*CSE 561: Modeling & Simulation Theory and Application*

**Amusement Park optimal ratio of fastpass buyers**

Jisoo Lee Ying Yu Wu

Computer Science PhD Student Computer Science Masters Student

School of Computing and Augmented Intelligence

Arizona State University, Tempe, AZ, USA, 85281

Due Date: 12-01-2023

**Contents**

1. Introduction

1.1 Objectives

1.2 Background

1. System Description

2.1 Configuration and Variables

1. Modeling Approach

3.1 Model Development

1. Visitor
2. Generator
3. Attraction
4. Transducer
5. Monitor
6. Simulation Experiments
7. Results and Evaluation
8. Conclusion
9. References

A Appendices

**Team Members Tasks (% effort is preliminary)**

| **List of tasks** | **Name1** | **% Effort** | **Name 2** | **% Effort** |
| --- | --- | --- | --- | --- |
| 1. Introduction | Jisoo | 80 | Ying Yu | 20 |
| 2. System description | Jisoo | 20 | Ying Yu | 80 |
| 3. Modeling approach | Jisoo | 50 | Ying Yu | 50 |
| A. Visitor | Jisoo | 50 | Ying Yu | 50 |
| B. Generator | Jisoo | 50 | Ying Yu | 50 |
| C. Attraction | Jisoo | 50 | Ying Yu | 50 |
| D. Transducer | Jisoo | 50 | Ying Yu | 50 |
| E. Monitor | Jisoo | 50 | Ying Yu | 50 |
| 4. Simulation Experiments | Jisoo | 20 | Ying Yu | 80 |
| 5. Results and Evaluation | Jisoo | 80 | Ying Yu | 20 |
| 6. Conclusion | Jisoo | 80 | Ying Yu | 20 |
| 7. References | Jisoo | 50 | Ying Yu | 50 |
| Appendices | Jisoo | 50 | Ying Yu | 50 |

Abstract

This project seeks to simulate the effectiveness of FastPass tickets in an amusement park using the DEVS-Suite Simulator (v6.1.0). The primary goal is to explore the correlation between the sales rate of FastPass tickets and the wait times experienced by both FastPass buyers and regular visitors. The simulation incorporates various factors, including the number of attractions, their durations, capacities, and visitor flow, which contribute to variations in waiting times for the two ticket types. The outcomes of this model will offer valuable insights to amusement park management, enabling them to make informed adjustments to ticket quantities or prices for effective control of the sales rate.

# 1 Introduction

# 1.1 Objective

The objective of this project is to design and simulate a model that elucidates the relationship between the sales rate of FastPass tickets, commonly introduced in places like amusement parks.. Specifically, the project aims to determine the optimal ratio of Fastpass ticket buyers to achieve an appropriate gap in waiting times between the Fastpass line and the normal line in an amusement park. The objective is to strike a balance that ensures efficient utilization of the Fastpass system, enhancing the overall visitor experience. By finding the optimal ratio, the goal is to minimize waiting times for Fastpass users while maintaining reasonable wait times for those in the normal line. This inquiry is crucial to optimizing the effectiveness of the Fastpass ticket system and improving overall park operations.

## 1.2 Background

The FastPass system was created by the Walt Disney Company to enhance customer access to attractions in Disney theme parks. After the onset of COVID-19, in 2021, Disney discontinued this system in Disneyland. However, they continue to operate a similar system under a different name, “Genie[1]”, allowing customers who pay an additional fee to access attractions quickly. Another popular theme park, Universal Studios, also sells tickets with similar functionality under the name "Express Pass tickets[2]". Many amusement parks, in a similar fashion, use systems where customers who pay extra for FastPass tickets can bypass regular queues and gain expedited access to attractions.

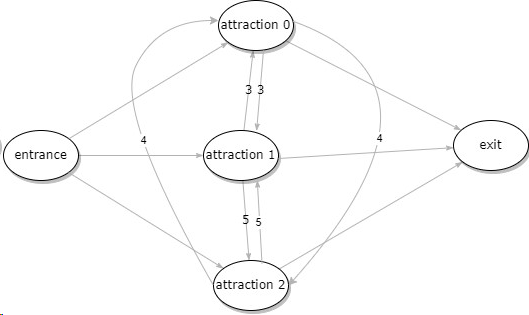
The way these FastPass tickets work is by allowing the buyers to enter a separate queue when using the amusement rides. Therefore, the significance of purchasing these tickets is that only a small proportion of the visitors to the amusement park purchase them. If all visitors to the amusement park were to purchase these tickets, unless there are no tickets with higher priority, all visitors would have to wait as long as regular visitors do.

# 2 System Description

The project assumes that the FastPass system is effective when the average waiting time for regular ticket buyers is twice as long as the average waiting time for FastPass ticket buyers, while ensuring that the waiting time for regular ticket buyers remains within reasonable limits. This ensures that the benefits of the FastPass system are evident in reducing waiting times for FastPass users while maintaining an acceptable waiting experience for those with regular tickets.

We will perform several experiments with different ratios and variable ranges to see how the waiting time changes depending on the ratio. Consequently, through simulation, we can determine to what extent we adjust the ratio of FastPass ticket buyers in order to achieve a targeted gap in waiting times. Considering the complexity of amusement park flow tracking, we make some assumptions to limit the problem statement.

Figure 1 illustrates the amusement park map and passenger flow. In this map, each Node represents an attraction, and the distances between two attractions are denoted as time intervals, measured in minutes. Visitors will enter the amusement park via the entrance from the park's opening time and make their attraction selections based on the priority order list. Visitors will be assigned to random order of attractions, and they will remain in their chosen line once they choose it. There are only two lines in every attraction: normal line and FastPass line. There is no higher priority line for elderies or disabled people. Visitors will leave the park immediately either upon experiencing all the attractions in their queue or at the park closing time. All FastPass buyers can ride the attraction first compared to normal visitors. Exit here represents the monitor in our actual simulation.



**Figure 1: amusement park map**

## 2.1 configuration & variables

There are variables that change depending on the ratio of FastPass ticket buyers.

| Variables | definition |
| --- | --- |
| Number of amusement park attractions | 3 |
| Number of visitors | 300-600 (5-10 per group generated per minutes) |
| Park simulation time | 1 hour |
| Entrance time for each tourist | A group of people will enter the park at 1-minute intervals. |
| Distance (time) between attractions (minutes) | We assume that people walk at a certain rate and it takes a specific time between two attractions. The time between attractions will be set to different constant numbers. |
| Priority order list | Each tourist will have a priority order list for attractions. Here is an example.   | priority order | Node | | --- | --- | | 1st | 1 | | 2nd | 2 | | 3rd | 3 | | 4th | 3 | | 5th | 2 | | 6th | 1 | | 7th | 0 | | 8th | 0 | | 9th | 2 | | 10th | 3 |   The range of the number of attractions in the queue is between 0 and 3. The value of 0 in the queue is used for the scenario where a visitor will ride fewer than 10 attractions. When a node is assigned the value 0, the visitor proceeds to the next ride in the queue.This priority list will be set randomly. Visitors can revisit attractions. |
| Duration for each attraction | Each amusement ride operates for a specific duration. The machines operate with a high degree of precision, and with each attempt, they run for the exact same duration of time.   | Node | Duration (minutes) | | --- | --- | | 1 | 4 | | 2 | 3 | | 3 | 5 | |
| Capacity of each attraction | Each attraction has its own capacity.   | Node | Capacity | | --- | --- | | 1 | 10 | | 2 | 7 | | 3 | 12 | |
| Ratio of FastPast Buyer | [20,50,70] (%) |
| Waiting time for each attraction | Waiting time for each attraction will be different depending on the ratio of FastPass ticket buyers   | attraction | FastPass waiting time (minutes) | normal waiting time (minutes) | | --- | --- | --- | | 1 | F1 | N1 | | 2 | F2 | N2 | | 3 | F3 | N3 | |

