

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI



Báo cáo Project II

MÔ PHỎNG ĐA TÁC TỬ SỬ DỤNG THƯ VIỆN REPAST

Sinh viên thực hiện:

Tạ Quang Tùng

MSSV: 20154280

Lớp: KSTN-CNTT-K60

Giáo viên hướng dẫn:

TS. Phạm Đăng Hải

Hà Nội, Ngày 23 tháng 6 năm 2018

Mục lục

1	Mô hình mô phỏng	1
2	Giới thiệu về hệ đa tác tử	2
2.1	Tác tử - Agent	2
2.2	Tác tử thông minh - Intelligent Agent	2
2.3	Hệ đa tác tử - Multi-agent System	3
3	Giới thiệu về thư viện Repast Symphony	5
3.1	Context	5
3.2	Context Loading	5
3.3	Projection	6
4	Mô phỏng giao thông sử dụng thư viện Repast Symphony	8
4.1	Bài toán	8
4.2	Tác tử tĩnh - đường đi	9
4.3	Tác tử động	9
4.4	Kết quả đạt được	10
	Tài liệu tham khảo	14

1 Mô hình mô phỏng

Một mô hình mô phỏng (simulation model) bao gồm một tập các quy tắc định nghĩa cách thức mà hệ thống thay đổi theo thời gian, với một trạng thái hiện tại bất kì. [5] Không như mô hình phân tích, mô hình mô phỏng không được giải (bằng tính toán hay chứng minh) mà dựa vào việc chạy và thay đổi trạng thái của hệ thống và quan sát giá trị tại trạng thái bất kì thời điểm nào. Nó cung cấp một cái nhìn vào bên trong hệ thống thay vì chỉ dựa vào một kết quả cụ thể nào. Giả lập không phải công cụ quyết định mà là công cụ hỗ trợ quyết định, cho phép tạo ra những lựa chọn tốt hơn.

Do sự phức tạp của các bài toán thực tế nên một mô hình mô phỏng chỉ có thể là một mô hình xấp xỉ của hệ thống đích. Do đó việc trừu tượng hóa và đơn giản hóa là công việc thiết yếu trong thiết kế một mô hình mô phỏng. Chỉ những đặc trưng quan trọng để nghiên cứu và phân tích của hệ thống đích mới nên được đưa vào trong mô hình.

Có ba loại kĩ thuật mô phỏng thường được sử dụng:

- Mô phỏng sự kiện rời rạc - Bao gồm một tập các thực thể được xử lý và tiến triển theo thời gian dựa vào sự xuất hiện hay biến mất của các tài nguyên hoặc sự xuất hiện của các sự kiện. Được xây dựng dựa trên một hàng đợi các sự kiện.
- Mô phỏng hệ thống động - Mô phỏng hệ thống theo thời gian, dựa vào mối quan hệ giữa các thực thể hoặc dựa vào một phương trình vi phân nào đó.
- Mô phỏng tác tử - Mô phỏng hệ thức ở mức thức. Những cấu trúc ở mức cao được tạo thành bởi các hành động của các tác tử và tương tác giữa chúng.

Mặc dù mô phỏng máy tính đã được sử dụng rộng rãi từ những năm 1960 nhưng mô phỏng tác tử chỉ trở nên phổ biến từ những năm 1990. Nó hiện tại là một công cụ mô hình mô phỏng được xây dựng mạnh mẽ và bắt đầu phổ biến trong giảng dạy và trong công nghiệp. Mô phỏng tác tử được sử dụng trong trường hợp mà sự tương tác giữa các tác tử không thể bị xem nhẹ. Nó cho phép nghiên cứu về tính động của hệ thống thông qua những đặc điểm của từng cá thể trong môi trường. Mô phỏng tác tử là một mô hình hứa hẹn vì nó dựa vào ý tưởng rằng mọi việc của con người được thực hiện nhờ sự thông minh, giao tiếp và hợp tác.

