Gent Technologiecampus, KU Leuven

Master of Science - Algoritmen voor beslissingsondersteuning

Project final report

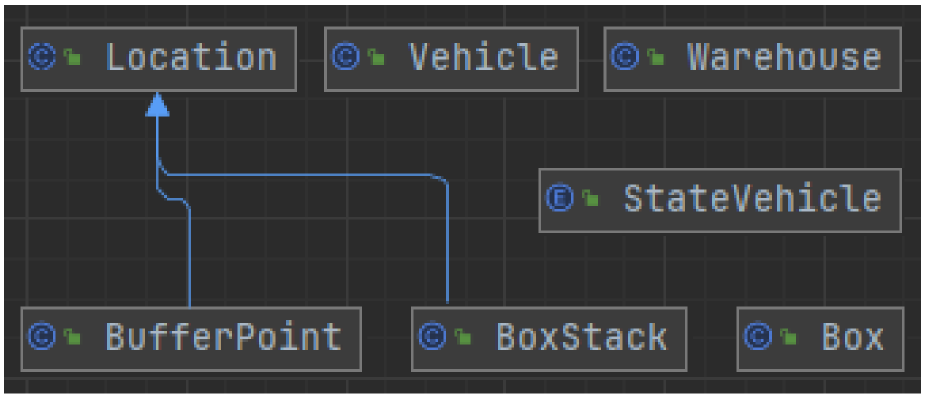
Milan Schollier

September 1, 2024

1 Intro

For this project the goal was to simulate one or more vehicles in a warehouse that has to move several boxes between bufferpoints or boxstacks. This as efficient as possible of course.

2 Design



In this project everything is based on a tick system. This guarantees the possibility to check on concurrency errors.

Warehouse & Box

Warehouse only keeps necessary data like array lists of locations, vehicles, boxes, move requests,.... The only function it has is searching the nearest box stack where a box can be moved out of the way. Box only keeps it's id, current and goal location.

Location

This class is an abstract class but with a lot of the used functions in vehicle. For example a $loadBox()$ to add a box to this location.

Vehicle & State vehicle

For the vehicle we use the class state vehicle which can be no target, load or unload. Also the vehicle can lock and occupy a box stack. Lock is always set for a load operation and only turned off if the claimed box is picked up. Occupy is true if the (un)loading operation is being performed.
However the timing of when locations get locked or occupied do need to be optimized. This to ensure in further improvements that boxes that don't need to be moved can temporally be on top of other boxes that do need to go out.

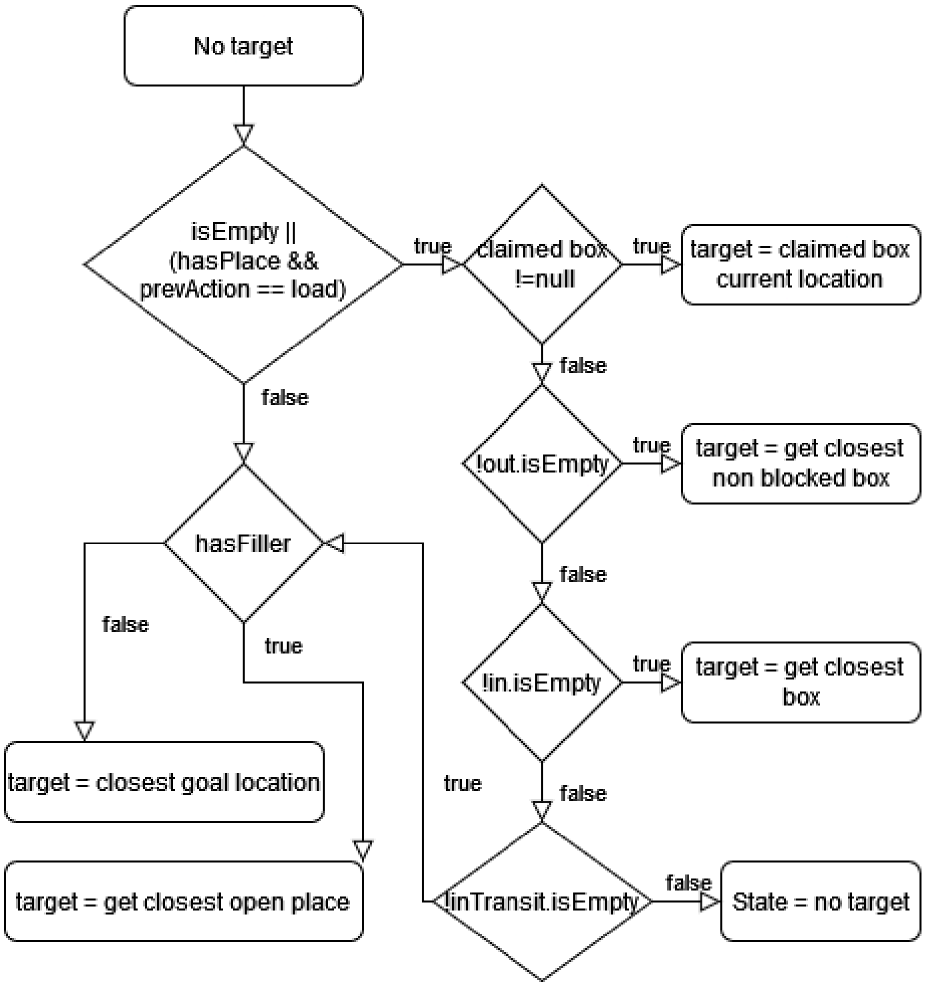

No target
isEmpty || (hasPlace && prevAction == load)
true
claimed box !=null
true
target = claimed box current location
false
!out.isEmpty
true
target = get closest non blocked box
false
!in.isEmpty
true
target = get closest box
false
!inTransit.isEmpty
false
State = no target
true
false
hasFiller
false
true
target = closest goal location
target = get closest open place

Figure 1: Activity diagram for Vehicle

Results

Instance	Tick()
I3_3_1	880
I3_3_1_5	180
I10_10_1	3040
I15_16_1_3	2630
I20_20_2_2_8b2	3260
I30_100_1_1_10	50580
I30_100_3_3_10	11610
I30_200_3_3_10	13210
I100_50_2_2_8b2	23380
I100_120_2_2_8b2	51720
I100_500_3_1_20b2	202880
I100_500_3_5_20	83740
I100_800_1_1_20b2	N/A
I100_800_3_1_20 b2	N/A

SWOT analysis

Strengths

- Because of the box by box approach we can always check for closest location.
- In the initializing phase we check how many spots we need for the incoming boxes. And keep this much space open. This prevents leveling during the process.
- The lock ensures other vehicles don't interrupt digging out a box.

Weaknesses

- The move requests happen in two stages. First everything out then everything in. This causes some inefficiencies.
- If the vehicle searches for the nearest stack with place, it looks if it is locked but not if it will still be locked once arrived. But this rarely occurs.
- In the last two instances the program panics. This because when a box is being dug out, the boxes above can only go on stacks without other boxes that still need to go out. But in these two instances every stack has a box that needs to be moved.

Opportunities

- Do all requests at the same time this would speed up the process because the vehicle could pick up boxes from the buffer point after sending boxes out. This could cause some problems with overfull stacks.

- A scheduling tree could be implemented. This could also make multiple solutions to then choose the best solution based on time. However this could take significant resources to calculate when faced with big datasets. Also would this require a redesign of a big portion of the code.

Threats

- If there are more requests to move in boxes then places available. This will cause the system to keep looking for places.
- If there is no stack without a box that still needs to be moved. The vehicle won't find a stack. This currently causes the program to halt.