

```
1. #!/usr/bin/python
2.
3. from Tkinter import *
4. from PIL import Image
5. from PIL import ImageTk
6. import tkFileDialog
7. import tkMessageBox
8. import re
9. import time
10. import Racecar.Tree
11. import Racecar.Compiler
12. import random
13. import pdb
14. import math
15.
16. random.seed()
17.
18. #current_program ised used to store the current file open in order to save back
19. #to that file
20. current_program = None
21.
22. #Variable that serves as an interrupt to stop the program
23. should_stop = False
24. collision_occurred = False
25.
26. #List of obstacles on the course at any given time
27. obstacles = []
28.
29. #List of walls on the course at any given time
30. walls = []
31.
32. #grid ticks
33. grid_ticks = []
34.
35.
36. class Obstacle:
37.     def __init__(self, x, y, width, height):
38.         self.obstacle_object = canvas.create_oval(
39.             x-width/2,
40.             y-height/2,
41.             x+width/2,
42.             y+height/2,
43.             fill="#000")
44.         self.width = width
45.         self.height = height
46.         self.center = (x, y)
47.         self.radius = width/2
48.
49.
50. #Wall class: can only be vertical or horizontal
51. class Wall:
52.     def __init__(self, start_x, start_y, length, is_horizontal):
53.         if is_horizontal:
54.             self.wall_object = canvas.create_line(
55.                 start_x,
56.                 start_y,
57.                 start_x+length,
58.                 start_y)
59.             self.start = start_x
60.             self.end = start_x+length
61.             self.constant_coord = start_y
62.         else:
63.             self.wall_object = canvas.create_line(
64.                 start_x,
```

```
65.         start_y,
66.         start_x,
67.         start_y+length)
68.     self.start = start_y
69.     self.end = start_y+length
70.     self.constant_coord = start_x
71.     self.is_horizontal = is_horizontal
72.
73.
74. class Program:
75.     def __init__(self):
76.         self.name = ''
77.         self.file_obj = None
78.
79.
80. #Static variables for turning the car
81. class WheelDirection:
82.     LEFT = 1
83.     RIGHT = -1
84.
85.
86. #Car direction object
87. #X and Y can be 1,0,-1 respectively. The only invalid combination is when x = 0
88. #and y = 0. Positive axes point right and up respectively
89. class CarDirection:
90.     FORWARDS = 1
91.     BACKWARDS = -1
92.
93.     def __init__(self):
94.         self.direction = 0
95.
96.     DIRECTIONS = [(1, 0),
97.                   (1, -1),
98.                   (0, -1),
99.                   (-1, -1),
100.                  (-1, 0),
101.                  (-1, 1),
102.                  (0, 1),
103.                  (1, 1)]
104.
105.     def get_direction(self):
106.         return CarDirection.DIRECTIONS[self.direction]
107.
108.     def turn_right(self):
109.         self.direction = (self.direction - 1) % len(CarDirection.DIRECTIONS)
110.
111.     def turn_left(self):
112.         self.direction = (self.direction + 1) % len(CarDirection.DIRECTIONS)
113.
114.     def opposite_direction(self):
115.         return DIRECTIONS[
116.             (self.direction + len(CarDirection.DIRECTIONS)/2) %
117.             len(CarDirection.DIRECTIONS)]
118.
119.
120. class Car:
121.     def __init__(self):
122.         self.position_x = 0
123.         self.position_y = 0
124.         #Car direction starts facing right
125.         self.car_direction = CarDirection()
126.         self.image = None
127.         self.image_tk = None
128.         self.car_object = None
129.         self.width = 27
```

```
130.         self.height = 27
131.         self.radius = 25
132.
133.         #Drive method that updates the car's position (in the model, not on the UI)
134.         #UI animation will need to be done moving x and y simultaneously
135.         def update_position(self, steps, movement_direction):
136.             self.position_x += (
137.                 self.car_direction.get_direction()[0]
138.                 * steps
139.                 * movement_direction)
140.
141.             self.position_y += (
142.                 self.car_direction.get_direction()[1]
143.                 * steps
144.                 * movement_direction)
145.
