Game Engine Optimisation – Assignment 1

Extending the functionality or realism of a physics engine (pge)

Program Listing: -

myspringgame.py

pge.py

pgeif.i

Gpgif.h

pgeif.c

twoDsim.c

GtwoDsim.h

Makefile.am

Changes Made: -

For the purposes of this assignment it was required to edit the main files of the game engine to add new functionality to it and to optimise older functionality to produce a smoother fps when the engine was run.

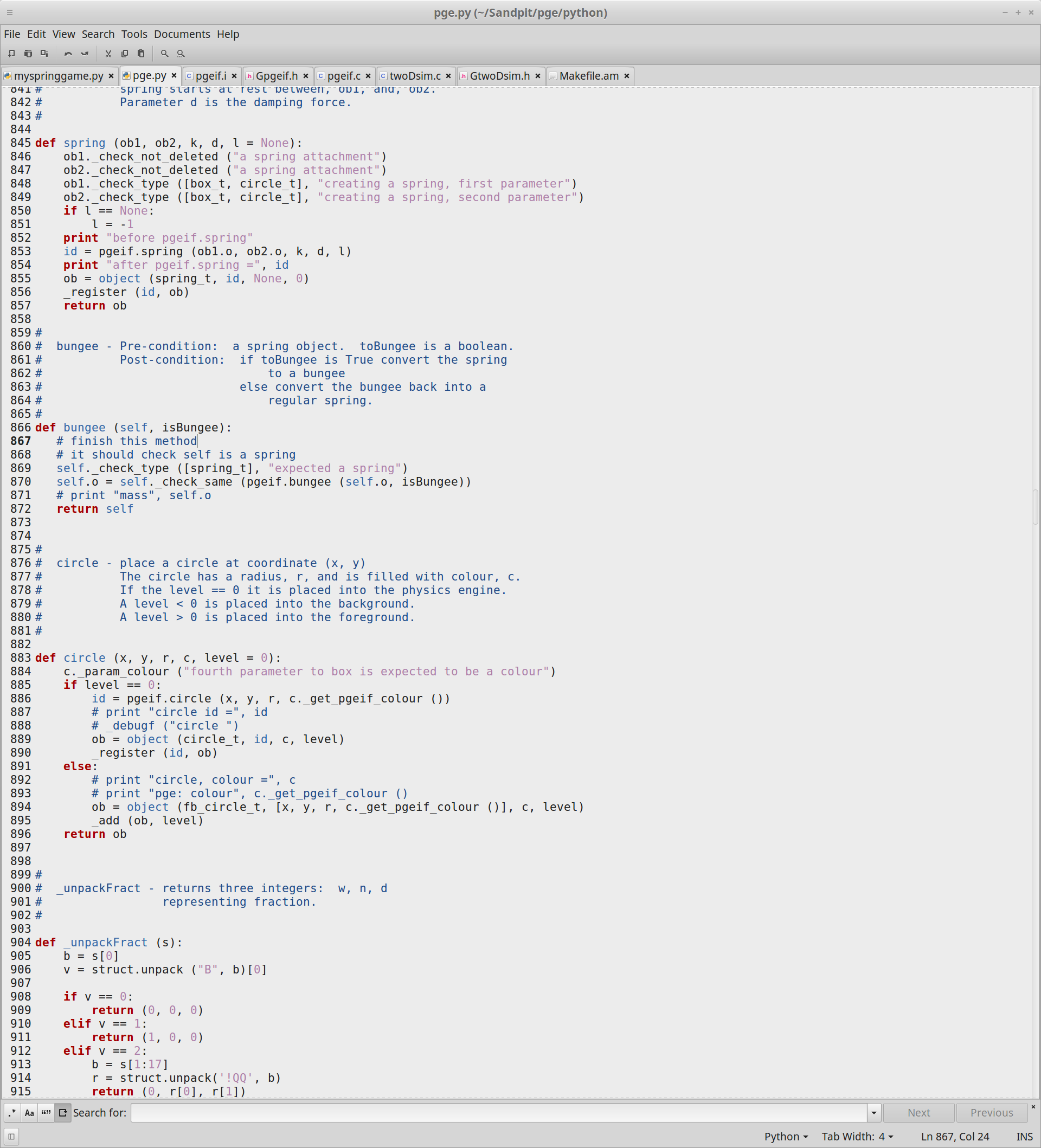
The changes made to the engine spanned multiple files for multiple purposes such as adding in the bungee functionality to the spring or to add in ways to optimise existing methods like the interpenetration of colliding circular objects. These will all be discussed within their respective file headers.

myspringgame.py:

This file is the python file that contains the code for the 2D game created to output the differences in the effects of the various optimisation methods. The file should also contain the usage and output of any additional code added to the engine which for example includes the code that allows for a spring to be turned into a bungee instead. The screenshot below shows the final version of my game created to show off the results of the optimisation methods used as well as the implemented bungie.

pge.py:

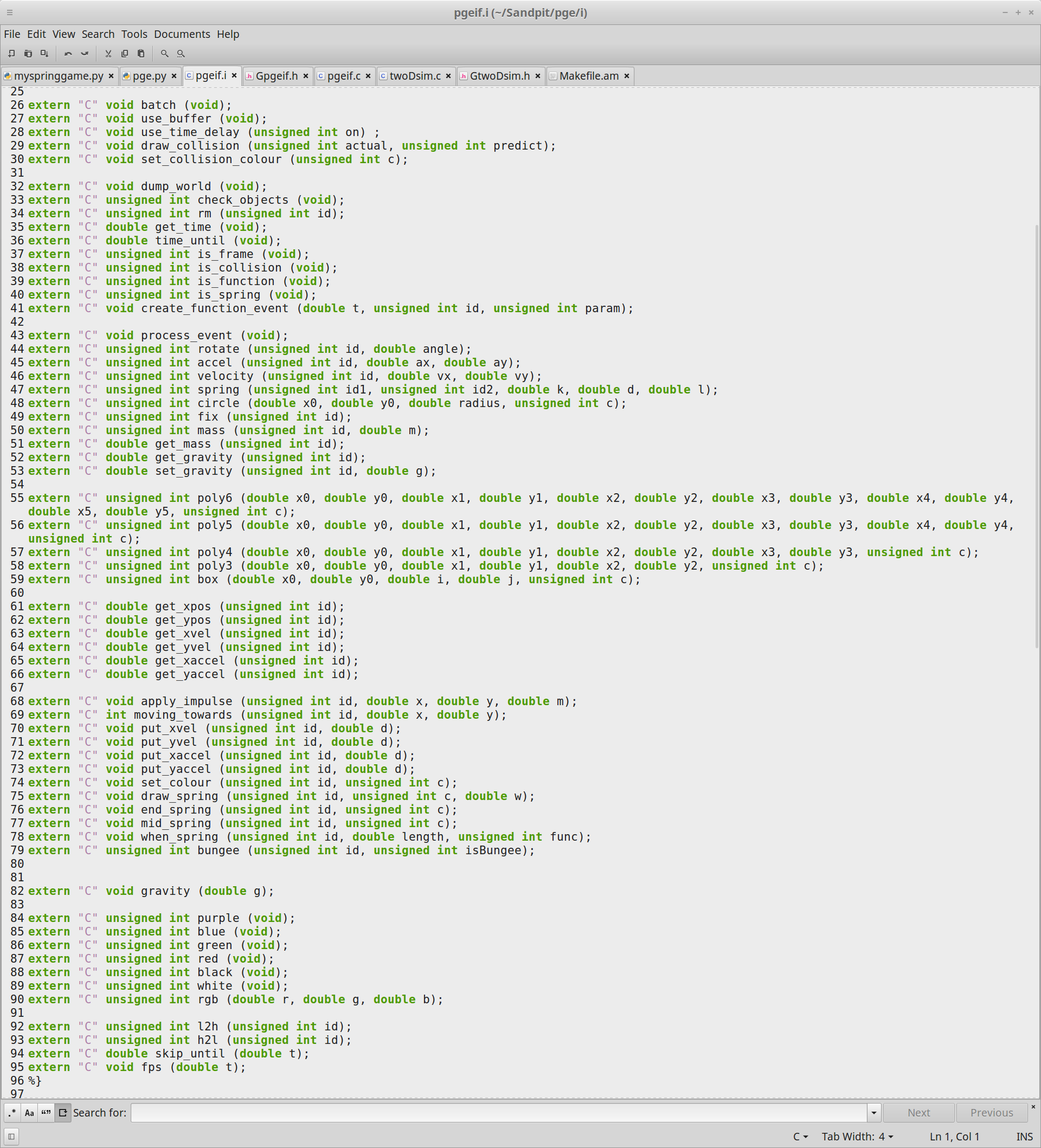
This file includes the definition for the bungee to check whether or not object is a spring or a bungee by using a Boolean that triggers when true to convert the spring into a bungee. The screenshot below shows the code in this file.



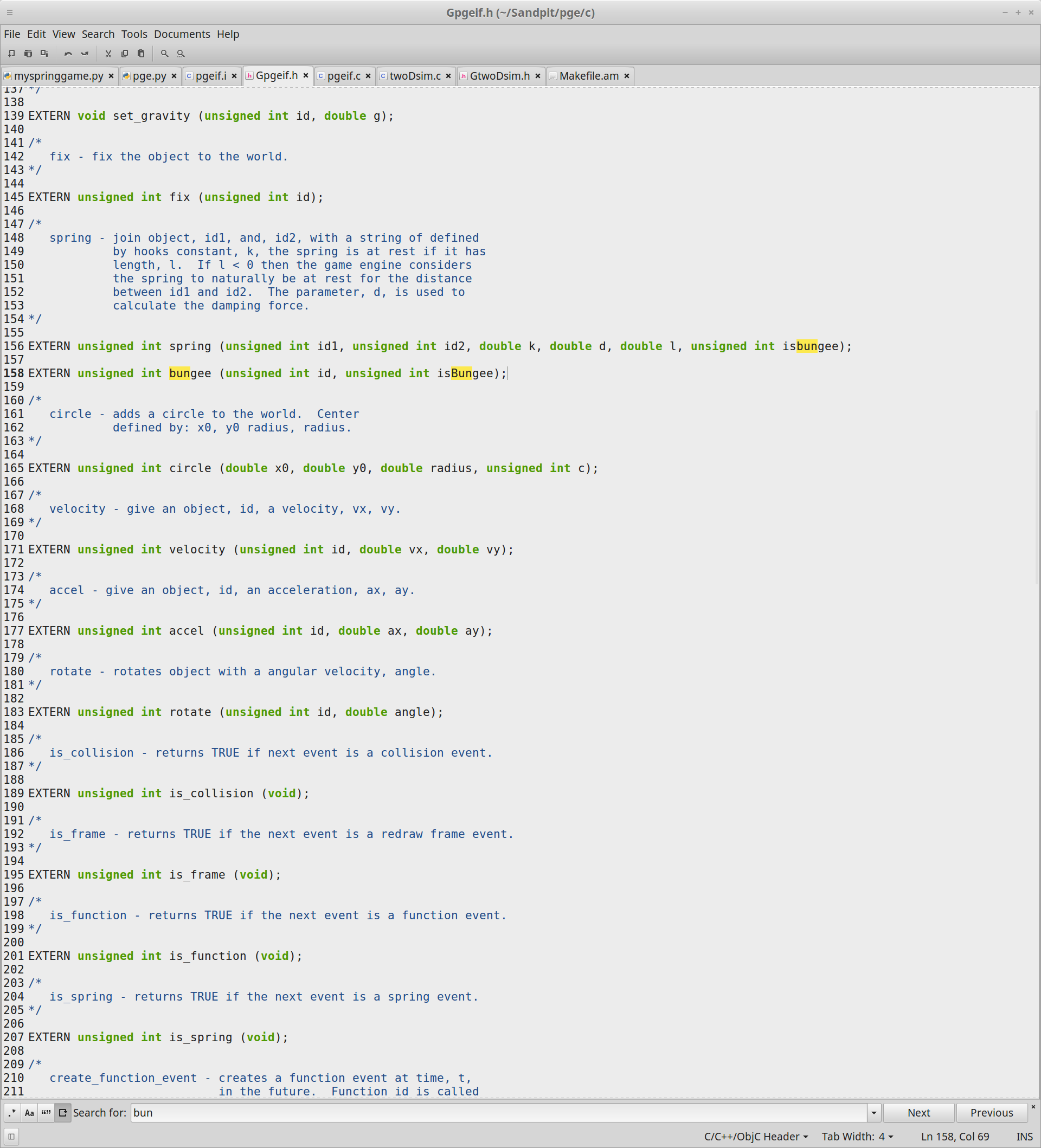
The code in this file shows that the definition of bungee uses checktype to see if theobject referred to is a spring then performs a checksame to find out whether the object has a bungee modifier.

pgeif.i:

This file includes the C and C++ definitions for the bungee code this has to be done twice, once for each language. The screenshot below shows an image of the code.

