|  |
| --- |
| Carleton University |
| Bi-directional Pedestrian Flow Modeling and Simulation with CELL-DEVS |
| SYSC5104 |

|  |
| --- |
| Wang Xiang 100680824  12/20/2010 |

# Introduction:

This project is the extension of the second course assignment of SYSC5104. Please skip to section 4 if the reader has seen the documentation and the implementation of the second assignment. Also, this report assumes the reader has prior knowledge about DEVS and cellDEVS. However, there is a brief background description included in section 2 of this report.

In the second assignment, a published paper with a title of “An Improved Cellular Automation Model for Urban Walkway Bi-directional Pedestrian Flow” [1] was studied and implemented with cellDEVS based on the suggested model and rules. The original work tended to study the behaviours of pedestrians with different walking speed and different actions based on some probabilities. There were 8 cellular automata rules the article suggested as listed below:

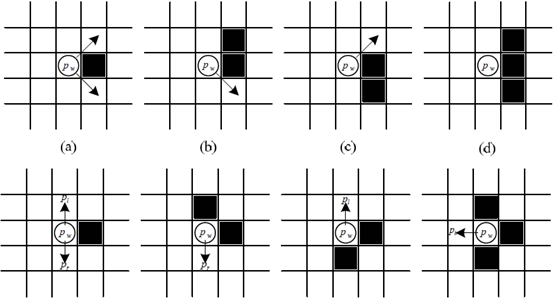


Figure : Cell-DEVS rules for pedestrians

These 8 rules can be further split into more rules because each black square can represent an obstacle or a pedestrian of either direction. With the provided 8 rules and their expansions, there were a huge amount of different cases to be considered for the CELL-DEVS implementation. It is almost impossible to cover all the combinations of the rules in the implementation. Thus the implementation of this project only used a few of the rules with some modification. Please see next section for the definitions of the basic model of the project.

# Background

The implementation of this project and assignment 2 are both based on the knowledge of cellDEVS, which is an extension of DEVS. DEVS stands for Discrete Event System Specification. It is a high level modeling and simulation framework and language. It features an introduction of time delays between discrete state changes of the modeled system, and also features a mechanism of differentiating the internal transitions versus the external causes for transition between states. In DEVS simulations, internal time delay and internal transitions happen all the time. After each time the internal time delay is consumed, an output may or may not get generated, and then an internal transition happens and starts the next cycle of internal process. During this internal process, if there is any external input interruption kicks in at any time, the simulation switches to external transition phase according to external transition rules.

DEVS is good for flow of discrete events simulation, but not good for spacial events simulation. Therefore cellDEVS is introduced. cellDEVS is an extension of DEVS that can be used to model cellular automata, which the modeled system can be represented as a cell space. Most of the time the locations and the state values of the cells are the interests of the simulation with cellDEVS. Each cell is basically an atomic DEVS model. Each cell considers its specified neighbourhood to figure out its transition according to a set of pre-defined rules. The beauty of cellDEVS is that in every execution cycle, not all the cells in the modeled cell space will get invoked. The cellDEVS simulator will only invoke the necessary cells and their inverse neighbourhood in any cycle. This saves a big amount of execution time in a simulation.

For more information on DEVS and cellDEVS, please see “Discrete-Event Modeling and Simulation, A Practitioner’s Approach”. [2]

# Basic Definition Of The Model:

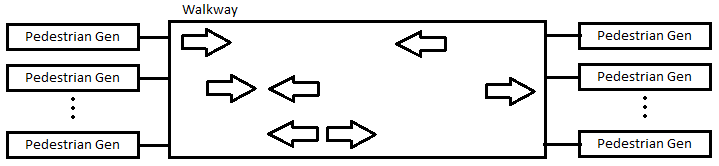
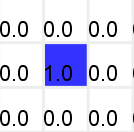
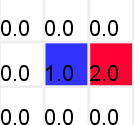


Figure : Cell-DEVS coupled model for the project

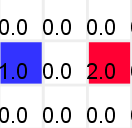
Figure 2 shows the coupled model definition of the project. The “Pedestrian Gen” blocks are CD++ built in random sequence generators. The “Walkway” is a Cell-DEVS model of size (8\*20) with the top most and the bottom most rows represent the walls of the walkway. Pedestrians are generated at the left and right ends of the walkway. The goal of the project is to program the “pedestrians” to travel to the opposite end of the walkway without collision. The speeds of all “pedestrians” are modeled at a constant rate, and everyone travels at that same speed. To do this, some rules are used to regulate the behaviours of the pedestrians as explain below in detail:

**Rule 1: Moving forward rule.**

If the cell in front of a “pedestrian” from either direction is empty, and no other “pedestrian” from the opposite direction is located two cells ahead, move forward. This is the highest priority rule. No action from other rules can take the empty square of moving forward rule.