146.
147.         #Function to get a unique position of object, in order to detect for collisions
148.         def get_position(x, y):
149.             return 1000 * int(x) + int(y)
150.
151.
152.         def getCurrentPosition():
153.             global car
154.             return get_position(car.position_x, car.position_y)
155.
156.
157.         #Checks if there is going to be a collision on the upcoming path
158.         #drive_direction has to be CarDirection.FORWARDS or CarDirection.BACKWARDS
159.         def can_move(num_steps, drive_direction):
160.             global car
161.             curr_x = int(car.position_x)
162.             curr_y = int(car.position_y)
163.             direction = car.car_direction.get_direction()
164.             path = []
165.
166.             #If the direction is backwards, just reverse the direction
167.             if drive_direction == CarDirection.BACKWARDS:
168.                 direction = car.car_direction.opposite_direction()
169.
170.             #Create path coordinates
171.             for i in range(0, steps_to_pixels(num_steps)):
172.                 pos = (curr_x + i * direction[0], curr_y + i * direction[1])
173.                 path.append(pos)
174.
175.             #Check each point in the path to see if it collides with any of the
176.             #obstacles
177.             for pos in path:
178.                 if is_collision(pos[0], pos[1]):
179.                     return False
180.
181.             return True
182.
183.
184.         #Number of steps on screen is proportional to screen size
185.         def steps_to_pixels(steps):
186.             return canvas_frame.winfo_reqwidth()/110*steps
187.
188.
189.         #Function to find the distance between two points
190.         def distance_between_points(x_1, y_1, x_2, y_2):
191.             return math.sqrt(math.pow((x_2-x_1), 2) + math.pow((y_2-y_1), 2))
192.
193.
194.         #API Functions
```

```
195. #direction must be either CarDirection.FORWARDS or CarDirection.BACKWARDS
196. def translate_car(steps, direction):
197.     global car
198.     global should_stop
199.     global collision_occurred
200.
201.     steps = int(steps)
202.     direction = int(direction)
203.
204.     curr_x = car.position_x
205.     curr_y = car.position_y
206.
207.     one_step = steps_to_pixels(1)
208.
209.     for i in range(0, steps_to_pixels(int(steps))):
210.         #Check interrupt variable
211.         if should_stop and i % one_step == 0:
212.             return
213.
214.         time.sleep(0.01)
215.         #car_direction is FORWARDS or BACKWARDS (1 and -1 respectively)
216.
217.         if is_collision(curr_x, curr_y):
218.             print_to_console("COLLISION")
219.             #Stop execution of program
220.             #TODO Deal with delay on collision
221.             should_stop = True
222.             collision_occurred = True
223.             return
224.         else:
225.             canvas.move(
226.                 car.car_object,
227.                 direction * car.car_direction.get_direction()[0],
228.                 direction * car.car_direction.get_direction()[1])
229.
230.             curr_x = curr_x + direction * car.car_direction.get_direction()[0]
231.             curr_y = curr_y + direction * car.car_direction.get_direction()[1]
232.             canvas.update()
233.
234.             car.update_position(1, direction)
235.
236.
237. #direction must be WheelDirection.LEFT or WheelDirection.RIGHT
238. #Note: only check interrupt variable at the beginning, because
239. #we shouldn't allow partial rotations
240. def rotate_car(direction):
241.     global car
242.     global should_stop
243.
244.     #Check interrupt variable
245.     if should_stop:
246.         return
247.
248.     #This is current index in DIRECTIONS array
249.     current_direction_deg = car.car_direction.direction*45
250.
251.     if direction == WheelDirection.LEFT:
252.         car.car_direction.turn_left()
253.     elif direction == WheelDirection.RIGHT:
254.         car.car_direction.turn_right()
255.     else:
256.         return
257.
258.     for i in range(0, 45):
259.         time.sleep(0.01)
```

```
260. canvas.delete(car.car_object)
261.