**Table 1: variables and their definitions**

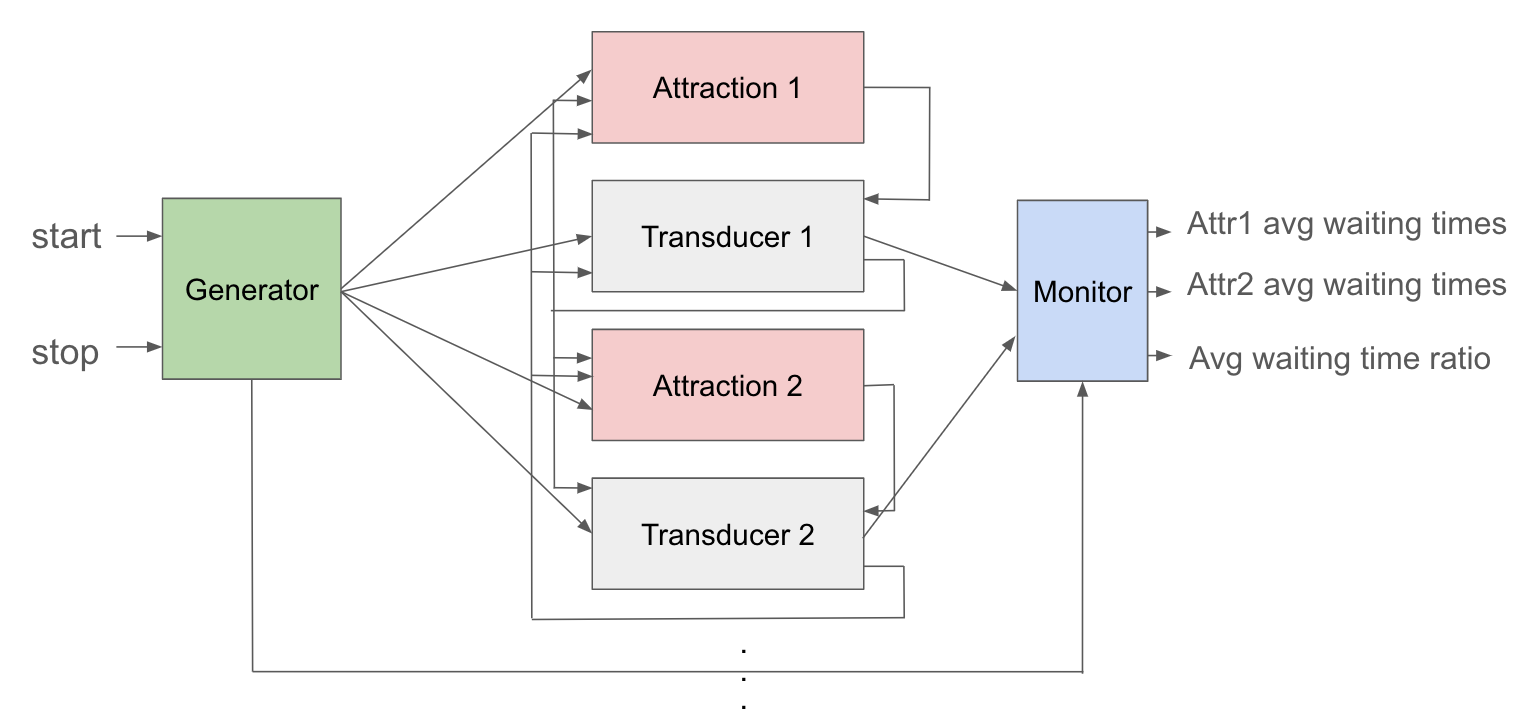
# 3 Modeling Approach

In this project, our goal is to implement a coupled model system where each component of the amusement park serves as an atomic component in the system, and visitors become a job entity. These components are interconnected, creating an integrated model. The objective is to simulate the entire system, all to achieve our simulation goal; finding the optimal sales ratio of Fastpass tickets.

This model comprises four components: the Generator, Attractions, Transducers, and Monitor. Initially, the Generator produces visitors, followed by multiple processors that that manage the various activities occurring within the attractions. Corresponding transducers track time-related events generated by these processors. Finally, the Monitor processes the generated output to present it in the desired format.

As visitors arrive with prioritized lists of rides to visit, our model is structured to ensure each visitor experiences rides in the order specified on their list. In the simulation model, all visitors generated by the generator are initially directed to all rides, mimicking the real-world movement of visitors between attractions. Subsequently, they complete their tasks by assessing priorities with each respective attraction, and this information is exchanged with other attractions through associated transducers.

Once the most recently generated visitors have completed all the rides on their priority lists at each processor, each transducer eventually sends the total waiting time and the number of visitors of the two types to the monitor. The monitor receives this information, calculates average values, and then displays the significance of the difference between the two waiting times.



**Figure2 : Amusement park system abstraction**

The specification of this coupled model is described below.

=

The following subsection describes and specifies each atomic component, including the implementations.

3.1 Model Development

## A. Visitor

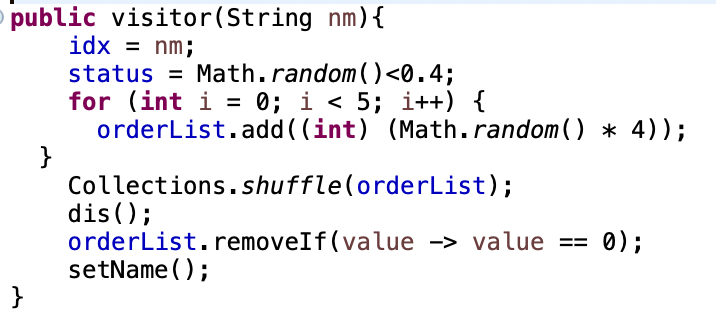
1. Description

Generator generates entities called “visitor”. Each visitor has their name, ticket type and a list of priority. Constructor for creating instances defines what they will have. They gives each instance String type of name, boolean type of status and a list that has random integer values from 0 to 3. When they have 0 in the order list, it means, they don’t get any attraction at that moment. By changing the probability when they assign status value, we can adjust the ratio of Fastpass ticket buyers. For example, if the probability is 0.4, 40% of visitors are holding Fastpass ticket.

1. Specification

1. Implementation

The implementation is done according to the description and specification. When there are 0s in the order list, they remove them before they output the visitor group.



**Figure3 : Visitor Entity**

## 

## B. Generator

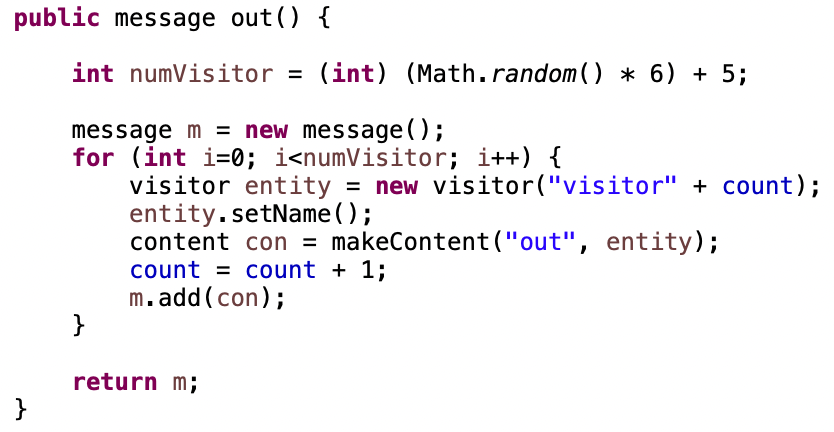
1. Description

The generator will generate a group of people every minute. Each group has the number of people specified in the configuration within the given range. When they generate people, they use for loop. After appending each entity to the message, they return all at the same time. By using the variable “index”, we make them enable to count the number of entire visitors.

1. Specification

where

1. Implementation



**Figure4 : Output function of Generator**

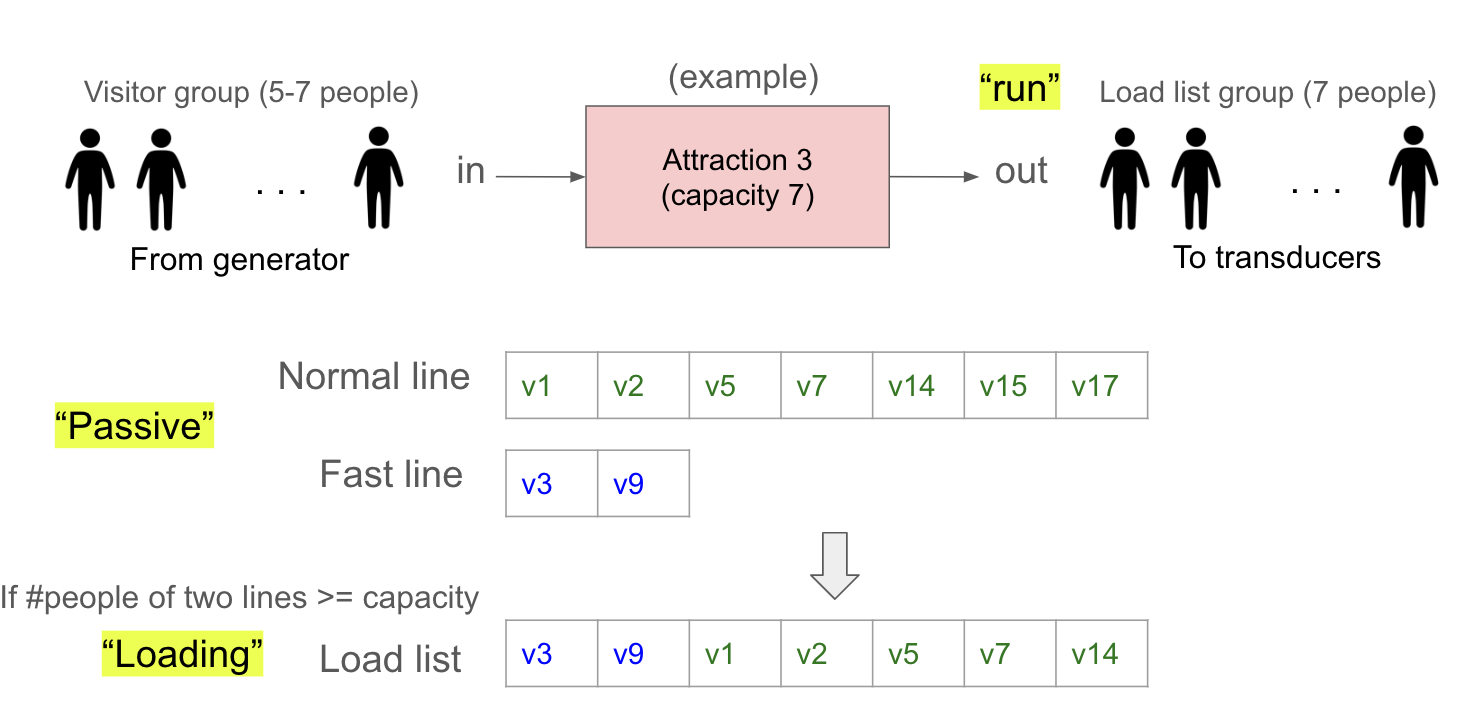
The most important part of the generator is output function. In order to output a visitor group every minute, they firstly assign a random number of visitor of a group in certain range, and then create each entity to attach them to the message. After iterating through a for loop to attach different objects to a message, the generator output the message.

