Russell Schwartz
Maker Profile

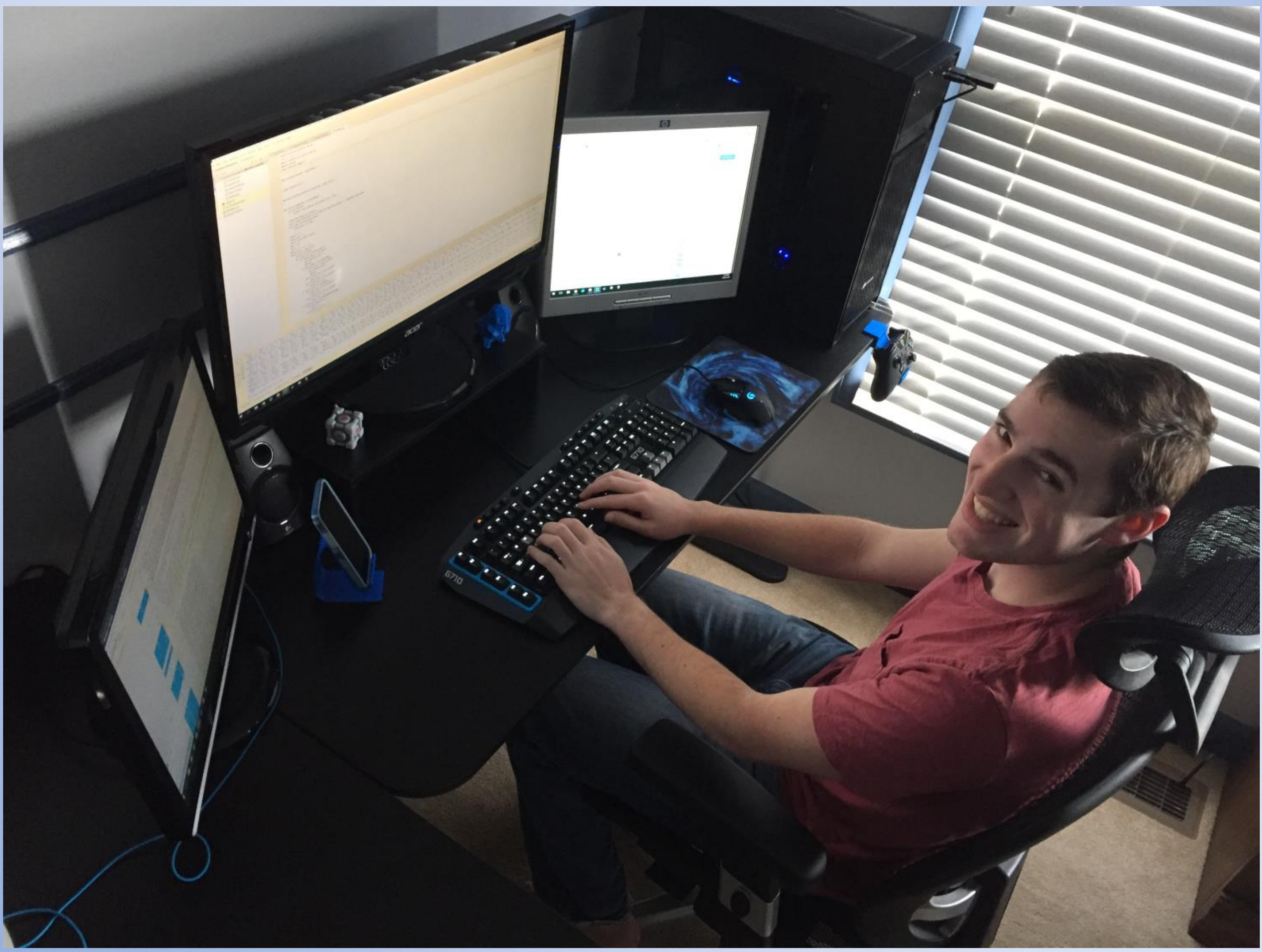


TABLE OF CONTENTS

Overview

This portfolio presents some of the projects that I am most proud of and that best represent me as a maker. I enjoy exploring a variety of topics by developing applications that each explore a unique area. Each of these projects provides insight into its field of study and has the potential to be used as a research instrument or teaching tool.

