VALIDATION REPORT

Railway Control System

Group 1
Author: Chenyang Zhang

Table of Contents

Overview 4

System Architecture 4

T1: Unit Test 5

T1.1: Controller Unit Test 5

T1.1.1: Test updateState() 5

T1.1.2: Test adjustTrainState() 5

T1.1.3: Test trainLeave() 6

T1.1.4: Test trainArrive() 7

T1.1.5: Test findFreeRail() 7

T1.1.6: Test switchRail() 8

T1.1.7: Test freezeRail() 9

T1.1.8: Test deceToAvoidCollision() 9

T1.1.9: Test recover() 10

T1.1.10: Test raiseMessage() 10

T1.1.11: Test findStation() 11

T1.1.12: Test findTrain() 11

T1.1.13: Test removeStoppingTrain() 12

T1.1.14: Test sortByLocation() 12

T1.1.15: Test sortByType() 13

T1.1.16: Test sortByLeaveTime() 13

T1.1.17: Test sortByTypeAndLeaveTime() 14

T1.1.18: Test findMaxFactor() 14

T1.2: Train Unit Test 14

T1.2.1: Test updateState() 14

T1.2.2: Test tryToStart() 15

T1.2.3: Test start() 15

T1.2.4: Test moveForward() 15

T1.2.5: Test arrive() 16

T1.2.6: Test toStopped() 16

T1.2.7: Test toAcceToCru() 17

T1.2.8: Test toAcceToMax() 17

T1.2.9: Test toDeceToCru() 17

T1.2.10: Test toDeceToZero() 17
T1.2.11: Test toCruisingSpeed() 18
T1.2.12: Test toMaximumSpeed() 18
T1.2.13: Test stopAtNext() 18
T1.2.14: Test accelerateToCruisingSpeed() 19
T1.2.15: Test accelerateToMaximumSpeed() 21
T1.2.16: Test distToStop() 23
T1.2.17: Test timeToStop() 23
T1.2.18: Test timeToStop() 24
T1.2.19: Test needAvoiding() 24
T1.2.20: Test Gbehind() 25
T1.3: Station Unit Test 26
T1.3.1: Test updateState() 26
T1.3.2: Test rearrangeTrainLeaveTime() 27
T1.4: System Unit Test 27
T1.4.1: Test updateState() 27
T1.4.2: Test beginSimulate() 28
T1.4.3: Test stopSimulate() 28
T1.5: Display Unit Test 28
T1.5.1: Test updateState() 28
T1.5.3: Test showRailState() 29
T1.6: TrainPanel Unit Test 30
T1.6.1: Test updateState() 30
T2: Integration Test 30
T2.1: Display Panel UI + Train Panel UI + Controller + Environment 30
T2.1.1: Test “Begin simulate” 30
T2.1.2: Test “Stop the train at next station” 31
T2.1.3: Test “Accelerate the train to cruising speed” button. 32
T2.1.4: Test “Accelerate the train to maximum speed” 33
T2.1.5: Test “Switch a rail” 33
T3: Functional Test 34
T3.1: Use Case “Auto control” 34
T3.2: Use Case “Manual control” 34

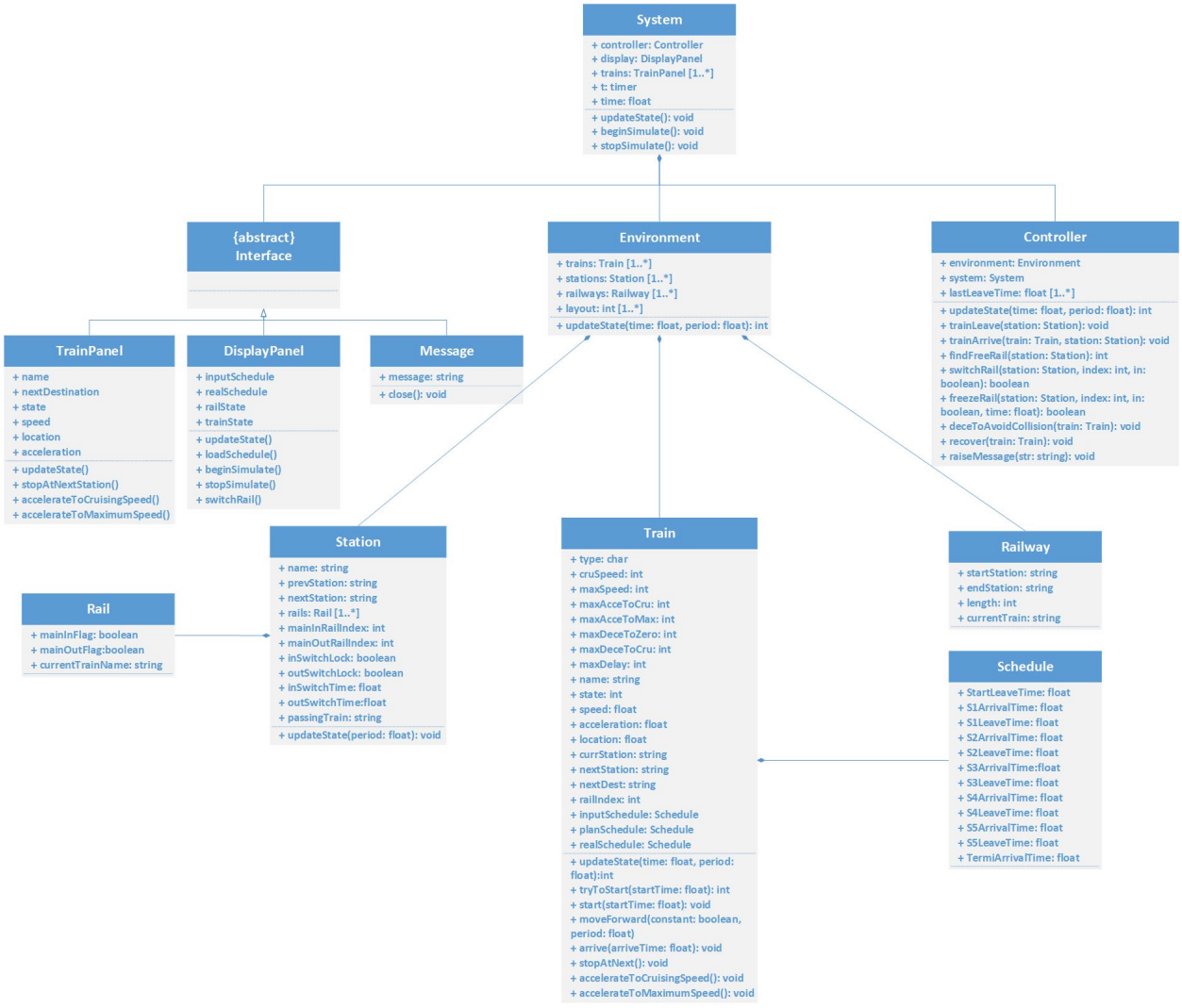
T4: Model Checking 35
Train Model 35
Rail Model 36
Controller Model 36
Check Property 36
P1: Completeness 36
P2: Safety 36
P3: Delay 37
P4: Permission 37
P5: Display 37

Overview

Since there are a large number of branches in the railway control system, our validation will focus on covering all branches. The functional test will cover a major amount of branches. Some small testcases will be added to test some individual branches which are not covered in the functional test.

System Architecture

The system architecture is shown below:



T1: Unit Test

In this part, we will only test some special branches with special testcases. The rest of the branches are tested by the functional test in T2. However, we ensure that all branches in the functions are covered in the unit test and the functional test.

