

Programming Assignment 5 - Gotta Catch 'Em All

Due Saturday by 11pm **Points** 100 **Submitting** an external tool
Available after Nov 22 at 11:50am

Update 11/29/23: Clarified HW4 solution (this is code you can reuse from your solution)

Update 11/27/23: ~~Lab 5 Overview and Demo Video~~; new video will be uploaded soon.

In this assignment which is built upon assignment 4, you will utilize new data structures (classes and dictionaries) to come up with a program that: generates children objects of a class (a Car class) with random colors and sizes, moves these objects, records the interaction with these objects (upon clicking on the Car objects), and calculates some score. Please attend your discussion and lab sessions, where necessary hints and help will be provided to aid in following the thorough content outlined in this document. Below is an overview of the graphical user interface:

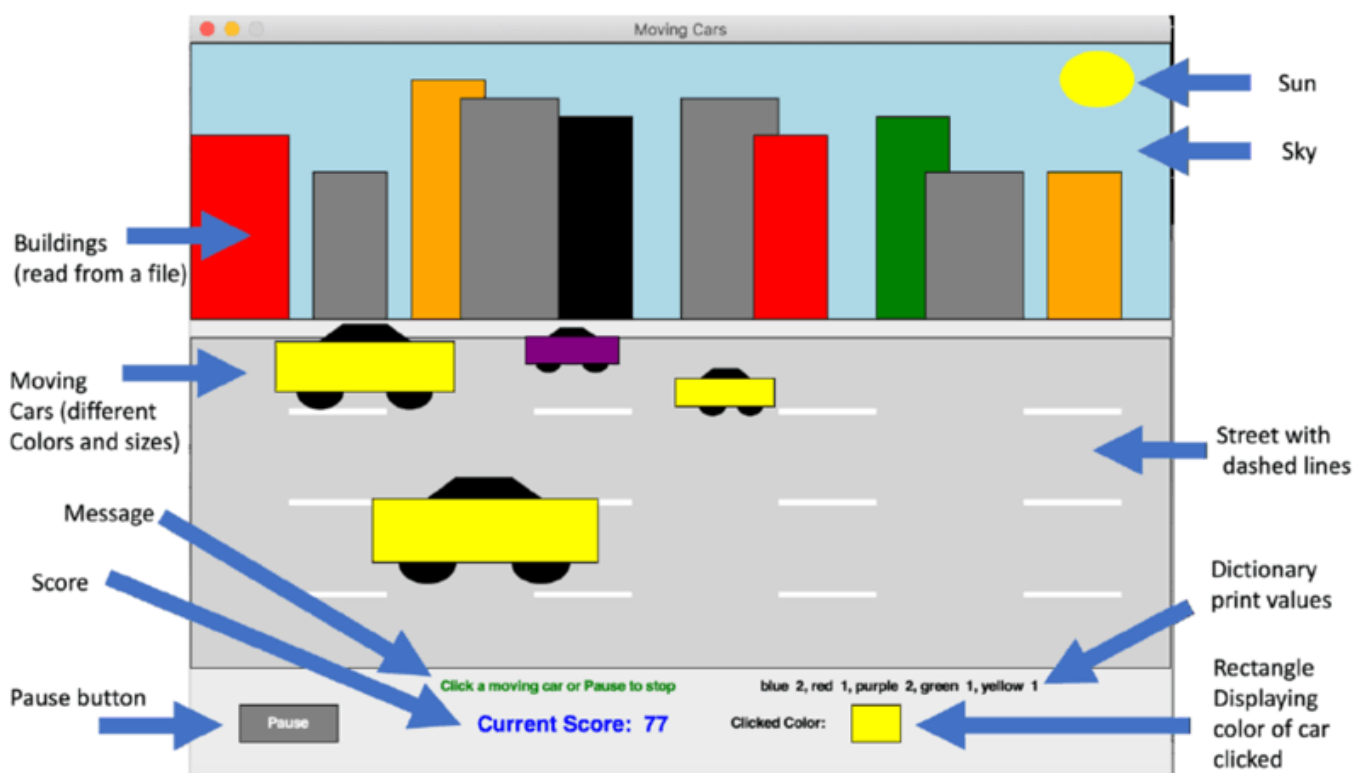


Figure 1: Example of Assignment 5 GUI in action

In particular, this is a car clicking game, in the sense that once the start button is clicked, the program starts to randomly generate cars (with random colors and sizes) that move from left to the right, and if any car is clicked, the score changes according to some defined formula and the car color counter in the dictionary is accumulated with the clicked car's color. The smaller the car, the higher the bonus added to the score. The two main data structures (new) used in

this assignment are “Class” and “Dictionary”. The Class is used to define a class of car objects with certain properties (shape of car, score that depends on the size of the car, the color of the car). This class also has some methods (functions) that facilitate access to object attributes in the class. The dictionary is used to keep track of the colors of the cars that were clicked, and the number of times each color was clicked.