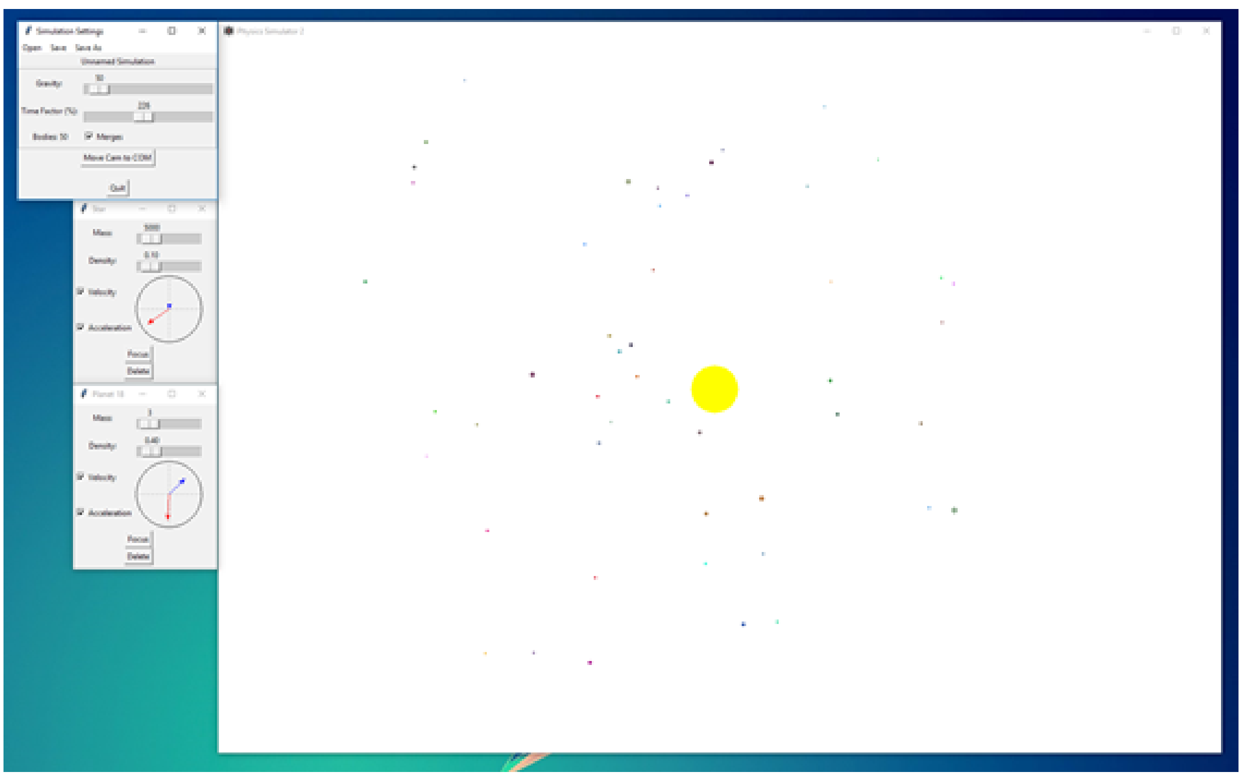
I have written the vast majority of the code on my own, but I have also enjoyed collaborating with friends on certain elements of these projects. I particularly found it helpful to brainstorm solutions to challenging problems with others.

I have published all of these projects under free-use licenses. The source code can be found at https://github.com/rschwa6308. Additionally, the source code for one select project is included in the appendix of this portfolio.

