Simulation Software Report for Simulating a Train Station Serving Two Types of Train

Name: Xiaoyu Yang
GTID: 903233131

1. Problem Statement:

To provide better and fast travel services to citizen by using high-speed train, City A plans to renew the original station to make it is available for two types of train, high-speed trains and low-speed trains. However, due to limitations of budget and space, only two platforms can be built. One platform can be used for both low and high-speed trains. However, another one can only be used by low-speed trains. In the nearly future, more number of high-speed trains will be deployed to replace the old low-speed trains while the total served customers maintain the same. As a result, the city hope that I can simulate the relationship between high-speed trains and average delay of trains.

SUI Detail:

Incoming trains need to use the platforms to unload and load customers. However, one platform can only serve one train at a time. For the special platform, which can only be used by low-speed train, it serves one slow-speed train at a time in first-come-first-server order. Due to the higher standard of high-speed trains, the shared platform serves the high-speed tree first when it comes or is in the queue. If both shared and special platform are empty, slow-speed trains are required to use the special platform first. Low-speed train can serve more customers compared to high-speed trains. However, it also takes longer for the customers to load and unload.

Assumptions and Simplifications:

1. Same type of train is identical in terms of waiting time inside of platforms;
2. The time of a train arrives is independent of other trains in the system;
3. The time between trains' arrival time follows the same probability distribution;
4. It is assumed that there is enough waiting area for both types of trains;
5. The waiting time is used to represent the delay of trains.

2. Conceptual Modelling Document

Project Goals

The specific interest for this study is looking forward the total waiting time when the number of high-speed trains increases.

Parameters

numHigh: represents the number of high-speed trains. Meanwhile, the numHigh also decide the average time between two coming high-speed trains. To serve constant number of customers, numLow, which represents the number of low-speed trains, is a dependent variable of numHigh. Same as numHigh, numLow decides the average time between two coming low-speed trains.

Output

avgWaitTime-High Speed: This output variable provides the average waiting time of high speed trains, which can be used to represent the situation of delay. Its increase indicates the overall delay for trains.
avgWaitTime-Low Speed: This output variable provides the average waiting time of low speed trains, which can be used to represent the situation of delay.
totalWaitTime: This output variable provides the total waiting time for both high and low speed trains, which can be used to represent the situation of delay. Its increase indicates the overall delay for trains. Waiting time for a train includes the total waiting time of trains before getting into the platform.

3. Train Station Model Summary

Entities

1. Train (attribute: TrainType)
2. Queue_High (attribute: Num_High)
3. Queue_Low (attribute: Num_Low)
4. Platform_Shared (attribute: Platform_Shared_Free)
5. Platform_Special (attribute: Platform_Special_Free)

Constants

1. Time_H: time of high speed train remains in the shared platform
2. Time_L: time of low speed train remains in platforms
3. Arrival_H: mean interarrival time for high speed trains
4. Arrival_L: mean interarrival time for low speed trains

Events

1. Arrivals
2. Departure

Activities

1. Boarding_Share
2. Boarding_Special

4. High-Level Conceptual Model

Simplification and Assumptions

1. We assume there are only two types of train;
2. We assume that capacity of low-speed trains are two times as the capacity of high-speed trains.
3. We assume the loading time of high-speed trains are three times faster than the low-speed trains.