Initial Program Sequence

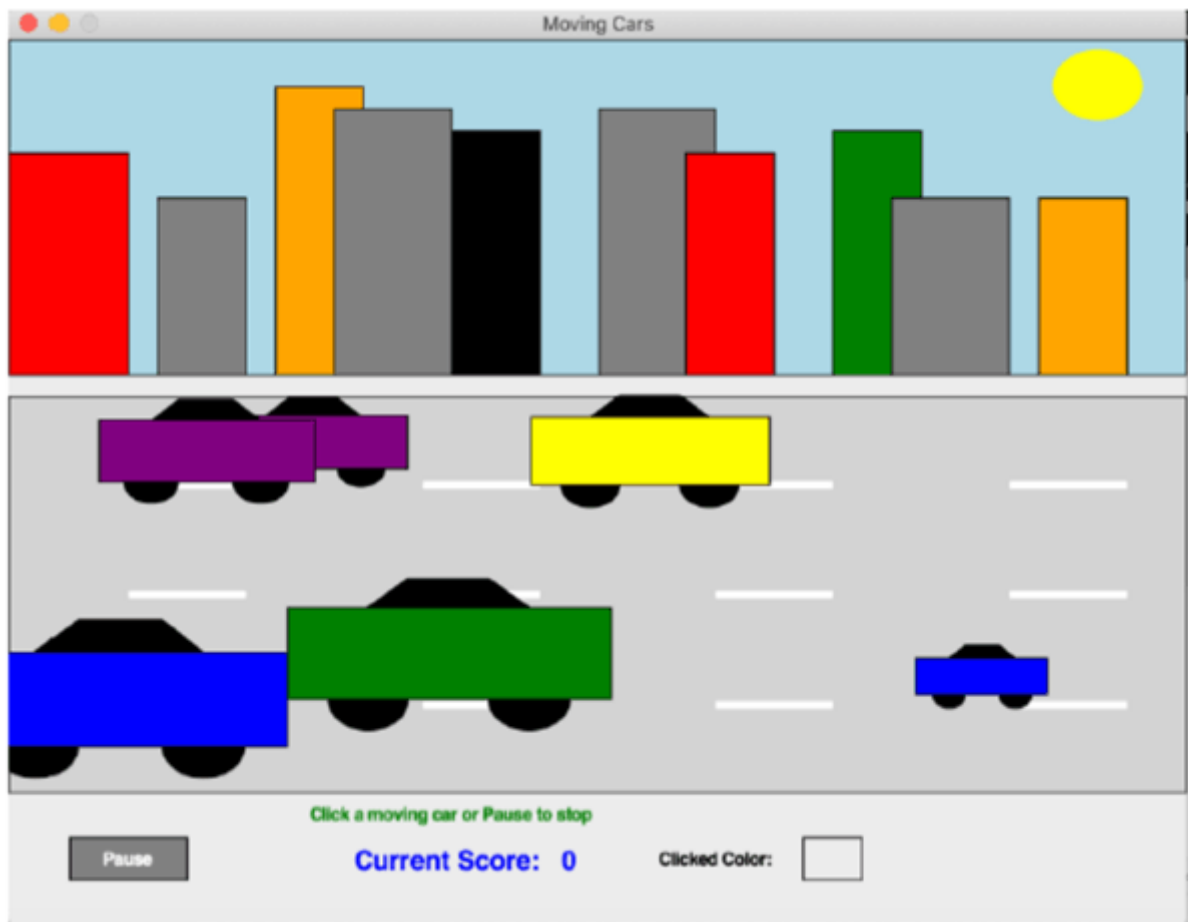
1. A window pops up with the street, sky, sun, buildings, a button, an empty rectangle and three message:



Initial Game Screen. Initial green text: "Please click the Start button to begin"