262. if direction == WheelDirection.LEFT:
263.     car.image_tk = ImageTk.PhotoImage(
264.         car.image.rotate(current_direction_deg + i))
265. elif direction == WheelDirection.RIGHT:
266.     car.image_tk = ImageTk.PhotoImage(
267.         car.image.rotate(current_direction_deg - i))
268. else:
269.     return
270.
271. car.car_object = canvas.create_image(
272.     car.position_x,
273.     car.position_y,
274.     image=car.image_tk)
275. canvas.update()
276.
277.
278. #Check for collision with walls of the maze and a finish line
279. def collision_with_internal_walls(pos_x, pos_y):
280.     for wall in walls:
281.         #horizontal wall
282.         if wall.is_horizontal:
283.             #in range of wall
284.             if wall.start <= pos_x <= wall.end:
285.                 #Current direction of the car
286.                 direction = car.car_direction.get_direction()
287.                 #distance from wall
288.                 dist_to_wall = math.fabs(pos_y-wall.constant_coord)
289.                 #Car is horizontally oriented
290.                 if direction == (1, 0) or direction == (-1, 0):
291.                     if dist_to_wall < car.radius/2:
292.                         return True
293.                 #Check for collision with car
294.                 #Car is not horizontally oriented
295.                 else:
296.                     if dist_to_wall < car.radius:
297.                         return True
298.             #vertical wall
299.             else:
300.                 #in range of wall
301.                 if wall.start <= pos_y <= wall.end:
302.                     direction = car.car_direction.get_direction()
303.                     #distance from wall
304.                     dist_to_wall = math.fabs(pos_x-wall.constant_coord)
305.                     #Car is vertically oriented
306.                     if direction == (0, 1) or direction == (0, -1):
307.                         if dist_to_wall < car.radius/2:
308.                             return True
309.                     #Car is not vertically oriented
310.                     else:
311.                         if dist_to_wall < car.radius:
312.                             return True
313.
314.     return False
315.
316.
317. def is_collision(curr_x, curr_y):
318.     #Check for collisions with obstacles and walls
319.
320.     #Check obstacles
321.     for obstacle in obstacles:
322.         distance = distance_between_points(
323.             curr_x,
324.             curr_y,
```

```
325.         obstacle.center[0],
326.         obstacle.center[1])
327.         if distance < (car.radius + obstacle.radius):
328.             return True
329.         #check internal walls for collision
330.         if collision_with_internal_walls(curr_x, curr_y):
331.             return True
332.         #Check boundary walls
333.         elif not (origin[0] <= curr_x <= anti_origin[0]):
334.             return True
335.         elif not (origin[1] <= curr_y <= anti_origin[1]):
336.             return True
337.         else:
338.             return False
339.
340.
341. def print_to_console(message):
342.     #Should console be cleared each time the program is restart?
343.     #Or should there be a button?
344.     console.config(state=NORMAL)
345.     console.insert(END, str(message) + '\n')
346.     console.config(state=DISABLED)
347.
348.
349. #Course generation functions
350.
351. #Course one is a slalom of blocks
352. def course_one():
353.     clear_course()
354.     obstacle_coord_x = 123
355.     obstacle_coord_y = int(canvas.winfo_reqheight())/2
356.     while obstacle_coord_x < anti_origin[0]:
357.         obstacle = Obstacle(obstacle_coord_x, obstacle_coord_y, 30, 30)
358.         obstacles.append(obstacle)
359.         obstacle_coord_x = obstacle_coord_x + 150
360.
361.
362. #TODO -- Fill in the rest of the courses
363. #Course two is a simple maze
364. finish_line = None
365.
366.
367. def course_two():
368.     global finish_line
369.
370.     clear_console()
371.
372.     message = "Try to navigate through the maze and cross the finish line!"
373.     print_to_console(message)
374.
375.     clear_course()
376.     wall_coord_x = 123
377.     wall_length = 4*int(canvas.winfo_reqheight())/5
378.
379.     #used to toggle position of line
380.     put_wall_on_top = True
381.
