POLITECHNIKA GDAŃSKA

INTELLIGENT LOGISTICS SYSTEMS

ACTIVITY #6

AGV – Automated Guided Vehicles

dr hab. Daniel Kaszubowski, prof. PG

Department of Transportation Engineering

ADTRANS
ADVANCED DATA ANALYSIS FOR TRANSPORTATION AND MOBILITY ACADEMIC TEACHERS

Co-funded by the European Union

Co-funded by the European Union. Views and opinions expressed are however those of the author or authors only and do not necessarily reflect those of the European Union or the Foundation for the Development of the Education System. Neither the European Union nor the entity providing the grant can be held responsible for them.

1. Objective and new skills

The aim of the task is to introduce the use of AGVs in simulation models. Two approaches to AGV control will be presented, including using elements of the 3D model and *Process Flow* with built-in templates.

New skills
Creating AGV paths and checkpoints
Incorporating AGV into a 3D model
Dispatcher Block
Using built-in *Process Flows* for AGV control

2. Part I – Model using 3D objects

2.1. Model assumptions

The model being created simulates basic activities related to picking up and putting away products using AGV. The basic part of the model is created in 3D mode, while the basic Process diagram is responsible for generating products. Assumptions:

1. Products are generated in the *Delivery (Dostawy)* queue using the *Product Generation* flowchart.
2. The generated products go to the *Sorter1 (Sortowanie1)* processor.
3. Two AGV vehicles, *AGV1* and *AGV2*, pick up products from the sorter and place them on the *Sink (Zrzut)* block .

The material was based on: K. Jurczyk, FlexSim, user manual.
InterMarium Sp. z o . o. , Krakow 2022

The target layout of the 3D model is shown in Fig. 1, and the *Product Generation* diagram is shown in Fig. 2.

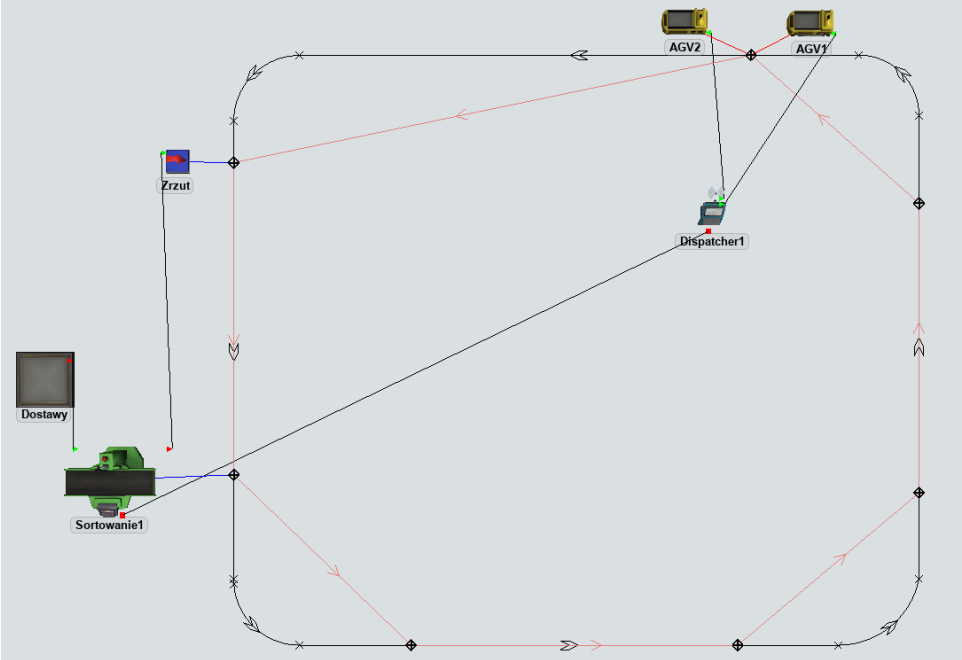


Fig. 1. 3D model layout

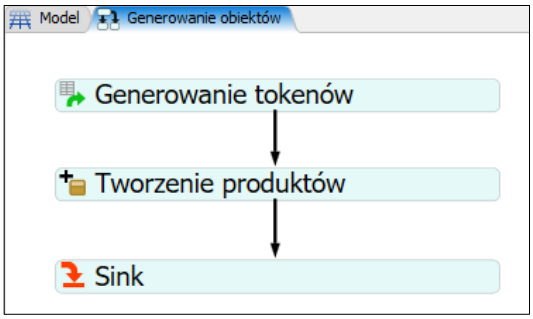


Fig. 2. Product Generation flow

2.2. Building a 3D model

2.2.1. AGV travel route

The route shown in Fig. 1 should be recreated, remembering to maintain the correct direction of movement (counterclockwise) and to precisely connect straight sections with arcs (Fig. 3). All objects necessary for building the AGV route are located in the AGV library.

