Ryan Phillips

UMKC CS449

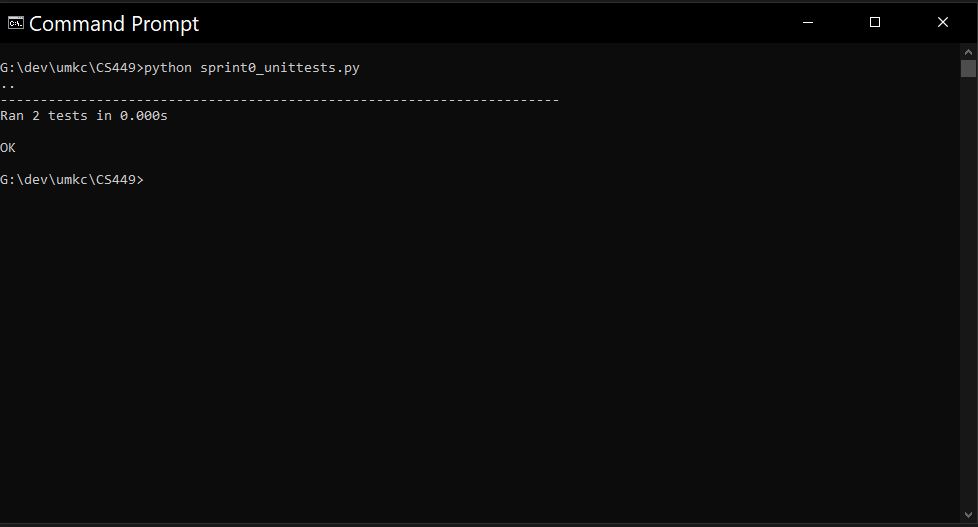
Sprint #0 Report

GitHub Repository: <https://github.com/RyanPhillipsUMKC/CS449>

1. Key Decisions of the SOS Project  
Object-oriented programming language: **Python**  
GUI library (strongly encouraged): **Tkinter**  
IDE (Integrated Development Environment): **Visual Studio**  
xUnit framework (e.g., JUnit for Java): **Python UnitTest**  
Programming style guide (must read it carefully): **Google Python Style Guide**  
Project hosting site Github.com: **Github.com**  
Other decisions if applicable: **N/A**

Code for requirements 2 and 3 can be found on <https://github.com/RyanPhillipsUMKC/CS449/tree/main/sprint0>. However, the direct source code is pasted below as well.  
  
2. Unit testing (4 points)

Output:



Source Code:

'''

Ryan Phillips

UMKC CS 449 Sprint 0 Unit Tests

Languag: Python

Unit Test Framework: unittest (pythons version of java JUnit)

sprint0\_unittests.py

'''

import unittest

'''

I use a basic trial division prime generator that I created for my part of CS 404 prject

to test on.

'''

import time

from math import sqrt

'''

Trial Division Prime Check

This is good for one off checks because it wont allocate any memory.

However, when generating primes dont use this check. It will be slower because of the absent use of a prevuious found primes cache.

trial\_division\_generator() below uses a previous found prime cache and prime gap to genrate much faster checks

Time Complexity = O(sqrt(N))

Space Complexity = O(1)

'''

def trial\_division\_check(number):

# Passed invalid argument

if not isinstance(number, (int, float)):

return False

# Number passed is non positive

if number <= 0:

return False

# one is a special case and is not prime because has only one factor

if number == 1:

return False

# sqrt of number is hard constraint for primality factorization

square\_root = int(sqrt(number))

# Check for any valid factors starting from 2 going to square root (Inclusive)

# Iff a valid factor is found then the number is not prime

# linear time complexity

for denominator in range(2, square\_root + 1): # -> O(sqrt(N))

if (number % denominator) == 0:

return False # Factor found -> Not Prime

# No factors found or square root was less than or equal to 1 -> Is Prime

return True

'''

Units tests for prime generator

'''

class TestPrimeGeneratorFunction(unittest.TestCase):

def test\_single\_prime(self):

self.assertEqual(trial\_division\_check(7), True)