## C. Attraction

1. Description

In the project, each ride at the amusement park is implemented as a processor component in the model. The system has as many processors as there are rides, and each processor interacts with generators and transducers, by exchanging their inputs and outputs.

The operation works as follows: When a visitor entity is created and sent to a processor, they are initially categorized into different groups based on their first-priority ride. Then, if the group corresponds to the processor, which is the ride they are assigned to, they are placed into two different queues within the processor according to a predefined algorithm. Figure 6 shows how queues work as two waiting lines. The initial state of the processor is "Loading." Subsequently, as the count reaches the capacity of the ride, the entities are removed from the queues, and the processor's status is changed to "Run." The status changes back to "Loading" after the duration of the ride, at which point the Transducer immediately conveys information about the visitors who used the previous ride. The Transducer passes this information to other processors, and this process results in the removal of elements from the visitors' priority lists. In the end, when all visitors' priority lists become null, the final waiting time is calculated.



**Figure5 : Attraction processor component diagram**

1. Specification

where

1. Implementation

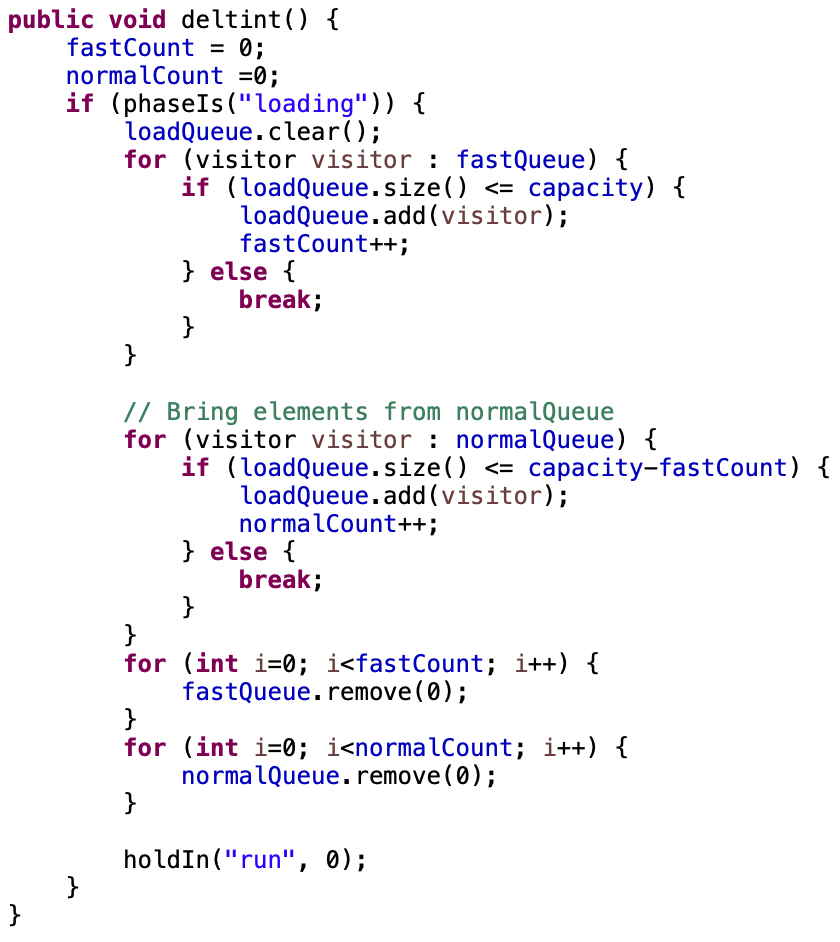


**Figure6 : External function of Attraction model**

In external function of Attraction model, they receive the input from the Generator. Once visitor group is received, they classify them into two lines: Fast line and normal line. After they added to two types of lines, the model check the total number of visitors in two lines. If they exceed the capacity, they change their phase from “passive” to “loading”.

Internal function check whether the current phase is “loading” or not. If the phase is loading, they load people to the attraction. They add people from two types of line to the load queue. After all visitors in the queue for the fast line have been added to the load queue, the remaining available spots are filled by visitors from the queue for the normal line. The added visitors are then removed from their respective lines, and the subsequent phase transitions to "run." Code is implemented in figure 7.

In output function, if the phase is “run”, they output the visitor group to the transducer.

****

**Figure7 : Internal function of Attraction model**

## 

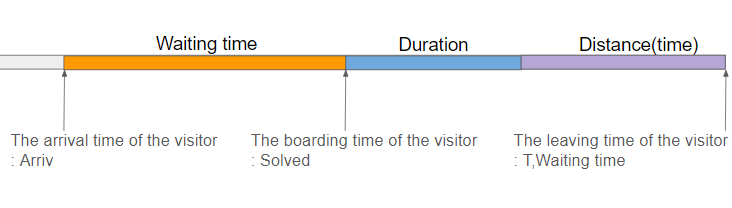
## D. Transducer

1. Description

Every time the generator generates output, every transducer will receive visitors from the generator through the “ariv” port. When a visitor arrives at an attraction managed by the transducer during their visit, the transducer records the arrival time of the visitor.

When the visitor finishes a ride at the attraction, the attraction will send its outputs to the corresponding transducer through the “solvd” port and the transducer will record visitors’ boarding time. By using the arrival time and boarding time, the transducer can calculate the waiting time.

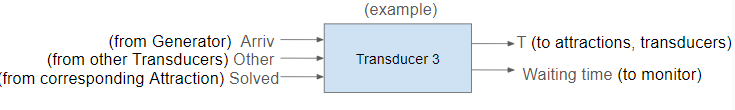
But since all visitors need to wait for the operation time of the attraction and the interval time required between the previous and next attraction, visitors cannot leave the transducer immediately.

****

**Figure8 : The flow chart of visitor in transducer**

The transducer receives the information that some visitors have finished their ride at other attractions through the “other” port. The reason why we need to receive this information is because we need to update every visitor’s arrival time.

In the end, it will send waiting time for every visitor to monitor through the “Waiting time” port. And also sent out visitors to all attractions and the rest of the transducers through the “T” port.



**Figure9: Transducer**

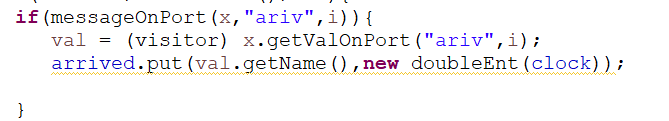
1. Specification

are visitors that are ready to send out

where

1. Implementation

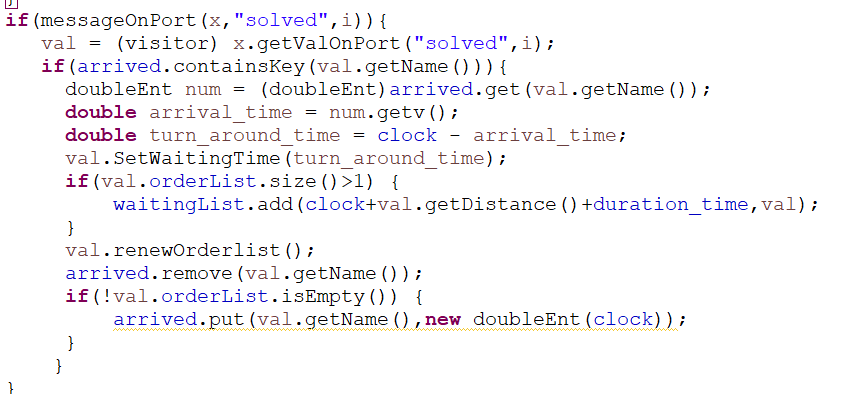
On the “ariv“ port, the transducer will update the "arrived" map by adding all visitors along with their current timestamps.



