ĐẠI HỌC ĐÀ NẴNG TRƯỜNG ĐẠI HỌC KINH TẾ

KHOA THƯƠNG MẠI ĐIỆN TỬ

MÔ PHỎNG VÀ TỐI ƯU HÓA BÃI ĐỖ XE TRƯỜNG ĐẠI HỌC

Mã chủ đề: PA - Prescriptive Analytics

GVHD: Lê Diên Tuấn

Lớp học phần: MIS3041_47K29.1

Thành viên:

Hoàng Như Vương (nhóm trưởng - 0977829788)	20%
Trần Duy Luân	20%
Nguyễn Đỗ Kim Thi	20%
Nguyễn Thị Anh Thơ	20%
Lê Thị Kim Yến	20%

Đà Nẵng, ngày 18 tháng 4 năm 2024

Mục lục

I. Giới thiệu

Việc quản lý hệ thống bãi đỗ xe của các trường đại học là một vấn đề quan trọng ảnh hưởng đến trải nghiệm của sinh viên. Thời gian chờ đợi khi ra khỏi bãi đỗ xe, đặc biệt vào giờ cao điểm, có thể gây ra sự bất mãn của sinh viên. Để giải quyết vấn đề này, dự án mô phỏng hệ thống bãi đỗ xe trường đại học được thực hiện nhằm cung cấp những đánh giá và khuyến nghị cho ban quản trị về việc nên đầu tư tăng thêm số cổng ra khỏi và lượng nhân viên để cải thiện sự hài lòng của sinh viên hay không.

Ý tưởng của dự án là sử dụng mô hình mô phỏng để tái tạo quá trình quản lý việc ra khỏi bãi đỗ xe, với các biến đầu vào như số lượng cổng từ và cổng thẻ giấy, xác suất lỗi của từng loại thẻ, thời gian xử lý của từng loại cổng và giờ cao điểm. Mục tiêu là tối ưu hóa tổng thời gian chờ đợi của mỗi xe, đặc biệt là vào giờ cao điểm, bằng cách điều chỉnh số lượng cổng và phương pháp xử lý thẻ.

Kết quả từ dự án sẽ cung cấp cơ sở dữ liệu để quyết định liệu cần xây thêm cổng từ hay không, và nếu có, thì cần xây thêm bao nhiêu và loại cổng nào là phù hợp nhất để tối ưu hóa hoạt động của bãi đỗ xe, giúp cải thiện trải nghiệm và sự hài lòng của sinh viên khi sử dụng bãi đỗ xe.

II. Cơ sở lý thuyết

1. Business Analytics và Prescriptive Analytics

Theo trong Prescriptive analytics: Literature review and research challenges, Business analytics đề cập đến việc sử dụng rộng rãi dữ liệu, thu thập từ các nguồn khác nhau, phân tích định lượng và thống kê, các mô hình dự đoán và giải thích, cũng như quản lý dựa trên bằng chứng để ra quyết định và hành động phù hợp với các bên liên quan (Davenport & Harris, 2007; Soltanpoor & Sellis, 2016). Để thực hiện điều này, phân tích kinh doanh sử dụng các phương pháp từ các lĩnh vực khoa học dữ liệu, nghiên cứu hoạt động, học máy và hệ thống thông tin (Mortenson, Doherty, & Robinson, 2015). Theo nghĩa này, business analytics không chỉ là về các mô hình mô tả mà còn về các mô hình có thể cung cấp các hiểu biết có ý nghĩa và hỗ trợ các quyết định về hiệu suất kinh doanh.

Business analytics được chia thành ba giai đoạn chính, được đặc trưng bởi độ khó, giá trị và độ thông minh khác nhau (Akerkar, 2013; Krumeich, Werth, & Loos, 2016; Šikšnys & Pedersen, 2016):

(i) descriptive analytics, trả lời các câu hỏi "Điều gì đã xảy ra?", "Tại sao nó lại xảy ra?", nhưng cũng "Điều gì đang xảy ra bây giờ?"

(ii) predictive analytics, trả lời các câu hỏi "Điều gì sẽ xảy ra?" và "Tại sao nó sẽ xảy ra?" trong tương lai

(iii) prescriptive analytics, trả lời các câu hỏi "Tôi nên làm gì?" và "Tại sao tôi nên làm điều đó?".

So với phân tích mô tả và dự đoán, prescriptive analytics vẫn còn chưa phát triển bằng (Gartner, 2017). Tuy nhiên, gần đây, prescriptive analytics đã thu hút ngày càng nhiều sự quan tâm nghiên cứu (Larson & Chang, 2016). Nó đã được coi là bước tiếp theo để tăng cường sự phát triển của data analytics và dẫn đến việc ra quyết định được tối ưu, sớm, kịp thời, để cải thiện hiệu quả kinh doanh (den Hertog & Postek, 2016; Gartner, 2017).

Prescriptive analytics, là loại phân tích kinh doanh phức tạp nhất và có thể mang lại thông tin và giá trị lớn nhất cho các doanh nghiệp (Šikšnys & Pedersen, 2016). Prescriptive analytics nhằm mục đích đề xuất (định hướng) các lựa chọn quyết định tốt nhất nhằm tận dụng tương lai được dự đoán bằng cách sử dụng lượng lớn dữ liệu (Šikšnys & Pedersen, 2016).

Prescriptive analytics có hai cấp độ can thiệp của con người: *hỗ trợ quyết định*, ví dụ như đưa ra khuyến nghị; *tự động hóa quyết định*, ví dụ như tự động thực hiện hành động được đề xuất (Gartner, 2017). Hiệu quả của các đề xuất phụ thuộc vào mức độ các mô hình này kết hợp dữ liệu cấu trúc và dữ liệu không cấu trúc, mức độ nắm bắt lĩnh vực đang nghiên cứu và nắm bắt tác động của các quyết định được phân tích (Basu, 2013; Šikšnys & Pedersen, 2016).

2. Mô phỏng sự kiện rời rạc - Discrete Event Simulation

Mô phỏng sự kiện rời rạc (DES) là một công cụ mạnh mẽ để phân tích hiệu suất của các hệ thống phức tạp. DES là một công cụ vô giá để mô hình hóa và hiểu hành vi của các hệ thống thực tế, từ sản xuất và kho bãi đến luồng giao thông và vận tải. DES có thể giúp xác định điểm nghẽn, cải thiện hiệu quả hoạt động và tối ưu hóa chi phí.

Mặc dù chúng ta không thể dự đoán được tương lai, nhưng chúng ta có thể dự đoán một hệ thống sẽ hoạt động như thế nào trong các điều kiện khác nhau. Bằng cách nhìn vào phản ứng của hệ thống trước sự thay đổi của tình huống, chúng ta có thể có những cái nhìn về tính chất của hệ thống và dự đoán các đầu ra.

DES là một loại mô phỏng máy tính mô hình hóa hành vi của một hệ thống dưới dạng một loạt các sự kiện xảy ra tại các thời điểm rời rạc. Đây là một kiểu mô hình hóa mối quan hệ trong hệ thống có tích hợp các mối quan hệ của các quy trình, hàng đợi và ra quyết định. DES được dùng để giả lập hành vi của những hệ thống động qua thời gian. Bằng cách mô hifnnh hóa hệ thống dưới hạng một chuỗi các sự kiện, DES có thể xác định những biến có tác động quan trọng lên hệ thống và tính toán hiệu suất của mô hình một cách nhanh chóng và tối ưu.

Bằng cách sử dụng DES, các doanh nghiệp có thể đưa ra những quyết định có cơ sở tốt hơn nhằm tối đa hóa năng suất và lợi nhuận. Hơn nữa, DES có thể giúp dự đoán và ngăn chặn các lỗi hệ thống tốn kém, đảm bảo chất lượng dịch vụ cao hơn.

DES cho phép chúng ta xác định những thành phần then chốt nhất của một hệ thống, như những thành phần ảnh hưởng đến hiệu suất, chi phí và độ ổn định. Nó cũng cung cấp sự linh hoạt để kiểm tra các kịch bản khác nhau và xác định giải pháp hiệu quả và tiết kiệm chi phí nhất. Ví dụ, bằng cách sử dụng DES, bạn có thể kiểm tra số lượng nhân viên tối ưu cần thiết để xử lý một khối lượng công việc cụ thể hoặc xác định bố trí tối ưu nhất cho một kho hàng.

3. Thư viện SimPy

a. Giới thiệu thư viện SimPy

SimPy là một thư viện mô phỏng hệ thống sự kiện rời rạc (Discrete-Event Simulation) được phát triển bằng ngôn ngữ Python. SimPy cung cấp các công cụ để mô phỏng và phân tích các hệ thống phức tạp, như hàng đợi, hệ thống sản xuất, mạng viễn thông và nhiều hơn nữa.

b. Những ý tưởng và thành phần cơ bản của SimPy

SimPy là một thư viện mô phỏng sự kiện rời rạc. Hành vi của các thành phần (như phương tiện, khách hàng hoặc tin nhắn) được mô hình hóa bằng các `processes`. Tất cả các process đều tồn tại trong một environment. Các process tương tác với environment và tương tác với nhau thông qua các events.

Trong suốt vòng đời của mình, các process tạo ra các events và `yield` (trả về) một khoảng thời gian để đợi các event xuất hiện

Khi một process định trả về một event, process đó sẽ bị tạm hoãn. SimPy tiếp tục process khi event đó đã hoàn tất sau một khoảng thời gian nhất định (hay còn nói là event đã được xử lí xong).

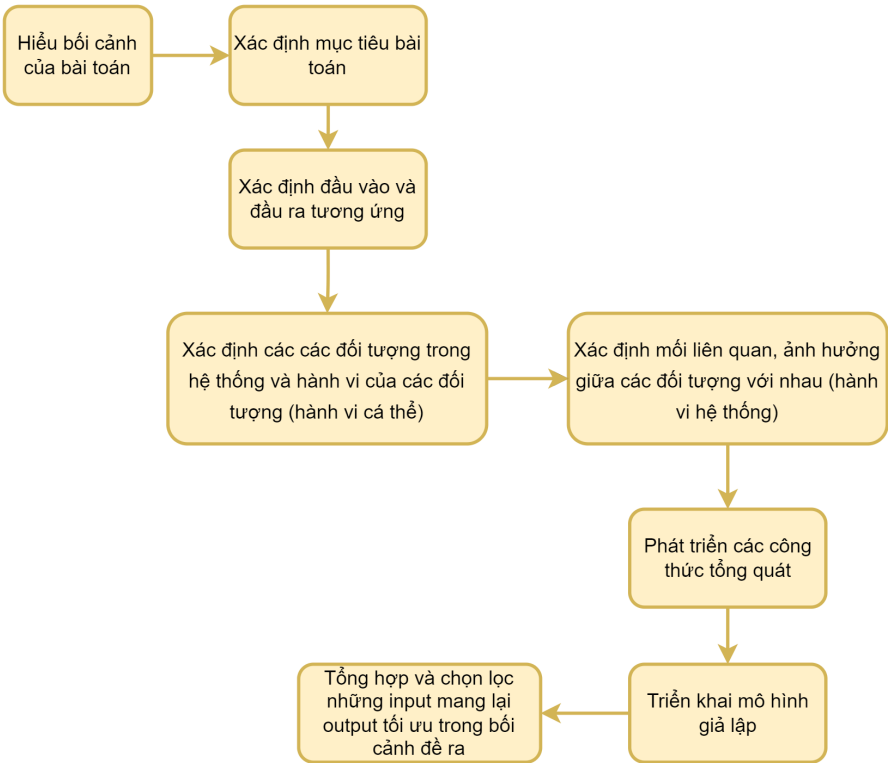
Một loại sự kiện quan trọng là Timeout. Những events thuộc loại này xuất hiện sau một khoảng thời gian (trong môi trường giả lập) nhất định đã trôi qua. Chúng cho phép các process tạm ngủ (giữ trạng thái hiện tại) trong một khoảng thời gian nhất định. Một sự kiện Timeout và tất cả các loại sự kiện khác có thể được tạo bằng cách gọi method phù hợp của Environent chứa các process đó (ví dụ, Environment.Timeout())

SimPy cung cấp class Resources để giúp mô hình hóa các vấn đề khi nhiều process muốn sử dụng một tài nguyên có sức chứa giới hạn (ví dụ, xe ô tô ở trạm xăng với số vòi xăng có giới hạn).

Khi một process muốn sử dụng tài nguyên, nó sẽ gọi method request() của tài nguyên để tạo ra một sự kiện, cho phép process chờ đợi cho đến khi tài nguyên khả dụng. Nếu process được tiếp tục (khi đã chờ xong), nó sẽ "sở hữu" tài nguyên cho đến khi process chủ động release (trả lại) tài nguyên đó.

Khi người dùng release một tài nguyên, quá trình chờ đợi tiếp theo sẽ được tiếp tục và "sở hữu" một trong các slot của tài nguyên. Tài nguyên Resource sắp xếp các process đang chờ theo FIFO (first in - first out).

III. Framework



IV. Bối cảnh và mục tiêu nghiên cứu

1. Thông tin bối cảnh

Một trường đại học sử dụng đồng thời cả thẻ từ và thẻ giấy để quản lý việc gửi xe của sinh viên tại bãi đỗ xe. Hiện tại, ngôi trường này có 2 (hai) làn xe cho thẻ từ và 2 (hai) làn cho vé giấy. Tuy nhiên, tình trạng kẹt xe và thời gian chờ đợi lâu đang gây ra sự không hài lòng của sinh viên. Trường muốn giải quyết vấn đề này.

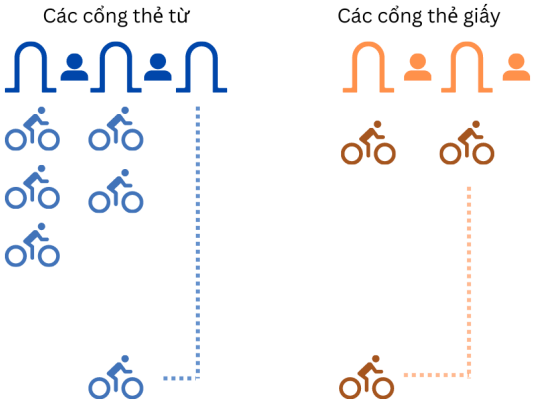
Giả định là đã có sẵn dữ liệu về quy trình và các phân phối của các biến liên quan trong quá trình ra vào bãi đỗ xe.

2. Mục tiêu bài toán

Xây dựng hệ thống mô phỏng dòng xe qua cổng ra vào bãi đỗ xe nhằm mô phỏng các tình huống với số lượng cổng và nhân viên khác nhau để đưa ra khuyến nghị tối ưu.

- Tính toán thời gian chờ trung bình theo các tiêu chí khác nhau như giờ cao điểm, giờ thấp điểm, loại cổng (từ hay giấy).
- Đánh giá lượng tài nguyên về số lượng nhân viên và số cổng mà trường cần đầu tư thêm.

V. Nội dung mô hình mô phỏng



1. Input và Output của hệ thống

Đầu vào (nhằm tạo các tình huống khác nhau)

- Số cổng từ $gate\ amt_{RFID} \in (1, 4)$
- Số cổng giấy $gate\ amt_{paper}$ (từ 1-2)$\in (1, 2)$
- Số nhân viên cổng từ $emp\ amt_{RFID} \in (\frac{gate\ amt_{RFID}}{2}, gate\ amt_{RFID})$

  Mỗi nhân viên cổng từ kiểm soát được tối đa 2 cổng cùng lúc, tối thiểu 1 cổng cùng lúc.