T1.1: Controller Unit Test

T1.1.1: Test updateState()

```
function allFinish = updateState(obj, time, period)
    %    The main control function of controller, executed for each
    %    time step.

    %    Update the state of environment
    allFinish = obj.hEnvironment.updateState(time, period);
    %    Adjust the train state globally
    obj.adjustTrainState(time);
    %    Statement - Tcover 1.1.1.1
end
```

The function has no branches. It's executed for each time step so it's covered in functional test.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.1.2: Test adjustTrainState()

The source code of this function is too long to insert here, so we just demonstrate our test process. The branches can be found in *adjustTrainState()* function in *Controller.m* file.

The branches 1.1.2.1, 1.1.2.3.2, 1.1.2.4.1 – 1.1.2.4.2, 1.1.2.4.4, 1.1.2.5.1 – 1.1.2.5.2, 1.1.2.5.4 – 1.1.2.5.5 are covered in the functional tests. So we just need to test 1.1.2.2, 1.1.2.3.1, 1.1.2.4.3 and 1.1.2.5.3.

- Test Case T1.1.2.1
  Coverage item: Tcover 1.1.2.2.
  State: obj.hEnvironment.trains = [train1, train2], train1.location = train2.location.
  Expected Output: The collision message is raised.
  Test Result: passed.

- Test Case T1.1.2.2
  Coverage Item: Tcover 1.1.2.3.1.
  State: obj.hEnvironment.trains = [train1, train2], train1.location = 4700, train2.location = 4800,
  train1.speed = 300, train2.speed = 0, train2.acceleration = 60.
  Expected Output: The train1 should begin decelerating (train1.state = 2).
  Test Result: passed.

- Test Case T1.1.2.3
  Coverage Item: Tcover 1.1.2.4.3.
  State: obj.hEnvironment.trains = [train1, train2], train1.isStopping = 0, train1.state = 2,

train1.speed = 150, train1.cruSpeed = 200, train1.location = 1000, train2.location = 3000.
Expected Output: The train1 should begin accelerating (train1.state = 1).
Test Result: passed.

- Test Case T1.1.2.4
  Coverage Item: Tcover 1.1.2.5.3.
  State: obj.hEnvironment.trains = [train1,train2], train1.state = 3, train1.speed = 150,
  train1.cruSpeed = 200, train1.location = 1000, train2.location = 3000.
  Expected Output: The train1 should begin accelerating (train1.state = 1).
  Test Result: passed.

- Test Coverage: 16/16 = 100% (4 in the unit test, 12 in the functional test).

T1.1.3: Test trainLeave()

```
function trainLeave(obj, station)
    %    Reset the rail information
    station.rails{station.mainOutRailIndex + 1}.currentTrainName = "None";
    in = 0;
    obj.switchRail(station, 0, in);    %  recover
    %    Record real leave time
    if station.name == "S1"
        %    Branch - Tcover 1.1.3.1
        obj.lastLeaveTime{2} = obj.hSystem.time;
    elseif station.name == "S2"
        %    Branch - Tcover 1.1.3.2
        obj.lastLeaveTime{3} = obj.hSystem.time;
    elseif station.name == "S3"
        %    Branch - Tcover 1.1.3.3
        obj.lastLeaveTime{4} = obj.hSystem.time;
    elseif station.name == "S4"
        %    Branch - Tcover 1.1.3.4
        obj.lastLeaveTime{5} = obj.hSystem.time;
    elseif station.name == "S5"
        %    Branch - Tcover 1.1.3.5
        obj.lastLeaveTime{6} = obj.hSystem.time;
    end
end
```

The branches 1.1.3.1 – 1.1.3.5 are all covered in the functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 5/5 = 100% (0 in the unit test, 5 in the functional test).

T1.1.4: Test trainArrive()

```
function trainArrive(obj, train, station)
    %    Set the rail information
    station.rails{station.mainInRailIndex + 1}.currentTrainName = train.name;
    train.railIndex = station.mainInRailIndex;
    in = 1;
    obj.switchRail(station, 0, in);
    %    Statement - Tcover 1.1.4.1
end
```

The function has no branches and it is executed when a train arrives at the station. So it's covered in functional test.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.1.5: Test findFreeRail()

```
function index = findFreeRail(obj, station)
    for index=2:station.totalRailNum
        rail = station.rails{index};
        if rail.currentTrainName == "None"
            %    Branch - Tcover 1.1.5.1
            index = index - 1; %#ok<FXSET>
            return
        end
    end
    %    Branch - Tcover 1.1.5.2
    obj.raiseMessage(string(['Rails are full in station ', station.name]));
    index = -1;
end
```

The branch 1.1.5.1 is covered in the functional tests. So we just need to test 1.1.5.2.

- Test Case T1.1.5.1
  Coverage Item: Tcover 1.1.5.2.
  State: station.totalRailNum = 1, station.rails{2}.currentTrainName = "G1".
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Coverage: 2/2 = 100% (1 in the unit test, 1 in the functional test).

T1.1.6: Test switchRail()

```
function result = switchRail(~, station, index, in)
    result = 0;
    if in == 1
        %    Branch - Tcover 1.1.6.1
        if station.inSwitchLock == 0
            %    Branch - Tcover 1.1.6.1.1
            return
        end
        %    Branch - Tcover 1.1.6.1.2
        station.inSwitchLock = 0;
        station.rails{station.mainInRailIndex + 1}.mainInFlag = 0;
        station.mainInRailIndex = index;
        station.rails{index + 1}.mainInFlag = 1;
        station.inSwitchTime = 1;
        result = 1;
    else
        %    Branch - Tcover 1.1.6.2
        if station.outSwitchLock == 0
            %    Branch - Tcover 1.1.6.2.1
            return
        end
        %    Branch - Tcover 1.1.6.2.2
        station.outSwitchLock = 0;
        station.rails{station.mainOutRailIndex + 1}.mainOutFlag = 0;
        station.mainOutRailIndex = index;
        station.rails{index + 1}.mainOutFlag = 1;
        station.outSwitchTime = 1;
        result = 1;
    end
end
```

The branches 1.1.6.1.2 and 1.1.6.2.2 are covered in the functional tests. So we just need to test 1.1.6.1.1 and 1.1.6.2.1.

- Test Case T1.1.6.1
  Coverage Item: Tcover 1.1.6.1.1.
  State: in = 1, station.inSwitchLock = 0.
  Expected Output: The function should return directly without switching the rail.
  Test Result: passed.

- Test Case T1.1.6.2
  Coverage Item: Tcover 1.1.6.2.1.
  State: in = 0, station.outSwitchLock = 0.

Expected Output: The function should return directly without switching the rail.
Test Result: passed.

- Test Coverage: 4/4 = 100% (2 in the unit test, 2 in the functional test).

T1.1.7: Test freezeRail()

```
function result = freezeRail(obj, station, index, in, time)
    result = 0;
    switchResult = obj.switchRail(station, index, in);
    if switchResult == 1
        %    Branch - Tcover 1.1.7.1
        result = 1;
        if time > 1
            %    Branch - Tcover 1.1.7.1.1
            if in == 1
                %    Branch - Tcover 1.1.7.1.1.1
                station.inSwitchTime = time;
            else
                %    Branch - Tcover 1.1.7.1.1.2
                station.outSwitchTime = time;
            end
        end
    end
end
```

The branches in 1.1.7.1 are all covered in the functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.1.8: Test deceToAvoidCollision()

```
function deceToAvoidCollision(~, train)
    if (train.speed > 200 && rem(train.speed, 10) == 0) || rem(train.speed, 20) == 0
        %    Branch - Tcover 1.1.8.1
        if train.speed > train.cruSpeed
            %    Branch - Tcover 1.1.8.1.1
            train.toDeceToCru();
        else
            %    Branch - Tcover 1.1.8.1.2
            train.toDeceToZero();
        end
    end
end
```

The branches in 1.1.8.1 are all covered in the functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.1.9: Test recover()

```
function recover(obj, train)
    if rem(train.location, 1) ~= 0
        %    Branch - Tcover 1.1.9.1
        return
    end
    %    Branch - Tcover 1.1.9.2
    if train.speed > train.cruSpeed
        %    Branch - Tcover 1.1.9.2.1
        train.toAcceToMax(obj.findMaxFactor(train.maxAcceToMax, -train.maxDeceToCru));
    elseif train.speed == train.cruSpeed
        %    Branch - Tcover 1.1.9.2.2
        train.toCruisingSpeed();
    elseif train.speed == train.maxSpeed
        %    Branch - Tcover 1.1.9.2.3
        train.toMaximumSpeed();
    else
        %    Branch - Tcover 1.1.9.2.4
        train.toAcceToCru(obj.findMaxFactor(train.maxAcceToCru, -train.maxDeceToZero));
    end
end
```

The branches 1.1.9.1 – 1.1.9.2 are all covered in functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 5/5 = 100% (0 in the unit test, 5 in the functional test).