**Figure10 : external function on “ariv” port**

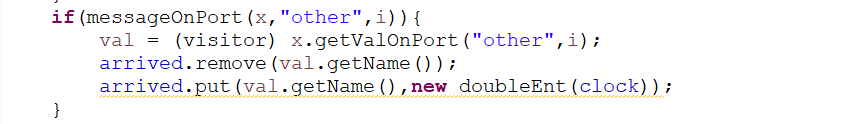
On the “solved'' port, the transducer will check every arriving visitor toidentify any matching visitors in the "arrived" map. Upon finding a match, it calculates the waiting time for that visitor and updates the waiting time of that visitor using the “setWaitingTime” function. At the same time, the transducer will add these visitors to a multivalued map called “waitingList”. Utilized their desired exit time as keys and visitors themselves as value.

Subsequently, the transducer refreshes the orderList of visitors by removing the first element of the priority list. Additionally it will update the clock of these visitors in the "arrived" map to obtain the latest arrival time for each visitor.



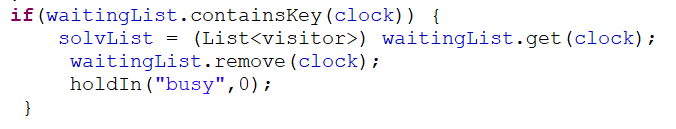
**Figure11 : external function on “solv” port**

On the “other” port, we update the clock of certain visitors by first removing them from “arrived” map and then add them back with the current clock.

****

**Figure12 : external function on “other” port**

They will be sent out once their intended departure time, as recorded in the "waitingList," has been reached.



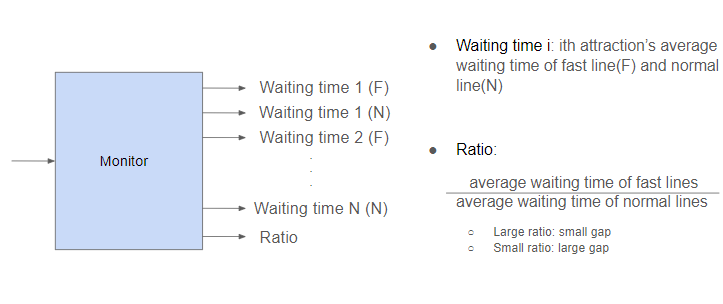
**Figure13 : achieve desired clock**

## 

## E. Monitor

1. Description

Monitor will receive waiting time from every transducer. For each attraction, it will calculate the average waiting time for both FastPass visitors and normal visitors. Also, it will calculate the ratio that is defined as dividing average waiting time of fast lines by average waiting time of normal lines. If the ratio is large, it means the gap between the waiting time of the fast line and waiting time of the normal line is small, vice versa, no. Ultimately, it provides information to determine the fastpass ticket sales ratio by analyzing the difference in waiting times between the two types of visitors.



**Figure14: Monitor**

1. Specification

,,,}

where Ai means the ith attraction. F means fast line, N means normal line

,,,}

where Ai means the ith attraction. F means fast line, N means normal line

where

1. Implementation

Waiting time will be added to different variables based on the port they are from. For example, waiting time for attraction1 will be sent through port “ Attr1WaitingTime”, therefore it will be added to a variable named “Attr1FastWaitingTime”.



**Figure15 : external function of monitor**

The monitor will eventually calculate the average waiting time of each attraction for both fast line and normal line.

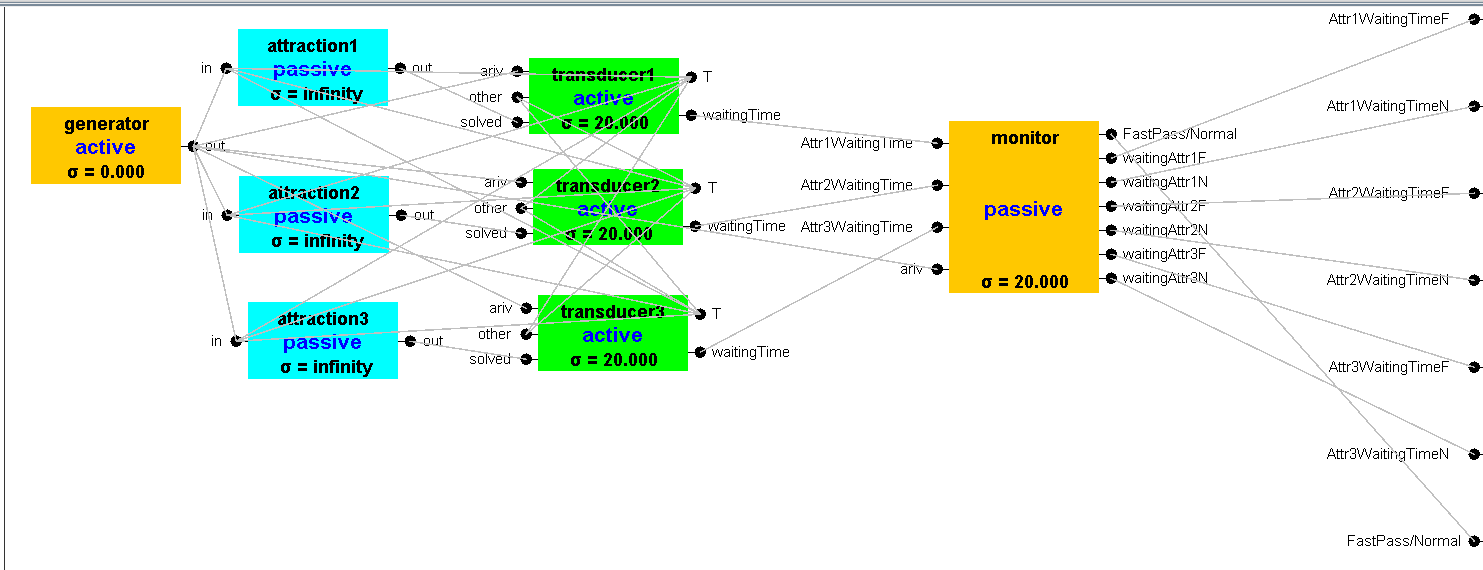
****

**Figure16 : output function of monitor**

# 4 Simulation Experiments

The simulation experiments were conducted three times for each of the three different fastpass ticket purchase ratios. Other variables are set as defined in the configuration. Each time, as the number of visitors is randomly determined, and as their priorities change, running multiple simulations for a single configuration will help obtain valid results. The fastpass ratios are set to 20%, 50%, and 70%, as specified in the configuration above. By observing the resulting differences in the final average waiting times, it will aid in determining the "optimal fastpass ticket sales ratio" targeted in this project.

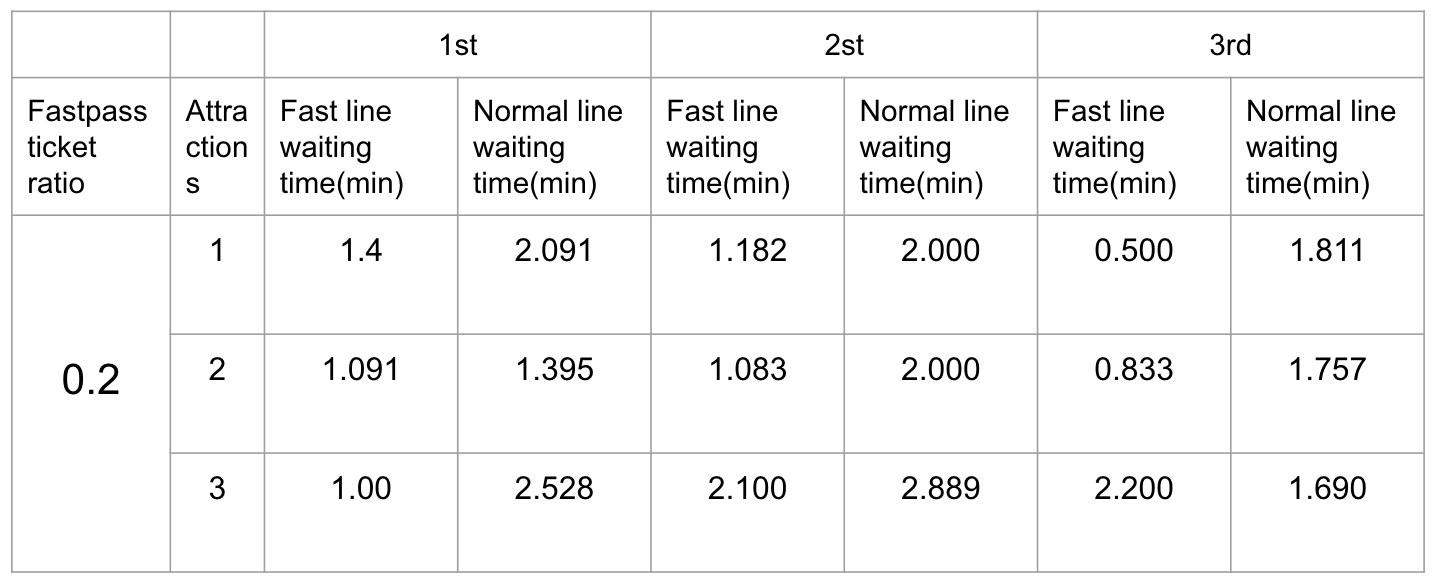
The diagram below provides a simplified overview of how our model operates in the DEVS simulator. Visitors generated by the generator are passed to each attraction and transducer, with the final step being the delivery of the Monitor’s output to the ultimate Results. Even though we use only three attractions in this simulation, this model is expandable.