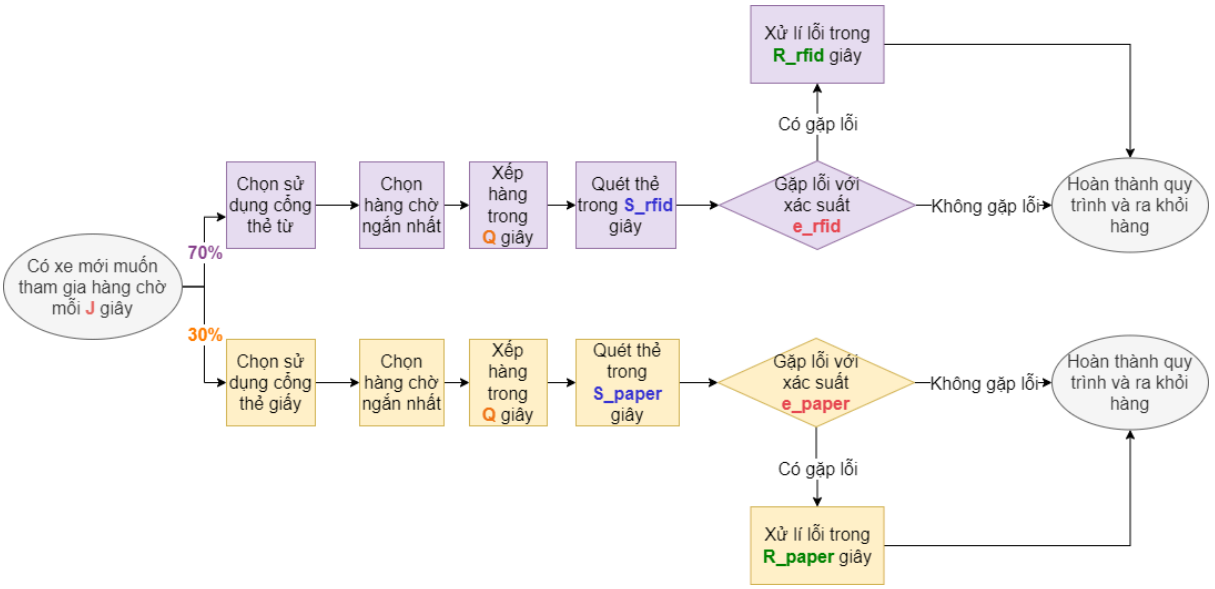
- Số nhân viên cổng giấy $emp\ amt_{paper} = gate\ amt_{paper}$

  Mỗi cổng thẻ giấy yêu cầu 1 nhân viên.

Đầu ra: Thời gian chờ W trung bình

- Thời gian chờ trung bình cho toàn thể sinh viên
- Thời gian chờ trung bình của mỗi loại thẻ từ, thẻ giấy
- Thời gian chờ trung bình tại các giờ cao điểm

2. Hành vi của hệ thống



Tổng thời gian chờ W (wait time) **là từ lúc xe tham gia vào hàng chờ cho đến lúc ra khỏi cổng.** Trong đó, bao gồm thời gian xếp hàng, thời gian quét thẻ, và thời gian xử lý lỗi nếu phát sinh.

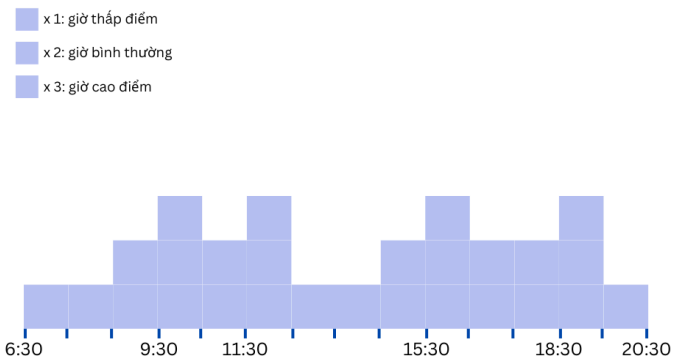
Bãi giữ xe mở cửa từ 6:30 đến 20:30 hàng ngày. Mỗi sinh viên có thẻ từ hoặc giấy đã mua từ trước.

70% sinh viên dùng thẻ từ và 30% sinh viên dùng thẻ giấy

Sinh viên sẽ tham gia vào làn có ít xe đang xếp hàng nhất, nếu chiều dài của các làn là bằng nhau, chọn ngẫu nhiên một trong số các làn.

Cứ mỗi J (join) giây, sẽ có một xe mới muốn tham gia vào quy trình.

- $J \sim N(3, 0.5)$ vào giờ cao điểm
- $J \sim N(30, 1)$ vào giờ bình thường
- $J \sim N(60, 10)$ vào giờ thấp điểm



Thời gian cao-bình thường - thấp điểm trong ngày

Thời gian xếp hàng là Q (queue time) giây, là từ lúc tham gia hàng chờ cho đến lúc được quét thẻ.

Thời gian kiểm tra, quét thẻ là S (scan time) giây.

- Với thẻ từ, $S_{RFID} \sim U(9, 13)$
- Với thẻ giấy, $S_{paper} \sim U(10, 17)$

Một nhân viên có thể kiểm soát lỗi cho tối đa 2 cổng thẻ từ cùng lúc. Phân bổ nguồn lực nhân viên cho mỗi cột là $emp\ rate = emp\ amt/gate\ amt$.
R giây là thời gian cần thiết để xử lí lỗi thẻ nếu phát sinh.

- Với thẻ từ, tại $emp\ rate = 1$ (một nhân viên kiểm soát 1 cổng), thì ta có $R_{RFID} \sim N(15, 5)$
- Với thẻ từ, tại $0.5 < emp\ rate < 1$, thì ta có $R_{RFID} \sim N(20, 5)$
- Với thẻ từ, tại $emp\ rate = 0.5$ (một nhân viên kiểm soát 2 cổng), thì ta có $R_{RFID} \sim N(30, 5)$
- Với thẻ giấy, tại $emp\ rate = 1$, thì ta có $R_{paper} \sim N(10, 2)$

Xác suất kiểm tra thẻ bị lỗi là e

- Với thẻ từ, $e_{RFID} = 1\%$
- Với thẻ giấy, $e_{paper} = 3.5\%$

3. Các công thức tổng quát

Công thức tổng quát cho thời gian chờ của một người:

Công thức 1:

$$W = Q + S + E$$

$$E = e.R$$

Chúng ta sẽ áp dụng công thức 1 để tính thời gian chờ của một người tại những tình huống cụ thể - theo khung giờ và theo sự lựa chọn cổng của người này.

Trong đó, thời gian xếp hàng Q của người thứ n trong một hàng là:

Công thức 2:

$$Q_n = Q_{n-1} + (S_{n-1} + E_{n-1})$$

$$Q_1 = 0$$

Công thức tổng quát cho thời gian chờ trung bình của toàn thể sinh viên trong ngày là

Công thức 3:

$$\overline{W} = \frac{\sum_{i=1}^{0.7N} W_i^{RFID} + \sum_{i=1}^{0.3N} W_i^{paper}}{N}$$

Trong đó:

$\overline{w}$ là thời gian chờ trung bình của toàn thể sinh viên trong cả ngày

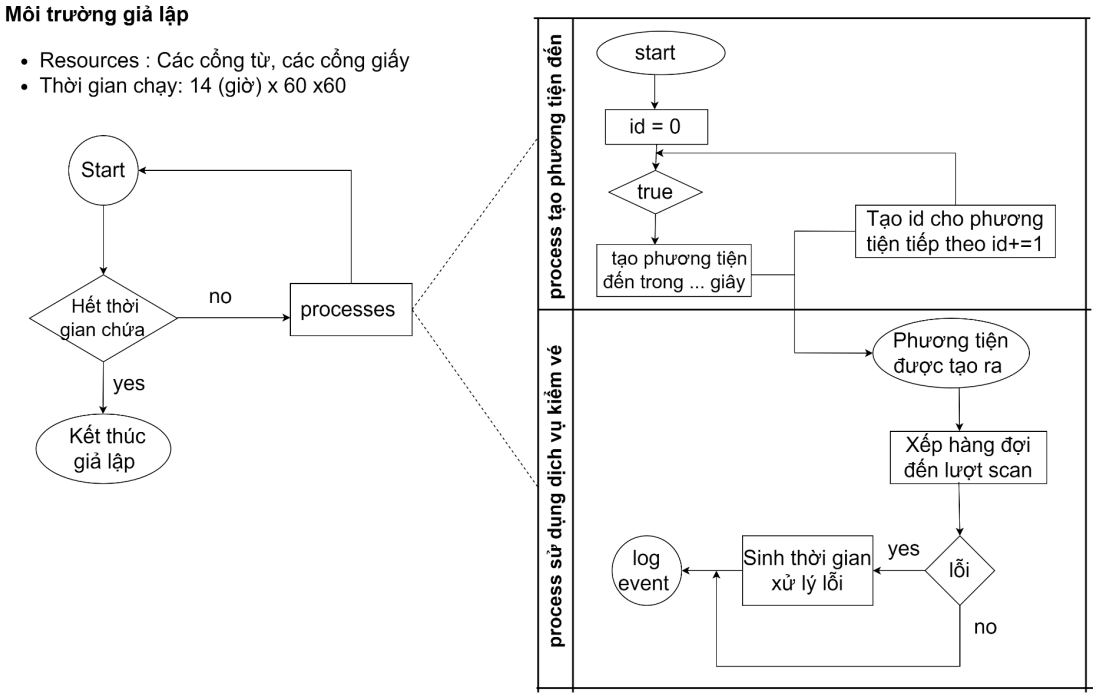
N là toàn bộ sinh viên đã tham gia quy trình trong cả ngày

W_i^{RFID} là thời gian chờ của sinh viên thứ i trong số sinh viên chọn cổng thẻ từ của ngày

W_i^{paper} là thời gian chờ của sinh viên thứ i trong số sinh viên chọn cổng thẻ giấy của ngày

VI. Lập trình chương trình giả lập

1. Luồng làm việc của chương trình



2. Diễn giải code

a. Input và thiết lập/ràng buộc cho môi trường giả lập

Biến	Giá trị	Diễn giải
RFID_GATE_LINES	2	Số lượng cổng thẻ từ (RFID)
PAPER_GATE_LINES	2	Số lượng cổng thẻ giấy
PAPER_EMPS_PER_LINE	1	Số lượng nhân viên tối thiểu trên mỗi cổng thẻ giấy
RFID_EMPS_PER_LINE	0.5	Số lượng nhân viên tối thiểu trên mỗi cổng thẻ từ
RFID_SELECTION_RATE	0.6	Tỷ lệ chọn cổng thẻ từ (RFID) của sinh viên
RFID_SCAN_TIME_MIN	9	Thời gian quét thẻ cổng thẻ từ tối thiểu (giây)
RFID_SCAN_TIME_MAX	13	Thời gian quét thẻ cổng thẻ từ tối đa (giây)

PAPER_SCAN_TIME_MIN	10	Thời gian quét thẻ cổng thẻ giấy tối thiểu (giây)
PAPER_SCAN_TIME_MAX	17	Thời gian quét thẻ cổng thẻ giấy tối thiểu (giây)
JOIN_RATE_HIGH_MEAN	3	Trung bình số lượng xe tham gia xếp hàng mỗi giây giờ CAO ĐIỂM
JOIN_RATE_HIGH_STD	0.5	Độ lệch chuẩn số lượng xe tham gia xếp hàng mỗi giây giờ CAO ĐIỂM
JOIN_RATE_AVG_MEAN	30	Trung bình số lượng xe tham gia xếp hàng mỗi giây giờ BÌNH THƯỜNG
JOIN_RATE_AVG_STD	1	Độ lệch chuẩn lượng xe tham gia xếp hàng mỗi giây giờ BÌNH THƯỜNG
JOIN_RATE_LOW_MEAN	60	Trung bình số lượng xe tham gia xếp hàng mỗi giây giờ THẤP ĐIỂM
JOIN_RATE_LOW_STD	10	Độ lệch chuẩn số lượng xe tham gia xếp hàng mỗi giây giờ THẤP ĐIỂM
ERROR_RATE_RFID	0.01	Tỷ lệ lỗi khi quét thẻ từ
ERROR_RATE_PAPER	0.035	Tỷ lệ lỗi khi quét thẻ giấy

b. Các hàm hỗ trợ

1. get_current_time

```
def get_current_time(elapsed_seconds):
    start_time = datetime.strptime('06:30:00', '%H:%M:%S')
    current_time = start_time + timedelta(seconds=elapsed_seconds)
    formatted_current_time = current_time.strftime('%H:%M:%S')
    return formatted_current_time
```

Tham Số Đầu Vào:

- **elapsed_seconds**: Đây là tham số đầu vào đại diện cho số giây đã trôi qua kể từ thời điểm bắt đầu của môi trường giả lập, mặc định thời điểm bắt đầu là 6:30:00.

Hoạt động :

- Hàm này sẽ nhận tham số **elapsed_seconds** và tính toán thời gian hiện tại dựa trên số giây đã trôi qua.
- Thời gian hiện tại được tính toán bằng cách thêm số giây đã trôi qua vào thời điểm bắt đầu của môi trường giả lập.

Kết quả của hàm

- Kết quả cuối cùng được định dạng lại để đảm bảo rằng nó được hiển thị dưới dạng chuỗi thời gian theo định dạng "HH:MM:SS".

2. check_traffic_status

```
def check_traffic_status(current_time):
    if current_time < "08:30:00":
        return "low"
    elif "08:30:00" <= current_time < "09:30:00":
        return "avg"
    elif "09:30:00" <= current_time < "10:30:00":
        return "high"
    elif "10:30:00" <= current_time < "11:30:00":
        return "avg"
    elif "11:30:00" <= current_time < "12:30:00":
        return "high"
    elif "12:30:00" <= current_time < "14:30:00":
        return "low"
    elif "14:30:00" <= current_time < "15:30:00":
        return "avg"
    elif "15:30:00" <= current_time < "16:30:00":
        return "high"
    elif "16:30:00" <= current_time < "18:30:00":
        return "avg"
    elif "18:30:00" <= current_time < "19:30:00":
        return "high"
    else:
        return "low"
```

Tham Số Đầu Vào:

- **current_time**: Thời gian hiện tại được đưa vào dưới dạng chuỗi trong định dạng 'HH:MM:SS'.

Hoạt động :

- Phân chia ngày thành các khoảng thời gian và xác định trạng thái giao thông tương ứng cho mỗi khoảng:
  i. 'low': Từ 06:30:00 đến 08:30:00 và từ 12:30:00 đến 14:30:00.
  ii. 'avg' (trung bình): Từ 08:30:00 đến 09:30:00, từ 10:30:00 đến 11:30:00, từ 14:30:00 đến 15:30:00 và từ 16:30:00 đến 18:30:00.
  iii. 'high' (cao): Từ 09:30:00 đến 10:30:00, từ 11:30:00 đến 12:30:00, từ 15:30:00 đến 16:30:00 và từ 18:30:00 đến 19:30:00.