Project Summaries

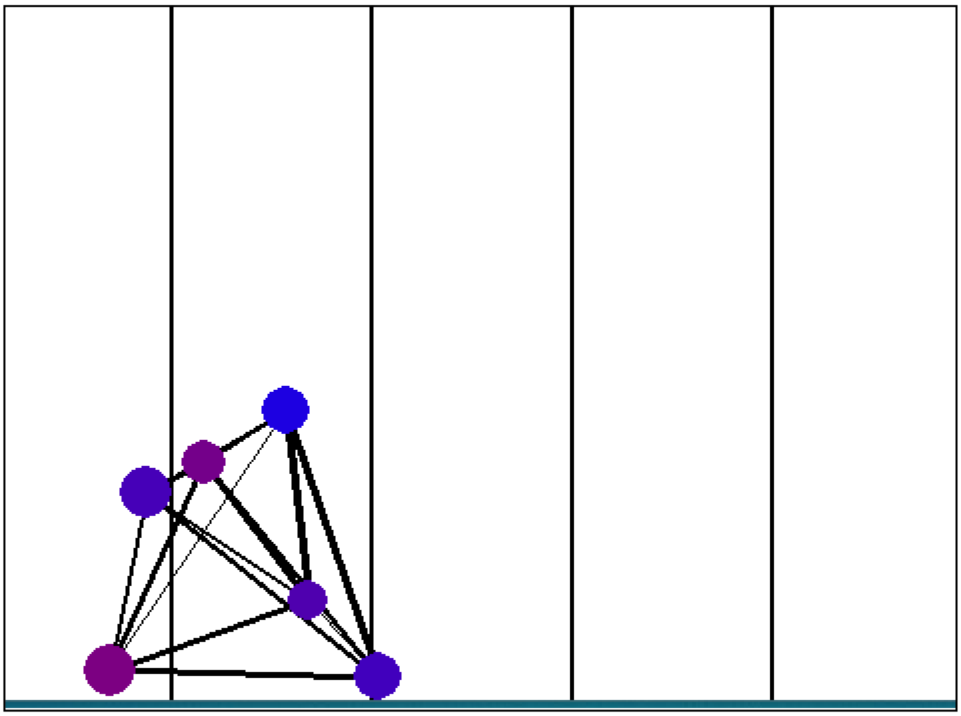
Physics-2.0

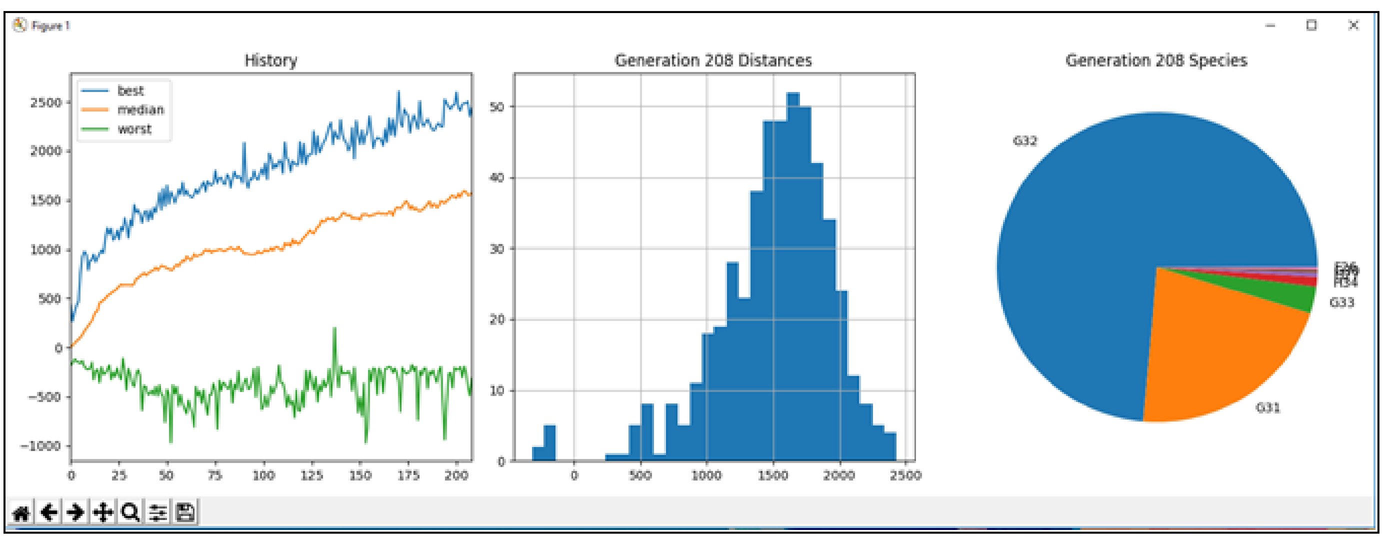
Physics-2.0 is a 2D n-body gravity simulator made in Python with the Pygame module. Bodies follow Newton's law of universal gravitation and conservation of momentum. This application can be used to demonstrate simple systems like binary stars and complex systems like planetary accretion and stellar dynamics. It can also be used to model fluids. Phenomena such as the Ideal Gas Law and Brownian Motion can be observed in these scenarios. *Physics-2.0* can be used in the classroom to help students develop an intuitive understanding of a wide range of topics in physics. **An in-depth examination and sample code are included in later sections of this document.**



Evolution Simulator

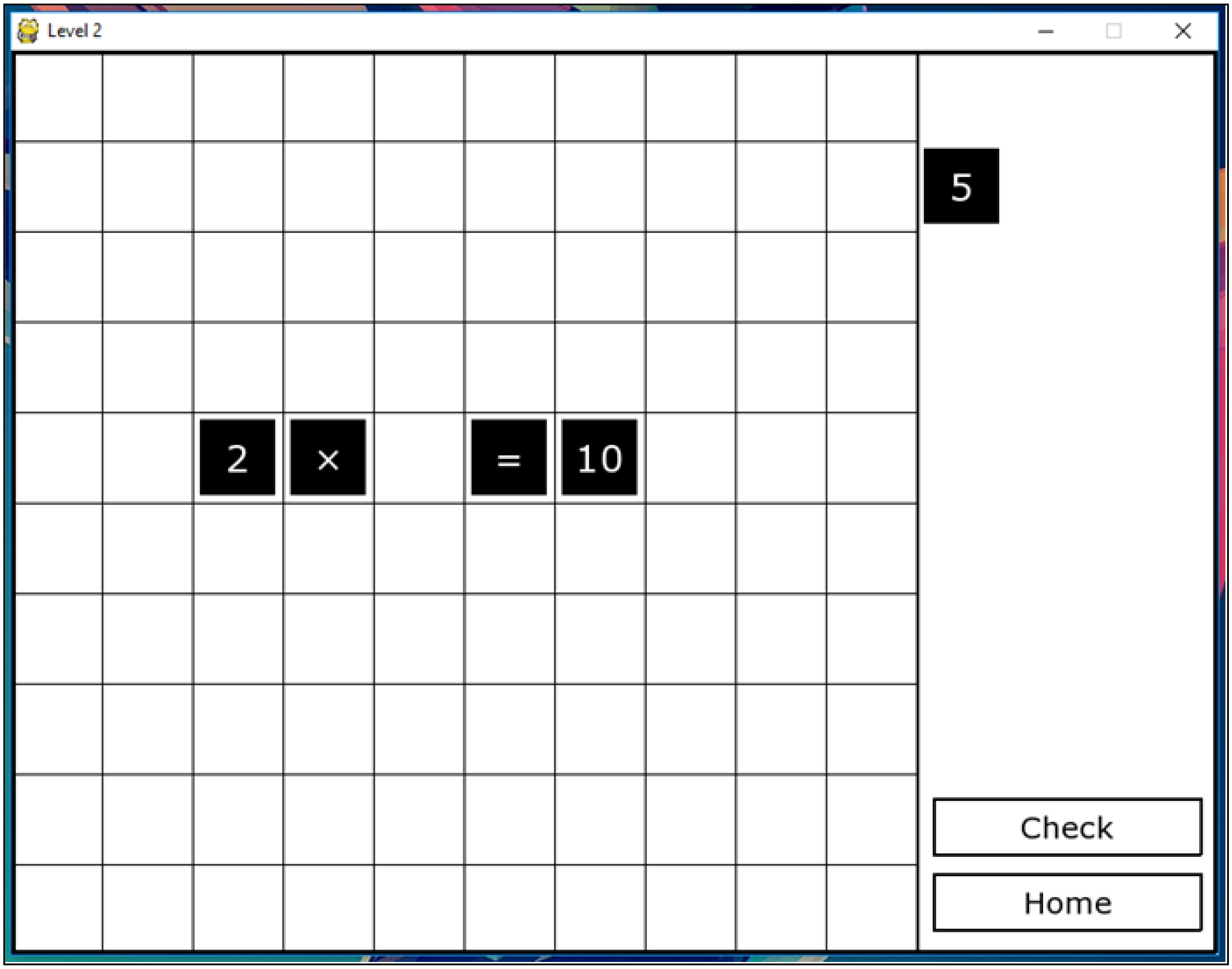
This application is a demonstration of genetic algorithms. A framework is provided for the creation of Organisms, which consist of joints and muscles. Organisms use their muscles in order to walk. Through repeated artificial selection and random mutation, a highly efficient population can evolve. Population statistics are tracked with live graphs.





