[8492575]

Abstract

Modern hardware allows uses to interact with external hardware, both inside and outside a computing unit. Connecting devices to a program allows more possibilities to a computer’s potential. By giving it access to more buttons, commands and prompts, you can account for more possibilities. This report will illustrate how I managed to implement this to make a simple game.

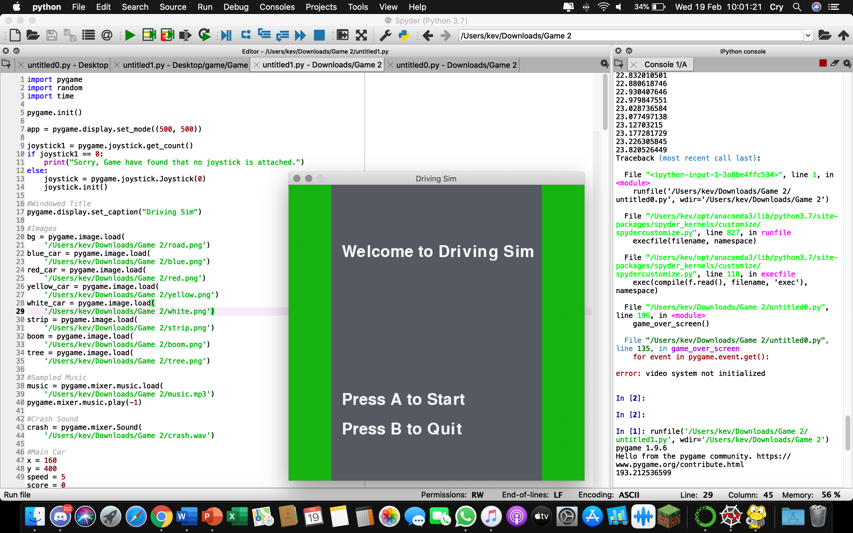
[221ITSC/221ITEL]

[Computer Hardware]