Kết quả của hàm

- Các kết quả ('low', 'avg', 'high')

c. Các hàm sinh thời gian

1. generate_arrival_time

```
def generate_arrival_time(env):
    current_time = get_current_time(env.now)
    traffic_status = check_traffic_status(current_time)
    if traffic_status == 'high':
        arrival_time = max(0, random.normalvariate(JOIN_RATE_HIGH_MEAN,
JOIN_RATE_HIGH_STD))
    elif traffic_status == 'low':
        arrival_time = max(0, random.normalvariate(JOIN_RATE_LOW_MEAN,
JOIN_RATE_LOW_STD))
    else: # NORMAL
        arrival_time = max(0, random.normalvariate(JOIN_RATE_AVG_MEAN,
JOIN_RATE_AVG_STD))
    return arrival_time, traffic_status
```

Tham Số Đầu Vào:

- env (môi trường giả lập)

Hoạt động :

- Sử dụng check_traffic_status(env) để xác định trạng thái giao thông ở thời điểm hiện tại.
- Dựa trên trạng thái giao thông, hàm sẽ tạo ra một thời gian đến (arrival_time) sử dụng phân phối chuẩn (normal distribution) với các tham số khác nhau:
  i. Nếu trạng thái giao thông là "high", sẽ sử dụng JOIN_RATE_HIGH_MEAN làm trung bình và JOIN_RATE_HIGH_STD làm độ lệch chuẩn.

ii. Nếu trạng thái giao thông là "low", sẽ sử dụng JOIN_RATE_LOW_MEAN làm trung bình và JOIN_RATE_LOW_STD làm độ lệch chuẩn.

iii. Nếu trạng thái giao thông là "avg" (normal), sẽ sử dụng JOIN_RATE_AVG_MEAN làm trung bình và JOIN_RATE_AVG_STD làm độ lệch chuẩn.

- Hàm đảm bảo rằng thời gian đến luôn lớn hơn hoặc bằng 0 (bằng cách sử dụng hàm max(0, ...)) vì không thể có thời gian đến âm.

Kết quả của hàm

- arrival_time (thời gian đến của một sự kiện)
- traffic_status (trạng thái giao thông)

2. generate_scan_time

```
def generate_scan_time(card_type):
    if card_type == 'RFID':
        return random.uniform(RFID_SCAN_TIME_MIN, RFID_SCAN_TIME_MAX)
    elif card_type == 'paper':
        return random.uniform(PAPER_SCAN_TIME_MIN, PAPER_SCAN_TIME_MAX)
```

Tham Số Đầu Vào:

- **card_type**: Đây là tham số đầu vào đại diện cho loại thẻ của phương tiện, có thể là "RFID" hoặc "paper".

Hoạt động :

- Hàm này sẽ sinh ra thời gian quét thẻ dựa trên loại thẻ của phương tiện
- Nếu loại thẻ là "RFID", thời gian quét thẻ sẽ được sinh ngẫu nhiên trong khoảng từ **RFID_SCAN_TIME_MIN** đến **RFID_SCAN_TIME_MAX**.
- Nếu loại thẻ là "paper", thời gian quét thẻ sẽ được sinh ngẫu nhiên trong khoảng từ **PAPER_SCAN_TIME_MIN** đến **PAPER_SCAN_TIME_MAX**.

Đầu ra của hàm

- Hàm trả về thời gian quét thẻ được sinh ra dưới dạng một số thực, biểu diễn số giây mà quá trình quét thẻ mất.

3. generate_error_correction_time

```
def generate_error_correction_time(card_type):
    if card_type == 'RFID':
        if RFID_EMPS_PER_LINE == 1:
            ERROR_CORRECTION_TIME = max(0, random.normalvariate(15, 5))
```

```
        elif 0.5 < RFID_EMPS_PER_LINE < 1:
            ERROR_CORRECTION_TIME = max(0, random.normalvariate(20, 5))
        else:
            ERROR_CORRECTION_TIME = max(0, random.normalvariate(30, 5))
    elif card_type == 'paper':
        ERROR_CORRECTION_TIME = max(0, random.normalvariate(10, 2))

    return ERROR_CORRECTION_TIME
```

Tham số đầu vào

- **card_type**: Đây là tham số đầu vào đại diện cho loại thẻ của phương tiện, có thể là "RFID" hoặc "paper".

Hoạt động

- Hàm này sẽ sinh ra thời gian cần để sửa lỗi sau khi phát hiện sự cố lỗi khi quét thẻ, dựa trên loại thẻ của phương tiện.
- Nếu loại thẻ là "RFID", thời gian sửa lỗi sẽ được sinh ngẫu nhiên từ một phân phối chuẩn với giá trị trung bình và độ lệch chuẩn được xác định trước.
  Tại vì có sự thay đổi nhân viên trực tại cổng RFID nên thời gian sửa lỗi sẽ khác nhau (trường hợp 1 cổng = 1 nhân viên, 2 cổng = 1 nhân viên)
- Nếu loại thẻ là "paper", thời gian sửa lỗi sẽ được sinh ngẫu nhiên từ một phân phối chuẩn khác với các giá trị trung bình và độ lệch chuẩn tương ứng.

Đầu ra của hàm

- Hàm trả về thời gian cần để sửa lỗi sau khi phát hiện sự cố lỗi, biểu diễn dưới dạng một số thực, biểu diễn số giây mà quá trình sửa lỗi mất.

d. Các hàm quy trình

1. is_error

```
def is_error(card_type):
    if card_type == 'RFID':
        # Sinh ra một số ngẫu nhiên từ 0-1
        # Nếu số này nhỏ hơn xác suất xuất hiện lỗi thẻ từ, trả về True và đây là trường hợp có phát sinh lỗi
        # Nếu số này lớn hơn xác suất xuất hiện lỗi thẻ từ, trả về False và đây là trường hợp không phát sinh lỗi
        return random.random() <= ERROR_RATE_RFID
    elif card_type == 'paper':
```

```
        # Tương tự cho thẻ giấy
        return random.random() <= ERROR_RATE_PAPER
```

Tham số đầu vào

- **card_type**: Đây là tham số đầu vào đại diện cho loại thẻ của phương tiện

Hoạt động

- Hàm này xác định xem có phát sinh lỗi khi quét thẻ hay không, dựa trên loại thẻ của phương tiện.
- Nếu loại thẻ là "RFID", hàm sẽ sinh ra một số ngẫu nhiên từ phân phối đều từ 0 đến 1. Nếu số này nhỏ hơn hoặc bằng tỷ lệ lỗi của thẻ RFID, hàm sẽ trả về True (lỗi). Hoạt động tương tự trên thẻ "giấy".

Đầu ra của hàm

- Hàm trả về giá trị TRUE hoặc FALSE

2. pick_shortest

```
def pick_shortest(lines):
    """
    Cho một danh sách các tài nguyên trong SimPy, xác định tài nguyên có hàng đợi ngắn nhất.

    Lưu ý rằng thứ tự hàng đợi được xáo trộn để không chọn hàng đợi đầu tiên quá nhiều.
    """
    # Tạo list chứa các hàng và index
    shuffled_lines = list(zip(range(len(lines)), lines))
    # Sắp xếp ngẫu nhiên vị trí các cổng trong list
    # Nhằm mô phỏng người tham gia chọn ngẫu nhiên trong số các cổng có cùng chiều dài ngắn nhất
    random.shuffle(shuffled_lines)

    # Tạm thời gán hàng chờ có độ dài ngắn nhất là hàng đầu tiên

    first_line = shuffled_lines[0]
    firt_line_length = first_line[0]
    idx_of_shortest = firt_line_length

    # Duyệt qua list các hàng đã được sắp xếp ngẫu nhiên
    for i, line in shuffled_lines:
        """
```

```
        Nếu chiều dài của hàng hiện tại ngắn hơn chiều dài của hàng đang giữ
vị trí ngắn nhất
        thì cập nhật index của hàng ngắn nhất là index của hàng hiện tại.
        Dùng .queue để xác định những đối tượng đang request hàng hiện tại,
khi mỗi hàng là một resource.
        Những đối tượng đang request resource được hiểu là những người đang
xếp hàng tại hàng hiện tại.
        Dùng len() để xác định chiều dài của hàng hiện tại
        """
        if len(line.queue) < len(lines[idx_of_shortest].queue):
            idx_of_shortest = i
    # Trả về hàng ngắn nhất trong số tất cả các hàng được chọn và một index
cho phần UI sử dụng
    return lines[idx_of_shortest], idx_of_shortest+1
```

Tham số đầu vào

- **lines :** danh sách các hàng đợi (các cổng trong tài nguyên SimPy).

Hoạt động

- Hàm này xác định hàng đợi có số lượng phần tử ít nhất, tức là hàng đợi ngắn nhất, từ danh sách các tài nguyên.
- Trước tiên, danh sách các hàng đợi được xáo trộn để ngẫu nhiên hóa việc chọn hàng đợi.
- Sau đó, hàm duyệt qua từng hàng đợi trong danh sách đã được xáo trộn.
- Với mỗi hàng đợi, hàm so sánh số lượng phần tử trong hàng đợi hiện tại với số lượng phần tử trong hàng đợi ngắn nhất đã xác định trước đó.Nếu số lượng phần tử trong hàng đợi hiện tại ít hơn, hàm cập nhật chỉ số của hàng đợi ngắn nhất là chỉ số của hàng đợi hiện tại.
- Cuối cùng, hàm trả về hàng đợi ngắn nhất từ danh sách các tài nguyên và một chỉ số cho phần giao diện người dùng sử dụng.

Đầu ra của hàm

- Một tuple gồm:
    i. Hàng đợi ngắn nhất trong số tất cả các hàng đợi được chọn.
    ii. Một chỉ số (index) cho phần giao diện người dùng sử dụng.

3. vehicle_arrival

```
def vehicle_arrival(env, rfid_gate_lines, paper_gate_lines):
    # Khởi tạo id cho xe đầu tiên
    next_person_id = 0
```

```
    # Bắt đầu vòng lặp
    while True:
        # Xác định loại thẻ xe này sẽ sử dụng
        card_type = random.choices(['RFID', 'paper'],
weights=[RFID_SELECTION_RATE, 1 - RFID_SELECTION_RATE])[0]

        # Nếu là sử dụng thẻ từ, các cổng sẽ là cổng từ
        if card_type == 'RFID':
            gate_lines = rfid_gate_lines

        # Nếu là sử dụng thẻ giấy, các cổng sẽ là cổng giấy
        else:
            gate_lines = paper_gate_lines

        # Tạo ra phương tiện với id, lựa chọn loại thẻ và các cổng họ có thể
đi như đã khai báo ở trên
        # Xác định thời gian cần chờ để phương tiện này xuất hiện và trạng
thái giao thông tương ứng
        next_arrival, traffic_status = generate_arrival_time(env)
        yield env.timeout(next_arrival) # Ghi nhận thời gian đã trôi qua trong
giả lập

        # Phương tiện này sau đó sẽ tham gia quá trình sử dụng dịch vụ kiểm vé
        env.process(using_gate(env, next_person_id, gate_lines, card_type,
traffic_status))

        # Tạo id cho phương tiện tiếp theo
        next_person_id += 1
```

Tham số đầu vào

- **env:** Môi trường giả lập SimPy, đại diện cho môi trường mô phỏng thời gian trong chương trình.
- **rfid_gate_lines:** Danh sách các hàng đợi của cổng RFID **paper_gate_lines:** Danh sách các hàng đợi của cổng giấy

Hoạt động

- Hàm này mô phỏng quá trình phương tiện đến cổng kiểm vé.
- Hàm sẽ lặp vô hạn, mô phỏng việc phương tiện liên tục đến cổng kiểm vé.

- Mỗi lần lặp, hàm sẽ xác định loại thẻ mà phương tiện sẽ sử dụng (RFID hoặc giấy) dựa trên một tỷ lệ xác định trước.
- Sau đó, hàm sẽ sinh ra một khoảng thời gian chờ đợi cho phương tiện này xuất hiện và xác định trạng thái giao thông tương ứng.
- Tiếp theo, hàm sẽ chờ cho đến khi đến lượt của phương tiện này sử dụng dịch vụ kiểm vé.
- Cuối cùng, hàm sẽ tạo một quá trình sử dụng dịch vụ kiểm vé cho phương tiện này và tăng biến đếm cho phương tiện tiếp theo.

4. using_gate

```
def using_gate(env, person_id, gate_lines, card_type, traffic_status):

    # Ghi nhận thời điểm xếp hàng
    queue_begin = env.now
    # Chọn cổng có ít người đang xếp hàng ở đó nhất
    gate_line = pick_shortest(gate_lines)[0]
    non_zero_gate_idx = pick_shortest(gate_lines)[1]

    # Thêm phương tiện vào cổng chờ trong giao diện
    if card_type == 'RFID':
        graphic_rfid_gates.add_to_line(non_zero_gate_idx)
    else:
        graphic_paper_gates.add_to_line(non_zero_gate_idx)

    # Tạo request được xếp hàng đến lượt "sử dụng tài nguyên (resource)"
    # Tài nguyên ở đây là cổng được chọn
    with gate_line.request() as req:
        yield req
        # Đến lượt sử dụng = đã xếp hàng xong
        # Vẫn còn nắm giữ "tài nguyên", tức là vẫn còn ở trong cổng này chưa
ra khỏi
        # Ghi nhận thời điểm xếp hàng xong
        queue_end = env.now



    # Bắt đầu "sử dụng tài nguyên (resource)"
    ### SCANNING
        # Sau khi xếp hàng xong, thẻ xe của phương tiện sẽ được quét
```

```
        # Ghi nhận thời điểm bắt đầu quét
        scan_begin = env.now
        # Sinh thời gian quét thẻ cho phương tiện này
        scan_time = generate_scan_time(card_type=card_type)
        # Ghi nhận thời gian trôi qua trong giả lập
        yield env.timeout(scan_time)
        # Ghi nhận thời điểm quét thẻ xong
        scan_end = env.now


    ### ERROR
        # Xác định có xảy ra lỗi hay không
        error_appearance = is_error(card_type)
        # Ghi nhận thời điểm bắt đầu kiểm lỗi
        correction_begin = env.now
        # Nếu có lỗi phát sinh
        if error_appearance:
            # Sinh thời gian sửa lỗi
            error_correction_time = generate_error_correction_time(card_type)
        # Nếu không có lỗi phát sinh
        else:
            # Thời gian kiểm lỗi bằng 0
            error_correction_time = 0

        # Ghi nhận thời gian trôi qua trong giả lập
        yield env.timeout(error_correction_time)
        # Ghi nhận thời điểm sửa lỗi xong
        correction_end = env.now

    # HOÀN THÀNH VÀ XÓA XE KHỎI HÀNG CHỜ TRÊN GIAO DIỆN
        if card_type == "RFID":
            graphic_rfid_gates.remove_from_line(non_zero_gate_idx)
        else:
            graphic_paper_gates.remove_from_line(non_zero_gate_idx)

    # LƯU CÁC SỰ KIỆN
        logging_events(person_id, card_type, non_zero_gate_idx,
traffic_status, queue_begin, queue_end, scan_begin, scan_end,
error_appearance, correction_begin, error_correction_time, correction_end)
```

Tham số đầu vào

- **env:** Môi trường giả lập SimPy
- **person_id:** ID của phương tiện, đại diện cho số thứ tự của phương tiện đến cổng kiểm vé.
- **gate_lines:** Danh sách các hàng đợi của cổng mà phương tiện có thể chọn để xếp hàng.
- **card_type:** Loại thẻ của phương tiện
- **traffic_status:** Trạng thái giao thông tại thời điểm phương tiện đến, có thể là "low", "avg" hoặc "high".

Hoạt động

- Ghi nhận thời điểm phương tiện bắt đầu xếp hàng tại cổng kiểm vé.
- Chọn hàng đợi có ít phương tiện xếp hàng nhất từ danh sách các hàng đợi.
- Chờ cho đến khi đến lượt sử dụng dịch vụ của phương tiện này.
- Khi đến lượt, phương tiện sử dụng dịch vụ kiểm vé tại cổng đã chọn.
- Ghi nhận thời gian sử dụng dịch vụ và các sự kiện liên quan.
- Nếu có lỗi xuất hiện khi quét thẻ:
    i. Mô phỏng quá trình sửa lỗi.
    ii. Ghi nhận thời gian cần thiết cho việc sửa lỗi.
- Ghi lại các sự kiện của quá trình sử dụng dịch vụ kiểm vé của phương tiện.

