Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/ specialization: Computer Science Specialisation Reliable and Secure Systems	Spring semester, 2021 Open
Author: Joel Fabiean Joseph	
Instructor: Reggie Davidrajuh Supervisor(s): Reggie Davidrajuh and Naeem Khademi	
Title of Master's Thesis: Developing Models of Road Tunnels with Petri Nets	
ECTS: 30	
Key words: Road tunnel, GPenSIM, Petri net, Single-lane	Pages: 74 Stavanger 15. June 2021

Developing Models of Road Tunnels with Petri Nets

Joel Fabiean Joseph

June 2021

Department of Electrical Engineering and Computer Science
Faculty of Science of Technology
University of Stavanger

Contents

Abstract

This thesis aimed to develop a mathematical model using a Petri net to simulate the traffic flow inside the road tunnel. The purpose was to create a single-lane tunnel that could simulate the flow while also including an event, which slows down a vehicle to cause congestion. The program uses modules to divided the tunnel into smaller segments and compute the vehicle distance inside these segments. Three different simulations using four tunnel segments yield an expected outcome, except for higher vehicle numbers due to the tunnel segment being too long. The program is still not usable and requires further development before put into practice.

Chapter 1

Introduction

1.1 Motivation

When deploying a road tunnel, several factors have to be taken into consideration during construction. One crucial factor is the security inside the tunnels since the accident is more severe than accidents outside. Therefore, it is vital to construct the tunnel to prevent an accident. However, it is then essential to implement several tools to detect as fast as possible and execute an appropriate method to solve the issue if it does happen. Another critical factor is the driver's well-being as the environment inside the tunnels is much darker and cramped, so it is crucial to improve the well-being and prevent or reduce the traffic queue. Using a simulation tool to simulate the traffic flow inside the tunnel would help determine how to construct a tunnel to solve issues to prevent or mitigate the possibility of a traffic queue. A simulation tool could also simulate the traffic flow during an accident, which could improve the current strategies for dealing with traffic flow or improve existing detection tools.

1.2 Related work

There are several mathematical models for traffic flow or collision. The authors of a paper created a queuing model based on traffic counts and model the behavior of traffic flow as a function of relevant determinants [10]. Another research created a crash-prediction model using road tunnel data from 2006-2009 of a single tube tunnel with unidirectional traffic [2]. There have also been developed other types of models using Petri net. One paper proposed an extension for triangular Batch Petri net, which is a hybrid Petri net, for better applicability for urban and road traffic [9]. A different paper using a fluid stochastic Petri net, which again is also a hybrid Petri net. This paper modeled a car safety controller in a road tunnel to have vehicles communicate to keep a certain distance between each other [1].

1.3 Outline

During the thesis, the chapters are organized as follow:

Chapter 2 describes what Petri net is and the extension of the Petri net used in this program.

Chapter 3 introduces the assumption of the program and the description of the program setup idea.

Chapter 4 presents the implementation of the program and the different functions created for this program.

Chapter 5 provide three different simulation tests and the expected output.

Chapter 6 discuss the output of the simulation and describe further work to improve the program.

At the end is a conclusion of everything presented.

Chapter 2

Background

This chapter presents an understanding of the Petri net and its different extensions of the Petri net. It will also go through the formulas used to realize the system.

2.1 Petri net

Petri net is a discrete mathematical graph model used for simulation and analysis. The Petri net consists of two components, which are places and transition. These components make up the static model of the system. The Petri net uses tokens to display the dynamic portion by traversing through the static model.

The first component mention is the places. These places serve as a placeholder or buffer for the tokens. A place cannot directly connect with other places, but it can connect with multiple transitions. The second component is the transition. They are used to transport the tokens between places. The transition cannot directly connect with other transitions but can connect with multiple places. The Petri net uses tokens, which symbolize an object that traverses through the system. The token itself has no value and only displays how the object in the Petri net flows through the system.

The connection between a place and a transition is called an arc. The arc is unidirectional, meaning it is either connected from place to transition or from transition to a place. The arc in Petri net at default takes one token, but the arc could also be weighted, increasing the number of tokens it requires.

The process of a transition taking a token from a place is called enabled while sending a transition to a place is called fire. A transition can only be enabled when the number of tokens in a place is greater than or equal to the weight of the individual arc. Thus, a transition can take a token from multiple places but can only be enabled when the arc of all those places has the correct number of tokens; when the transition fire, The arc from the transition to the place will then produce a token corresponding with its weight.

The classical Petri net has some weaknesses or problems, but these issues are solved using different extensions. Therefore, this chapter will not discuss all of them but rather the ones used in the program.

2.1.1 Timed petri net

In classical Petri net, the transition fire immediately, the firing time is zero. However, in the real world, the action does not happen immediately but takes time. An extension called timed Petri net solved this issue. This extension allows the transition to hold the token for a certain amount of time and fire. However, the timed Petri net is not backward compatible with the classical Petri net; all the transitions require a time value different from zero.

2.1.2 Colored petri net

Tokens in the Petri net do not possess any value but are all equal and indistinguishable. However, the colored Petri net introduces the ability to distinguish tokens by giving them values. In addition, the colored Petri net allows a token transition to choose a specific token based on arrival time or a specific value.

2.1.3 Prioritized petri net

If two transitions take tokens from the same place, which transition will take the token? There is no specified method in classical Petri net, meaning both transitions will compete for the token. The competition causes the system to be unpredictable and leads the token to be chosen by an undesirable transition. The extension of prioritized Petri net introduces the ability to prioritize which transition can choose first. Instead of a competition between two transitions, the transition with a higher priority will take the token before the next transition gets to chose.

Chapter 3

Program Design

This chapter present with the design of the model used for the implementing the program and the different methods used to compute or decision making.

3.1 Assumptions

Several assumptions had to be defined when implementing this program due to a lack of data or the Petri net itself.

3.1.1 Blind vehicle

The first assumption is that vehicles in the tunnel cannot see other vehicles in front or behind. This assumption is due to the transition only work with one token at a time. When defining the arc, we can choose how many tokens to take, but it has to be an equal or higher number of tokens in the transition to be enabled. If we were to take two tokens, the program would never run if there was only one token. Another problem is that transitions are only aware of the value of the tokens, but not what they did previously. The vehicles are not aware of each other.

3.1.2 Constant Speed

In a real-life situation, the vehicle speed varies over time and has a speed approximate to the speed limit in the tunnel. In this implementation, the vehicles have a constant speed, and the speed will match the speed limit unless something else is specified.

3.1.3 Constant Acceleration

Inside the tunnel, the speed limit might change. When a vehicle starts changing the speed to match the speed limit, it has to accelerate. Of course, the vehicle's acceleration is not the same for all vehicles, but the program assumes that they all accelerate with a constant acceleration unless specified. The reason is that the vehicle is not aware of each other, to prevent inconsistency in their movement, they operate with a constant acceleration.

3.1.4 Trigger upon discovery

Another assumption to react immediately. When a vehicle sees a speed sign, it may start changing the speed before passing it or after. In this simulation, if any situation causes a change to the vehicle, it will react at the speed sign, aka at the new segment. This is due to the difficulty of simulating a person's reaction time or the driver's behavior. For the sake of simplicity, it will trigger immediately.

3.1.5 global event

In this program, there is an event, which slows down the speed of a vehicle. Upon triggering, all vehicles that are behind the position of the event will take immediate action. This is due to assumptions from 3.1.1 and 3.1.2.

3.2 Model Design

Figure 3.1: Double-lane road tunnel [7]

The idea behind the design of the program was easy deployment and expansion. The current model consists of three different components.

The first component is the generator, which simulates the vehicles' arrival time and sets the vehicle's initial value. These values are:

- Speed
- Distance
- Length of the vehicle
- Event

The second component is a tunnel segment. This component's purpose is to compute the position of the vehicles in the tunnel. A *tunnel segment* is a module that consists of one place and three transitions.

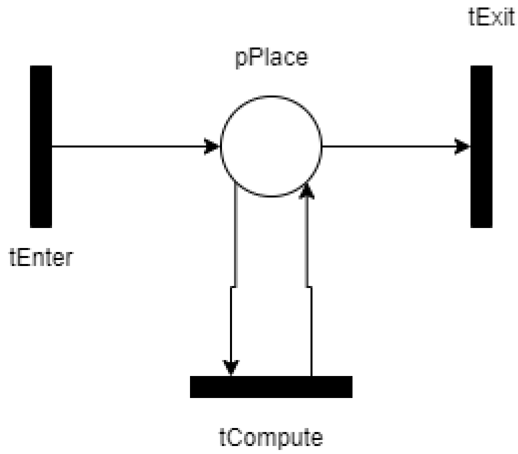

Figure 3.2: Petri net module of tunnel segment

The first transition represents an entry point of the tunnel segment. A vehicle can enter the segment only if there is enough space inside the tunnel. The second transition is for computation, where it computes the new position of the vehicle. The last transition is the exit point of the segment, and this transition only fires if there is enough space in the next segment. If there is not no more segment, it will just fire.

The last component is the buffer, which only consists of a place. These buffers can be found right before the entrance of the tunnel and after the end. The placement of the buffers is between the tunnel segments. These buffers currently act as a placeholder for tokens during the transitions of segments.