2 Giới thiệu về hệ đa tác tử

2.1 Tác tử - Agent

Một tác tử phần mềm, hay gọi tắt là tác tử là một chương trình máy tính tồn tại trong một môi trường nhất định, tự động hành động phản ứng lại sự thay đổi của môi trường nhằm đáp ứng một mục tiêu đã được thiết kế trước. Thoả mãn các tính chất sau:

- Bền bỉ - Code không được thực thi theo yêu cầu mà chạy liên tục và tự động quyết định khi nào nên thực hiện hành động nào.
- Tự động - Có khả năng lựa chọn nhiệm vụ, sắp xếp mức độ ưu tiên, hành động hướng mục tiêu, có khả năng tự quyết định mà không cần can thiệp từ con người.
- Xã hội - Có khả năng hợp tác để thực hiện nhiệm vụ thông qua giao tiếp và phối hợp.
- Phản ứng - Tác tử nhận thức từ môi trường và phản ứng lại nó một cách hợp lý.

Phân biệt tác tử với chương trình

Tất cả tác tử là chương trình, nhưng chương trình có thể không là tác tử. Sự khác biệt đến từ các tính chất cơ bản của tác tử so với một chương trình bất kì.

Phân biệt tác tử với đối tượng

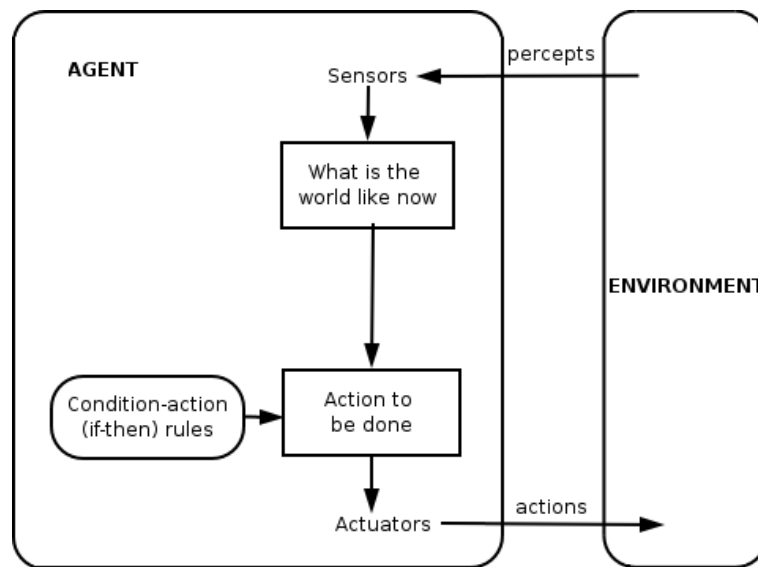
- Tác tử có tính tự động hơn là đối tượng.
- Tác tử có sự mềm dẻo trong hành vi, có sự phản ứng, tính xã hội.
- Tác tử có thể có nhiều hơn một luồng xử lý.

Phân biệt tác tử với hệ chuyên gia

- Hệ chuyên gia không được gắn với môi trường.
- Hệ chuyên gia không được thiết kế với khả năng phản xạ.
- Hệ chuyên gia không xét đến tính xã hội của các vấn đề.

2.2 Tác tử thông minh - Intelligent Agent

Tác tử thông minh là một thực thể tự động, tiếp nhận thông tin từ môi trường và có khả năng tương tác lại môi trường và hướng hành động của nó nhằm đạt được một mục tiêu nào đó. Tác tử thông minh có thể học và sử dụng kiến thức để đạt được mục tiêu. [1]



Hình 1: Simple reflex agent

Tác tử thông minh có thể có các đặc trưng sau:

- Có thể bổ sung các quy tắc (rule) giải quyết bài toán.
- Thích ứng với môi trường một cách trực tuyến và thời gian thực.
- Có khả năng phân tích được hành động của bản thân, sự thành công và thất bại.
- Học và cải thiện thông qua tương tác với môi trường.
- Học nhanh chóng từ một lượng dữ liệu lớn.
- Có bộ nhớ để lưu trữ và truy cập.
- Có các tham số để đại diện bộ nhớ ngắn hạn và dài hạn, tuổi tác, mức độ quên, ...

2.3 Hệ đa tác tử - Multi-agent System

Hệ đa tác tử (multi-agent system) là một hệ tính toán được tạo thành bởi nhiều tác tử thông minh [2]. Hệ đa tác tử có thể được sử dụng để giải các bài toán khó hoặc không thể giải bằng các phương pháp đơn tác tử hoặc một hệ đơn khối nào. Hệ đa tác tử được cấu thành bởi các tác tử và môi trường của nó.