a. Structural View

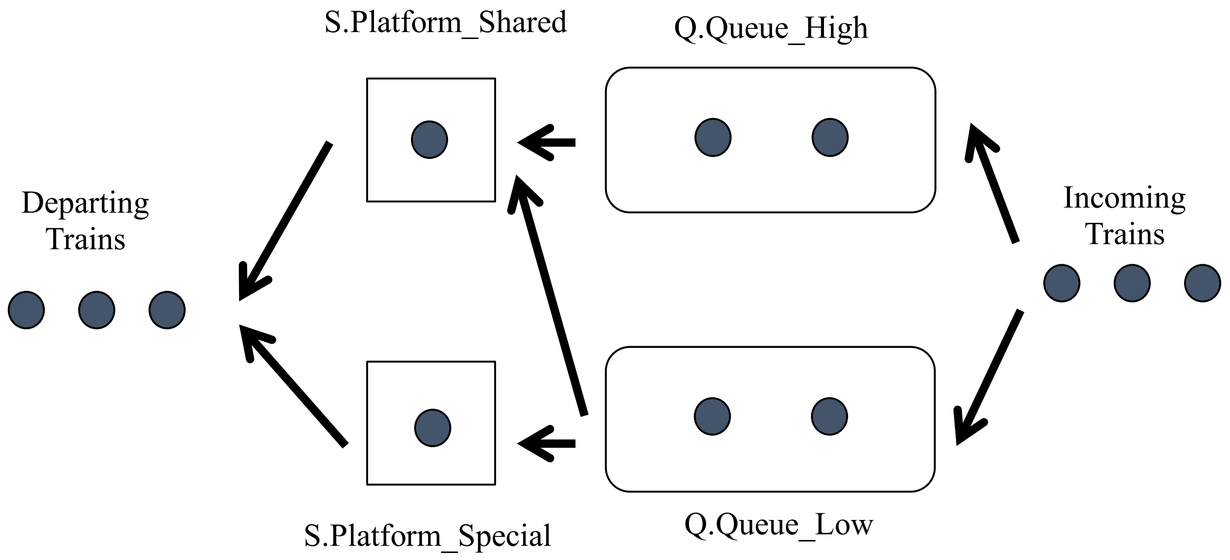


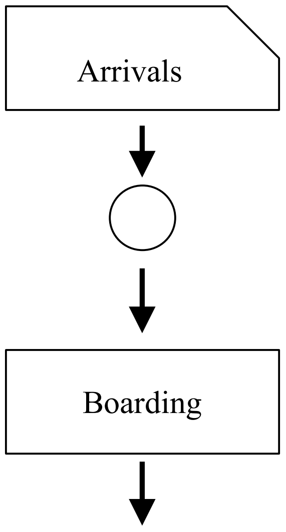
Figure 1. structure view

Entity categories:

1. Train (attribute: TrainType)
2. Queue_High (attribute: Num_High)
3. Queue_Low (attribute: Num_Low)
4. Platform_Shared (attribute: Platform_Shared_Free)
5. Platform_Special (attribute: Platform_Special_Free)

1. Train: An entity structure with *role* = Consumer and *scope* = Class that represents the collections of trains requiring loading service at the train station. There are two types of trains: the low_speed train and the high_speed train. Trains are distinguished using the train attribute TrainType, where TrainType = 0 represent low_speed train and TrainType = 1 represent high_spend train.
2. Queue_High: An entity structure with *role* = Queue and *scope* = Unary that represents the queue of high_speed trains waiting for loading/unloading. The attribute Num_High represents how many high_speed trains are waiting in the queue.
3. Queue_Low: An entity structure with *role* = Queue and *scope* = Unary that represents the queue of low_speed trains waiting for loading/unloading. The attribute Num_Low represents how many low_speed trains are waiting in the queue.
4. Platform_Shared: An entity structure with *role* = Resource and *scope* = Unary that represents the platform where both high_speed and low_speed trains load and unload.
5. Platform_Special: An entity structure with *role* = Resource and *scope* = Unary that represents the platform where only low_speed trains load and unload.

b. Behavioral View



Train Life Cycle
Figure 2: behavioral view

Actions:

1. Arrivals: This scheduled action generates the input entity stream of trains. For different types of train, the average time gap trains is going to be different.

Activities:

1. Boarding_Shared: This activity represents the Train entity unloading and loading customers on the Platform_Shared entity.
2. Boarding_Special: This activity represents the Train entity where TrainType equals to 0, which represent the low_speed train.

