Problem statement:

Parking lot system: be able to simulate it using:

- Stacks for tracking parked cars
- Queues for waiting cars

Objective:

- Handle car entry, exit (they exit automatically), and display parking status

Algorithm design and choice of data structures.

- All searching algorithms are implemented with bruteforce implementation (everything gets updated every)
- The data structures used were:
    - Stacks (maximum of 1 depth)
    - Queues (indefinite)
    - List (for handling multiple stacks)
    - 2D list (for handling the list that contains stacks)

Challenges faced and solutions.

- Wrestling with python's special methods such as __str__ and __repr__ to name a few. The solution is that I searched online and that includes chatgpt.
- I'm a bit overwhelmed with how much python is so advanced. From how python code is structured to features like decorators and handling dynamic dispatch is foreign to me yet. Though I didn't use such in this project. The solution is looking into other people's code in github hehe.
- I wanted to automatically remove the parked vehicles. That one took me a bit of time to figure out whether I handle it globally (i.e. update all according to a global time) or have each slots handle when they will get out on their own. The solution is the latter.
- There are tons of things that I haven't tried in raylib yet. The add vehicle button has too small of a text. With how ahmed's [8] did it using a separate rectangle, text, and checking whether it collides with the mouse after being pressed – I didn't want to refactor now that I am 1 and a half hours away from the deadline.
- The most problematic one is the calculation of where the graphical element would be. I used krita first to calculate. However, I didn't like the workflow. And the solution was I told myself that I can just progressively iterate the position in the next few draw calls. And surprise-surprise, it wasn't the solution at all.
- Bugs like passing wrong arguments, wrong conditions, and incorrect updates in a loop weren't a bit time consuming to know where my logic went wrong.

- I approached this way too naively. I thought I need flowchart first, but dismissed it. I instead focused in the data structure aspect of the project. Though this still became helpful, there were still many iterations with the requirement (the one I want including the things I need to show). This includes:
    - Code structure
    - Main loop mess (is it normal for the update and draw functions having too many parameters?)
    -

Sample input/output:

With the program running:

1. Add a vehicle how many times you want (click the add button; **try 16 times**)
2. Wait and see as the queue grows and the parking spaces becomes red
3. Adjust the multiplier however you want (**try 10x**)
4. Go back to 1st step until u r satisfied
5. Press esc to exit

Testing methodology and results:

I provided a unitTests folder inside the src/. This is particularly used when doing unit tests (I didn't use the unittest module from python hehehehehehehehehehehe). The tests included testing whether it will crash the program or not, looking into the contents, updating the contents, and also removing the contents. When running the unitTest.py, it cannot be interpreted by default, *unless u comment the first line hehe.*

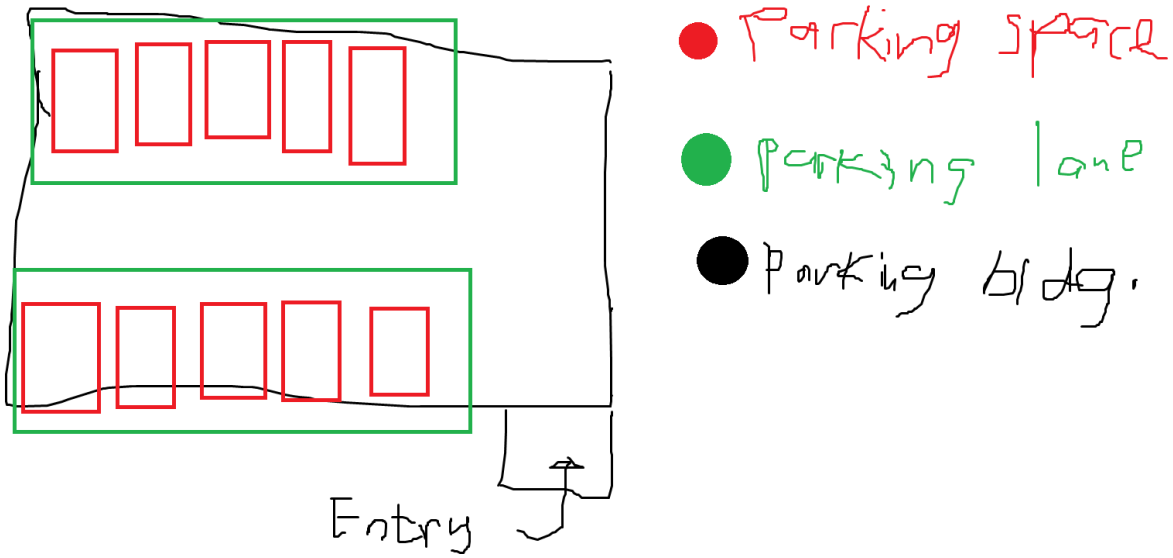
The following pages shows the development documentation (before I attempted to read the DSAL Final Project.docx) in each day. It is unstructured for the first few days, and it soon became commit by commit (git).