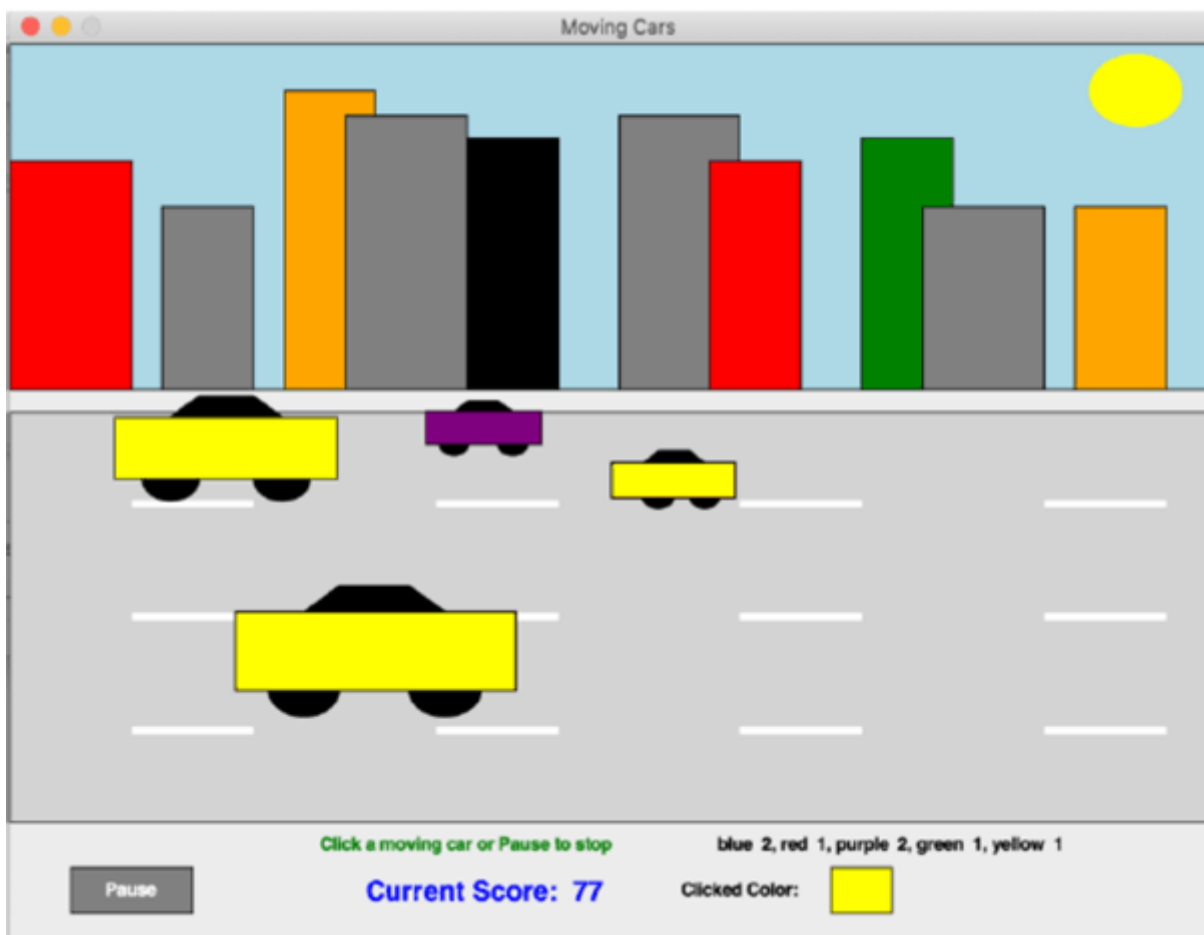
2. When the “Start” button is clicked, car objects should start to appear randomly on the street, with different sizes and colors. The message in green changes as well:





Starting Game Screen. Green text: "Click a moving car or Pause to stop"

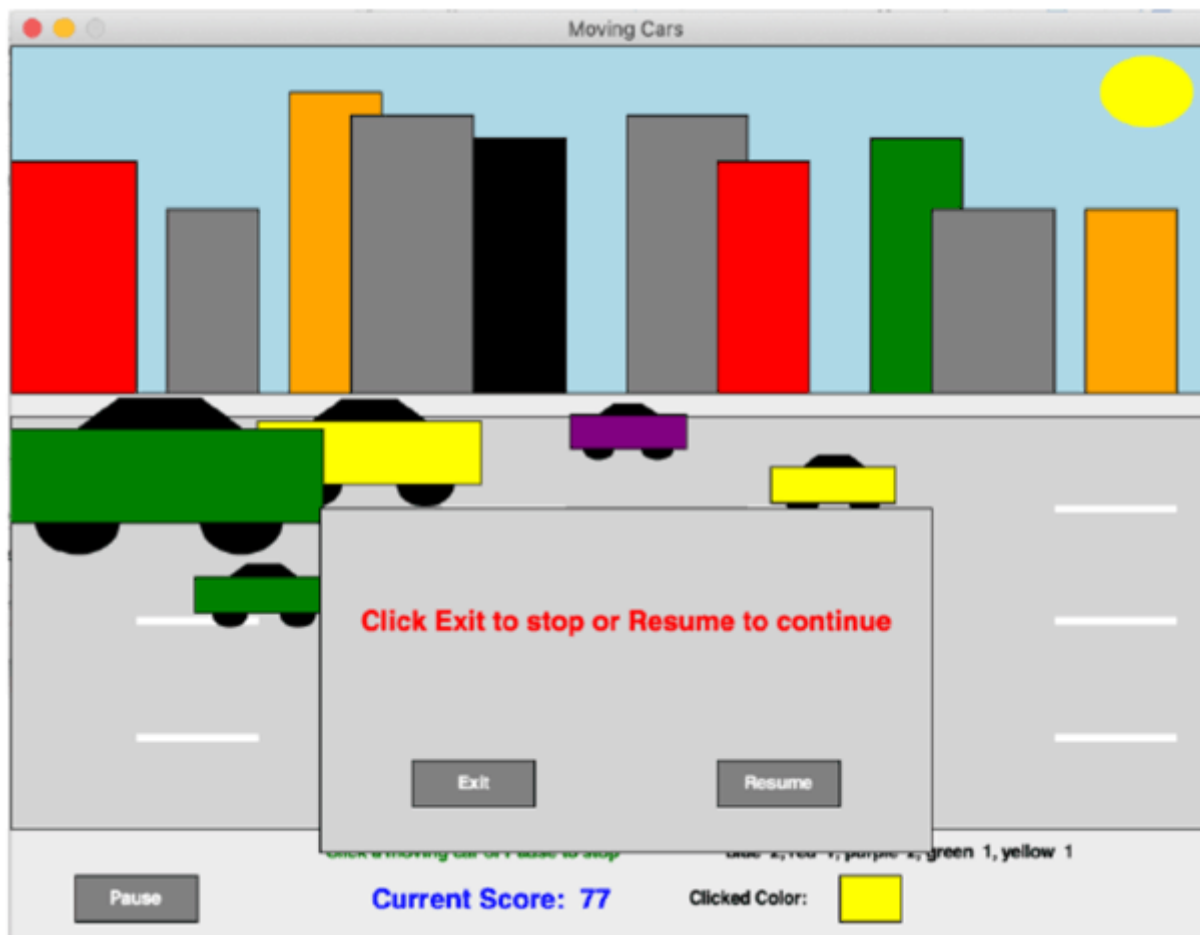
3. As the user starts to click the "body" (not the roof, not the tires) of certain cars, the score starts to change, the empty rectangle should be displaying the color of the car that has just been clicked, and a dictionary appears showing the colors that were clicked and the number of times each color was clicked:



Game Screen as user clicks car bodies.

4. When the user presses “Pause” the cars should stop moving in their places (they should pause), and a new window pops up that shows a message and two buttons “Exit” and “Resume”:



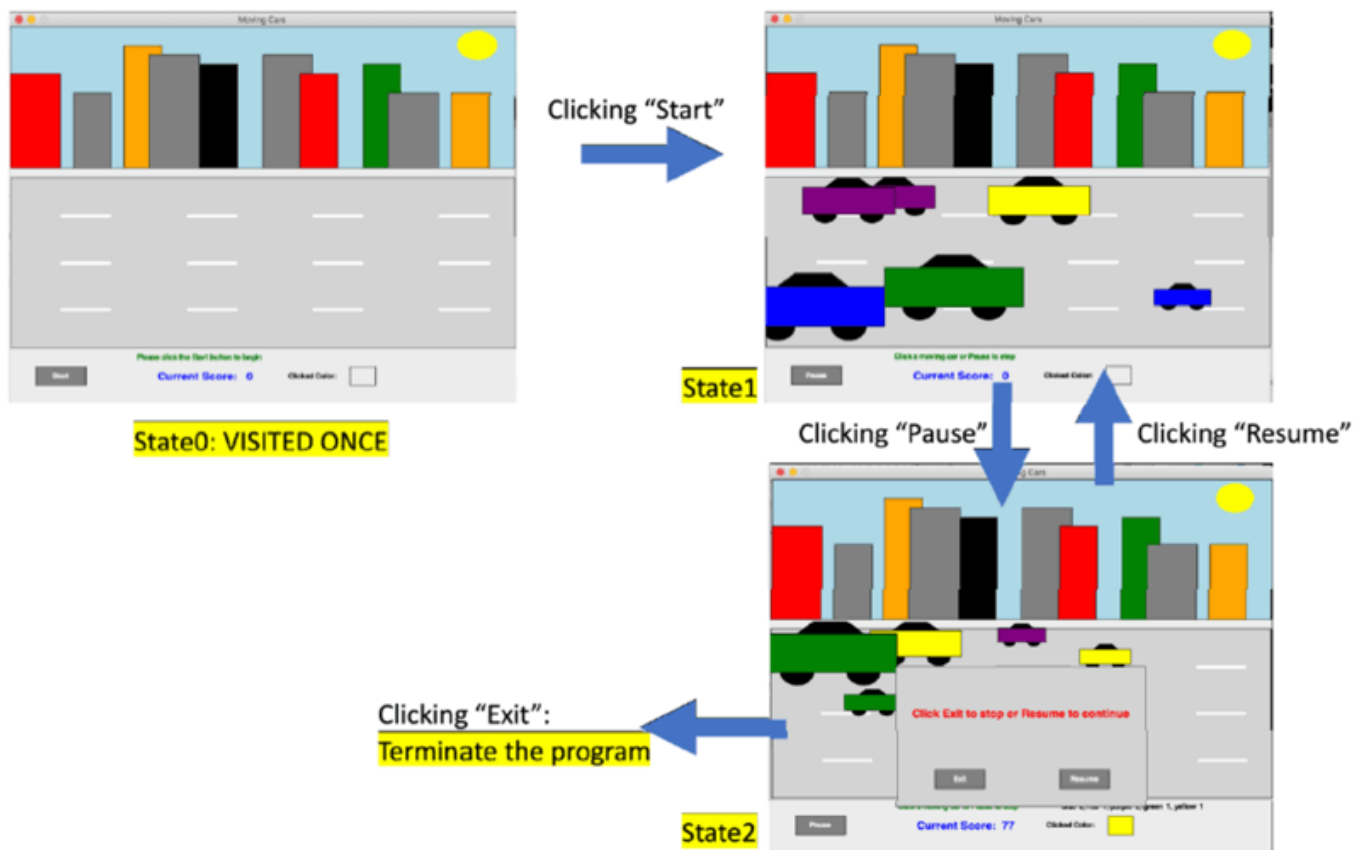


Paused Game Screen. Pop-up window text: "Click Exit to stop or Resume to continue"

5. If the user presses "Exit", the window should close and the program terminates. Otherwise, if the user presses "Resume", the cars should proceed their motion from where they have stopped before, and the game should proceed regularly by generating new cars that move and that the user can click on to allow changes in the score and dictionary.

GUI Control Sequence

The scene is changed upon the user clicking on the buttons as follows:



State Transitions for Game

There are 3 states that can occur when the user interacts with the buttons:

- **State0**: This is the initial state of the program. It is only visited once (upon running the program), and with the later interaction with the buttons, we do not revisit this state. In this state, the stationary scene (sky, street, buildings, sun, and messages) can be seen along with the "Start" button.
- **State1**: In this state, cars with different colors and sizes are randomly generated and rendered in motion (the cars are moving). The button "Pause" is there for the user to interact with. Also, when the user clicks on cars, the score changes, the dictionary showing the clicked colors gets updated, and this state persists. The only way to change this state is clicking on the "Pause" button.
- **State2**: In this state, the cars in the background pause their motion, a new window pops up and waits for a click. If the user clicks on "Resume", the state changes into State1, and the cars continue their motion. Otherwise if the user clicks on "Exit", the program terminates and the window closes.

Coding Your Program

We have posted on canvas page a template that you can use as your starting `hw5.py` available here: [hw5_template.py \(https://canvas.eee.uci.edu/courses/55954/files/23815513?wrap=1\)](https://canvas.eee.uci.edu/courses/55954/files/23815513?wrap=1). This template has some code that you need to complete by adding your own code. To complete this HW, divide and conquer the tasks, and follow the steps outlined below in SEQUENCE. This will

help you debug faster and achieve incremental results while you code towards completing the whole assignment.

Step 1: Creating the stationary scene of state0

In the template, the whole background scene including the sky, buildings, street, dashes on the street, sun, "Start" button, the initial text message "Please click the Start button to begin", and the empty rectangle (along with its text "Clicked color:") are given to you. You will load the [hw5_input.txt \(https://canvas.eee.uci.edu/courses/55954/files/23815506?wrap=1\)](https://canvas.eee.uci.edu/courses/55954/files/23815506?wrap=1). [↓ \(https://canvas.eee.uci.edu/courses/55954/files/23815506/download?download_frd=1\)](https://canvas.eee.uci.edu/courses/55954/files/23815506/download?download_frd=1) file to generate the scene.