Các tác tử trong hệ đa tác tử có thể chia thành:

- Tác tử thụ động hay “tác tử không có mục đích” (Ví dụ: vật cản).
- Tác tử chủ động với mục đích đơn giản (Ví dụ: xe cộ).
- Tác tử nhận thức với tính toán phức tạp.

Môi trường của các tác tử có thể chia thành:

- Ảo.

- Rời rạc.
- Liên tục.

Môi trường cũng có thể được tổ chức dựa theo các thuộc tính như “khả năng tiếp cận”, “tính xác định”, “tính động”, “tính rời rạc”, “tính giai đoạn”, “số chiều”.

Đặc trưng của hệ đa tác tử:

- Tự trị: Độc lập ít nhất một phần, tự nhận thức, tự động.
- Tầm nhìn giới hạn: Không có tác tử nào có một tầm nhìn cho toàn bộ môi trường, hoặc hệ thống quá phức tạp để một tác tử có thể khai thác được lượng kiến thức đó.
- Phân cấp: Không tác tử nào được chỉ định là tác tử chỉ huy.

Hệ đa tác tử đa phần được thực hiện bằng chương trình giả lập. Hệ thống thực hiện lần lượt từng “bước thời gian”.

3 Giới thiệu về thư viện Repast Symphony

Repast Symphony (version hiện tại 2.5) là một hệ đa nền tảng, tích hợp, tương tác chạy bằng Java. Nó hỗ trợ phát triển những mô hình mềm dẻo đa tác tử cho các máy chủ và các cụm máy tính. [4] Kiến trúc của Repast Symphony dựa trên nguyên nhiều các hệ thống mô phỏng tác tử xây dựng trước đó. Nguyên lý thiết kế của Repast S: [3]

- Có sự tách biệt rõ ràng giữa mô hình, lưu trữ dữ liệu và hiển thị.
- Phần lớn các chức năng có thể được thực hiện mà không phải implement một interface, mở rộng một lớp hay sử dụng proxy.
- Các chức năng cơ bản nên được tự động khi có thể.
- Những đoạn code lặp lại, rườm rà nhưng bắt buộc nên bị loại bỏ hoặc thay thế bởi những thiết lập ở runtime khi có thể.
- Những code cơ bản (như duyệt qua danh sách các tác tử) nên đơn giản và trực tiếp.

Trong project này tôi sử dụng Java để xây dựng các mô hình đa tác tử.

3.1 Context

Context là đối tượng cơ bản trong Repast S. Nó cung cấp cấu trúc dữ liệu để tổ chức các tác tử ở cả phương diện mô hình hóa và phương diện lập trình. Một cách cơ bản thì Context là một đối tượng chứa các tác tử và các chức năng liên quan. Nó là một loại container dựa trên cấu trúc tập hợp. Bất kì một kiểu Object nào cũng có thể được lưu vào Context. Ở phương diện mô hình hóa, Context đại diện cho một quần thể.

Context có thể tổ chức dưỡng dạng phân cấp với Context cha và các Context con. Context còn có thể chứa các Projection. Tác tử có thể được thêm hoặc xóa khỏi Context tại bất kì thời điểm nào. Projection là đối tượng chỉ định mối quan hệ giữa tác tử với môi trường hoặc các tác tử khác.

3.2 Context Loading

Context Loading là quá trình xây dựng Context gốc với các tác tử, Project và các Context con. Repast S cung cấp một interface **ContextBuilder** để thực hiện context loading.

```
public interface ContextBuilder<T> {  
    Context build(Context<T> context);  
}
```

Mô hình chỉ nên có một **ContextBuilder**. Tham số vào hàm build là đối tượng context chưa chứa các đối tượng khác. Các lớp implement interface này tạo và thêm vào context các đối tượng cần thiết. Để lớp implement của **ContextBuilder** được sử dụng thì cần phải thiết lập Data Loader.

3.3 Projection

Projection là các đối tượng thể hiện mối quan hệ giữa các đối tượng trong Context. Mỗi một Projection gắn với một Context và được áp dụng cho tất cả các tác tử trong context đó. Một Context có thể có nhiều Projection. Projection có thể là:

- Lưới rời rạc đa chiều.
- Không gian liên tục đa chiều.
- Đồ thị.
- Không gian GIS.