Day 1 (dec 20)

Rescoping project

As stated in the project proposal, there are some of the things I might need to further clarify and take out. Here are the features that the program will retain:

- A parking space (1 stack) will contain the vehicle (a random string for the plate number)
- A parking lane (1 list) will contain 5 parking spaces
- A parking building (one 2D list) will contain 2 parking lanes
  this illustration might help:



- The graphics is CLI based: standard output and text-based input.

I decided that it's better if the interaction between the program and the operator (user) models the real world. So, this goes like this:

- the queue of cars outside indefinitely grows with the click of a button.
- the list of stacks (array of parking lots) randomly shrinks;
- the time of the shrink (stack) is controlled by the user for the purposes of simulation.

After those are implemented and if time allows, the features I want to expand more are:

- ~~have the user be able to click and drag cars and collide with any objects present in the parking bldg.~~
- The graphics will resemble the above picture using raylib

Day 2 (dec 21):

Base project data structures

- Making a pseudocode
    - I found it hard to wrap how understand the requirement. The pseudocode below shows my initial attempt. However, I realized that in python, in order to enforce such a structure of data, we can use classes.

```
Que = []

// when queue insert
When lane a is not empty
     Insert from queue to whatever stacks that are empty
Or when lane b is not empty
    Insert from queue to whatever stacks that are empty

LaneA = [Stack(), stack(), stack(), stack()]
LaneB = [Stack(), stack(), stack(), stack()]

PrkingSpace = [LaneA, LaneB]
```

- I made a class for each said thing (parking space, lane, and bldg.) for a more coherent way to interact.
- I also made a few changes from the lab activities we did on stacks and queues. I also did some tests to check whether it was okay for this project. Although I am not sure how will I handle the errors. I asked chatgpt for some guidance, particularly for whatever this means: OverflowError, and IndexError. I might have to do a proper unit test, but im feeling not that too inquisitive this week.

Code structuring

- I was wondering how I could structure the codebase the same way we do in c/cpp. These references [1; 2] has a weird work around and I don't want to work with abstract base class (abc) package (*python is weird)*.
- I initially decided I will store all the required data structures in one file. However, after reading further on how python projects are structured [3; 4], there was a git repository that provides a sample python codebase [5]. I cant really accommodate this kind of structure fully due to the constraints of the project, and I understand little craps. Where dahek is the main function? ?? Why do I need to run a python program the same way I run scripts and why is the main functions only included in unit tests?? ??? ? ? I thought I understood python that much, but I realized I understood less than 10% of the full experience. Sorry for the rant. Looking a bit further, I found that I can use pyinstaller to build an executable [6]. And I seem to have a bit of trouble setting it up as well as running the test cases from the structure provided in samplemod [5].