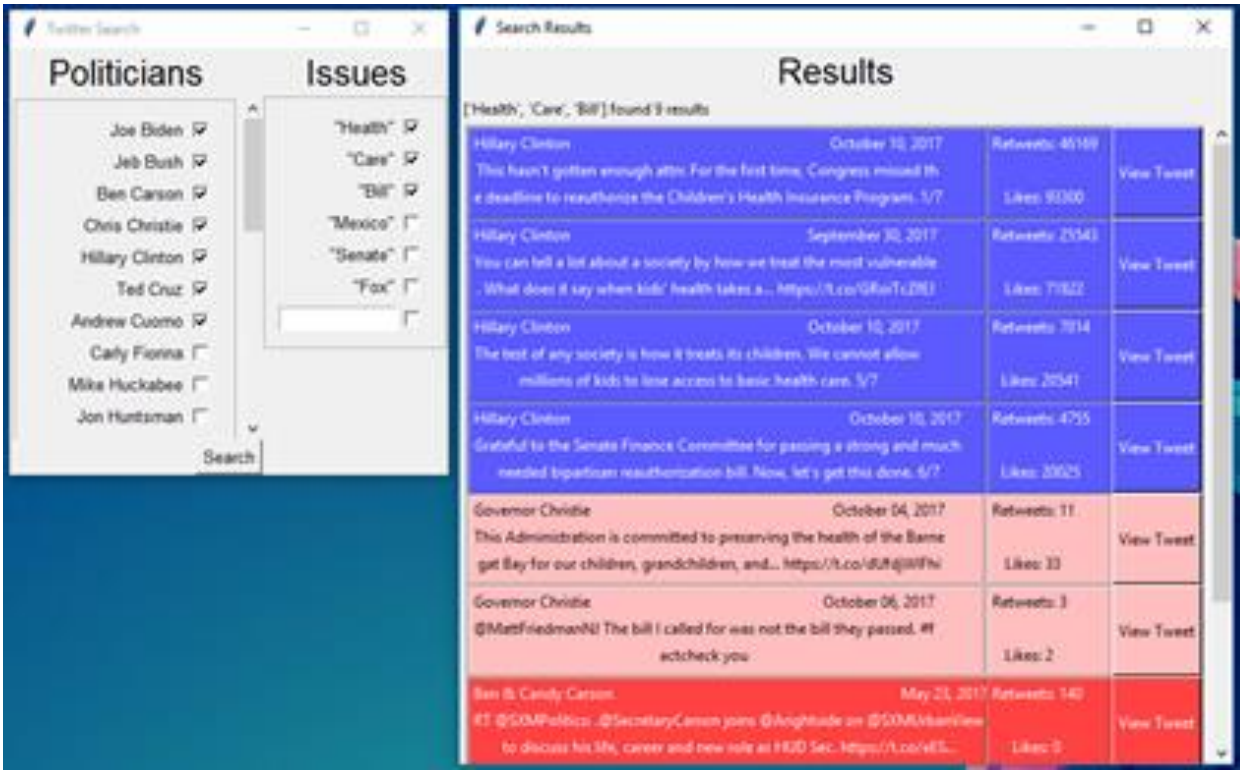
Arithma

Arithma is a simple puzzle game using Python. The player is supplied with tiles labeled with numbers and mathematical symbols. The player must place all of the tiles on the board such that one or more valid equations are formed. Arithma challenges numerical skills as well as spatial reasoning skills and critical thinking. The game has a fully featured UI and custom color schemes.



Twitter Search Engine

This python program provides a graphical utility that allows the user to perform an intelligent search of different politicians' tweets. The GUI uses Python's Tkinter module. Tweepy is used to access Twitter's public API service. Results are color coded based upon the given politician's political affiliation. This application was the winning entry at the 2017 HoCo Hacks coding competition.