5. Simulation Software

Overall structure

To solve the mentioned problem, a discrete-event simulation software was developed. There are totally three files. First one is an application file, which is named as trainStation.c and contains event processing loop, event handlers, constants, and parameters. The application was implemented with 2 event handlers. The first one called Arrive, which represents the event when a train comes and waits to get into a certain platform. The second one called Departure. The reason to choose Departure as one of event type is that each type of trains spends constant time at stations and departure event is also the triggering function for updating platform status. Performance of the system is measured with the statistics of waiting time for both high and low speed trains. For each event, there are two even parameters, which are the type of trains and the type of platforms. The train type is used to record the type of train, which is needed when schedules next the arriving of next train. The platform is needed to schedule departure event where only the platform which are free can be used for a certain type of trains.

The second is an engine file, which is named as heapEngine.c. I use heap structure to implement future event list (FEL). Detailed introduction and information are described in the following session.

The third file is an interface file, which is named as trainStation.h. This file is used as a connection between the priority queue structure and application.

Heap Structure
I use heap structure to achieve the requirement of future event list. Heap is a complete binary tree, and every node's timestamp is less than or equal to that of each child node, which is shown as Figure 3. It supports two main operations: insert and delete, which are used as Schedule and Remove functions in this study.

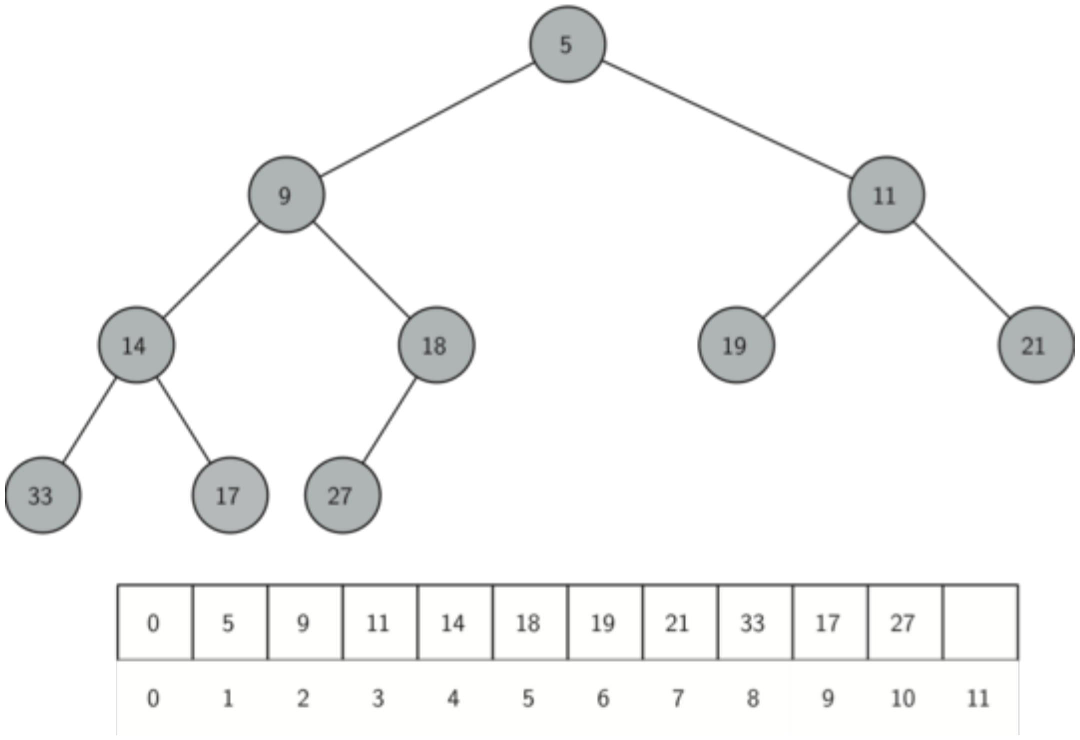
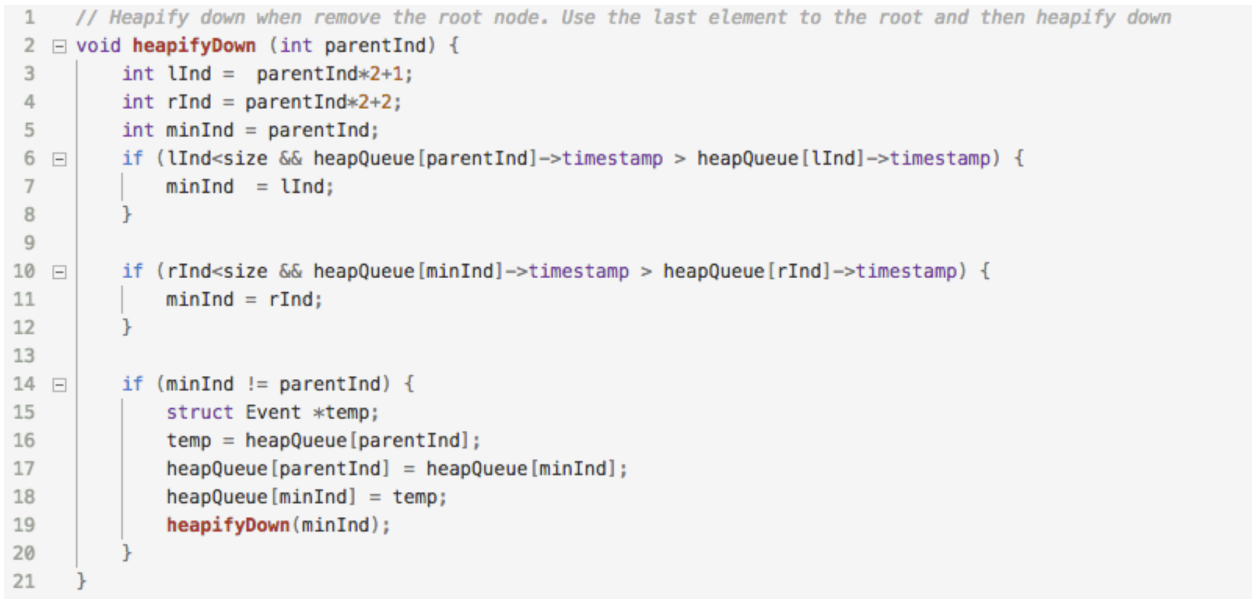