**Rule 2: Side stepping when straight movement is blocked.**

If a “pedestrian” is blocked by the opposite side “pedestrian”, both “pedestrians” try to step to the right if that cell is empty. If the “pedestrian” is blocked by another one traveling to the same direction, simply do nothing but wait the other one to move first. This is the second highest priority rule.

**Rule 3: Side stepping when trying to take the same cell for straight movement.**

If two “pedestrians” from opposite side trying to take the same empty cell, both step to the right. If they are travelling in the same direction, Rule 1 takes care of it. This is the third highest priority rule.

**Rule 4: Side step to the left if the right side is a wall.**

For the cases where side step to the right is impossible because they are by a wall, they the “pedestrians” try to step to the left if there is an empty cell. This rule applies to both side-stepping when blocked or trying to take the same cell.

These rules are subset or a generalization of the 8 rules provided in the literature [1] according to human natural way to walking. Moving straight forward is the most fundamental and most important form of traveling. Pedestrians change lane only when necessary. Also, people tend to step to the right when encounter another pedestrian traveling the opposite direction. Given everyone travel at same speed, it does not make sense to make a side step when encounter someone with the same direction as the pedestrian.

# Problem Statement and Project Definition

Having the modeled space and the four basic rules understood, an implementation was carried out in the second assignment. The implementation was mostly successful and promising, but not complete. Due to complex nature of the project, the preconditions of cellDEVS rules were extremely complex and lengthy. For any cell attempting to move into another cell, cellDEVS needed to check if the target cell was empty and there was no other cell can potentially move into the same cell in the same cycle. The test needed to be thorough and covers all the cases, so that no conflict can occur. For example, there are multiple causes to block a cell from moving straight forward. Thus it attempts to make a side-step up or down. Then cellDEVS needs to check if the cell above or below is empty, and if there is any other cells can move straight into the empty cell. Also cellDEVS needs to check if there are other cell can sidestep into the empty cell from the other side, again for multiple reasons. Clearly illustrated in the example above, the check procedure was very complicated, and the neighbourhood resulted from this check procedure could be very large. The check procedure could be both execution time expensive and memory storage expensive. Therefore, due to the time constraint for assignment 2, check for sidestepping was not implemented. That means if a cell is attempt to sidestep up or down, assignment 2 only implemented to check if there is any other cell can move straight into the empty cell, but did not check if a cell can sidestep into the empty cell from the other side.

Therefore, very obviously, the major purpose for the second version of project is to find an efficient way to implement this sidesteps check aiming to achieve the following two goals:

* To optimize the execution time
* To optimize the neighbourhood size

In order to optimize the execution time, the implementation needs to avoid having lengthy precondition checks for all the rules. Also, a small neighbourhood helps to optimize execution time. A priority scheme is proposed to apply to different rules. Such that if higher level rules try to take over an empty cell, then lower level rules cannot take that cell. By applying this priority method, higher level priority rules will fill up empty cells first, and then lower level rules don’t need to check higher level cases since it has ready been done. To achieve this, several approaches were considered and examined. Please see next section to find the detailed explanation and experiment results.

# Project Approaches and Experiment Results:

Having the goal of the project understood, the focus of the project is then to figure out an appropriate way to apply the proposed priority scheme. The following subsections outline the conceptual and theoretical plans of implementation, and the results of each approaches.

## 5.1 Approach I: Using state variable

It is clear that in order to apply the priority scheme, a method is needed to indicate which cells have been modified by a level of priority rule. A new version of CD++ was introduced by Professor Wainer. This new version features capability of assigning multiple state variables to each cell. In the older version of CD++, each cell can only contain one state variable. This makes cellDEVS very inconvenient. For example, if cellDEVS is used to model a battlefield game, and each cell represents a soldier in the game. Other than the necessary state variable records the health of the soldier, the soldier may also need to have information of ammo count, type of soldier, strength of the soldier, etc. In older version of CD++, the user needs to have multiple layers to model all these state variables. On the other hand, the new version of CD++ only needs one layer.

This new version of CD++ seems very promising to use for the goal of the project. A state variable for each cell can be used to indicate if such cell has already been modified by higher level priority rules. For example, for each cycle of transition, starting from highest priority rule to the lowest one we set the change indication state variable to be true as soon as a rule modifies the current cell for every cell. Then lower level priority rules checks to see if this indication variable is set to decide whether the lower priority rule needs to be executed.

