

Report Document

This documents provides the MAS solution for Block World for Team problem using ASTRA. Following are the system and software specifications used to implement this solution.

System Specifications		Software Specifications	
Operating Systems	Windows 10 x64	Programming Language	ASTRA
RAM	8 GB	Development Tool	Eclipse Neon
Hard Disk	1 TB	Environment	BW4T, jre 8
Processor	Intel i5 2.4GHz		

The characteristics exhibited by the implemented BW4T system

Characteristics Exhibited	
<input checked="" type="checkbox"/>	The ability to visit each room in the environment
<input checked="" type="checkbox"/>	The ability to bring a block from a room to the drop zone
<input checked="" type="checkbox"/>	The ability to locate and retrieve the next block in the specified sequence
<input checked="" type="checkbox"/>	Some form of coordination strategy that uses multiple bots to locate and retrieve blocks.
<input type="checkbox"/>	Some form of optimisation of the coordination to support concurrent retrieval of blocks. <i>Note: In this implementation, a master bot is used to fetch the blocks. While the other bots assist the master bot in doing so.</i>
<input checked="" type="checkbox"/>	Some consideration of how to minimise the total travel distance of the elevators.
<input checked="" type="checkbox"/>	The system should be able to adapt to different maps
<input checked="" type="checkbox"/>	The system should be able to cater for differing numbers of bots.

Code Structure:

Source Code Files

1. *BlockMovement.astra*

Starting ASTRA agent File Implementing Block World for Team (BW4T) system using 'N' number of Robots.

2. *ChildBots.astra*

This ASTRA agent File helps in creating generics multiple agents and binds agents with the Block World Entities (Bots). The communication between the master agent and child agents is done with the help of FIPA messaging

Strategies adopted to complete the six tasks assigned:

TASK C1:

A working agent program that connects to the BW4T Environment and Configures the environment correctly for at least one scenario that includes 1 bot.

TASK C2:

The system is able to control the bot that is able to collect and drop the relevant sequence of blocks in the drop zone.

Strategies adopted for the TASK C1 and TASK C2

1. The **init ()** goal [*BlockMovement.astra*] is adopted which makes sure that the agent program connects with the BW4T environment and the whole system is implemented. Various sub-goals are used for this purpose.

The configurations are specified in the init rule. The different tasks within this rule are as follows:

- a. Launching and connecting the BW4T environment
 - b. Starting the environment
 - c. Calling of sub-goal **agentGenerate ()** to link the agent/s to the environment
2. The sub-goal **agentGenerate ()** [*BlockMovement.astra*] takes the help of the child agent file [*ChildBots.astra*] to create the multiple child agents and link them to the BW4T environment.

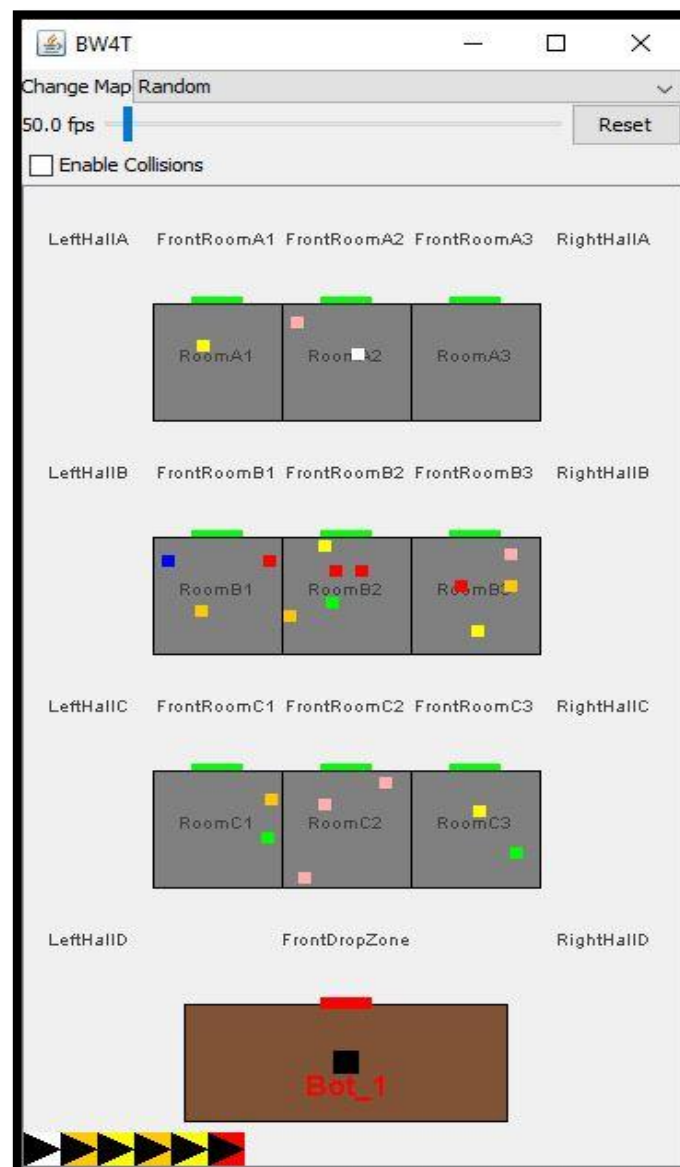
For the single bot implementation, the child agents are not created. The current agent file [*BlockMovement.astra*] acts as an agent for it and is linked with the BW4T environment.