Tác tử có thể tham chiếu đến các Projection thông qua Context mà đang chứa nó và tên của Projection.

```
Context context = ContextUtils.getContext (this)
Projection projection = context.getProjection("mynetwork");
```

Để tạo một Projection trong **ContextBuilder** thì phải thông qua các bước:

1. Tìm Factory tương ứng.
2. Sử dụng Factory để tạo Project.

```
ContinuousSpaceFactory factory = ContinuousSpaceFactoryFinder
    .createContinuousSpaceFactory(new HashMap());
ContinuousSpace space = factory.createContinuousSpace(
    "simple space", context, ...);
```

Grid

Grid là trụ cột trong mô phỏng đa tác tử. Rất nhiều những mô phỏng xuất hiện sơ khai được xây dựng trên Grid. Về cơ bản grid là một cấu trúc dữ liệu đa chiều chia không gian thành các cell. Các cell này có thể được tham chiếu tới thông qua tọa độ của chúng. Hay nói cách khác, grid là một ma trận n chiều. Grid được sử dụng phổ biến ngay cả bây giờ. Cấu trúc dữ liệu này rất hiệu quả và có nhiều cách xác định hàng xóm của một cell.

Tạo một grid:

```
GridFactory factory = GridFactoryFinder
    .createGridFactory(new HashMap());
Grid grid = factory.createGrid(
    "My Grid", context, gridBuilderParameters);
```

“My Grid” là tên của Grid, **context** là đối tượng kiểu Context, **gridBuilderParameters** là những tham số thiết lập của Grid.

Ví dụ **GridBuilderParameters**:

```
GridBuilderParameters<Object> params =  
    new GridBuilderParameters<Object> (  
        new InfiniteBorders<Object>(),  
        new SimpleGridAdder<Object>(), true,  
        100, 200);
```

Tham số chỉ định grid có biên vô hạn, cho phép nhiều object trong một cell, kích thước 100x200.

Continuous Space

Một projection không gian liên tục (giá trị thực) là một không gian mà vị trí của mỗi một tác tử được đại diện bằng các giá trị float, ngược lại so với giá trị nguyên của Grid. Tuy nhiên nó không hiệu quả để tìm kiếm hàng xóm của một tác tử nào đó.

Các bước tạo một Continuous Space tương tự như tạo một Grid.

Network

Nhiều những mô hình đa tác tử sử dụng mối quan hệ giữa các tác tử thông qua mạng (network) hay đồ thị (graph). Trong Repast thì mạng là một Project có thể được sử dụng dễ dàng như các Project khác.

Tạo một Network:

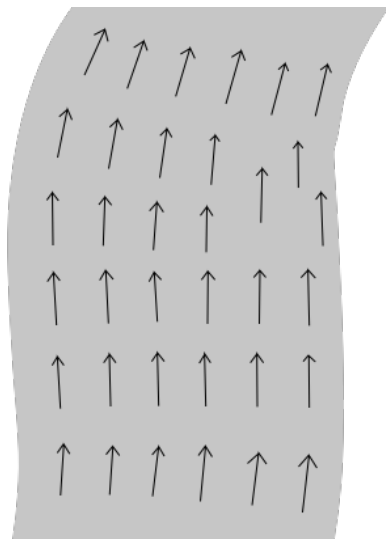
```
NetworkBuilder builder =  
    new NetworkBuilder("Network", context, true);  
Network network = builder.buildNetwork();
```

Tham số cuối cùng của **NetworkBuilder** chỉ định đồ thị có hướng hay không.

4 Mô phỏng giao thông sử dụng thư viện Repast Symphony

4.1 Bài toán

Mô phỏng sự di chuyển của các xe trên các đường và tại các ngã tư. Từ đó có thể tính các thông số trong quá trình di chuyển như quãng đường đi được, thời gian đi, lượng năng lượng tiêu thụ, ... của mỗi xe. Các xe di chuyển dựa trên hướng đi được xác định sẵn.