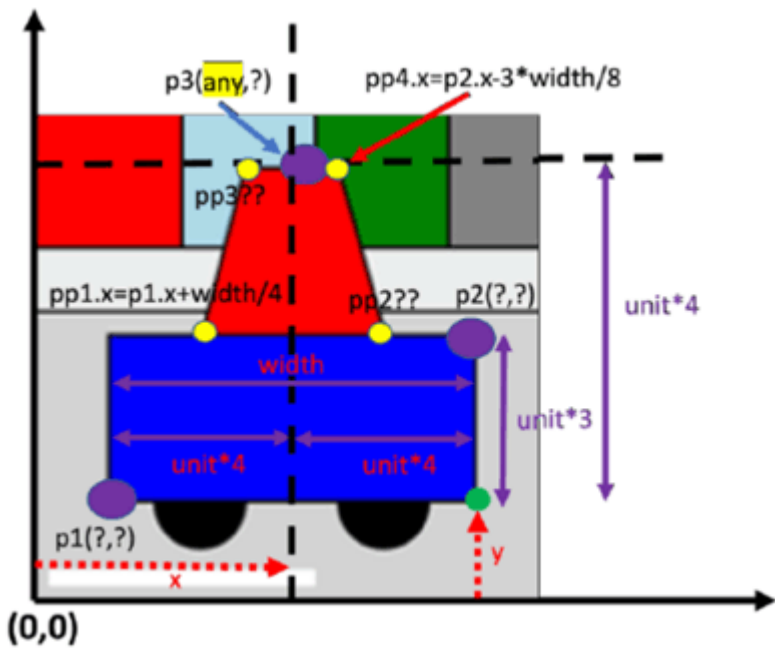
Step 2: Creating a Car Class

In the template file, the **Car** class is composed of **__init__()**, **getColor()**, **getScore()**, **undraw()**, **drawCar()**, and **move()** methods. Only the **getColor()** and **getScore()** are given to you as complete methods. The class inherits the **Rectangle** class for the ease of moving and click checking. You need to add code in order to complete this class. First you need to complete the **__init__()** method. Note that this method calls another method **drawCar()**.

__init__(self, win, x, y, unit):

- This method takes self (an instance of the class), win (the window), x (a point abscissa created randomly (see below)), y (a point ordinate created randomly (see below)), and unit(a random value that would help draw the car (see below)).
- x, y, and unit are defined in main(), and in particular in the infinite while loop inside the condition of being in state 1. But these are passed from main() to the Class to generate the car (see later)
- In the **__init__()** method, first, a colorlist is given to you containing 6 colors. By using **self**, which represents the instance of the class, you need to define a "**color**" attribute variable of the class. This attribute should randomly choose one of the 6 colors in the colorlist. Doing this, when a new Car object of the class Car is created, a random color is assigned to the car. The **color** attribute defined by **self** is then used the method **getColor(self)** already defined for you in the template. The purpose of **getColor(self)** is to return that attribute when it is called. The name of the instance variable "color" should be consistent with the one used in **getColor(self)** as well.
- Next, you need to define three points p1, p2, and p3 that depend on x, y and unit (for now ASSUME you know x,y and unit. We will define them in the while loop and pass them as arguments to the class when we generate cars later on). After finding the coordinates of p1, p2 and p3, you need to call the function "**drawCar()**" which draws the whole car (very similar to the 2 functions you defined in hw4 for drawing body, tires and roof but this time combine these 2 functions as one function here => START FROM the two function given in your HW4 solution, because the same dimensions for tires, roof, etc are used given p1,p2 and p3). In particular, see the below figure for help in calculating the coordinates of p1, p2, p3, pp1,

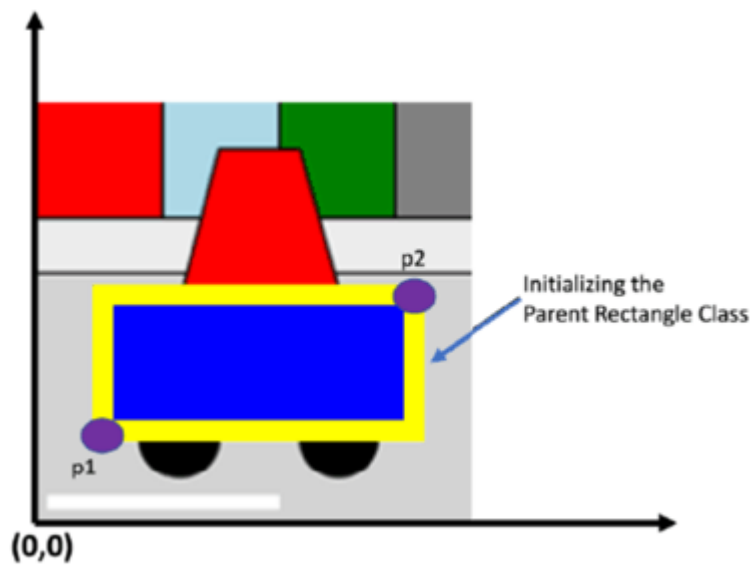
pp2, pp3, and pp4. In drawCar() method, and different from HW4 functions make sure you define the tires, body and roof as attributes using the self keyword.



Car Pixel Info

Next, use self to define the attribute score as the integer (use int()) of “unit*10”. This attribute is used by the method getScore(self) already defined for you in the template. The purpose of getScore(self) is to return that attribute when it is called. The name of the instance variable “score” should be consistent with the one used in getScore(self) as well.

- Finally, you can use p1 and p2 to initialize the parent class (the Rectangle class) that the Car class inherits from using super() keyword. You will understand the purpose of this initialization later in this documentation.
- There is no return value for this method



Initializing parent rectangle

After you work on the `__init__()` method (and `drawCar()` method), you need to complete the `move()` method:

move(self, Xdistance):

- Note that this method overrides its parent class (**Rectangle**)'s **move()** method. (you do not need to explicitly code anything that indicates that, that happens when you define this function)
- It uses `move(Xshift,Yshift)` of the parent rectangle class. You need to determine the **Xshift** and **Yshift**. In particular, you need to move the **left tire**, **right tire**, **body**, **roof** and **super()** (note: use `self` to access tires, body and rood) for a relative distance **Xdistance** and vertical relative distance always set to 0 in our case since we do not move the cars vertically. For now, **Xdistance** is just a variable, since we will use this method of the class outside the class when we call it from main to move the cars (see below)
- There is nothing to return for this method.