Figure 5. Heap Structure

For the delete operation, it first pop out the root, which contains the smallest timestamp. Then it replaces the root with the last node in the last level. After that, it migrates the root down the tree as far as necessary to maintain heap property, which are achieved by heapifyDown function. The pseudocode for the heapifyDown function is shown as Figure 4.



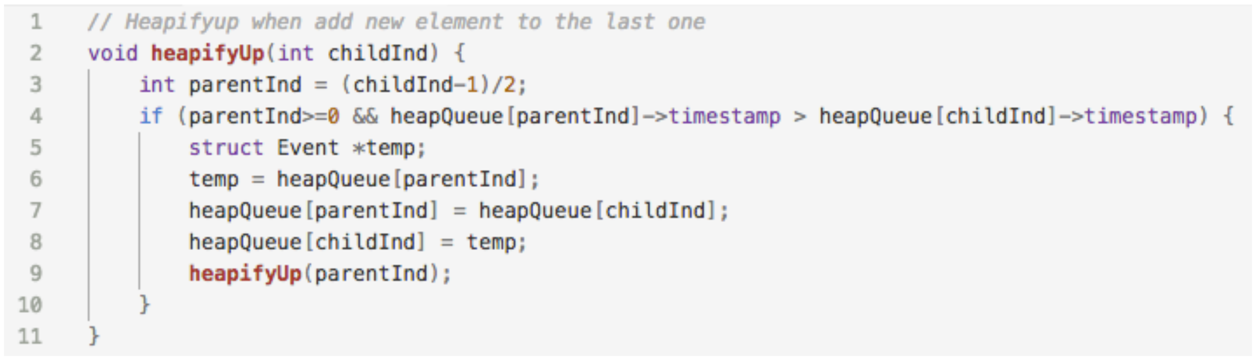
```
// Heapify down when remove the root node. Use the last element to the root and then heapify down
void heapifyDown (int parentInd) {
    int lInd =  parentInd*2+1;
    int rInd = parentInd*2+2;
    int minInd = parentInd;
    if (lInd<size && heapQueue[parentInd]->timestamp > heapQueue[lInd]->timestamp) {
        minInd  = lInd;
    }

    if (rInd<size && heapQueue[minInd]->timestamp > heapQueue[rInd]->timestamp) {
        minInd = rInd;
    }

    if (minInd != parentInd) {
        struct Event *temp;
        temp = heapQueue[parentInd];
        heapQueue[parentInd] = heapQueue[minInd];
        heapQueue[minInd] = temp;
        heapifyDown(minInd);
    }
}
```

Figure 4. heapifyDown Pseudocode

For the insert operation, it adds the new node to the next available spot in the last level. Then, it moves the new node up to restore the heap property, which are achieved by heapifyUp function. The pseudocode for the heapifyUp function is shown as Figure 5.



```
// Heapifyup when add new element to the last one
void heapifyUp(int childInd) {
    int parentInd = (childInd-1)/2;
    if (parentInd>=0 && heapQueue[parentInd]->timestamp > heapQueue[childInd]->timestamp) {
        struct Event *temp;
        temp = heapQueue[parentInd];
        heapQueue[parentInd] = heapQueue[childInd];
        heapQueue[childInd] = temp;
        heapifyUp(parentInd);
    }
}
```

Figure 5. heapifyUp Pseudocode

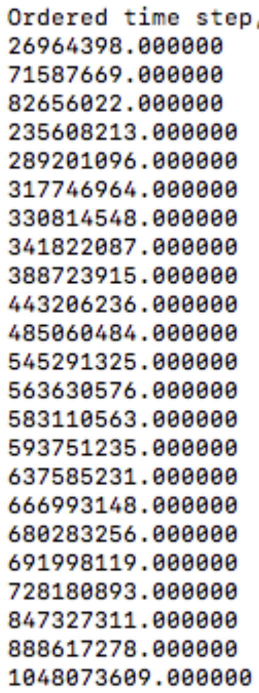
In this project, schedule operation can be used for applying both events type:
1. a new train arrives,
2. platforms are available and a train loads/unloads customers.

6. Verification and validation

Verification

To verify the correctness for both application and engine, a series of tests were designed and conducted.

The first one is to verify the correctness of the priority queue structure. I first generated a great number of random positive numbers as timestamps for events and used Schedule function to insert them into the built priority queue structure. Then I used Remove function to get and print out the timestamp of the scheduled events one by one. The results are shown as Figure 6. It is clear that the heap structure can output the timestamp in an increasing order. In addition, I also applied the developed heap structure for the airport simulation application. The simulation results are same as the one use linear list. Due to these reasons, we can verify that the heap structure we developed is functional correct.



```
Ordered time step, 50
26964398.000000
71587669.000000
82656022.000000
235608213.000000
289201096.000000
317746964.000000
330814548.000000
341822087.000000
388723915.000000
443206236.000000
485060484.000000
545291325.000000
563630576.000000
583110563.000000
593751235.000000
637585231.000000
666993148.000000
680283256.000000
691998119.000000
728180893.000000
847327311.000000
888617278.000000
1048073609.000000
```