T1.1.10: Test raiseMessage()

```
function raiseMessage(obj, str)
    e = Message;
    e.hSystem = obj.hSystem;
    e.Label.Text = str;
    stop(obj.hSystem.t);
    %    Statement - Tcover 1.1.10.1
end
```

The function has no branches and it is not covered in the functional test, so we need to test it here.

- Test Case T1.1.10.1
  Coverage Item: Tcover 1.1.10.1.
  Input: str = "Error message!"

Expected Output: The message should pop out with the text “Error message!”.
Test Result: passed.

- Test Coverage: 1/1 = 100% (1 in the unit test, 0 in the functional test).

T1.1.11: Test findStation()

```
function station = findStation(obj, name)    %  helper function
    for i=1:obj.hEnvironment.stationNum
        station = obj.hEnvironment.stations{i};
        if char(station.name) == name
            %    Branch - Tcover 1.1.11.1
            return
        end
    end
    %    Branch - Tcover 1.1.11.2
    obj.raiseMessage("Invalid station name!!!");
end
```

The branch 1.1.11.1 is covered in functional test, so we just need to test 1.1.11.2.

- Test Case T1.1.11.1
  Coverage Item: Tcover 1.1.11.2.
  Input: name = “Error”
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Coverage: 2/2 = 100% (1 in the unit test, 1 in the functional test).

T1.1.12: Test findTrain()

```
function train = findTrain(obj, name)    %  helper function
    for i=1:obj.hEnvironment.trainNum
        train = obj.hEnvironment.trains(i);
        if char(train.name) == name
            %    Branch - Tcover 1.1.12.1
            return
        end
    end
    %    Branch - Tcover 1.1.12.2
    obj.raiseMessage("Invalid train name!!!");
end
```

The branch 1.1.12.1 is covered in functional test, so we just need to test 1.1.12.2.

- Test Case T1.1.12.1
  Coverage Item: Tcover 1.1.12.2.

Input: name = “Error”
Expected Output: The error message should be raised.
Test Result: passed.

- Test Coverage: 2/2 = 100% (1 in the unit test, 1 in the functional test).

T1.1.13: Test removeStoppingTrain()

```
function trainList = removeStoppingTrain(~, trains)    %  helper function
    i = 1;
    trainList = trains;
    while i <= length(trainList)
        if trainList(i).state == 0
            %    Branch - Tcover 1.1.13.1
            trainList(i) = [];
            continue
        end
        %    Branch - Tcover 1.1.13.2
        i = i + 1;
    end
end
```

The branches 1.1.13.1 – 1.1.13.2 are covered in functional test, so we don’t have additional testcases to cover these branches.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.1.14: Test sortByLocation()

```
function sortedTrainList = sortByLocation(~, trainList)    %  helper function
    sortedTrainList = trainList;
    if length(sortedTrainList) == 1
        %    Branch - Tcover 1.1.14.1
        return
    end
    %    Branch - Tcover 1.1.14.2
    for i=2:length(sortedTrainList)
        train = sortedTrainList(i);
        j = i-1;
        while j>0 && train.location < sortedTrainList(j).location
            sortedTrainList(j+1) = sortedTrainList(j);
            j = j-1;
        end
        sortedTrainList(j+1) = train;
    end
end
```

The branches 1.1.14.1 – 1.1.14.2 are covered in functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.1.15: Test sortByType()

```
function sortedTrainList = sortByType(~, trainList)    %  helper function
    sortedTrainList = trainList;
    if length(sortedTrainList) == 1
        %    Branch - Tcover 1.1.15.1
        return
    end
    %    Branch - Tcover 1.1.15.2
    for i=2:length(sortedTrainList)
        train = sortedTrainList(i);
        j = i-1;
        while j>0 && train.maxSpeed > sortedTrainList(j).maxSpeed
            sortedTrainList(j+1) = sortedTrainList(j);
            j = j-1;
        end
        sortedTrainList(j+1) = train;
    end
end
```

The branches 1.1.15.1 – 1.1.15.2 are covered in functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.1.16: Test sortByLeaveTime()

```
function sortedTrainList = sortByLeaveTime(obj, trainList)    %  helper function
    station = obj.findStation(trainList(1).currStation);
    sortedTrainList = trainList;
    if length(sortedTrainList) == 1
        %    Branch - Tcover 1.1.16.1
        return
    end
    %    Branch - Tcover 1.1.16.2
    for i=2:length(sortedTrainList)
        train = sortedTrainList(i);
        j = i-1;
        currLeaveTime = eval(['train.planSchedule.', char(station.name), 'LeaveTime', ':']);
        while j>0 && currLeaveTime < eval(['sortedTrainList(j).planSchedule.', char(station.name), 'LeaveTime', ':'])
            sortedTrainList(j+1) = sortedTrainList(j);
            j = j-1;
        end
        sortedTrainList(j+1) = train;
    end
end
```

The branches 1.1.16.1 – 1.1.16.2 are covered in functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.1.17: Test sortByTypeAndLeaveTime()

```
function sortedTrainList = sortByTypeAndLeaveTime(obj, trainList)    %  helper function
    sortedTrainList = obj.sortByLeaveTime(trainList);
    sortedTrainList = obj.sortByType(sortedTrainList);
    %    Statement - Tcover 1.1.17.1
end
```

The function has no branch and it is covered in functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.1.18: Test findMaxFactor()

```
function n = findMaxFactor(~, a, b)    %  helper function
    for n=a:-1:1
        if rem(b/n, 1) == 0
            %    Branch - Tcover 1.1.18.1
            return
        end
    end
end
```

The branch 1.1.18.1 is covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2: Train Unit Test

T1.2.1: Test updateState()

The source code of this function is too long to insert here, so we just demonstrate our test process. The branches can be found in *updateState()* function in *Train.m* file.

The function is executed for each time step. The branches 1.2.1.1 – 1.2.1.9 are covered in the functional tests. So we just need to test 1.2.1.10.

- Test Case T1.2.1.1
  Coverage Item: Tcover 1.2.1.10.
  State: obj.speed = 300, obj.location = 4700, obj.targetLocation = 4800.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Coverage: 25/25 = 100% (1 in the unit test, 24 in the functional test).

T1.2.2: Test tryToStart()

The source code of this function is too long to insert here, so we just demonstrate our test process. The branches can be found in *tryToStart()* function in *Train.m* file.

The branches 1.2.2.1 – 1.2.2.9 are covered in the functional tests. So we don't have additional testcases to cover these.

- Test Coverage: 10/10 = 100% (0 in the unit test, 10 in the functional test).