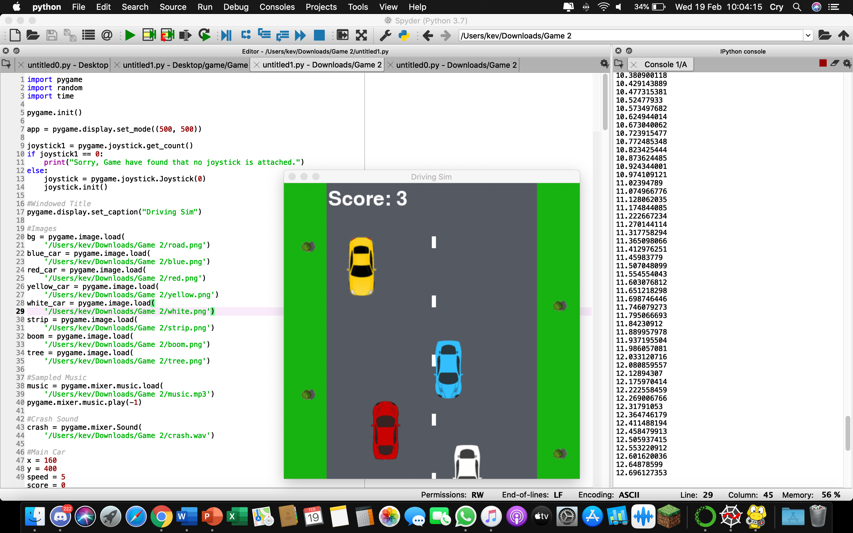


# Summary or Introduction

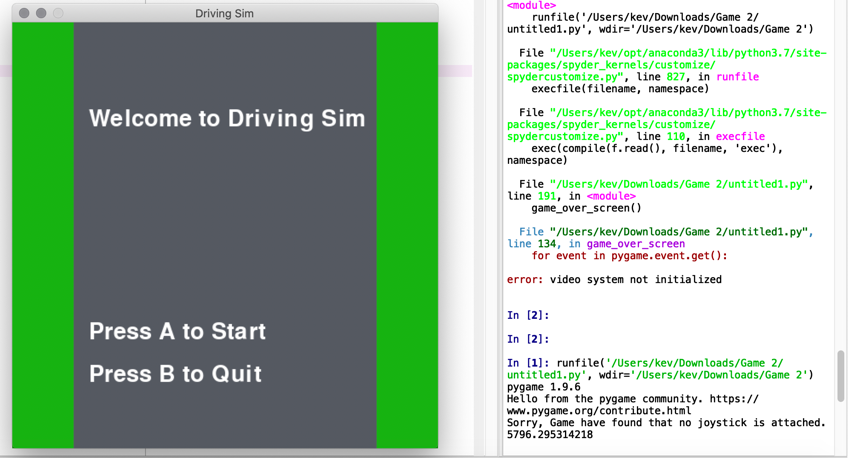
The premise of my program was to use a basic Nintendo Switch Pro Controller, attach it to a python program and using the buttons and joystick, control something around a window. This took me time to grasp the idea I wanted to do but I think my execution went fairly well. I went with an external piece of hardware which interacted with a python program in a Mac OS. After choosing my idea, I narrowed my thoughts down to a vertical side scroller with the theme of car dodging. I’d seen these types of games and wanted to replicate this. Then, by sampling some images off the internet I was able to write some code which worked around these parameters, arriving to a working version of the vertical car scroller game.



A very basic game though it fulfils the parameters of integrating external hardware to make it work. This screenshot shows the program running and some of the basic functions which the user can do.



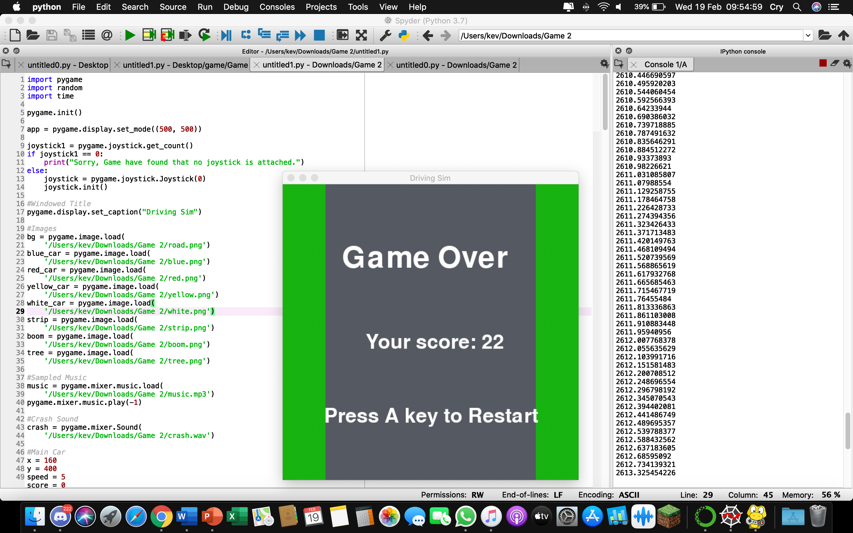
This screenshot shows the code running and the visuals of the game. As soon as the user presses the correct corresponding button on the controller, the game starts and this is what they’ll see.



I’ve also made it so the game detects if the user has a controller connected and had it display an appropriate message that no joystick has been detected and that the game cannot be played.

# Programming language and hardware details: programming language and hardware.

So, I went with python since, currently, it’s the programming language I have the most experience with and I prefer how syntax is structured. My idea was to make a program using ‘pygame’ which is a module in python which a user can import to write video games. This took me time to adjust to commands and new prompts to call on to make a controller connect to a program and have the buttons and analogue stick function as they’re supposed to do; this was one of the hardest parts to adjust to. Having the program run python in a Mac OS gave me trouble sometimes since it’s mostly optimised to run on a Windows type OS but everything still worked as I expected.

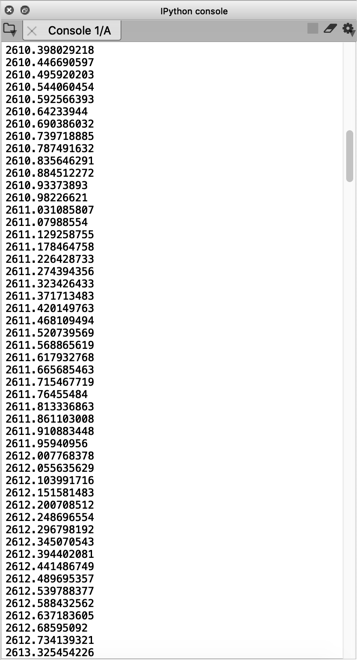
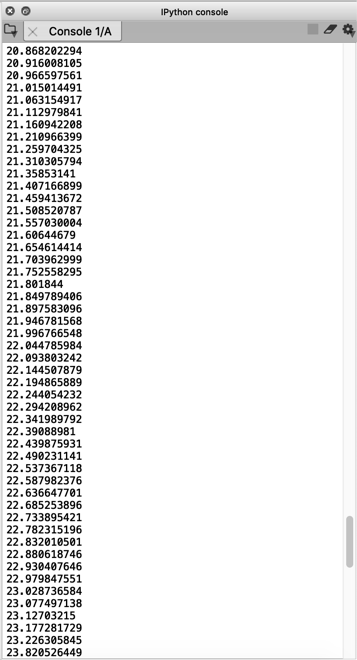