Figure 6: Output result using heap structure

Furthermore, I tested the performance of the engine by examining its hold time versus the number of original elements stored in the queue. Based on theory, the hold time costed by heap structure is O(log(n)), while the time costed by linear list structure is O(n). The hold time is defined to be the time for putting one element into and getting one element out of the queue. To validate the correctness of the time complexity, the hold time for both heap and linear list structure, which is provided by Dr. Fujimoto, were calculated and shown as Figure 7. It shows that the hold time increases linearly as the queue size grows in the linear list while increasing in a logarithmic way in the heap. In conclusion, I can validate that the time complexity of the data structure developed meets the requirement of heap structure. With the verification of functional correctness, I can conclude that the developed heap structure is right. Furthermore, it is clear that the heap is an efficient data structure for the implementation of simulation engine.

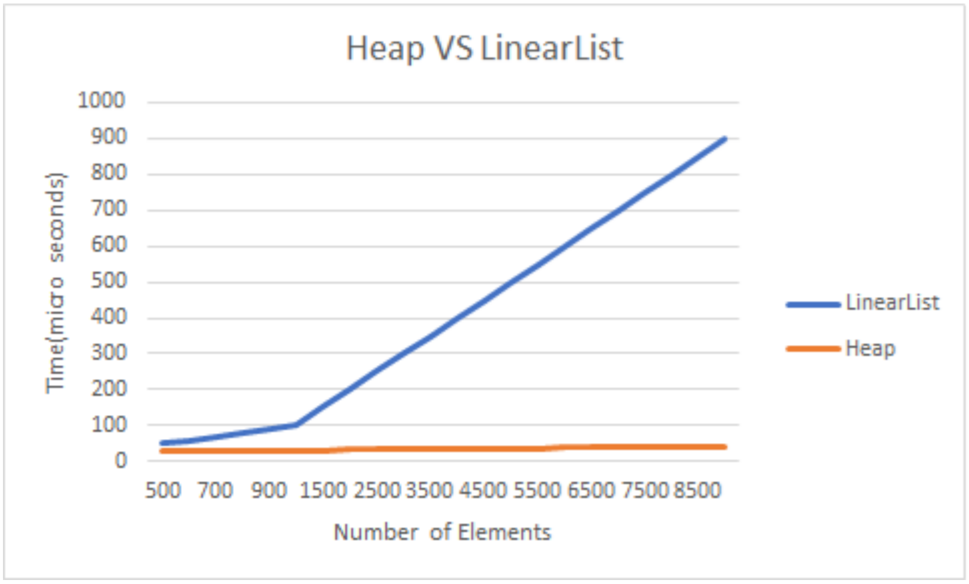


Figure 7: Time hold for Heap & Linear List structure

Validation

Based on theory, we cannot prove a model is valid. As a result, in this study, I designed and conducted several tests to show that the model is trust-worthy for the intended application under certain circumstances. Meanwhile, the concept model was explained and reviewed by a subject expert who has more than 15 years' experience on customers' service of trains. The statistic analysis and validation of the model is shown as following.

The specific interest for this study is looking forward the change of waiting time when the number of high-speed trains increases while still meet the requirement of customer transportation. In this study, I referenced the design standard for low and high-speed trains in China, where a low speed train provide two times carrying capacity than a high speed train. As a result, every time decrease one low speed train, I add two high speed trains to meet the carrying capacity requirement in the test. By decreasing the number of low speed trains from 20 to 0 and increasing the number of high speed trains from 0 to 20, I got the waiting time related variables by running the software developed.

Figure 8 shows the trend of average waiting time for high speed trains when the number of high speed trains increases. It shows that the average waiting time of high speed train decrease first, but then increase a little bit. This may cause by the great number of low speed trains which will occupy the shared platform when there are no waiting low speed trains, which will cause higher waiting time of arriving high speed trains. Then the waiting time tend to decrease continuously, which may cause by the decreasing number of low speed trains which share the platform.