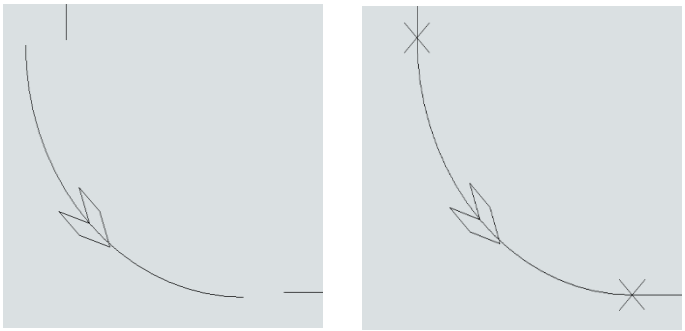


Fig. 3. Connecting the AGV route elements

Next, control points *(DecisionPoint)* will be added, which will be connected directionally, creating a route that the AGV will follow. They also allow for connecting 3D objects to the network. Control points are marked

with a crossed-out square (Fig. 4) and their default names and numbers can be retained. Their arrangement is illustrated in Fig. 1.

All control points in the model (7 pcs.) should be connected directionally, in accordance with the direction of movement adopted in the AGV movement route (Fig. 4). The connections created in this way will be added to the diagram as red lines with arrows indicating the direction of AGV movement.

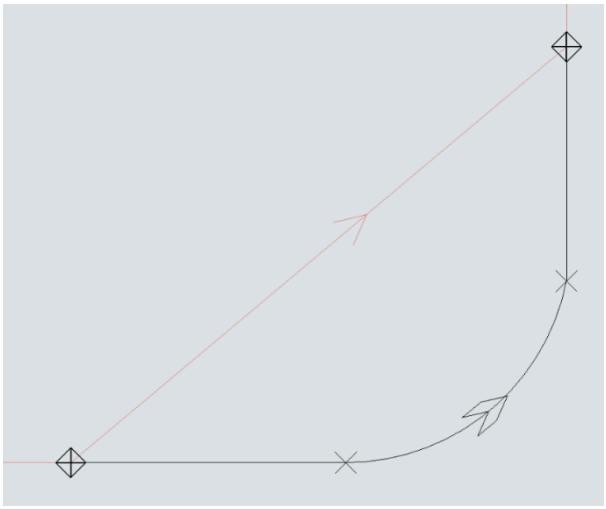


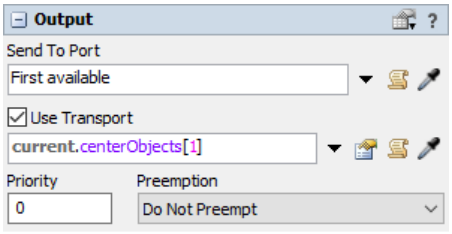
Fig. 4 Directionally connected control points

2.2.2. Adding objects to the network

The following elements will be added to the created network:

1. 3D objects: *Delivery* queue, *Sortowanie1* processor, and *Sink (Zrzut)* .
2. Two AGVs, *AGV1* and *AGV2*.
3. *Dispatcher* control object .

There is no typical Source in the model that creates flow elements, because these will be generated by a separate *Process Flow* in the *Deliveries (Dostawy)* queue. Therefore, the default program settings will be applied to this queue and the dump object. Remember the directional connection *Deliveries → Sorter1 → Sink*.

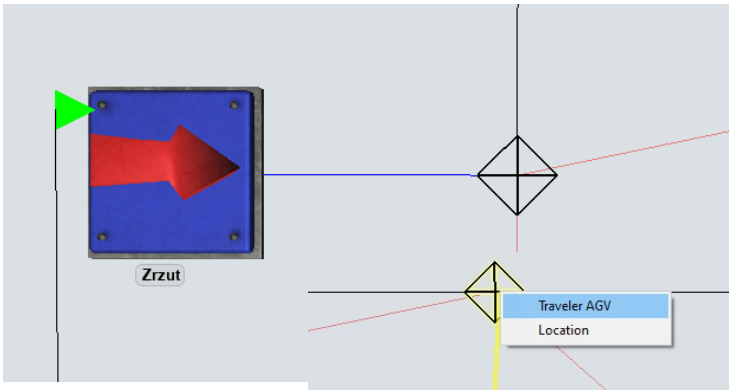


Rys. 5 Ustwienia Output na Sorter1

For the *Sotorwanie1* processor, the default options will also be applied, except for the *Output* setting, taking into account the connection of the *Dispatcher* control object central port (Fig. 5). These settings should be remembered when adding this object later.