382.     #walls
383.     while wall_coord_x < anti_origin[0]:
384.         if put_wall_on_top:
385.             wall = Wall(
386.                 wall_coord_x,
387.                 0,
388.                 wall_length,
389.                 False)
```

```
390.         walls.append(wall)
391.     else:
392.         wall = Wall(
393.             wall_coord_x,
394.             int(canvas.wininfo_reqheight())/5+23,
395.             wall_length,
396.             False)
397.         walls.append(wall)
398.         put_wall_on_top = not put_wall_on_top
399.         wall_coord_x = wall_coord_x+100
400.
401.     #finish line
402.     wall_coord_x = wall_coord_x-100
403.     finish_line = canvas.create_line(
404.         wall_coord_x,
405.         wall_length,
406.         wall_coord_x,
407.         canvas.wininfo_reqheight()+23,
408.         fill="black",
409.         dash=(4, 4))
410.
411.
412. def course_three():
413.     clear_course()
414.
415.     max_x = canvas.wininfo_reqwidth()
416.     max_y = canvas.wininfo_reqheight()
417.
418.     while len(obstacles) < 30:
419.         pos_x = random.randrange(0, max_x, 1)
420.         pos_y = random.randrange(0, max_y, 1)
421.         radius = random.randrange(10, 50, 1)
422.
423.         if is_collision(pos_x, pos_y):
424.             continue
425.         #Check for collision with car
426.         elif distance_between_points(
427.             pos_x,
428.             pos_y,
429.             car.position_x,
430.             car.position_y) < (car.radius + radius):
431.             continue
432.         else:
433.             obstacle = Obstacle(pos_x, pos_y, radius, radius)
434.             obstacles.append(obstacle)
435.
436.
437. def course_four():
438.     clear_course()
439.     obstacle_coord_x = 123
440.     obstacle_coord_y = 60
441.     while obstacle_coord_y < anti_origin[1]:
442.         while obstacle_coord_x < anti_origin[0]:
443.             obstacle = Obstacle(obstacle_coord_x, obstacle_coord_y, 30, 30)
444.             obstacles.append(obstacle)
445.             obstacle_coord_x = obstacle_coord_x + 150
446.             obstacle_coord_y = obstacle_coord_y + 80
447.             obstacle_coord_x = 123
448.
449.     for obstacle in obstacles:
450.         print obstacle.center
451.
452.
453. def clear_course():
454.     global obstacles
```

```
455.     global walls
456.     global finish_line
457.     #remove obstacles from the course
458.     for obstacle in obstacles:
459.         canvas.delete(obstacle.obstacle_object)
460.
461.     #Needs to take care of finish line too, which isn't a wall object
462.     for wall in walls:
463.         canvas.delete(wall.wall_object)
464.
465.     if finish_line is not None:
466.         canvas.delete(finish_line)
467.     #clear the obstacles and walls array
468.     obstacles = []
469.     walls = []
470.     finish_line = None
471.
472.
473. #Menu functions
474. def open_file():
475.     global current_program
476.
477.     #Keep returning to the file dialog if they didn't select a .race file
478.     while True:
479.         file_name = tkFileDialog.askopenfilename(defaultextension=".race")
480.         if file_name == '':
481.             return
482.
483.         #Check validity of file being opened
484.         file_regex = re.compile("\w*\..race$")
485.         if len(file_regex.findall(file_name)) == 0:
486.             tkMessageBox.showwarning(
487.                 "Open File Error",
488.                 "You must open a .race file")
489.         else:
490.             break
491.
492.         file_object = open(file_name, 'r')
493.         current_program = Program()
494.         current_program.name = file_name
495.         current_program.file_obj = file_object
496.         code.delete(1.0, END)
497.         code.insert(1.0, file_object.read())
498.         current_program.file_obj.close()
499.
500.
501. def save():
502.     global current_program
503.     if current_program is None:
504.         save_file_as()
505.     else:
506.         save_file()
507.
508.
509. def save_file():
510.     global current_program
511.     if not current_program.file_obj.closed:
512.         current_program.file_obj.close()
513.     #Open file for writing (will clear it)
514.     current_program.file_obj = open(current_program.name, 'w')
515.     current_program.file_obj.truncate()
516.     current_program.file_obj.write(code.get(1.0, END))
517.     current_program.file_obj.close()
518.