**Figure 16 : DEVS simulator view**

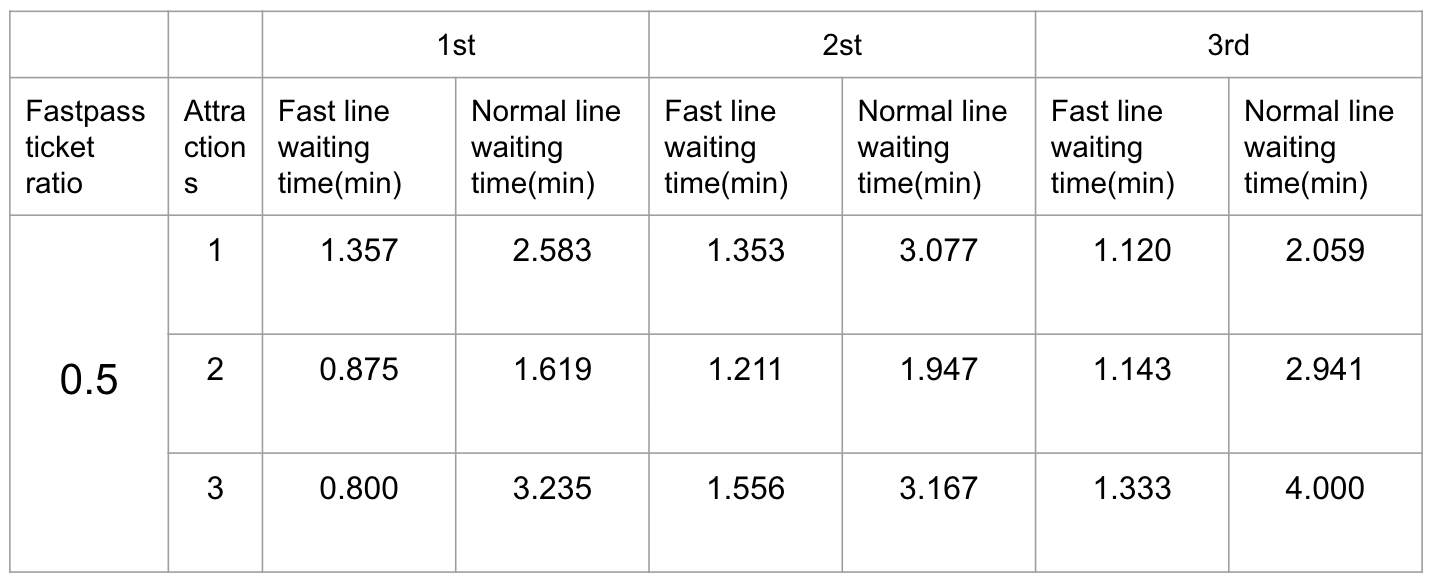
The following are results of setting the fastpass ratios to 0.2, 0.5 and 0.7. All waiting times in the normal line are longer than in the fast line. The fast line’s waiting line ranged from 0.800 to 2.200 while the normal line’s waiting time ranged from 1.395 to 9.000. Notably, in the third table, the waiting time of normal has escalated significantly compared to the other two scenarios.

After conducting numerous experiments, we have obtained more robust results. The consistency across multiple similar experiments confirms the reliability and validity of our experimental approach. Further analysis of the data substantiates the outcomes aligned with our initial hypotheses.

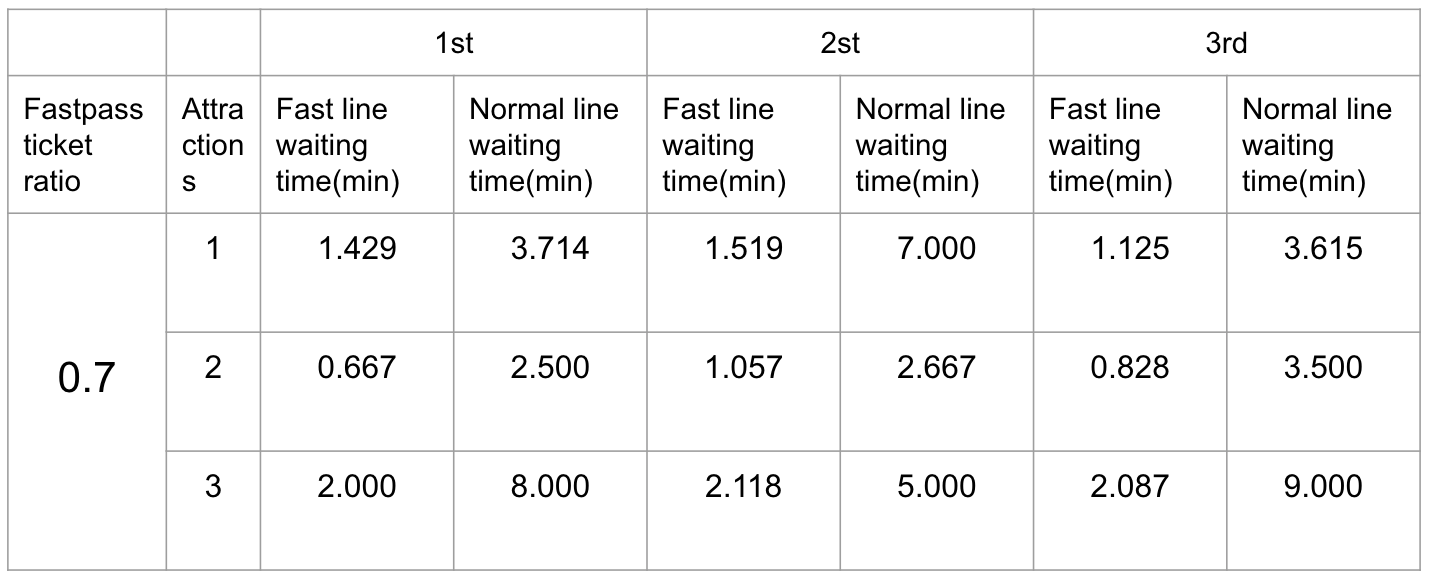
# 



**Table 2: The waiting time results for a ratio of 0.2**



**Table 3: The waiting time results for a ratio of 0.5**



**Table 4: The waiting time results for a ratio of 0.7**

5 Results and Evaluation

The assumption we made prior to the experiment was that an increase in the sales ratio of Fastpass tickets would lead to a diminishing perceived "priority" advantage for Fastpass ticket holders, resulting in an extended waiting time for the fast line. Specifically, assuming that all visitors possess Fastpass tickets, we conducted the experiment with the consideration that their waiting times would be nearly indistinguishable from those in a system without this feature.

An intriguing observation emerged: as the sales ratio increased, the ratio of waiting times for the fast line to the normal line decreased. In other words, it indicated a widening gap in waiting times between these two groups. Despite the increase in waiting time for the fast line, this result suggested that the increase of Fastpass holders significantly and abruptly escalated the waiting time for the normal line.

If we aim to establish a fastpass ticket sales ratio that results in a waiting time for the two groups with approximately a twofold difference, we should select a value for the ratio of waiting times between the two groups in the simulation results that is around 0.5.