3.3 Computation

The single-lane tunnel can be seen as one-dimensional since they only need to consider the x-coordinate. The computation that occurs in the tunnel is to find the displacement of the vehicle. The formulas used in the programs are:

$$t_s - t_l = \begin{cases} 1, \geq 1 \\ t, \text{otherwise} \end{cases} \tag{3.1}$$

$$d = d_0 + vt + \frac{1}{2}at^2 \tag{3.2}$$

$$d = d_0 + vt \tag{3.3}$$

Equation 3.2 is the displacement formula with constant acceleration. Value d_0 is the current displacement of the vehicle, **v** is the velocity, **a** is the acceleration and **t** is the time. If the acceleration value is zero, it will become equation 3.3. This formula is to compute the displacement, when the velocity is constant.

Equation 3.1 is to compute the time the vehicle has been waiting at the place. The value t_s referrer to the system time, while t_l is the time token was released from a transition. The formula state that the time **t** the token spent in the tunnel segment cannot be zero but becomes one. The vehicle inside the tunnel is continuously moving unless something happens in the tunnel. If **t** is zero, the displacement of the vehicle will not change.

3.4 Token design

The tokens in this program are simulating the vehicles. Since the transition is not aware of the token's displacement, all the values are in the tokens. Table 3.1 consists of all the values stored in the token.

The value **Dist, Speed** and **VLen** from table 3.1 is easy to understand from the description. Each tunnel segment has a fixed length, but they can be different. The **CurDist** is the remaining distance inside these segments. This value will decrements equal to the value of the distance traveled. When the value is zero or less, the token will include the **True** flag. If during computation, the **CurDist** is less or equal to zero, and there is still extra

Value name	Description
Dist	The total travel distance or displacement in the tunnel
CurDist	The remaining travel distance inside a tunnel segment
Speed	The velocity of the vehicle
VLen	The lenght of the vehicle
True	A flag to indicate that the token is ready for transition
Time	The extra time spent inside the tunnel segment after the True flag is active

Table 3.1: General values used in a token

time. The extra time will be store in the **Time** value. This value will be included in the computation after transitioning.

There is also a special type of value, which is referred to as events. Currently, there is only one type of event object available, which is the **slow** event. This event causes all the vehicles behind, including itself, to slow down their speed, regardless of the speed limit. This value consists of three types of information. The first one is the name of the event, the second is the distance of activation, and the last value is the new speed of the vehicles.

Chapter 4

Implementation

This chapter focuses on the implementation of the program and the different functions made to realize this.

4.1 Program description

The program implemented is implemented using **MATLAB R2021a - academic use**. The requirement of deploying the program is to download the library **GPENSIM** [4]. This library is used for deploying a Petri net model. During the writing of this paper, the version of **GPenSIM** was 10.0 [8].

4.2 Petri net Definition File

The Petri net Definition File, or PDF for short, is the static model for the Petri net system. **GPenSIM** can simplify the PDF by splitting it up into smaller modules [5, 6]. These modules can then be combined to create the whole system. When creating a PDF, four different values need to be defined.

The first value is the name of the PDF. The Main Simulation File or MSF uses this value to include this file in the simulation. The second and third value is the places and transitions, while the final value is the arcs. When splitting the file into smaller values, the ports of the module have to be defined. The ports represent the input or output of the modules.

4.2.1 Tunnel segment module

Before constructing the tunnel, the tunnel segment has to be defined. As mentioned, the tunnel segment is a PDF module that computes the vehicle inside the segment. The module is a component with one entry point and one exit.

Algorithm 4.1: Snippet code tunnel segment A module

```
1 png.PN_name = 'SegmentA';
2 png.set_of_Ps = {'pA'};
3 png.set_of_Ts = {'tAEnter','tAExit','tComputeA'};
4 png.set_of_As = ...
      {'tAEnter','pA',1,'pA','tAExit',1,'pA','tComputeA',1,...
5     'tComputeA','pA',1};
6 png.set_of_Ports = {'tAEnter','tAExit'};
```

Algorithm 4.1 show a snippet code of the tunnel segment module used in the experiment. The variable in line 2, **set_of_Ps** define the names of the places in Petri net. In a tunnel segment, it only contains one place, which is **pA**. The **set_of_Ts** consists of three different transitions, which represent the entry and exit point of the tunnel segment, while the **tComputeA** is the computation portion of the segment. **set_of_Ports** is what define it as a module, as display in algorithm 4.1. The variable **set_of_As** defined the arcs between the transitions and place. The weight of all the arcs in the transition is one, and the connection of **tComputeA** and **pA** consists of two arcs, one in each direction.

4.2.2 Tunnel PDF

After creating the tunnel segment modules, it is time to finalize the static model of the tunnel. Finally, the PDF of the tunnel combine the generator, buffers and all the tunnel segment to create a tunnel.

Algorithm 4.2: Snippet of Connect_PDF

```
1 png.PN_name = 'Tunnel System';
2 png.set_of_Ps = ...
      {'pOutside1','pOutside2','pBuffer1','pBuffer2','pBuffer3'};
3 png.set_of_Ts = {'tGenerator'};
4 png.set_of_As = ...
      {'tGenerator','pOutside1',1,'pOutside1','tAEnter',1,...
5      'tAExit','pBuffer1',1,'pBuffer1','tBEnter',1,'tBExit',...
6      'pBuffer2',1,'pBuffer2','tCEnter',1,'tCExit',...
7      'pBuffer3', 1,'pBuffer3','tDEnter',1, ...
           'tDExit','pOutside2',1};
```

The snippet from 4.2 shows the PDF of the tunnel system used in the experiment. This tunnel consist of four tunnel segment module. The modules use a letter from A-D to name the segments, for example, **SegmentA**. The tunnel consists of one transistor called **tGenerator** and five places. The two first places in line 2 from algorithm 4.2 named **pOutside1** and **pOutside2** represent the area outside the tunnels. **pOutside1** is outside the entry point of the tunnel, while **pOutside2** is outside the exit point. The three other places represent the buffers between the segments. They currently serve as a placeholder for tokens since the transitions can only connect with places. Observing algorithm 4.2 line 4-7 shows the arcs connection. Observe how the connection between a tunnel segment uses the name of the entry and exit point of the tunnel.

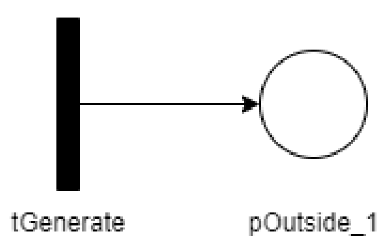

Figure 4.1: Outside the tunnel with only a Generator

The arcs between the tunnel can be explained by first creating a arc from **tGenerator** to **pOutside1** as displayed in figure 4.1.

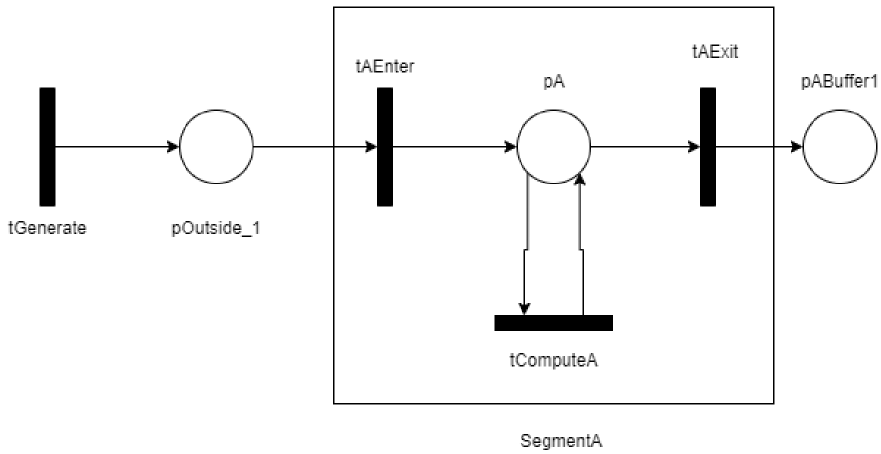

Figure 4.2: Connection between the outside and the first tunnel segment

Afterward, the outside **pOutside1** is connected to the entry point of the tunnel **tAEnter** as displaed in 4.2.

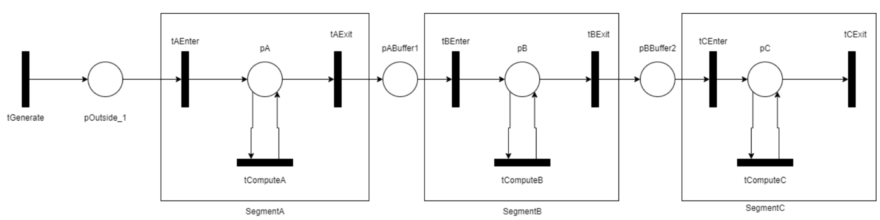

Figure 4.3: The middle part of the tunnel

Then use the buffers to connect the tunnel segment as shown in figure 4.3

In the end, connect the last segment's entry point with the buffer and the exit point with **pOutside2** to finalize the tunnel structure as shown in figure 4.4.

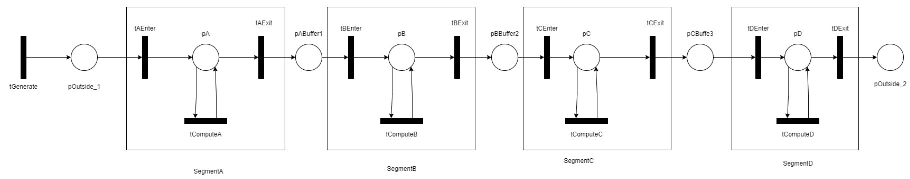

Figure 4.4: The complete tunnel structure with four segments

4.3 Variables

init.m file consists of the variables used in the program. This file consists mainly of the definitions for the global variables.