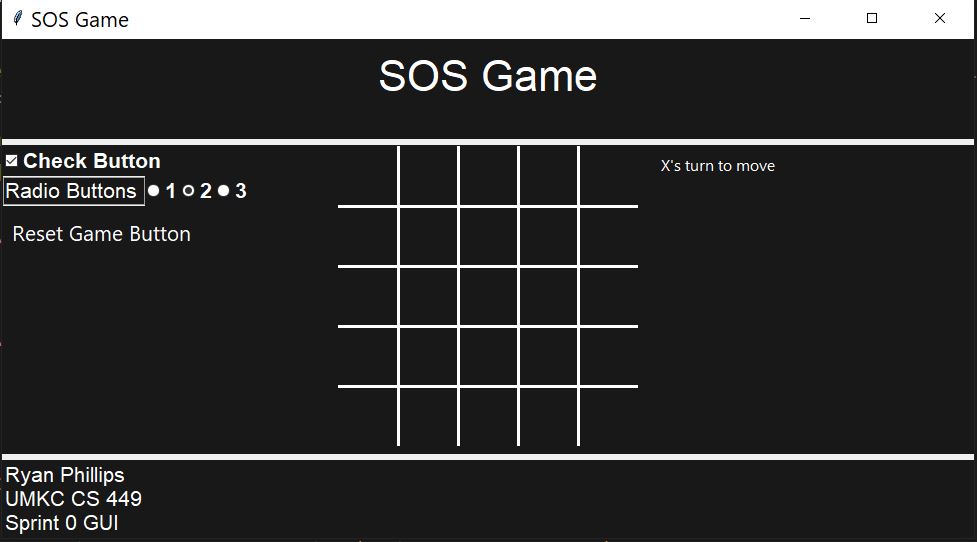
def test\_multiple\_primes(self):

self.assertEqual([trial\_division\_check(3), trial\_division\_check(7), trial\_division\_check(6)], [True, True, False])

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

3. GUI programming (4 points)  
Output GUI:



Source Code:

'''

Ryan Phillips

UMKC CS 449 Sprint 0 GUI

Languag: Python

GUI Framework: TKinter

sprint0\_gui.py

'''

import tkinter as tk

from tkinter import ttk

'''

For now this is just a basic GUI to mock how to SOS game GUI will look

There is no state handeling in this yet

'''

class App(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Setup Main App Config

self.title('SOS Game')

# default to half user screen size and allow resizing and sacling to an adjustable window

screen\_width = self.winfo\_screenwidth()

screen\_height = self.winfo\_screenheight()

self.geometry(f'{int(screen\_width \* 0.5)}x{int(screen\_height \* 0.5)}')

self.resizable(1, 1)

self.columnconfigure(0, weight=1)

self.rowconfigure(0, weight=1)

# configure styles

self.bg\_color = "#181818"

self.board\_cell\_size = 60

self.board\_line\_width = 3

self.turn\_text = ("X's turn to move", "O's turn to move")

self.style = ttk.Style(self)

self.style.configure('TLabel', font=('Helvetica', 22))

self.style.configure('TEntry', font=('Helvetica', 32))

self.style.configure('Heading.TLabel', background=self.bg\_color, foreground="white", font=('Helvetica', 32))

self.style.configure('FooterInfo.TLabel', background=self.bg\_color, foreground="white", font=('Helvetica', 16))

self.style.configure("TRadiobutton", font=('Helvetica', 16, 'bold'), foreground='white', background=self.bg\_color)

self.style.configure("TCheckbutton", font=('Helvetica', 16, 'bold'), foreground='white', background=self.bg\_color)

# main parent frame

self.mainframe = ttk.Frame(self, style="TFrame")

self.mainframe.grid(column=0, row=0, sticky="nsew")

self.mainframe.rowconfigure(0, weight=0)

self.mainframe.rowconfigure(1, weight=1)

self.mainframe.rowconfigure(2, weight=0)

self.mainframe.columnconfigure(0, weight=1)

# header frame

self.header\_frame = tk.Frame(self.mainframe, bg=self.bg\_color, bd=1, height=100)

self.header\_frame.grid(column=0, row=0, sticky="nsew")

self.header\_frame.pack\_propagate(False)

self.heading = ttk.Label(self.header\_frame, text='SOS Game', style='Heading.TLabel').pack(pady=10)

# main content game frame

self.main\_content\_frame = tk.Frame(self.mainframe)

self.main\_content\_frame.grid(row=1, column=0, sticky="nsew", pady=6)

self.main\_content\_frame.rowconfigure(0, weight=1)

self.main\_content\_frame.columnconfigure([0, 1 ,2], weight=1, uniform="cols")