**Table 5: The waiting time gap in ratio**

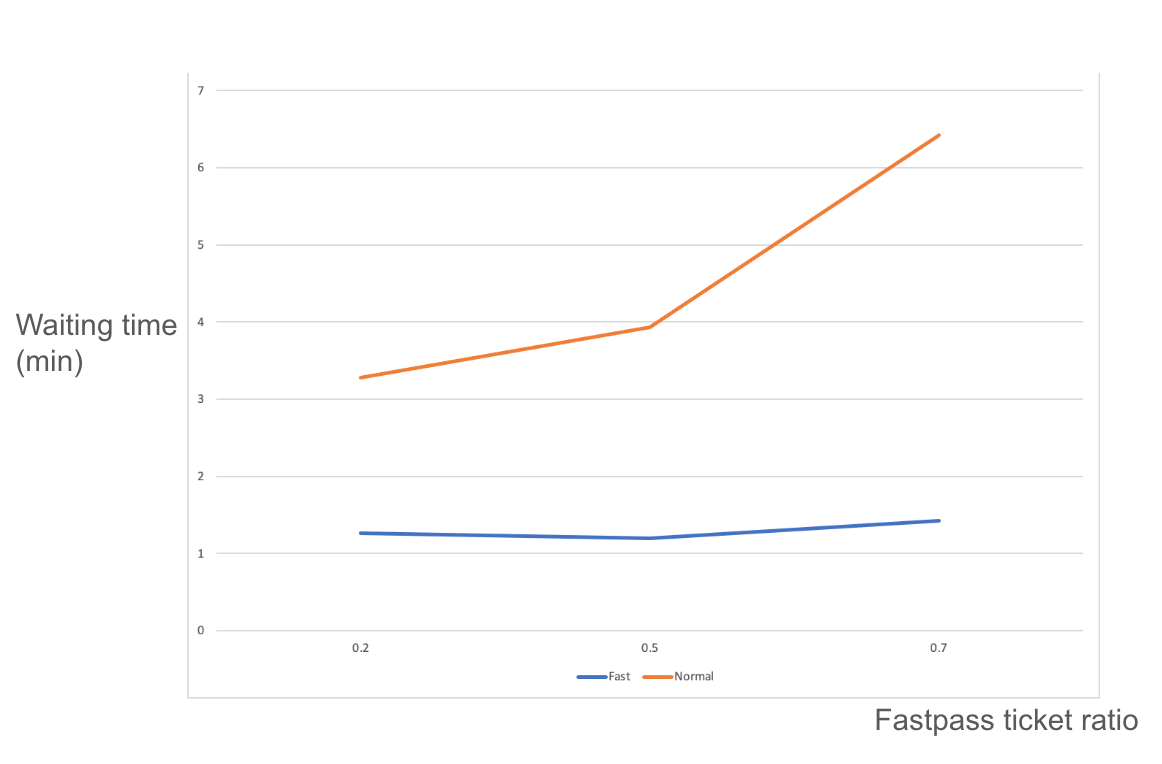
However, since this experiment was conducted with a limited number of visitors and a restricted set of attractions, adjustments to the parameters would be necessary if one intends to apply the findings to the real world. Additionally, in actual amusement parks, visitors engage in a variety of activities beyond just riding attractions. Therefore, it is essential to consider the possibility of discrepancies between experimental results and real-world applicability.

# 

# 6 Conclusion

Through the experiment results, it was observed that even with an increase in the sales ratio of fastpass tickets, the waiting time for visitors in the fastpass line did not significantly change. In contrast, the waiting time for visitors in the normal line experienced a sharp increase.

The chart below illustrates the average waiting times for the normal line and fast line based on the fastpass ticket sales ratio. As the number of fastpass passes increases, the proportion of normal ticket holders decreases relatively. Consequently, individuals with normal tickets have to wait for a longer duration.



**Figure 16 : Fastpass ticket ratio vs Waiting time of two lines**

Given our objective to minimize waiting times for Fastpass users while ensuring reasonable wait times for those in the normal line, as depicted in the aforementioned chart, raising the Fastpass ticket ratio to 70% seems impractical, as it substantially elongates waiting times for individuals in the normal line. We posit that establishing the ticket ratio within the range of 20% to 50% aligns with our considerations and is deemed optimal.

Although our experiments were conducted with a small number of visitors and a limited number of attractions, our model is designed to be scalable. When the range of variables is set on a larger scale, the experiment results are expected to closely approximate real-world scenarios. However, it's important to note that our design does not account for certain variables that may influence results, such as policies prioritizing elderly or disabled individuals, and the presence of other entertainment options. As a result, the simulation results may not perfectly reflect the complexities of a real amusement park system.

Furthermore, our approach to generating priority lists for visitors is random-based. This design choice does not consider the fact that many visitors may choose their ride order based on the distance between attractions, potentially deviating from a purely random selection process. Adding more variables to reflect the real world can be considered as future work.

# 7 References

[1]<https://disneylandtourguide.com/disney-genie-the-new-replacement-for-fastpass-at-disneyland-and-walt-disney-world/>

[2]<https://www.universalorlando.com/web/en/us/tickets-packages/express-passes>

# 

# 8 Appendix A: Source Code

A.1 gpt.java - Java script for implementing a coupled model

A.2 visitor.java - Java script for defining visitor class

A.3 genr.java - Java script for Generator

A.4 proc.java - Java script for Attractions

A.5 transd.java - Java script for Transducers

A.6 monitor.java - Java script for Monitor

| /\*  \* Author : ACIMS(Arizona Centre for Integrative Modeling & Simulation)  \* Version : DEVSJAVA 2.7  \* Date : 08-15-02  \* Modified by Jisoo Lee, Yinyu Wu (12-01-2023)  \*/  **package** Component.Final;  **import** java.awt.\*;  **import** GenCol.\*;  **import** model.modeling.\*;  **import** model.simulation.\*;  **import** view.modeling.ViewableAtomic;  **import** view.modeling.ViewableComponent;  **import** view.modeling.ViewableDigraph;  **import** view.simView.\*;  **public** **class** gpt **extends** ViewableDigraph{  **public** gpt(){  **super**("gpt");    ViewableAtomic g = **new** genr("generator",1);  ViewableAtomic p = **new** proc("attraction1",10);  ViewableAtomic p2 = **new** proc("attraction2",7);  ViewableAtomic p3 = **new** proc("attraction3",12);  ViewableAtomic t1 = **new** transd("transducer1",20,4);  ViewableAtomic t2 = **new** transd("transducer2",20,3);  ViewableAtomic t3 = **new** transd("transducer3",20,5);  ViewableAtomic m = **new** monitor("monitor",20);    add(g);  add(p);  add(p2);  add(p3);  add(m);  add(t1);  add(t2);  add(t3);    addOutport("FastPass/Normal");  addOutport("Attr1WaitingTimeF");  addOutport("Attr2WaitingTimeF");  addOutport("Attr3WaitingTimeF");  addOutport("Attr1WaitingTimeN");  addOutport("Attr2WaitingTimeN");  addOutport("Attr3WaitingTimeN");    addTestInput("start",**new** entity());  addTestInput("stop1",**new** visitor());    addCoupling(**this**,"in",g,"in");  addCoupling(**this**,"start",g,"start");  addCoupling(**this**,"stop",g,"stop");    addCoupling(g,"out",p,"in");  addCoupling(g,"out",p2,"in");  addCoupling(g,"out",p3,"in");    addCoupling(g,"out",t1,"ariv");  addCoupling(p,"out",t1,"solved");    addCoupling(g,"out",t2,"ariv");  addCoupling(p2,"out",t2,"solved");    addCoupling(g,"out",t3,"ariv");  addCoupling(p3,"out",t3,"solved");    addCoupling(g,"out",m,"ariv");    addCoupling(t1,"waitingTime",m,"Attr1WaitingTime");  addCoupling(t1,"T",p,"in");  addCoupling(t1,"T",p2,"in");  addCoupling(t1,"T",p3,"in");  addCoupling(t1,"T",t2,"other");  addCoupling(t1,"T",t3,"other");    addCoupling(t2,"waitingTime",m,"Attr2WaitingTime");  addCoupling(t2,"T",p,"in");  addCoupling(t2,"T",p2,"in");  addCoupling(t2,"T",p3,"in");  addCoupling(t2,"T",t1,"other");  addCoupling(t2,"T",t3,"other");    addCoupling(t3,"waitingTime",m,"Attr3WaitingTime");  addCoupling(t3,"T",p,"in");  addCoupling(t3,"T",p2,"in");  addCoupling(t3,"T",p3,"in");  addCoupling(t3,"T",t2,"other");  addCoupling(t3,"T",t1,"other");    addCoupling(m,"FastPass/Normal",**this**,"FastPass/Normal");  addCoupling(m,"waitingAttr1F",**this**,"Attr1WaitingTimeF");  addCoupling(m,"waitingAttr2F",**this**,"Attr2WaitingTimeF");  addCoupling(m,"waitingAttr3F",**this**,"Attr3WaitingTimeF");  addCoupling(m,"waitingAttr1N",**this**,"Attr1WaitingTimeN");  addCoupling(m,"waitingAttr2N",**this**,"Attr2WaitingTimeN");  addCoupling(m,"waitingAttr3N",**this**,"Attr3WaitingTimeN");  addCoupling(**this**,"stop",m,"stop");  }      /\*\*  \* Automatically generated by the SimView program.  \* Do not edit this manually, as such changes will get overwritten.  \*/  @Override  **public** **void** layoutForSimView()  {  preferredSize = **new** Dimension(988, 406);  ((ViewableComponent)withName("generator")).setPreferredLocation(**new** Point(18, 89));  ((ViewableComponent)withName("transducer3")).setPreferredLocation(**new** Point(298, 214));  ((ViewableComponent)withName("monitor")).setPreferredLocation(**new** Point(521, 99));  ((ViewableComponent)withName("attraction3")).setPreferredLocation(**new** Point(126, 219));  ((ViewableComponent)withName("attraction2")).setPreferredLocation(**new** Point(120, 135));  ((ViewableComponent)withName("attraction1")).setPreferredLocation(**new** Point(119, 37));  ((ViewableComponent)withName("transducer2")).setPreferredLocation(**new** Point(294, 135));  ((ViewableComponent)withName("transducer1")).setPreferredLocation(**new** Point(292, 56));  }  } |
| --- |