After adding the basic 3D objects to the model, you need to connect them to the network that the AGV will move on. Therefore, *Sorter1* and *Sink (Zrzut)* are connected directionally to the nearest decision points

– the blue line in Fig. 6. It also shows a directional connection to *Sorting1* (on the left with a green arrow), and a decision point with directional connections (red lines) to two other decision points.



Rys. 6 Pol

The last task is to add both AGV and *Dispatcher* object and connecting them to the network. All objects are added from the *Task Executors* library. AGV should be connected directionally to the nearest decision point and the *Traveler AGV* option should be selected (Fig. 7).

Rys. 7 Włączanie AGV do sieci

Dispatcher is responsible for controlling the AGV. It should be connected to *Sortowanie1* via a central port (Fig.8), remembering to select *Use Transport* and `current.centerObjects [1]`. This notation means that the first object connected to *Sorter1* via a central port will be used. The figure shows the central connection to *Sorter1* (ended with a red square) and the directional connections *Dispatcher* → *AGV*, ended with red arrows.

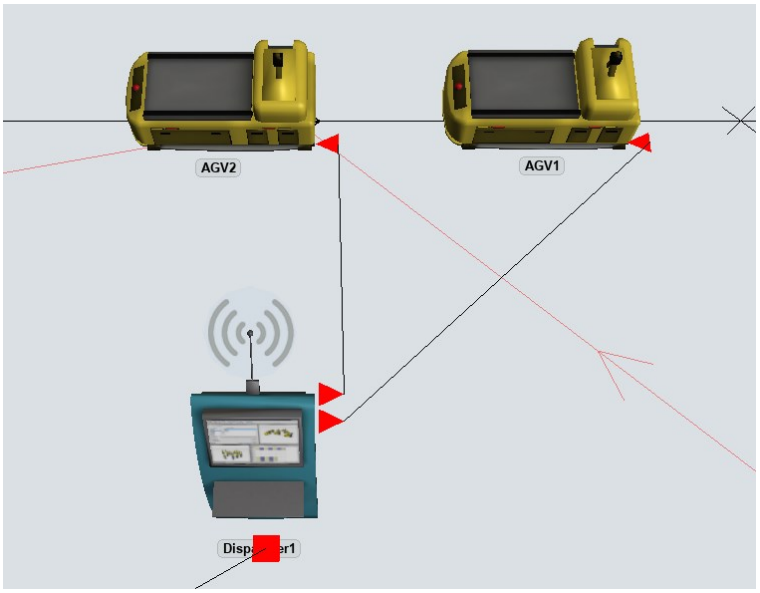


Fig. 8 Dispatcher connection diagram with related objects

2.3. Generating Products – *Process Flow*

Products are generated using a simple sequence of blocks including creating tokens according to a schedule *(Scheduled Source),* creating products in a specified 3D object, and deleting tokens (Fig. 9).

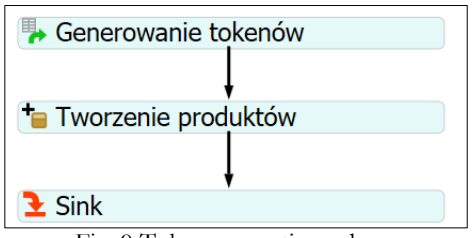


Fig. 9 Token generation scheme

The settings of the first two blocks are shown in Fig. 10. One token generated in the *Token Generation* block creates 10 products in the *Deliveries* queue.

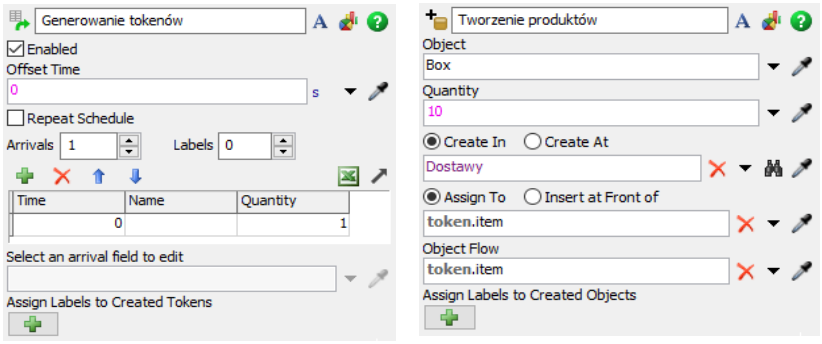


Fig. 10 Token generating block settings

3. Part II – Using the Built-in *Basic AGV Template*

3.1. Introduction to AGV Templates

Operations using AGVs in practice are characterized by both high complexity and repeatability. Therefore, FlexSim provides several templates that allow for mapping typical AGV applications using *Process Flow* . This is to facilitate the creation of complex models by automating repetitive tasks. In *Process Flow* → *Add an Object Process Flow* tab offers four templates for AGVs with a gradually increasing level of complexity, resulting in the addition of new functions such as a task distribution mechanism or parking while waiting for a task assignment.