Day 3 (dec 24)

Actually working with the base project

- I am feeling it that I'm stretching this project way too much, so I'll just work with the base cases and worry about the code structure later. I have made a github repository for tracking this (see appendix).
- From the three classes I have made (parking space, lane, and bldg), I have provided these methods:

Park space:

- Add vehicle
- Pop vehicle
- Get vehicle
- Is empty (private)
- Is full (private)

And starting with the parking lane, it is requiring quite a bit of brain power. I am thinking that I need to handle the randomized sa pagalis nung mga vehicle in this class. I would first need to randomize which in the list (parking space) will get out, then have a time multiplier that controls the whole thing. To get such effect, I need time generator that I got from Code Pope [7].

> *However, before going any further I tried sampling if we are to print the lane park without contents as I am using the __repr__ to print the first element of the stack. I wanted to see a list of None when printing the lane, however I only see a list of memory locations.* ***The second commit*** *shows that. I don't think stack is appropriate for this case.*
>
> *Well that was a joke. I just realized I don't need to access the first element of the stack. In the __repr__ I just need to check whether if the stack is empty or not as well as some other problems.* ***The third commit*** *shows the fix that I can still use the stack as dsa*

Going back to the parking lane implemention of parking lane, I cannot seem to iterate over the parking lane object, its length, get the content, and remove the vehicle. Omy snarky python. I thought this was automatic. I added the iter, len, and getitem special methods thanks to chatgpt hehe. **The fourth commit** addresses such. I also separated this test case to the unitTests.py inside the unit test folder (now named as unitTests) as I wanted a clean main function. I just realized that the context.py isn't that much needed when it's this small and I cant exactly resolve how to use this dependency. I just left two choices that you may need to comment whatever whether you want to *standalonely* compile the unit test, or just call the tests in the main function. But dang, I feel so dumb now with all these dependencies.

It's time to work with the parking building then, I didn't know what name should I give for the lanes. I also did the same test as I did in the previous lane_test. The **sixth commit** shows that.

Going back to the randomized popping, I thought that if I would generate random numbers in every frame, it would probably be very inefficient? Siguro mas okay ung merong sariling list ng countdown ang each parking spaces. Tas every frame nalang decremental magupdate according sa time. It will only generate another arbitrary number once it reaches 0 and the moment a vehicle is parked at that space. I also want to have the ability to multiply speed.

Day 4 (dec 25)

How do I do randomized popping

I thought that I will handle the randomized popping with a method in the parklane class. You can see the recent commit (6^{th}). The approach isn't appropriate when we are going with the plan stated at the paragraph above. I would probably need these:

- A global time variable
- The space park must have an **attribute** of **timeout**
- The space park must have a **method** that generates a random number from 5 to 30 (a floating point number, and seconds) that we invoke for **updating the timeout**.
- *The time multiplier? Im still not sure. I will get back to this later after building the above things.*

I tested the randomized popping as usual and it can be seen in the unitTests.py

Prototype development

It's probably time to work with raylib. A good example of time function that I referenced from is ahmeds work [8]. A problem also arises as I also need to calculate the height and width of the parking spaces for the screen**. The 7^{th} commit** shows an unsuccessful calculation of whatever the rectangles are. I was sure that it will resemble these 5 columns then 2 rows, but it was interchanged? The python raylib api reference is really helpful [9].

I was distracted a bit as I am testing a few raylib functions (gui sliders, gui buttons, and rectangles). To clearn my mind, I need to provide the interface first and the layout of the program.