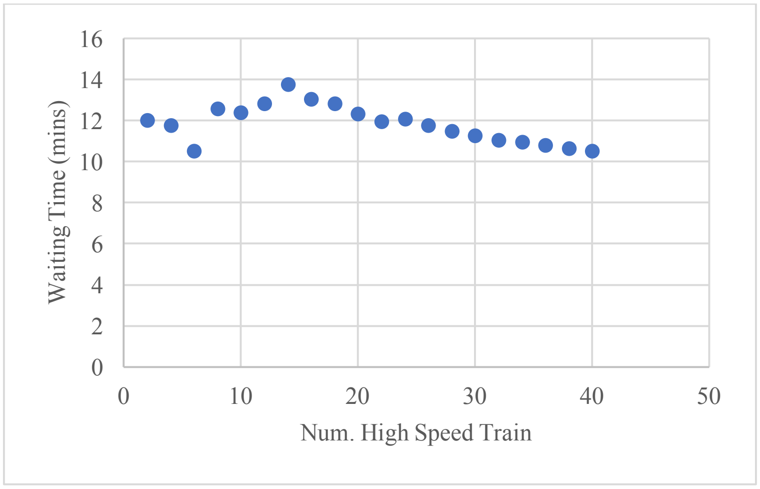


Figure 8: Average waiting time for high speed train (mins)

Figure 9 shows the trend of average waiting time for low speed trains when the number of high speed trains increases. It shows that the average waiting time of low speed train decrease first, keep stable for a while, and then increase. This may cause by the shared platform. Initially, the great number of low speed trains will cause high waiting time even though two platforms is provided. With the increasing number of high speed trains, the number of low speed trains decrease and the time confliction of low speed trains is able to reduce. However, with increasing number of high speed trains, the shared platform tend to be occupied much more often by high speed trains, which will force the low speed trains to load/unload only at the special platform, which may cause greater waiting time.

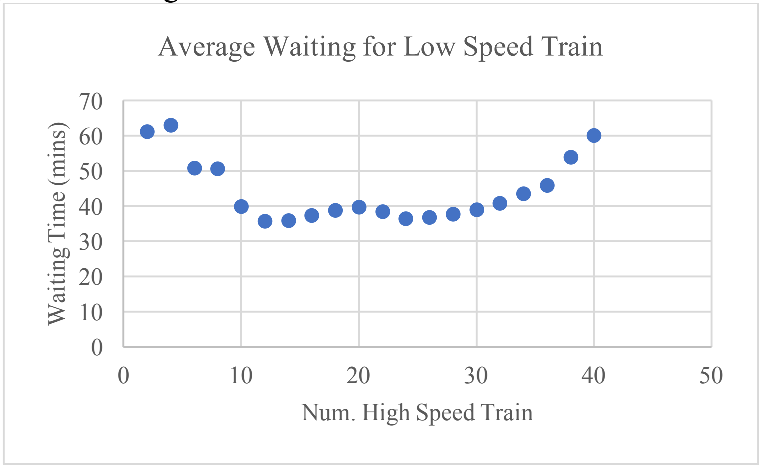


Figure 9: Average waiting time for low speed train (mins)

Figure 10 shows the trend of total waiting time for both types of train when the number of high speed trains increases. It is clear that by increasing the number of low speed train, the overall waiting time can be reduced.