However after careful examine of this approach, two problems makes this approach not applicable. First, to make this method to work, CD++ needs to reset the change indication state variable to be false for all the cells at the beginning of each execution cycle. This contradicts with the purpose of cellDEVS which avoids evaluate/update quiescent cells. The fundamental reason that causes a need to reset in each cycle is that the change indication variable is not continuous. It does not carry its value to next cycle. One way I tried to fix the problem was to apply kind of alternating bit protocol to the change indication variable. For example, in cycle 1 a value of 1 means current cell has been changed, then in cycle 2 a value of 1 means the current cell has not been changed. In this way no reset is needed anymore. However this still won’t work because not all cells will be changed in one cycle. This means by next cycle, some cells have change indication variable of value 1 indicating a change has been made, while others indicating no change made.

Secondly, and most importantly, this method theoretically speaking won’t work at all. The reason is all the rules of different level of priority happen at the same time. Let’s consider a run of an execution cycle. Assume rule 1 (highest priority level rule) is applicable to the situation and changed the value of the cell, thus the change indication variable is set to true. However, this change does not occur immediately. Instead, this change is scheduled after a time delay. Then being a sequential machine, CD++ moves on to the second rule, and since the change from first rule has not yet occur, then the second rule is also applicable to the situation. Thus another change is scheduled with the same time delay and basically over writes the first rule changes in the queue. This not only did not make the execution simpler/faster, but also reversed the priority level. After the time delay, CD++ will execute the lower level rule action instead of the higher ones. Therefore, this method won’t do any good to the project.

## 5.2 Approach II: Using multiple layers with different time delay on each layer

Observed from the first approach, the priority scheme is basically an order of in which sequence the rules should be executed in TIME. The basic idea is that after higher priority rules made the changes, the lower priority rules only needs to check if the cell it attempts to move to is empty. This is because all other higher level priority rule has already been executed, and thus the only empty cells left are for the current level of priority rule to use. After many implementation experiments performed, the following is the final design of this approach:

Figure 3 illustrates a 4-layer cell space model of the project. Each layer has a size of (8\*20) cells same as the 2D model. Layer has all the information of all the cells resulted from all the rules. Layer2, 3, 4 only contains the information of specific set of rules. Rules that can be executed on layer 4 has the highest priority; layer 3 has the second priority; and layer 2 has the third priority. Having the layers defined, an execution cycle is illustrated in Figure 4 below:

Figure : Layer definition of the model

* **Layer 4: Move straight forward**
* **Layer 3: Sidestep upward**
* **Layer 2: Sidestep downward**
* **Layer 1: Overall space**

Figure : Action sequence of one execution cycle

Figure 4 illustrates the action sequence of one execution cycle. To understand this graph, let us consider a particular run of the execution cycle. To start, at the beginning of each execution cycle, all four layers copy all the cells on layer 4 to their initial state. Since layer 4 contains the overall information of all the cells left from last execution cycle, all four layers reset their initial state to last execution cycle. Then, at time 4, layer 2 and 3 goes to sleep. Layer 4 executes only the move straight rule based on the initial state, and makes the same changes to layer 1 as well. At time 7, both layer 1 and layer 4 will have the changes made by the straight movement rule. At time 8, layer 4 goes to sleep but wakes up layer 3. Then layer 3 executes the set of rules that will only result in a step upwards based on the information on layer 1. Since straight movements are already done on layer 1, there will be no conflict. Again the same changes are also made on layer 1, so that at time 12 layer 1 would have changes made by layer 4 and 3. Lastly, at time 14, layer 2 makes changes that would result in a sidestep downwards based on layer 1. Again there will be no conflict. Therefore, at time 17 or a bit later, layer 1 would have the completed transformation of one execution cycle with no conflict. Then this action sequence will be repeated for the duration of the simulation. If this approach is applicable, then the neighbourhood will have the following shape:

It is a 3 dimensional cell space with 20 cells neighbourhood. The implementation in assignment 2 had a cell neighbourhood of 17 cells. Therefore, with 3 more cells this approach is able to achieve a conflict free simulation. However, with respect to the execution time optimization, this approach does not save much time because now it has 4 layers which three of them will be individually executed in a cycle. Thus this cannot be an optimal execution time design, but might be an optimal neighbourhood design.

Nevertheless, very unfortunately, this approach cannot be implemented in cellDEVS for very obvious reason. Individual layers cannot go to sleep for a period of time. The execution will be continuous in time, and there seems to be no way of getting around this fact. Experiments were performed to assign different time delays to different layers. However the starting of all layers are the same, and copying of all the cells to all the layers are just impossible and wasteful. Therefore, the second approach was again a failure.