519.
```



```
520. def save_file_as():
521.     global current_program
522.     file_name = tkFileDialog.asksaveasfilename(defaultextension=".race")
523.
524.     #Defaults to saving on the desktop
525.     if file_name == '':
526.         file_name = '~/Desktop/racecar_program.race'
527.
528.     current_program = Program()
529.     current_program.name = file_name
530.     current_program.file_obj = open(file_name, 'w')
531.     current_program.file_obj.write(code.get(1.0, END))
532.     current_program.file_obj.close()
533.
534.
535. def clear():
536.     if code.get(1.0, END) == '':
537.         return
538.
539.     if tkMessageBox.askyesno(
540.         "Clear code",
541.         "Are you sure you want to delete all of your code?"):
542.         code.delete(1.0, END)
543.
544.
545. def clear_console():
546.     console.config(state=NORMAL)
547.     console.delete(1.0, END)
548.     console.config(state=DISABLED)
549.
550.
551. #Triggers interrupt
552. def stop_program():
553.     global should_stop
554.     should_stop = True
555.
556.
557. #Code generation and compilation
558. #Runs code
559. def generate_program(code):
560.     global should_stop
561.     global collision_occurred
562.
563.     #Set the interrupt variable whenever a program is run
564.     should_stop = False
565.     collision_occurred = False
566.     if len(code) > 1:
567.         #print code[:-1]
568.         #demo(code)
569.         python_code, errors, correct = verify_program(code)
570.         if(correct):
571.             #Print message to console saying program is executing
572.             print_to_console("Program executing")
573.             console.tag_add("Correct", "1.0", "1.end")
574.             console.tag_config("Correct", foreground="Green")
575.
576.             #Toggle the buttons on the bottom and run program
577.             toggle_buttons(True)
578.             tempGlobal = globals().copy()
579.             exec(python_code, tempGlobal)
580.             toggle_buttons(False)
581.
582.             #If collision occurred
583.             if should_stop:
584.                 if collision_occurred:
```

```

585.         if tkMessageBox.showwarning(
586.             "Oops!", "You crashed! Try again"):
587.             reset_car_position()
588.         else:
589.             #Print message to console saying program is finished executing
590.             print_to_console("Done running program")
591.             console.tag_add("End", "end -2 1", END)
592.             console.tag_config("End", foreground="Green")
593.             print "OUTPUT: " + console.index("end -1 1")
594.         else:
595.             #Print message to console saying program has errors
596.             print_to_console(
597.                 "You have " +
598.                 str(len(errors)) +
599.                 " error(s) in your program")
600.             console.tag_add("Error", "1.0", "1.end")
601.             console.tag_config("Error", foreground="Red")
602.
603.             for error in errors:
604.                 print_to_console(error)
605.         else:
606.             print "Blank"
607.
608.
609. #Checks if program is a valid Racecar program and returns corresponding python
610. #code if necessary
611. def verify_program(code):
612.     clear_console()
613.     if len(code) < 2:
614.         return ("BLANK", False)
615.     code, errors = Racecar.Compiler.getPythonCode(code)
616.     if errors:
617.         return (code, errors, False)
618.     else:
619.         return (code, errors, True)
620.
621.
622. #Called when verify program is called
623. def verify_program_callback(code):
624.     verification = verify_program(code)
625.     if verification[2]:
626.         print_to_console("Program syntax correct")
627.         console.tag_add("Correct", "1.0", "1.end")
628.         console.tag_config("Correct", foreground="Green")
629.     else:
630.         errors = verification[1]
631.         print_to_console(
632.             "You have " +
633.             str(len(errors)) +
634.             " error(s) in your program")
635.         console.tag_add("Error", "1.0", "1.end")
636.         console.tag_config("Error", foreground="Red")
637.
638.         for error in errors:
639.             print_to_console(error)
640.
641.
