: 14.84 MB

20190317\_163333 Memory Used: 277 MB, Change: 10.49 MB

20190317\_163349 Memory Used: 293 MB, Change: 16.52 MB

20190317\_163406 Memory Used: 304 MB, Change: 10.33 MB

20190317\_163422 Memory Used: 314 MB, Change: 10.59 MB

20190317\_163439 Memory Used: 327 MB, Change: 12.60 MB

20190317\_163455 Memory Used: 336 MB, Change: 9.00 MB

20190317\_163513 Memory Used: 347 MB, Change: 11.56 MB

20190317\_163531 Memory Used: 357 MB, Change: 9.65 MB

20190317\_163548 Memory Used: 386 MB, Change: 28.94 MB

20190317\_163607 Memory Used: 395 MB, Change: 9.35 MB

20190317\_163901 args: Namespace(btmove=1.0, ew\_display=3, ew\_select=5, ew\_standoff=5, height=600, loop=True, loop\_after=0.0, min\_xlen=10, nx=15, ny=15, run\_game=True, show\_id=False, show\_players=True, show\_score=True, speed\_step=-1, stroke\_move=True, trace='memory', width=600)

15

20190317\_163901 Memory Used: 111 MB, Change: 111.13 MB

20190317\_163922 Memory Used: 238 MB, Change: 126.57 MB

20190317\_164002 Memory Used: 294 MB, Change: 56.79 MB

20190317\_164106 Memory Used: 349 MB, Change: 54.26 MB

20190317\_164211 Memory Used: 393 MB, Change: 44.03 MB

20190317\_164310 Memory Used: 456 MB, Change: 62.78 MB

20190317\_164350 Memory Used: 495 MB, Change: 39.29 MB

20190317\_164431 Memory Used: 538 MB, Change: 43.47 MB

20190317\_164510 Memory Used: 583 MB, Change: 45.13 MB

20190317\_164547 Memory Used: 627 MB, Change: 43.41 MB

20190317\_164630 Memory Used: 703 MB, Change: 76.64 MB

While a bit erratic the growth appears, after the first couple of loops, closely affected by the grid size.

# Finding the Causes

Memory Leaks, like performance (slowness) problems, are often more difficult than functional problems because memory usage and performance are invisibly spread throughout the program. While one can often trace through the program in a linear fashion to find a functional problem, the same is often very difficult to do for a memory problem. This invites a search for helpful tools. The following are some tools that I have found helpful.

## memory-profiler 0.55.0

**installation**

Install via pip:

$ pip install -U memory\_profiler

NOTE: Be sure to install for the Python you intend to use.

The following python script can be used to install Python modules.

# install\_module.py

*""" Installing a python module*

*"""*

from subprocess import call

import sys

import os

exe\_dir = os.path.dirname(sys.executable)

print(*"Python exe dir"*, exe\_dir)

pip\_exe = os.path.join(exe\_dir, *"Scripts"*, *"pip.exe"*)

print(*"Pip exe:"*, pip\_exe)

module = *"*memory\_profiler*"* # Default module

md=input(*'Name module:[%s]'* % module)

if md != *""*:

module = md

print(*"Installing"*, module)

try:

call([pip\_exe, *'install'*, module])

except Exception:

print(*'Error'*)

print(*"End of install"*)

## Profiling over Time

### Generating the data

PS C:\users\raysm\workspace\python\imageSelect\src> mprof run crs\_squares.py

mprof: Sampling memory every 0.1s

running as a Python program...

Creating Log File Name: C:\users\raysm\workspace\python\imageSelect\log\crs\_squares\_20190318\_101744.sllog

loadTraceFlags: button,add\_part,display,part\_info,enter,region\_rect,is\_over,show\_id,highlight,leave,stroke,turning\_on,new\_edge,square\_completion,get\_parts,motion,down,complete\_square,get\_color,regno,destroy frame,on\_motion,controls,in\_stroke,execute,verbose,execute\_print,execute\_select,execute\_stack,full\_stack,full\_undo\_stack,complete\_test,show\_move,completed\_square,annotate\_square,keycmd,part\_info\_over,execute\_part\_change,show\_move\_print,blink,track\_move\_edge,keybd,execute\_keybd\_edge\_change,execute\_edge\_change,dbg,execute\_undo\_stack,selected,deepcopy,part\_check\_ok,select\_copy,copy,play\_strategy,after\_move,trace\_scores,player\_trace,resize\_window,score,pgm\_stack,player,move,square,message,pgm\_stack\_list,memory

crs\_squares.py

args: Namespace(btmove=1.0, ew\_display=3, ew\_select=5, ew\_standoff=5, height=600, loop=False, loop\_after=5, min\_xlen=10, nx=5, ny=5, run\_game=True, show\_id=False, show\_players=True, show\_score=True, speed\_step=-1, stroke\_move=False, trace='', width=600)

Loop 1

Memory Used: 78 MB, Change: 78.49 MB

reset\_score

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 2

Memory Used: 83 MB, Change: 4.58 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 3

Memory Used: 84 MB, Change: 1.36 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 4

Memory Used: 86 MB, Change: 1.25 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 5