Due to the capabilities of the software version used, the capabilities of the basic available template, i.e. *Basic AGV*, will be presented.

☐ Detailed explanation of the rules for working with the *AGV template* is in the online user manual:

Working with task executers → Travel and Transportation → AGV Networks → Using the AGV Process Flow templates

AGVs are task executors *(Task Executors)* connected to the AGV movement network *(path network)*. The assumptions of this process were presented in the first part of the material, where 3D elements were responsible for most of the activities.

If the AGV is controlled by *Process Flow*, three types of actions are defined successively to determine the behavior of the AGV and other model elements (Fig. 11):

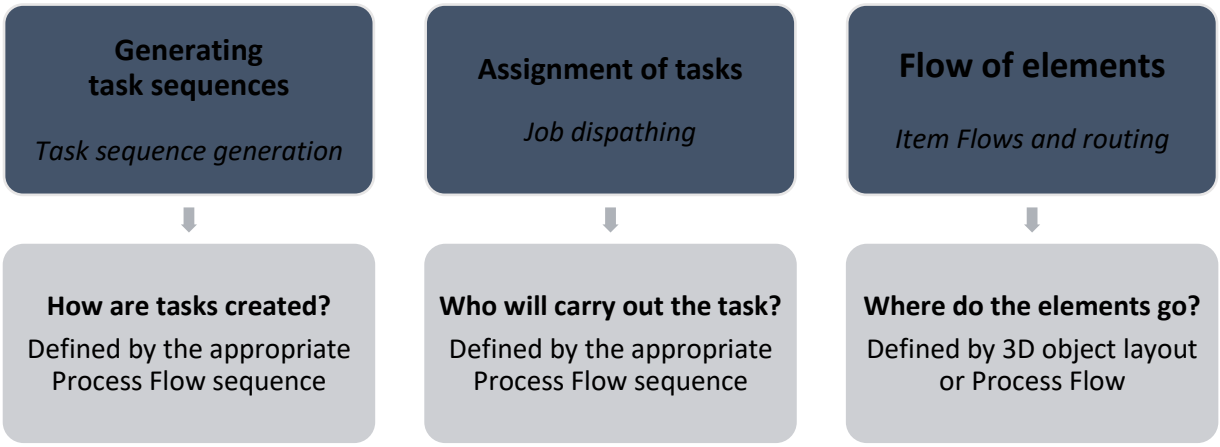


Fig. 11. The logic of the *Process Flow* controlling the behavior of the AGV

According to this scheme, in the developed model, the *Basic AGV* template is responsible for generating the sequence of tasks and their allocation, while the flow of elements will be handled by a simple 3D model to which the ready template will be connected.

3.2. Basic AGV Template Structure

3.2.1. Basics of operation and arrangement of elements

The structure of elements in the *Basic AGV* template is shown in Fig. 12. The separated groups of activities are:

1. *Work Generation* – generating tasks based on items entered into the *AGV Work List* by elements appearing in *Queue1 and Queue2* in the 3D model *(Output → Push to* ***Item List****)* .
2. *Item Pickup* – saves generated tasks on the *AGV Work List* in partitions corresponding to the numbers *WorkPoints* , which allows the AGV to identify where the item was picked up.
3. *Main Control Loop* – a group controlling the assignment of tasks to AGVs based on *the AGV Work List.*

4. *Loading Unloading* – controlling the loading of AGVs at a specific *WorkPoint* and unloading at a designated location.