This is a snippet of the code and program running, this illustrates everything I’m importing and the images I’m drawing from my folder. As soon as a collision is made, the end screen will show up and tell the user their score for the round and a button prompt if they want to restart the game. I’ve had to completely design a start and game over screen for the user to make the interface more inviting than just clicking the run button and everything starts moving instantly. I felt like making a wait variable was a good idea.

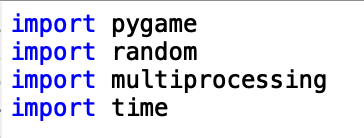
# Application details and design decision: which application are you parallelising? Why you choose this application? What is interesting about this application?

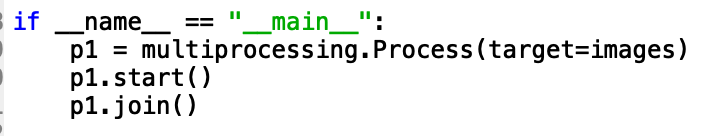
So why did I parallelise it? I wanted something that ran quicker and was more optimised to run the game I was designing. This is done fairly easily through the importing of multiprocessing, I could see instantly that the game ran smoother and quicker through the time function which displayed the time elapsed in the console. I preferred doing multiprocessing over multithreading in python since increasing the speed of the computer won’t always guarantee the speed of a program to increase in return, though by integrating multiprocessing, I am able to use more central processing units and allocate them to the program to improve the rate at which it can run the program and process information and data.

I chose to parallelise it due to entities not loading in quick enough sometimes and I wanted a solution, thus, by making a duplicate, I was able to add several lines of code which allow for it to run faster than usual. In most cases, this is what most people want when designing something.



The difference between these 2 stems from the console, where the left image is before parallelisation and the right is after. What’s clear is how lower the time after-parallelisation takes to complete a task over the left piece of code. This can mean everything when used by people who want to run even more processes at a time.





Through using these lines of code, I was able to import a multiprocessing function which then, let me target a specific area of which to allocate more CPUs to. In my case, I wanted my images to load faster so my defined ‘images’, after the multiprocessing, is able to load my image from my folder much faster and unload them once they leave the screen and recycle the process. This is done with the cars, trees and road.

# Communication Factors for parallel programs

Speed-up in Time Communication

The performance of the application’s images was able to load much faster, unload faster when it left the screen, the code was able to be looped so this can occur until the user decides to hit the prompts which close down the application.

Latency

Aside from the blatant increase in speed from the console, there isn’t much indication that the latency has been improved. The program tends to load faster and the images come in more smoothly than when it isn’t parallelised though mostly everything else runs like the original version.

Visibility

Visibility computing allows for a program to increase rendering speed, tapping into images quicker and loading them without too much hassle. This most likely won’t be the optimum approach for very large scale companies though for simple to intermediate tasks due to lack of parallelism.

Scope

You have a goal of making something run faster, through targeting more CPUs to a designated place, you’re able to make images load faster, processes initialise quicker, commands take effect sooner. In terms of my code, this is the images since the speed of the car can sometimes be different and by having images that can load quicker on standby, it alleviates the need to continuously reboot a single piece of code for one purpose.

Synchronous and Asynchronous

Apart from the instance you run the code, there’s very little synchronous communication going on. Everything from the moment you run it is designed to receive and reply autonomously. I’ve designed it to run, restart and quit in the hands of the controller, without needing to manually close it. This means everything is more efficient and streamlined to fit a working environment.