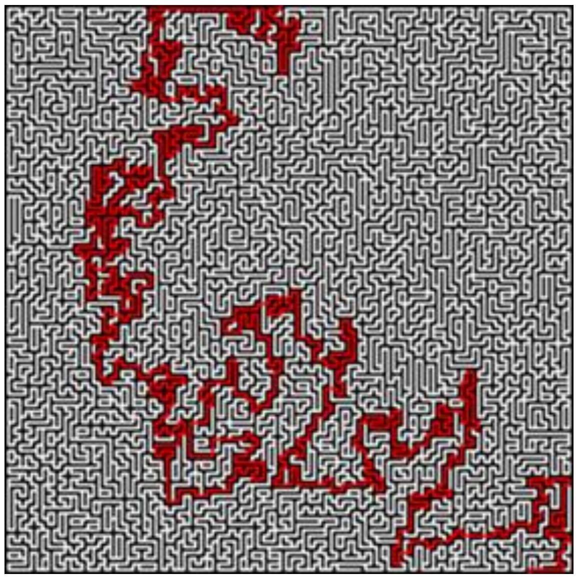
Derivative Calculator

Derivative Calculator is a command line utility for efficiently computing symbolic derivatives. The user inputs a function using standard mathematical notation, and the program will use the rules of differentiation and an application-specific computer-algebra-system to compute the derivative. Pictured here is a function that would be quite tedious to differentiate by hand. The *Derivative Calculator* can do the job in milliseconds.

```
d/dx(ln(sin(ln((x ^ arcsin((x ^ ln(x)))))))) =
(-x**log(x) + 1)**(-0.5)*(2*x**log(x)*log(x)**2 + (-x**log(x) + 1)**0.5*arcsin
(x**log(x)))/(x*tan(log(x**arcsin(x**log(x)))))
```

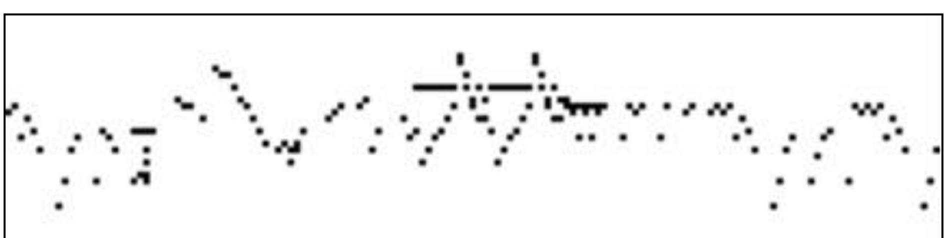
Maze Solver

This graphical application can be used to quickly solve classic mazes. Mazes are represented by black and white images. The user selects which maze he/she would like to solve, and the computer will generate an output image with the solution path drawn on top. The program does this by first generating a network of all hallways and intersections, and then searches through the network for a solution by using a breadth-first algorithm.



Mechanical Maestro

Mechanical Maestro is a computer program that composes classical music. It is an experiment in convolutional neural networks. A neural network was trained on a set of 1000 pieces of classical music, represented in image format. When the program is launched, the network attempts to generate images which are similar to the ones it was trained on, producing pseudo-original classical music.