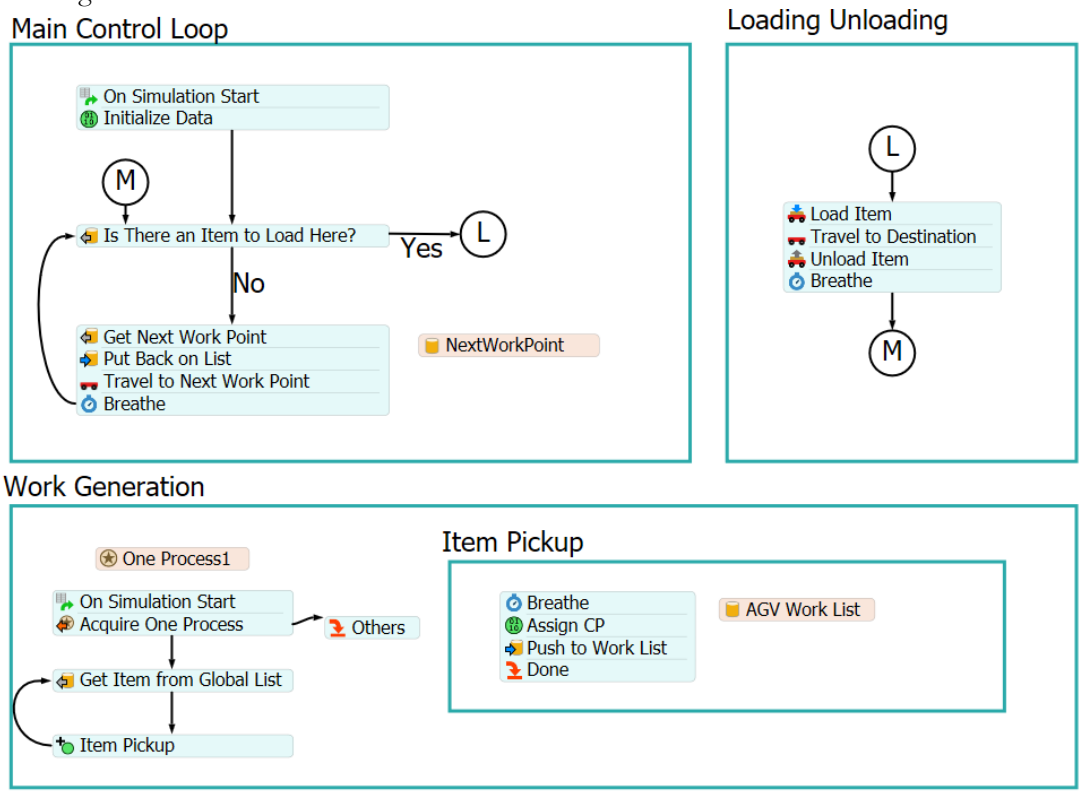


Fig. 12 General structure of the *Basic AGV template*

The template's operation uses the principle of AGV movement in a loop defined by interconnected *WorkPoints,* which are control points *(Control Points).* At each *WorkPoint* The AGV checks if there are tasks to be performed. If there are objects to be collected at a given point, the associated sequence of actions is performed. If there are no items to be collected, the AGV goes to the next point. The process is repeated at each point.

In this approach, the allocation of tasks is determined primarily by the position of the AGV and the arrangement of other objects in the 3D model (e.g. *Sink*).

3.2.2. Work Generation section

In the **Work Generation group** the following repeatable activities are performed:

1. Downloading a task from *the AGV Work List,* saved there by a 3D object. The list is created automatically after downloading a template *(→ Toolbox).*
2. Check of *WorkPoint* corresponding to the downloaded task.
3. Re-save the task to the list, this time in the partition corresponding to a specific *WorkPoint* (fig. 13).

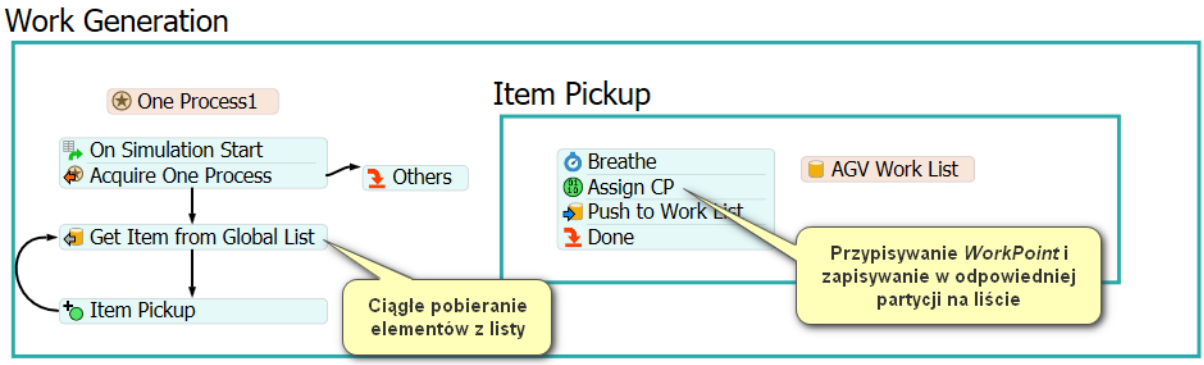


Fig. 13 Downloading a task from the list, setting the *WorkPoint* and saving it to a partition.