After you complete this method, you need to complete the `undraw()` method.

undraw(self):

- This method should undraw the left tire, right tire, body, and roof of the car using the keyword **self**. You need to call **undraw()** for each of them. Again, this method overrides the undraw method in the parent class.
- It has no return value

As mentioned above, you need to use `p1` and `p2` to initialize the parent class it inherits using `super` keyword so that it is much easier to check if a Car is clicked by the mouse or not by using the `P1` and `P2` points.

Step 3: Infinite loop, state transitioning

Outside the infinite while loop, initialize your game state to 0. Define the function **isClicked()** which takes a `pcklick` and a `button` as arguments. This function is identical to the **isHitBtn()** that you have used in assignment 4. The first thing you need to do in the infinite loop is checking whether there is any mouse clicking by using **checkMouse()** (Since the **sleep()** function from the `time` module will block your program, you will need a non-blocking way like **checkMouse()** as opposed to **getMouse()**) function which is built in the `graphics` library. Then, you need to utilize the function **isClicked()** to check whether the button was clicked and accordingly update the state. There are three states in total, for details about these states please refer to a section above describing these states. The conditional statements leading to the state transitioning should be defined according to the following rules.

- If the game is in state 0, and the “Start” button is clicked, then it changes the message to be “Click a moving car or Pause to stop” and switches to state 1 where the cars start being generated and moving

- If the game is in the state 1 and the “Pause” button is clicked, then the cars should pause their motion and a pop-up menu is displayed (i.e. you need to draw **exit_bg**, **exit_message**, **confirm_button**, **confirm_label**, **go_back_button**, **go_back_label**) and the state switches to state 2
- If the game is in state 2 and the mouse clicks on the “Exit” button, then you should break out of the infinite loop, close the window, and terminate the program. Otherwise, if the “Resume” button is clicked, the pop-up window should disappear (you should undraw **exit_bg**, **exit_message**, **confirm_button**, **confirm_label**, **go_back_button**, **go_back_label**), and the state should change back to state 1 where the cars should continue their paused motion

Step 4: Creating the cars

In state 1, you need to generate a car object from the Car class every 0.2 second. For that,

1. Measure the current time by calling **time.time()** function which returns as the current time the number of seconds (a floating number) elapsed since January 1, 1970, 00:00:00 at UTC. Assign this time to a variable **current_time**. Also, you need a variable **start_time1** initialized to 0 outside of the infinite while loop.
2. Now if **(current_time - start_time1) >= 0.2**, you need to define x, y and unit.
3. Recall that x, y, and unit are defined here and then used by the Class Car to define p1,p2, p3 and the score. We define x, y and unit by using **random.random()** and **random.uniform()**. **random.random()** gives you a random floating point number in the range [0.0, 1.0) (so including 0.0, but not including 1.0 which is also known as a semi-open range). **random.uniform(a, b)** gives you a random floating point number in the range [a, b]. Define x by using **random.random()** where you subtract “0.5” from the **random.random()** value then add “4” to that (i.e. randomly sampled between 3.5 and 4.5). Define y by using **random.uniform(a,b)** with a range of [7, 23]. Define unit by using **random.random()** multiplied with “0.6” then added to “0.4” (i.e. sampled between 0.4 and 1). As such unit belongs to [0.4, 1).
4. Now that you have x, y and unit, you need to generate an object car from the Car class then append this car into a list of cars “car_list” (initialized outside the loop) (This list will eventually have all the car objects generated and it is used for car movement, click checking, and deletion.).
5. Set **start_time1** to be equal to **current_time+1** (the variable **start_time1** is updated with the **current_time+1** whenever the program creates a Car so that in the next iteration it will be the latest previous time when the Car was created.)

Step 5: Moving the cars

In state 1 as well, you need to add lines to move the cars in the list. First you need to check whether **current_time - start_time2 >= refresh_sec**. Note that start_time2 is another variable initialized to 0 outside of the while loop, and you DO NOT need to set

current_time=time.time() again. Refresh_sec is another variable initialized to 0.05 outside of the while loop. Now if this condition is met, and for each while iteration (in the infinite loop), you have to move each of the cars in the car_list by calling their **move()** method (Similar to the loop