Figure : Cell neighbourhood of the second approach

## 5.3 Approach III: Apply priority scheme to directions

Although the second approach did not work out, one key discovery was found during the experiments. The fundamental conflict of sidestepping is due to two cells try to sidestep into the same empty cell from the opposite side. One of the cells has to step upwards, while the other one has to step downwards to result in the conflict. Then instead of applying the priority scheme to the time of execution, apply priorities with respect to the directions of pedestrian movements. For example, the highest priority is assigned to horizontal movements or the straight movements from both directions. For this level of priority, cellDEVS only needs to check if the cell they are going to move in is empty, and there is no other cell attempts to move into the same cell from the other direction. Then assign the second priority to the rules that would result in sidesteps upwards. These rules includes sidestepping to the right if the pedestrian is traveling to the left; also includes sidestepping to the left if the pedestrian is travelling to the right and there is a wall to the right of the pedestrian. This level of priority rules not only need to check if they need to perform a sidestep possibly due to blocked my another cell, they also need to check if level 1 priority rules can happen to the cell they attempt to move to. Lastly, the lowest priority is assigned to the rules that would result in sidesteps downwards. These rules basically are composed of the opposite rules as priority 2 rule set. However, this lowest priority rule set need to check all higher priority rules. If there is another cell can step upwards into the target cell, then the testing cell cannot step downwards to that same cell. It seems the check procedure grows exponentially towards lower priority levels. However, there are only 3 levels in the system, so this design is good.

With the new priority scheme developed, the implementation of the scheme was surprisingly simple and successful. No 3D cell space is needed, nor multiple state variable are required. It is further discovered that the original implementation for assignment 2 already had the overall skeleton for the implementation of the priority scheme. Only changes required to make was to regroup the rules and develop a few more precondition checks for level 3 priority rules. It is even better when later discovered the added precondition checks are the same for all four rules composes the level 3 priority rule set. Therefore only copy and paste work needed to be done for the later 3 rules. With this approach, there are only 5 more cells added to the neighbourhood comparing to the implementation of assignment 2. Please see the next section to see the formal specification. This is an acceptable sacrifice to achieve conflict free simulation. It may seem to have 2 more cells comparing to the second approach, but the fact that the second approach does not work makes 5 more cells optimal. With respect to the execution time aspect, there are indeed a few more checks need to be performed every time when a cell attempts to sidestep downwards. However, comparing this to an implementation without priority scheme at all where all rules need to check all other rules, this approach is much better/faster. Unless there is another way to implement the priority scheme, this approach is optimal in execution time.

# CELL-DEVS Formal Specification:

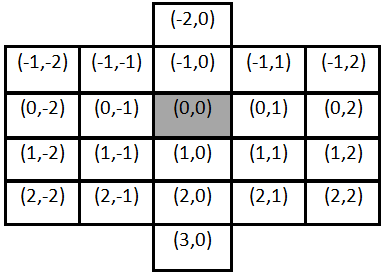


Figure : Neighbourhood definition

**M = <Xlist, Ylist, I, X, Y, ƞ, N, {r,c}, C, B, Z, select>**

**Xlist = {(-1,-2)…(2,-2) (-1,2)…(2,2)};**

**Ylist = {};**

**I = <Px, Py>; with Px = {<X(-1,-2), binary>…<X(2,2),binary>}; Py={};**

**N = {(-2,0), (-1,-2), (-1,-1), (-1,0), (-1,1), (-1,2), (0,-2), (0,-1), (0,0), (0,1), (0,2), (1,-2), (1,-1), (1,0), (1,1), (1,2), (2,-2), (2,-1), (2,0), (2,1), (2,2), (3,0)};**

**Ƞ = 22;**

**r = 8; c = 20;**

**X=Y={0, 1, 2}:**

**B = not-wrapped**

**C = {Cij | i ϵ [1, 6], j ϵ [0, 19]};**

**Z: Inverse neighbourhood of N**

**Select = {(0,1), (0,-1), (-1,0), (1,0)};**

# Project Implementation

This section will present the pseudo code implementation of the rules of the cellular automata. Also the expected behaviours of the rules will be explained.

**Entrance rule:**

Assign a value of 2 to the pedestrians who are walking towards left if the right most cells receive external inputs, and their own values are either 0 or 1.

Assign a value of 1 to the pedestrians who are walking towards right if the left most cells receive external inputs, and their own values are either 0 or 2.

**Exit rule:**

If cells with value of 1 reach the left most cells, exit.

If cells with value of 2 reach the right most cells, exit.

**Straight movement rule:**

Move forward (left or right) if the cell in front of the current cell is empty, and two cells ahead does not have a cell moving in the opposite direction.

**Sidestep upwards rule set**

For cells with value of 2, if the cell to its left has a value of 1, try to step to the cell above. Perform the moving action only when the cell above is empty, and there is no straight movement into the cell.

For cells with value of 2, if the second cell to its left has a value of 1, try to step to the cell above. Perform the moving action only when the cell above is empty, and there is no straight movement into the cell.

For cells with value of 1, if the cell to its right has a value of 2 and the cell below has a value of 50 (the wall), try to step to the cell above. Perform the moving action only when the cell above is empty, and there is no straight movement into the cell.