5. logging_events

```
def logging_events(person, card_type, gate_line, traffic_status, queue_begin, queue_end, scan_begin, scan_end, error_appearance, correction_begin, error_correction_time, correction_end):
    queue_duration = queue_end - queue_begin
    scan_duration = scan_end - scan_begin
    error_correcting_duration = error_correction_time
    wait = queue_duration + scan_duration + error_correcting_duration

    event_log.append({"event": "WAITING TO BE SCANNED", "person": f"id_{person}", "selected line": f"{card_type}_{gate_line}", "traffic status": traffic_status, "begin time": get_current_time(queue_begin), "end time": get_current_time(queue_end), "duration": round(queue_duration, 2)})
    event_log.append({"event": "SCAN TICKET", "person": f"id_{person}", "selected line": f"{card_type}_{gate_line}", "traffic status": traffic_status, "begin time": get_current_time(scan_begin), "end time": get_current_time(scan_end), "duration": round(scan_duration, 2)})
```

```
    if error_appearance:
        event_log.append({"event": "ERROR OCCURENCE AND CORRECTION", "person":
f"id_{person}", "selected line": f"{card_type}_{gate_line}", "traffic status":
traffic_status, "begin time": get_current_time(correction_begin), "end time":
get_current_time(correction_end), "duration": round(error_correcting_duration,
2)})


    wait_end_mark = scan_end if error_appearance==False else correction_end
    total_waits[int(wait_end_mark)].append(wait)
    if card_type == 'RFID':
            rfid_total_waits[int(wait_end_mark)].append(wait)
    else:
        paper_total_waits[int(wait_end_mark)].append(wait)
```

Tham số đầu vào

- **person**: Đây là tham số đầu vào đại diện cho id của sinh viên (một sinh viên điều khiển 1 xe nên cũng có thể hiểu là id cho mỗi xe)
- **card_type**: Đây là tham số đầu vào đại diện cho loại thẻ của phương tiện.
- **gate_line**: Đây là tham số đầu vào đại diện cho dòng cổng mà phương tiện đang sử dụng.
- **traffic_status**: Đây là tham số đầu vào đại diện cho tình trạng giao thông tại thời điểm đó.
- **queue_begin**: Thời điểm bắt đầu xếp hàng.
- **queue_end**: Thời điểm kết thúc xếp hàng.
- **scan_begin**: Thời điểm bắt đầu quét thẻ.
- **scan_end**: Thời điểm kết thúc quét thẻ.
- **error_appearance**: Biến boolean cho biết liệu có sự cố lỗi xảy ra hay không.
- **correction_begin**: Thời điểm bắt đầu xử lý sự cố lỗi.
- **error_correction_time**: Thời gian cần thiết để sửa lỗi.
- **correction_end**: Thời điểm kết thúc xử lý sự cố lỗi.

Hoạt động :

- Hàm này ghi nhận các sự kiện liên quan đến quá trình kiểm vé và xử lý sự cố lỗi của phương tiện.
- Ghi lại thời điểm bắt đầu và kết thúc của các giai đoạn quan trọng trong quá trình kiểm vé và xử lý lỗi.

- Khi phương tiện vào hàng đợi, ghi lại thời điểm bắt đầu xếp hàng và thời điểm kết thúc xếp hàng.
- Khi quét thẻ, ghi lại thời điểm bắt đầu và kết thúc của quá trình quét thẻ.
- Nếu xảy ra sự cố lỗi, ghi lại thời điểm bắt đầu và kết thúc của quá trình sửa lỗi, cùng với thời gian mất để xử lý sự cố.
- Mỗi sự kiện được ghi lại với các thông tin chi tiết như ID của phương tiện, loại thẻ, số thứ tự của hàng đợi, trạng thái giao thông, thời điểm bắt đầu và kết thúc của mỗi giai đoạn, cũng như thời gian mất để xử lý sự cố lỗi.

e. Chạy giả lập và ghi lại events

```
# TẠO MÔI TRƯỜNG
env = simpy.Environment()

# Tạo các tài nguyên của môi trường
# Mỗi tài nguyên chỉ được một xe sử dụng (scan, error) cùng lúc, còn những xe khác phải chờ
rfid_gate_lines = [simpy.Resource(env, capacity=1) for _ in range(RFID_GATE_LINES)]
paper_gate_lines = [simpy.Resource(env, capacity=1) for _ in range(PAPER_GATE_LINES)]
all_gate_lines = [rfid_gate_lines, paper_gate_lines]



env.process(vehicle_arrival(env, rfid_gate_lines, paper_gate_lines))
env.process(create_clock(env))

# Từ 6:30 đến 20:30 là 14 tiếng đồng hồ
hours = 14
# Đổi từ giờ sang giây
seconds = hours*60*60
env.run(until=seconds)
root.mainloop()




# Writing data to a JSON file
with open(r'output\GUI_events.json', 'w') as outfile:
```

```
    json.dump({"RFID GATES": RFID_GATE_LINES,
               "PAPER GATES": PAPER_GATE_LINES,
               "RFID EMPLOYEES": int(RFID_GATE_LINES * RFID_EMPS_PER_LINE),
               "PAPER EMPLOYEES": PAPER_GATE_LINES * PAPER_EMPS_PER_LINE,
               "events": event_log}, outfile, indent=4)
```

Thiết lập cấu trúc của cổng thẻ từ và thẻ giấy

- rfid_gate_lines , paper_gate_lines : thiết lập các thông tin từ các input có sẵn như số nhân viên trực mỗi cổng và số lượng cổng
- Quá trình mô phỏng sẽ được thực hiện trong khoảng thời gian từ 6:30 sáng đến 20:30 tối (tổng cộng 14 giờ)
- Kết quả của quá trình mô phỏng sẽ được ghi lại qua file cấu trúc JSON với các thông tin events như ("event", "person", "selected line", "traffic status", "begin time", "end time", "duration")

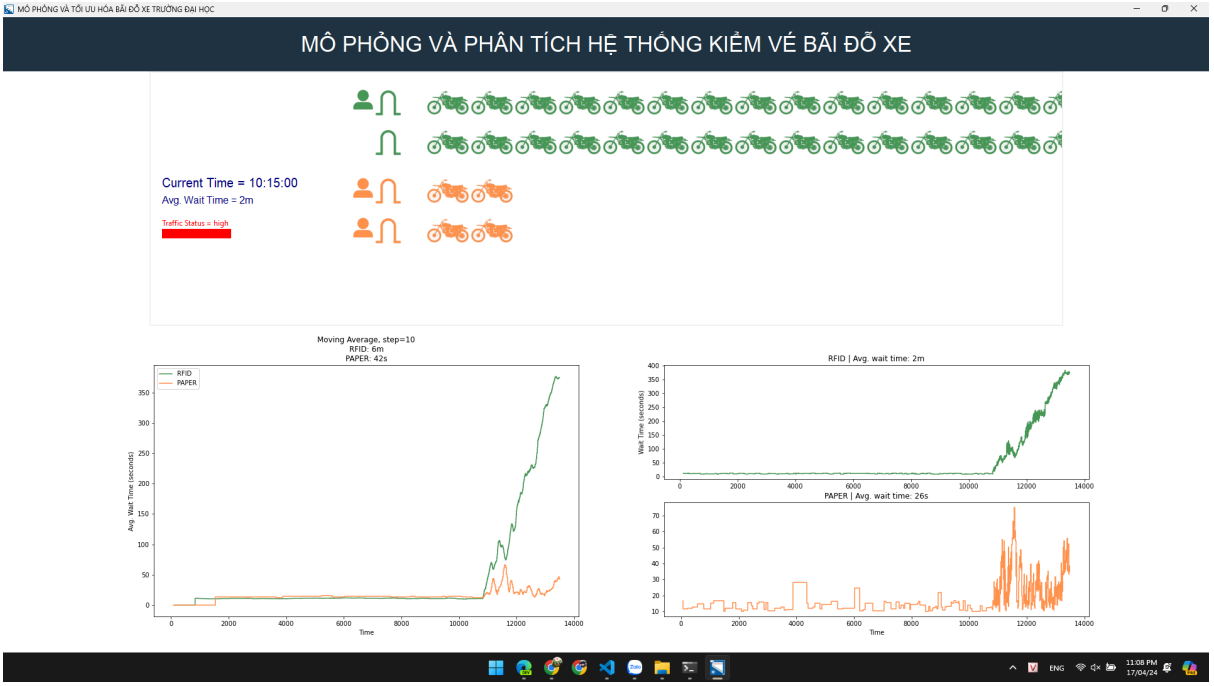
VII. Xây dựng giao diện đồ họa

Sau khi đã xây dựng xong chương trình mô phỏng các tình huống và sự kiện bằng SimPy, nhóm quyết định tạo một giao diện người dùng đồ họa (Graphical User Interface - GUI) để người dùng không có kiến thức chuyên sâu về lĩnh vực này cũng có thể dễ dàng sử dụng và hiểu rõ. Nhóm lựa chọn sử dụng thư viện Tkinter trong Python để tạo một giao diện trực quan.

Mục tiêu chính khi tạo ra giao diện mô phỏng này là:

1. Hiển thị trực quan quá trình xếp hàng và quét thẻ theo thời gian thực trong môi trường mô phỏng, bao gồm:
   - Hiển thị các cổng, nhân viên, các xe xếp hàng theo thời gian tùy theo mỗi loại cổng
   - Hiển thị thời gian tại môi trường mô phỏng
   - Hiển thị thời gian chờ đợi trung bình của sinh viên theo thời gian thực mô phỏng
   - Hiển thị trạng thái cao điểm và thấp điểm của khung giờ trong ngày
2. Hiển thị biểu đồ trực quan biểu diễn thời gian chờ đợi trung bình của từng loại cổng.

3. Hiển thị trung bình trượt của thời gian chờ đợi cổng cho cả thẻ từ và thẻ giấy, cung cấp một cái nhìn tổng quan về xu hướng chung của thời gian chờ đợi.



Diễn giải code

a. Tạo canvas

```
root = tk.Tk()
# size
root.geometry('1500x800+10+0')
# chiều ngang, chiều cao, khoảng cách với mép trái, khoảng cách với mép trên
# tilte
root.title('MÔ PHỎNG VÀ TỐI ƯU HÓA BÃI ĐỖ XE TRƯỜNG ĐẠI HỌC')
#icon
root.iconbitmap(r'images\due.ico')
#background
root.configure(bg='#fff')
#label
top_frame = tk.Frame(root)
top_frame.pack(side=tk.TOP, expand = False)
label = Label(root,text="MÔ PHỎNG VÀ PHÂN TÍCH HỆ THỐNG KIỂM VÉ BÃI ĐỖ XE",
font =('Arial',26),bg='#223442',fg='white', height=2)
label.pack(side=TOP, fill=X, expand=False)
#canvas
canvas_width = 1450
canvas_height = 400
```

```
canvas = tk.Canvas(root, width = canvas_width, height = canvas_height, bg =
"white")
canvas.pack(side=tk.TOP, expand = False)
```

Đầu tiên, ta tiến hành tạo cửa sổ giao diện, tiêu đề, thiết lập biểu đồ và cửa sổ cùng các biểu tượng cho cửa sổ.
Cụ thể các công việc như sau:

1. Tạo cửa sổ GUI:
2. Tạo frame và nhãn:
3. Tạo widget canvas:
4. Tạo đồ thị: a1,a2,a3
5. Nhúng đồ thị vào canvas:

b. Tạo đồ họa cho các thành phần trong canvas

Mục đích và ý nghĩa : Lớp QueueGraphics được thiết kế để hiển thị đồ họa của hàng chờ tại các cổng kiểm vé trong giao diện của ứng dụng mô phỏng hệ thống bãi đỗ xe.

Trong lớp QueueGraphics, các hàm chính bao gồm:

+ __init__: Hàm khởi tạo, thiết lập các thông số ban đầu và tạo các biến đồ họa cần thiết.
+ Các biến đầu vào của hàm :

vehicle_file	File chứa icon xe
vehicle_icon_width	Chiều rộng icon xe
gate_image_file	File chứa icon cổng
gate_line_list	List các cổng của từng loại
canvas	canvas
x_top	Tọa độ x trên cùng
y_top	Tọa độ y trên cùng
emp_icon_file	File chứa icon nhân viên
num_emps	Số nhân viên

```
class QueueGraphics:
    text_height = 50
    icon_top_margin = 5
```

```
    def __init__(self, vehicle_file, vehicle_icon_width, gate_image_file,
gate_line_list, canvas, x_top, y_top, emp_icon_file,num_emps):
        # File chứa icon xe
        self.vehicle_file = vehicle_file
        # Chiều rộng icon
        self.vehicle_icon_width = vehicle_icon_width
        # File chứa icon cổng
        self.gate_image_file = gate_image_file
        # List các cổng của từng loại
        self.gate_line_list = gate_line_list
        # Những graphics này sẽ được hiển thị trên canvas cho việc trình bày
giả lập các luồng xe
        self.canvas = canvas
        # Tọa độ x trên cùng
        self.x_top = x_top
        # Tọa độ y trên cùng
        self.y_top = y_top
        # File chứa icon nhân viên
        self.emp_icon_file = emp_icon_file
        # Số nhân viên
        self.num_emps = num_emps
        self.rectangles = []

        # Tải ảnh lên TK và lưu vào các biến tương ứng
        self.gate_image = tk.PhotoImage(file=self.gate_image_file)
        self.vehicle_image = tk.PhotoImage(file=self.vehicle_file)
        self.emp_icon_image = tk.PhotoImage(file=self.emp_icon_file)
        # Tạo dictionary lưu các icon phương tiện của từng cổng
        self.vehicle_icons = defaultdict(lambda: [])

        # Tạo các hình ảnh cổng
        for i in range(gate_line_list):
            canvas.create_image(x_top, y_top + (i * 1.25 * self.text_height),
anchor=tk.NW, image=self.gate_image)
        self.canvas.update()

        # Tạo các hình ảnh nhân viên
        for i in range(num_emps):
```

```
            canvas.create_image(x_top - 35, y_top + (i * 1.25 *
self.text_height), anchor=tk.NW, image=self.emp_icon_image)
        self.canvas.update()

    # Thêm icon xe máy vào hàng dựa trên chiều dài của hàng hiện tại
    def add_to_line(self, gate_line):
        count = len(self.vehicle_icons[gate_line])
        x = self.x_top + 100 + (count * self.vehicle_icon_width)
        y = self.y_top + ((gate_line - 1) * 1.25 * self.text_height) +
self.icon_top_margin
        self.vehicle_icons[gate_line].append(
                self.canvas.create_image(x, y, anchor=tk.NW,
image=self.vehicle_image)
        )
        self.canvas.update()

    # Bỏ icon xe máy khỏi hàng
    def remove_from_line(self, gate_line):
        if len(self.vehicle_icons[gate_line]) == 0: return
        to_del = self.vehicle_icons[gate_line].pop()
        self.canvas.delete(to_del)
        self.canvas.update()
```

+ add_to_line(self, line, position): Tính toán vị trí tọa độ x, y cho biểu tượng mới dựa trên số lượng biểu tượng hiện có và kích thước của biểu tượng. Thêm một biểu tượng vào hàng chờ ở vị trí được chỉ định.
+ remove_from_line(self, line): Xóa biểu tượng xe ra khỏi hàng chờ tương ứng nếu xe đó đã hoàn thành quy trình.