642. #Resets car's position and orientation to original
643. def reset_car_position():
644.     global car
645.     canvas.delete(car.car_object)
646.     car.image_tk = ImageTk.PhotoImage(car.image)
647.     car_height = int(canvas.winfo_reqheight())/2
648.     car.car_object = canvas.create_image(
649.         23,

```

```
650.         car_height,
651.         image=car.image_tk)
652.     car.position_x = 23
653.     car.position_y = car_height
654.     car.car_direction = CarDirection()
655.
656. #car object
657. car = Car()
658.
659.
660. #User interface
661. #Toggle enabled and disabled buttons when program is run and stopped
662. def toggle_buttons(stop_button_should_be_enabled):
663.     if stop_button_should_be_enabled:
664.         run_button.config(state=DISABLED)
665.         stop_button.config(state=NORMAL)
666.         reset_car_position_button.config(state=DISABLED)
667.         clear_button.config(state=DISABLED)
668.     else:
669.         run_button.config(state=NORMAL)
670.         stop_button.config(state=DISABLED)
671.         reset_car_position_button.config(state=NORMAL)
672.         clear_button.config(state=NORMAL)
673.
674. root = Tk()
675. root.title('Racecar')
676. #Height is always three fourths the width of the window
677. window_width = root.winfo_screenwidth() - 100
678. window_height = 9*window_width/16
679. root.geometry("%dx%d" % (window_width, window_height))
680. root.resizable(width=FALSE, height=FALSE)
681.
682. menu_bar = Menu(root)
683.
684. menu = Menu(menu_bar, tearoff=0)
685. menu.add_command(label="Open", command=open_file)
686. menu.add_command(label="Save", command=save)
687. menu.add_command(label="Save As", command=save_file_as)
688. menu.add_separator()
689. menu.add_command(label="Quit", command=exit)
690. menu_bar.add_cascade(label="File", menu=menu)
691.
692. menu = Menu(menu_bar, tearoff=0)
693.
694. command = lambda: verify_program_callback(code.get(1.0, END))
695. menu.add_command(label="Verify Code", command=command)
696.
697. command = lambda: generate_program(code.get(1.0, END))
698. menu.add_command(label="Run Code", command=command)
699.
700. menu.add_command(label="Clear Code", command=clear)
701. menu.add_command(label="Clear Console", command=clear_console)
702. menu_bar.add_cascade(label="Code", menu=menu)
703.
704. menu = Menu(menu_bar, tearoff=0)
705. menu.add_command(label="Course 1", command=course_one)
706. menu.add_command(label="Course 2", command=course_two)
707. menu.add_command(label="Course 3", command=course_three)
708. menu.add_command(label="Course 4", command=course_four)
709. menu.add_command(label="Course 5", command=course_five)
710. menu.add_separator()
711. menu.add_command(label="Clear course", command=clear_course)
712. menu_bar.add_cascade(label="Courses", menu=menu)
713.
714. root.config(menu=menu_bar)
```

```
715.
716. #frame for left side of window
717. left_frame = Frame(root)
718.
719. #label for code window
720. code_label = Label(left_frame, text="Enter code here", anchor=W, pady=5)
721.
722. #frame for code window to hold textbox and scrollbar
723. code_frame = Frame(
724.     left_frame,
725.     width=int(0.3*window_width),
726.     height=9*window_height/10)
727. code_frame.grid_propagate(False)
728.
729. #scrollbar for code window
730. code_scrollbar = Scrollbar(code_frame)
731. code_scrollbar.pack(side=RIGHT, fill=Y)
732.
733. #code is the window in which the code is written
734. code = Text(
735.     code_frame,
736.     width=50,
737.     #height=window_height/16-8,
738.     wrap=WORD,
739.     yscrollcommand=code_scrollbar.set)
740.
741. #Frame for buttons
742. button_frame = Frame(left_frame)
743.
744. #run_button passes code into a run program method
745. command = lambda: generate_program(code.get(1.0, END))
746. run_button = Button(
747.     button_frame,
748.     text="Run Code",
749.     pady=5,
750.     padx=5,
751.     command=command)
752.