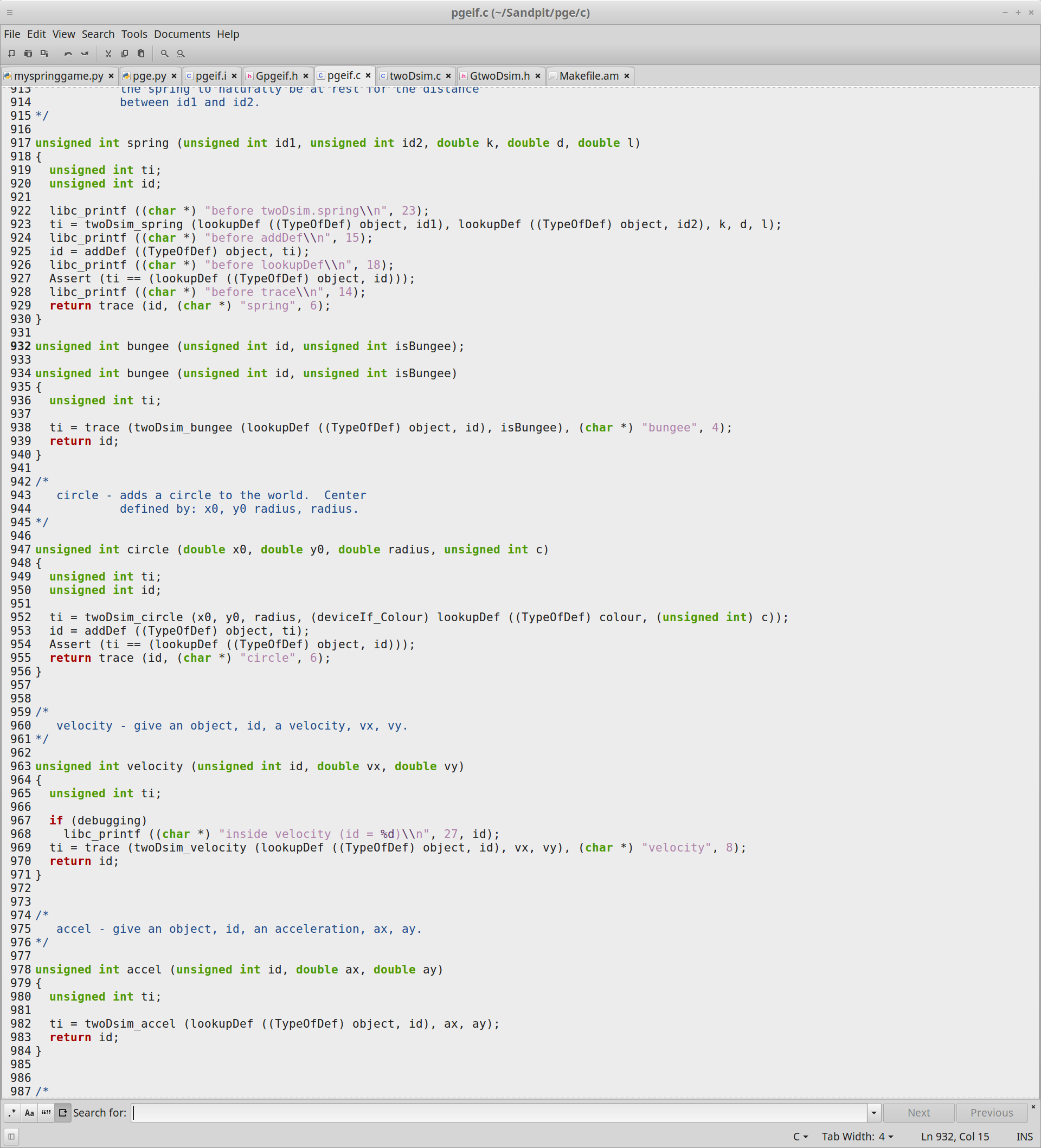


Gpgif.h:

This file is the header file for pgeif.c which contains the external functions implemented in that file. The screenshot below contains the code in this file.

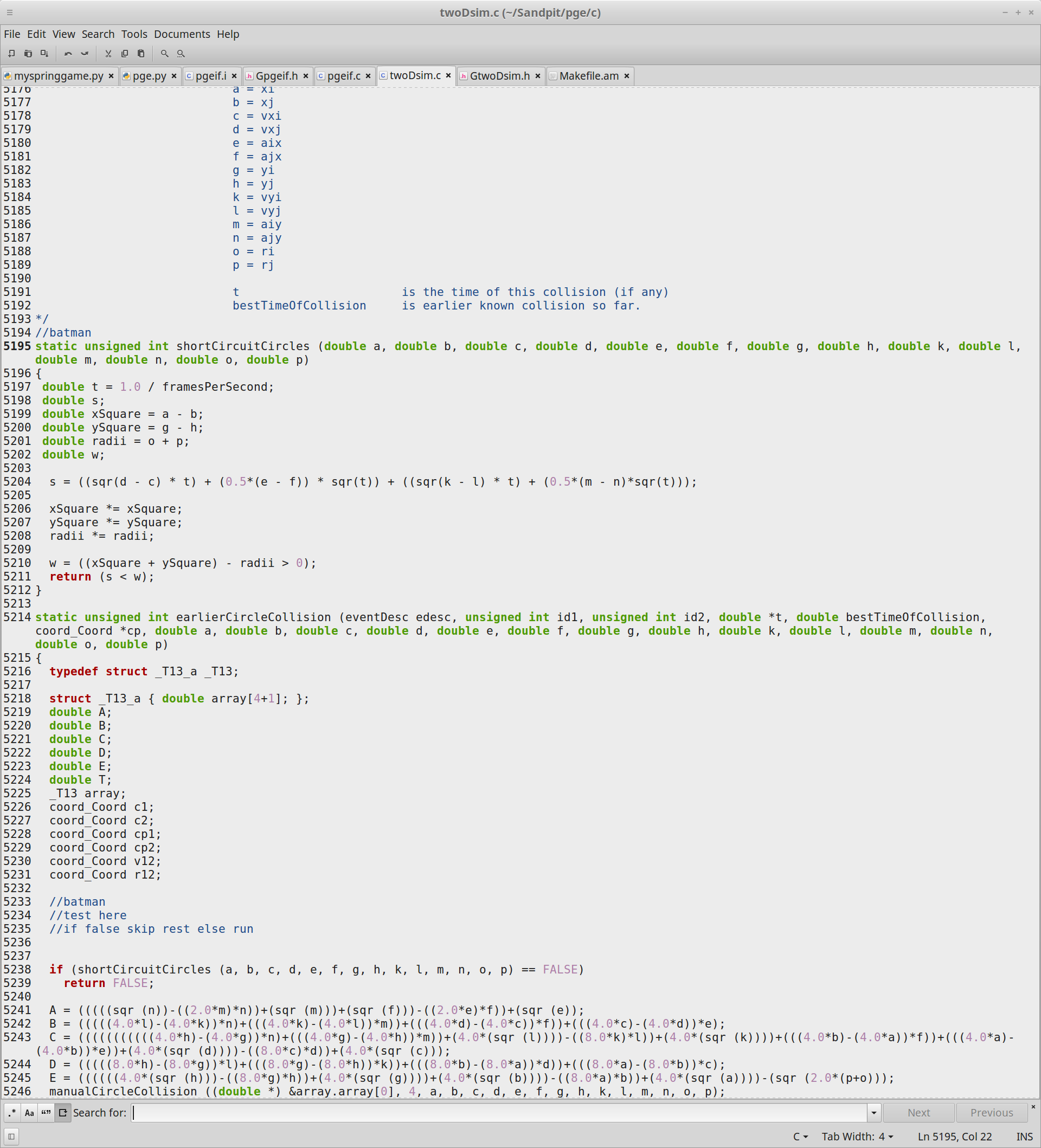
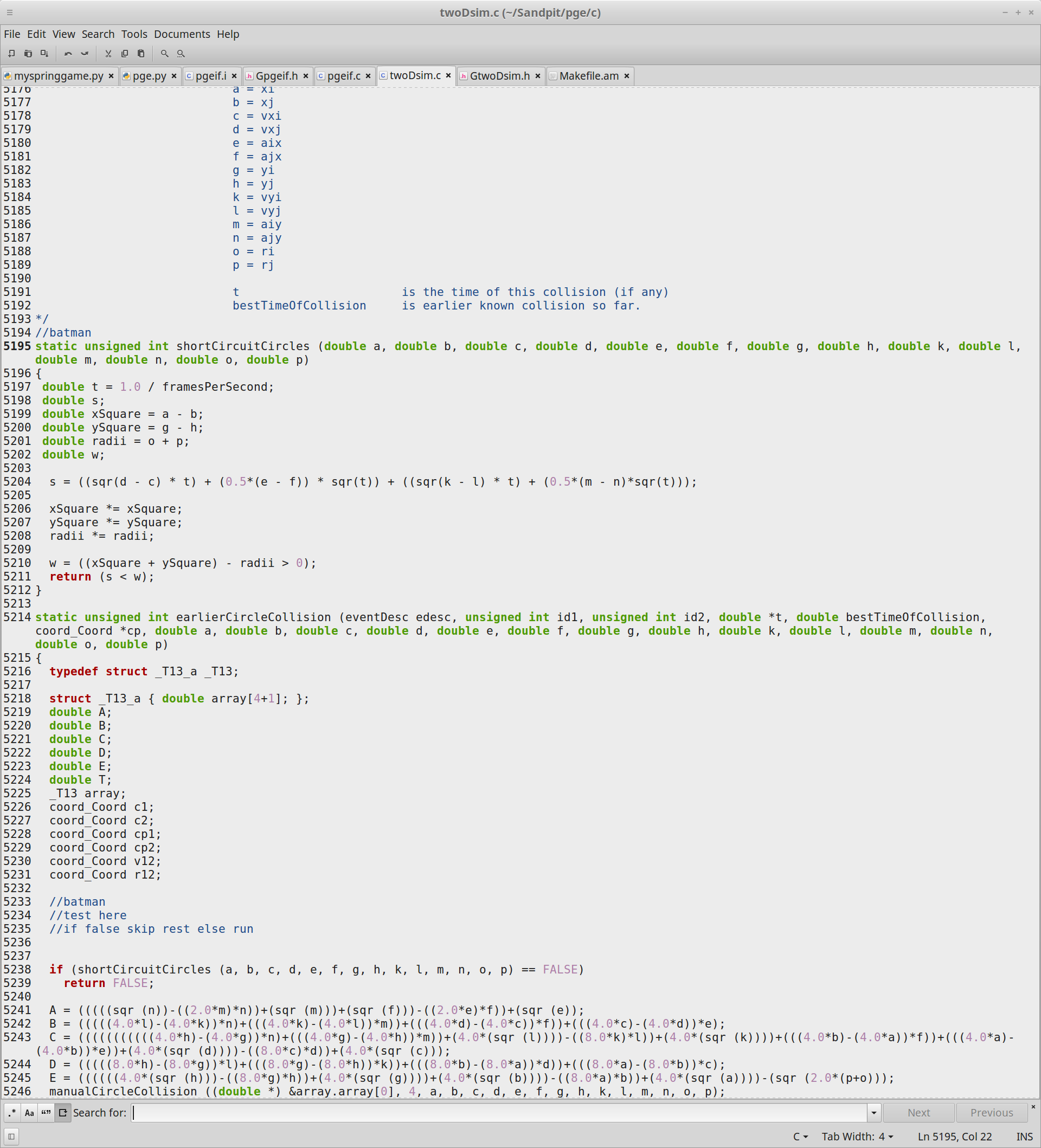
pgeif.c:

This file allows created objects to be given unique ids which allows all objects to be called using these ids. The screenshot below contains the code in this file.

