Visualisation of Vehicle Routes

Original Problem

The original script was written in python which used the matplotlib library to manually create an image for each frame and stitch each frame into one video via FFMPEG. However, this method took a significant amount of time and was not dynamic. Furthermore, an interactive interface was also needed to showcase the algorithms to users.

Analysis of Alternatives

To determine the best language for this project, a select number of suitable languages were shortlisted and further explored based on their advantages and disadvantages as listed below:

Language	Advantages	Disadvantages
C++	• Speed • Enforced object-oriented programming	• No memory management tools. • Personal: Not Familiar with the language, moderate learning curve
R	• Inbuilt animation slider and playback • Clean, aesthetic, and powerful outputs	• Automatic memory garbage collector • Personal: Not Familiar with the language, steep learning curve • Orientated for data visualisations rather than application building. • Animation smooth transitions are not possible for the project requirements
Python	• Personal: Can leverage existing experience to build the application • Large standard library • Flexibility and readability	• Automatic memory garbage collector • "Slow" execution due to it being an interpreted language
HTML/CSS/JS	• High flexibility and scalability • Incorporation into other web applications • Personal: Can leverage existing experience to build the application	• Dependant on the web browser which may affect performance as browsers usually limit the amount of access to system resources.

For Python, its plotting/animation options were considered and evaluated below:

Library	Advantages	Disadvantages
matplotlib	• Personal: Can leverage existing experience to lessen learning curve. • Project: existing compatible code • Graphs appearance is clean and modern	• Slower time to plot than other libraries. • Verbose library code
PyQtGraph	• Runs much faster • No dynamic animation	• Newer and hence less documentation and features particularly in creating dynamic animations.

For Python, the following UI development libraries for the interactive GUI were evaluated below:

Library	Advantages	Disadvantages
PyQt	• Better Documentation • Qt Designer to speed up GUI development	• Pyqt5 and 6 only compatible with python 3 • GPL licensing prevents commercialisation of product unless source code is given. • Requires c++ knowledge
PySide	• Backward compatibility with python 2.7 • Qt Designer to speed up GUI development. • LGPL licensing does not have any restriction on commercialisation	• Poor documentation • Requires c++ knowledge

Tkinter	• Simple and fast to implement.
	• No additional package installation as it is bundled with Python.
	• No additional licenses to consider.
		• Does not have advanced widgets.
		  ○ Database interface
		  ○ Mp4 player
		• Does not have a QtDesigner application which can speed up complex GUI development

As a result, given the project and technological constraints, Python with PyQt and matplotlib was the best choice.

Software Development

After determining the software requirements from the client, the following backlog items were created, with the highest priority at the top.

- Dynamic animation player with Pause and play buttons and sliders.
- Point and Click Interactive Plotter to generate Solver Input
- Integrate Solver into Application
- Map and Solver Solutions Database to hold maps and existing solver outputs.

The first step was identifying the packages involved, installing them and testing them for compatibility. The following package list can be found in the requirements.txt file at the root. To isolate the packages from external interference, a virtual environment was built to hold them.

In one branch, the visualisation was optimised, at first, I tried a direct low level matplotlib API implementation to achieve animation but was unsuccessful. A rasterization technique called Blit Blitting (a.k.a blitting) where the canvas is screenshotted and saved to use in future frames, decreases the time to make the frames as it prevents time consuming redrawing of objects. However, using the API to do this manually was complex and ultimately suffered from many undesirable visualisation errors. Hence, it was deemed infeasible. As an alternative, matplotlib's animation wrapper FuncAnimation was used and adapted with existing scripts. As a result, the animation can be dynamically generated in under **1 second**. After the FuncAnimation implementation was complete, the software processing was reviewed, and bloat code was removed, and excessive copy functions were eliminated leading to faster frame generation.

In another branch called editor-UI, the purpose of the branch was to focus on building the interactive plotter and the rest of the application for a smoother experience. To improve the quality of the application, the QtDesigner was used to create the overall architecture of the application and each file was converted into a python file and customised based on the project's needs. Whereas custom components such as the matplotlib widgets were manually created and integrated.

Both branches will be merged to complete the application.

Further implementation

As of the current version, it suffers from inconsistent FPS Issues related to the generation of frames. This can be attributed to the number of objects on the screen. Currently two measures have been put in place such as turning off cached data frames for the FuncAnimation which has slightly improved the frame creation time as the frames do not have to be cached but it comes with the downside that the video that may be created from these frames will take longer. Another technique that has been attempted with moderate success is FPS frame lock technique, where the time difference is calculated from the time taken to produce the frame and the average time and the script will pause for a multiple of that time difference to ensure a smooth animation playback. Other approaches to speeding up the animation frame update were tried such as hiding the visibility of the objects and line paths rather than removing them but were unsuccessful due to their synchronicity as it would lead to tasks that were being picked up to be delayed to 1-3 seconds.

In the future, further iterations for the project could include the following:

- Current Configuration
    - Cython use or Numba Python compiler to optimise machine code runtime.
    - Multi-Threading/multi-processing to better utilise available system resources.
- Matplotlib alternative: PyQtGraph
    - Most promising Python alternative: reasons can be read in the graphing libraries table.
- Qtpy abstraction layer to switch between pyside and pyqt with ease.
- Python alternative: C++ Implementation
    - More reasons can be explained in the languages table.

Files

Figma Wireframes: https://www.figma.com/file/ZYw2vMJITS8YiyGfwnSc17/Visualisation-of-Vehicle-Routes?type=design&node-id=0%3A1&mode=design&t=7vc3EwRcWrchEleQ-1

Github: https://github.com/benwoang/VehicleRoutesVisualisation