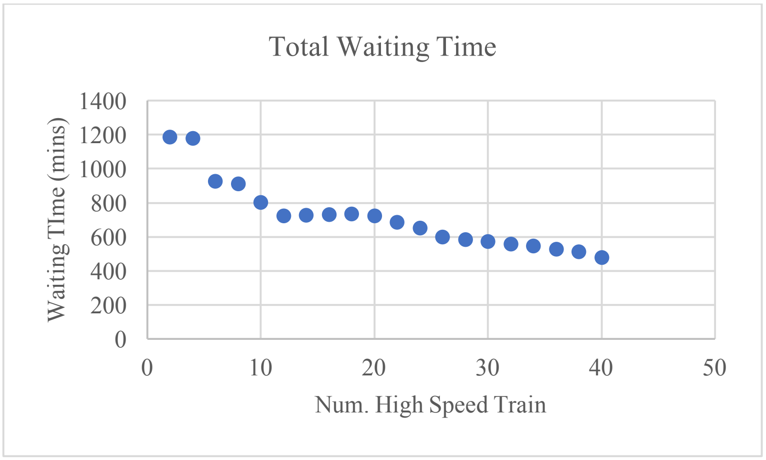


Figure 10: Total Waiting Time (mins)

Based on a series of tests, I can find out that the performance of the developed application is constant and the results make sense. As I result, I can assume that the simulation model is trust worthy for this problem and can provide useful information under certain conditions.

Appendix:

Detailed ABCmod Conceptual Model

Structural Components

Constant		
Name	Description	Values
Time_H	Time of high speed train remains in the shared platform to load and unload customers	8 minutes
Time_L	Time of low speed train remains in platforms to load and unload customers	30 minutes
Arrive_H	Mean inter arrival time for high speed trains	20 minutes
Arrive_L	Mean inter arrival time for low speed trains	50 minutes
Parameters		

Name	Description	Values
numHigh	Represents the number of high-speed trains. Meanwhile, the numHigh also decide the average time between two coming high-speed trains. To serve constant number of customers, numLow, which represents the number of low-speed trains, is a dependent variable of numHigh. Same as numHigh, numLow decides the average time between two coming low-speed trains.	5 and 10

Consumer Class: Train	
Represents trains which load/unload customers through the platforms	
Attributes	**Description**
TrainType	Integer. Represent the train type, which belongs to either high-speed train or low-speed train. 1 represents the high_speed train and 0 represents the low_speed train.

Resource Unary: Platform_Shared	
Represents shared platform which can be used to load and unload customers for both high_speed and low_speed trains.	
Attributes	**Description**
Platform_Shared_Free	Boolean, true if platform is available, false if platform is being used

Resource Unary: Platform_Special	
Represent the platform which can only be used to load and unload customers for low_speed trains.	
Attributes	**Description**
Platform_Special_Free	Boolean, true if platform is available, false if platform is being used

Queue Unary: Queue_High	
FIFO queue. This queue entity structure represents the high-speed trains which are waiting to use the shared platform.	
Attributes	**Description**

Num_High	Integer. Represent the number of high_speed trains that are waiting to use the Platform_Shared entity

Queue Unary: Queue_Low	
FIFO queue. This queue entity structure represents the low-speed trains which are waiting to go either the special platform or the shared platform.	
Attributes	**Description**
Num_Low	Integer. Represent the number of low_speed trains that are waiting to use the Platform_Shared or Platform_Special entity.

Behavioural Construct

Activity: Boarding_Special	
Models a low_speed train uses the special platform to load/unload.	
Precondition	(Platform_Special_Free AND <low_speed train at front of queue>)
Initiating Event	Platform_Special_Free = FALSE
Duration	Time L units of time
Terminating Event	Num_Low = Num_Low – 1 Platform_Special_Free = TRUE

Activity: Boarding_Shared	
Models a train used the shared platform to load/unload.	
Precondition	(Platform_Shared_Free AND <high_speed train at front of queue>) OR (Platform_Shared_Free AND <high_speed train not at front of queue> AND <low_speed train at front of queue> AND (NOT Platform_Special_Free))
Initiating Event	Platform_Shared_Free = FALSE
Duration	Time H unit of time IF (TrainType == 1) Time L unit of time IF (TrainType == 0)
Terminating Event	Num_High = Num_High – 1 IF (TrainType == 1); Num_Low = Num_Low – 1 IF (TrainType == 0); Platform_Shared_Free = TRUE