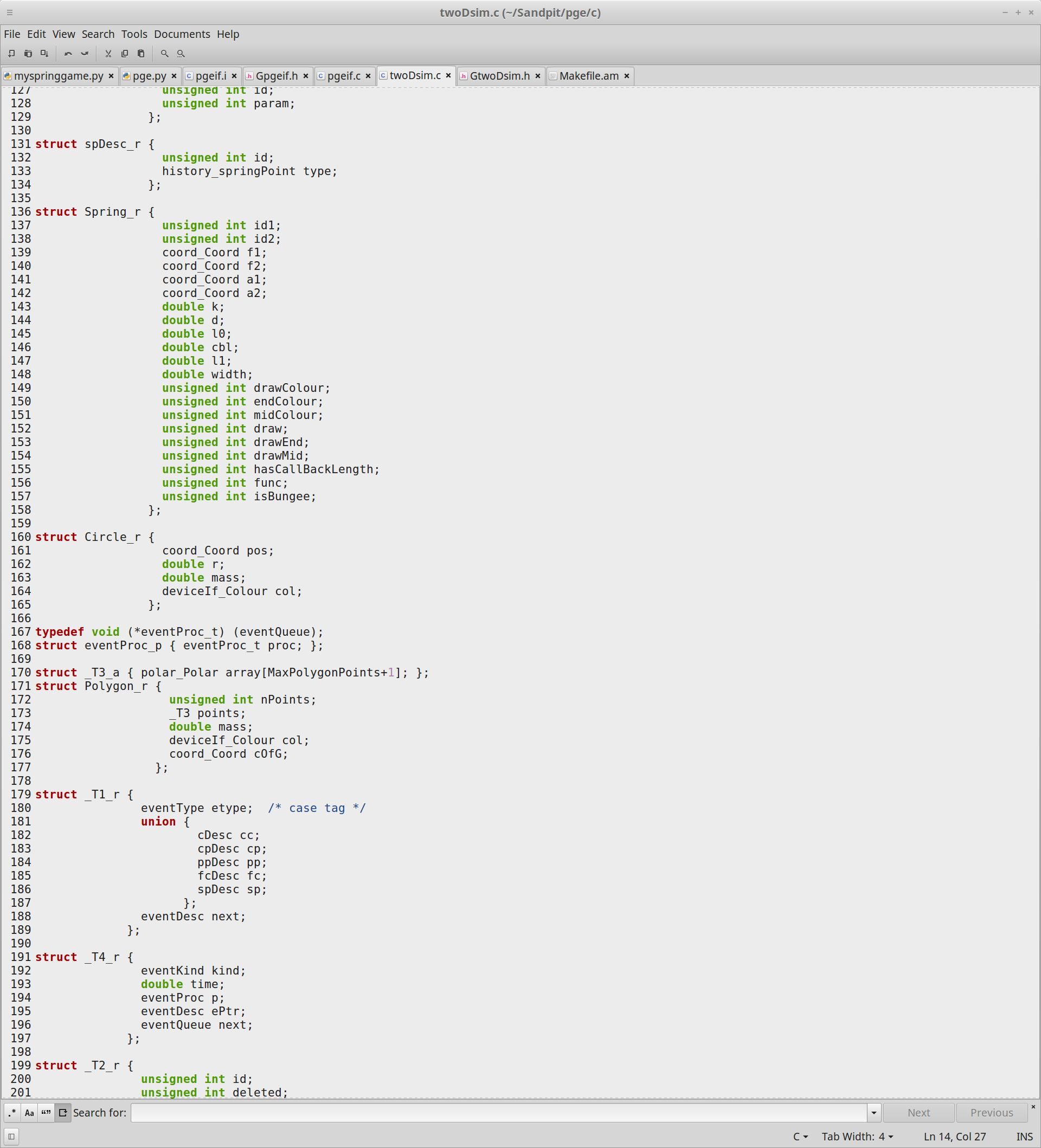


twoDsim.c:

This file has had most of the edits added to it, it has been updated to include a short circuit test for circle collisions that instead of testing every second for whether a collision has occurred it calculates whether two circles might collide in the near future by using a complex algorithm that uses information from the positions, velocities, accelerations and radii of both circles to determine whether the circles have a gap between them then testing that gap against the total relative distance. The screenshot below shows the code for the shortCircuitCircleCollisions method created in this file.

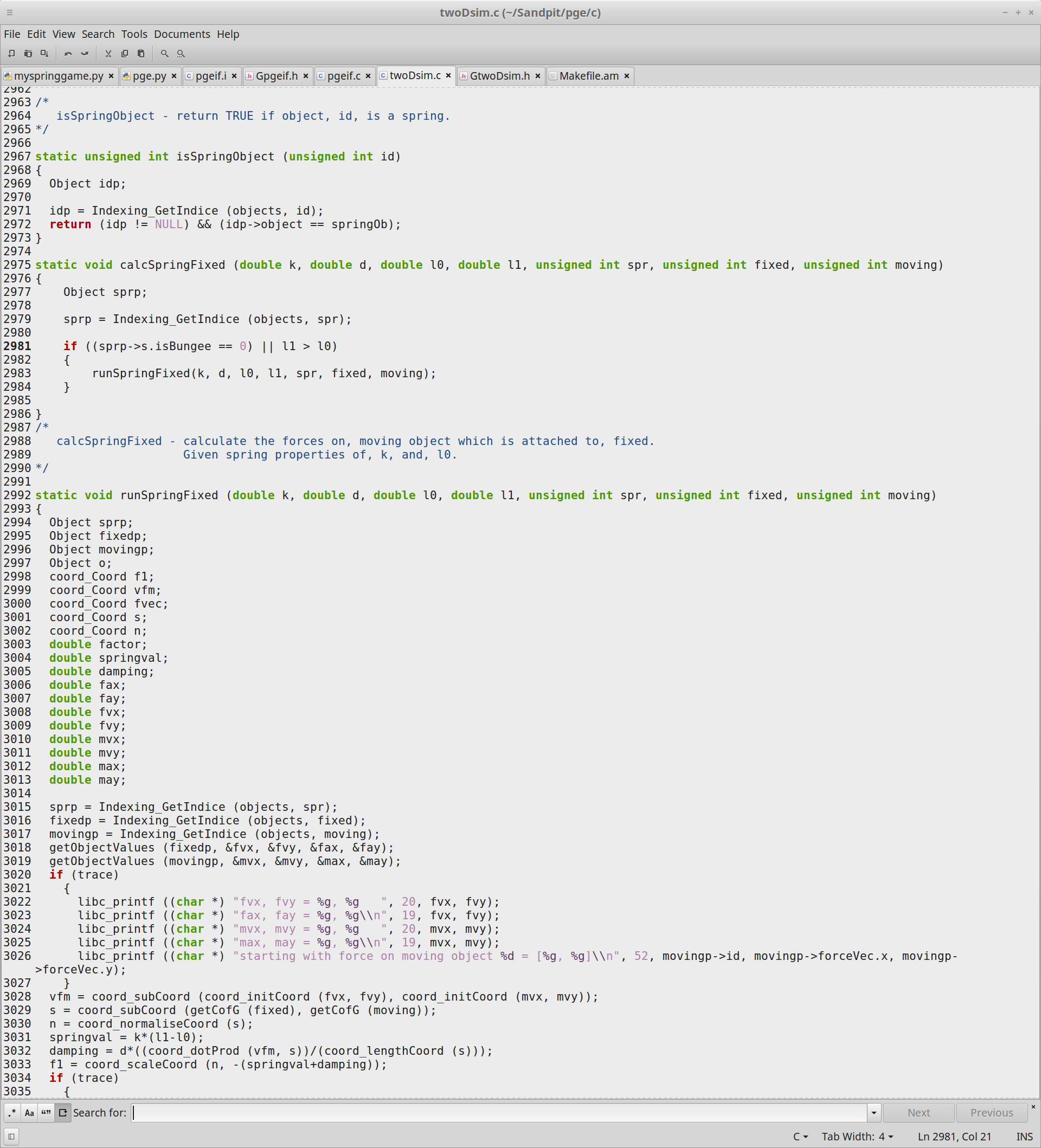
The code in this part shows that we add a shortCircuitCircles method above the earlierCircleCollisions method. This method is set to run as part of a test the method tests the whether short circuit circles returns a true result from the algorithm it runs. If it does that means that two circles will collide in the next few frames and will run the next bit of code, otherwise if false is returned the code will instead bypass running the rest of the code as the two circles will not collide in the next few frames.