# Conclusion & Recommendations

I’ve managed to make a program which successfully interacts with both the software and a piece of hardware. I was able to sample the sound and images from different places online and place them into the code to give visuals to the boxes and text. Designing an interface, start screen and a game over screen to give more variety to the program was also a challenge to keep the appeal of a video game. Once I was done with this, I had to parallelise it which optimises the images to run better by making the CPU run multiple images at once, pushing them to the window much quicker than without parallelisation.

If I were to re-do this, I’d choose either a different piece of hardware to interact with or a different type of game for the controller to connect to, since this was a rushed attempt to pair the controller with the code, though successful.

# References

Quora (2018) Can Python directly interact with the hardware?[online]available from: <https://www.quora.com/Can-Python-directly-interact-with-the-hardware> [accessed 19 Feb. 20]

Pygame (2020) *pygame.joystick* [online] available from: <https://www.pygame.org/docs/ref/joystick.html#pygame.joystick.init> [accessed 19 Feb. 20]

YouTube (2017) Pygame Tutorial #1 - Basic Movement and Key Presses [online] available from: [https://www.youtube.com/watch?v=i6xMBig-pP4&list=PLzMcBGfZo4- lp3jAExUCewBfMx3UZFkh5&index=1](https://www.youtube.com/watch?v=i6xMBig-pP4&list=PLzMcBGfZo4-%20lp3jAExUCewBfMx3UZFkh5&index=1) [accessed 19 Feb. 20]

YouTube (2013) Pygame Tricks: Xbox 360 Controller/Joystick Input [ online] available from: <https://www.youtube.com/watch?v=89tLH16_38M&t=> [accessed 19 Feb. 20]

Pygame (2020) Downloads [online] available from: <https://www.pygame.org/download.shtml> [accessed 19 Feb. 20]

ProgramArcadeGames (2020) joystick\_calls.py [online] available from: <http://programarcadegames.com/python_examples/show_file.php?file=joystick_calls.py> [accessed 19 Feb. 20]

Nerd Paradise (2020) Configuring the Joystick in PyGame [online] available from: <https://nerdparadise.com/programming/pygamejoystick> [accessed 19 Feb. 20]

StackOverflow (2017) Pygame Get Value of Button [online] available from: <https://stackoverflow.com/questions/41436629/pygame-get-value-of-button> [accessed 19 Feb. 20]

Pygame (2004) Input Handling [online] available from: <https://www.pygame.org/ftp/contrib/input.html> [accessed 19 Feb. 20]

YouTube (2017) PYTHON PYGAME- ADD A Xbox 360 CONTROLLER to your pygame- Real Easy [online] available from: <https://www.youtube.com/watch?v=IweETFomp3c&t=> [accessed 19 Feb. 20]

ProgramCreek (2020) Python pygame.JOYBUTTONDOWN() Examples [online] available from: <https://www.programcreek.com/python/example/56153/pygame.JOYBUTTONDOWN> [accessed 19 Feb. 20]

TechDifferences (2017) Difference Between Multiprocessing and Multithreading [online] available from: <https://techdifferences.com/difference-between-multiprocessing-and-multithreading.html> [accessed 19 Feb. 20]

Quora (2016) What is the scope of parallel processing in computer science? [online] available from: <https://www.quora.com/What-is-the-scope-of-parallel-processing-in-computer-science> [accessed 19 Feb. 20]

SpringerProfessional (2020) Parallel Implementation and Optimizations of Visibility Computing of 3D Scene on Tianhe-2 Supercomputer [online] available from: <https://www.springerprofessional.de/en/parallel-implementation-and-optimizations-of-visibility-computin/16320846> [accessed 19 Feb. 20]

# Appendices (Include any diagrams or code).

Starting out, I went onto YouTube to research about how to, first make the initial interaction between hardware and software. Then how to make myself a pygame program to run said program. I was able to come across multiple videos of people making games with them coding in button prompts on the keyboard but it took even longer to find help from someone with a controller. I took inspiration from various places, from the basics of how everything should be condensed/cut down to become visually appealing to essential code that makes the game run.

After getting caught up with how everything should be structured, I decided on a side scrolling game with a twist, whereby it’d be vertical. I, since, had ideas ranging from space invaders to run away games, but settled on a car dodging concept. I had to take different videos and make implementations from each one and through trial and error, had myself a good foundation for a car dodging game.

I think re-doing this with a different piece of hardware like a camera or microphone might be a challenge though this also proved to test my familiarity with both hardware and the python coding language.