Khởi tạo các đối tượng dựa trên class đã định nghĩa trước đó:

```
# Tạo các cổng và số nhân viên tương ứng
def graphic_gates(canvas, x_top, y_top):
    rfid_emp_num = int(round(RFID_GATE_LINES * RFID_EMPS_PER_LINE, 0))
    paper_emp_num =  int(round(PAPER_GATE_LINES * PAPER_EMPS_PER_LINE, 0))

    # Position the RFID gate above the PAPER gate
    graphic_rfid_gates = QueueGraphics(r"images\xe xanh.png", 70,
r'images\cong xanh.png', RFID_GATE_LINES, canvas, x_top, y_top, r'images\nguoi
xanh.png',rfid_emp_num)
```

```
    graphic_paper_gates = QueueGraphics(r"images\xe cam.png", 70,
r'images\cong cam.png', PAPER_GATE_LINES, canvas, x_top, y_top *
RFID_GATE_LINES * 7.5, r'images\nguoi cam.png',paper_emp_num)


    return graphic_rfid_gates, graphic_paper_gates
```

c. Tạo bảng hiển thị thông tin

Mục đích và ý nghĩa: Lớp này được sử dụng để hiển thị thông tin về thời gian hiện tại, thời gian chờ trung bình và tình trạng giao thông trên canvas.

x1, y1	Tọa độ x, y của góc trái trên của hình chữ nhật đại diện cho khu vực của lớp ClockAndData.
x2, y2	Tọa độ x, y của góc phải dưới của hình chữ nhật đại diện cho khu vực của lớp ClockAndData.
time	Thời gian hiện tại.

```
# Tạo class cho bảng thông tin chứa đồng hồ đếm giờ và thống kê thời gian chờ
trung bình cho đến hiện tại trên UI
class ClockAndData:
    def __init__(self, canvas, x1, y1, x2, y2, time):
        # Khai báo tọa độ của bảng
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.canvas = canvas


        # Display thời gian hiện tại lên bảng này
        self.time = canvas.create_text(self.x1 + 10, self.y1, text = "Current
Time = "+ get_current_time(time), anchor = tk.NW, font=('Helvetica', 15),
fill='#000080')


        # Display thời gian chờ trung bình
        wait_time_string = avg_wait(total_waits)
        self.overall_avg_wait = canvas.create_text(self.x1 + 10, self.y1 + 35,
text = "Avg. Wait Time = "+ wait_time_string, anchor = tk.NW,
font=('Helvetica', 12), fill='#000080')


        # Display tình trạng giao thông và đổi màu tương sng
        traffic_status = check_traffic_status(get_current_time(time))
```

```
        traffic_status_color = {'high': 'red', 'avg': 'orange', 'low': 'blue'}
        self.traffic = canvas.create_text(self.x1 + 10, self.y1 + 68, text =
"Traffic Status = "+ traffic_status, anchor = tk.NW,
fill=traffic_status_color[traffic_status])
        self.traffic_rectangle = canvas.create_rectangle(self.x1 + 10, self.y1
+ 100, self.x2-70, self.y2+15, fill=traffic_status_color[traffic_status],
outline='')

        # CẬP NHẬT lại canvas
        self.canvas.update()

    # Qua mỗi nhịp thời gian sẽ xóa thông tin cũ và hiển thị thông tin mới
    def tick(self, time):
        # Xóa thông tin cũ
        self.canvas.delete(self.time)
        self.canvas.delete(self.overall_avg_wait)
        self.canvas.delete(self.traffic)

        # Display thời gian hiện tại lên bảng này
        self.time = canvas.create_text(self.x1 + 10, self.y1, text = "Current
Time = "+ get_current_time(time), anchor = tk.NW, font=('Helvetica', 15),
fill='#000080')

        # Display thời gian chờ trung bình
        wait_time_string = avg_wait(total_waits)
        self.overall_avg_wait = canvas.create_text(self.x1 + 10, self.y1 + 35,
text = "Avg. Wait Time = "+ wait_time_string, anchor = tk.NW,
font=('Helvetica', 12), fill='#000080')

        # Display tình trạng giao thông và đổi màu tương sng
        traffic_status = check_traffic_status(get_current_time(time))
        traffic_status_color = {'high': 'red', 'avg': 'orange', 'low': 'blue'}
        self.traffic = canvas.create_text(self.x1 + 10, self.y1 + 68, text =
"Traffic Status = "+ traffic_status, anchor = tk.NW,
fill=traffic_status_color[traffic_status])
        self.traffic_rectangle = canvas.create_rectangle(self.x1 + 10, self.y1
+ 100, self.x2-70, self.y2+15, fill=traffic_status_color[traffic_status],
outline='')

        # HIỂN THỊ CÁC PLOT
```

```
        a1.cla()
        a1.set_title(f"RFID | Avg. wait time: {avg_wait(rfid_total_waits)}")
        a1.set_ylabel("Wait Time (seconds)")
        # Step plot, cập nhật theo thời gian
        a1.step([ t for (t, waits) in rfid_total_waits.items() ], [
np.mean(waits) for (t, waits) in rfid_total_waits.items() ], color='#4A9658')



        a2.cla()
        a2.set_title(f"PAPER | Avg. wait time: {avg_wait(paper_total_waits)}")
        a2.set_xlabel("Time")
        # Step plot, cập nhật theo thời gian
        a2.step([ t for (t, waits) in paper_total_waits.items() ], [
np.mean(waits) for (t, waits) in paper_total_waits.items() ], color='#FF914C')

        # Tính trung bình trượt ở thời điểm hiện tại
        from collections import OrderedDict
        def moving_average(totals_dict, step):
            mean_waits_dict = {}
            moving_averages = {}
            for key, waits in totals_dict.items():
                if len(waits) == 0:
                    key_mean = 0
                else:
                    key_mean = np.mean(waits)
                mean_waits_dict[key] = key_mean
            for i, key in enumerate(mean_waits_dict.keys()):
                if i >= step - 1:
                    window = list(mean_waits_dict.values())[i - step + 1:i +
1]
                    moving_averages[key] = np.mean(window)
            for key in totals_dict.keys():
                if key not in moving_averages:
                    moving_averages[key] = 0
            moving_averages_sorted =
OrderedDict(sorted(moving_averages.items()))
            return moving_averages_sorted

        # Define số step cho trung bình trượt
```

```
        step = 10
        # Tính trung bình trượt cho từng loại cổng
        moving_average_rfid = moving_average(rfid_total_waits, step=step)
        moving_average_paper = moving_average(paper_total_waits, step=step)



        a3.cla()
        # Chuyển thành dạng list để dễ dàng truyền vào title của biểu đồ a3 và cập nhật qua thời gian
        current_moving_avg_rfid = list(moving_average_rfid.values())[-1] if moving_average_rfid else 0
        current_moving_avg_paper = list(moving_average_paper.values())[-1] if moving_average_paper else 0

        a3.set_title(f"Moving Average, step={step}\nRFID: {seconds_to_minutes_string(current_moving_avg_rfid)}\nPAPER: {seconds_to_minutes_string(current_moving_avg_paper)}")
        a3.set_xlabel("Time")
        a3.set_ylabel("Avg. Wait Time (seconds)")

        # Step plot, cập nhật theo thời gian
        # Plot chứa 2 đường, mỗi đường đại diện cho một cổng
        a3.step([ t for (t, moving_avg) in moving_average_rfid.items() ], [ moving_avg for (t, moving_avg) in moving_average_rfid.items() ], label='RFID', color='#4A9658')
        a3.step([ t for (t, moving_avg) in moving_average_paper.items() ], [ moving_avg for (t, moving_avg) in moving_average_paper.items() ], label='PAPER', color='#FF914C')

        a3.legend()



        data_plot.draw()
        self.canvas.update()
```

d. Điều chỉnh tốc độ của môi trường giả lập

Trước hết, hàm yêu cầu người dùng nhập số giây trong mô phỏng tương đương với một giây thực. Điều này sẽ quyết định tốc độ mà thời gian trong mô phỏng tiến triển so

với thời gian thực.

Ví dụ : 500 giây mô phỏng trong 1 giây thực tế

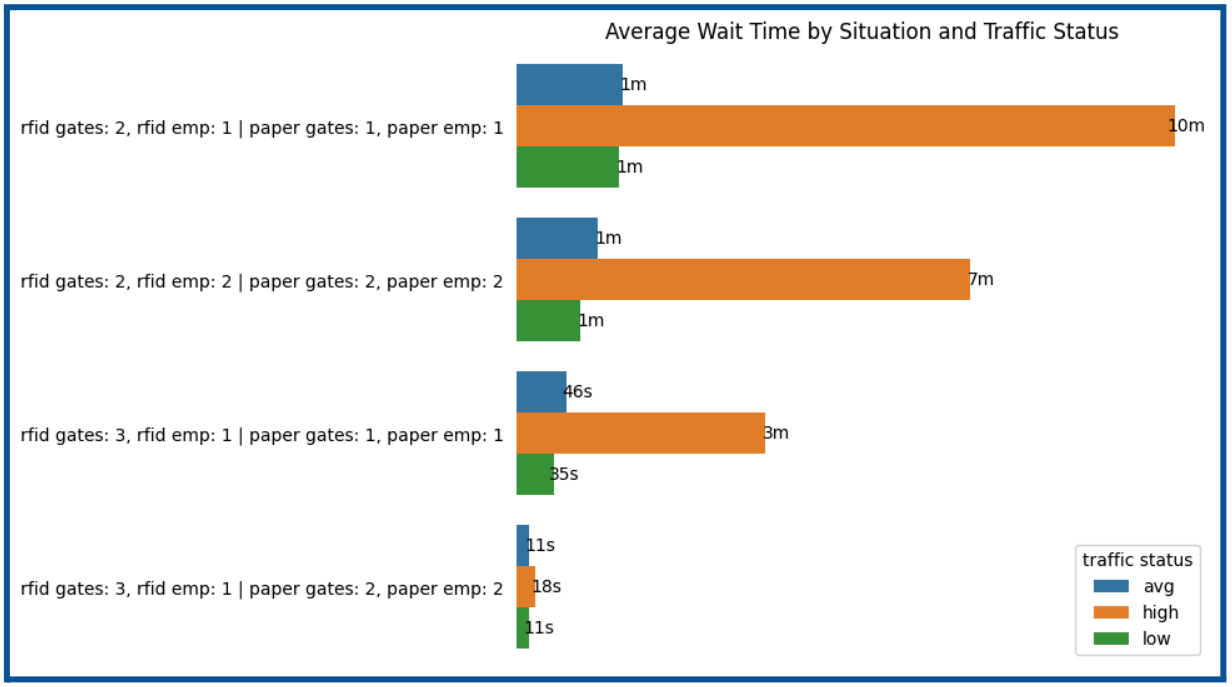
```
Seconds passed in simulation per real second: 500
```

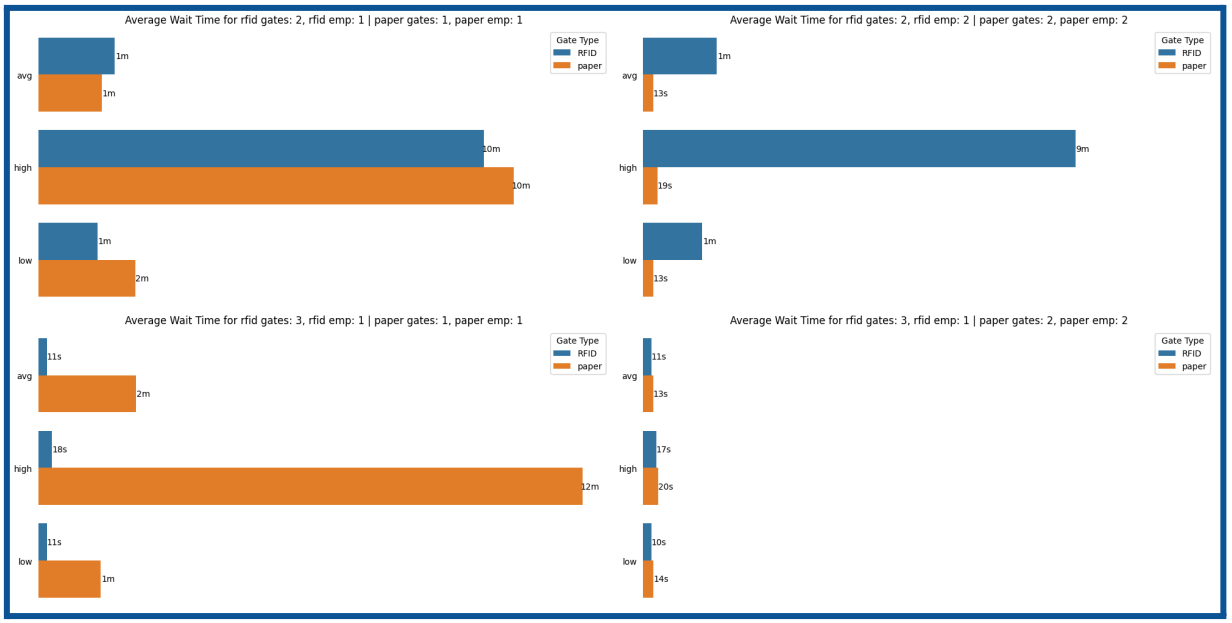
- Hàm sử dụng một vòng lặp vô hạn (while True) để liên tục sinh ra sự kiện cập nhật đồng hồ, trong mỗi lần lặp, hàm sẽ sinh ra một sự kiện với thời gian là số giây mô phỏng tương ứng với một giây thực.