The two first variables **ComputationTime** and **TransitionTime** are used to control the time of the transitions. The higher value is, the faster the simulation is, but at the cost of accuracy. The **Sim(Hour/Min/Sec)** define how long to simulate the program. The variable **Arrival** is a list containing the firing time of all the tokens without an event. These variables are created based on three parameters. Two of the parameter are start and end. The start is zero, while the end is the simulation time in seconds. The last parameter is the frequency of the arrival of the vehicle. The variable that controls the frequency is **TokenFreq**. The variable **EventArrival** contains the arrival time of tokens with events. The event time has to define manually. The variable **RoadLengthX**, **speedLimitX** and **TotalVLenX** are parameters for the tunnel structure. The **TotalVLenX** sum all the vehicles length in the segment and compare it to **RoadLengthX**. This is to prevent overfilling the tunnel segment with vehicles.

4.3.1 Token Variables

The token variables are values stored in the tokens. Information about these variable can be found in table 3.1 in section 3.4. The tokens' variables are a character array consisting of two values divided by a space. The first value is the variable type, while the second is the value. The character array is again stored as a cell value inside an array, referred to as a color list.

The event variables consist of three values, which are event type, distance,

Variable name	Value type	Description of use
ComputationTime	Integer	Frequency of Computation transition enables
TransitionTime	Integer	Frequency of Enter and Exit transition enables
Sim(Hour/Min/Sec)	Float	Simulation running time in hours/minutes/seconds
Arrival	Array of Integer	List of time to create a token
EventArrival	Array of Integer	List of time to create tokens with events
EventType	Array of Cell	List of events used for creating tokens with events
Events	Array of structure	List of events
Acceleration	Integer	Acceleration of vehicles
Speed	Float	The initial speed of vehicles
RoadLengthX	Float	Length of a Tunnel segment
speedLimitX	Float	Speed limit of a tunnel segment
TotalVLenX	Array of Float	List of all the vehicle in tunnel segment
Display(A-D)	Integer	Display the output of each computation

Table 4.1: Table of global variables inside the **init.m** file

and speed. Each of these value is again divided using space.

4.3.2 Event structure

In table 4.1, the variable **Events** consist of a list of structure. The event structure store four different values. These value are:

- Character array from the event list
- The current distance of the vehicle
- The Event speed

- The time event stopped

4.4 Functions

This section will be focusing on explaining different functions used or created to realize this program. It will first explain short about the functions from **GPenSIM** and afterward explain in detail about the functions created for this program.

4.4.1 GPenSIM functions

In **GPenSIM**, the function used in the programs is either related to time or color. The tokens have three different attributes, which are id, time, and colors. The id of the token allows for fetching the colors and time. The time of the tokens is related to the moment a transition released the token. The time value will be updated every time a new transition is firing the token. The color is a list of values, which was defined in section 4.3.1. It is required to loop through the color list to fetch a specific value. The values inside the list do not always appear in the same order as upon release.

Function name	Input	Output	Description
tokenArrivedEarly	Place_From Tok_Number	Tok_Id	fetch the earliest arrived token
tokenAnyColor	Place_From Tok_Number Colors	Tok_ID	fetch token with with at least one of the colors
get_color	Place_From Tok_Id	Colors	return color of the token
get_tokCT	Place_From Tok_Id	Tok_Time	return time of the token
current_time	N/A	Sys_Time	return current time of the system

Table 4.2: GPenSIM functions used in the program

Table 4.2 shows the different functions from **GPenSIM** that was used during the implementation of the program [3].

4.4.2 FindElement

As mentioned, the **GPenSIM** colors are stored inside a cell value where the ordering is unknown. To solve this issue, the function **FindElement** was created.

Algorithm 4.3: Snippet of **FindElement** function

```
1     for i = 1:length(col)
2          split = strsplit(col{i});
3          switch split{1}
4              case 'Dist'
5                  dist = str2double(split(2));
6              case 'Speed'
7                  speed = str2double(split(2));
8              case 'CurDist'
9                  rem = str2double(split(2));
10             case 'Time'
11                 time = str2double(split(2));
12             case 'VLen'
13                 len = str2double(split(2));
14             case 'True'
15                 continue;
16             otherwise
17                 event = col{i};
18         end
19     end
```

Algorithm 4.3 shows the loop used to find all the values. First, it fetches the value in position **i**. Then it proceeds to split the character array into two string values using the space. The first element will be the variable type, and the second is the value. Using the first string to check the value type as seen in line 3. The value **Dist, Speed, CurDist, Time** and **Vlen**, will transformed into integer or float. The variable **True** will just be ignored and proceed to the next element. If the token has an event, it will just return the event without doing anything. After looping through the colors, it returns all the values, except for **True**.

4.4.3 ComputeDistance

The **ComputeDistance** is the primary function file for computing the displacement of the vehicles. The operation of this function can be explained in the list below:

1. Check for events
2. Compute the distance
3. Update the values
4. Repeat

In the first step of checking the events, the file uses a helper function called **FindEvent** to check for events. It will first check if there is an event in the variable **Events**. If there are any events, it will proceed to check if these events are relevant for the vehicle. The condition for the event to be relevant are:

- if the distance of the vehicle is equal or less than the event
- if the event speed is equal or less than the other events
- if the time is equal or less than the event

In the second step it computes the distance the vehicle travel. The computation occurs in a helper-function called **traveling**. This function compares the vehicle speed and the speed limit of the tunnel and decides how to compute the distance.

Algorithm 4.4: snippet of traveling

```
    diff = maxSpeed - currentSpeed;
     FinSpeed = currentSpeed;
     if abs(diff) < 0.05
```

```
4           travelDistance = (FinSpeed / 3.6);
5       elseif diff < acc && diff > - acc
6           travelDistance = (FinSpeed / 3.6) + (1/2) * diff;
7           FinSpeed = FinSpeed + diff;
8       else
9           if diff > 0
10              travelDistance = (FinSpeed / 3.6) + (1/2) * acc;
11              FinSpeed = FinSpeed + acc;
12          else
13              travelDistance = (FinSpeed / 3.6) - (1/2) * acc;
14              FinSpeed = FinSpeed - acc;
15          end
16      end
```

In algorithm 4.4 line 1, it computes the difference between the speed limit and the vehicle speed. If the outcome is positive, it means that it has a higher speed than the speed limit. The vehicle will then proceed to decelerate using the global **acceleration** variable. If the outcome is negative, the vehicle is lower than the speed limit and will accelerate. If the outcome is zero, the speed is the same as the speed limit and will use equation 3.3 to compute the distance. The difference is not zero in the scenario, but the absolute value of the speed is less than global acceleration. The new acceleration will be equal to the remaining speed required to reach the speed limit.

In step three of updating the values, four values need to be updated.

- Vehicle distance
- Remaining distance
- Speed
- Time

The vehicle distance will increase, while the remaining distance will decrease based on the output from the function **traveling**. If the speed is not constant, the speed will also update based on the same function. The time value will only be updated whenever the remaining distance is less or equal to zero. The time value will compute the new distance in the next segment. The time value will increment equal to the remaining loop.

As mentioned in the equation 3.2 and 3.3, it will compute the displacement. In the implementation, instead of computing the displacement immediately, it treats the time value as one and uses a for loop to compute the displacement. Computing step by step will reduce the lost data in between the time interval.

4.4.4 Element2Char

The function **FindElement** finds all the element and transform the values into integers or float. After computing the values, they have to be transformed back to a character array and place back into the color list. This function's purpose is to transform the value back to character arrays.

Algorithm 4.5: Element2Char function

```
1 function elem = Element2Char(text, val)
2 val2str = string(val);
3 cat = strcat(text,{' '}, val2str);
4 elem = convertStringsToChars(cat);
5 end
```

This function takes in two values, which are the variable code and value. It first starts by transforming the value into a string as shown in algorithm 4.5 in line 2. In line 3, it concatenates the variable code and the string value with a space between them. In the end, it transforms the concatenated value into a character array.

4.4.5 VLengthGenerator

The function **VLengthGenerator** is a simple function to generate a vehicle length randomly. Currently, this function only generates two different vehicle lengths, which are 3m and 5m. This function uses Matlab's built-in number generator, which is uniformly distributed. The probability of generating a vehicle length of 3m is 80%, while a vehicle with a length of 5m is 20%.

4.4.6 FindVLength and RemoveVLen

The function **FindVLength** is a simplified version of **FindElement** function. The **FindVLength** only find the vehicle's current length, which is currently used by the first segment in the enter transition.

The **RemoveVLen** does not remove the vehicle from the segment but instead finds the position of the first vehicle length that matches its own. Only the segment's exit transition uses this to remove the vehicle from the **TotalVLenX**.

4.4.7 EventHandler and RemoveEvent

After using the function **FindElement**, the token might have an event in the color list. This event will be sent to the **EventHandler**. This function will first check whether the event is the global variable **Events**. If the event is registered, it will only update the distance store in the structure. Otherwise, it will add the event to the list if the vehicle distance is greater or equal to the event distance.

When the vehicle with an event exits the tunnel, the vehicles behind will increase their speed. Since all the drivers in the simulation are blind, the vehicle does not know when it leaves the tunnel. If the event is removed from the list, the other vehicle would react immediately and adapt to the tunnel speed.