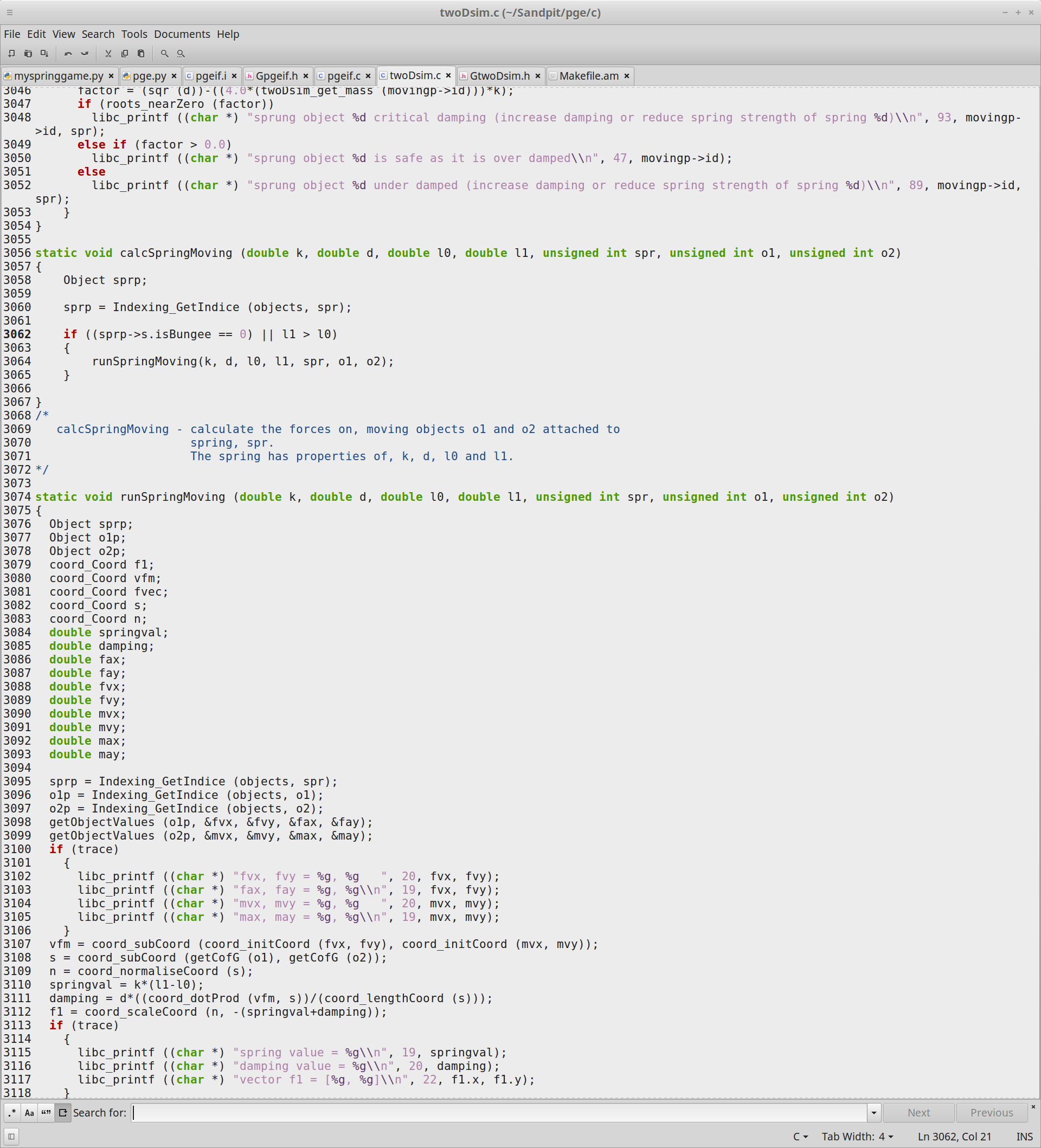
This file also includes the code for the bungee modifications to the spring code. This was done by adding in a Boolean to check whether the spring is a bungee or not, some additional methods were also added to the calcSpringFixed and calcSpringMoving methods that first check whether the object is a spring or a bungee and then decides whether it will use the calculations for springs or not. The screenshot below shows the code for the bungee object.



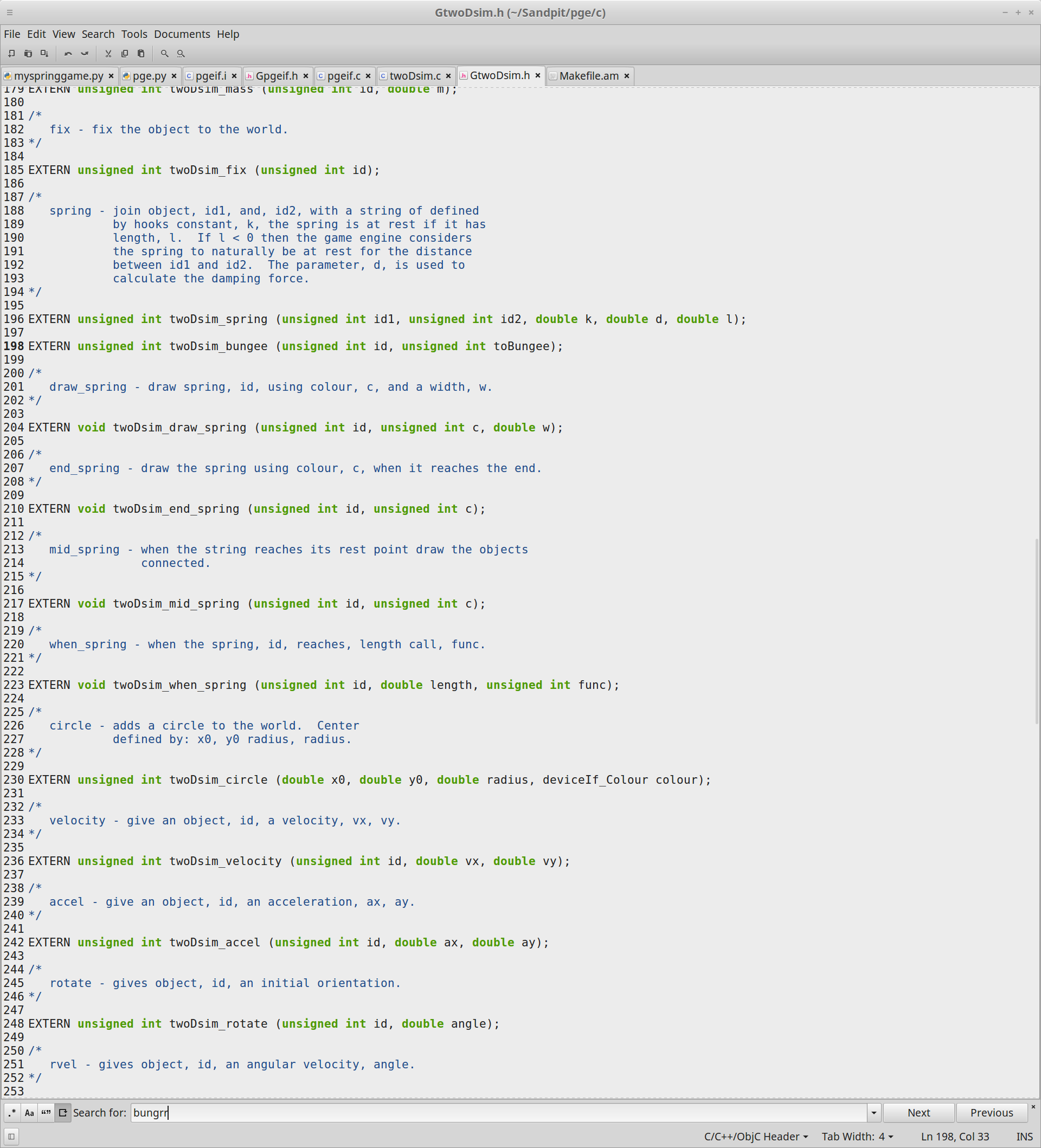
There are multiple ways that can be used to add in the bungee functionality onto the spring one of the ways was to create an entirely new object seperate from all of the others, however this would take a long time and since most of the functionality already exists in the spring that was adapted instead. In order to do this multiple changes had to be made to the spring to check for whether the object was a spring or a bungee, to do this the isBungee boolean was added to the spring.



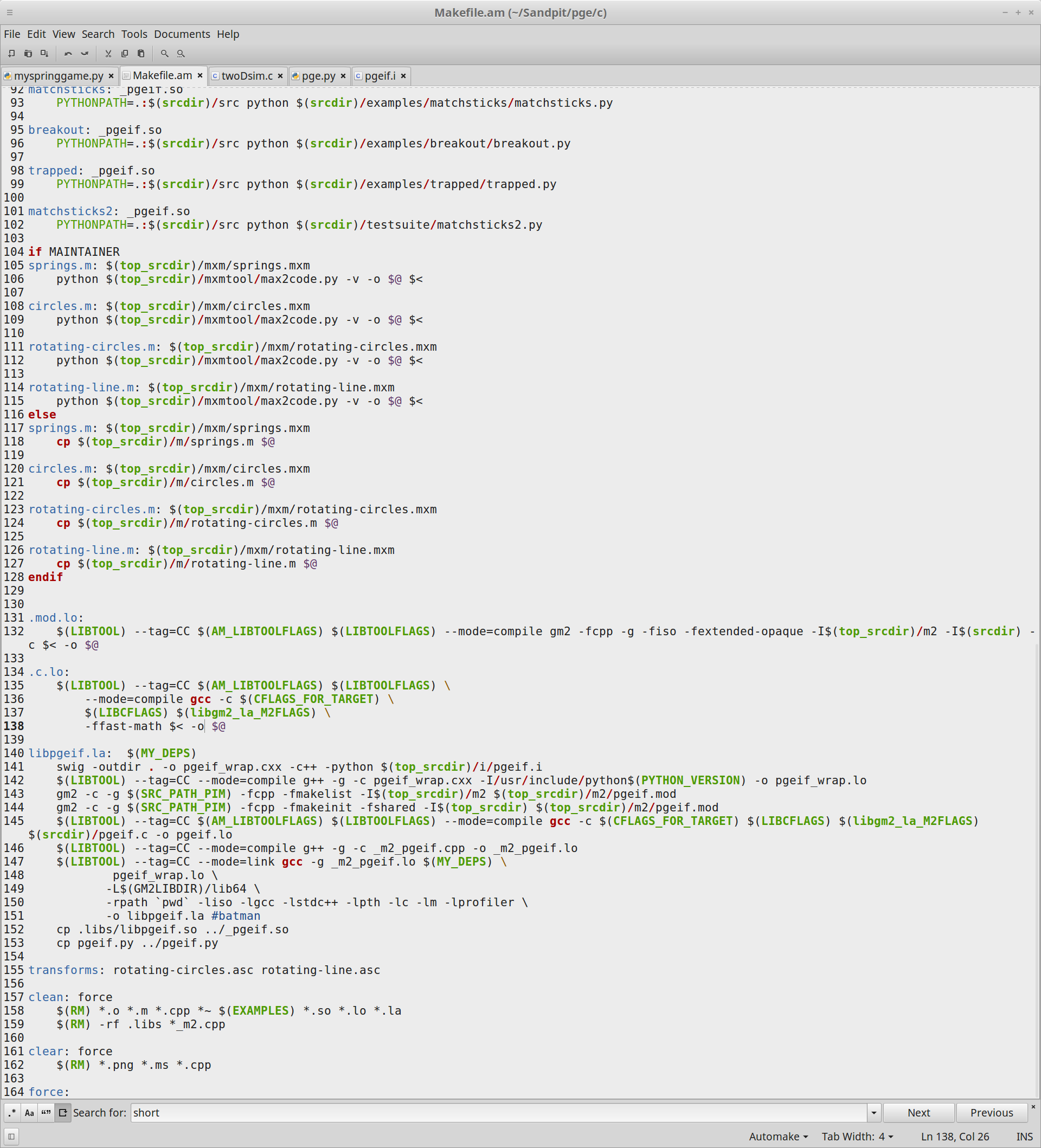
In this code the twoDsim\_bungee method was added it is used to set the isBungee variable in spring to true.

The code shown here is part of the calculations of the spring fixed and moving methods these methods were made to replace the original methods used which when called check whether the isBungee is false or whether l1 is greater than l0 if either of these is correct it means that the spring is in fact a bungee and will not run the original code that was in these methods by calling the methods that they were moved to.