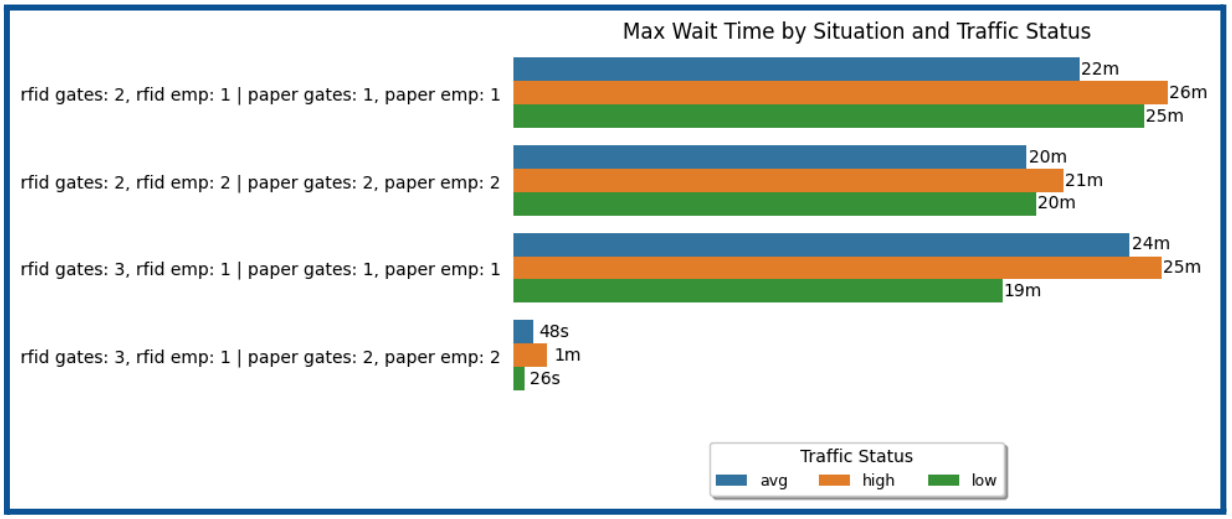
```
def create_clock(env):
    """
        Tạo đồng hồ để điều chỉnh tốc độ của giả lập khi hiển thị lên UI.
        Bằng cách điều chỉnh kích thước mỗi nhịp thời gian trong giả lập.
        Với mỗi giây trôi qua ngoài đời thật, thì bao nhiêu giây trôi qua
trong giả lập.
        secs_passed_in_sim_per_real_sec sẽ là một nhịp giả lập.
    """
    secs_passed_in_sim_per_real_sec = int(input('Seconds passed in simulation
per real second: '))


    while True:
        yield env.timeout(secs_passed_in_sim_per_real_sec)
        clock.tick(env.now)
```

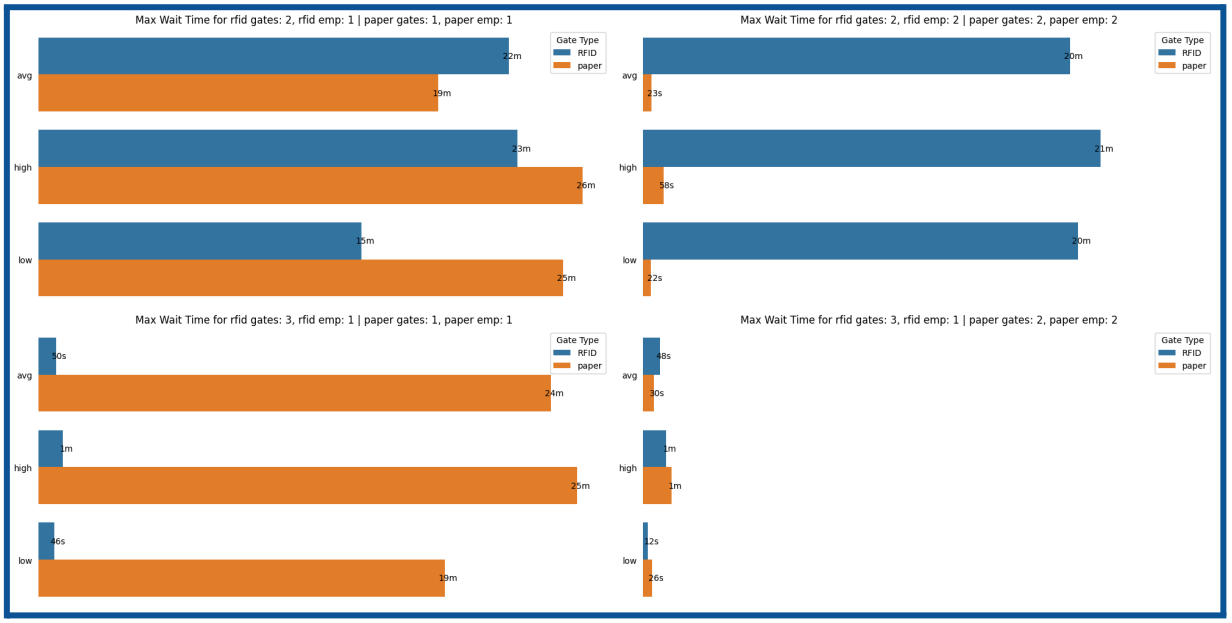
VIII. Kết luận các điểm chính

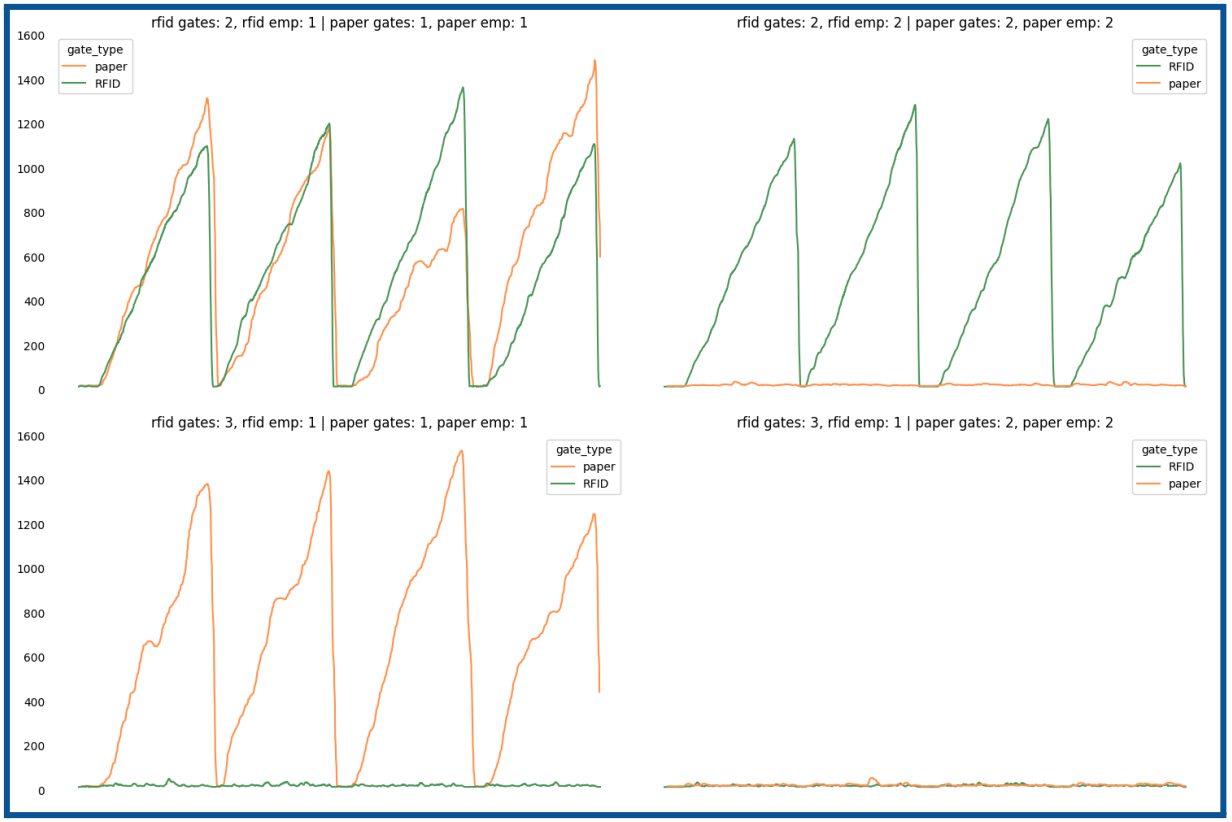
```
single_mean_wait()
situational_mean_wait()
single_max_wait()
situational_max_wait()
plot_moving_avg(sharey=True)
plot_moving_avg(sharey=False)
```

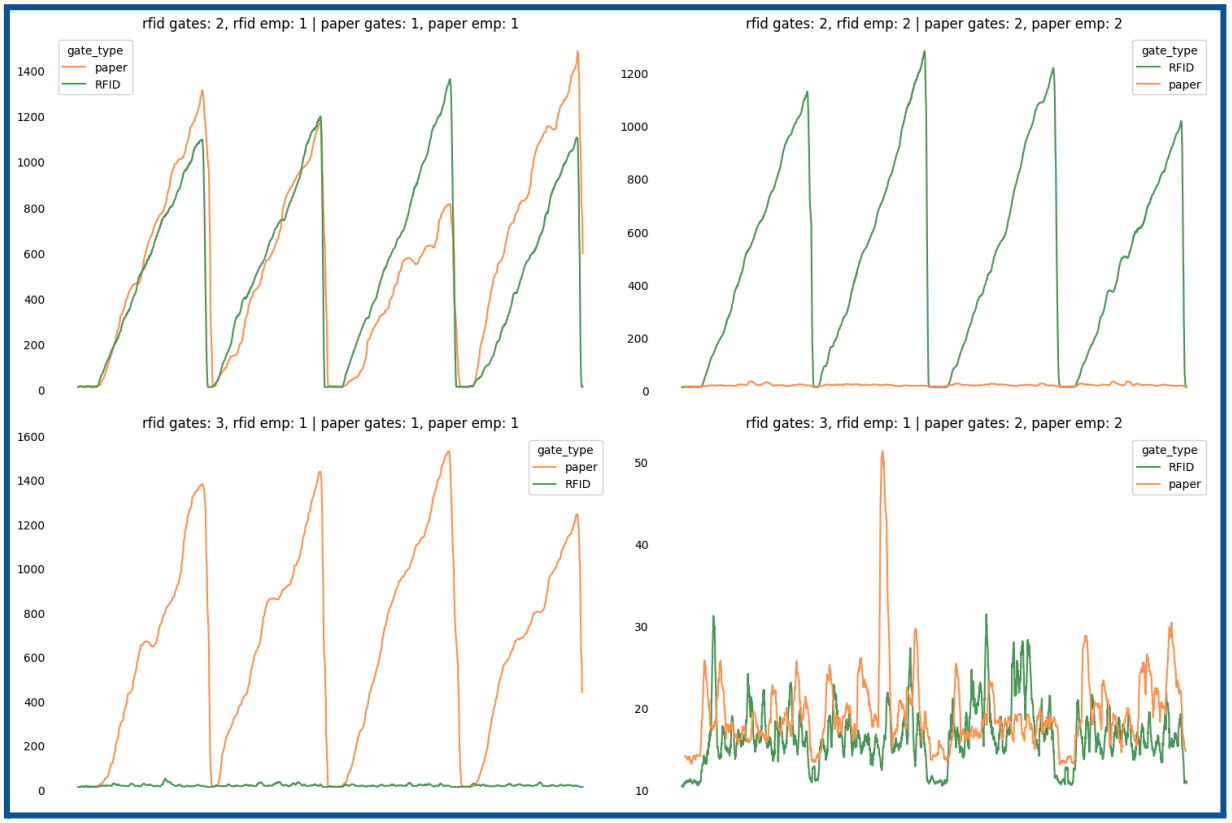

Average Wait Time by Situation and Traffic Status
rfid gates: 2, rfid emp: 1 | paper gates: 1, paper emp: 1
1m
10m
1m
rfid gates: 2, rfid emp: 2 | paper gates: 2, paper emp: 2
1m
7m
1m
rfid gates: 3, rfid emp: 1 | paper gates: 1, paper emp: 1
46s
3m
35s
rfid gates: 3, rfid emp: 1 | paper gates: 2, paper emp: 2
11s
18s
11s
traffic status
avg
high
low


Average Wait Time for rfid gates: 2, rfid emp: 1 | paper gates: 1, paper emp: 1
Average Wait Time for rfid gates: 2, rfid emp: 2 | paper gates: 2, paper emp: 2
Average Wait Time for rfid gates: 3, rfid emp: 1 | paper gates: 1, paper emp: 1
Average Wait Time for rfid gates: 3, rfid emp: 1 | paper gates: 2, paper emp: 2
Gate Type
RFID
paper
avg
high
low


Max Wait Time by Situation and Traffic Status
rfid gates: 2, rfid emp: 1 | paper gates: 1, paper emp: 1
22m
26m
25m
rfid gates: 2, rfid emp: 2 | paper gates: 2, paper emp: 2
20m
21m
20m
rfid gates: 3, rfid emp: 1 | paper gates: 1, paper emp: 1
24m
25m
19m
rfid gates: 3, rfid emp: 1 | paper gates: 2, paper emp: 2
48s
1m
26s
Traffic Status
avg
high
low


Max Wait Time for rfid gates: 2, rfid emp: 1 | paper gates: 1, paper emp: 1
Gate Type
RFID
paper
avg
22m
19m
high
23m
26m
low
15m
25m
Max Wait Time for rfid gates: 2, rfid emp: 2 | paper gates: 2, paper emp: 2
Gate Type
RFID
paper
avg
20m
23s
high
21m
58s
low
20m
22s
Max Wait Time for rfid gates: 3, rfid emp: 1 | paper gates: 1, paper emp: 1
Gate Type
RFID
paper
avg
30s
24m
high
1m
25m
low
46s
19m
Max Wait Time for rfid gates: 3, rfid emp: 1 | paper gates: 2, paper emp: 2
Gate Type
RFID
paper
avg
48s
30s
high
1m
1m
low
12s
26s


rfid gates: 2, rfid emp: 1 | paper gates: 1, paper emp: 1
gate_type
paper
RFID
1600
1400
1200
1000
800
600
400
200
0
rfid gates: 2, rfid emp: 2 | paper gates: 2, paper emp: 2
gate_type
RFID
paper
rfid gates: 3, rfid emp: 1 | paper gates: 1, paper emp: 1
gate_type
paper
RFID
1600
1400
1200
1000
800
600
400
200
0
rfid gates: 3, rfid emp: 1 | paper gates: 2, paper emp: 2
gate_type
RFID
paper



Các tình huống được mô tả là:
1. 2 cổng từ, 1 cổng giấy, 1 nhân viên cổng từ
2. 2 cổng từ, 2 cổng giấy, 2 nhân viên cổng từ (tình huống hiện tại)
3. 3 cổng từ, 1 cổng giấy, 1 nhân viên cổng từ
4. 3 cổng từ, 2 cổng giấy, 1 nhân viên cổng từ (phương án tối ưu)

Từ dữ liệu được cung cấp, rõ ràng có sự chênh lệch đáng kể về thời gian chờ bình quân giữa các tình huống, đặc biệt vào giờ cao điểm.

Trong tình huống 1 và 2 (tình trạng hiện tại), thời gian chờ bình quân ở giờ cao điểm lên tới 23-26 phút, tăng gấp khoảng 25 lần so với những khung giờ khác. Điều này cho thấy vấn đề tắc nghẽn tại các cổng vào giờ cao điểm là rất nghiêm trọng, gây bức xúc và bất tiện lớn cho sinh viên.

Nguyên nhân chính là do tỷ lệ sinh viên lựa chọn cổng từ (70%) cao hơn nhiều so với cổng giấy, trong khi số lượng cổng của mỗi loại thẻ lại bằng nhau (2 cổng). Do đó, cổng từ bị quá tải vào giờ cao điểm, dẫn đến tình trạng chờ đợi kéo dài.

Nếu tăng số lượng cổng từ lên 3 và giữ nguyên 2 cổng giấy (tình huống 4 - phương án tối ưu), thời gian chờ bình quân vào giờ cao điểm được cải thiện đáng kể, chỉ còn

khoảng 10 giây, thậm chí trong giai đoạn cao điểm cũng chỉ khoảng 1 phút. Điều này sẽ giúp giảm đáng kể sự bức xúc và bất tiện cho sinh viên.

Trường hợp nhà trường không thể xây thêm cổng mới do hạn chế tài chính, một giải pháp khác là **giảm tỷ lệ sinh viên sử dụng cổng giấy xuống còn 60%.** Điều này vừa giúp giảm thời gian chờ bình quân, vừa tiết kiệm chi phí so với phương án tăng số lượng cổng. Đây không phải là hướng đi nhóm đề ra ban đầu, tuy nhiên, cũng là một ý tưởng sáng đáng cân nhắc và phát triển. Nhóm đã kiểm thử kết quả của phương án này trong lúc triển khai.

Tóm lại, phương án tối ưu nhất, theo định hướng đề ra ban đầu là tăng số cổng từ lên 3 và giữ nguyên 2 cổng giấy. Điều này sẽ giúp giải quyết triệt để vấn đề tắc nghẽn tại các cổng vào giờ cao điểm, mang lại trải nghiệm tốt nhất cho sinh viên. Nếu không thể thực hiện, việc điều chỉnh tỷ lệ sử dụng cổng giấy cũng là một lựa chọn đáng cân nhắc.

Nhà trường cần xem xét các yếu tố như khả năng tài chính, lộ trình triển khai, cũng như khả năng kiểm soát tỷ lệ sử dụng các cổng để đưa ra quyết định phù hợp nhất. Mục tiêu là cải thiện trải nghiệm của sinh viên trong quá trình ra vào khuôn viên trường, đồng thời đảm bảo sử dụng hiệu quả các nguồn lực hiện có.