4.4.8 ComputeSegment

This function was design to merge all the functions used in the computation transition to simplify the implementation. It first finds all the elements using **FindElement** and then proceed to compute using **ComputeDistance**. Afterward, transform all the variables into a character array and store them inside a list as cell values.

4.5 Processors

In **GPenSIM**, there are particular processor files used to add additional conditions for the transitions. There are two types of processor files which are pre and post-processor files. The pre-processor runs before firing, while the post-processor runs after firing. This program does not use any post-processor files, only pre-processor files.

The processor files can be categories into three different types. The first one is the specific processor files, which only applies to one transition. The second is the processor for a module, which only applies to transition inside a module. The last one is the common processor files, which is global file for transition. The common processor file serves the purpose of reducing the specific processor files. This program only uses the specific processor file for the *generator*, while each module has its processor file.

4.5.1 Generator processor

The processor file for the generator is to create tokens and fire them based on the variables **Arrival** and **EventArrival**. The generator first proceeds to create a vehicle length based on the function **VLengthGenerator**. Afterward, it transforms the speed and vehicle length into a character array using the function **Element2Char**.

To determine whether to create a token, it will check the first value in the list of both **Arrival** and **EventArrival**. If the values from one of them match with the system time, it will fire a token. If the value in both lists is the same, it will treat the token as a value. After checking, it will delete the first value in the lists.

4.5.2 Segment processor

The module of a tunnel segment consists of an entry point, an exit point, and the computation. The implementation of the tunnel segment defines three types of segments. The first segment is the start of the tunnel segment, the second is the middle part, while the last is the exit segment. The start and end only have one each, while the rest of the segment is considered the middle part. The difference of these segments are how the **tEnter** transition for the start segment and **tExit** for the last segment are different.

The transition for computation first collects their tokens for the place inside the modules. The transition used the function **tokenArrivedEarly** to fetch the token. It then proceed to compute the time using equation 3.1 and run the function **ComputeSegment**. Afterward, it fires the token with the new colors.

When entering a tunnel segment, it first has to check whether there is space inside the tunnel. If there is enough space in the tunnel, it will add the vehicle length to the variable **TotalVLengthX**. To check if there is enough space, it sums the value inside the list and compares it to the tunnel length. The start segment will set the remaining length of the tunnel equal to the road tunnel length. The other tunnel segments would have to sum the current remaining distance with the tunnel length.

When exiting a tunnel segment, it first has to check whether there is enough space inside the next tunnel segment. The process is done the same way as entering the tunnel segment. This segment uses the function **tokenAnyColor** to fetch a token with the color **True**. It will then check if there is available space in the next segment and remove a value from the list of vehicle length. Before firing the token, the transition will run the function **ComputeDistance** to compute using the time from the color list. In the last segment, there is no need to check the road tunnels list or compute the distance.

4.6 Main simulation file

The MSF is where **GPenSIM** runs the simulates. The MSF defined the PDFs that will be used in the simulation together with the initial dynamics. In this program, the initial dynamics that are defined are the transition time and priorities. The time value can be defined in the **init.m** file. The transitions with the highest priority are the exit point of each module. There is a race condition between the computation and the exit point, but the exit point only takes tokens with the **True** tag. Allowing the exit point to take the token first will prevent the recomputation of an already completed token.

At the end of the MSF, the function **plotp** triggers after the simulation. This function plot the number of tokens at the different places defined in the functions over time. This function checks the number of tokens inside each tunnel segment throughout the simulation.

Chapter 5

Simulation

In this chapter, there will be three different simulations using four tunnel segments. This segment will focus on the simulations' parameters while also predicting the expected outcome of this simulation.

5.1 Initial parameter

For these simulations, the parameters can be found in table 5.1 .

Parameter	**value**
Simulation time	10min
Arrival Frequency	4s
Arrival speed	60km/h
Acceleration	3km/h
Tunnel length	800m

Table 5.1: Parameters used in the simulations

5.2 Constant speed

In this simulation, all the tunnel segments will have a speed limit of 60km/h. There will no change of speed during simulation. The time a vehicle would use inside the tunnel segment would be $\frac{200m}{60km/h} = 12sec$. If the vehicle arrives at an interval of 4sec, then the number of vehicles inside a segment should be four.

5.3 Different speed

In this simulation, the speed limit of both segments A and B will have a speed limit of 60km/h, while segments C and D will have a speed limit of 80km/h. In this simulation, the expected outcome would be identical to simulation from 5.2 for both segments A and B. For segments C and D, the number of vehicles should be less due to the vehicles traveling faster. As the vehicle starts accelerating in segment C, the number of vehicles should be less in segment D as it should reach a speed close to or equal to the speed limit.

5.4 Constant speed with event

This simulation uses the same parameters as 5.2 but will generate a vehicle with an event. This event will cause the vehicle to reduce its speed to 30km/h, but it will only start reducing after traveling 100m. This event will generate 5min in the simulation. The expected outcome should be similar to 5.2 before the 5min mark, but also after the event vehicle leaves the tunnel. The vehicle inside will start accelerating again, and it will start to normalize itself.

Chapter 6

Result and discussion

This section will focus on discussing the simulation output while also discussing the further improvement of this program.

6.1 Simulation result

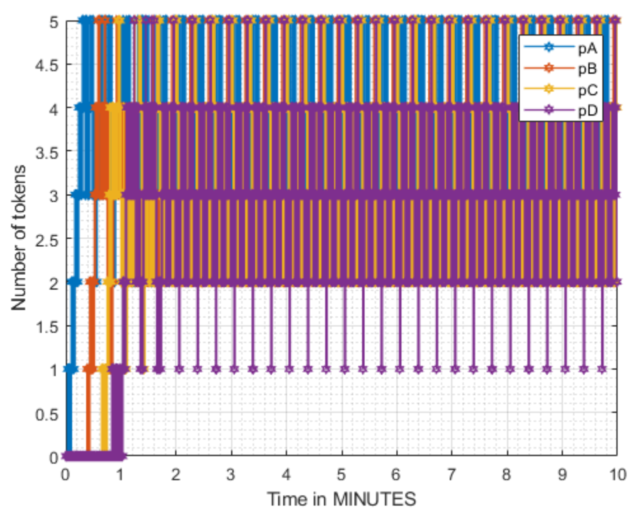

Figure 6.1: Simulation with constant speed

Figure 6.1 shows the output of the simulation. This graph displays the number of tokens at that time in each tunnel segment. Based on the result, the number of vehicles inside the tunnel segment is consistent due to no specific change. One of the predictions was that the number of tokens in a segment should be four, but it is sometimes five from the output. The reason for this is the way the computation transition operates. The transition can only compute one vehicle at a time, but as the number of vehicles increases in a segment, it takes longer to recompute the exact vehicle. This causes a higher number of vehicles inside due to not computing the vehicle that is supposed to exit.

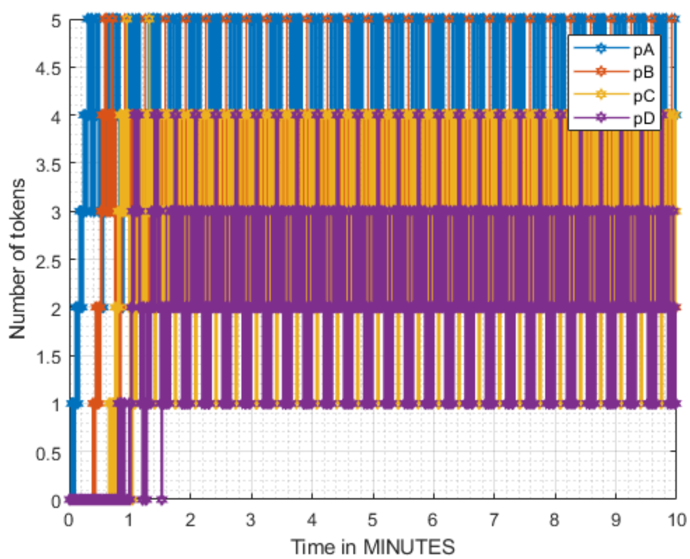

Figure 6.2: Simulation with different speed

Figure 6.2 shows the output of simulation with different speeds. As observed, the graph is identical to 6.1 for both segments A and B. For segment C, the number of vehicles is lower, and segment D is even lower. In comparison, there are instances where the number fluctuates higher between 3-4 for both segments. The fluctuation is due to the method of computation mentioned previously. There is less fluctuation at segment D by observing these fluctuations, where the value is closer to three. These changes in fluctuation are due to the constant speed of 80km/h in segment D, while at segment C, it accelerates from 60km/h to 80km/h.