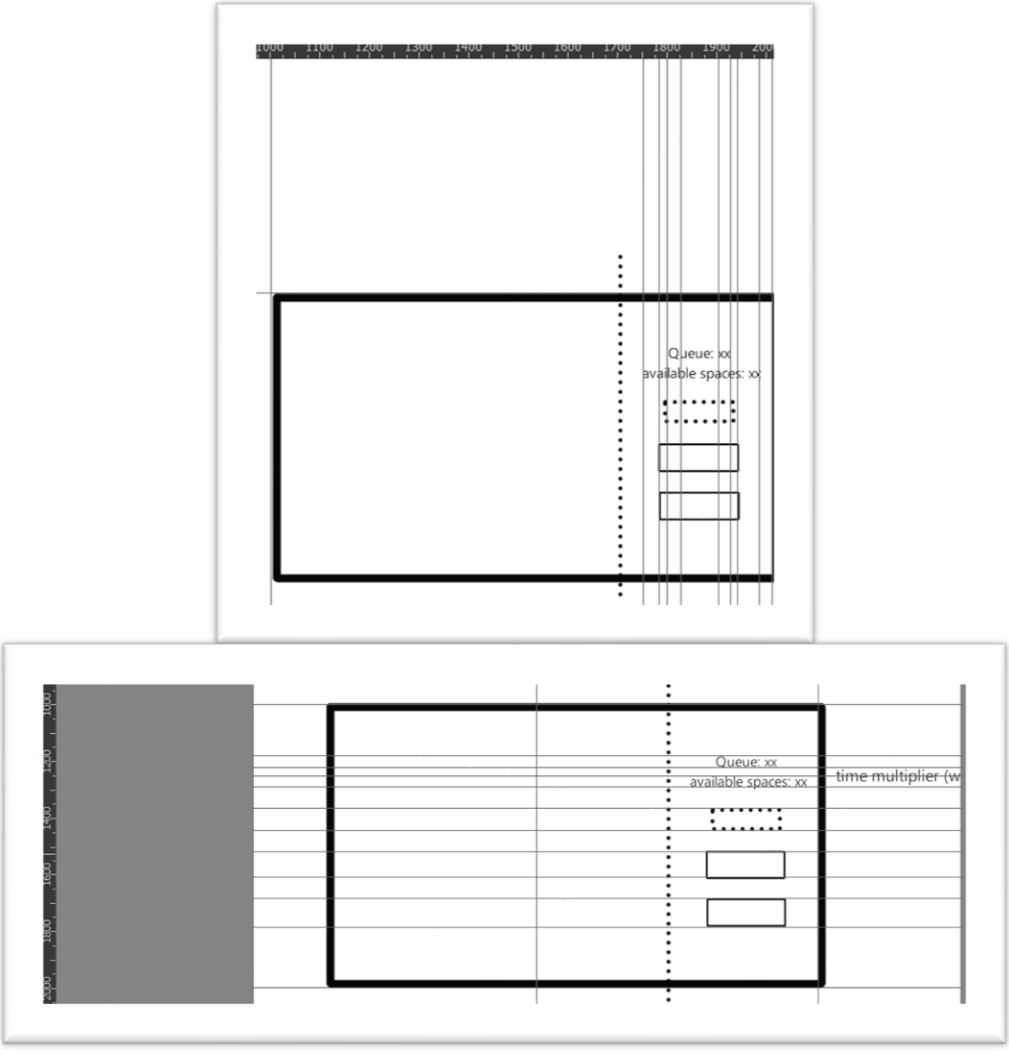
Interface requirements:

- Time multiplier slider
- Add vehicle to queue: Button to add something to the queue
- ~~(not sure) physics toggle: button to turn on physics~~

Layout requirements:

- Calculate this using percentages based off of the screenwidth and height (origin is top left)
- width:
    - interface rect: 70% to 100%
    - queue: 80.2 to 93
    - available space: 74.4 to 98.5
    - *time multiplier text*
    - time multiplier slider: 83.7% to 92%
    - add vehiclebutton : 78.3% to 94.2%
    - ~~*physics button: 78.3% to 94.2%*~~
- height:
    - interface rect: 0 to 100
    - queue: 18.6 to 22
    - available space: 25.7 to 29.7
    - *time multiplier text*
    - time multiplier slider: 35.3 to 43.7
    - add vehiclebutton : 51.8 to 61.5
    - ~~*physics button: 68.5 to 78.6*~~

this was done using this:



The hardcoded values wont be reflected on the final code as there are some micro-changes that I keep on iterating to make it presentable. The values instead would be used as a guideline as to where the general form is (I also separated the time multiplier text and the ~~physics~~ button)

I just realized that it would probably be better if I define a set of width and height for the buttons. But byebye. Lets go with wat I have.