Tổng kết những kết quả thu được của dự án

Mô phỏng sự kiện

Nhóm đã tiến hành mô phỏng các tình huống khác nhau về lưu lượng ra vào tại các cổng của trường.
Các tình huống bao gồm:

+ 2 cổng từ, 1 cổng giấy, 1 nhân viên cổng từ
+ 2 cổng từ, 2 cổng giấy, 2 nhân viên cổng từ
+ 3 cổng từ, 1 cổng giấy, 1 nhân viên cổng từ
+ 3 cổng từ, 2 cổng giấy, 1 nhân viên cổng từ

Thiết kế giao diện animation

Nhóm đã thiết kế các giao diện animation sinh động, trực quan để minh họa kết quả mô phỏng. Các animation này giúp làm rõ tình trạng tắc nghẽn tại các cổng, đặc biệt vào giờ cao điểm.

Phân tích và đề xuất

- Dựa trên kết quả mô phỏng và animation, nhóm đã tiến hành phân tích sâu sắc tình hình hiện tại.
- Xác định số nhân viên tại mỗi cổng không làm ảnh hưởng đáng kể đến thời gian chờ trung bình.
- Xác định nguyên nhân chính là tỷ lệ sinh viên lựa chọn cổng từ cao hơn nhiều so với cổng giấy.
- Đề xuất phương án tối ưu là tăng số cổng từ lên 3 và giữ nguyên 2 cổng giấy. Với phương án này, có thể giảm số thời gian chờ trung bình của 2 loại cổng và qua các khung giờ xuống đáng kể, chỉ dao động trong khoảng 20 giây.
- Nếu không thể thực hiện phương án tối ưu, giải pháp thay thế là giảm tỷ lệ sử dụng cổng giấy xuống khoảng 60%.

Tóm lại, nhóm đã thực hiện các bước quan trọng trong dự án, từ mô phỏng sự kiện, thiết kế animation minh họa, cho đến phân tích tình hình và đề xuất giải pháp hiệu quả. Những kết quả này sẽ giúp nhà trường có cơ sở vững chắc để đưa ra quyết định phù hợp, nhằm cải thiện trải nghiệm của sinh viên khi ra vào khuôn viên.