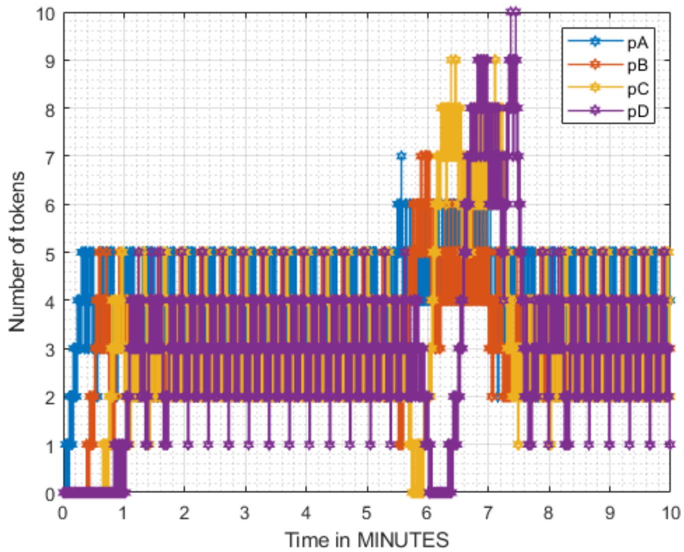

Figure 6.3: Simulation with constant speed, including an event

In the last simulation, the parameters were the same as the first, but with an event at five minutes. Figure 6.3 show the same output as figure 6.1 between 0-5 minute. Afterward, the number of tokens increases in segment A, while the other segment has fewer vehicles in the segment. The changes in the graph are due to vehicles in front of the event vehicle will drive based on the speed limit, while the vehicle behind will start to decelerate. As the event vehicle entering a new segment, the number of vehicles increases for the previous segment. The increase of vehicles in the segment will continue until the event vehicle exit the tunnel. After exiting, the tunnel traffic starts to normalize itself. Segment A has a lower number of vehicles during congestion due to vehicles entering with a speed of 60km/h. These vehicles will start to reduce the speed upon entering the segment.

6.2 Discussion

By observing the simulation output, it works as it is supposed to do. However, there is the problem of showing a higher number of vehicles inside the segments. The problem is caused by having too many vehicles inside the segment at a time. The solution is to reduce the number of vehicles in a segment. One solution would be to increase the segment's speed limit, which would speed up the time a vehicle has to spend inside, but this is not a viable solution. The program simulates a tunnel, and the speed limit cannot be changed just for the simulation. Another solution is to reduce the length of the tunnel segment. Instead of a tunnel segment of 200m, increase the number of segments to four with a length of 50m. Thus, the simulation becomes more accurate at the cost of memory and speed. As the segment increases, the simulation speed decreases, but since this is a program where information is essential, the speed is not an issue.

6.3 Further work

6.3.1 Extend to multiple lanes

This program can currently only simulate one lane, but most tunnels usually have more than one lane in the same direction. One way of implementing this is to have multiple single lanes and use a transition to connect between the lanes. The transition can be connected via the buffers to determined whether to change lanes. Another extension would be to include lanes in opposite directions.

6.3.2 Multiple events

This program only has one event, which is to reduce the speed of the vehicle. One improvement would be to include other types of events. For example, one event would be to increases the speed of the vehicles. This event would be easy to implement when there are multiple lanes in a tunnel. Another event would be a complete stop, which would be possible when vehicles are somehow aware of each other if there is more than one lane in the opposite direction, maybe an event to change direction in the middle of the tunnel.

6.3.3 Non-blind drivers

One problem of this program is the blind driver assumption, which restricts some implementation in the program. One proposed solution would be to store the distance of each vehicle in a list and update the list whenever computed. Using this list would allow us to determine some action. A further improvement would be implementing a simple prediction function to compare the distance between vehicles.

6.3.4 More user friendly

Even though there was an attempt to simplify the implementation for others to use, a possible improvement would be to make it easier to deploy. Another improvement would be to simplify the variables in **init.m** file. For example, speed limit and tunnel length could be simplified using a structure.

Chapter 7

Conclusion

This thesis introduces a program that simulates the traffic inside a one-lane tunnel. As of now, the program only presents an event that slows down vehicles. There were also done three simulations with different scenarios, where one of them used the event. Comparing the implementation with the result from the simulation, it operates as expected with the exemption of showing higher vehicle numbers in the segments due to slow computation with an increased number of tokens. The high number of tokens causes a less accurate simulation but could be solved by reducing the tunnel segment's length and increasing the number of tunnel segments. The program requires further work and expansion before it is usable as a simulation tool.

Bibliography

[1] Andrea Bobbio, Marco Gribaudo, and Andras Horvath. Modelling a car safety controller in road tunnels using hybrid petri nets. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1436–1441. IEEE, 2006.

[2] Ciro Caliendo, Maria Luisa De Guglielmo, and Maurizio Guida. A crash-prediction model for road tunnels. *Accident Analysis & Prevention*, 55:107–115, 2013.

[3] Reggie Davidrajuh. Gpensim: A new Petri net simulator. In *Petri Nets Applications*. IntechOpen, 2010.

[4] Reggie Davidrajuh. *Modeling Discrete-Event Systems with GPenSIM*. Springer International Publishing, Cham, 2018.

[5] Reggie Davidrajuh. A new modular Petri net for modeling large discrete-event systems: A proposal based on the literature study. *Computers*, 8(4):83, 2019.

[6] Reggie Davidrajuh. Extracting petri modules from large and legacy petri net models. *IEEE Access*, 8:156539–156556, 2020.

[7] engin akyurt. `https://unsplash.com/photos/iNlNrMjdI_Q`, 2018.

[8] GPenSIM. General-purpose Petri net simulator. Technical report, http://www.davidrajuh.net/gpensim, 2019. accessed on 20 July 2020.

[9] Youness Riouali, Laila Benhlima, and Slimane Bah. Petri net extension for traffic road modelling. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–6. IEEE, 2016.

[10] Nico Vandaele, Tom Van Woensel, and Aviel Verbruggen. A queueing based traffic flow model. *Transportation Research Part D: Transport and Environment*, 5(2):121–135, 2000.

Appendix A

User manual

When running this program, there are two files to take into consideration. The first file is the **MSF.m** file, which is where to run the simulation. The second file is the **init.m** file. This is where to change the parameters for running the program.

The parameters in the **init.m** file have a comment which describes the use for each parameter. Some parameters are specified with capital letters not to change.

The **MSF.m** file is where to run the simulation. When running the program, make sure the current location is the **MSF.m** file before running the program.

Appendix B

Program code

Algorithm B.1: init.m

```
global global_info;


% --------------- Time information --------------- %

global_info.ComputationTime = 1;     % The time for computation
global_info.TransitionTime = 1;      % The time of ...
    transition between segments


global_info.SimHour = 0;         % Simulation time in hours
global_info.SimMin = 10;         % Simulation time in minutes
global_info.SimSec = 0;          % Simulation time in seconds

% --------------- Arrival and Event --------------- %

TokenFreq = 4;       % Frequency of tokens arrival

tokenFinalArrival = (global_info.SimHour*3600) + ...
    (global_info.SimMin*60) + global_info.SimSec; %End ...
    time of Arrival

global_info.Events = [];         % Event during simulation ...
    DON'T CHANGE

```

```
global_info.Arrival = 0:TokenFreq:tokenFinalArrival; ...
                % Arrival list
global_info.EventArrival = [300 320]; ...
                                % Event Arrval list
global_info.EventType = [{'Slow 100 30'} {'Slow 100 30'}]; ...
         % Event type
% {'Slow 0 40'}

% --------------- Troubleshooting --------------- %

%Set as one to display the output of the computation
global_info.DisplayA = 0;
global_info.DisplayB = 0;
global_info.DisplayC = 0;
global_info.DisplayD = 0;

% --------------- Tunnel parameter --------------- %

% Tunnel length of each segment
global_info.RoadLengthA = 200;
global_info.RoadLengthB = 200;
global_info.RoadLengthC = 200;
global_info.RoadLengthD = 200;

% Speed limit of each segment
global_info.speedLimitA = 60;
global_info.speedLimitB = 60;
global_info.speedLimitC = 80;
global_info.speedLimitD = 80;

% Vehicle length for each segment DON'T CHANGE
global_info.TotalVLenA = [];
global_info.TotalVLenB = [];
global_info.TotalVLenC = [];
global_info.TotalVLenD = [];


% --------------- Vechile parameter --------------- %

global_info.Acceleration = 3;       % Global acceleration
global_info.Speed = 60;             % Initial speed
```

Algorithm B.2: MSF.m

```
clear; clc;


global global_info;
run("init");        % Running the init.m file


% Simulation time
global_info.START_AT = [0, 0, 0];
global_info.STOP_AT = [global_info.SimHour, ...
    global_info.SimMin, global_info.SimSec];


% Initial dynamics
pns = ...
    pnstruct({'SegmentA_pdf','SegmentB_pdf','SegmentC_pdf',...
'SegmentD_pdf','Connect_pdf'});
dyn.ft = ...
    {'tComputeA',global_info.ComputationTime,'tComputeA',...
global_info.ComputationTime,'tComputeA',...
global_info.ComputationTime,'tComputeA',...
global_info.ComputationTime,'tAEnter',1,'allothers',...
global_info.TransitionTime};

dyn.ip = {'tAExit',1,'tBExit',1,'tCExit',1,'tDExit',1}; % ...
    Priority

pni = initialdynamics(pns,dyn);

sim = gpensim(pni);
plotp(sim, {'pA', 'pB', 'pC', 'pD'});  % Plotting the places
```

Algorithm B.3: segmentA_pdf.m

```
function [png] = SegmentA_pdf()

png.PN_name = 'SegmentA';
png.set_of_Ps = {'pA'};
png.set_of_Ts = {'tAEnter','tAExit','tComputeA'};
png.set_of_As = ...
    {'tAEnter','pA',1,'pA','tAExit',1,'pA','tComputeA',1,...
     'tComputeA','pA',1};
png.set_of_Ports = {'tAEnter','tAExit'};
```

Algorithm B.4: segmentB_pdf.m

```
function [png] = SegmentB_pdf()

png.PN_name = 'SegmentB';
png.set_of_Ps = {'pB'};
png.set_of_Ts = {'tBEnter','tBExit','tComputeB'};
png.set_of_As = ...
    {'tBEnter','pB',1,'pB','tBExit',1,'pB','tComputeB',1,...
     'tComputeB','pB',1};
png.set_of_Ports = {'tBEnter','tBExit'};
```