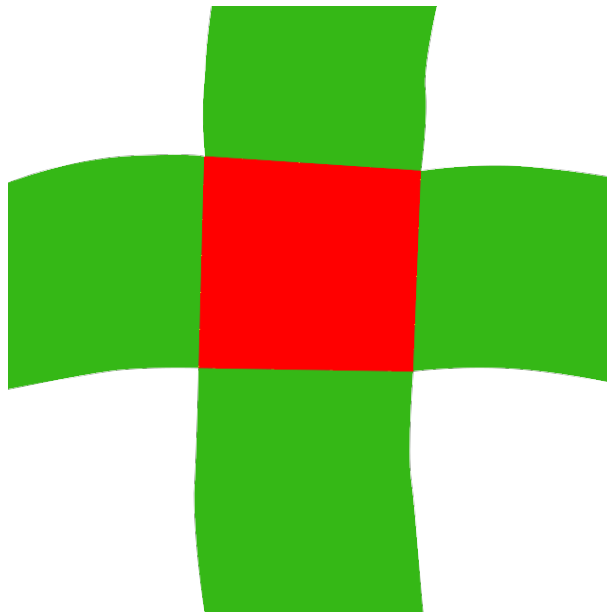
Hình 2: Trường vector chỉ định hướng đi trên một đoạn đường

Các tác tử được chia thành hai loại:

- Tác tử tĩnh: Đại diện cho đường đi.
- Tác tử động: Đại diện cho các xe và các điểm sinh ra các xe.

Đường đi được chia làm hai loại:

- Đường đi một chiều
- Ngã rẽ.

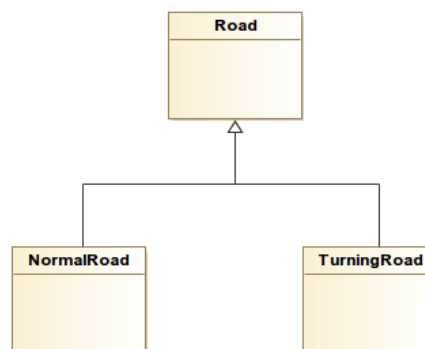


Hình 3: Màu xanh: Đường đi bình thường - Màu đỏ: Ngã rẽ

4.2 Tác tử tĩnh - đường đi

Các đường đi được xây dựng bởi các cell, mỗi cell là một object thuộc một trong 2 kiểu:

- **NormalRoad**: Chứa thông tin về hướng di chuyển của các xe trên đường một chiều, chứa cạnh tương ứng trên đồ thị đường đi.
- **TurningRoad**: Chứa thông tin về hướng di chuyển của các xe tương ứng từ cạnh này sang cạnh kia của đồ thị đường đi.



Hình 4: Biểu đồ lớp

4.3 Tác tử động

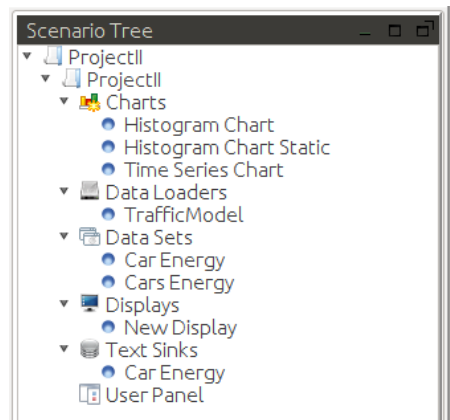
- Các xe - **Car**: Mỗi xe chứa thông tin về vận tốc, năng lượng tiêu thụ. Ở mỗi bước thời gian của mô phỏng, xe xác định hướng dựa trên đường đi mà nó đang đứng (nếu đang ở đường đi một chiều - **NormalRoad**) hoặc dựa vào cặp cạnh (chọn ngẫu nhiên) trên đồ thị

đường đi để xác định hướng đi dựa trên ngã rẽ mà nó đang đứng (nếu đang ở đoạn ngã rẽ - **TurningRoad**).