T1.2.3: Test start()

```
function start(obj, startTime)
    obj.isStarting = 0;    %  recover
    obj.railIndex = 0;    %  recover
    inputLeaveTime = eval(['obj.', 'inputSchedule.', char(obj.currStation), 'LeaveTime', ';']);
    if startTime > inputLeaveTime && abs(startTime - inputLeaveTime) > 1e-6 ...
            && obj.distFromZeroToCru + obj.distFromCruToMax + obj.distFromMaxToCru + obj.distFromCruToZero <= obj.targetLocation - obj.location
        %    Branch - Tcover 1.2.3.1
        obj.toMaximum = 1;
    end
    obj.toAcceToCru(obj.maxAcceToCru);
    obj.updateScheduleFlag = 1;    %  update real schedule
    eval(['obj.', 'realSchedule.', char(obj.currStation), 'LeaveTime', '=', num2str(startTime), ';']);    %  record the real leave time

    if obj.currStation ~= "Start"
        %    Branch - Tcover 1.2.3.2
        obj.controller.trainLeave(obj.controller.findStation(obj.currStation));
    else
        %    Branch - Tcover 1.2.3.3
        obj.controller.lastLeaveTime{1} = startTime;
    end
end
```

The branches 1.2.3.1 – 1.2.3.3 are covered in the functional tests. So we don't have additional testcases to cover these.

- Test Coverage: 3/3 = 100% (0 in the unit test, 3 in the functional test).

T1.2.4: Test moveForward()

```
function moveForward(obj, constant, period)
    if constant == 0
        %    Branch - Tcover 1.2.4.1
        prevSpeed = obj.speed;
        obj.speed = obj.speed + obj.acceleration * period;
        currSpeed = obj.speed;
        obj.location = obj.location + (currSpeed+prevSpeed)/2 * period;
    else
        %    Branch - Tcover 1.2.4.2
        obj.location = obj.location + obj.speed * period;
    end
end
```

The branches 1.2.4.1 – 1.2.4.2 are covered in the functional tests. So we don't have additional testcases to cover these.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.2.5: Test arrive()

```
function arrive(obj, arriveTime)
    obj.isStopping = 0;    %  recover
    obj.freezeRail = 0;    %  recover
    obj.currStation = obj.nextDest;    %  update current station
    obj.updateScheduleFlag = 1;    %  update real schedule
    eval(['obj.', 'realSchedule.', char(obj.currStation), 'ArrivalTime', '=', num2str(arriveTime), ';']);    %  Record the real arrival time
    %    Update the parameters if it isn't terminal station
    if obj.currStation ~= "Termi"
        %    Branch - Tcover 1.2.5.1
        %    Set into the rail
        station = obj.controller.findStation(obj.currStation);
        obj.controller.trainArrive(obj, station);
        %    Set the default leave time of current station if it isn't a destination in plan or it has been late
        planArrivalTime = eval(['obj.', 'planSchedule.', char(obj.currStation), 'ArrivalTime']);
        planLeaveTime = eval(['obj.', 'planSchedule.', char(obj.currStation), 'LeaveTime']);
        if planLeaveTime == -1
            %    Branch - Tcover 1.2.5.1.1
            %    The default leave time is 5s after the arrival
            eval(['obj.', 'planSchedule.', char(obj.currStation), 'LeaveTime', '=', num2str(arriveTime+5), ';']);
        elseif planArrivalTime < arriveTime && abs(planArrivalTime - arriveTime) > 1e-6
            %    Branch - Tcover 1.2.5.1.2
            interval = planLeaveTime - planArrivalTime;
            eval(['obj.', 'planSchedule.', char(obj.currStation), 'LeaveTime', '=', num2str(arriveTime+interval), ';']);
        end
        obj.nextStation = obj.stations(1);    %  set the next station(maybe not the destination)
        obj.nextDest = obj.destinations(1);    %  set the next destination
        obj.targetLocation = obj.environment.getStationPosition(obj.nextDest);    %  set the position of next destination
        obj.stations(1) = [];    %  pop the next station
        obj.destinations(1) = [];    %  pop the next destination
    else
        obj.nextDest = "Finished";
    end
end
```

The branch 1.2.5.1.2 is covered in the functional test, so we just need to test 1.2.5.1.1.

- Test Case T1.2.5.1
  Coverage Item: Tcover 1.2.5.1.1
  State: time = 60, obj.speed = 0, obj.currStation = “S1”, obj.planSchedule.S1LeaveTime = -1.
  Expected Output: obj.planSchedule.S1LeaveTime = 65.
  Test Result: passed.

- Test Coverage: 2/2 = 100% (1 in the unit test, 1 in the functional test).

T1.2.6: Test toStopped()

```
function toStopped(obj)    %  state = 0
    obj.state = 0;    %  set to stopped
    obj.acceleration = 0;    %  set the acceleration
    %    Statement - Tcover 1.2.6.1
end
```

The function has no branches and it’s covered in the functional test, so we don’t have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.7: Test toAcceToCru()

```
function toAcceToCru(obj, acceleration)    %  state = 1
    obj.state = 1;    %  set to accelerating
    obj.acceleration = acceleration;    %  set the acceleration
    %    Statement - Tcover 1.2.7.1
end
```

The function has no branches and it's covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.8: Test toAcceToMax()

```
function toAcceToMax(obj, acceleration)    %  state = 1
    obj.state = 1;
    obj.acceleration = acceleration;    %  set the acceleration to maximum speed
    obj.toMaximum = 0;
    %    Statement - Tcover 1.2.8.1
end
```

The function has no branches and it's covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.9: Test toDeceToCru()

```
function toDeceToCru(obj)    %  state = 2
    obj.state = 2;
    obj.acceleration = obj.maxDeceToCru;
    %    Statement - Tcover 1.2.9.1
end
```

The function has no branches and it's covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.10: Test toDeceToZero()

```
function toDeceToZero(obj)    %  state = 2
    obj.state = 2;
    obj.acceleration = obj.maxDeceToZero;
    %    Statement - Tcover 1.2.10.1
end
```

The function has no branches and it's covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.11: Test toCruisingSpeed()

```
function toCruisingSpeed(obj)    %  state = 3
    obj.state = 3;    %  set to crusing speed
    obj.acceleration = 0;    %  set the acceleration
    %    Statement - Tcover 1.2.11.1
end
```

The function has no branches and it's covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.12: Test toMaximumSpeed()

```
function toMaximumSpeed(obj)    %  state = 4
    obj.state = 4;    %  set to maximum speed
    obj.acceleration = 0;    %  set the acceleration
    %    Statement - Tcover 1.2.12.1
end
```

The function has no branches and it's covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.13: Test stopAtNext()

```
function stopAtNext(obj)    %  UI
    %    Return directly if the train will stop at the next station
    if obj.nextDest == obj.nextStation
        %    Branch - Tcover 1.2.13.1
        return
    end
    newTarget = obj.environment.getStationPosition(obj.nextStation);
    if newTarget - obj.location < obj.distToStop(obj.speed)
        %    Branch - Tcover 1.2.13.2
        obj.controller.raiseMessage("Too late! The train cannot stop at next station!");
        return
    end
    %    Branch - Tcover 1.2.13.3
    obj.destinations = [obj.nextDest, obj.destinations];    %  Store old destination
    obj.nextDest = obj.nextStation;    %  Update the destination
    obj.targetLocation = obj.environment.getStationPosition(char(obj.nextDest));    %  Update the target locations
end
```

The branch 1.2.13.3 is covered in the functional test, so we just need to test 1.2.13.1, 1.2.13.2.

- Test Case T1.2.13.1
  Coverage Item: Tcover 1.2.13.1.
  State: obj.nextStation = "S1", obj.nextDest = "S1".

Expected Output: The function should return directly without any change.
Test Result: passed.