753. #Stop execution of running program
754. stop_button = Button(
755.     button_frame,
756.     text="Stop Program",
757.     padx=5,
758.     pady=5,
759.     command=stop_program)
760. stop_button.config(state=DISABLED)
761.
762. #reset car position button puts the car back in its original position and
763. #orientation
764. reset_car_position_button = Button(
765.     button_frame,
766.     text="Reset Car Position",
767.     pady=5,
768.     padx=5,
769.     command=reset_car_position)
770.
771. #clear_button clears the code in the text box
772. clear_button = Button(
773.     button_frame,
774.     text="Clear Code",
775.     command=clear)
776.
777. #canvas is where the car will go
778. canvas_frame = Frame(
779.     root,
```

```
780.     width=window_width/1.5,
781.     height=window_height/1.5,
782.     padx=2,
783.     pady=2)
784.
785. canvas_frame.configure(borderwidth=1.5, background='black')
786. canvas = Canvas(
787.     canvas_frame,
788.     width=window_width/1.5,
789.     height=window_height/1.5)
790.
791. car.image = Image.open('Racecar/RacecarGUI/images/racecar.png')
792. car.image_tk = ImageTk.PhotoImage(car.image)
793.
794. car.car_object = canvas.create_image(
795.     23,
796.     int(canvas.winfo_reqheight())/2,
797.     image=car.image_tk)
798.
799. car.position_x = 23
800. car.position_y = int(canvas.winfo_reqheight())/2
801.
802. #label above the console
803. console_label = Label(root, text="Console", anchor=W, pady=5)
804.
805. #frame for the console to hold the textbox and the scrollbar
806. console_frame = Frame(root)
807.
808. #scrollbar for the console
809. console_scrollbar = Scrollbar(console_frame)
810. console_scrollbar.pack(side=RIGHT, fill=Y)
811.
812. #console to print to
813. console = Text(
814.     console_frame,
815.     width=int(window_width/1.5),
816.     height=8,
817.     padx=2,
818.     pady=2,
819.     wrap=WORD,
820.     yscrollcommand=console_scrollbar.set)
821.
822. console.config(state=DISABLED)
823.
824. #add them to GUI Window
825. #These are grouped logically in order to better see what's going on
826. left_frame.pack(side=LEFT, fill=BOTH)
827.
828. code_label.pack()
829.
830. code_frame.pack(expand=1, fill=BOTH)
831. code.pack(expand=1, fill=BOTH)
832.
833. button_frame.pack(fill=BOTH)
834. run_button.grid(row=1, column=1)
835. stop_button.grid(row=1, column=2)
836. reset_car_position_button.grid(row=1, column=3)
837. clear_button.grid(row=1, column=4)
838.
839. canvas_frame.pack(expand=1, fill=BOTH)
840. canvas.pack(expand=1, fill=BOTH)
841.
842. console_label.pack()
843.
844. console_frame.pack(expand=1, fill=BOTH, pady=(0, 10))
```

```
845. console.pack(expand=1, fill=BOTH)
846.
847. code_scrollbar.config(command=code.yview)
848. console_scrollbar.config(command=console.yview)
849.
850. root.update_idletasks()
851.
852. #Origin and antiorigin are limits on the canvas where the car moves
853. origin = (23, 26)
854. anti_origin = (
855.     23+106*canvas_frame.winfo_width()/110,
856.     26+56*canvas_frame.winfo_width()/110)
857.
858. #horizontal grid lines
859. position = 0
860. while position < anti_origin[0]:
861.     tick = canvas.create_line(
862.         position,
863.         anti_origin[1]-5+35,
864.         position,
865.         anti_origin[1]+35,
866.         fill="#000",
867.         width=2)
868.     grid_ticks.append(tick)
869.     position += steps_to_pixels(5)
870.
871. #vertical grid lines
872. position = 0
873. while position < anti_origin[1]:
874.     tick = canvas.create_line(
875.         0,
876.         position,
877.         5,
878.         position,
879.         fill="#000",
880.         width=2)
881.     grid_ticks.append(tick)
882.     position += steps_to_pixels(5)
883.
884. print anti_origin
885. #Run the GUI
886. root.mainloop()
```