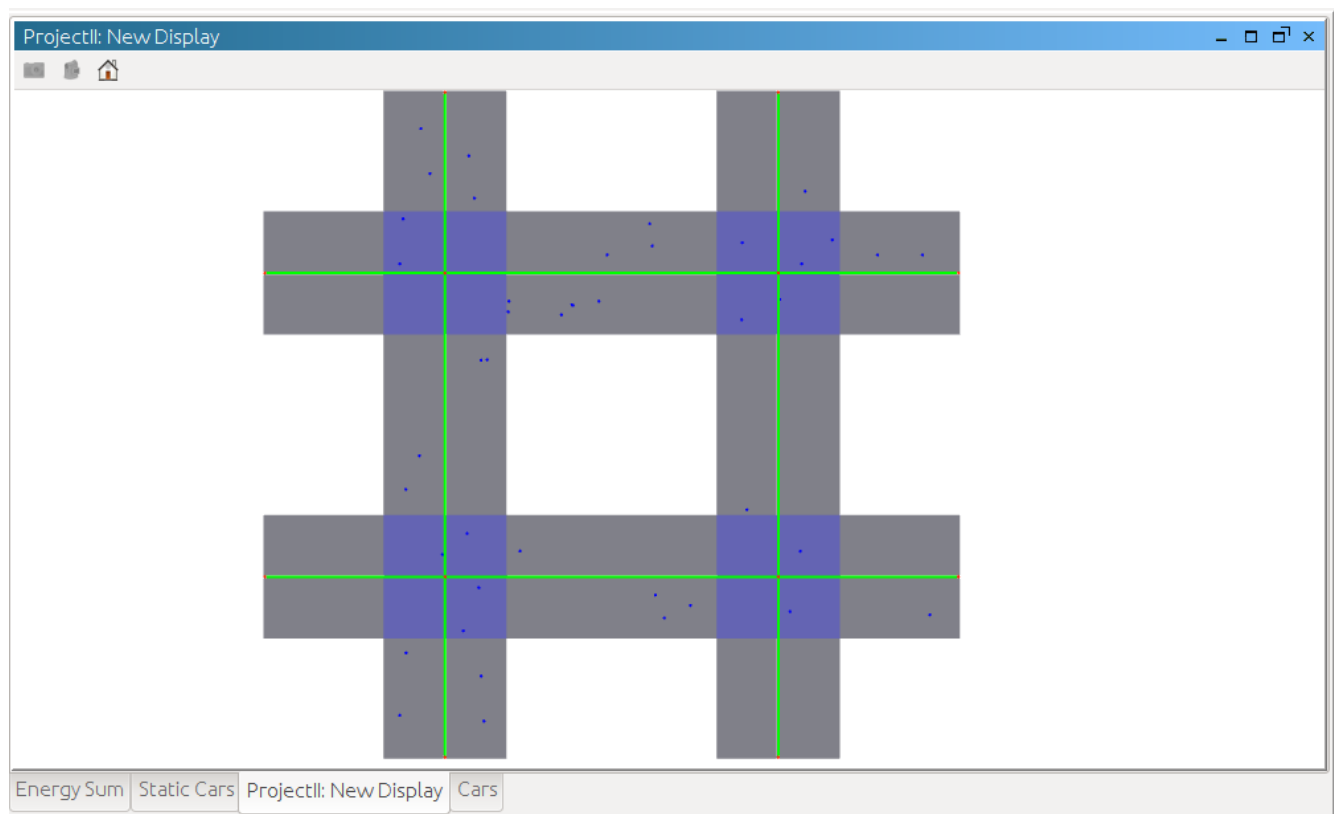
Từ vận tốc và hướng đi xác định được vị trí ở bước thời gian tiếp theo. Các xe có thể quan sát vị trí các xe xung quanh để quyết định có nên giảm tốc độ / tăng tốc độ hay không.

- Các điểm sinh xe - **CarGenerator**: Có tọa độ là một vị trí nào đó trên các đường một chiều, tọa độ của các xe được sinh ngẫu nhiên xung quanh điểm đó.