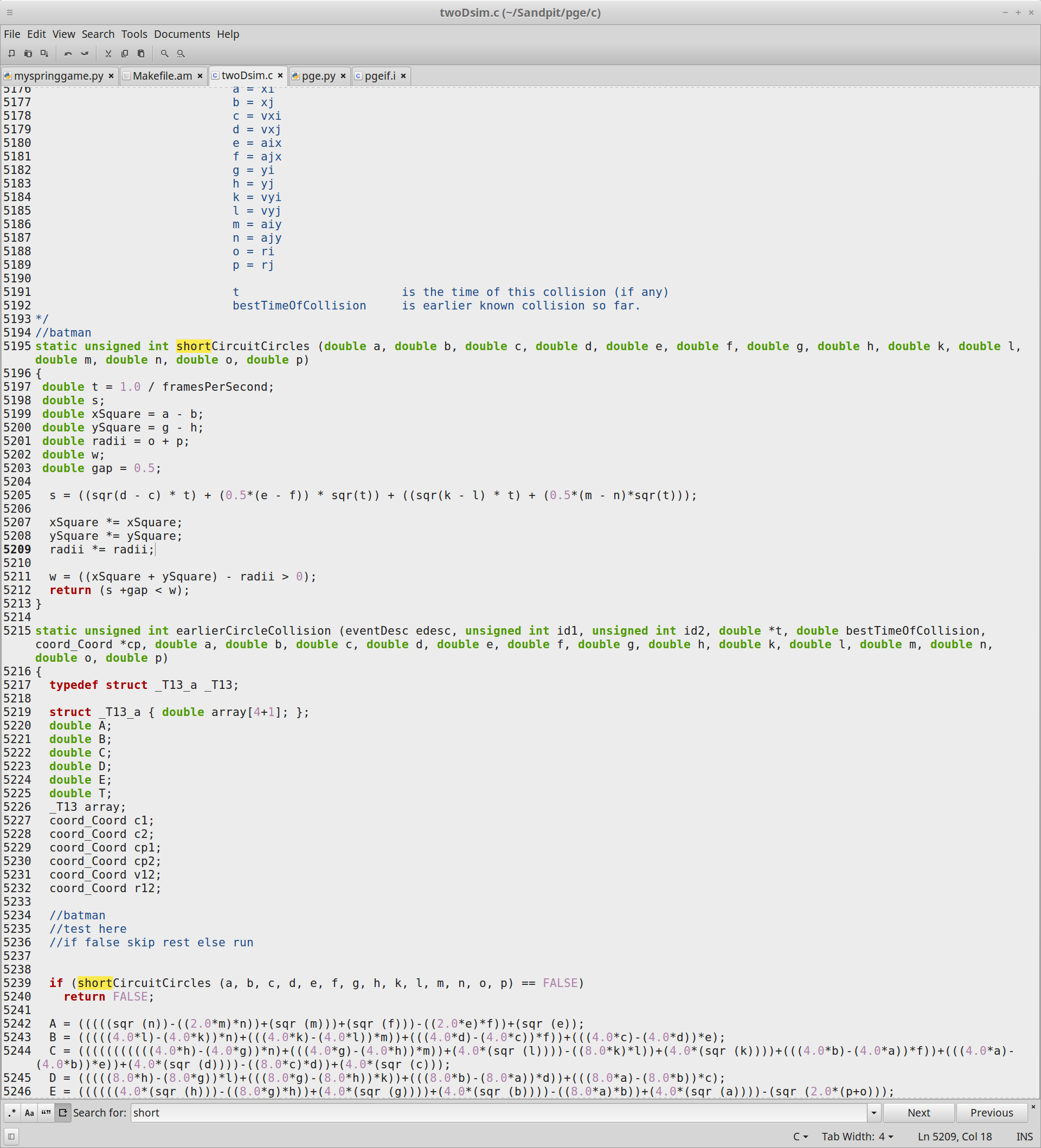
GtwoDsim.h:

The header file for twoDsim.c that defines all external functions. The screenshot below shows the code for this file.

Makefile.am:

This file has been edited for optimisation purposes I have included the lbprofiler into the file to include some optimisation in this file. I have also edited the optimisation flags multiple times to get varying degrees optimisation results as well as using fast math to increase the speed of the calculations in the code. The screenshot below shows the code edited in this file.

Bugfixes:

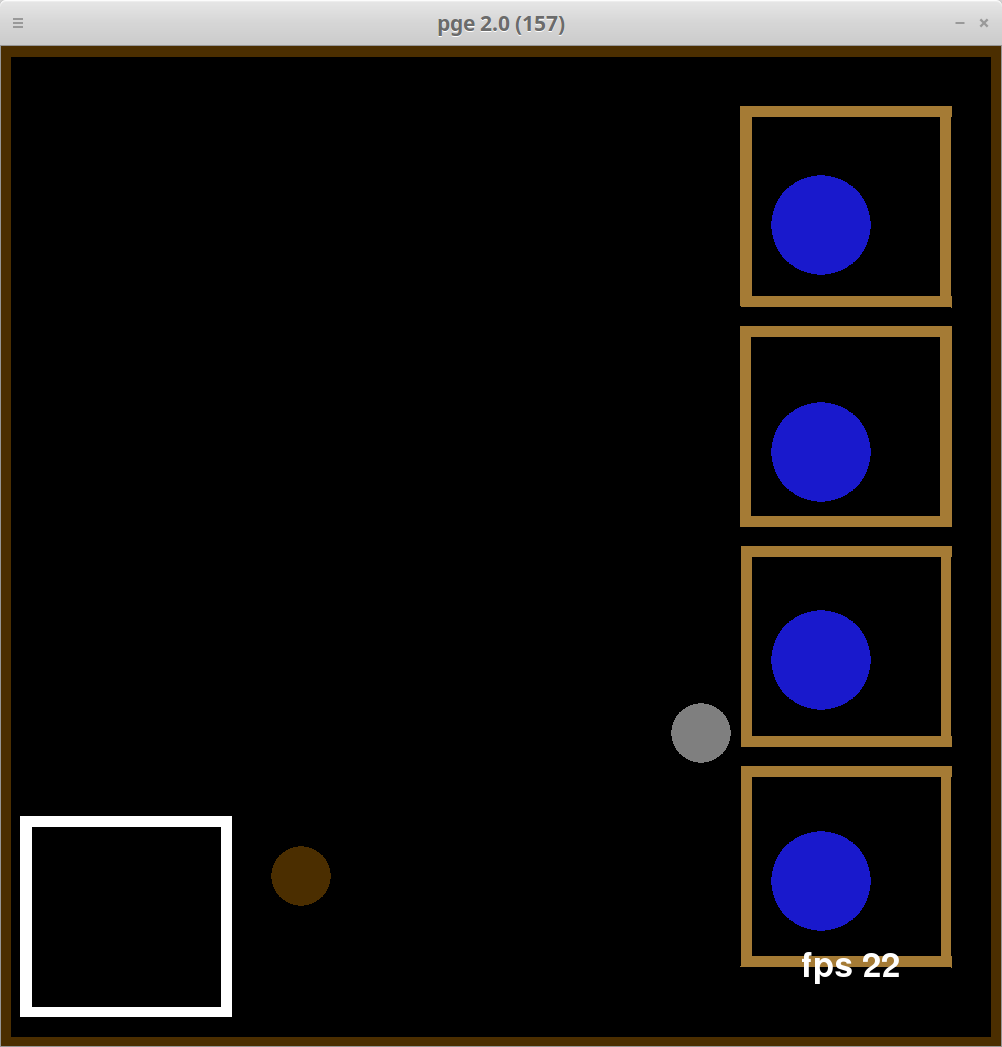
In the twoDsim.c file it required having a slight bug fix on the shortCiruitCircles method that adds a short gap between the circle collision detection which is used to account for some errors incurred by this method

Test Cases: -

The test cases I have created for my program involves using multiple ways and levels of optimisation to check how the fps of my program is affected for each one. The screenshots below show how much fps each level of optimisation can achieve and will be compared to see which is better to use over another.

Standard Results:

To show the differences between the results of the different types of optimisation used I am going to include an unedited screenshot version of my game in this section.

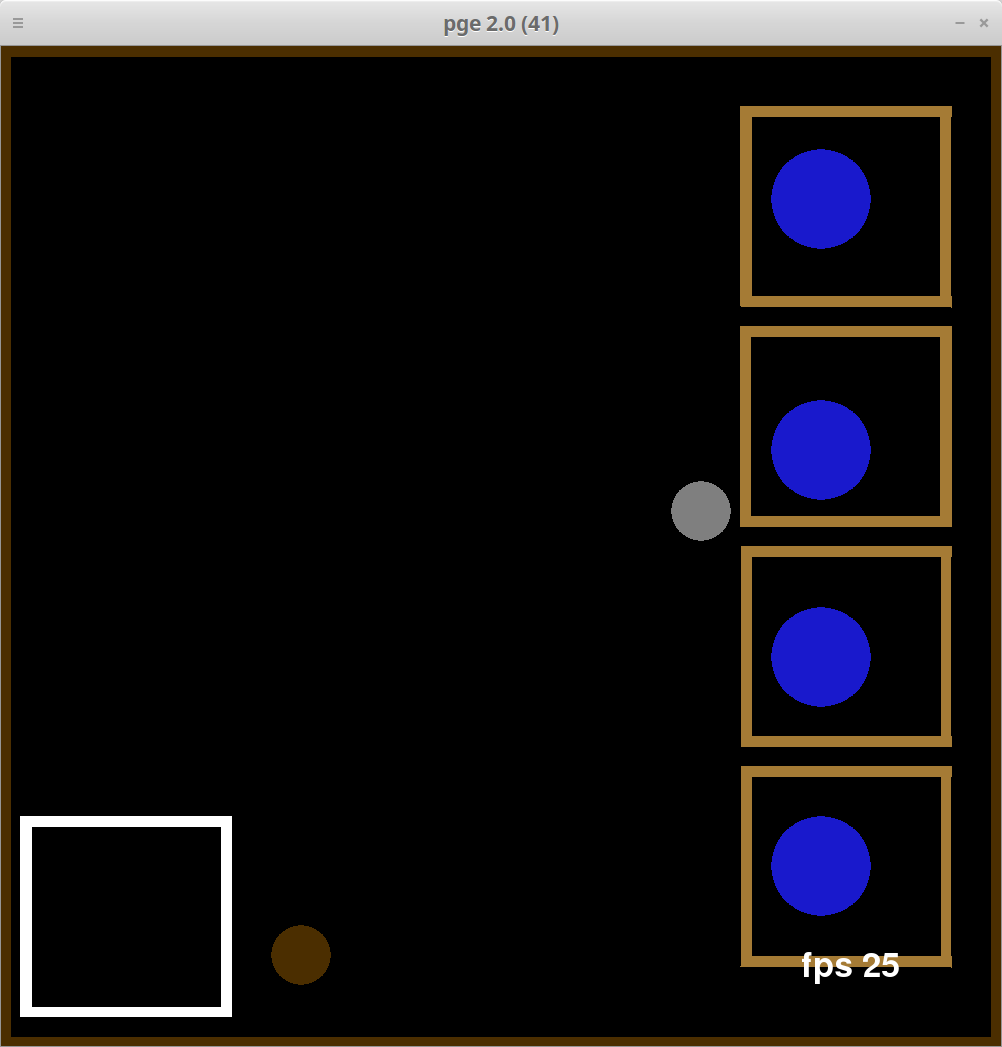


As the screenshot shows the game that has been created which involves using a combination of springs, bungees and resets in order to break the blue balls out of the boxes with the silver ball and move them over to the white box which removes them from the game. Once all the blue balls have been removed the game will display that you have won and close. A time limit for the game has not been set as the difficulty of the game depends on whether the silver ball decides to mess you around as it is hard to hit and can be difficult to get bouncing again if it has slowed down considerably.

In order to run the tests in the next part of the code there are commands that are needed to run the next parts of the code is the command to reconfigure the make file so that these new tests can be performed and run the next thing is the program has to be remade using the make command.

Fast Math Results:

The results in this section show what would happen to the program if it were to be run with fast math enabled. Fast math speeds up the calculations of algorithms with more efficient code use.