3. Now for implementing the BW4T system using the single bot, **singleRobotMovement ()** sub-goal is invoked which takes the help of **robotGoingToRoom ()** sub-goal to complete this problem.

The main purpose of these sub-goals is to implement following functionality:

- a. For every colour in the sequence,
 - i. Check if the room is already visited,
 - ii. If the room is already visited;
 - Send it to that room only when the block with the required colour is present in that room
 - Pick the block and drop it in drop zone
 - Update the room information (list of colours available within a room)

- iii. If the room is not visited,
 - enter the room and check whether the block with the specified colour is present in the room.
 - If the block is present, pick it up and drop it in drop zone. Otherwise, move to the next room.
 - Update the room information (list of colours available within a room)
- b. Once all the blocks in the sequence are gathered in the drop zone, the system is terminated



The above snapshot depicts the completion of Task C1 and C2 using single Bot.

TASK C3:

The system uses a fixed number (> 1) of bots and some form of coordination that uses all of the bots to solve the problem.

TASK C4:

The system attempt to optimise the performance of the team of bots

TASK C5:

The system is able to adapt to different numbers of bots.

Strategies adopted for the TASK C3, TASK C4 and TASK C5

For implementing the task C3, C4 and C5.

A coordination strategy has been used between different bots to locate the blocks in different rooms in the environment.

Here the master agent/main agent/ main bot acts as the head/leader, and is responsible for collecting the blocks in the room based on the basis of **Result Shared** by the client agents/client bots.

The **Task Sharing** is done between the master bot/agent and the child bots i.e. the task of locating the colours in each room is shared between the master and the child bots.

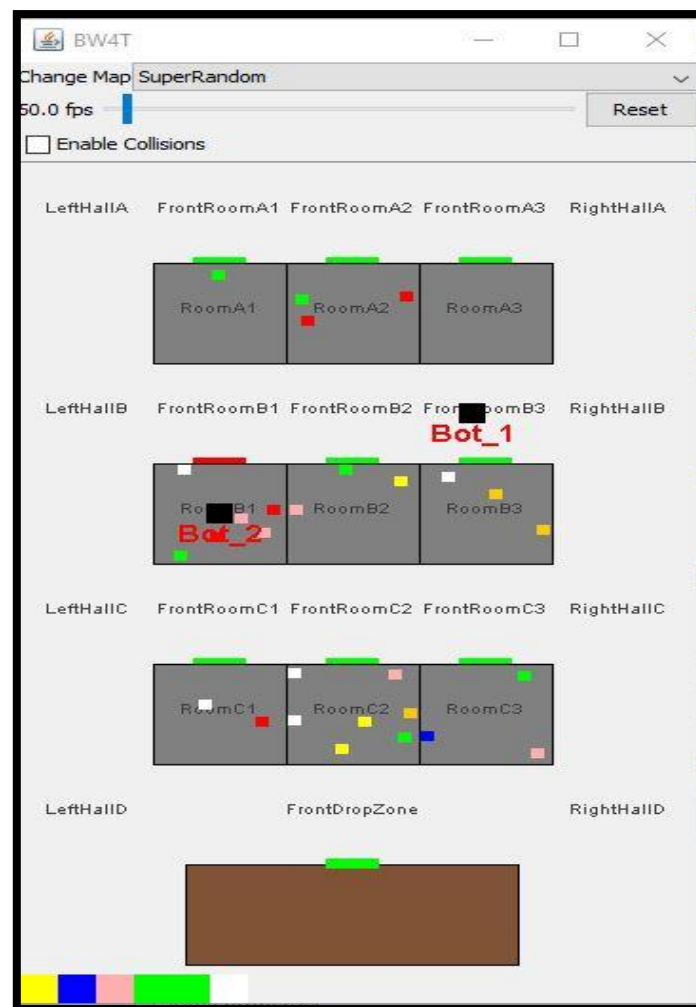
The main bot requests the child bots to concurrently help it in locating the coloured blocks in the rooms.