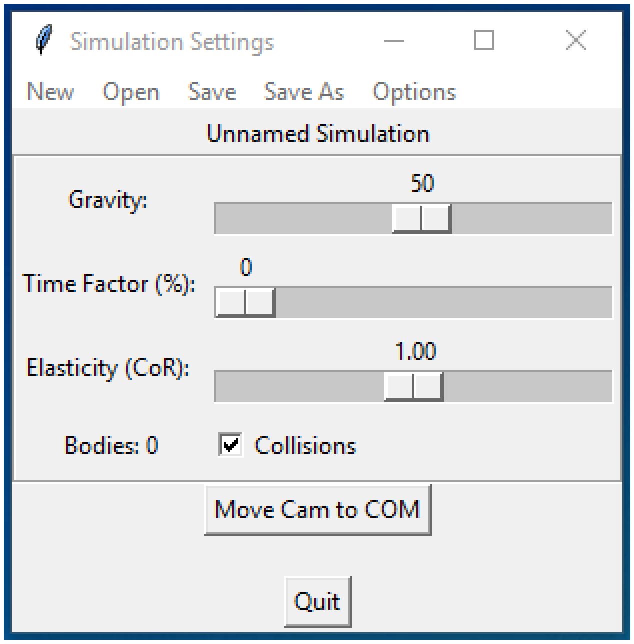


Excerpt from Für Elise

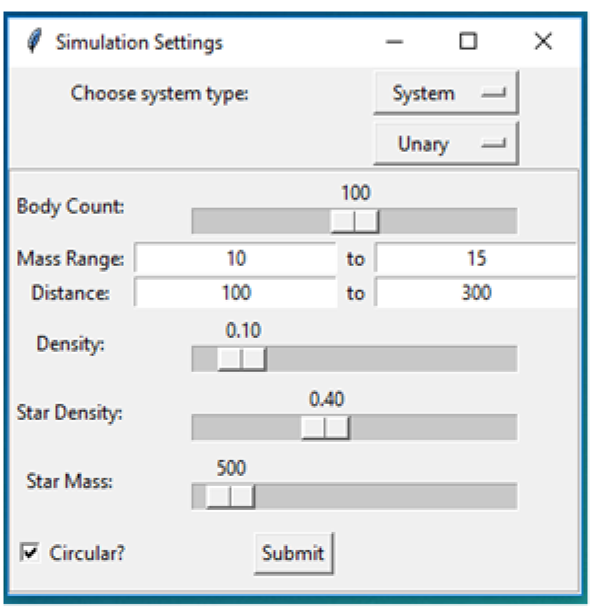
In-Depth Examination - Physics-2.0

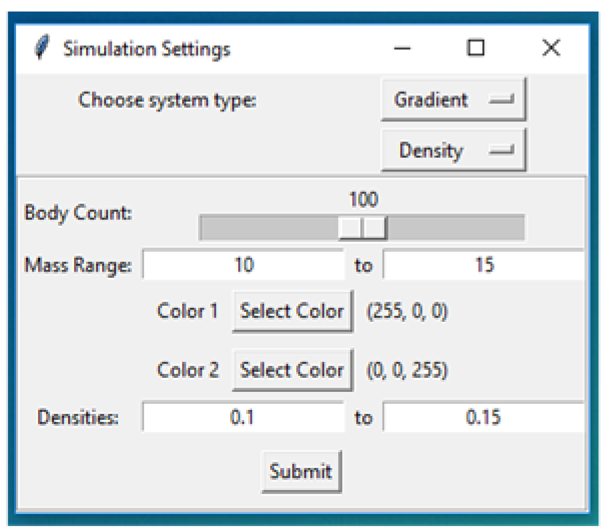
Physics-2.0 is a 2D n-body gravity simulator made in Python with the Pygame module. Bodies follow Newton's law of universal gravitation and conservation of momentum. It provides an intuitive interface for creating, running, and analyzing simulations of real world physical systems for the purpose of research and education.

The main menu screen is shown below. From this window, the user can create new simulations, open existing files, or adjust application settings. Files are saved using the custom '.sim' format. This window can also be used to adjust features of the simulation while it is running.

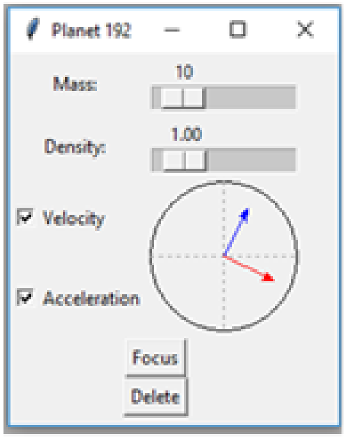


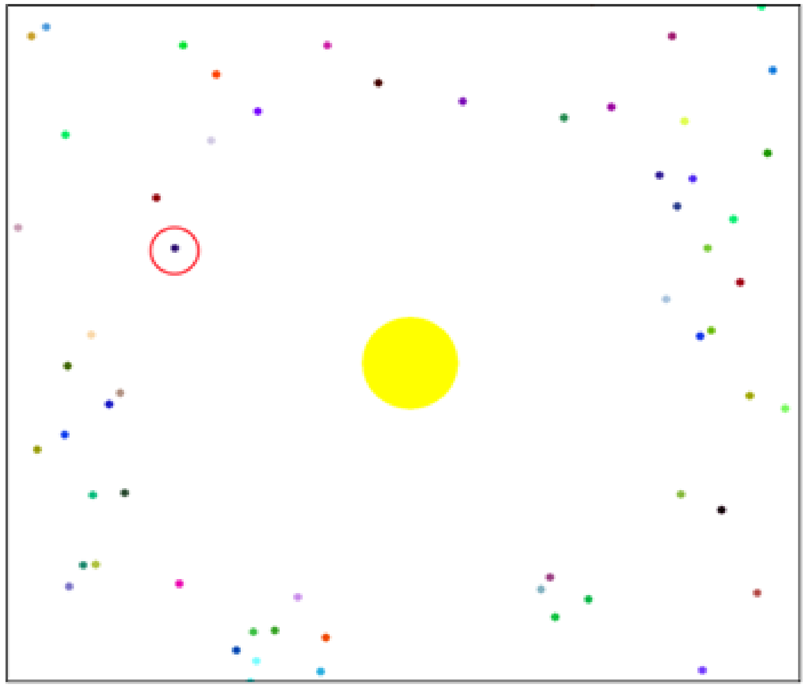
Pressing the 'New' button opens a simulation-creation wizard. The wizard for two different system types is show below.



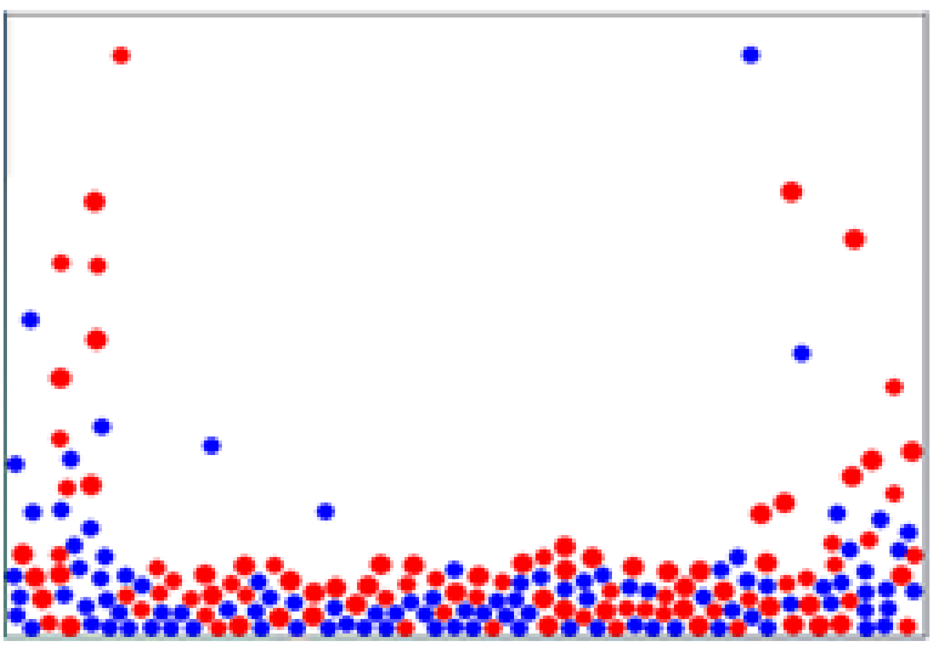


While a simulation is running, the user can select one or more bodies to analyze. A smaller side window is opened displaying live velocity and acceleration graphs. In the images below, Planet 192 (circled) is being analyzed. This is a simulation of a unary star system.