Algorithm B.5: segmentC_pdf.m

```
function [png] = SegmentC_pdf()

png.PN_name = 'SegmentC';
png.set_of_Ps = {'pC'};
png.set_of_Ts = {'tCEnter','tCExit','tComputeC'};
png.set_of_As = ...
    {'tCEnter','pC',1,'pC','tCExit',1,'pC','tComputeC',1,...
     'tComputeC','pC',1};
png.set_of_Ports = {'tCEnter','tCExit'};
```

Algorithm B.6: segmentD_pdf.m

```
function [png] = SegmentD_pdf()

png.PN_name = 'SegmentD';
png.set_of_Ps = {'pD'};
png.set_of_Ts = {'tDEnter','tDExit','tComputeD'};
png.set_of_As = ...
    {'tDEnter','pD',1,'pD','tDExit',1,'pD','tComputeD',1,...
    'tComputeD','pD',1};
png.set_of_Ports = {'tDEnter','tDExit'};
```

Algorithm B.7: Connect_pdf.m

```
function [png] = Connect_pdf()

png.PN_name = 'Tunnel System';
png.set_of_Ps = ...
    {'pOutside1','pOutside2','pBuffer1','pBuffer2','pBuffer3'};
png.set_of_Ts = {'tGenerator'};
png.set_of_As = ...
    {'tGenerator','pOutside1',1,'pOutside1','tAEnter',1,...
    'tAExit','pBuffer1',1,'pBuffer1','tBEnter',1,'tBExit',...
    'pBuffer2',1, ...
        'pBuffer2','tCEnter',1,'tCExit','pBuffer3',1,...
    'pBuffer3','tDEnter',1,'tDExit','pOutside2',1};
```

Algorithm B.8: MOD_SegmentA_PRE.m

```
function [fire, trans] = MOD_SegmentA_PRE(trans)
global global_info;
switch trans.name
    case 'tComputeA'
        tok = tokenArrivedEarly('pA',1);    % Fetch the ...
            earliest token

        % Get the token color and time
        col = get_color('pA',tok);
        tokTime = get_tokCT('pA',tok);

        % System time
        cTime = current_time();

        % The waiting time of the token
        t = cTime - tokTime;

        % If waiting time is less than one, then make it one
        if t < 1
            t = 1;
        end

        % Compute the parameters and create a new color list
        newcol = ComputeSegment(col, ...
            global_info.speedLimitA, t, tokTime);

        % Display the content of the new color list
        if global_info.DisplayA == 1
            disp("Segment A");
            disp(newcol);
        end

        % Adding the color to token and fire it
        trans.override = 1;
        trans.new_color = newcol;
        trans.selected_tokens = tok;
        fire = tok;
    case 'tAEnter'
        tok = tokenArrivedEarly('pOutside1',1);     % ...
            Fetch the earliest token

        % Get the color
        col = get_color('pOutside1', tok);

        % Fetch the vehicle length
        vLen = FindVLength(col);

```

```
        % The total number of vehicle currently in the segment
        numOfV = length(global_info.TotalVLenA);

        % The sum of all the vehicle length in the segment
        S = sum(global_info.TotalVLenA) + numOfV;

        % If the length of all the vehicle pluss itself is ...
            less than tunnel
        % segment, then proceed
        if (S+vLen+1) ≤ global_info.RoadLengthA

            %Add the vehicle to list
            global_info.TotalVLenA(numOfV + 1) = vLen;

            % Add the value current length
            cur = Element2Char('CurDist', ...
                global_info.RoadLengthA);

            % Adding the color to token and fire it
            trans.new_color = cur;
            trans.selected_tokens = tok;
            fire = tok;
        else
            % Don't fire if segment is full
            fire = 0;
        end
    case 'tAExit'
        % Fetch the token with the True flag
        tok = tokenAnyColor('pA',1,{'True'});
        if tok == 0
            % Don't fire, if it pick a token with zero id
            fire = 0;
        else

            % Get the colors and fetch all the values
            col = get_color('pA',tok);
            [dist, speed, rem, overTime, vLen, event] = ...
                FindElement(col);

            %  Total number of vehicle in the next segment
            numOfV = length(global_info.TotalVLenB);

            % The total vehicle length in the next segment
            S = sum(global_info.TotalVLenB) + numOfV;

            % If the length of all the vehicle pluss ...
                itself is less than
            % next tunnel segment, then proceed
            if (S+vLen+1) ≤ global_info.RoadLengthB
```

```

                % Remove the vehicle length from the ...
                    current segment
                pos = RemoveVLen(global_info.TotalVLenA, ...
                    vLen);
                global_info.TotalVLenA(pos) = [];

                % Update and transform the parameters to ...
                    character
                distChar = Element2Char('Dist',dist);
                speedChar = Element2Char('Speed',speed);
                remChar = Element2Char('CurDist',rem);
                tChar = Element2Char('Time',(overTime + ...
                    global_info.TransitionTime*2));
                vLenChar = Element2Char('VLen',vLen);

                % The new color list
                newcol = {distChar,speedChar,remChar, ...
                    tChar, vLenChar};

                % If vehicle has an event, include it into ...
                    new color list
                if event ≠ 0
                    newcol{end + 1} = event;
                end

                % Adding the color to token and fire it
                trans.override = 1;
                trans.selected_tokens = tok;
                trans.new_color = newcol;
                fire = tok;
            else
                fire = 0;
            end
        end
    otherwise
        disp(trans.name);
        fire= 1;
end
```

Algorithm B.9: MOD_SegmentB_PRE.m

```
function [fire, trans] = MOD_SegmentB_PRE(trans)
global global_info;
switch trans.name
    case 'tComputeB'
       tok = tokenArrivedEarly('pB',1);      % Fetch the ...
           earliest token

       % Get the token color and time
       col = get_color('pB',tok);
       tokTime = get_tokCT('pB',tok);

       % System time
       cTime = current_time();

       % The waiting time of the token
       t = cTime - tokTime;

       % If waiting time is less than one, then make it one
        if t < 1
            t = 1;
        end

        % Compute the parameters and create a new color list
        newcol = ComputeSegment(col, ...
            global_info.speedLimitB, t, tokTime);

        % Display the content of the new color list
        if global_info.DisplayB == 1
            disp("Segment B");
            disp(newcol);
        end

        % Adding the color to token and fire it
        trans.override = 1;
        trans.new_color = newcol;
        trans.selected_tokens = tok;
        fire = tok;
    case 'tBEnter'
        tok = tokenArrivedEarly('pBuffer1',1);      % ...
            Fetch the earliest token

        % Get the color and time
        col = get_color('pBuffer1', tok);
        tokTime = get_tokCT('pBuffer1',tok);

        % Find all the elements in the color list
        [distVal, speedVal, curDist, t, vLen, event] = ...
```

```
            FindElement(col);

        % The total number of vehicle currently in the segment
        numOfV = length(global_info.TotalVLenB);

         % The sum of all the vehicle length in the segment
        S = sum(global_info.TotalVLenB) + numOfV;

        % If the length of all the vehicle pluss itself is ...
            less than tunnel
        % segment, then proceed
        if (S+vLen+1) ≤ global_info.RoadLengthB

            %Add the vehicle to list
            global_info.TotalVLenB(numOfV + 1) = vLen;

            % Update the current distance variable
            newCur = curDist + global_info.RoadLengthB;

            % Compute the variable and transform them to ...
                character array
            [FinDist, FinRem, FinSpeed, remT ] = ...
                ComputeDistance(distVal, newCur, speedVal, ...
                global_info.speedLimitB, t,tokTime, event);
            distChar = Element2Char('Dist',FinDist);
            curDistChar = Element2Char('CurDist',FinRem);
            speedChar = Element2Char('Speed',FinSpeed);
            timeChar = Element2Char('Time',remT);
            lenChar = Element2Char('VLen',vLen);

            % New color list
            newCol = {distChar, speedChar, curDistChar, ...
                lenChar, timeChar};

            % If vehicle has the event, add it to the new ...
                color list
            if event ≠ 0
                newCol{end + 1} = event;
            end

            % Adding the color to token and fire it
            trans.override = 1;
            trans.selected_tokens = tok;
            trans.new_color = newCol;
            fire = tok;
        else
            % Don't fire if segment is full
            fire = 0;
        end
```

```
    case 'tBExit'
        % Fetch the token with the True flag
        tok = tokenAnyColor('pB',1,{'True'});
        if tok == 0
             % Don't fire, if it pick a token with zero id
            fire = 0;
        else

            % Get the colors and fetch all the values
            col = get_color('pB',tok);
            [dist, speed, rem, overTime, vLen, event] = ...
                FindElement(col);

            %  Total number of vehicle in the next segment
            numOfV = length(global_info.TotalVLenC);

            % The total vehicle length in the next segment
            S = sum(global_info.TotalVLenC) + numOfV;

            % If the length of all the vehicle pluss ...
                itself is less than
            % next tunnel segment, then proceed
            if (S+vLen+1) ≤ global_info.RoadLengthC

                % Remove the vehicle length from the ...
                    current segment
                pos = RemoveVLen(global_info.TotalVLenB, ...
                    vLen);
                global_info.TotalVLenB(pos) = [];

                % Update and transform the parameters to ...
                    character
                distChar = Element2Char('Dist',dist);
                speedChar = Element2Char('Speed',speed);
                remChar = Element2Char('CurDist',rem);
                tChar = Element2Char('Time',(overTime + ...
                    global_info.TransitionTime*2));
                vLenChar = Element2Char('VLen',vLen);

                % The new color list
                newcol = {distChar,speedChar,remChar, ...
                    tChar, vLenChar};

                % If vehicle has an event, include it into ...
                    new color list
                if event ≠ 0
                    newcol{end + 1} = event;
                end