Memory Used: 87 MB, Change: 1.07 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 6

Memory Used: 88 MB, Change: 1.48 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 7

Memory Used: 89 MB, Change: 1.17 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 8

Memory Used: 91 MB, Change: 1.29 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 9

Memory Used: 92 MB, Change: 1.24 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 10

Memory Used: 93 MB, Change: 1.11 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 11

Memory Used: 94 MB, Change: 1.17 MB

delete board\_canvas

PlayerControl

end of game

Restarting game after 0 seconds

Starting New Game

Loop 12

Memory Used: 95 MB, Change: 1.21 MB

delete board\_canvas

PlayerControl

Closing windows

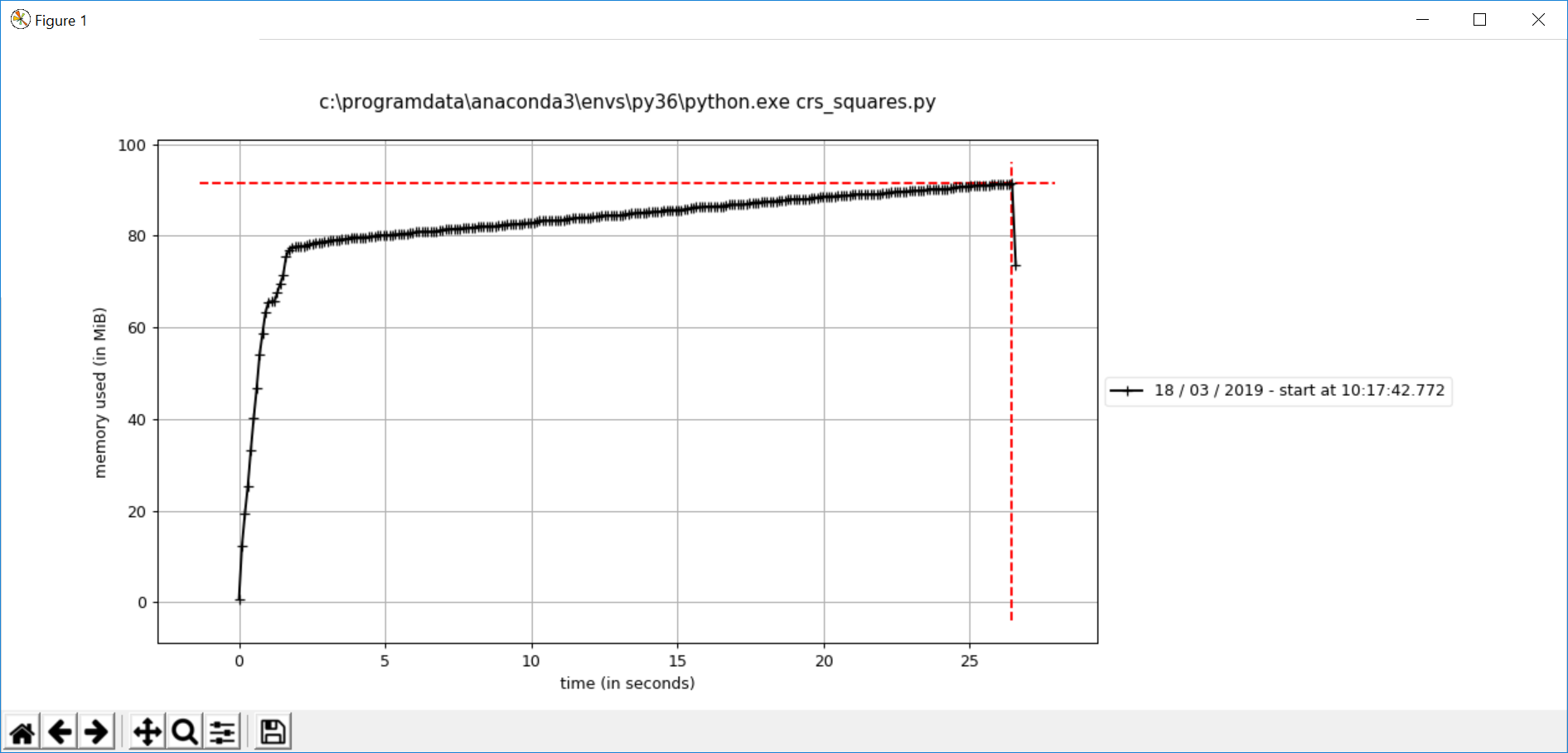
Saving properties file C:\users\raysm\workspace\python\imageSelect\crs\_squares.properties

Closing log file C:\users\raysm\workspace\python\imageSelect\log\crs\_squares\_20190318\_101744.sllog

PS C:\users\raysm\workspace\python\imageSelect\src>

### Ploting Memory Usage over time

mprof plot



## [tracemalloc](https://docs.python.org/3/library/tracemalloc.html#module-tracemalloc) — Trace memory allocations

This is a tool which can facilitating the pinpointing the areas of memory usage.