**Figure 17 : gpt.java (A.1)**

| /\*  \*  \* Author : ACIMS(Arizona Centre for Integrative Modeling & Simulation)  \* Version : DEVSJAVA 2.7  \* Date : 08-15-02  \* \* Modified by Jisoo Lee, Yinyu Wu (12-01-2023)  \*/  package GenCol;  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class visitor extends entity{  public String name,idx;  public Boolean status;  public double waitingTime;  public List<Integer> orderList = new ArrayList<>();  public Double[][] distance;  public Double dist = 0.0;    public visitor(){  name = "anEntity";  }    public visitor(String nm){  idx = nm;  status = Math.random()<0.4;  for (int i = 0; i < 5; i++) {  orderList.add((int) (Math.random() \* 4));  }  Collections.shuffle(orderList);  dis();  orderList.removeIf(value -> value == 0);  setName();  }    public void dis() {  distance = new Double[3][3];  distance[0][0] = 0.0;  distance[0][1] = 3.0;  distance[0][2] = 4.0;  distance[1][0] = 3.0;  distance[1][1] = 0.0;  distance[1][2] = 5.0;  distance[2][0] = 4.0;  distance[2][1] = 5.0;  distance[2][2] = 0.0;  }  public boolean eq(String nm){  return getName().equals(nm);  }    public Object equalName(String nm){  if (eq(nm)) return this;  else return null;  }    @Override  public int hashCode() {  return name.hashCode();  }    public double getDistance() {  return distance[orderList.get(0)-1][orderList.get(1)-1];  }    public boolean equals(Object o){ //overrides pointer equality of Object  if (!(o instanceof visitor))return false;  else return eq(((visitor)o).getName());  }    public ExternalRepresentation getExtRep(){  return new ExternalRepresentation.ByteArray();  }    public String getName(){  String ticket;  if (status){  ticket="Fast";  }  else {  ticket="Normal";  }  return idx+" "+orderList+" "+ticket;  }    public void setName() {  String ticket;  if (status){  ticket="Fast";  }  else {  ticket="Normal";  }  name = idx+" "+orderList+" "+ticket;  }    public void renewOrderlist() {  if(!orderList.isEmpty())orderList.remove(0);  }    public void SetWaitingTime(double waitTime) {  waitingTime = waitTime;  }    public List<Integer> getOrderList(){  return orderList;  }    public boolean getStatus(){  return status;  }    public synchronized void addSelf(ensembleCollection<Object> c){  c.add(this);  }  public synchronized void removeSelf(ensembleCollection<Object> c){  c.remove(this);  }        public String toString(){  return getName();  }    public void print(){  System.out.println(name);  }  } |
| --- |

**Figure 18 : visitor.java (A.2)**

| /\*  \* Author : ACIMS(Arizona Centre for Integrative Modeling & Simulation)  \* Version : DEVSJAVA 2.7  \* Date : 08-15-02  \* Modified by Jisoo Lee, Yinyu Wu (12-01-2023)  \*/  package Component.Final;  import java.awt.Color;  import java.lang.\*;  import java.util.ArrayList;  import java.util.List;  import GenCol.\*;  import model.modeling.\*;  import model.simulation.\*;  import view.modeling.ViewableAtomic;  import view.simView.\*;  public class genr extends ViewableAtomic {  protected double int\_arr\_time;  protected int count;  protected int time;  static int c = 0;  protected int index=0;  public genr() {  this("genr", 10);  }  public genr(String name, double Int\_arr\_time) {  super(name);  addOutport("out");  int\_arr\_time = Int\_arr\_time;  addTestInput("start", new entity(""));  addTestInput("stop", new entity(""));    setBackgroundColor(Color.orange);  }  public void initialize() {  holdIn("active", int\_arr\_time);  sigma = 0;  count = 0;  time = 0;  super.initialize();  }  public void deltext(double e, message x) {  Continue(e);  if (phaseIs("passive")) {  for (int i = 0; i < x.getLength(); i++)  if (messageOnPort(x, "start", i)) {  holdIn("active", int\_arr\_time);  }  }  if (phaseIs("active"))  for (int i = 0; i < x.getLength(); i++)  if (messageOnPort(x, "stop", i))  phase = "finishing";  }  public void deltint() {  if (phaseIs("active")) {  holdIn("active", int\_arr\_time);  } else  passivate();  }    public message out() {  int numVisitor = (int) (Math.random() \* 6) + 5;  message m = new message();  for (int i=0; i<numVisitor; i++) {  visitor entity = new visitor("visitor" + count);  entity.setName();  content con = makeContent("out", entity);  count = count + 1;  m.add(con);  }    return m;  }  public void showState() {  super.showState();  }  public String getTooltipText() {  return super.getTooltipText() + "\n" + " int\_arr\_time: " + int\_arr\_time + "\n" + " time: " + time;  }  } |
| --- |

**Figure 19 : genr.java (A.3)**

| /\*  \* Author : Savitha and Anindita ACIMS(Arizona Centre for Integrative Modeling & Simulation)  \* Version : DEVSJAVA 2.7  \* Date : 15-April-2012  \* Modified by Jisoo Lee, Yinyu Wu (12-01-2023)  \*/  package Component.Final;  import java.awt.Color;  import java.util.ArrayList;  import GenCol.entity;  import GenCol.visitor;  import GenCol.intEnt;  import java.util.List;  import model.modeling.message;  import view.modeling.ViewableAtomic;  public class proc extends ViewableAtomic {  protected intEnt job;  protected visitor visitor;  protected Integer attr;  protected Integer capacity;  protected Integer firstElement;  protected Integer fastCount, normalCount;    private List<visitor> fastQueue;  private List<visitor> normalQueue;  private List<visitor> loadQueue;    public proc(String name, int Capacity) {  super(name);  addInport("in");  addOutport("out");  if ("attraction1".equals(name)) {  attr = 1;  }  if ("attraction2".equals(name)) {  attr = 2;  }  if ("attraction3".equals(name)) {  attr = 3;  }  capacity = Capacity;    fastQueue = new ArrayList<>();  normalQueue = new ArrayList<>();  loadQueue = new ArrayList<>();    addTestInput("in", new visitor("visitor0"));  addTestInput("in", new visitor("visitor1"), 20);  addTestInput("none", new visitor("visitor"));    setBackgroundColor(Color.cyan);  }  public void initialize() {  phase = "passive";  sigma = INFINITY;  super.initialize();  }  public void deltext(double e, message x) {  Continue(e);  if (phaseIs("passive") || phaseIs("run")) {    for (int i = 0; i < x.getLength(); i++) {  if (messageOnPort(x, "in", i)) {  visitor = (visitor)x.getValOnPort("in", i);  if (!visitor.getOrderList().isEmpty()) firstElement = visitor.getOrderList().get(0);    if(firstElement==attr) {  if (visitor.getStatus()) {  // If status is true, add to the queue with status true  fastQueue.add(visitor);  } else {  // If status is false, add to the queue with status false  normalQueue.add(visitor);  }  if (fastQueue.size()+normalQueue.size()>=capacity){  // start loading  holdIn("loading",0);  }  }  }  }  }    }    public void deltint() {  fastCount = 0;  normalCount =0;  if (phaseIs("loading")) {  loadQueue.clear();  for (visitor visitor : fastQueue) {  if (loadQueue.size() <= capacity) {  loadQueue.add(visitor);  fastCount++;  } else {  break;  }  }  // Bring elements from normalQueue  for (visitor visitor : normalQueue) {  if (loadQueue.size() <= capacity-fastCount) {  loadQueue.add(visitor);  normalCount++;  } else {  break;  }  }  for (int i=0; i<fastCount; i++) {  fastQueue.remove(0);  }  for (int i=0; i<normalCount; i++) {  normalQueue.remove(0);  }    holdIn("run", 0);  }  }  public void deltcon(double e, message x) {  deltint();  deltext(e, x);  }  public message out() {  message m = new message();    if (phaseIs("run")) {  for (visitor visitor : loadQueue) {  visitor.setName();  m.add(makeContent("out", visitor));  }  passivate();  }      return m;  }  public void showState() {  super.showState();  }  } |
| --- |