For cells with value of 1, if the second cell to its right has a value of 2 and the cell below has a value of 50 (the wall), try to step to the cell above. Perform the moving action only when the cell above is empty, and there is no straight movement into the cell.

**Sidestep downwards rule set**

For cells with value of 1, if the cell to its right has a value of 2, try to step to the cell below. Perform the moving action only when the cell above is empty; and there is no straight movement into the cell; and there is no sidestep into the empty cell resulted from any sidestep upwards rules.

For cells with value of 1, if the second cell to its right has a value of 2, try to step to the cell above. Perform the moving action only when the cell above is empty; and there is no straight movement into the cell; and there is no sidestep into the empty cell resulted from any sidestep upwards rules.

For cells with value of 2, if the cell to its left has a value of 1 and the cell above has a value of 50 (the wall), try to step to the cell above. Perform the moving action only when the cell above is empty; and there is no straight movement into the cell; and there is no sidestep into the empty cell resulted from any sidestep upwards rules.

For cells with value of 2, if the second cell to its left has a value of 1 and the cell above has a value of 50 (the wall), try to step to the cell above. Perform the moving action only when the cell above is empty; and there is no straight movement into the cell; and there is no sidestep into the empty cell resulted from any sidestep upwards rules.

**Expected behaviour**

It should be a fluent flow of pedestrians with no conflict. If at any time a potential sidestep conflict could happen, the pedestrian who steps upwards has the priority to take over the cell, and the one steps downwards will be waiting in that cycle. There should be no traffic gam unless there are continuous traffic generation at two adjacent rows of the model lasting to the end of simulation.

# Testing of the project implementation

This section will present the testing results of the project. The Cell-DEVS animation tool will be used to demonstrate the results. A simulation video can be found at the following link:

http://www.youtube.com/watch?v=6xvDVrhgNMM

**Entrance rule testing:**

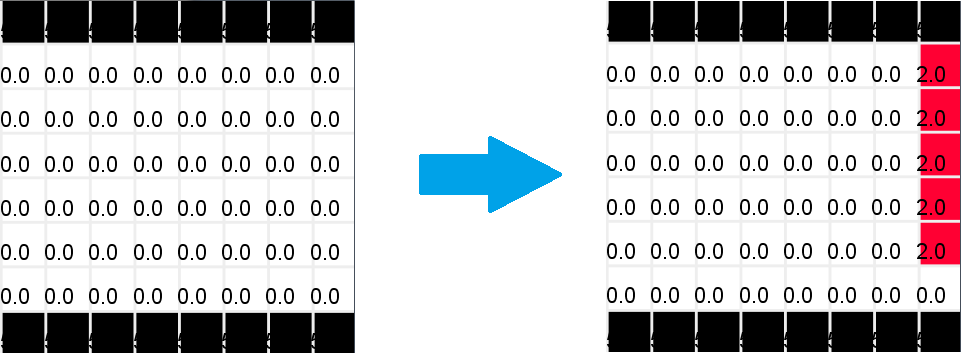


Figure : Entrance rule testing 1

The most basic entrance rule, when there is no one on the edge of the walkway, enter freely.

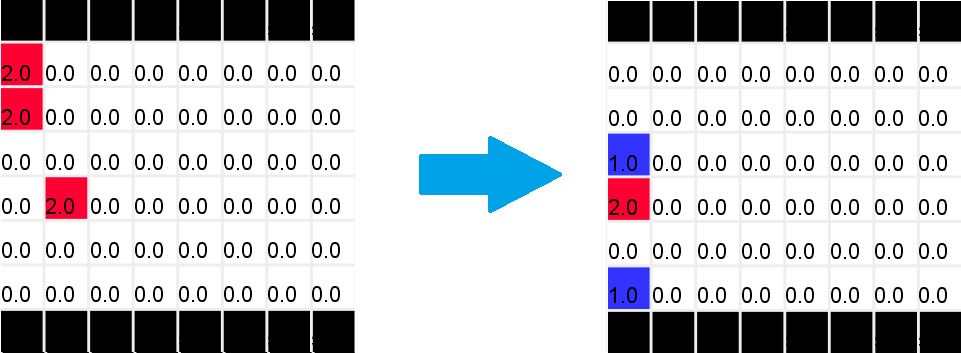


Figure : Entrance rule testing 2

While the pedestrian from the right (two cells with value of 2) leaves the modeled space, new entrance from left (cells with value of 1) enters the space.

As shown in the course of simulation, entrance behaves as expected, no collision happens, nor any sight of losing of a cell.

**Exit rule testing:**

Please refer to Figure 5 to see the illustration of the cells leaving the modeled space. Over the course of the simulation, the behaviour of the rule functions well.