- Test Case T1.2.13.2
  Coverage Item: Tcover 1.2.13.2.
  State: obj.speed = 300, obj.location = 4700, obj.nextStation = “S1”, obj.nextDest = “S2”.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Coverage: 3/3 = 100% (2 in the unit test, 1 in the functional test).

T1.2.14: Test accelerateToCruisingSpeed()

```
function accelerateToCruisingSpeed(obj)    %  UI
    switch obj.state
        case 0    %    Start if it is stopped
            %    Branch - Tcover 1.2.14.1
            eval(['obj.', 'planSchedule.', char(obj.currStation), 'LeaveTime', '=', num2str(obj.controller.hSystem.time), ';']);
        case 1    %    Invalid if it is accelerating
            %    Branch - Tcover 1.2.14.2
            if obj.acceleration == obj.maxAcceToCru
                %    Branch - Tcover 1.2.14.2.1
                obj.controller.raiseMessage("Already accelerating to cruising speed!");
            else
                %    Branch - Tcover 1.2.14.2.2
                obj.controller.raiseMessage("Already accelerating to maximum speed!");
            end
        case 2    %    Two circumstances when it is decelerating
            %    Branch - Tcover 1.2.14.3
            if obj.isStopping == 1    %    The train is decelerating to stop, cannot accelerate!
                %    Branch - Tcover 1.2.14.3.1
                obj.controller.raiseMessage("Cannot accelerate! The train is decelerating to stop!");
            else    %    The train is decelerating to avoid collison, cannot accelerate, raise error!
                %    Branch - Tcover 1.2.14.3.2
                obj.controller.raiseMessage("Cannot accelerate! The train is decelerating to avoid collision!");
            end
        case 3    %    Invalid if it is running with cruising speed
            %    Branch - Tcover 1.2.14.4
            if obj.speed == obj.cruSpeed
                %    Branch - Tcover 1.2.14.4.1
                obj.controller.raiseMessage("Already running with cruising speed!");
            else%    Branch - Tcover 1.2.14.4.2
                obj.controller.raiseMessage("Cannot accelerate! The train is avoiding collision!");
            end
        case 4    %    Invalid if it is running with maximum speed
            %    Branch - Tcover 1.2.14.5
            obj.controller.raiseMessage("Already running with maximum speed!");
    end
end
```

The branch 1.2.14.1 is covered in the functional test, so we need to test 1.2.14.2 – 1.2.14.5.

- Test Case T1.2.14.1
  Coverage Item: Tcover 1.2.14.2.1.
  State: obj.state = 1, obj.acceleration = obj.maxAcceToCru.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.14.2
  Coverage Item: Tcover 1.2.14.2.2.
  State: obj.state = 1, obj.acceleration = obj.maxAcceToMax.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.14.3
  Coverage Item: Tcover 1.2.14.3.1.
  State: obj.state = 2, obj.isStopping = 1.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.14.4
  Coverage Item: Tcover 1.2.14.3.2.
  State: obj.state = 2, obj.isStopping = 0.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.14.5
  Coverage Item: Tcover 1.2.14.4.1.
  State: obj.state = 3, obj.speed = obj.cruSpeed.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.14.6
  Coverage Item: Tcover 1.2.14.4.2.
  State: obj.state = 3, obj.speed = 100, obj.cruSpeed = 200.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.14.7
  Coverage Item: Tcover 1.2.14.5.
  State: obj.state = 4.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Coverage: 8/8 = 100% (7 in the unit test, 1 in the functional test).

T1.2.15: Test accelerateToMaximumSpeed()

```
function accelerateToMaximumSpeed(obj)    %  UI
    switch obj.state
        case 0    %    Start if it is stopped
            %    Branch - Tcover 1.2.15.1
            obj.toMaximum = 1;
            eval(['obj.', 'planSchedule.', char(obj.currStation), 'LeaveTime', '=', num2str(obj.controller.hSystem.time), ';']);
        case 1    %    set the flag if it is accelerating to cruising speed, raise a warning otherwise
            %    Branch - Tcover 1.2.15.2
            if obj.acceleration == obj.maxAcceToCru
                %    Branch - Tcover 1.2.15.2.1
                obj.toMaximum = 1;
            else
                %    Branch - Tcover 1.2.15.2.2
                obj.controller.raiseMessage("Already accelerating to maximum speed!");
            end
        case 2    %    Two circumstances when it is decelerating, both invalid
            %    Branch - Tcover 1.2.15.3
            if obj.isStopping == 1    %    The train is decelerating to stop, cannot accelerate!
                %    Branch - Tcover 1.2.15.3.1
                obj.controller.raiseMessage("Cannot accelerate! The train is decelerating to stop!");
            else    %    The train is decelerating to avoid collison, cannot accelerate, raise error!
                %    Branch - Tcover 1.2.15.3.2
                obj.controller.raiseMessage("Cannot accelerate! The train is decelerating to avoid collision!");
            end
        case 3    %    begin accelerating if it is running with cruising speed and the distance is enough
            %    Branch - Tcover 1.2.15.4
            if obj.speed < obj.cruSpeed
                %    Branch - Tcover 1.2.15.4.1
                obj.controller.raiseMessage("Cannot accelerate! The train is avoiding collision!");
            elseif obj.distFromCruToMax <= obj.targetLocation - obj.location - obj.distToStop(obj.maxSpeed)
                %    Branch - Tcover 1.2.15.4.2
                obj.toAcceToMax(obj.maxAcceToMax);
            else
                %    Branch - Tcover 1.2.15.4.3
                obj.controller.raiseMessage("Cannot accelerate! The train will decelerate to stop soon!");
            end
        case 4    %    Invalid if it is running with maximum speed
            %    Branch - Tcover 1.2.15.5
            obj.controller.raiseMessage("Already running with maximum speed!");
    end
end
```

The branch 1.2.15.4.2 is covered in the functional test, so we need to test 1.2.15.1 – 1.2.15.3, 1.2.15.4.1, 1.2.15.4.3, 1.2.15.5.

- Test Case T1.2.15.1
  Coverage Item: Tcover 1.2.15.1.
  State: obj.state = 1, obj.speed = 0.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.15.2
  Coverage Item: Tcover 1.2.15.2.1.
  State: obj.state = 1, obj.acceleration = obj.maxAcceToCru, obj.toMaximum = 0.
  Expected Output: obj.toMaximum = 1.

Test Result: passed.

- Test Case T1.2.15.3
  Coverage Item: Tcover 1.2.15.2.2
  State: obj.state = 1, obj.acceleration = obj.maxAcceToMax.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.15.4
  Coverage Item: Tcover 1.2.15.3.1
  State: obj.state = 2, obj.isStopping = 1.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.15.5
  Coverage Item: Tcover 1.2.15.3.2
  State: obj.state = 2, obj.isStopping = 0.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.15.6
  Coverage Item: Tcover 1.2.15.4.1
  State: obj.state = 3, obj.speed = 100, obj.cruSpeed = 200.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.15.7
  Coverage Item: Tcover 1.2.15.4.3
  State: obj.state = 3, obj.speed = 200, obj.cruSpeed = 200, obj.location = 4600, obj.targetLocation = 4800.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Case T1.2.15.8
  Coverage Item: Tcover 1.2.15.5.
  State: obj.state = 4.
  Expected Output: The error message should be raised.
  Test Result: passed.

- Test Coverage: 9/9 = 100% (8 in the unit test, 1 in the functional test).