**Figure 20 : proc.java (A.4)**

| /\*  \* Author : ACIMS(Arizona Centre for Integrative Modeling & Simulation)  \* Version : DEVSJAVA 2.7  \* Date : 08-15-02  \* Modified by Jisoo Lee, Yinyu Wu (12-01-2023)  \*/  package Component.Final;  import java.awt.Color;  import java.util.ArrayList;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  import GenCol.doubleEnt;  import GenCol.visitor;  import model.modeling.content;  import model.modeling.message;  import view.modeling.ViewableAtomic;  import org.springframework.util.MultiValueMap;  import org.springframework.util.LinkedMultiValueMap;  public class transd extends ViewableAtomic{  protected Map arrived, solved,others;  protected double clock,total\_ta,observation\_time,waiting\_time,duration\_time;  protected visitor sol,oth;  protected List<visitor> solvList = new ArrayList<visitor>();  protected ArrayList<visitor> otherList = new ArrayList<visitor>();  public Double dist = 0.0;  protected MultiValueMap<Double,visitor> waitingList = new LinkedMultiValueMap<>();    public transd(String name,double Observation\_time,double duration){  super(name);    addOutport("waitingTime");  addOutport("T");    addInport("ariv");  addInport("solved");  addInport("other");    arrived = new HashMap();  solved = new HashMap();  observation\_time = Observation\_time;  duration\_time = duration;  addTestInput("ariv",new visitor("val1"));  addTestInput("solved",new visitor("job1"));  addTestInput("other",new visitor("val3"));  initialize();    setBackgroundColor(Color.green);  }  public transd() {this("transd", 200,10);}  public void initialize(){  phase = "active";  sigma = observation\_time;  clock = 0;  total\_ta = 0;  solvList = new ArrayList<visitor> ();  otherList = new ArrayList<visitor> ();    super.initialize();    }  public void showState(){  super.showState();  System.out.println("arrived: " + arrived.size());  System.out.println("solved: " + solved.size());  System.out.println("TA: "+compute\_TA());  System.out.println("Thruput: "+compute\_Thru());  }  public void deltext(double e,message x){    clock = clock + e;  Continue(e);  visitor val;    for(int i=0; i< x.size();i++){  if(messageOnPort(x,"ariv",i)){  //from generator  val = (visitor) x.getValOnPort("ariv",i);  arrived.put(val.getName(),new doubleEnt(clock));  }    if(messageOnPort(x,"solved",i)){  val = (visitor) x.getValOnPort("solved",i);  if(arrived.containsKey(val.getName())){  doubleEnt num = (doubleEnt)arrived.get(val.getName());  double arrival\_time = num.getv();  double turn\_around\_time = clock - arrival\_time;  val.SetWaitingTime(turn\_around\_time);  if(val.orderList.size()>1) {  waitingList.add(clock+val.getDistance()+duration\_time,val);  }  val.renewOrderlist();  arrived.remove(val.getName());  if(!val.orderList.isEmpty()) {  arrived.put(val.getName(),new doubleEnt(clock));  }  }  }  if(messageOnPort(x,"other",i)){  val = (visitor) x.getValOnPort("other",i);  arrived.remove(val.getName());  arrived.put(val.getName(),new doubleEnt(clock));  }  if(waitingList.containsKey(clock)) {  solvList = (List<visitor>) waitingList.get(clock);  waitingList.remove(clock);  holdIn("busy",0);  }  }  }  public void deltint(){  if(phaseIs("busy")) {  if(clock<observation\_time) {  clock = clock + sigma;  sigma = observation\_time - clock;  phase = "passive";  }  }  }    public message out( ){  message m = new message();  if(phase == "busy") {  for(int i=0;i<solvList.size();i++) {  visitor v = solvList.get(i);  v.setName();  content con1 = makeContent("waitingTime",v);  content con2 = makeContent("T",v);  m.add(con1);  m.add(con2);  }  }  return m;  }      public double compute\_TA(){  double avg\_ta\_time = 0;  if(!solved.isEmpty())  avg\_ta\_time = ((double)total\_ta)/solved.size();  return avg\_ta\_time;  }      public double compute\_Thru(){  double thruput = 0;  if(clock > 0)  thruput = solved.size()/(double)clock;  return thruput;  }  } |
| --- |

**Figure 21 : transd.java (A.5)**

| /\*  \*  \* Author : ACIMS(Arizona Centre for Integrative Modeling & Simulation)  \* Version : DEVSJAVA 2.7  \* Date : 08-15-02  \* Modified by Jisoo Lee, Yinyu Wu (12-01-2023)  \*/  package Component.Final;  import java.awt.Color;  import java.lang.\*;  import java.util.ArrayList;  import java.util.List;  import GenCol.\*;  import model.modeling.\*;  import model.simulation.\*;  import view.modeling.ViewableAtomic;  import view.simView.\*;  public class monitor extends ViewableAtomic {  protected double int\_arr\_time;  protected int count;  static int c = 0;  protected int index=0;  protected double totalFastWaitTime = 0,totalNorWaitTime = 0;  protected double Attr1FastWaitingTime = 0,Attr2FastWaitingTime = 0,Attr3FastWaitingTime = 0;  protected double Attr1NorWaitingTime = 0,Attr2NorWaitingTime = 0,Attr3NorWaitingTime = 0;  protected int totalFastNum = 0,totalNorNum = 0;  protected int Attr1FastNum = 0,Attr1NorNum = 0;  protected int Attr2FastNum = 0,Attr2NorNum = 0;  protected int Attr3FastNum = 0,Attr3NorNum = 0;  public monitor() {  this("genr", 10);  }  public monitor(String name, double Int\_arr\_time) {  super(name);  addInport("Attr1WaitingTime");  addInport("Attr2WaitingTime");  addInport("Attr3WaitingTime");  addOutport("FastPass/Normal");  addOutport("waitingAttr1F");  addOutport("waitingAttr2F");  addOutport("waitingAttr3F");  addOutport("waitingAttr1N");  addOutport("waitingAttr2N");  addOutport("waitingAttr3N");  //addInport("stop");  addInport("ariv");  int\_arr\_time = Int\_arr\_time;  addTestInput("start", new entity(""));  addTestInput("stop", new entity(""));    setBackgroundColor(Color.orange);  }  public void initialize() {  holdIn("passive", int\_arr\_time);  sigma = int\_arr\_time;  super.initialize();  }  public void deltext(double e, message x) {  Continue(e);  visitor val;  if(phaseIs("passive")) {  for (int i = 0; i < x.getLength(); i++) {  if (messageOnPort(x, "Attr1WaitingTime", i)) {  val = (visitor) x.getValOnPort("Attr1WaitingTime",i);  if(val.status == true) {  Attr1FastWaitingTime+=val.waitingTime;  Attr1FastNum += 1;  }  else if(val.status == false) {  Attr1NorWaitingTime+=val.waitingTime;  Attr1NorNum += 1;  }  }  if (messageOnPort(x, "Attr2WaitingTime", i)) {  val = (visitor) x.getValOnPort("Attr2WaitingTime",i);  if(val.status == true) {  Attr2FastWaitingTime+=val.waitingTime;  Attr2FastNum += 1;  }  else if(val.status == false) {  Attr2NorWaitingTime+=val.waitingTime;  Attr2NorNum += 1;  }  }  if (messageOnPort(x, "Attr3WaitingTime", i)) {  val = (visitor) x.getValOnPort("Attr3WaitingTime",i);  if(val.status == true) {  Attr3FastWaitingTime+=val.waitingTime;  Attr3FastNum += 1;  }  else if(val.status == false) {  Attr3NorWaitingTime+=val.waitingTime;  Attr3NorNum += 1;  }  }  }  }  }  public void deltint() {  }    public void deltcon(double e, message x) {  deltint();  deltext(e, x);  }  public message out() {  message m = new message();  if(phaseIs("passive")){  System.out.println("this is monitor output function");  double AverageAttr1FastpassWaitingTime ;  if(Attr1FastNum!=0)AverageAttr1FastpassWaitingTime = Attr1FastWaitingTime/Attr1FastNum ;  else AverageAttr1FastpassWaitingTime = 0.0;  double AverageAttr2FastpassWaitingTime;  if(Attr2FastNum!=0)AverageAttr2FastpassWaitingTime = Attr2FastWaitingTime/Attr2FastNum ;  else AverageAttr2FastpassWaitingTime = 0.0;  double AverageAttr3FastpassWaitingTime;  if(Attr3FastNum!=0)AverageAttr3FastpassWaitingTime = Attr3FastWaitingTime/Attr3FastNum ;  else AverageAttr3FastpassWaitingTime = 0.0;  double a1 = AverageAttr1FastpassWaitingTime/ (Attr1NorWaitingTime/Attr1NorNum);  double a2 = AverageAttr2FastpassWaitingTime/ (Attr2NorWaitingTime/Attr2NorNum);  double a3 = AverageAttr3FastpassWaitingTime/ (Attr3NorWaitingTime/Attr3NorNum);  content con = makeContent("waitingAttr1F", new doubleEnt(AverageAttr1FastpassWaitingTime));  content con\_1 = makeContent("waitingAttr1N", new doubleEnt((Attr1NorWaitingTime/Attr1NorNum)));  content con1 = makeContent("waitingAttr2F", new doubleEnt(AverageAttr2FastpassWaitingTime));  content con1\_1 = makeContent("waitingAttr2N", new doubleEnt((Attr2NorWaitingTime/Attr2NorNum)));  content con2 = makeContent("waitingAttr3F", new doubleEnt(AverageAttr3FastpassWaitingTime));  content con2\_1 = makeContent("waitingAttr3N", new doubleEnt((Attr3NorWaitingTime/Attr3NorNum)));  content con3 = makeContent("FastPass/Normal", new doubleEnt((a1+a2+a3)/3));  m.add(con);  m.add(con\_1);  m.add(con1);  m.add(con1\_1);  m.add(con2);  m.add(con2\_1);  m.add(con3);  }  phase = "done";  return m;  }  public void showState() {  super.showState();  }  public String getTooltipText() {  return super.getTooltipText() + "\n" + " int\_arr\_time: " + int\_arr\_time + "\n" + " count: " + count;  }  } |
| --- |

**Figure 22 : monitor.java (A.6)**