**Straight movement rule testing:**

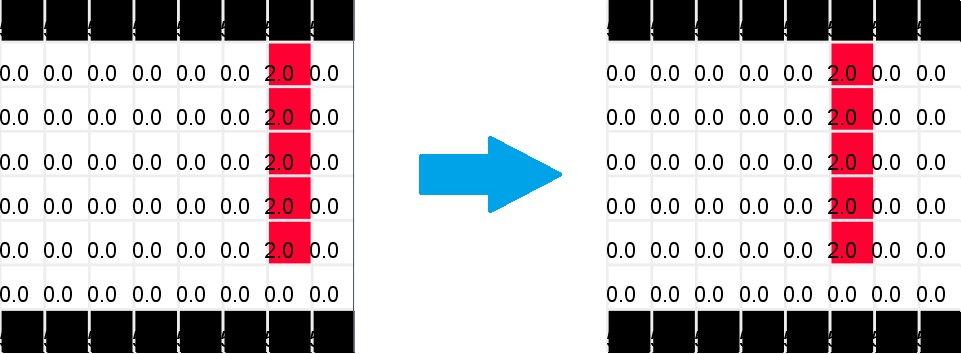


Figure : Straight movement rule testing 1

For each of the 5 cells traveling toward left, since there is nothing in front of them they move forward together.

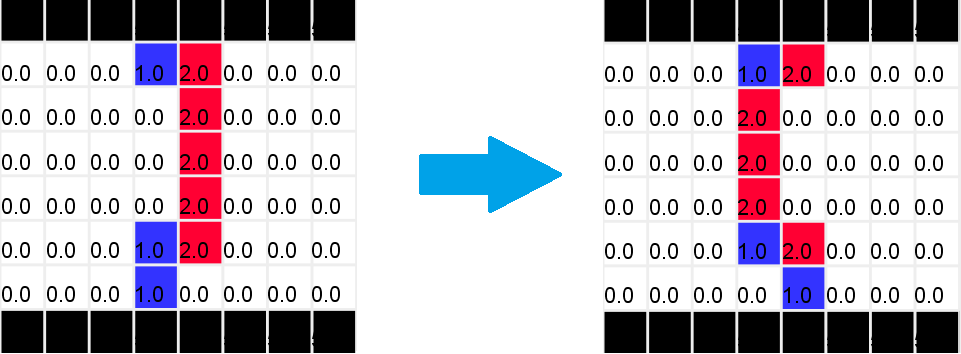


Figure : Straight movement rule testing 2

Cells from both directions travel forward together. The blocked ones dose not perform side step in this case because there is no empty space above or below, or there is a cell moving straight into the empty cell. Overall, straight movement behaves as expected and no collision observed.

**Side stepping when block rule testing:**

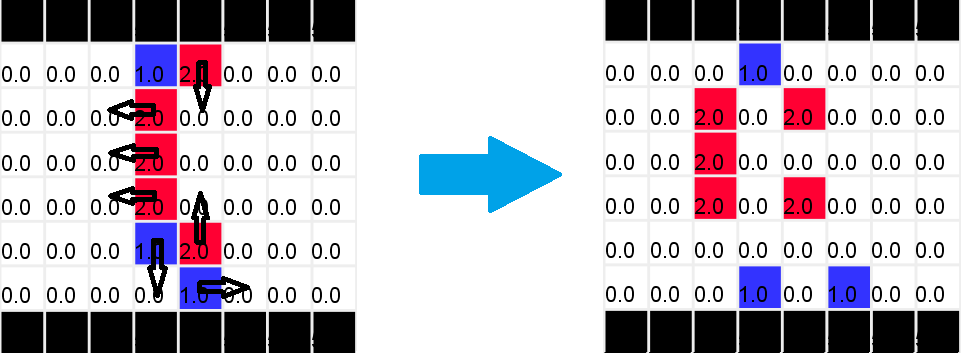


Figure : Side stepping rule when blocked testing 1

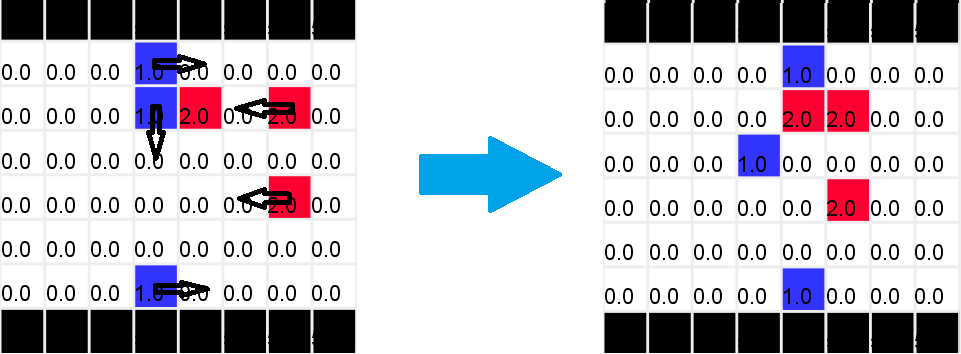


Figure : Side stepping rule when blocked testing 2

As illustrated with the arrows, the side stepping rule behaves well. In this example, no potential conflict due to sidestepping exists. The particular cast testing will come later in this section.