IX. Toàn bộ code giả lập và giao diện

```
import itertools
import time
import simpy
from tkinter import *
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import tkinter as tk
import json
import math
import random
import numpy as np
import pandas as pd
from collections import defaultdict
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from PIL import ImageTk
import os


######################
##### INPUTS #########
```

```
######################
RFID_GATE_LINES = 2
PAPER_GATE_LINES = 2

PAPER_EMPS_PER_LINE = 1
RFID_EMPS_PER_LINE = 0.5

######################
### CONFIGURATIONS ###
######################
RFID_SELECTION_RATE = 0.6

RFID_SCAN_TIME_MIN = 9
RFID_SCAN_TIME_MAX = 13
PAPER_SCAN_TIME_MIN = 10
PAPER_SCAN_TIME_MAX = 17

JOIN_RATE_HIGH_MEAN = 3
JOIN_RATE_HIGH_STD = 0.5

JOIN_RATE_AVG_MEAN = 30
JOIN_RATE_AVG_STD = 1

JOIN_RATE_LOW_MEAN = 60
JOIN_RATE_LOW_STD = 10

ERROR_RATE_RFID = 0.01
ERROR_RATE_PAPER = 0.035
# Creating the 'images' directory if it doesn't exist
os.makedirs('images', exist_ok=True)

# Khởi tạo dictionary chứa list tổng thời gian chờ tại mỗi giây diễn ra trong giả lập của tất cả các loại cổng
total_waits = defaultdict(list)
# Khởi tạo dictionary chứa list tổng thời gian chờ tại mỗi giây diễn ra trong giả lập của cổng thẻ từ
rfid_total_waits = defaultdict(list)
# Khởi tạo dictionary chứa list tổng thời gian chờ tại mỗi giây diễn ra trong giả lập của cổng thẻ giấy
```

```
paper_total_waits = defaultdict(list)

# Hàm xác định thời điểm (giờ:phút:giây) hiện tại dựa trên số giây đã trôi qua trong giả lập
def get_current_time(elapsed_seconds):
    start_time = datetime.strptime('06:30:00', '%H:%M:%S')
    current_time = start_time + timedelta(seconds=elapsed_seconds)
    formatted_current_time = current_time.strftime('%H:%M:%S')
    return formatted_current_time

# Xác định trạng thái giao thông dựa trên thời điểm hiện tại
def check_traffic_status(current_time):
    if current_time < "08:30:00":
        return "low"
    elif "08:30:00" <= current_time < "09:30:00":
        return "avg"
    elif "09:30:00" <= current_time < "10:30:00":
        return "high"
    elif "10:30:00" <= current_time < "11:30:00":
        return "avg"
    elif "11:30:00" <= current_time < "12:30:00":
        return "high"
    elif "12:30:00" <= current_time < "14:30:00":
        return "low"
    elif "14:30:00" <= current_time < "15:30:00":
        return "avg"
    elif "15:30:00" <= current_time < "16:30:00":
        return "high"
    elif "16:30:00" <= current_time < "18:30:00":
        return "avg"
    elif "18:30:00" <= current_time < "19:30:00":
        return "high"
    else:
        return "low"

# Sinh thời gian quét thẻ dựa trên mỗi loại cổng
def generate_scan_time(card_type):
    if card_type == 'RFID':
```

```
        return random.uniform(RFID_SCAN_TIME_MIN, RFID_SCAN_TIME_MAX)
    elif card_type == 'paper':
        return random.uniform(PAPER_SCAN_TIME_MIN, PAPER_SCAN_TIME_MAX)

# Sinh thời gian xe đến dựa vào tình trạng giao thông hiện tại
def generate_arrival_time(env):
    current_time = get_current_time(env.now)
    traffic_status = check_traffic_status(current_time)
    if traffic_status == 'high':
        arrival_time = max(0, random.normalvariate(JOIN_RATE_HIGH_MEAN,
JOIN_RATE_HIGH_STD))
    elif traffic_status == 'low':
        arrival_time = max(0, random.normalvariate(JOIN_RATE_LOW_MEAN,
JOIN_RATE_LOW_STD))
    else: # NORMAL
        arrival_time = max(0, random.normalvariate(JOIN_RATE_AVG_MEAN,
JOIN_RATE_AVG_STD))
    return arrival_time, traffic_status

# Sinh thời gian cần thiết để sửa lỗi nếu phát sinh dựa trên loại thẻ
def generate_error_correction_time(card_type):
    if card_type == 'RFID':
        if RFID_EMPS_PER_LINE == 1:
            ERROR_CORRECTION_TIME = max(0, random.normalvariate(15, 5))
        elif 0.5 < RFID_EMPS_PER_LINE < 1:
            ERROR_CORRECTION_TIME = max(0, random.normalvariate(20, 5))
        else:
            ERROR_CORRECTION_TIME = max(0, random.normalvariate(30, 5))
    elif card_type == 'paper':
        ERROR_CORRECTION_TIME = max(0, random.normalvariate(10, 2))

    return ERROR_CORRECTION_TIME

# Tạo list rỗng để chứa các ghi nhận event
event_log = []

# Hàm sinh trường hợp lỗi của mỗi loại thẻ dựa trên xác suất
def is_error(card_type):
    if card_type == 'RFID':
```

```
        # Sinh ra một số ngẫu nhiên từ 0-1
        # Nếu số này nhỏ hơn xác suất xuất hiện lỗi thẻ từ, trả về True và đây là trường hợp có phát sinh lỗi
        # Nếu số này lớn hơn xác suất xuất hiện lỗi thẻ từ, trả về False và đây là trường hợp không phát sinh lỗi
        return random.random() <= ERROR_RATE_RFID
    elif card_type == 'paper':
        # Tương tự cho thẻ giấy
        return random.random() <= ERROR_RATE_PAPER

# Hàm chọn hàng có ít người đang xếp hàng nhất
# Input là các hàng có thể chọn dựa trên loại thẻ
def pick_shortest(lines):
    """
    Cho một danh sách các tài nguyên trong SimPy, xác định tài nguyên có hàng đợi ngắn nhất.

    Lưu ý rằng thứ tự hàng đợi được xáo trộn để không chọn hàng đợi đầu tiên quá nhiều.
    """
    # Tạo list chứa các hàng và index
    shuffled_lines = list(zip(range(len(lines)), lines))
    # Sắp xếp ngẫu nhiên vị trí các cổng trong list
    # Nhằm mô phỏng người tham gia chọn ngẫu nhiên trong số các cổng có cùng chiều dài ngắn nhất
    random.shuffle(shuffled_lines)

    # Tạm thời gán hàng chờ có độ dài ngắn nhất là hàng đầu tiên

    first_line = shuffled_lines[0]
    firt_line_length = first_line[0]
    idx_of_shortest = firt_line_length

    # Duyệt qua list các hàng đã được sắp xếp ngẫu nhiên
    for i, line in shuffled_lines:
        """
        Nếu chiều dài của hàng hiện tại ngắn hơn chiều dài của hàng đang giữ vị trí ngắn nhất
        thì cập nhật index của hàng ngắn nhất là index của hàng hiện tại.
```

```
        Dùng .queue để xác định những đối tượng đang request hàng hiện tại,
khi mỗi hàng là một resource.
        Những đối tượng đang request resource được hiểu là những người đang
xếp hàng tại hàng hiện tại.
        Dùng len() để xác định chiều dài của hàng hiện tại
        """
        if len(line.queue) < len(lines[idx_of_shortest].queue):
            idx_of_shortest = i
    # Trả về hàng ngắn nhất trong số tất cả các hàng được chọn và một index
cho phần UI sử dụng
    return lines[idx_of_shortest], idx_of_shortest+1



# Hàm ghi nhận lại các events
def logging_events(person, card_type, gate_line, traffic_status, queue_begin,
queue_end, scan_begin, scan_end, error_appearance, correction_begin,
error_correction_time, correction_end):
    queue_duration = queue_end - queue_begin
    scan_duration = scan_end - scan_begin
    error_correcting_duration = error_correction_time
    wait = queue_duration + scan_duration + error_correcting_duration

    event_log.append({"event": "WAITING TO BE SCANNED", "person":
f"id_{person}", "selected line": f"{card_type}_{gate_line}", "traffic status":
traffic_status, "begin time": get_current_time(queue_begin), "end time":
get_current_time(queue_end), "duration": round(queue_duration, 2)})
    event_log.append({"event": "SCAN TICKET", "person": f"id_{person}",
"selected line": f"{card_type}_{gate_line}", "traffic status": traffic_status,
"begin time": get_current_time(scan_begin), "end time":
get_current_time(scan_end), "duration": round(scan_duration, 2)})
    if error_appearance:
        event_log.append({"event": "ERROR OCCURENCE AND CORRECTION", "person":
f"id_{person}", "selected line": f"{card_type}_{gate_line}", "traffic status":
traffic_status, "begin time": get_current_time(correction_begin), "end time":
get_current_time(correction_end), "duration": round(error_correcting_duration,
2)})

    wait_end_mark = scan_end if error_appearance==False else correction_end
    total_waits[int(wait_end_mark)].append(wait)
    if card_type == 'RFID':
            rfid_total_waits[int(wait_end_mark)].append(wait)
```

```
    else:
        paper_total_waits[int(wait_end_mark)].append(wait)

# QUÁ TRÌNH TẠO PHƯƠNG TIỆN ĐẾN
def vehicle_arrival(env, rfid_gate_lines, paper_gate_lines):
    # Khởi tạo id cho xe đầu tiên
    next_person_id = 0

    # Bắt đầu vòng lặp
    while True:
        # Xác định loại thẻ xe này sẽ sử dụng
        card_type = random.choices(['RFID', 'paper'],
weights=[RFID_SELECTION_RATE, 1 - RFID_SELECTION_RATE])[0]

        # Nếu là sử dụng thẻ từ, các cổng sẽ là cổng từ
        if card_type == 'RFID':
            gate_lines = rfid_gate_lines

        # Nếu là sử dụng thẻ giấy, các cổng sẽ là cổng giấy
        else:
            gate_lines = paper_gate_lines

        # Tạo ra phương tiện với id, lựa chọn loại thẻ và các cổng họ có thể
đi như đã khai báo ở trên
        # Xác định thời gian cần chờ để phương tiện này xuất hiện và trạng
thái giao thông tương ứng
        next_arrival, traffic_status = generate_arrival_time(env)
        yield env.timeout(next_arrival) # Ghi nhận thời gian đã trôi qua trong
giả lập

        # Phương tiện này sau đó sẽ tham gia quá trình sử dụng dịch vụ kiểm vé
        env.process(using_gate(env, next_person_id, gate_lines, card_type,
traffic_status))

        # Tạo id cho phương tiện tiếp theo
        next_person_id += 1

# QUÁ TRÌNH SỬ DỤNG DỊCH VỤ KIỂM VÉ
def using_gate(env, person_id, gate_lines, card_type, traffic_status):
```

```
    # Ghi nhận thời điểm xếp hàng
    queue_begin = env.now
    # Chọn cổng có ít người đang xếp hàng ở đó nhất
    gate_line = pick_shortest(gate_lines)[0]
    non_zero_gate_idx = pick_shortest(gate_lines)[1]

    # Thêm phương tiện vào cổng chờ trong giao diện
    if card_type == 'RFID':
        graphic_rfid_gates.add_to_line(non_zero_gate_idx)
    else:
        graphic_paper_gates.add_to_line(non_zero_gate_idx)

    # Tạo request được xếp hàng đến lượt "sử dụng tài nguyên (resource)"
    # Tài nguyên ở đây là cổng được chọn
    with gate_line.request() as req:
        yield req
        # Đến lượt sử dụng = đã xếp hàng xong
        # Vẫn còn nắm giữ "tài nguyên", tức là vẫn còn ở trong cổng này chưa
ra khỏi
        # Ghi nhận thời điểm xếp hàng xong
        queue_end = env.now


    # Bắt đầu "sử dụng tài nguyên (resource)"
    ### SCANNING
        # Sau khi xếp hàng xong, thẻ xe của phương tiện sẽ được quét
        # Ghi nhận thời điểm bắt đầu quét
        scan_begin = env.now
        # Sinh thời gian quét thẻ cho phương tiện này
        scan_time = generate_scan_time(card_type=card_type)
        # Ghi nhận thời gian trôi qua trong giả lập
        yield env.timeout(scan_time)
        # Ghi nhận thời điểm quét thẻ xong
        scan_end = env.now


    ### ERROR
```

```
        # Xác định có xảy ra lỗi hay không
        error_appearance = is_error(card_type)
        # Ghi nhận thời điểm bắt đầu kiểm lỗi
        correction_begin = env.now
        # Nếu có lỗi phát sinh
        if error_appearance:
            # Sinh thời gian sửa lỗi
            error_correction_time = generate_error_correction_time(card_type)
        # Nếu không có lỗi phát sinh
        else:
            # Thời gian kiểm lỗi bằng 0
            error_correction_time = 0

        # Ghi nhận thời gian trôi qua trong giả lập
        yield env.timeout(error_correction_time)
        # Ghi nhận thời điểm sửa lỗi xong
        correction_end = env.now

    # HOÀN THÀNH VÀ XÓA XE KHỎI HÀNG CHỜ TRÊN GIAO DIỆN
        if card_type == "RFID":
            graphic_rfid_gates.remove_from_line(non_zero_gate_idx)
        else:
            graphic_paper_gates.remove_from_line(non_zero_gate_idx)

    # LƯU CÁC SỰ KIỆN
        logging_events(person_id, card_type, non_zero_gate_idx,
traffic_status, queue_begin, queue_end, scan_begin, scan_end,
error_appearance, correction_begin, error_correction_time, correction_end)

# Chuyển đổi giây thành phút với hậu tố là s nếu là giây và m nếu là phút
# Nếu dưới 60s thì không chuyển thành phút và giữ nguyên hậu tố s
# Nếu trên 60s thì chuyển thành phút và chuyển đổi thành hậu tố m
def seconds_to_minutes_string(seconds):
    if math.isnan(seconds):
        return 0
    if seconds >= 60:
        minutes = int(round(seconds / 60, 0))
        time_str = str(minutes) + 'm'
    else:
```

```
        seconds = int(round(seconds, 0))
        time_str = str(seconds) + 's'
    return time_str

# Tính toán thời gian chờ trung bình của các phương tiện tại mỗi thời điểm trong giả lập
# Mỗi thời điểm có một list thời gian chờ, do mỗi thời điểm có thể phát sinh nhiều phương tiện chờ
def avg_wait(raw_waits):
    waits = [w for i in raw_waits.values() for w in i]
    avg_wait_time = round(np.mean(waits), 1) if waits else 0
    wait_time_string = seconds_to_minutes_string(avg_wait_time)
    return wait_time_string

root = tk.Tk()
# size
root.geometry('1500x800+10+0')
# chiều ngang, chiều cao, khoảng cách với mép trái, khoảng cách với mép trên
# tilte
root.title('MÔ PHỎNG VÀ TỐI ƯU HÓA BÃI ĐỖ XE TRƯỜNG ĐẠI HỌC')
#icon
root.iconbitmap(r'images\due.ico')
#background
root.configure(bg='#fff')
#label
top_frame = tk.Frame(root)
top_frame.pack(side=tk.TOP, expand = False)
label = Label(root,text="MÔ PHỎNG VÀ PHÂN TÍCH HỆ THỐNG KIỂM VÉ BÃI ĐỖ XE", font =('Arial',26),bg='#223442',fg='white', height=2)
label.pack(side=TOP, fill=X, expand=False)
#canvas
canvas_width = 1450
canvas_height = 400
canvas = tk.Canvas(root, width = canvas_width, height = canvas_height, bg = "white")
canvas.pack(side=tk.TOP, expand = False)

#plot
f = plt.Figure(figsize=(2, 2), dpi=72)
```

```
# f.subplots_adjust(hspace=0.5, wspace=0.5)
a1 = f.add_subplot(222)
a1.plot()
a2 = f.add_subplot(224)
a2.plot()
a3 = f.add_subplot(121)
a3.plot()



data_plot = FigureCanvasTkAgg(f, master=root)
data_plot.get_tk_widget().config(height = 400)
data_plot.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

# Tạo class cho các cổng chờ
class QueueGraphics:
    text_height = 50
    icon_top_margin = 5

    def __init__(self, vehicle_file, vehicle_icon_width, gate_image_file,
gate_line_list, canvas, x_top, y_top, emp_icon_file,num_emps):
        # File chứa icon xe
        self.vehicle_file = vehicle_file
        # Chiều rộng icon xe
        self.vehicle_icon_width = vehicle_icon_width
        # File chứa icon cổng
        self.gate_image_file = gate_image_file
        # List các cổng của từng loại
        self.gate_line_list = gate_line_list
        # Những graphics này sẽ được hiển thị trên canvas cho việc trình bày
giả lập các luồng xe
        self.canvas = canvas
        # Tọa độ x trên cùng
        self.x_top = x_top
        # Tọa độ y trên cùng
        self.y_top = y_top
        # File chứa icon nhân viên
        self.emp_icon_file = emp_icon_file
        # Số nhân viên
        self.num_emps = num_emps
```

```
        self.rectangles = []

        # Tải ảnh lên TK và lưu vào các biến tương ứng
        self.gate_image = tk.PhotoImage(file=self.gate_image_file)
        self.vehicle_image = tk.PhotoImage(file=self.vehicle_file)
        self.emp_icon_image = tk.PhotoImage(file=self.emp_icon_file)
        # Tạo dictionary lưu các icon phương tiện của từng cổng
        self.vehicle_icons = defaultdict(lambda: [])

        # Tạo các hình ảnh cổng
        for i in range(gate_line_list):
            canvas.create_image(x_top, y_top + (i * 1.25 * self.text_height),
anchor=tk.NW, image=self.gate_image)
        self.canvas.update()

        # Tạo các hình ảnh nhân viên
        for i in range(num_emps):
            canvas.create_image(x_top - 35, y_top + (i * 1.25 *
self.text_height), anchor=tk.NW, image=self.emp_icon_image)
        self.canvas.update()

    # Thêm icon xe máy vào hàng dựa trên chiều dài của hàng hiện tại
    def add_to_line(self, gate_line):
        count = len(self.vehicle_icons[gate_line])
        x = self.x_top + 100 + (count * self.vehicle_icon_width)
        y = self.y_top + ((gate_line - 1) * 1.25 * self.text_height) +
self.icon_top_margin
        self.vehicle_icons[gate_line].append(
                self.canvas.create_image(x, y, anchor=tk.NW,
image=self.vehicle_image)
        )
        self.canvas.update()

    # Bỏ icon xe máy khỏi hàng
    def remove_from_line(self, gate_line):
        if len(self.vehicle_icons[gate_line]) == 0: return
        to_del = self.vehicle_icons[gate_line].pop()
        self.canvas.delete(to_del)
        self.canvas.update()
```

```
# Tạo các cổng và số nhân viên tương ứng
def graphic_gates(canvas, x_top, y_top):
    rfid_emp_num = int(round(RFID_GATE_LINES * RFID_EMPS_PER_LINE, 0))
    paper_emp_num =  int(round(PAPER_GATE_LINES * PAPER_EMPS_PER_LINE, 0))

    # Position the RFID gate above the PAPER gate
    graphic_rfid_gates = QueueGraphics(r"images\xe xanh.png", 70,
r'images\cong xanh.png', RFID_GATE_LINES, canvas, x_top, y_top, r'images\nguoi
xanh.png',rfid_emp_num)
    graphic_paper_gates = QueueGraphics(r"images\xe cam.png", 70,
r'images\cong cam.png', PAPER_GATE_LINES, canvas, x_top, y_top *
RFID_GATE_LINES * 7.5, r'images\nguoi cam.png',paper_emp_num)

    return graphic_rfid_gates, graphic_paper_gates



# Tạo class cho bảng thông tin chứa đồng hồ đếm giờ và thống kê thời gian chờ
trung bình cho đến hiện tại trên UI
class ClockAndData:
    def __init__(self, canvas, x1, y1, x2, y2, time):
        # Khai báo tọa độ của bảng
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.canvas = canvas

        # Display thời gian hiện tại lên bảng này
        self.time = canvas.create_text(self.x1 + 10, self.y1, text = "Current
Time = "+ get_current_time(time), anchor = tk.NW, font=('Helvetica', 15),
fill='#000080')

        # Display thời gian chờ trung bình
        wait_time_string = avg_wait(total_waits)
        self.overall_avg_wait = canvas.create_text(self.x1 + 10, self.y1 + 35,
text = "Avg. Wait Time = "+ wait_time_string, anchor = tk.NW,
font=('Helvetica', 12), fill='#000080')

        # Display tình trạng giao thông và đổi màu tương sng
```

```
        traffic_status = check_traffic_status(get_current_time(time))
        traffic_status_color = {'high': 'red', 'avg': 'orange', 'low': 'blue'}
        self.traffic = canvas.create_text(self.x1 + 10, self.y1 + 68, text =
"Traffic Status = "+ traffic_status, anchor = tk.NW,
fill=traffic_status_color[traffic_status])
        self.traffic_rectangle = canvas.create_rectangle(self.x1 + 10, self.y1
+ 100, self.x2-70, self.y2+15, fill=traffic_status_color[traffic_status],
outline='')

        # CẬP NHẬT lại canvas
        self.canvas.update()

    # Qua mỗi nhịp thời gian sẽ xóa thông tin cũ và hiển thị thông tin mới
    def tick(self, time):
        # Xóa thông tin cũ
        self.canvas.delete(self.time)
        self.canvas.delete(self.overall_avg_wait)
        self.canvas.delete(self.traffic)

        # Display thời gian hiện tại lên bảng này
        self.time = canvas.create_text(self.x1 + 10, self.y1, text = "Current
Time = "+ get_current_time(time), anchor = tk.NW, font=('Helvetica', 15),
fill='#000080')

        # Display thời gian chờ trung bình
        wait_time_string = avg_wait(total_waits)
        self.overall_avg_wait = canvas.create_text(self.x1 + 10, self.y1 + 35,
text = "Avg. Wait Time = "+ wait_time_string, anchor = tk.NW,
font=('Helvetica', 12), fill='#000080')

        # Display tình trạng giao thông và đổi màu tương sng
        traffic_status = check_traffic_status(get_current_time(time))
        traffic_status_color = {'high': 'red', 'avg': 'orange', 'low': 'blue'}
        self.traffic = canvas.create_text(self.x1 + 10, self.y1 + 68, text =
"Traffic Status = "+ traffic_status, anchor = tk.NW,
fill=traffic_status_color[traffic_status])
        self.traffic_rectangle = canvas.create_rectangle(self.x1 + 10, self.y1
+ 100, self.x2-70, self.y2+15, fill=traffic_status_color[traffic_status],
outline='')
```

```
        # HIỂN THỊ CÁC PLOT
        a1.cla()
        a1.set_title(f"RFID | Avg. wait time: {avg_wait(rfid_total_waits)}")
        a1.set_ylabel("Wait Time (seconds)")
        # Step plot, cập nhật theo thời gian
        a1.step([ t for (t, waits) in rfid_total_waits.items() ], [
np.mean(waits) for (t, waits) in rfid_total_waits.items() ], color='#4A9658')



        a2.cla()
        a2.set_title(f"PAPER | Avg. wait time: {avg_wait(paper_total_waits)}")
        a2.set_xlabel("Time")
        # Step plot, cập nhật theo thời gian
        a2.step([ t for (t, waits) in paper_total_waits.items() ], [
np.mean(waits) for (t, waits) in paper_total_waits.items() ], color='#FF914C')

        # Tính trung bình trượt ở thời điểm hiện tại
        from collections import OrderedDict
        def moving_average(totals_dict, step):
            mean_waits_dict = {}
            moving_averages = {}
            for key, waits in totals_dict.items():
                if len(waits) == 0:
                    key_mean = 0
                else:
                    key_mean = np.mean(waits)
                mean_waits_dict[key] = key_mean
            for i, key in enumerate(mean_waits_dict.keys()):
                if i >= step - 1:
                    window = list(mean_waits_dict.values())[i - step + 1:i +
1]
                    moving_averages[key] = np.mean(window)
            for key in totals_dict.keys():
                if key not in moving_averages:
                    moving_averages[key] = 0
            moving_averages_sorted =
OrderedDict(sorted(moving_averages.items()))
            return moving_averages_sorted
```

```
        # Define số step cho trung bình trượt
        step = 10
        # Tính trung bình trượt cho từng loại cổng
        moving_average_rfid = moving_average(rfid_total_waits, step=step)
        moving_average_paper = moving_average(paper_total_waits, step=step)



        a3.cla()
        # Chuyển thành dạng list để dễ dàng truyền vào title của biểu đồ a3 và cập nhật qua thời gian
        current_moving_avg_rfid = list(moving_average_rfid.values())[-1] if moving_average_rfid else 0
        current_moving_avg_paper = list(moving_average_paper.values())[-1] if moving_average_paper else 0

        a3.set_title(f"Moving Average, step={step}\nRFID: {seconds_to_minutes_string(current_moving_avg_rfid)}\nPAPER: {seconds_to_minutes_string(current_moving_avg_paper)}")
        a3.set_xlabel("Time")
        a3.set_ylabel("Avg. Wait Time (seconds)")

        # Step plot, cập nhật theo thời gian
        # Plot chứa 2 đường, mỗi đường đại diện cho một cổng
        a3.step([ t for (t, moving_avg) in moving_average_rfid.items() ], [ moving_avg for (t, moving_avg) in moving_average_rfid.items() ], label='RFID', color='#4A9658')
        a3.step([ t for (t, moving_avg) in moving_average_paper.items() ], [ moving_avg for (t, moving_avg) in moving_average_paper.items() ], label='PAPER', color='#FF914C')

        a3.legend()



        data_plot.draw()
        self.canvas.update()

# Nhận các thông tin về hình ảnh cổng từ và cổng giấy trên giao diện
graphic_rfid_gates, graphic_paper_gates = graphic_gates(canvas, 340, 10)

# Tính toán tọa độ để hiển thị bảng đồng hồ và data
```

```
clock_and_data_width = 190
clock_and_data_height = 70
x1 = 10
# Bảng ở giữa theo chiều dọc canvas
y1 = (canvas_height / 2) - (clock_and_data_height / 2)
# Tọa độ trục hoành - phía bên phải của bảng
x2=x1+clock_and_data_width
# Tọa độ trục tung - phía dưới của bảng
y2=y1+clock_and_data_height
clock = ClockAndData(canvas,
                     x1=x1,
                     y1=y1,
                     x2=x2,
                     y2=y2,
                     time=0)

def create_clock(env):
    """
        Tạo đồng hồ để điều chỉnh tốc độ của giả lập khi hiển thị lên UI.
        Bằng cách điều chỉnh kích thước mỗi nhịp thời gian trong giả lập.
        Với mỗi giây trôi qua ngoài đời thật, thì bao nhiêu giây trôi qua
trong giả lập.
        secs_passed_in_sim_per_real_sec sẽ là một nhịp giả lập.
    """
    secs_passed_in_sim_per_real_sec = int(input('Seconds passed in simulation
per real second: '))

    while True:
        yield env.timeout(secs_passed_in_sim_per_real_sec)
        clock.tick(env.now)



# TẠO MÔI TRƯỜNG
env = simpy.Environment()

# Tạo các tài nguyên của môi trường
# Mỗi tài nguyên chỉ được một xe sử dụng (scan, error) cùng lúc, còn những xe
khác phải chờ
```

```
rfid_gate_lines = [simpy.Resource(env, capacity=1) for _ in
range(RFID_GATE_LINES)]
paper_gate_lines = [simpy.Resource(env, capacity=1) for _ in
range(PAPER_GATE_LINES)]
all_gate_lines = [rfid_gate_lines, paper_gate_lines]



env.process(vehicle_arrival(env, rfid_gate_lines, paper_gate_lines))
env.process(create_clock(env))

# Từ 6:30 đến 20:30 là 14 tiếng đồng hồ
hours = 14
# Đổi từ giờ sang giây
seconds = hours*60*60
env.run(until=seconds)
root.mainloop()



# Writing data to a JSON file
with open(r'output\GUI_events.json', 'w') as outfile:
    json.dump({"RFID GATES": RFID_GATE_LINES,
               "PAPER GATES": PAPER_GATE_LINES,
               "RFID EMPLOYEES": int(RFID_GATE_LINES * RFID_EMPS_PER_LINE),
               "PAPER EMPLOYEES": PAPER_GATE_LINES * PAPER_EMPS_PER_LINE,
               "events": event_log}, outfile, indent=4)
```

Nguồn tham khảo

1. *Basic Concepts — SimPy 4.1.1 documentation*
2. *What Is GUI? Graphical User Interfaces, Explained*
3. *What is Tkinter used for and how to install this Python Framework? - ActiveState*
4. *Prescriptive analytics: Literature review and research challenges - ScienceDirect*

LINK YOUTUBE: Optimizing University Parking Ticket Checking with Discrete Event Simulation | SimPy & Tkinter