T1.2.16: Test distToStop()

```
function distance = distToStop(obj, speed)    %  helper function
    digits(10);
    dist1 = 0;
    currSpeed = speed;
    if currSpeed > obj.cruSpeed
        %    Branch - Tcover 1.2.16.1
        t1 = (obj.cruSpeed - currSpeed)/obj.maxDeceToCru;
        dist1 = (obj.cruSpeed + currSpeed)/2 * t1;
        dist1 = vpa(dist1);
        currSpeed = obj.cruSpeed;
    end
    t2 = -currSpeed / obj.maxDeceToZero;
    dist2 = currSpeed / 2 * t2;
    distance = dist1 + dist2;
    distance = vpa(distance);
end
```

The branch 1.2.16.1 is covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.17: Test timeToStop()

```
function time = timeToStop(obj, speed)    %  helper function
    digits(10);
    t1 = 0;
    currSpeed = speed;
    if currSpeed > obj.cruSpeed
        %    Branch - Tcover 1.2.17.1
        t1 = (obj.cruSpeed - currSpeed)/obj.maxDeceToCru;
        currSpeed = obj.cruSpeed;
    end
    t2 = -currSpeed / obj.maxDeceToZero;
    time = t1 + t2;
    time = vpa(time);
end
```

The branch 1.2.17.1 is covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.2.18: Test timeToStop()

```
function distance = distToCru(obj, speed, acceleration)    %  helper function
    distance = 0; %#ok<NASGU>
    if speed < obj.cruSpeed
        %    Branch - Tcover 1.2.18.1
        time = (obj.cruSpeed - speed) / acceleration;
        distance = (obj.cruSpeed + speed) / 2 * time;
    else
        %    Branch - Tcover 1.2.18.2
        time = (speed - obj.cruSpeed) / acceleration;
        distance = (obj.cruSpeed + speed) / 2 * time;
    end
end
```

The branches 1.2.18.1 – 1.2.18.2 are covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.2.19: Test needAvoiding()

```
function avoid = needAvoiding(obj)    %  helper function
    avoid = 0;
    station = obj.controller.findStation(obj.currStation);
    if station.passingTrain{1}(1) == 'G' || (station.passingTrain{1}(1) == 'D' && obj.name(1) == 'K')    %  all wait G passing and K wait D passing
        %    Branch - Tcover 1.2.19.1
        passingTrain = obj.controller.findTrain(station.passingTrain);
        %  don't need to avoid if passing train is the same type and has passed
        if passingTrain.location > obj.location && abs(passingTrain.location - obj.location) > 1e-6 && obj.name(1) == station.passingTrain{1}(1)
            %    Branch - Tcover 1.2.19.1.1
            avoid = 0;
        else
            %    Branch - Tcover 1.2.19.1.2
            avoid = 1;
        end
    end
end
```

The branches 1.2.19.1.1 – 1.2.19.1.2 are covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.2.20: Test Gbehind()

```
function wait = Gbehind(obj)    %  helper function
    wait = 0;
    for i=1:obj.environment.trainNum
        train = obj.environment.trains(i);
        if train.name(1) == 'G' && train.location <= obj.environment.getStationPosition('S3')...
                && train.location > obj.environment.getStationPosition('S1')
            %    Branch - Tcover 1.2.20.1
            wait = 1;
            return
        end
    end
    %    Branch - Tcover 1.2.20.2
end
```

The branches 1.2.20.1 – 1.2.20.2 are covered in the functional test, so we don't have additional testcases to cover this.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.3: Station Unit Test

T1.3.1: Test updateState()

```
function updateState(obj, period)
    if obj.inSwitchTime ~= 0
        %    Branch - Tcover 1.3.1.1
        obj.inSwitchTime = obj.inSwitchTime - period;
    end

    if obj.outSwitchTime ~= 0
        %    Branch - Tcover 1.3.1.2
        obj.outSwitchTime = obj.outSwitchTime - period;
    end

    if obj.inSwitchTime < 0.001
        %    Branch - Tcover 1.3.1.3
        obj.inSwitchTime = 0;
    end

    if obj.outSwitchTime < 0.001
        %    Branch - Tcover 1.3.1.4
        obj.outSwitchTime = 0;
    end

    if obj.inSwitchTime == 0
        %    Branch - Tcover 1.3.1.5
        obj.inSwitchLock = 1;
    end

    if obj.outSwitchTime == 0
        %    Branch - Tcover 1.3.1.6
        obj.outSwitchLock = 1;
    end

    %    Rearrange the leave time of trains
    obj.rearrangeTrainLeaveTime();
end
```

The branches 1.3.1.1 – 1.3.1.6 are covered in the functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 6/6 = 100% (0 in the unit test, 6 in the functional test).

T1.3.2: Test rearrangeTrainLeaveTime()

```
function rearrangeTrainLeaveTime(obj)
    trainList = [];
    for i=1:obj.totalRailNum - 1
        if obj.rails{i+1}.currentTrainName ~= "None"
            %    Branch - Tcover 1.3.2.1
            trainList = [obj.hController.findTrain(obj.rails{i+1}.currentTrainName), trainList]; %#ok<AGROW>
        end
    end
    if length(trainList) >= 2
        %    Branch - Tcover 1.3.2.2
        sortedTrainList = obj.hController.sortByTypeAndLeaveTime(trainList);
        firstTrain = sortedTrainList(1);
        leaveTime = eval(['firstTrain.planSchedule.', char(firstTrain.currStation), 'LeaveTime']);
        for i=2:length(sortedTrainList)
            train = sortedTrainList(i);
            prevLeaveTime = eval(['train.planSchedule.', char(train.currStation), 'LeaveTime']);
            newLeaveTime = max(prevLeaveTime, leaveTime+5);
            eval(['train.planSchedule.', char(train.currStation), 'LeaveTime=newLeaveTime', ';']);
            leaveTime = newLeaveTime;
        end
    end
end
```

The branches 1.3.2.1 – 1.3.2.2 are covered in the functional test, so we don't have additional testcases to cover these branches.

- Test Coverage: 2/2 = 100% (0 in the unit test, 2 in the functional test).

T1.4: System Unit Test

T1.4.1: Test updateState()

```
function updateState(obj, ~, ~)
    %    Update the state of environment and display
    obj.time = obj.time + obj.t.Period;
    allFinish = obj.hController.updateState(obj.time, obj.t.Period)
    obj.hDisplay.updateState;
    if allFinish == 1
        %    Branch - Tcover 1.4.1.1
        obj.stopSimulate();
    end
end
```

The branch 1.4.1.1 is covered in the functional test, so we don't have additional tests to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.4.2: Test beginSimulate()

```
%    1.4.2
function beginSimulate(obj)
    %    Begin simulate
    start(obj.t);
    %    Statement - Tcover 1.4.2.1
end
```

The function has no branches and it is covered in functional test, so we don't have additional tests to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.4.3: Test stopSimulate()

```
function stopSimulate(obj)
    %    Begin simulate
    stop(obj.t);
    %    Statement - Tcover 1.4.3.1
end
```

The function has no branches and it is covered in functional test, so we don't have additional tests to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.5: Display Unit Test

T1.5.1: Test updateState()

The source code of this function is too long to insert here, so we just demonstrate our test process. The branches can be found in *updateState()* function in *Display.mlapp* file.

The branched 1.5.1.1.1 – 1.5.1.1.7 are covered in the functional test, so we don't have additional tests to cover these branches.

- Test Coverage: 7/7 = 100% (0 in the unit test, 7 in the functional test).

T1.5.2: Test moveTrain()

```
function moveTrain(app, index, position)
    eval(['app.Train', num2str(index), 'Label.Position(1)=app.railwayStart+position*app.lengthPerMeter-app.Train', num2str(index), 'Label.Position(3)/2', ';']);
    eval(['app.Train', num2str(index), 'Image.Position(1)=app.railwayStart+position*app.lengthPerMeter-app.Train', num2str(index), 'Image.Position(3)/2', ';']);
    %    Statement - Tcover 1.5.2.1
end
```

The function has no branches and it is covered in functional test, so we don't have additional tests to cover this.