The task allocation mechanism used in the *Basic AGV template* is summarized in Table 1.

Table 1. Task allocation rules

Selecting a resource (task executer)	**Selecting a task**
The template does not assign a performer to a task immediately after it is created by a 3D object. When **a new task** appears (is saved in the *AGV WorkList*), **it waits for an AGV to appear** at a specific *WorkPoint*. Therefore, the task is assigned to the first AGV at that point. This also means selecting the closest AGV. Delayed resource selection can be beneficial in a situation where a certain AGV unloads an item at a point close to *the WorkPoint* where the task occurs, but after the task occurs. In the case of immediate task assignment, it could be assigned to another AGV, located further away from the reported point.	The task is selected when the AGV arrives at the *WorkPoint*. It then checks **if there are any tasks** to perform. This involves taking a task from the *AGV WorkList partition* that corresponds to a given *WorkPoint* . If there are no tasks to be performed, the AGV will go to the next *WorkPoint*.

This part of the model contains the following blocks:

1. *On Simulation Start (Scheduled Source)* – generates one token at the start of the simulation, which will circulate in a loop, initiating retrieval from the list and re-writing to the list, taking into account the *Control Point* number and partition number.
2. *Acquire One Process (Acquire Resource)* and its associated *One Process (Resource)* – a pair of blocks responsible for occupying one numerical resource. This solution, in connection with retrieving tasks from the list, allows the use of one repeatable process instead of separate processes for each AGV in the model.
3. *Get Item from Global List (Pull from List)* – a block connected to *the AGV Work List (List)* block, which in turn refers to the *AGV Work* global list . If there is a task on the list entered there by a 3D object (queue), the token retrieves it and passes it on to the *Item Pickup* block.
4. *Item Pickup (Create Tokens)* – is related to *Breathe* in the *Item Pickup* section . Once the token is entered, it creates a new independent token that initiates the process of adding a *Control Point* number, allowing the AGV to pick up an item from that point. The original token loops back to *Get Item*

from Global List , allowing for continuous searching of the task list and passing them to *Item Pickup* , from where the tasks are picked up by the AGV.

Item Pickup is an integral part of the *Work section Generation,* responsible for entering tasks supplemented with the *Control Point* number into the list. This is the starting point for downloading tasks by the AGV. This group consists of the following blocks:

1. *Breathe* – a block generating a 0 second delay, added to separate individual tasks.
2. *Assign CP (Assign Labels)* – adds labels to the token specifying the *ControlPoint (CP)* for the task. This is done by:
   - saved under the label `isInCP` comparison of the location of the flow element with the list of control points, saved in *Toolbox → Global Lists → Control Port Connections → NextWorkPoint;* the formula `isclasstype(token.item.up,"AGV::ControlPoint ")` is responsible for this,
   - saved under the label `cp` formula `token.isInCP ? token . item. up : AGV.Connections (token.item.up," Locatio ")[ 1 ];` this is a ternary operator – if the first condition is correct (`isInCP` , i.e. the flow element is in *the ControlPoint)* then the current CP will be saved in the list as the location of the element to be transported.
3. *Push to Work List* – saves the task to the list, adding the appropriate partition number using the previously created label `cp` (*Partition ID* → `token . cp`).
4. Once a task is saved to the list, the token created in *Item Pickup* is deleted and the list contains tasks for the AGV sorted into partitions corresponding to the *ControlPoints* numbers .

3.2.3. Main Control Loop Section

The section is responsible for controlling the AGV movement along a defined loop. Vehicles will search for tasks in the checkpoints on the list and perform them according to the logic specified in the *Loading/ Unloading* section. If there is no task at a given checkpoint, the AGV will go to the next checkpoint. *Main Loop* consists of the following blocks:

1. *On Simulation Start* – generating a single token at the beginning of the simulation , which will circulate in a loop initiating AGV actions according to the availability of tasks on the *AGV Work List.*
2. *Initialize Data* – adding labels:
   - `cp` with the value `AGV (current).currentCP,` pointing to the current ControlPoint,
   - `capacity` with value `getvarnum(current,"maxcontent"),` determining the occupancy of the AGV,
   - `agv` by the value `current,` identifying the current AGV,
   - `parkPoint,` with value `token.cp` indicating the control point to which the AGV is associated.