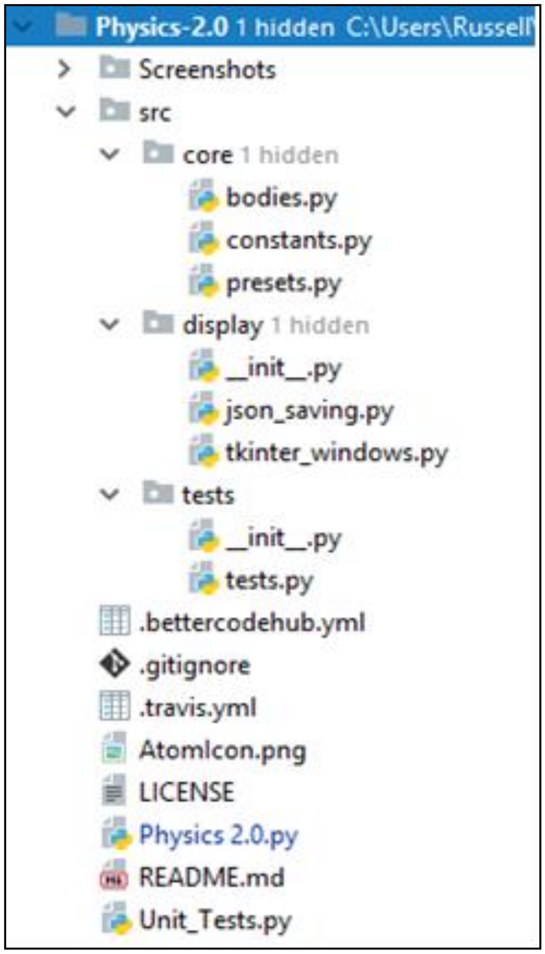


The image below is a simulation of a liquid sloshing as it is dropped into a container.



A sample of the source code for *Physics-2.0* is shown in the following section. The excerpt of python code references multiple classes which are defined in other files. The file structure of the entire project is displayed immediately below. The source code can be viewed in its entirety at *http://www.github.com/rschwa6308/Physics-2.0*. Finally, *Physics-2.0* is complete with a detailed readme, unit tests, and code quality documentation.