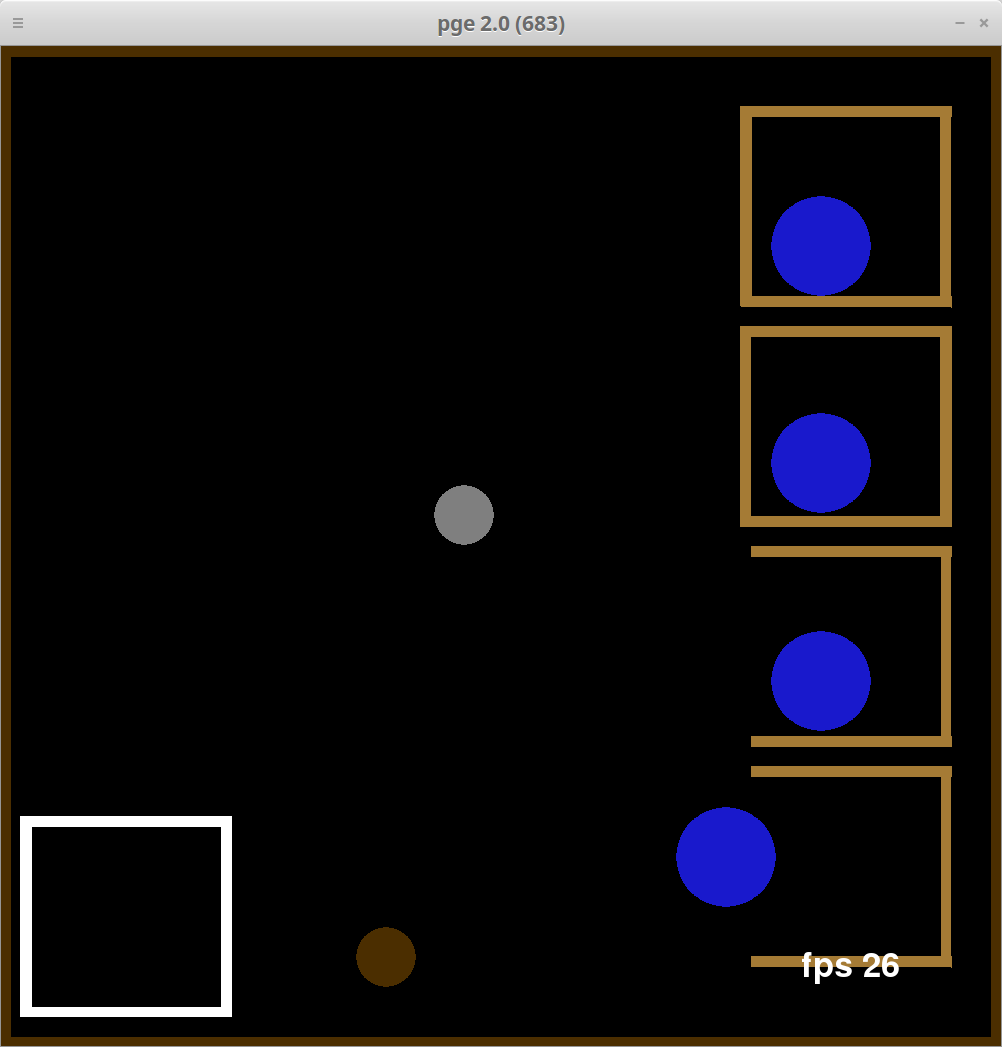
As the screenshot shows using fast math optimisation on the game gives it a few extra fps at the beginning and keeps it from slowing down quite as often as usual.

Optimisation flags:

Optimisation flags are used to perform varying levels of optimisation on certain aspects during compilation these flags do have downsides in other areas however, but they can be used quite effectively to produce a better output from the code. For example, o0 optimisation flag focuses on optimising the execution time and size of the code with downsides to memory usage and compile time and o1 to o3 have downsides for execution time but have varying levels of upsides for memory usage and compile time.

O0 optimisation flags Results:

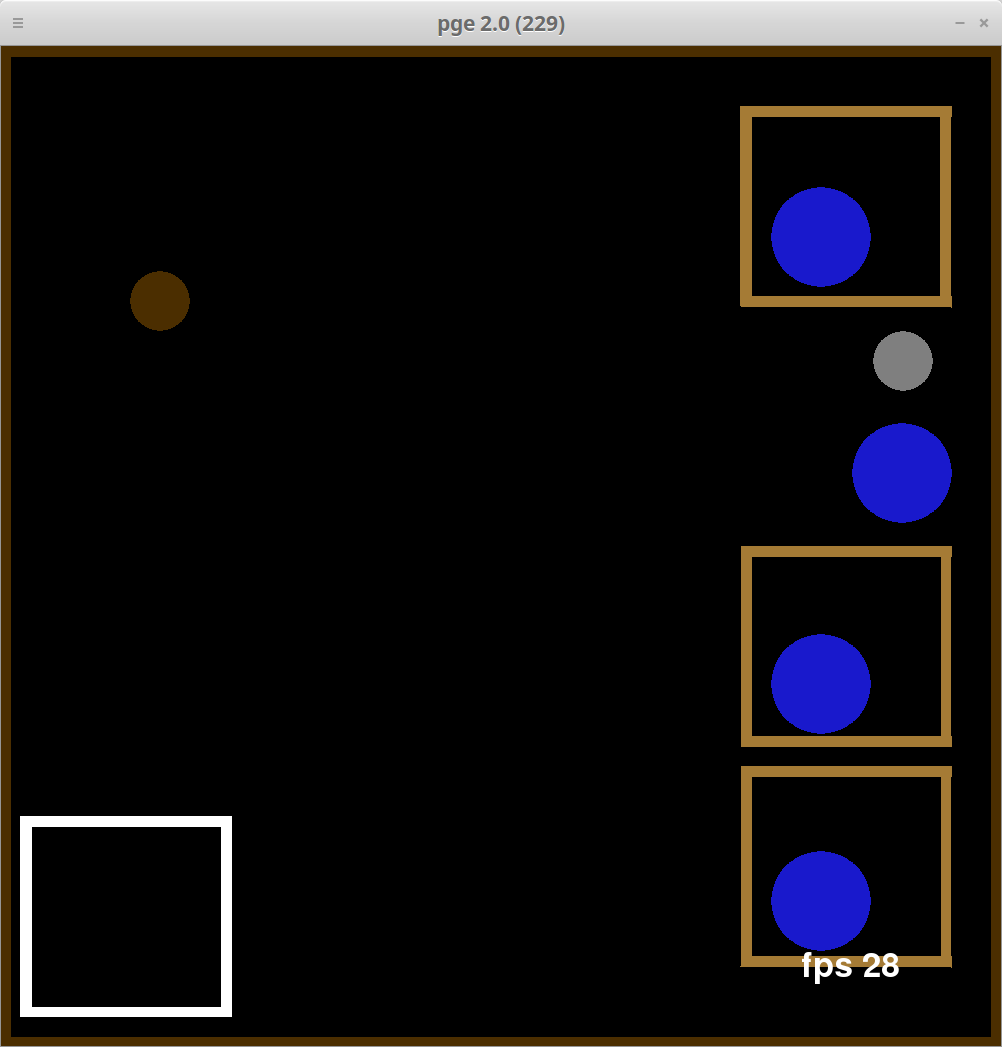
The results in this section show what would happen to the program if it were to be run with o0 optimisation



As the screenshot shows the fps maximum increases slightly but still tends to dip below 20fps as often as the original. It also runs a bit more smoothly but not by a lot.

O1 optimisation flags Results:

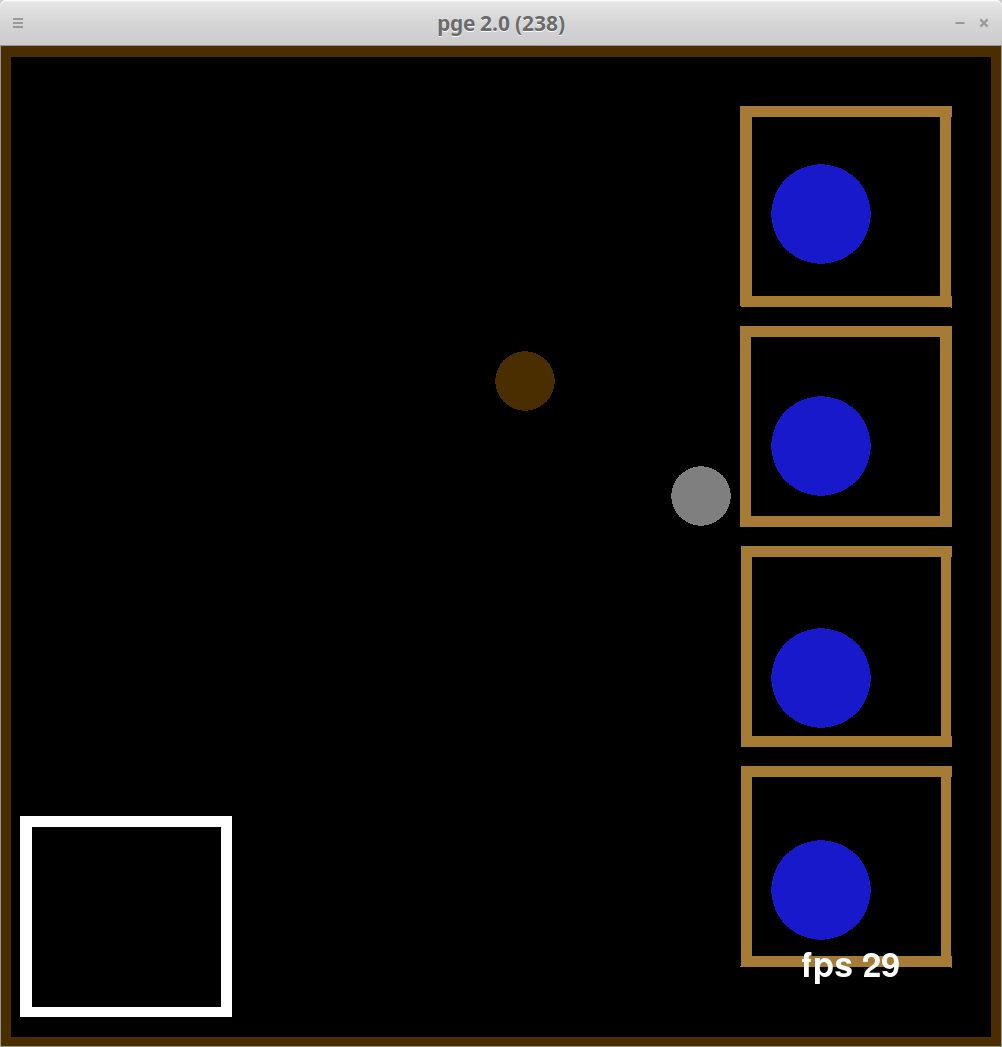
The results in this section show what would happen to the program if it were to be run with o1 optimisation



As this screenshot shows the max fps increases a bit more again and it only tends to dip under 20 fps a bit less often than the previous two tests.