3. *Is there an Item to Load Here* – retrieves tasks from the *AGV Work List* , searching for them in the partition with the number corresponding to the *Control Point* number *(PartitionID* field → AGV(current).currentCP).
4. If the task is found at a given point (output Yes), the token is redirected to the *Loading Unloading* block.
5. If there are no tasks at a given point (output No), the token goes to the *Get Next Work Point* block where from the *NextWorkPoint* list information about the next work point is retrieved.
6. In *Put Back on List* the information is returned to the workpoint list.
7. *Travel to Next Work Point* – directs the AGV to the next work point.
8. From *the Breathe* block the feedback to *Is there an Item to Load Here* is derived, which means the AGV searches for tasks at the next work point.

Related to *the Main Control Loop* is the ***Loading group Unloading*** , controlling activities related to flow elements:

1. *Load Item* – loading of the flow item.
2. *Travel to Destination* – in the *Destination* field there is a formula current.first.destination, where destination is a reference to the label created when the connections between 3D model elements were created, i.e. between the queue and the *Sink*.
3. *Unload Item* – AGV unloading, save current.first in the *Item* field denotes the first flow element on the AGV.
4. In the *Breathe* block there is a feedback loop with *Is there an Item to Load Here,* i.e. returning to the AGV's search for tasks at the next work point.

3.3. 3D model associated with the *Basic AGV template*

The template described earlier should be connected to a 3D model, which will be responsible for the arrangement of individual objects. Due to the universality of the template, it can be easily connected to various object arrangements. Fig. 14 shows such a model. It contains:

1. Two sources *Source1 and Source2* and their connected queues *Queue1 and Queue2.*
2. *Sink1 and Sink2* queue related sinks.
3. Six *Control Points ,* four of which have directionally connected queues and drops.
4. One AGV, directionally connected to the same control point as *Queue1.*

5. Control points connected with each other by directional connections, creating the so-called: *WorkPoint Loop* . It is marked with red arrows connecting points, maintaining the direction of movement marked by the path for the AGV.

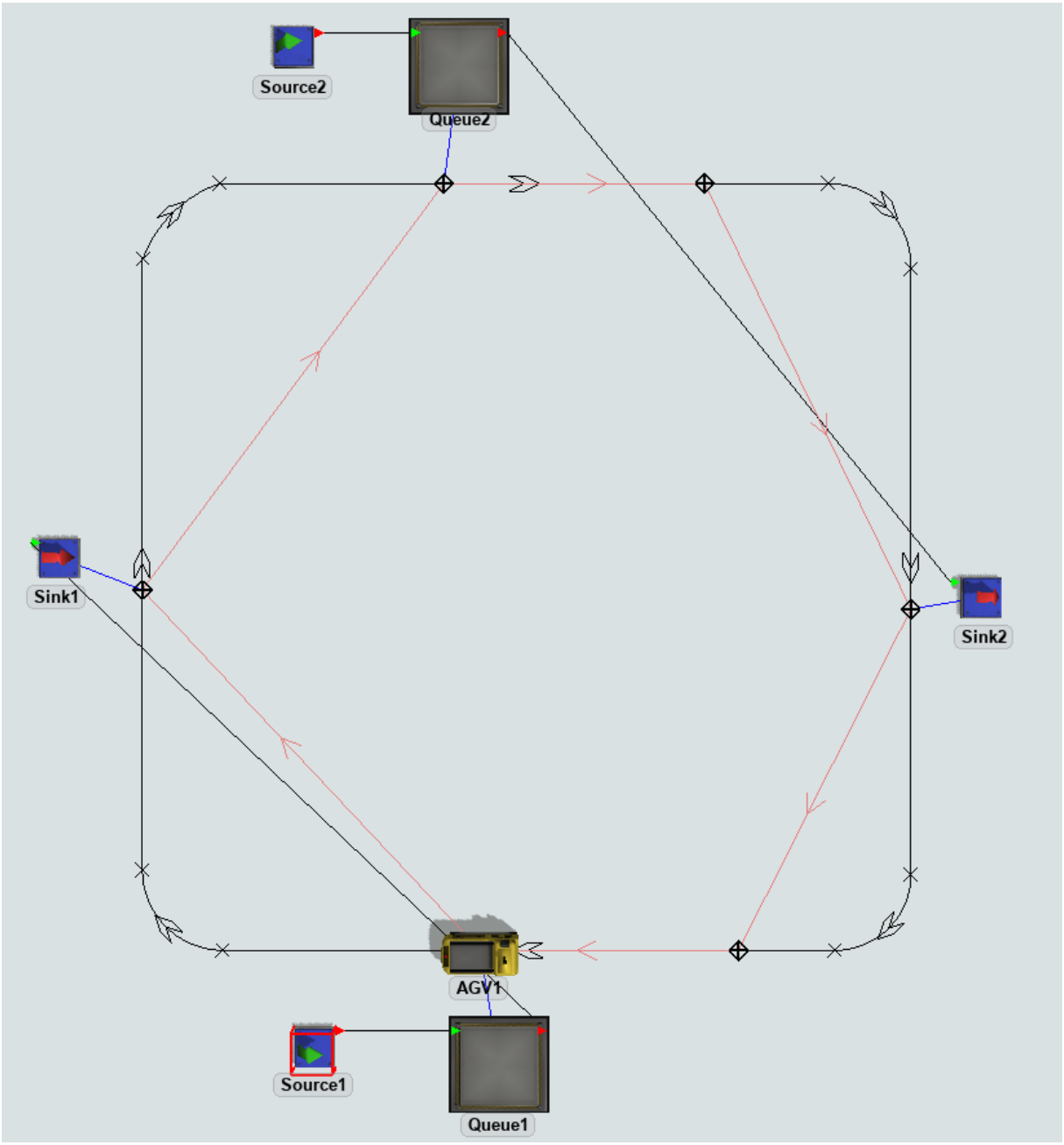
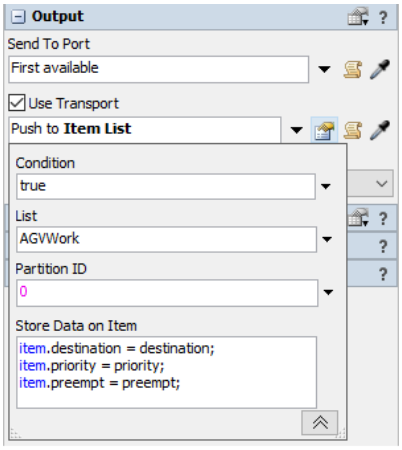