**Side stepping due to heading to same cell testing:**

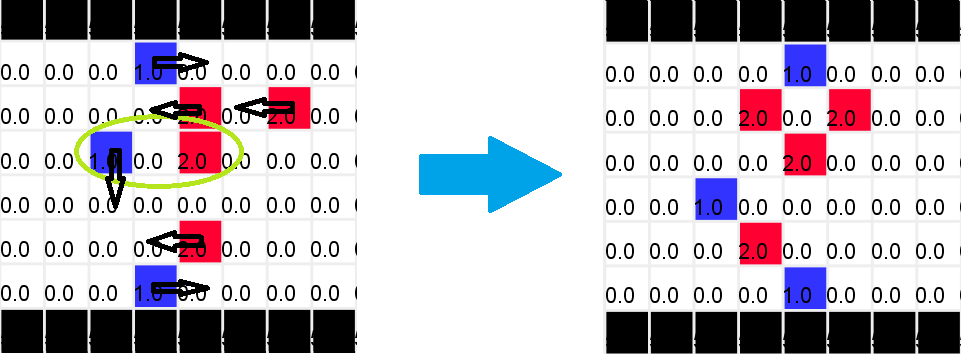


Figure : Side stepping due to heading to same cell testing

Same as the “side stepping when blocked” rule, over the course of this simulation, the “side stepping due to heading to same cell” rule behaves very well too. No collision is detected.

**Boarder behaviour rule testing:**

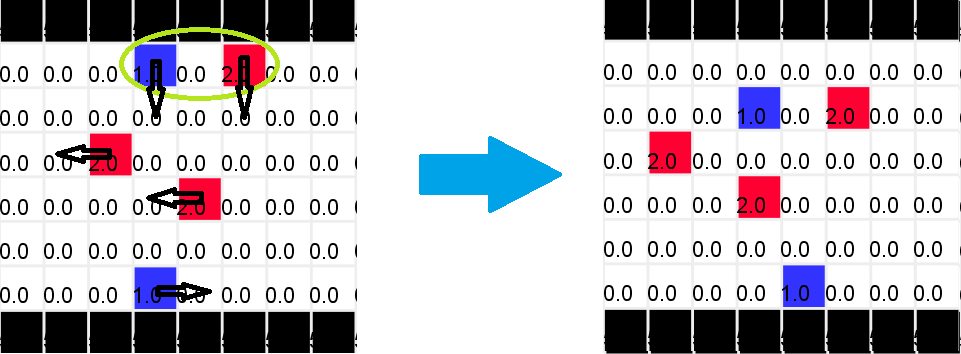


Figure : boarder behaviour rule testing 1

As shown with the green circle, two pedestrians attempts to move into the same cell and thus need to perform side step. Originally, they both should step to the right of their walking direction, but the red cell is not able to because there is a wall at its right, so it stepped to the left.

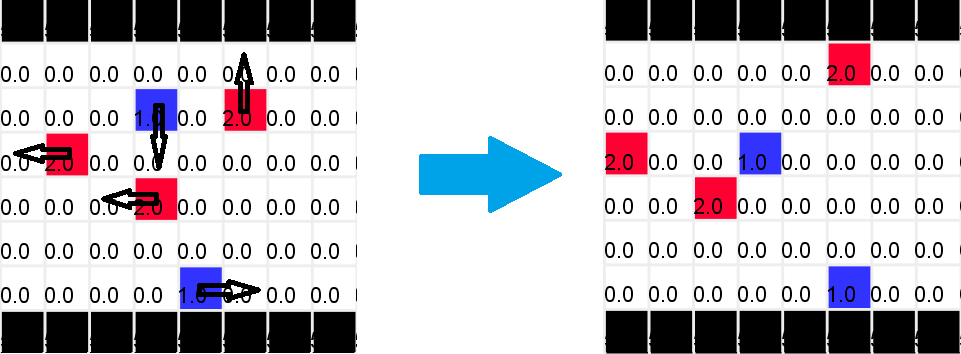


Figure : Boarder behaviour rules testing 2

After the movement illustrated in Figure 11, this figure shows the next action of the cells.