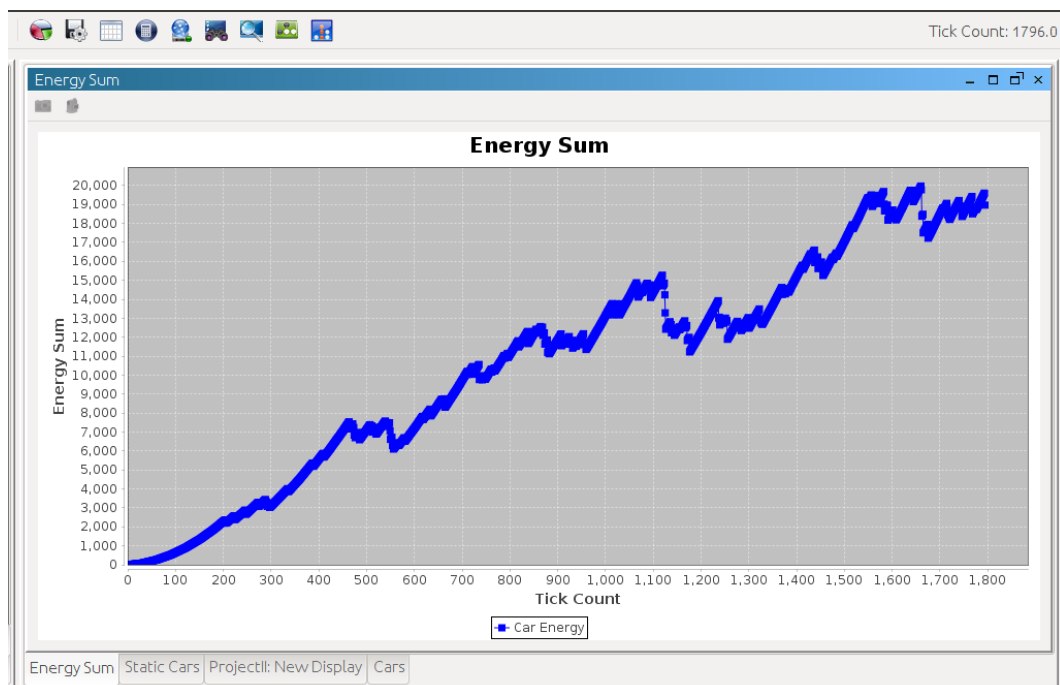
4.4 Kết quả đạt được



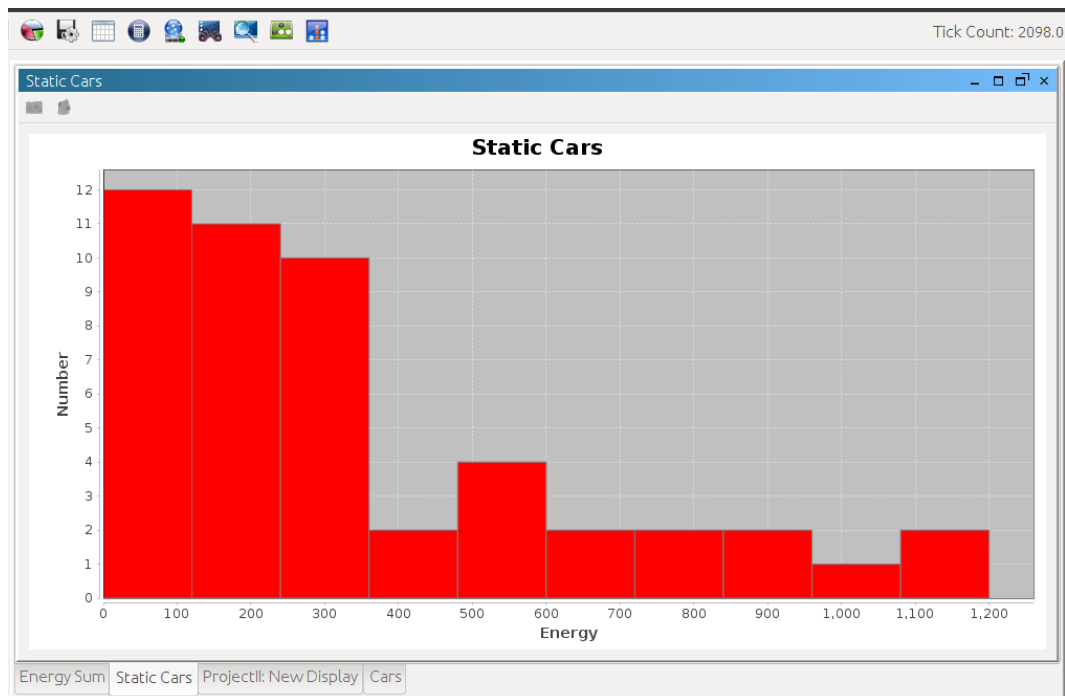
Hình 5: Thiết lập được sử dụng



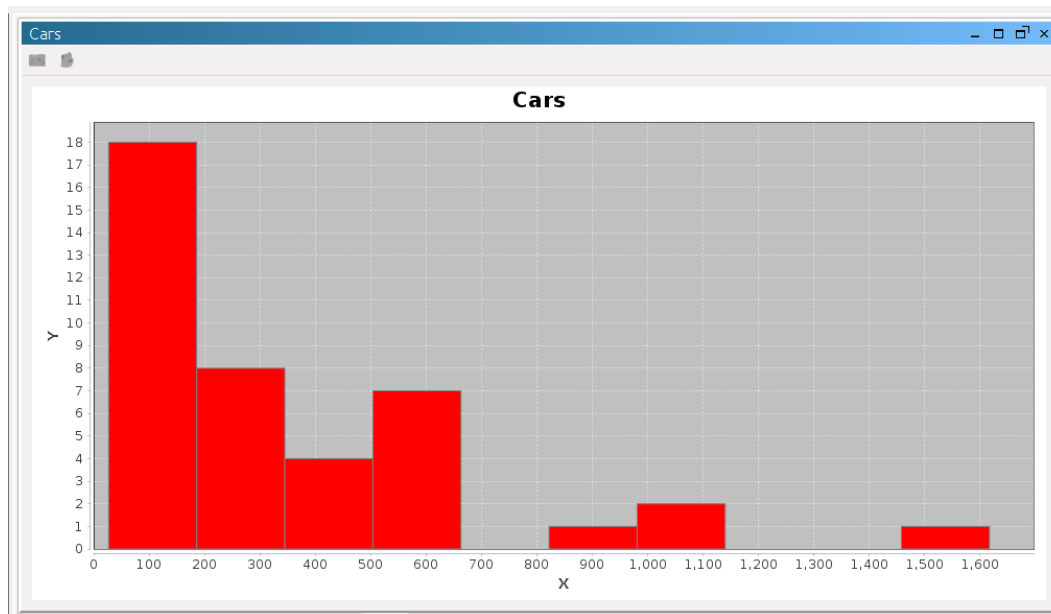
Hình 6: Mô phỏng được hiển thị



Hình 7: Biểu đồ tổng năng lượng tiêu thụ theo thời gian

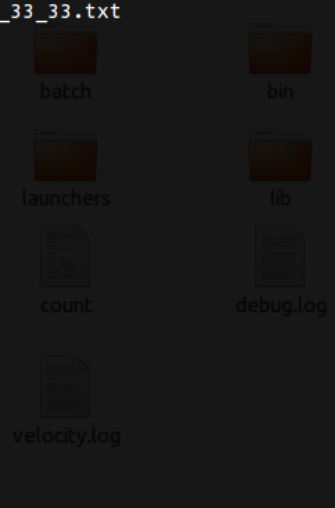


Hình 8: Biểu đồ histogram năng lượng tiêu thụ với biên cố định



Hình 9: Biểu đồ histogram năng lượng tiêu thụ với biên thay đổi

```
% cat ModelOutput.2018.Jun.23.12_33_33.txt
"tick":"Car Energy"
1.0:0.0
2.0:3.0
3.0:6.0
4.0:9.0
5.0:12.0
6.0:15.0
7.0:18.0
8.0:21.0
9.0:24.0
10.0:27.0
11.0:30.0
12.0:33.0
13.0:36.0
14.0:39.0
15.0:42.0
16.0:45.0
17.0:48.0
18.0:51.0
19.0:54.0
20.0:57.0
```



Hình 10: File chứa tổng năng lượng tiêu thụ ở mỗi bước thời gian

Tài liệu tham khảo

- [1] Intelligent agent - wikipedia. https://en.wikipedia.org/wiki/Intelligent_agent.
- [2] Multi-agent system - wikipedia. https://en.wikipedia.org/wiki/Multi-agent_system.
- [3] Repast simphony documentation. <https://repast.github.io/docs/RepastReference/RepastReference.html>.
- [4] Repast suite documentation. https://repast.github.io/repast_simphony.html.
- [5] Peer-Olaf Siebers và Uwe Aickelin. Introduction to multi-agent simulation. <https://arxiv.org/pdf/0803.3905.pdf>.