Project File Structure



Code Quality Badges

Physics 2.0.py - Sample Code

```
from functools import reduce
from operator import add
from pygame.math import Vector2 as V2
import pygame as pg, os

from src.display.tkinter_windows import create_menu
from src.core import constants


def init_display():
    pg.init()
    info = pg.display.Info()
    dims = (int(info.current_w * 0.6), int(info.current_h * 0.75))
    os.environ['SDL_VIDEO_CENTERED'] = '1'
    pg.display.set_icon(pg.image.load('AtomIcon.png'))
    screen = pg.display.set_mode(dims, pg.RESIZABLE)
    pg.display.set_caption("Physics Simulator 2.0")
    return screen, V2(dims)


def refresh_display(settings_window, screen, bodies, cam):
    screen.fill(settings_window.bg_color)  # comment out this line for a fun time ;)
    if settings_window.walls.get():
        pg.draw.rect(screen, (0, 0, 0), pg.Rect(0, 0, *cam.dims), 3)
    for b in bodies:
        # Calculate coordinates and radius adjusted for camera
        x, y = (b.position - cam.position - cam.dims / 2) * cam.scale + cam.dims / 2
        pg.draw.circle(screen, b.color, (int(x), int(y)), int(b.radius * cam.scale), 0)
        # The radius should be calculated in such a way that the camera can be zoomed
indefinitely.
        # Currently, the properties of an object can reach a distinct threshold, after
which they become invisible.
    pg.display.update()


def update_windows(settings_window):
    arr = [0, 0, [0] * 5]
    if settings_window.alive:
        settings_window.update()
        try:
            arr = [settings_window.gravity_slider.get() / 100,
settings_window.COR_slider.get(),
                   [settings_window.time_slider.get() / 100,
                    settings_window.collision.get(), settings_window.walls.get(),
settings_window.g_field.get(),
```

```
                    settings_window.gravity_on.get()]]
        except:
            pass
    for window in settings_window.properties_windows:
        if window.alive:
            window.update()
        else:
            settings_window.properties_windows.remove(window)
    return arr


def handle_mouse(*args):
    settings_window, camera, event, bodies, dims, G, COR, scroll = args
    if event.button == 1:
        pos = camera.position + (pg.mouse.get_pos() - dims / 2) / camera.scale + dims /
2
        for b in bodies:
            if b.click_collision(pos) and b not in [win.body for win in
settings_window.properties_windows]:
                if not settings_window.alive:  # Respawn the main window if it is dead
                    settings_window.__init__(bodies, camera, dims, [G, COR])  # This
still does not fix all errors
                settings_window.properties_windows.append(
                    create_menu("BodyProperties", bodies, camera, dims,
len(settings_window.properties_windows), b))
    elif event.button == 4:
        camera.scale = min(camera.scale * 1.1, 100)
        scroll.scale /= 1.1
    elif event.button == 5:
        camera.scale = max(camera.scale / 1.1, 0.01)
        scroll.scale *= 1.1


def handle_events(*args):
    settings_window, camera, scroll, done, dims, screen, bodies, G, COR = args
    for event in pg.event.get():
        if event.type == pg.VIDEORESIZE:
            width, height = event.w, event.h
            dims, screen = V2(width, height), pg.display.set_mode((width, height),
pg.RESIZABLE)
        elif event.type == pg.KEYDOWN:
            scroll.key(event.key, 1)
            camera.key_down(event.key)
        elif event.type == pg.KEYUP:
            scroll.key(event.key, 0)
            camera.key_up(event.key)
        elif event.type == pg.MOUSEBUTTONDOWN:
            handle_mouse(settings_window, camera, event, bodies, dims, G, COR, scroll)
        done |= event.type == pg.QUIT
    return done, dims, screen
```

```
def handle_bodies(*args):
    G, COR, time_factor, collision, walls, g_field, gravity, scroll, bodies, camera,
dims, frame_count, settings_window = args

    for body in bodies:  # Reset previous calculations
        body.acceleration = V2(0, 0)

    for b, body in enumerate(bodies):  # Calculate forces and set acceleration, if
mutual gravitation is enabled
        for o in range(len(bodies) - 1, b, -1):
            if collision and bodies[o].test_collision(body):
                if not COR:  # Only remove second body if collision is perfectly
inelastic
                    bodies[o].merge(bodies[b], settings_window.properties_windows)
                    bodies.pop(b)
                    break
                bodies[o].collide(bodies[b], COR)
            if gravity:
                force = body.force_of(bodies[o], G)  # This is a misnomer; `force` is
actually acceleration / mass
                body.acceleration += bodies[o].mass * force
                bodies[o].acceleration -= body.mass * force
        body.acceleration.y += G / 50 * g_field  # Uniform gravitational field
        body.apply_motion(time_factor)
        body.position += scroll.val
        if not frame_count % 100 and body.position.length() > 100000:  # TODO: find a
good value from this boundary
            bodies.remove(body)
            for window in settings_window.properties_windows:
                if window.body is body:
                    settings_window.properties_windows.remove(window)
                    window.destroy()
                    break
        if walls:  # Wall collision
            d, r = ((body.position - camera.position) - dims / 2) * camera.scale + dims
/ 2, body.radius * camera.scale
            for i in 0, 1:
                x = d[i]  # x is the dimension (x,y) currently being tested / edited
                if x <= r or x >= dims[i] - r:
                    body.velocity[i] *= -COR  # Reflect the perpendicular velocity
                    body.position[i] = (2 * (x < r) - 1) * (r - dims[i] / 2) /
camera.scale + dims[i] / 2 + \
                                       camera.position[i]  # Place body back into frame


class Scroll:
    def __init__(self):
        self.down, self.map, self.val, self.scale = [0, 0, 0, 0], [pg.K_a, pg.K_w,
pg.K_d, pg.K_s], V2(0, 0), 1

    def key(self, key, down):
```

```
        if key in self.map:
            self.down[self.map.index(key)] = down

    def update_value(self):
        self.val = (self.val + self.scale * (V2(self.down[:2]) - self.down[2:])) * .95


class Camera:
    def __init__(self, dims):
        self.position, self.velocity, self.dims, self.scale, self.map = V2(0, 0), V2(0,
0), dims, 1, [pg.K_RIGHT,

pg.K_LEFT,

pg.K_UP,

pg.K_DOWN]

    def key_down(self, key):
        if key in self.map:
            self.velocity = V2((3 / self.scale, 0) if key in self.map[:2] else (0, 3 /
self.scale)).elementwise() * (
                (self.map.index(key) not in (1, 2)) * 2 - 1)

    def key_up(self, key):
        if key in self.map:
            self.velocity = self.velocity.elementwise() * ((0, 1) if key in
self.map[:2] else (1, 0))

    def move_to_com(self, bodies):
        total_mass = sum(b.mass for b in bodies)
        self.position = reduce(add, (b.position * b.mass for b in bodies)) / total_mass
- self.dims / 2

    def move_to_body(self, body):
        self.position = body.position - self.dims / 2

    def apply_velocity(self):
        self.position += self.velocity


def main():
    screen, dims = init_display()
    bodies, camera, scroll = [], Camera(dims), Scroll()

    settings_window, clock, done, frame_count = create_menu("Settings", bodies, camera,
dims,
                                                            [constants.G,
constants.COR]), pg.time.Clock(), False, 0

    while not done:
```

```
        clock.tick(constants.clock_speed)
        frame_count += 1

        camera.apply_velocity()
        G, COR, misc_settings = update_windows(settings_window)
        done, dims, screen = handle_events(settings_window, camera, scroll, done, dims,
screen, bodies, G, COR)
        handle_bodies(G, COR, *misc_settings, scroll, bodies, camera, dims,
frame_count, settings_window)
        refresh_display(settings_window, screen, bodies, camera)
        scroll.update_value()

    pg.quit()
    if settings_window.alive: settings_window.destroy()


if __name__ == "__main__":
    main()
```