- Test Coverage: 1/1 = 100% (0 in the unit test, 1 in the functional test).

T1.5.3: Test showRailState()

```
function showRailState(app, index)
    station = app.hEnvironment.stations{index};
    currentInRailIndex = station.mainInRailIndex;
    currentOutRailIndex = station.mainOutRailIndex;
    eval(['app.S', num2str(index), 'In', num2str(currentInRailIndex), '.Value=1', ';']);
    eval(['app.S', num2str(index), 'Out', num2str(currentOutRailIndex), '.Value=1', ';']);
    inSwitchLock = station.inSwitchLock;
    outSwitchLock = station.outSwitchLock;
    if inSwitchLock == 0
        %    Branch - Tcover 1.5.3.1
        eval(['app.S', num2str(index), 'InSwitchLamp.Color = [1, 0, 0]', ';']);
    else
        %    Branch - Tcover 1.5.3.2
        eval(['app.S', num2str(index), 'InSwitchLamp.Color = [0.5, 0.5, 0.5]', ';']);
    end

    if outSwitchLock == 0
        %    Branch - Tcover 1.5.3.3
        eval(['app.S', num2str(index), 'OutSwitchLamp.Color = [1, 0, 0]', ';']);
    else
        %    Branch - Tcover 1.5.3.4
        eval(['app.S', num2str(index), 'OutSwitchLamp.Color = [0.5, 0.5, 0.5]', ';']);
    end

    for i=2:station.totalRailNum
        eval(['app.S', num2str(index), 'Rail', num2str(i-1), '.Value=station.rails{i}.currentTrainName', ';']);
    end
end
```

The branched 1.5.3.1 – 1.5.3.4 are covered in the functional test, so we don't have additional tests to cover these branches.

- Test Coverage: 4/4 = 100% (0 in the unit test, 4 in the functional test).

T1.6: TrainPanel Unit Test

T1.6.1: Test updateState()

```
function updateState(app)
    app.SpeedGauge.Value = app.train.speed;
    app.NextStationEditField.Value = app.train.nextDest;
    app.CurrentSpeedEditField.Value = num2str(app.train.speed);
    app.AccelerationEditField.Value = num2str(app.train.acceleration);
    app.LocationEditField.Value = num2str(app.train.location);
    switch app.train.state
        case 0
            %    Branch - Tcover 1.6.1.1
            app.CurrentStateEditField.Value = 'stopped';
        case 1
            %    Branch - Tcover 1.6.1.2
            app.CurrentStateEditField.Value = 'accelerating';
        case 2
            %    Branch - Tcover 1.6.1.3
            app.CurrentStateEditField.Value = 'decelerating';
        case 3
            %    Branch - Tcover 1.6.1.4
            if app.train.speed == app.train.cruSpeed
                %    Branch - Tcover 1.6.1.4.1
                app.CurrentStateEditField.Value = 'cruising speed';
            else
                %    Branch - Tcover 1.6.1.4.2
                app.CurrentStateEditField.Value = 'avoiding collision';
            end
        case 4
            %    Branch - Tcover 1.6.1.5
            app.CurrentStateEditField.Value = 'maximum speed';
    end
end
```

The branched 1.6.1.1 – 1.6.1.5 are covered in the functional test, so we don't have additional tests to cover these branches.

- Test Coverage: 6/6 = 100% (0 in the unit test, 6 in the functional test).

T2: Integration Test

T2.1: Display Panel UI + Train Panel UI + Controller + Environment

T2.1.1: Test "Begin simulate"

```
function beginSimulateTest(tc)
    %    test begin simulate function
    tc.press(tc.display.LoadInitialScheduleButton);
    pause(5);
    tc.press(tc.display.BeginSimulateButton);
end
```

- Test Case T2.1.1
  Operation: Boot the system with "final_test_2.txt", press "Load Initial Schedule" and press "Begin Simulate" button.
  Expected Behavior: The trains begin moving along the railway, the information of trains and stations, the real schedule and delay are updated for each time step.
  Test Result: passed.

T2.1.2: Test “Stop the train at next station”

```
function result = stopAtNextStationTest(tc)
    %    test stop at next station function
    result = 0;
    press = 0;
    tc.press(tc.display.LoadInitialScheduleButton);
    pause(5);
    tc.press(tc.display.BeginSimulateButton);
    pause(30);
    trainList = tc.environment.trains;
    while press == 0
        for i=1:tc.environment.trainNum
            train = trainList(i);
            if train.nextDest ~= train.nextStation
                tc.press(train.panel.StopatnextstationButton);
                press = 1;
                break
            end
        end
    end
end
```

- Test Case T2.1.2
  Operation: Boot the system with “final_test_4.txt”, press “Load Initial Schedule”, press “Begin Simulate” button and press “Stop at next station” button in K1’s panel.
  Expected Behavior: K1’s next destination is updated to “S1” and it stops at S1.
  Test Result: passed.

T2.1.3: Test “Accelerate the train to cruising speed” button.

```
function acceToCruSpeedTest(tc)
    %    test accelerate to cruising speed function
    press = 0;
    tc.press(tc.display.LoadInitialScheduleButton);
    pause(5);
    tc.press(tc.display.BeginSimulateButton);
    pause(30);
    trainList = tc.environment.trains;
    while press == 0
        for i=1:tc.environment.trainNum
            train = trainList(i);
            if train.state == 0
                tc.press(train.panel.AcceleratetocruisingspeedButton);
                press = 1;
                break
            end
        end
    end
end
```

- Test Case T2.1.3
  Operation: Boot the system with “final_test_3.txt”, press “Load Initial Schedule”, press “Begin Simulate” button. Wait for 10 seconds and press “Accelerate to cruising speed” button in G1’s panel.
  Expected Behavior: G1 starts from the start station and accelerates to cruising speed.
  Test Result: passed.

T2.1.4: Test “Accelerate the train to maximum speed”

```
function acceToMaxSpeedTest(tc)
    %    test accelerate to maximum speed function
    press = 0;
    tc.press(tc.display.LoadInitialScheduleButton);
    pause(5);
    tc.press(tc.display.BeginSimulateButton);
    pause(30);
    trainList = tc.environment.trains;
    while press == 0
        for i=1:tc.environment.trainNum
            train = trainList(i);
            if train.state == 3
                tc.press(train.panel.AcceleratetomaximumspeedButton);
                press = 1;
                break
            end
        end
    end
end
```

- Test Case T2.1.4
  Operation: Boot the system with “final_test_3.txt”, press “Load Initial Schedule”, press “Begin Simulate” button. Wait for 10 seconds and press “Accelerate to maximum speed” button in K1’s panel.
  Expected Behavior: K1 accelerates to maximum speed.
  Test Result: passed.

T2.1.5: Test “Switch a rail”

```
function switchRailTest(tc)
    %    test switch rail function
    tc.press(tc.display.LoadInitialScheduleButton);
    pause(5);
    tc.press(tc.display.BeginSimulateButton);
    pause(30);
    tc.press(tc.display.S1In1);
    pause(5);
    tc.press(tc.display.S1Out2);
end
```

- Test Case T2.1.5
  Operation: Boot the system with “final_test_1.txt”, press “Load Initial Schedule”, press “Begin Simulate” button. Wait for 10 seconds and press “1” button in S1-in and “2” button in S1-out.

Expected Behavior: The entry rail of S1 switches to 1 and the exit rail of S1 switches to 2. The switch lamps are on during switching.
Test Result: passed.

T3: Functional Test

T3.1: Use Case "Auto control"

- Test Case T3.1.1
  Operation: Boot the system with "final_test_1.txt", press "Load Initial Schedule", press "Begin Simulate" button. Wait until the simulation is finished.
  Expected Behavior: All train should arrive the terminal without collision and within the max delay time.
  Test Result: passed.