Once this task is completed, the main bot goes and picks the located blocks and drops them in the drop zone. The inter-agent communication is done using FIPA messages

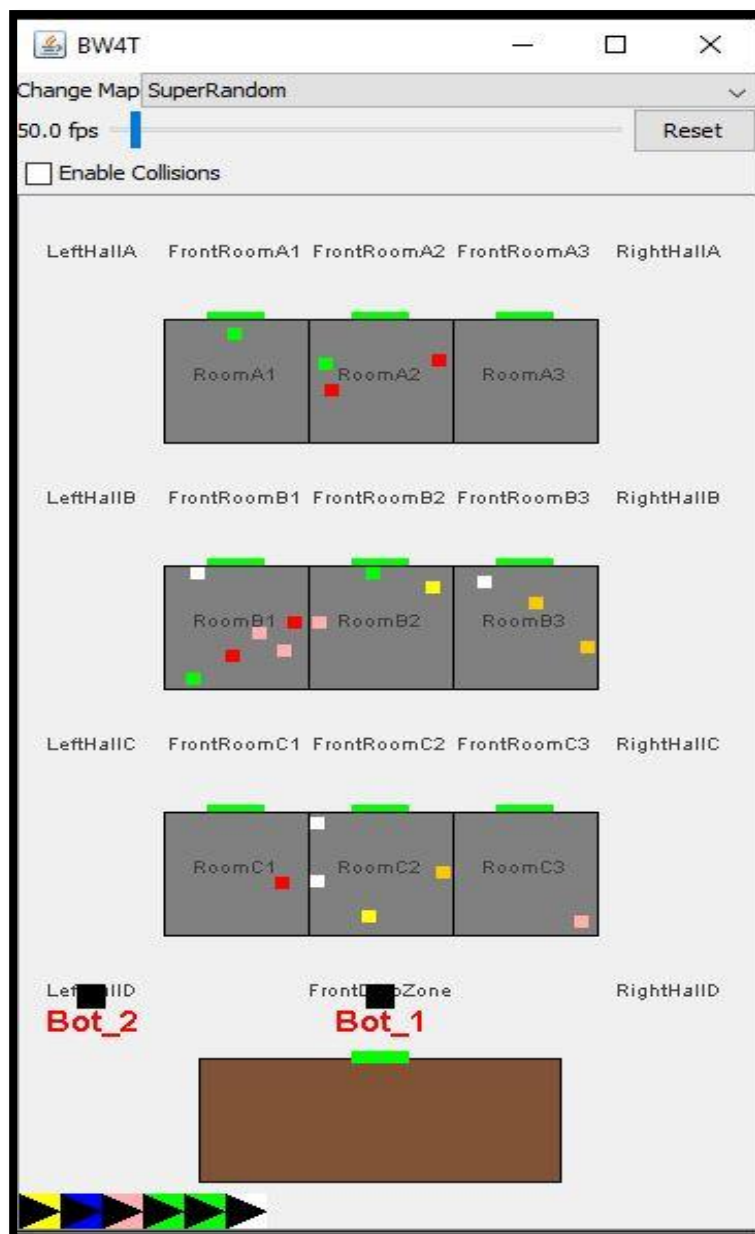
The system implemented is capable of handling different number of bots.