Fig. 14. 3D model linked to the *Basic AGV template*

To connect the 3D model with the template, it is necessary to introduce previously indicated changes in the operation of the queues, which will result in the creation of the *AGVWork list* tasks of moving the element by AGV. This list is created automatically in the *Toolbox* tab after adding a template.



Rys. 15 Ustawienia Use Transport w kolejkach

In each queue, you must check the *Use Transport box* and select *Use List → Push to List* by selecting the appropriate list (Fig. 15). Also make sure that the *Store Data on Item field* is selected correctly. Notation item. destination = destination saves for each flow element under the destination label its destination, which in this case is the *Sink* connected to the queue.

You should also assign the AGV to the *AGV template* . This can be done from the template options *Properties → Attached Objects (instances),* where the AGV is indicated in the 3D model with the picker. You can also do this directly from the AGV by selecting it with the right mouse button and then *Process Flow → AGV* (fig. 16).

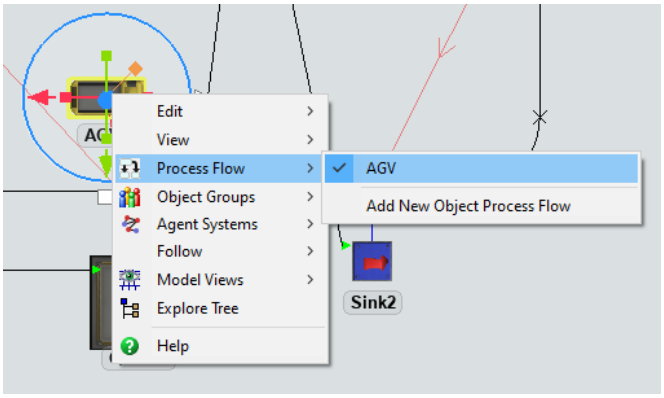


Fig. 16. Connecting AGV with a template

After starting the simulation, both sources will start generating flow elements for two queues, which will enter them into the *AGVWork list* into partitions corresponding to the work point numbers *WorkPoints* to which these queues are assigned (Fig.17).

AGVWork Entries

value	Type	age	distance	queueSize
Partition ID: /ControlPoint1				
/Queue1/Box		84.49	Puller Required	10
/Queue1/Box~2		69.30	Puller Required	10
/Queue1/Box~3		62.89	Puller Required	10
/Queue1/Box~4		57.95	Puller Required	10
/Queue1/Box~5		32.03	Puller Required	10
/Queue1/Box~6		27.42	Puller Required	10
/Queue1/Box~7		26.51	Puller Required	10
/Queue1/Box~8		23.88	Puller Required	10
/Queue1/Box~9		15.87	Puller Required	10
/Queue1/Box~10		13.24	Puller Required	10
Partition ID: /ControlPoint3				
/Queue2/Box		60.65	Puller Required	4
/Queue2/Box~2		38.62	Puller Required	4
/Queue2/Box~3		20.76	Puller Required	4
/Queue2/Box~4		0.23	Puller Required	4

AGVWork Table after starting the simulation