```

```
                % Adding the color to token and fire it
                trans.override = 1;
                trans.selected_tokens = tok;
                trans.new_color = newcol;
                fire = tok;
            else
                fire = 0;
            end
        end
    otherwise
        disp(trans.name);
        fire= 1;
end
```

Algorithm B.10: MOD_SegmentC_PRE.m

```
function [fire, trans] = MOD_SegmentC_PRE(trans)
global global_info;
switch trans.name
    case 'tComputeC'
       tok = tokenArrivedEarly('pC',1);     % Fetch the ...
           earliest token

       % Get the token color and time
       col = get_color('pC',tok);
       tokTime = get_tokCT('pC',tok);

       % System time
       cTime = current_time();

       % The waiting time of the token
       t = cTime - tokTime;

       % If waiting time is less than one, then make it one
        if t < 1
            t = 1;
        end

        % Compute the parameters and create a new color list
        newcol = ComputeSegment(col, ...
            global_info.speedLimitC, t, tokTime);

        % Display the content of the new color list
        if global_info.DisplayC == 1
            disp("Segment C");
            disp(newcol);
        end

        % Adding the color to token and fire it
        trans.override = 1;
        trans.new_color = newcol;
        trans.selected_tokens = tok;
        fire = tok;
    case 'tCEnter'
        tok = tokenArrivedEarly('pBuffer2',1);      % ...
            Fetch the earliest token

        % Get the color and time
        col = get_color('pBuffer2', tok);
        tokTime = get_tokCT('pBuffer2',tok);

        % Find all the elements in the color list
        [distVal, speedVal, curDist, t, vLen, event] = ...
```

```
            FindElement(col);

        % The total number of vehicle currently in the segment
        numOfV = length(global_info.TotalVLenC);

         % The sum of all the vehicle length in the segment
        S = sum(global_info.TotalVLenC) + numOfV;

        % If the length of all the vehicle pluss itself is ...
            less than tunnel
        % segment, then proceed
        if (S+vLen+1) ≤ global_info.RoadLengthC

            %Add the vehicle to list
            global_info.TotalVLenC(numOfV + 1) = vLen;

            % Update the current distance variable
            newCur = curDist + global_info.RoadLengthB;

            % Compute the variable and transform them to ...
                character array
            [FinDist, FinRem, FinSpeed, remT ] = ...
                ComputeDistance(distVal, newCur, speedVal, ...
                global_info.speedLimitC, t,tokTime, event);
            distChar = Element2Char('Dist',FinDist);
            curDistChar = Element2Char('CurDist',FinRem);
            speedChar = Element2Char('Speed',FinSpeed);
            timeChar = Element2Char('Time',remT);
            lenChar = Element2Char('VLen',vLen);

            % New color list
            newCol = {distChar, speedChar, curDistChar, ...
                lenChar, timeChar};

             % If vehicle has the event, add it to the new ...
                 color list
            if event ≠ 0
                newCol{end + 1} = event;
            end

            % Adding the color to token and fire it
            trans.override = 1;
            trans.selected_tokens = tok;
            trans.new_color = newCol;
            fire = tok;
        else
            % Don't fire if segment is full
            fire = 0;
        end
```

```
    case 'tCExit'
        % Fetch the token with the True flag
        tok = tokenAnyColor('pC',1,{'True'});
       if tok == 0
            % Don't fire, if it pick a token with zero id
            fire = 0;
       else

           % Get the colors and fetch all the values
            col = get_color('pC',tok);
            [dist, speed, rem, overTime, vLen, event] = ...
                FindElement(col);

            %  Total number of vehicle in the next segment
            numOfV = length(global_info.TotalVLenD);

            % The total vehicle length in the next segment
            S = sum(global_info.TotalVLenD) + numOfV;

            % If the length of all the vehicle pluss ...
                itself is less than
            % next tunnel segment, then proceed
            if (S+vLen+1) ≤ global_info.RoadLengthD

                % Remove the vehicle length from the ...
                    current segment
                pos = RemoveVLen(global_info.TotalVLenC, ...
                    vLen);
                global_info.TotalVLenC(pos) = [];

                % Update and transform the parameters to ...
                    character
                distChar = Element2Char('Dist',dist);
                speedChar = Element2Char('Speed',speed);
                remChar = Element2Char('CurDist',rem);
                tChar = Element2Char('Time',(overTime + ...
                    global_info.TransitionTime*2));
                vLenChar = Element2Char('VLen',vLen);

                % The new color list
                newcol = {distChar,speedChar,remChar, ...
                    tChar, vLenChar};

                % If vehicle has an event, include it into ...
                    new color list
                if event ≠ 0
                    newcol{end + 1} = event;
                end

```

```
                    % Adding the color to token and fire it
                    trans.override = 1;
                    trans.selected_tokens = tok;
                    trans.new_color = newcol;
                    fire = tok;
                else
                    fire = 0;
                end
            end
        otherwise
            disp(trans.name);
            fire= 1;
end
```

Algorithm B.11: MOD_SegmentD_PRE.m

```
function [fire, trans] = MOD_SegmentD_PRE(trans)
global global_info;
switch trans.name
    case 'tComputeD'
        tok = tokenArrivedEarly('pD',1);        % Fetch ...
            the earliest token

        % Get the token color and time
        col = get_color('pD',tok);
        tokTime = get_tokCT('pD',tok);

        % System time
        cTime = current_time();

        % The waiting time of the token
        t = cTime - tokTime;

        % If waiting time is less than one, then make it one
        if t < 1
            t = 1;
        end

        % Compute the parameters and create a new color list
        newcol = ComputeSegment(col, ...
            global_info.speedLimitD, t, tokTime);

        % Display the content of the new color list
        if global_info.DisplayD == 1
            disp("Segment D");
            disp(newcol);
        end

        % Adding the color to token and fire it
        trans.override = 1;
        trans.new_color = newcol;
        trans.selected_tokens = tok;
        fire = tok;
    case 'tDEnter'
        tok = tokenArrivedEarly('pBuffer3',1);      % ...
            Fetch the earliest token

        % Get the color and time
        col = get_color('pBuffer3', tok);
        tokTime = get_tokCT('pBuffer3',tok);

        % Find all the elements in the color list
        [distVal, speedVal, curDist, t, vLen, event] = ...
```

```
            FindElement(col);

        % The total number of vehicle currently in the segment
        numOfV = length(global_info.TotalVLenD);

         % The sum of all the vehicle length in the segment
        S = sum(global_info.TotalVLenD) + numOfV;

        % If the length of all the vehicle pluss itself is ...
            less than tunnel
        % segment, then proceed
        if (S+vLen+1) ≤ global_info.RoadLengthD

            %Add the vehicle to list
            global_info.TotalVLenD(numOfV + 1) = vLen;

            % Update the current distance variable
            newCur = curDist + global_info.RoadLengthB;

            % Compute the variable and transform them to ...
                character array
            [FinDist, FinRem, FinSpeed, remT ] = ...
                ComputeDistance(distVal, newCur, speedVal, ...
                global_info.speedLimitD, t,tokTime, event);
            distChar = Element2Char('Dist',FinDist);
            curDistChar = Element2Char('CurDist',FinRem);
            speedChar = Element2Char('Speed',FinSpeed);
            timeChar = Element2Char('Time',remT);
            lenChar = Element2Char('VLen',vLen);

            % New color list
            newCol = {distChar, speedChar, curDistChar, ...
                lenChar, timeChar};

             % If vehicle has the event, add it to the new ...
                 color list
            if event ≠ 0
                newCol{end + 1} = event;
            end

            % Adding the color to token and fire it
            trans.override = 1;
            trans.selected_tokens = tok;
            trans.new_color = newCol;
            fire = tok;
        else
            % Don't fire if segment is full
            fire = 0;
        end
```

```
    case 'tDExit'
        % Fetch the token with the True flag
        tok = tokenAnyColor('pD',1,{'True'});
       if tok == 0
            % Don't fire, if it pick a token with zero id
            fire = 0;
       else

           % Get the colors and fetch all the values
            col = get_color('pD',tok);
            [dist, speed, rem, overTime, vLen, event] = ...
                FindElement(col);

            % Remove the vehicle length from the current ...
                segment
            pos = RemoveVLen(global_info.TotalVLenD, vLen);
            global_info.TotalVLenD(pos) = [];

            % Update and transform the parameters to character
            distChar = Element2Char('Dist',dist);
            speedChar = Element2Char('Speed',speed);
            remChar = Element2Char('CurDist',rem);
            tChar = Element2Char('Time',(overTime + ...
                global_info.TransitionTime*2));
            vLenChar = Element2Char('VLen',vLen);

            % The new color list
            newcol = {distChar,speedChar,remChar, tChar, ...
                vLenChar};

            % If vehicle has an event, Add the time when ...
                it left the tunnel
            if event ≠ 0
                RemoveEvent(event);
            end

            % Adding the color to token and fire it
            trans.override = 1;
            trans.selected_tokens = tok;
            trans.new_color = newcol;
            fire = tok;
        end
    otherwise
        disp(trans.name);
        fire= 1;
end
```