O2 optimisation flags Results:

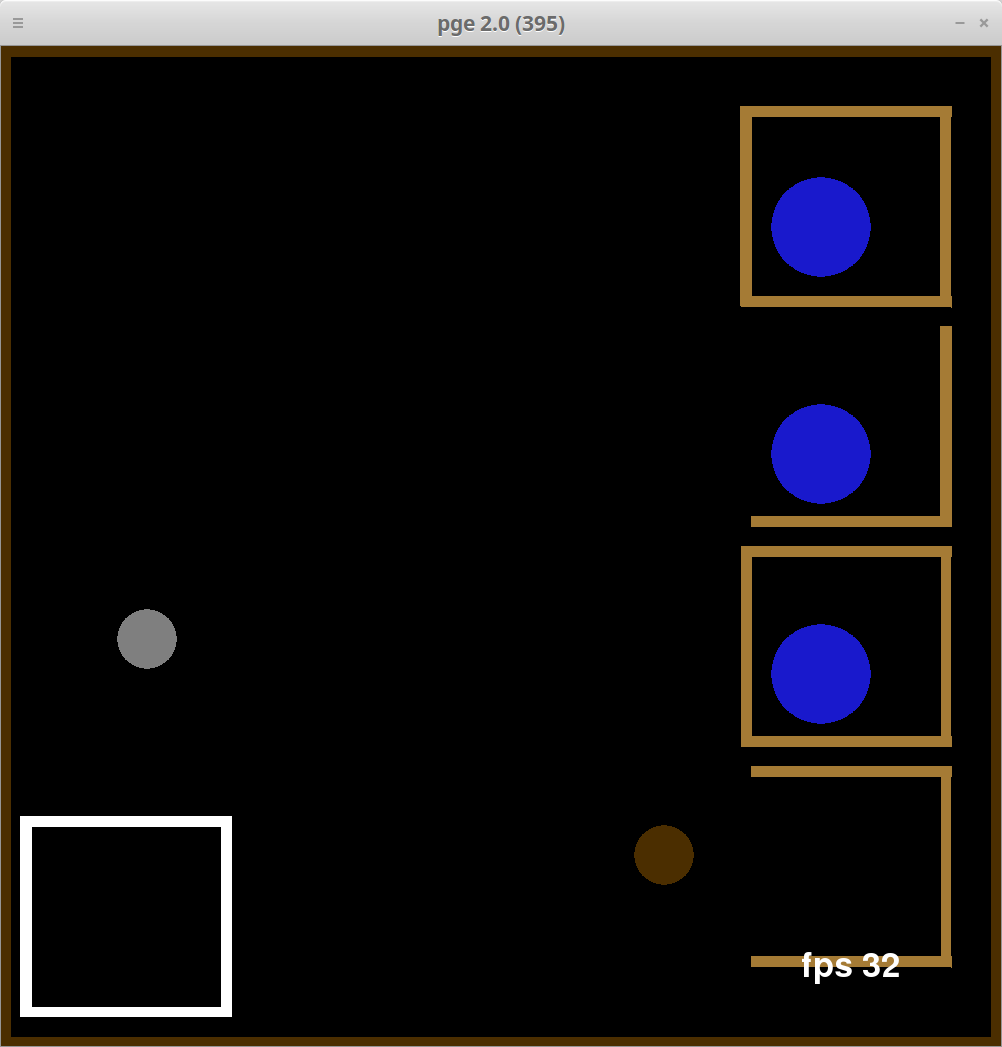
The results in this section show what would happen to the program if it were to be run with o2 optimisation



As the screenshot shows the maximum fps has increased yet again if only slightly and the fps dips under 20 fps a lot less often than the previous tests as it appears to be noticeably smoother during runtime.

O3 optimisation flags Results:

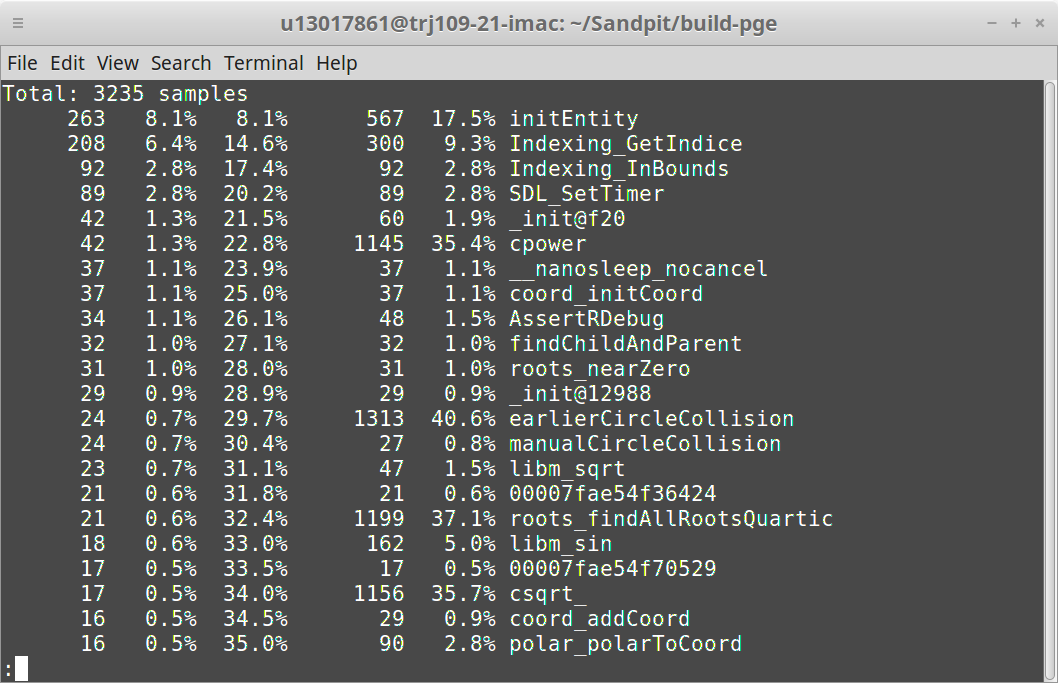
The results in this section show what would happen to the program if it were to be run with o3 optimisation



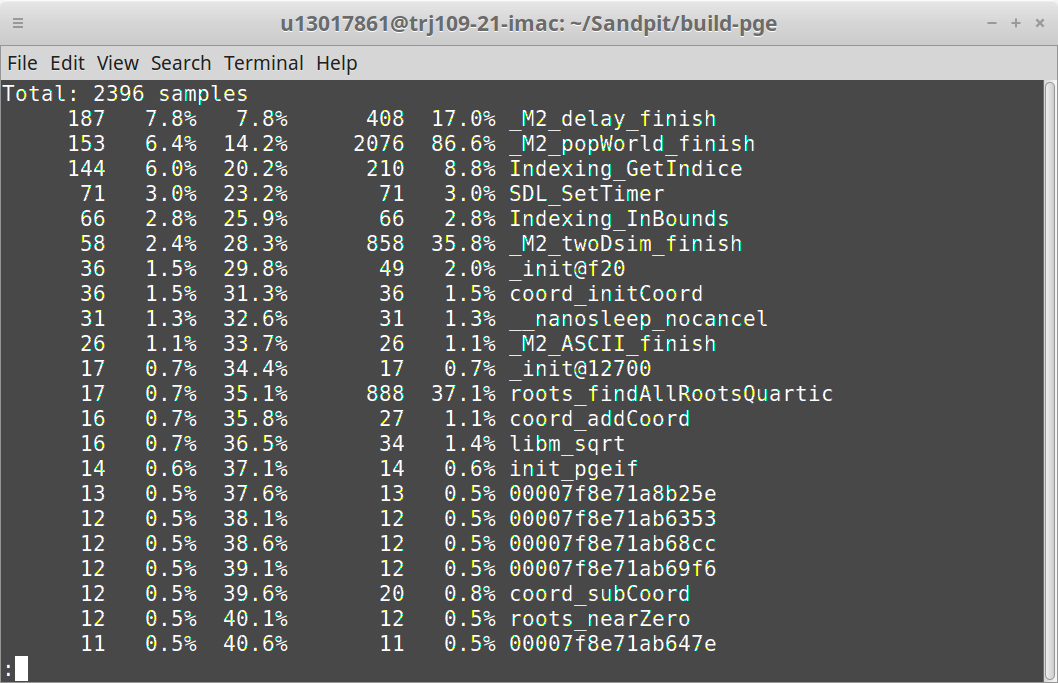
As the screenshot shows the maximum fps increases again slightly and the fps hardly ever dips below 22 fps also the game seems to run a lot more smoothly than before in all areas.

Profiling Libraries (-lprofiler):

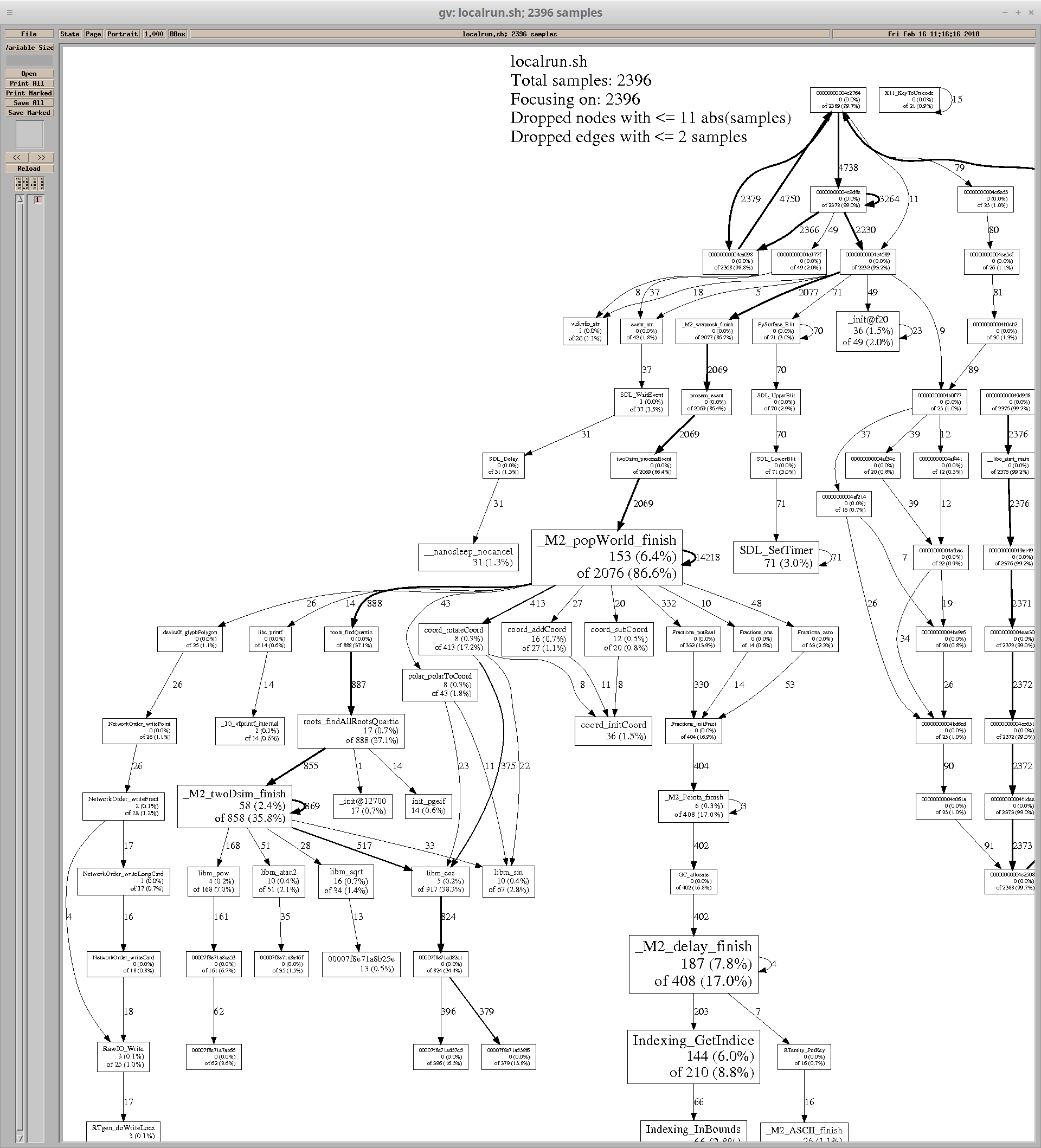
The for this test the code will be run with a profiling library the test will be run twice to get a more accurate reading on which functions are being run the most and for the longest. The profiling library was added into the code by editing Makfile.am and adding -lprofiler into the code. Then the program was reconfigured and remade and using the command line command CPUPROFILE\_FREQUENCY=10000 LD\_PRELOAD=/usr/lib/libprofiler.so CPUPROFILE = dump.txt ./localrun.sh) ../pge/examples/springs/myspringgame.py . This command runs the game and creates a text file called dump.txt\_SomeRandomNumber this file can then be opened with one of two commands either google-pprof -–text localrun.sh dump.txt\_SomeRandomNumber | less or google-pprof -–gv localrun.sh dump.txt\_SomeRandomNumber | less.. These commands allow the file to be output into a readable format via either a text format that displays which methods are used most often and for how long from top to bottom.