1. The sub-goal **agentGenerate ()** [*BlockMovement.astra*] takes the help of the child agent file [*ChildBots.astra*] to create the multiple child agents and link them to the BW4T environment.
2. Once, the child agents are linked to the environment entities. They send inform message to the parent agent [*BlockMovement.astra*] which then adds the details of these agents in the list. **+agents(list agentList)**.
3. If the size of this agentList (*contains only the child agents. Parent agent is not included*) is more than zero, then the **multipleRobotsMovement ()** sub-goal is called to implement the working of coordinated multiple bots to finish the BW4T task.
4. The child agents are sent the **information of the rooms** in the environment by the parent bot using FIPA inform message

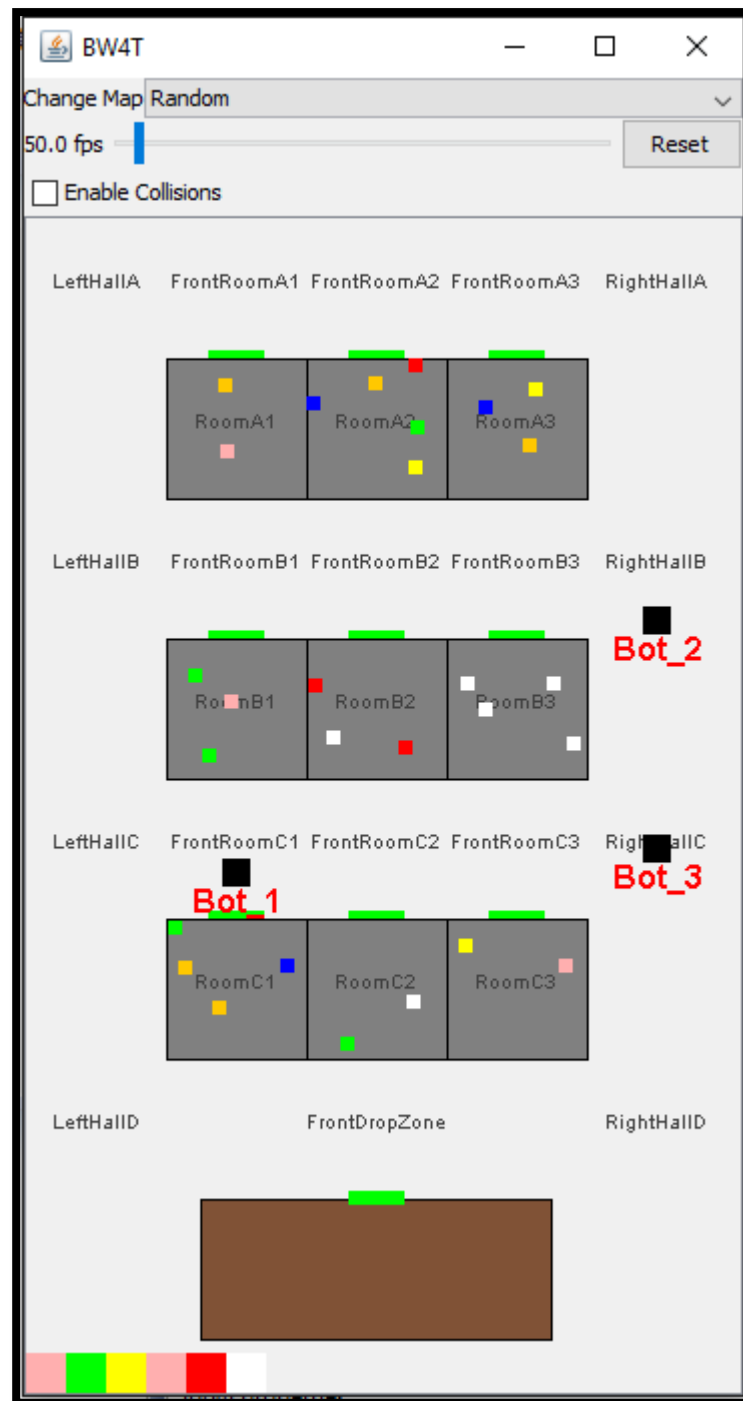
5. The **request** (FIPA Request message) is made to child bots, to travel to different rooms in the environment and collect the information of the rooms and blocks present within them
6. The child agents/bots use **findColorsInRooms ()** [*ChildBots.astra*] sub-goal to complete this task assigned to them
7. The child agents/bots share and receive the information of the blocks locations with the other child agents and the main agent.
8. The child agents/bots also share and receive the information of the room they have already visited. This avoids repetition of movement of a bot to the room whose information has already been collected
9. The main agent also coordinates with the child agents/bots to find the information of the blocks locations in the environment.
10. This information of the blocks location and room visited status is notified to the child bots.
11. Once all the rooms are traversed and the required information is acquired, the master or main bot collects the blocks on the basis of colour sequence of blocks.
12. The **Information on Blocks location** assists them to collect the blocks precisely and quickly
13. Once all the blocks are collected successfully, the system is terminated.