# left side game frame

self.leftside\_frame = tk.Frame(self.main\_content\_frame, bg=self.bg\_color, bd=1)

self.leftside\_frame.grid(column=0, row=0, sticky="nsew")

self.checkbox\_value = tk.BooleanVar()

self.checkbox = ttk.Checkbutton(self.leftside\_frame, text="Check Button", variable=self.checkbox\_value, takefocus=0)

self.checkbox.grid(column=0, row=0, sticky="w")

self.radio\_button\_frame = ttk.Frame(self.leftside\_frame)

self.radio\_button\_frame.grid(row=1, column=0, sticky="w")

self.radio\_button\_text = ttk.Label(self.radio\_button\_frame, text="Radio Buttons ", style="FooterInfo.TLabel", relief="solid")

self.radio\_button\_text.grid(column=0, row=1, sticky="w")

self.radio\_button\_selected = tk.StringVar()

self.radio\_button1 = ttk.Radiobutton(self.radio\_button\_frame, text='1', value='1', variable=self.radio\_button\_selected, takefocus=0, style="TRadiobutton")

self.radio\_button1.grid(column=1, row=1, sticky="w")

self.radio\_button2 = ttk.Radiobutton(self.radio\_button\_frame, text='2', value='2', variable=self.radio\_button\_selected, takefocus=0, style="TRadiobutton")

self.radio\_button2.grid(column=2, row=1, sticky="w")

self.radio\_button3 = ttk.Radiobutton(self.radio\_button\_frame, text='3', value='3', variable=self.radio\_button\_selected, takefocus=0, style="TRadiobutton")

self.radio\_button3.grid(column=3, row=1, sticky="w")

self.reset\_button = tk.Button(self.leftside\_frame, text="Reset Game Button", command=self.reset\_board, background=self.bg\_color, foreground="White", relief="flat")

self.reset\_button.grid(row=2, column=0, sticky="w")

# center game frame / main board frame

self.middle\_frame = tk.Frame(self.main\_content\_frame, bg=self.bg\_color, bd=1)

self.middle\_frame.grid(column=1, row=0, sticky="nsew")

# right side game frame

self.rightside\_frame = tk.Frame(self.main\_content\_frame, bg=self.bg\_color, bd=1)

self.rightside\_frame.grid(column=2, row=0, sticky="nsew")

# footer frame

self.footer\_frame = tk.Frame(self.mainframe, bg=self.bg\_color, bd=1, height=75)

self.footer\_frame.grid(column=0, row=2, sticky="nsew")

self.footer\_frame.pack\_propagate(False)

self.footer\_info = ttk.Label(self.footer\_frame, text="Ryan Phillips\nUMKC CS 449\nSprint 0 GUI", style="FooterInfo.TLabel")

self.footer\_info.grid(row=0, column=0)

# TODO: Adjust board size from this entry value

self.board\_size = tk.IntVar(value=5)

#self.board\_size\_entry = ttk.Entry(

# self.mainframe, textvariable=self.board\_size, style="TEntry", font=self.style.lookup("TEntry", "font"))

board\_size = self.get\_total\_board\_draw\_size()

self.board\_canvas = tk.Canvas(self.middle\_frame, width=board\_size, height=board\_size, bg=self.bg\_color, highlightthickness=0)

self.board\_canvas.pack()

#self.board\_canvas.grid(row=0, column=0)

self.board\_canvas.bind("<Button-1>", self.on\_board\_cell\_click)

self.status = ttk.Label(self.rightside\_frame, text=self.turn\_text[0], font=("Segoe UI", 12), style="FooterInfo.TLabel")

self.status.grid(row=4, column=0, sticky="w", padx=8, pady=(6, 10))

self.reset\_board()

# on game window adjustment

self.bind("<Configure>", self.on\_configure)

def get\_total\_board\_draw\_size(self):

return self.board\_cell\_size \* self.board\_size.get()

def reset\_board(self):

self.status.config(text=self.turn\_text[0])

self.draw\_board()

def on\_board\_cell\_click(self, event):

print(f"CLicked on board cell {event}")

pass

# Draw the board lines on the canavs

def draw\_board(self):

self.board\_canvas.delete("all")

total\_board\_draw\_size = self.get\_total\_board\_draw\_size()

# Vertical lines

for i in range(1, self.board\_size.get()):

x = i \* self.board\_cell\_size

self.board\_canvas.create\_line(x, 0, x, total\_board\_draw\_size, width=self.board\_line\_width, fill="White")

# Horizontal lines

for i in range(1, self.board\_size.get()):

y = i \* self.board\_cell\_size

self.board\_canvas.create\_line(0, y, total\_board\_draw\_size, y, width=self.board\_line\_width, fill="White")

def on\_configure(self, event):

print(f"on\_configure {event}")

pass

# run main app loop

if \_\_name\_\_ == "\_\_main\_\_":

app = App()

app.mainloop()