# dc3bd04c-6094-4628-aadf-94a356999c34 | CMU CS Academy

```
#!/usr/bin/env python3
#-*-coding: utf-8 -*-

"""
#---------------------------------------------------------------------------------#
Evaluation of Computational Capability of CMU CS Academy in 2d Vector and Raster
Graphics Workloads
Created by S Tessar for AP CSP
Reviewed by K Schoeberlein
##2023##
#---------------------------------------------------------------------------------#
"""

"""
#---------------------------------------------------------------------------------#
##LICENSE##
This program is free software. you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation, either version 3 of the License, or (at your option) any later
version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY
WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNES FOR A
PARTICULAR PURPOSE. See the GNU General Public license for more details.
You should have received a copy of the GNU General Public License along with
this program. If not, see <https://www.gnu.org/licenses>
#---------------------------------------------------------------------------------#
"""

"""
#---------------------------------------------------------------------------------#
##INFORMATION##
This program is a test of the computational ability of CMU CS Academy's ide in
2 dimensional vector and raster graphics.
For the performance metrics, this program creates a test set of 2 dimensional
objects repeatedly, while simultaneously updating a count with the amount of
objects being rendered by CMU. The test
set was seected due to including regular shapes that are rendered as vector
images, in addition to text, which based off of looking into network requests,
re standard bitmap raster fonts from fonts.google.com
For a standard example in Firefox on a Windows 10 PC with a 12th gen Intel core
i7, 16 Gigabytes of ram and an Nvidia T1000 with 4 GB of GDDR6. Across 10 runs
this computer achieved an average of 5423 shapes after around 5 minutes each,
the time it took the page becomes unresponsive and eventually crash either
Firefox or the system. In Google Chrome on the same system however, this code
achieves much less consistent scores with an average across 5 runs of 1736,
but massive run to run variance. The individual scores in Chrome were 1041,
2103, 4332, and 1202. Due to this variance in my data set, I believe that this
test is not suitable for Chromium based browsers.
For performance logging from Firefox's built in profiler, here is a capture of
one of the runs:

   <https://go.poweredge.xyz/perf/>

As far as I can tell based off of some reverse engineering of how CMU CS Academy
runs Python and renders shapes, a new process of
/static/js/cmu-graphics.4da25947.js is created along with every shape. The
reason that CMU has done it this way is most likely to simplify development, as
CMU CS academy uses Brython for running the code in a python 3 like environment,
and Ace Editor as the code editor. While Brython supports outputting to a
display, doing so requires specific JS frameworks that support doing so, and due
to CMU's use of Ace, it seems that none of these frameworks would be compatible.
To avoid this, it seems they wrote a graphic library that renders direct to the
webpage, but did not make it able to handle multiple shapes at once. To get
around this, upon a shape or a text label being created, CMU CS Academy spawns
a new process of their JS graphics library.
To get around CMU's paste restrictions for the license and much of the
whitepaper, I created a basic program using pynput that can be found here:
   <https://go.poweredge.xyz/paste>
To read the condensed whitepaper, go to
   <https://cmu.poweredge.xyz>
or <https://cmu.adas.software>
#------------------------------------------------------------------------------#
"""

"""
#------------------------------------------------------------------------------#
Source for the website is available at https://github.com/cmdada/cmuct
(the site has no backend so it's also available with inspect element.)
#------------------------------------------------------------------------------#
"""

__name__ = "Evaluation of Computational Capability of CMU CS Academy"
__license__ = "GPL v3"
__version__ = "1.05"
__status__ = "Production"
__lastEdited__='11/15/2023'
__url__ = "https://cmu.adas.software/"

print("#"+__name__+"#")
print()
print('#'+'='*56+'#')
print('License: ' +__license__)
print('Version: ' +__version__)
print('Status: ' +__status__)
print('Last edited ' +__lastEdited__)
print('More info at ' +__url__)
print('#'+'='*56+'#')
print()
print('Click on the canvas to start or pause.')

##CODE##
#In trying to embed this into a site, the "resume" button is hidden, so a
#bandaid fix is to have this run on a click
app.started=False

#Program Info and UX (not really my best skill)#
```

```python
title=Label('Browser stress test', 200, 20, size=20, bold=True)
infoText=Label('Shapes created before crash. Click on the canvas to start or pause',
200, 33, size=10)

#This snippet of code makes CMU ignore that above 2000 shapes are being used.
#The number was derived from attempting larger numbers and iterating down to
#find the highest integer it would accept in a creative task, and while a
#smaller number could be used, this removes possible bottlenecking in test
#results
app.setMaxShapeCount(18446744073709556)

#this defines the speed that the app runs at, and initializes a global variable
#that the program uses to display the amount of shapes. While in practice CMU CS
#Academy never gets near this speed, this removes a possbile bottleneck.
app.shapes=0
app.stepsPerSecond = 100000

#The increasing points on the stars the later function creates visualize to the
#user the amount of shapes and creates an interesting visual to watch while
#running tests.

app.starpoints=2

#app.paused=True (with the new "click on the canvas to start or pause", the old
#implementation of having the user press the cmu resume button has been removed
#due to the button not being rendered on the embedded or shared version of the
#project)
def perfTest():

    #Same way of deriving this number as before. Definitely excessive but good
    #for testing purposes to remove a possible bottleneck.
    if -1<app.shapes<18446744073709555:

        #This part of the code is for outputting the amount of shapes to the
        #user, and visualizing them with the star. It increases by one every
        #time this function is run.
        app.shapes=app.shapes+1
        app.starpoints=app.starpoints+1

        #This is the part of the function that actually loades the system by
        #creating the 2d shapes repeatedly (theoretically) at the speed set in
        #stepsPerSecond.
        cover = Rect(0,40,400,360, fill='white')
        starIndicator=Star(200,200,10,app.starpoints)
        shapesNum=Label((app.shapes*3)+2 , 200, 150, )

    elif app.shapes>18446744073709555:

        #this part is more of a joke and a way to use an elif
        print('You have 1.660276e+18 Bytes of memory????')
        shapesNum.value='You have 1.660276e+18 MB of memory????'
    else:
        #Error handling in case something weird happens, this was originally
        #being called by CMU CS Academy artificially inflating the shape count
        #on lag, but hasn't been caused since the new implementation of the
        #shape counter using app.stepsPerSecond.
        print('Oh No, It seems something went wrong. This code should be stopped by CMU before getting anywhere near this point and overflowing.')
        title.value='Oh No, It seems something went wrong. This code should be stopped by CMU before getting anywhere near this point and overflowing.'
#The mouseX, mouseY are dummy arguments being passed into it, even if I don't
#use them in my code
def onMousePress(mouseX,mouseY):
    #Conditionals to handle starting/pausing on mouse click
    if app.started ==  True:
        app.started =  False

    else:
        app.started =  True

#CMU CS Academy attempts to repeat this at the given step speed, defined in
#app.stepsPerSecond
def onStep():

    if app.started==True:
    #Checking if the mouse has been clicked using the global variable defined
    #earlier, no parameters are needed.
        perfTest()
```