URL:

<https://docs.python.org/3/library/tracemalloc.html#module-tracemalloc>

### Synopsis of my code:

1. The function "set\_squares\_button" gets called at the beginning of each game loop.
2. select\_trace.pl/SlTrace is my tracing/logging module/class.
3. SlTrace.trace() returns True if tracing this flag
4. SlTrace.lg logs a string with a time stamp.
5. SlTrace.getMemory returns current memory usage in bytes.
6. SlTrace.getMemory returns memory change.
7. tracemalloc.take\_snapshot() records Python memory usage.
8. snapshot2.compare\_to() compares two memory usage snapshots.

import tracemalloc

snapshot1 = None

snapshot2 = None

def **set\_squares\_button**():

global loop\_no

global snapshot1, snapshot2 # tracemalloc instances

loop\_no += 1

SlTrace.lg(*"\nLoop %d"* % loop\_no)

SlTrace.lg(*"Memory Used: %.0f MB, Change: %.2f MB"*

% (SlTrace.getMemory()/1.e6, SlTrace.getMemoryChange()/1.e6))

if SlTrace.trace(*"memory"*):

if snapshot1 is None and snapshot2 is None:

snapshot1 = tracemalloc.take\_snapshot()

elif snapshot2 is None:

snapshot2 = tracemalloc.take\_snapshot()

else:

snapshot1 = snapshot2

snapshot2 = tracemalloc.take\_snapshot()

if snapshot2 is not None:

top\_stats = snapshot2.compare\_to(snapshot1, *'lineno'*, True)

SlTrace.lg(*"[ Top 25 differences]"*)

for stat in top\_stats[:25]:

SlTrace.lg(str(stat))

snapshot1 = snapshot2

snapshot2 = None

#### Sample Output

20190317\_182031 [ Top 25 differences]

20190317\_182031 C:\ProgramData\Anaconda3\envs\py36\lib\site-packages\objgraph.py:447: size=5408 KiB (+341 KiB), count=173048 (+10910), average=32 B

20190317\_182031 … C:\Users\raysm\workspace\python\imageSelect\src\select\_command.py:37: size=124 KiB (+61.9 KiB), count=430 (+213), average=296 B

…

**Abbreviating NNNNNN\_NNNNNN C:\Users\raysm\workspace\python\imageSelect\src to src**

**Our source files, line numbers in bold)**

src\**select\_squares.py:405**: size=67.7 KiB (+33.9 KiB), count=340 (+170), average=204 B

C:\ProgramData\Anaconda3\envs\py36\lib\tracemalloc.py:180: size=71.4 KiB (+27.5 KiB), count=1305 (+503), average=56 B

src\**select\_edge.py:193**: size=51.0 KiB (+25.5 KiB), count=989 (+494), average=53 B

src\**select\_command\_play.py:134**: size=47.8 KiB (+24.2 KiB), count=437 (+220), average=112 B

src\**select\_loc.py:53**: size=48.0 KiB (+15.9 KiB), count=959 (+318), average=51 B

src\**select\_part.py:343**: size=33.9 KiB (+12.7 KiB), count=543 (+203), average=64 B

src\**select\_part.py:297**: size=63.0 KiB (+11.8 KiB), count=192 (+36), average=336 B

…

src\**select\_message.py:11**: size=20.9 KiB (+10.5 KiB), count=276 (+139), average=78 B

src\**select\_command\_play.py:296**: size=20.5 KiB (+10.3 KiB), count=328 (+165), average=64 B

20190317\_182033 Closing windows

20190317\_182033 Saving properties file C:\Users\raysm\workspace\python\imageSelect\crs\_squares.properties

20190317\_182033 Closing log file C:\Users\raysm\workspace\python\imageSelect\log\crs\_squares\_20190317\_182020.sllog

#### Looking at the code (snippets at allocation line)

src\select\_squares.py:405: size=67.7 KiB (+33.9 KiB), count=340 (+170), average=204 B

pt.\_\_dict\_\_ = part.\_\_dict\_\_.copy()

src\select\_edge.py:193: size=51.0 KiB (+25.5 KiB), count=989 (+494), average=53 B

multi\_tags = [[], [], []] # Multiple sets for ripple

src\select\_command\_play.py:134: size=47.8 KiB (+24.2 KiB), count=437 (+220), average=112 B

part\_by\_id = {}

src\select\_loc.py:53: size=48.0 KiB (+15.9 KiB), count=959 (+318), average=51 B

*self*.coord = [(c1x,c1y), (c3x,c3y)]

src\select\_part.py:343: size=33.9 KiB (+12.7 KiB), count=543 (+203), average=64 B

*self*.text\_tags = [] # appended texts if any

src\select\_part.py:297: size=63.0 KiB (+11.8 KiB), count=192 (+36), average=336 B

*self*.sel\_area = sel\_area

src\select\_message.py:11: size=20.9 KiB (+10.5 KiB), count=276 (+139), average=78 B

*self*.text = text

src\select\_command\_play.py:296: size=20.5 KiB (+10.3 KiB), count=328 (+165), average=64 B

prev\_selects = list(*self*.prev\_selects.values())

# Larger Picture - More Encompassing Changes

Rather than address directly these code snippets, can we make more sweeping changes to reduce memory leaks?

Reusing control windows

Reusing game objects such as the squares board