Algorithm B.12: tGenerator_pre.m

```
function [fire, trans] = tGenerator_pre(trans)
    global global_info;

    % System time
    cTime = current_time();

    % Generate a vehicle length and transform to character ...
        array
    vLen = VLengthGenerator();
    vLenChar = Element2Char('VLen', vLen);

    % Transform the speed into charachter array
    Speed = Element2Char('Speed', global_info.Speed);

    % Store the vehicle length adn speed in color list
    col = {'Dist 0', Speed, vLenChar};

    % check if any vehicle and event greater than system time
    if isempty(global_info.EventArrival) == 0 && cTime ≥ ...
        global_info.EventArrival(1)

        % fetch the event time and remove from list
        b = global_info.EventArrival(1);
        global_info.EventArrival(1) = [];

        % if event is equal to non event time, remove ...
            non-event time
        if isempty(global_info.Arrival) == 0 && b == ...
            global_info.Arrival(1)
            global_info.Arrival(1) = [];
        end

        % Add the event to the list of color
        col{end + 1} = global_info.EventType{1};

        % Remove the event
        global_info.EventType(1) = [];

        % Append the color to the token and fire
        trans.new_color = col;
        fire = 1;

    % If the arrival list is not empty and it's greater ...
        than the time
    elseif isempty(global_info.Arrival) == 0 && cTime ≥ ...
        global_info.Arrival(1)

```

```
        % Remove the time and fire the token
        global_info.Arrival(1) = [];
        trans.new_color = col;
        fire = 1;
    else
        % If not time yet, don't fire
        fire = 0;
    end
end
```

Algorithm B.13: ComputeDistance.m

```
function [FinDist, FinRem, FinSpeed, remT] = ...
    ComputeDistance(dist, curDist, speed, speedL, time, ...
    VLife , event)
global global_info;

    % Initial values
    newDist = dist;
    FinDist = dist;
    FinSpeed = speed;
    rem = curDist;
    FinRem = curDist;
    remT = 0;
    vL = VLife;


    for i = 1:time

        % Check if event is valid
        ok = FindEvent(FinDist, vL);

        % If invalid, compute using speed limit, else ...
            compute using event
        if ok == 0
            [travelDistance, FinSpeed] = traveling(speedL, ...
                speed, global_info.Acceleration);
        else
            %disp(global_info.Events(1).Dist);
            [travelDistance, FinSpeed] = ...
                traveling(global_info.Events(ok).Speed, ...
                speed, global_info.Acceleration);
        end

        % Update temporary distance travel and temporary ...
            remaining distance
        newDist = newDist + travelDistance;
        rem = rem - travelDistance;

        % if vehicle update lead to leaving the segment
        if rem ≤ 0 && FinRem > 0

            % Update the distance travel and remainng distance
            FinDist = newDist;
            FinRem = rem;

        % If remaining is zero or less
        elseif rem ≤ 0

```

```
            % Increment the extra time value
            remT = remT + 1;
        else
            % permanent distance and remaing is eual to temp
            FinDist = newDist;
            FinRem = rem;
        end

        % if event is found, send it to event handler
        if event ≠ 0
            EventHandler(event, FinDist);
        end

        % Used for determinng event validity after event ...
            vehicle leave the
        % tunnel
        vL = vL + 1;

    end
end


function [ travelDistance, FinSpeed ] = ...
    traveling(maxSpeed,currentSpeed, acc)

        % Difference of allowed speed with vehicle speed
        diff = maxSpeed - currentSpeed;
        FinSpeed = currentSpeed;

        % if there is no speed change, compute using ...
            constant speed
        if abs(diff) < 0.05
            travelDistance = (FinSpeed / 3.6);
        % if the speed change is between [-acc, acc], use ...
            the diff as acc
        elseif diff < acc && diff > - acc
            travelDistance = (FinSpeed / 3.6) + (1/2) * diff;
            FinSpeed = FinSpeed + diff;
        else
            % if positive is negative, accelerate
            if diff > 0
               travelDistance = (FinSpeed / 3.6) + (1/2) * ...
                   acc;
               FinSpeed = FinSpeed + acc;
           % if diff is negative, decelerate
           else
               travelDistance = (FinSpeed / 3.6) - (1/2) * ...
                   acc;
               FinSpeed = FinSpeed - acc;
```

```
            end
        end
end


function ok = FindEvent(dist, VLife)
global global_info;

% Initial parameters
minDist = dist;
minSpeed = 999;
test = 0;
ok = 0;

% if no event, return
if isempty(global_info.Events) == 1
    return;
end


for i = 1:length(global_info.Events)

    % If the vehicle time is less or equal to event end time
    if global_info.Events(i).End == 0 || ...
        global_info.Events(i).End > VLife

        % If the vehicle distance is less than event distance
        if minDist ≤ global_info.Events(i).Dist

            % If the event speed is less than current ...
                eveny speed
            if minSpeed ≥ global_info.Events(i).Speed
                minDist = global_info.Events(i).Dist;
                minSpeed = global_info.Events(i).Speed;
                test = i;
            end
        end
    end
end
ok = test;
end
```

Algorithm B.14: ComputeSegment.m

```
function newcol =  ComputeSegment(col, speedLimit, t, tokTime)

% Find the elements in the color list
[dist, speed, currDist, overTime, vLen, event] = ...
    FindElement(col);

% Compute the parameters distance, remaing distance, speed ...
    and extra time
[newDist, rem, newSpeed, remT ] = ComputeDistance(dist, ...
    currDist, speed, speedLimit, t,tokTime, event);

% Transform the values into character array
distChar = Element2Char('Dist', newDist);
curDistChar = Element2Char('CurDist', rem);
speedChar = Element2Char('Speed', newSpeed);
vLenChar = Element2Char('VLen',vLen);
timeChar = Element2Char('Time',overTime +remT);

% New color list
newcol = {distChar, speedChar,curDistChar, vLenChar,timeChar};

% Include the True value if remainig distance is zero or less
if rem ≤ 0
    newcol{end + 1} = 'True';
end

% Include the event, if the vehicle already had an event
if event ≠ 0
    newcol{end + 1} = event;
end

end
```

Algorithm B.15: Element2Char.m

```
function elem = Element2Char(text, val)
% transform the value to string
val2str = string(val);

% Concate the value and text
cat = strcat(text,{' '}, val2str);

% Transform the concated value to character array
elem = convertStringsToChars(cat);
end
```

Algorithm B.16: FindElement.m

```
function [dist, speed, rem, time, len, event] = ...
    FindElement(col)
event = 0;
time = 0;

% Loop through the color list
    for i = 1:length(col)
        split = strsplit(col{i});
        switch split{1}
            case 'Dist'
                dist = str2double(split(2));    % ...
                    Transform to number
            case 'Speed'
                speed = str2double(split(2));   % ...
                    Transform to number
            case 'CurDist'
                rem = str2double(split(2));     % ...
                    Transform to number
            case 'Time'
                time = str2double(split(2));    % ...
                    Transform to number
            case 'VLen'
                len = str2double(split(2));     % ...
                    Transform to number
            case 'True'
                continue;   % Continue
            otherwise
                event = col{i};     % Also return the value
        end
    end
end
```

Algorithm B.17: FindVLength.m

```
function len = FindVLength(col)

    % Loop throught the color
    for i = 1:length(col)
        split = strsplit(col{i});   % Split the color by space
        if strncmp('VLen',split(1),4)
            len = str2double(split(2)); % transform to number
            break;
        end
    end
end
```

Algorithm B.18: RemoveVLen.m

```
function pos = RemoveVLen(seg, vLen)
    % Loop through the vehicle list
    for i = 1:length(seg)

        % if vehicle match current vehicle length, return ...
            the position
        if seg(i) == vLen
            pos = i;
            break;
        end
    end
end
```

Algorithm B.19: VLengthGenerator.m

```
function len = VLengthGenerator()

% Generate a uniformly distributed random value from 0-1 using
p = rand;

% if less than 0.8, len is 3, else length is 5
if p < 0.8
    len = 3;
else
    len = 5;
end
end
```

Algorithm B.20: EventHandler.m

```
function EventHandler(event, dist)
global global_info;

% split the event by space
data = strsplit(event);

% The event distance
edist = str2double(data(2));

% The event speed
speed = str2double(data(3));

ok = 0;

% loop througt the list of events in use
for i = 1:length(global_info.Events)

    % check if current event is equal to any active event ...
        in the list
    if strcmp(event, global_info.Events(i).Code) && ...
        global_info.Events(i).End == 0
        ok = i;
        break;
    end
end

% If event is already in the list, update the event distance
if ok ≠ 0
    global_info.Events(ok).Dist = dist;

% otherwise add the event if vehicle distance is less than ...
    event distance
else
    if dist ≥ edist
        AddEvent(event, dist, speed);
    end
end


end

function AddEvent(event, dist, speed)
global global_info;

% Add the event to event list
global_info.Events(end + 1).Code = event;
global_info.Events(end).Dist = dist;
```

```
global_info.Events(end).Speed = speed;
global_info.Events(end).End = 0;
end
```

Algorithm B.21: RemoveEvent.m

```
function RemoveEvent(event)
global global_info;

% System time
cTime = current_time();

% find the current event in the event list
for i = 1:length(global_info.Events)

    % set the event end time as system time
    if strcmp(event, global_info.Events(i).Code)
        global_info.Events(i).End = cTime;
        break;
    end
end

end
```