As the library file shows on the first run the methods at the top are the methods that run for the longest over the entire program shown in the percentages on the left. On the right the percentages show how much time it takes to run through the method. As the results of this show the method that takes up most of the time according to this run is the initEntity method however this file would be hard to optimise. The other thing to note is that the percentage on the right shows that the cpower and csqrt\_ math functions take 70% of the time to run through both so each of these could use some optimisation to reduce the amount of time they need to run.



According to the second run of the library profiler the \_M2\_delay\_finish function runs the most throughout the program and \_M2\_popWorld\_finish takes 86% of the time to run through it. Unfortunately, these files are not in a position to be optimised

The image above shows the information from the second test in a graphical format as a function tree it shows all the functions run over the course of the program as well as length of time each of the functions is run. The larger of the boxes on this tree are the functions with the most impact on the run.

Appendix (my game code):

#!/usr/bin/env python

import pge, sys, pygame, time

from pygame.locals import \*

print "starting exampleBoxes"

#pge.batch ()

pge.interactive ()

t = pge.rgb (1.0/2.0, 2.0/3.0, 3.0/4.0)

wood\_light = pge.rgb (166.0/256.0, 124.0/256.0, 54.0/256.0)

wood\_dark = pge.rgb (76.0/256.0, 47.0/256.0, 0.0)

red = pge.rgb (1.0, 0.0, 0.0)

green = pge.rgb (0.0, 1.0, 0.0)

blue = pge.rgb (0.0, 0.0, 1.0)

blue\_dark = pge.rgb (0.1, 0.1, 0.8)

steel = pge.rgb (0.5, 0.5, 0.5)

copper = pge.rgb (0.5, 0.3, 0.2)

gold = pge.rgb (0.8, 0.6, 0.15)

ball\_size = 0.02

boarder = 0.01

white = pge.rgb (1.0, 1.0, 1.0)

gap = 0.01

captured = None

sides = []

winner = False

yellow = pge.rgb (0.8, 0.6, 0.15)

fps\_text = None

last\_fps = 0

spring\_power = 1000.0

damping = 10.0

snap\_length = 0.16

snap\_length2 = 0.0000001

def myquit (e):

print "goodbye"

sys.exit (0)

def key\_pressed (e):

if e.key == K\_ESCAPE:

myquit (e)

def placeBoarders (thickness, color):

print "placeBoarders"

e1 = pge.box (0.0, 0.0, 1.0, thickness, color).fix ()

e2 = pge.box (0.0, 0.0, thickness, 1.0, color).fix ()

e3 = pge.box (1.0-thickness, 0.0, thickness, 1.0, color).fix ()

e4 = pge.box (0.0, 1.0-thickness, 1.0, thickness, color).fix ()

return e1, e2, e3, e4

def rescueBox (thickness, pos, width, color):

global safe

newf = pge.box (pos[0], pos[1], width, thickness, color).fix ()

newl = pge.box (pos[0]-0.01, pos[1], thickness, width, color).fix ()

newr = pge.box (pos[0]+width-thickness, pos[1], thickness, width, color).fix ()

newt = pge.box (pos[0]-0.01, pos[1]+width-thickness, width, thickness, color).fix ()

safe = [newf ,newl ,newr, newt]

def box\_of (thickness, pos, width, color):

global captured, sides, ally, safe

floor = pge.box (pos[0], pos[1], width, thickness, color).fix ()

left = pge.box (pos[0]-0.01, pos[1], thickness, width, color).fix ()

right = pge.box (pos[0]+width-thickness, pos[1], thickness, width, color).fix ()

top = pge.box (pos[0], pos[1]+width-thickness, width, thickness, color).fix ()

floor2 = pge.box (pos[0], pos[1]+0.22, width, thickness, color).fix ()

left2 = pge.box (pos[0]-0.01, pos[1]+0.22, thickness, width, color).fix ()

right2 = pge.box (pos[0]+width-thickness, pos[1]+0.22, thickness, width, color).fix ()

top2 = pge.box (pos[0], pos[1]+width-thickness+0.22, width, thickness, color).fix ()

floor3 = pge.box (pos[0], pos[1]+0.44, width, thickness, color).fix ()

left3 = pge.box (pos[0]-0.01, pos[1]+0.44, thickness, width, color).fix ()

right3 = pge.box (pos[0]+width-thickness, pos[1]+0.44, thickness, width, color).fix ()

top3 = pge.box (pos[0], pos[1]+width-thickness+0.44, width, thickness, color).fix ()

floor4 = pge.box (pos[0], pos[1]+0.66, width, thickness, color).fix ()

left4 = pge.box (pos[0]-0.01, pos[1]+0.66, thickness, width, color).fix ()

right4 = pge.box (pos[0]+width-thickness, pos[1]+0.66, thickness, width, color).fix ()

top4 = pge.box (pos[0], pos[1]+width-thickness+0.66, width, thickness, color).fix ()

sides = [floor, left, right, top, floor2, left2, right2, top2, floor3, left3, right3, top3, floor4, left4, right4, top4]

captured = placeBall (blue\_dark, pos[0]+2.0\*thickness+0.05, pos[1]+1.0\*thickness+0.05, 0.05).mass(1.5).on\_collision\_with (safe, delete\_me)

captured2 = placeBall (blue\_dark, pos[0]+2.0\*thickness+0.05, pos[1]+1.0\*thickness+0.25, 0.05).mass(1.5).on\_collision\_with (safe, delete\_me)

captured3 = placeBall (blue\_dark, pos[0]+2.0\*thickness+0.05, pos[1]+1.0\*thickness+0.45, 0.05).mass(1.5).on\_collision\_with (safe, delete\_me)

captured4 = placeBall (blue\_dark, pos[0]+2.0\*thickness+0.05, pos[1]+1.0\*thickness+0.65, 0.05).mass(1.5).on\_collision\_with (safe, delete\_me)

ally = [captured, captured2, captured3, captured4]

def placeBall (kind, x, y, r):

return pge.circle (x, y, r, kind)

def first ():

global right, step

right = placeBall (wood\_light, 0.75, 0.45, 0.03).fix ()

step = placeBall (wood\_dark, 30 / 100.0, 0.33, 0.03).mass (0.9).on\_collision\_with(sides, delete\_it)

s = pge.spring (right, step, spring\_power, damping, 0.1).draw (yellow, 0.002)

s.when (snap\_length, snap\_it)

def snap\_it (e, o):

global right

o.rm ()

right.rm ()

def update\_fps (e, o):

global last\_fps, fps\_text

fn = pge.get\_frame\_no ()

s = "fps %d" % (fn - last\_fps)

if fps\_text != None:

fps\_text.rm ()

fps\_text = pge.text (0.8, 0.1, s, white, 50, 1)

last\_fps = fn

local\_fps ()

def timer():

start = time.time()

time.clock()

elapsed = 0

while elapsed:

elapsed = time.time() - start

timer\_text = pge.text (0.8, 0.7, elapsed, white, 50, 1)

time.sleep(1)

def drop\_gb (e, o):

gb = placeBall (steel, 0.7, 0.92, 0.03).mass (2.0).on\_collision\_with(sides, delete\_it)

#pge.at\_time (3.0, drop\_gb)

def reset\_ball (e,o):

global step , right, pos

step.rm ()

right.rm ()

step = placeBall (wood\_dark, 30 / 100.0, 0.33, 0.03).mass (0.9)

#s.remove (o) #remember this

def delete\_it (o, e):

global sides, winner, loser

for i in e.collision\_between ():

if i in sides:

i.rm ()

def delete\_me (o, e):

global ally, winner, loser

ally.remove (o)

o.rm ()

if ally == []:

winner = True

pge.text (0.2, 0.3, "Winner", white, 100, 1)

pge.at\_time (4.0, myquit)

def local\_fps ():

f = pge.at\_time (1.0, update\_fps)

def pCollisions ():

step.on\_collision\_with(sides, reset\_ball) #on collision with walls new spring is drawn

def on\_click (e):

global step, right, s

mouse = pge.pyg\_to\_unit\_coord (e.pos)

if e.button == 1:

right.rm ()

right = placeBall (wood\_light, mouse[0], mouse[1], 0.01).fix ()

s = pge.spring (right, step, spring\_power, damping, 0).draw (yellow, 0.002)

s.bungee = 0

s.when (snap\_length, snap\_it)

elif e.button == 3:

right = placeBall (wood\_light, mouse[0], mouse[1], 0.01).fix ()

s = pge.spring (right, step, spring\_power, damping, 0.05).draw (yellow, 0.002)

s.bungee = 1

#s.when (snap\_length, snap\_it)

#pge.at\_time(0.5, gb.rm())

elif e.button == 4:

reset\_ball(step, right)

#s.when (snap\_length2, snap\_it)

#snap\_it#fi this somehow

def main ():

global gb, sides, springs, right, step, pos

placeBoarders (0.01, wood\_dark)

#start\_time = time.time()

#print("--- %s seconds ---" % round(time.time() - start\_time,2)

#left = placeBall (wood\_light, 0.25, 0.45, 0.03).fix ()

#step = placeBall (wood\_dark, 30 / 100.0, 0.33, 0.03).mass (0.9)

#right = placeBall (wood\_light, 0.75, 0.45, 0.03).fix ()

#right.rm ()

first()

#prev = left

springs = []

#pge.at\_time(0.5, gb.rm())

#springs += [s]

#prev = step

#s = pge.spring (right, prev, spring\_power, damping, 0.1).draw (yellow, 0.002)

#s.when (snap\_length, snap\_it)

#pge.at\_time (0.5, drop\_gb)

rescueBox (boarder, [0.03, 0.03], 0.2, white)

box\_of (boarder, [0.75, 0.08], 0.2, wood\_light)

drop\_gb (None, None)

print "before run"

pge.record ()

pge.draw\_collision (True, False)

pge.collision\_colour (red)

pge.gravity ()

pge.dump\_world ()

pge.slow\_down (6.0) # slows down real time by a factor of

pge.register\_handler (myquit, [QUIT])

pge.register\_handler (key\_pressed, [KEYDOWN])

pge.register\_handler (on\_click, [MOUSEBUTTONDOWN])

pge.display\_set\_mode ([1000, 1000])

local\_fps ()

pCollisions ()

timer()

pge.run (30.0)

pge.finish\_record ()

print "before main()"

main ()