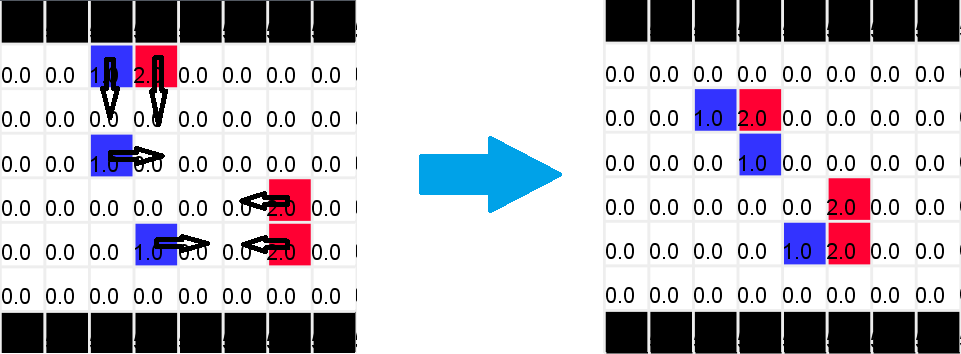


Figure : Boarder behaviour rules testing 3

Figure 11 shows the boarder behaviour with two cells heading to the same cell. Figure 13 shows the boarder behaviour when blocked.

All above testing cases were successfully accomplished in the implementation of assignment 2. The following testing case is the critical one in charge of proving the implementation of the priority scheme is successful.

**Conflicting sidestep rules testing:**

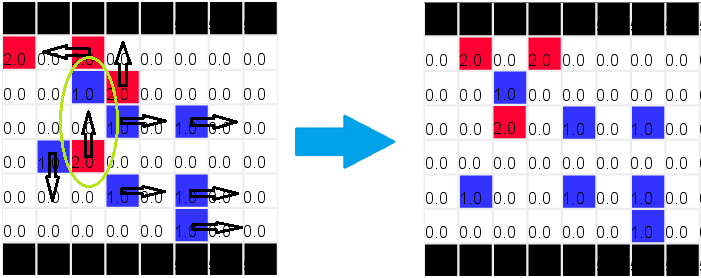


Figure : Conflicting sidestep rules testing

Figure 17 shows the new behaviour after the implementation of the sidestep checks. As shown in the green circle, the cell with value of 1 is blocked for moving straight to the right; and the cell with value of 2 is blocked for moving straight to the left. This is the situation where the conflict of sidestepping would occur. In the implementation of assignment 2, the cell with value of 1 would step downwards, and the cells with a value of 2 would step upwards. Thus one cell will overlap another and thus “kill” one pedestrian. Now, after the implementation of sidestep check with priority scheme, this conflict does not occur anymore. After those two cells in the green circle discover they are blocked for move straight forward, the second level priority rule tells the cell with value of 2 to check if the cell above is empty. If it is empty, step upward. Then the third level priority rule tells the cell with value of 1 to check if the cell below is empty and if there is other cell moves into it. Since there is indeed a cell moving into the empty cell, the cell with value of 1 waits in this cycle. One more thing worth to mention is that the priority scheme makes the pedestrian flow very naturally. Please imagine that there was no priority scheme applied to the model, but all rules need to check all other rules. Those two cells in the green circle will both detect the other is potentially moving into the empty cell. Thus both cells will just wait in that cycle. This is not very desirable and might cause a traffic gam in some situation. Figure 17 does not really illustrate this situation, but it is definitely desirable to have one cell to move to ease the traffic.

This kind of sidestepping conflict situation illustrated in Figure 17 happens vary rarely. It is also the reason that the implementation of assignment 2 was quite promising. The testing case illustrated in Figure 17 took quite a few simulations with different parameters for the generators to finally obtain such case. Thus there will be only one illustration included in this report for this test case. After the observation of this behaviour, the overall project can be concluded to be successful.

# Conclusion:

To conclude, the project was very successfully designed and implemented. It is very pleasing to figure out an optimal way to apply the priority scheme to the project. The final method was very simple and should be obvious from the very beginning. However, it took quite a long way around to arrive at this seemingly obvious implementation. Nevertheless, much knowledge were gained and strengthened over the process. The completed simulation runs very well. Pedestrians flow very naturally. There were no conflict detected at all, and there was no traffic gam observed.

Due to most of the time was consumed on exploring multiple layer approach, there are a few ideas did not get a chance to examine. For example, change the shape of the pedestrian path will definitely introduce more problems; some more natural human behaviours did not get a chance to be implemented; and there might be even better way than the priority scheme exists to explore. Despite these regrets, the project was a success.

# Reference

[1] T. Wang, J. Chen. An Improved Cellular Automation Model for Urban Walkway Bi-directional Pedestrian Flow: International Conference on Measuring Technology and Mechatronics Automation, 2009.

[2] Gabriel A. Wainer. Discrete-Event Modeling and Simulation: A Practitioner’s Approach. CRC Press. April 30, 2009.