For the purposes of making the main loop shorter, im thinking whether I would make the parking bldg. instead have the individual space width and height. Instead of putting it globally. The **8th commit** shows that, along with a working prototype for the interface. Additional major changes included in the commit:

- As well as the bug that shows 2 columns and 5 rows – I passed the wrong arguments to the parkbldg initialization.
- It also seems that when I iterate through the park bldg. object, we get a lane class, and iterate again, then get the spaces. If I get that specific space (we are returned a parkspace class) When we have an accumulator, we get 10, I don't know whether the issue was on the special method of __getitem__ or whatever. This resulted me in making the __is_empty a public method that was formerly private in the parkspace class.

The ninth commit:

Issues

- One major issue was the naming schemes. I wish I had been more structured before starting this. Particularly, some temp variables and flags. If it were temporary and just a flag, then there should not be underscore. Also, the object attribute followed with the context and underscore is what I made (eg. text_queue).
- I also have issues with sa paghiwalay nung update and draw routines sa main loop in regards when adding a vehicle. And I realized that there are many things that I should have put in a function to keep the code tidy. I don't know if I'm going to refactor it in time or not.
- This document is a bit hard to follow as it isn't structured that well. So for any future updates, I will be listing the features and issues with conceived solution. I forgot to read the bottom part of the provided DSAL final project docx. I didn't know there was a project report format was provided hehe, how stupid I am. Regardless, I will be making a summary in the first few pages of this document after I am done with the project.

Features

- Add vehicle to queue when the available space is full

Tenth commit:

Issues

- Bug with available space being 0 that was supposed to be initially ten. I just reverted the not condition
- Problem with queue only entering the parking bldg. after I press the button. I just made another for loop to handle.
- Big problem is the main loop. There were many for loops that I did, so some of it were merged. Also with the separation of the update and draw routines. The update only has 4 responsibilities, and the drawing function has 6 responsibilities. Though it has sosososososo many parameters. I didn't even have enough experience with with these many even in cpp. So crazy.

Features

- I referenced ahmed's work for [8] and oka's work [10] for working with time to remove the parked vehicles.

Eleventh commit:

Features:

- Update README

Tweleveth commit:

Issues

- None EZPZ

Features

- String generator for the vehicle name
- Also draw this vehicle (string name) in the parking space

thirteenth commit:

Issues

- Terrible workflow to do graphics

Features

- Better representation of the park bldg., lanes, and spaces
- Better text font

Fourteenth commit:

- Finishing up the documentation, report and video
- ~~Also added toggle show timeout button~~

Reference:

[1] https://stackoverflow.com/questions/70382261/define-function-and-classes-separately-from-their-declarations
[2] https://stackoverflow.com/questions/12542111/separating-class-definition-and-implementation-in-python
[3] https://realpython.com/python-application-layouts/
[4] https://stackoverflow.com/questions/448271/what-is-init-py-for
[5] https://github.com/navdeep-G/samplemod
[6] https://stackoverflow.com/questions/51455765/how-to-build-multiple-py-files-into-a-single-executable-file-using-pyinstaller
[7] https://stackoverflow.com/questions/61713551/how-to-properly-use-time-time
[8] https://github.com/ahmedllshafiey/Raylib-Stopwatch/tree/main
[9] https://electronstudio.github.io/raylib-python-cffi/pyray.html
[10] https://stackoverflow.com/questions/78468857/implementation-of-an-incrementing-variable-after-a-specified-period-of-time

Appendix

Github repository: https://github.com/Jeo0/perking-spaiyc