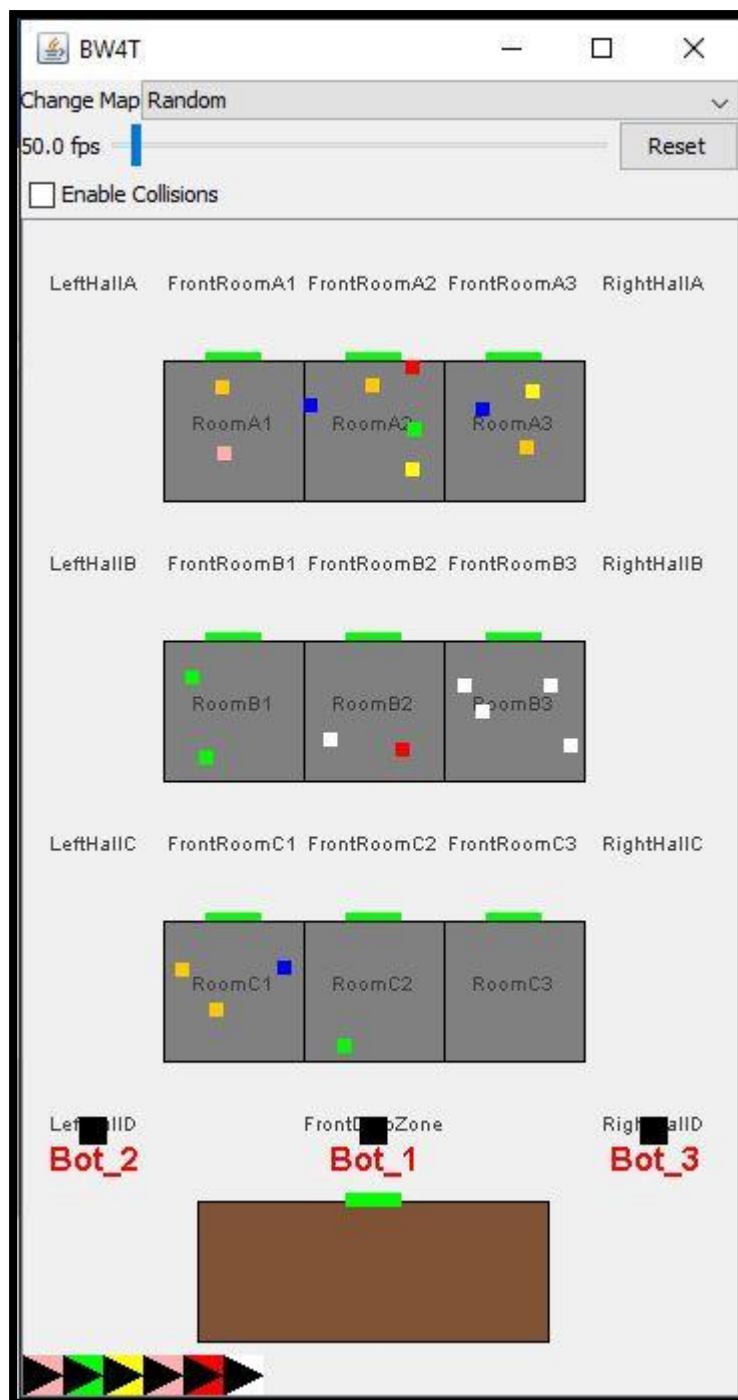
The above figure depicts the BW4T system, where Task of locating the blocks is shared between two bots



The above figure depicts the BW4T system implemented using 2 entities



The above figure depicts the BW4T system, where Task of locating the blocks is shared between three bots



The above figure depicts the BW4T system implemented using 2 entities

TASK C6:

Discretionary mark to reflect contributions that are not specified elsewhere in the marking scheme (e.g. a particularly good coordination strategy, some neat technique for minimising/optimising block retrieval)

Techniques:

1. For the single bot implementation, it was made sure that once the bot entered any room in the BW4T environment, it had the details of all the colours (coloured blocks) present in the entered room. The repetition of the coloured blocks in any room was taken into account.