you used for moving car parts in HW4, but here you move the whole car in the list in one shot, or in other words iterate over each car in the list rather than over each part of each car in the list!! because we use the **move()** function of class Car that overrides that of the parent class). The horizontal distance that you need to move these cars at, is `pixel_per_second * refresh_sec` where `pixel_per_second` is a variable initialized as 10 outside of the while loop. If any car reaches the right end of the window, you can FIRST delete it by calling its **undraw()** method and THEN remove it from the list. You need to test if the car's P2's X abscissa (Note that by iterating over the cars in the list and checking **getP2().getX()** we would automatically be checking the rectangle's `p2.x` that we have defined via **super()** to be the body of our car) is out of the boundary of the window to decide if it should be deleted and taken out of the list. Still satisfying the if condition (in the body of that if statement) that **current_time - start_time2 >= refresh_sec**, but after the for loop that moves the cars and checked whether they have exceeded the boundary, you need to update `start_time2` with current time. This simply means that we are checking whether the time lag (or elapsed since current time) exceeds 0.05 seconds, and it so we move the cars by a distance of 10 pixels. Note that according to your computer's delay, the distance moved by the cars may be large or small (fast or slow), do not worry about it.

After you complete this step, you should see cars being generated on the left side of the window with random colors and sizes and moving towards the right side of the window. When they exceed the boundary, these cars disappear.

Step 6: Calculation of score and color counter (using dictionary) when cars are clicked

You have already added an instance variable (or attribute) to the class in the **__init__()** method for the score in step 2 (see above). Recall that the score is defined in the class as `unit*10`. In state 1, you need to check whether a car is clicked by iterating over the cars in `car_list` and using **isClicked()**. Here along with the mouse click, you need to pass the item in the `car_list` to the **isClicked()** method. Remember that the car class inherits Rectangle class and you can take advantage of the **isClicked()** function which checks the P1 and P2 of the rectangle drawn on the car's body (that was determined by **super()** in class as initializing the parent rectangle class) without the need to check the points of the car body rectangle, or corner points of the polygon.

Now if a car is clicked, you need to update the score and define a dictionary. Before we proceed, let us review **getColor()**, and **getScore()**:

getColor(self):

- This given method returns the car color string instance variable color previously stored in the constructor **__init__()** by self keyword.

getScore(self):

- This given method returns the integer instance variable score previously stored in the constructor **__init__()** by self keyword.



If any car is being clicked, you need to retrieve the score by calling the car's method **getScore()** and the car's color string by calling the method **getColor()**. Afterwards, you call the car's method **undraw()** and remove it from the list. So if a car is clicked, we retrieve its color and score then undraw it and remove it from the list (in this order).

With the car color string from the **getColor()**, you show numbers of the cars with the same color have been clicked so far using the Text **clicked_colors_message**. To do so, you first introduce an empty dictionary (**clicked_colors**) in the **main()** function before the infinite loop and access it in the infinite loop (inside state 1 conditional statement) with the car color string as the key and integer counter as the value. This counter counts how many times each color was clicked. For correctly incrementing the counter (i.e. the value of the dictionary access), use **clicked_colors.get(..., 0)+1**. The **get()** method of the dictionary (it is built in) returns the value of the item with the specified key. Passing a value is usually optional to this method, but we pass a value of 0 so that if the color was never in the dictionary, we return the value of 0 then increment 1 to it hence we get a counter of 1 to this color. You also need to use the retrieved color of the clicked car to display that color inside the empty rectangle on the bottom right of the window (See demo images above).

Now that you have updated the dictionary with the color clicked and incremented its counter and updated the rectangle with the color, you need to update the score to print it after the message "Current Score: ". For that, define a variable to store the total score and change the score text message based on that variable. If the retrieved score from the clicked car is stored in variable **bonus**, then your total score is equal to the previous total score added to $\text{int}(100 * (1/\text{bonus}))$. The reason we use the reciprocal of the score is to give higher score to smaller cars, because recall that the score depends on unit. So, the smaller the unit (i.e. the smaller the car), the higher the bonus and hence the total score increases more with clicking on smaller cars.

After calculating the score and the counter, to display the score, you need to update the Text **score_message** with the current total score by its **setText()** method. The score message showing the total score should be blue. For the display of the dictionary, you should show the key (car color) and the value (counter) for each of the pairs in the dictionary like the following figure (string with a comma then space between the key and the value) using the Text **clicked_colors_message** by the **setText()** method.



blue 2, red 1, purple 2, green 1, yellow 1

Score section of game

Now that all of these steps are complete, you should have a program that runs just like the demo.

Grading Criteria

- Correct usage of dictionary (step 6) - 10 pts
- Correct class definition and methods (step 2) - 20 pts




- Correct game logic, i.e. calculating and displaying score (step 6) - 20 pts
- Correct car generation (step 4) - 20 pts
- Correct movement adjustment (step 5) - 20 pts
- Correct state transitions (step 4) - 10 pts

Submission Information

Submit your homework using GradeScope. Submit a Python file with the name "hw5.py". Your program only needs to work with the template hardcoded hw5_input.txt file.

Submit Programming Assignment

 Upload all files for your submission

Submission Method

 Upload

 GitHub

 Bitbucket

Drag & Drop

Any file(s) including .zip. Click to browse.