- Test Case T3.1.2
  Operation: Boot the system with "final_test_2.txt", press "Load Initial Schedule", press "Begin Simulate" button. Wait until the simulation is finished.
  Expected Behavior: All train should arrive the terminal without collision and within the max delay time.
  Test Result: passed.

- Test Case T3.1.3
  Operation: Boot the system with "final_test_3.txt", press "Load Initial Schedule", press "Begin Simulate" button. Wait until the simulation is finished.
  Expected Behavior: All train should arrive the terminal without collision and within the max delay time.
  Test Result: passed.

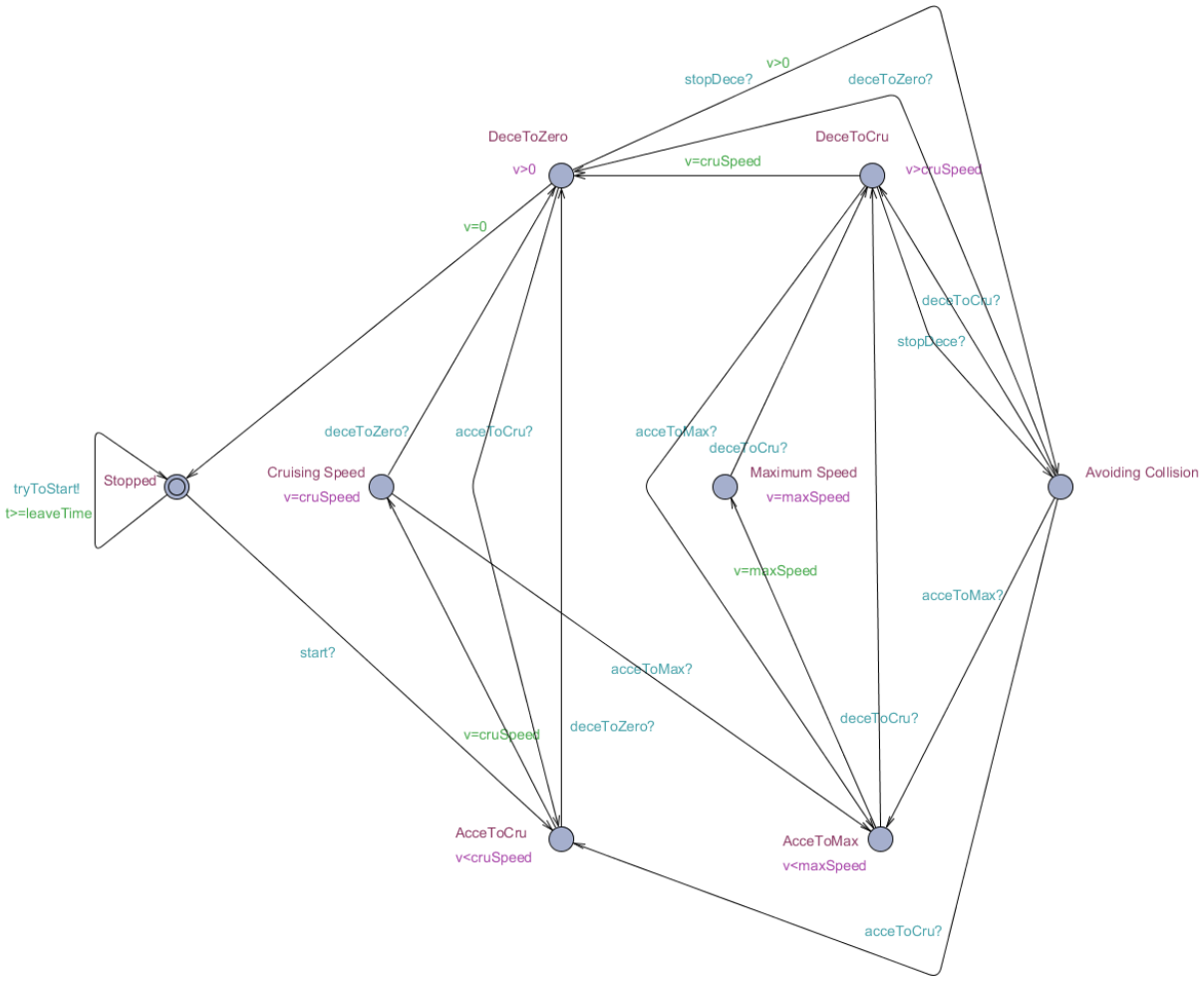
- Test Case T3.1.4
  Operation: Boot the system with "final_test_4.txt", press "Load Initial Schedule", press "Begin Simulate" button. Wait until the simulation is finished.
  Expected Behavior: All train should arrive the terminal without collision and within the max delay time.
  Test Result: passed.

T3.2: Use Case "Manual control"

- Test Case T3.2.1
  Operation: Boot the system with "final_test_1.txt", press "Load Initial Schedule", press "Begin Simulate" button. Press "Stop at next station", "Accelerate to cruising speed", "Accelerate to maximum speed" and switch a rail randomly.
  Expected Behavior: The simulation should react according to the input commands. Some collisions may happen.
  Test Result: passed.

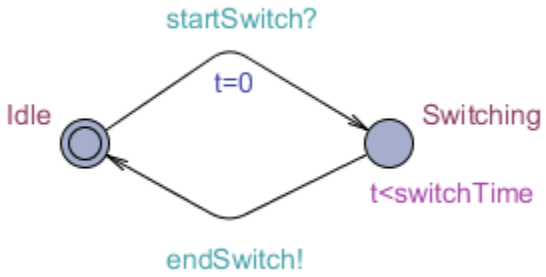
T4: Model Checking

Train Model



The trains are stopped in the beginning and try to start when the leave time is passed. When receiving the start signal from the controller, it begins accelerating to the cruising speed. It may decelerate or continue accelerating to maximum speed depending on the controller signal. When a train is decelerating, it can also begin accelerating if receiving a acceleration signal. When the train is decelerating to avoid collision, it may run with a constant speed lower than cruising speed or maximum speed, that's called "Avoiding Collision" state. When a train decelerates to 0 speed, it becomes stopped.

Rail Model



A rail is idle in the beginning. When receiving a switch signal, it begins switching for the switch time, during which no switch request is allowed. After switching, the rail returns to normal idle state.

Controller Model

The automated control system is too complex to be implemented in the UPPAAL model, so we check the properties according to the result of MATLAB simulation result.

Check Property

P1: Completeness

- P1.1
  Description: In automated control mode, all trains should arrive the terminal.
  Result: passed.

P2: Safety

- P2.1
  Description: In automated control mode, there is no collision between each pair of trains.
  Result: passed.

- P2.2
  Description: In automated control mode, the speed of train cannot exceed its maximum speed.
  Result: passed.

- P2.3
  Description: In automated control mode, the speed of train cannot exceed its maximum speed.
  Result: passed.

- P2.4
  Description: In automated control mode, the train cannot start before the plan leave time.
  Result: passed.

- P2.5
  Description: In both automated control mode and manual control mode, the rail cannot be switched during switching.
  Result: passed.

- P2.6
  Description: In both automated control mode and manual control mode, the rail cannot be switched when a running train is passing.
  Result: passed.

P3: Delay

- P3.1
  Description: In automated control mode, all trains should arrive the terminal within their max delay.
  Result: passed.

P4: Permission

- P4.1
  Description: In some conditions, the controller should reject some illegal commands by user.
  Result: passed.

P5: Display

- P5.1
  Description: During simulation, the state of trains and stations should be updated in real time.
  Result: passed.

- P5.2
  Description: In both automated control mode and manual control mode, the real schedule and accumulated delay should be updated in real time.
  Result: passed.