It was also made sure that the colours information was updated once any block was picked by the bot.

This helped that “Bot” in fetching the blocks with respect to the colour sequence more precisely as the bot had the complete information about the room it had visited.

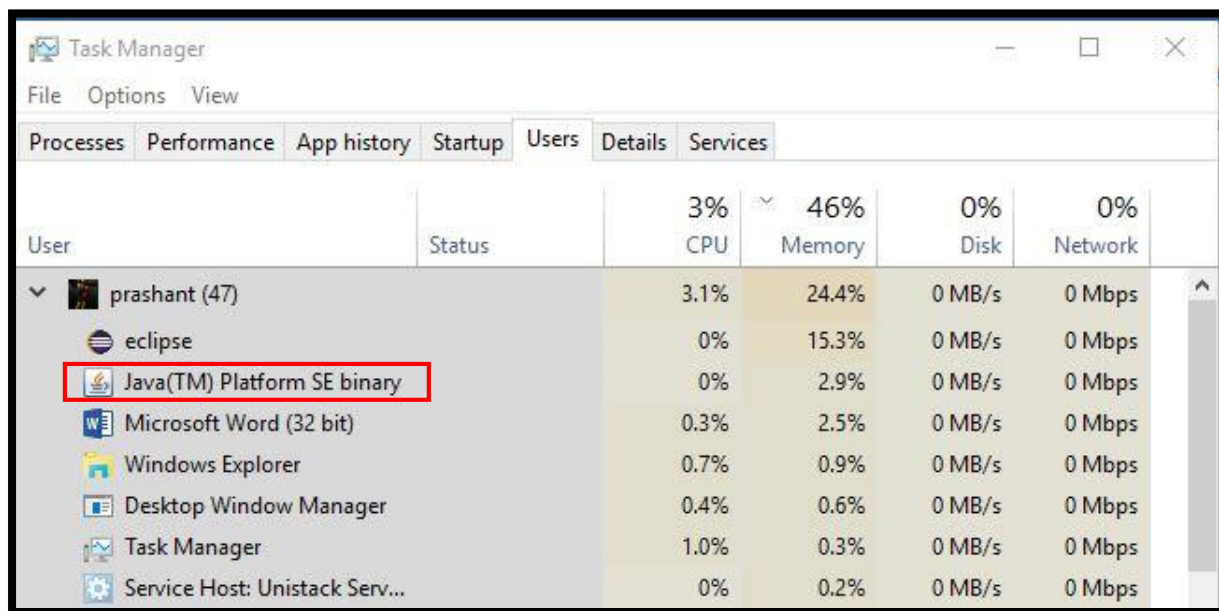
2. For both single/multiple bot’s implementations the client processes couldn’t terminate leading to un-terminated threads in the memory. The `system.exit()` was used to make sure that after the completion of the activity of the agent, the client processes were terminated.

This helped in removing the unwanted performance loss for the system.

Before Change: Server thread and client threading running even after completion of task

User	Status	CPU	Memory	Disk	Network
prashant (52)		9%	53%	3%	0%
eclipse		7.4%	35.9%	0.1 MB/s	0 Mbps
Java(TM) Platform SE binary		0%	19.4%	0 MB/s	0 Mbps
Microsoft Word (32 bit)		1.5%	2.7%	0.1 MB/s	0 Mbps
Java(TM) Platform SE binary		0%	1.9%	0 MB/s	0 Mbps
Java(TM) Platform SE binary		0%	1.3%	0 MB/s	0 Mbps
Java(TM) Platform SE binary		0.4%	1.3%	0 MB/s	0 Mbps
Java(TM) Platform SE binary		0%	1.3%	0 MB/s	0 Mbps
Java(TM) Platform SE binary		0%	1.3%	0 MB/s	0 Mbps
Java(TM) Platform SE binary		0.2%	1.2%	0 MB/s	0 Mbps
Java(TM) Platform SE binary		0%	1.0%	0 MB/s	0 Mbps

After Change: Only, the server thread running in the memory after the task is completed by the bots

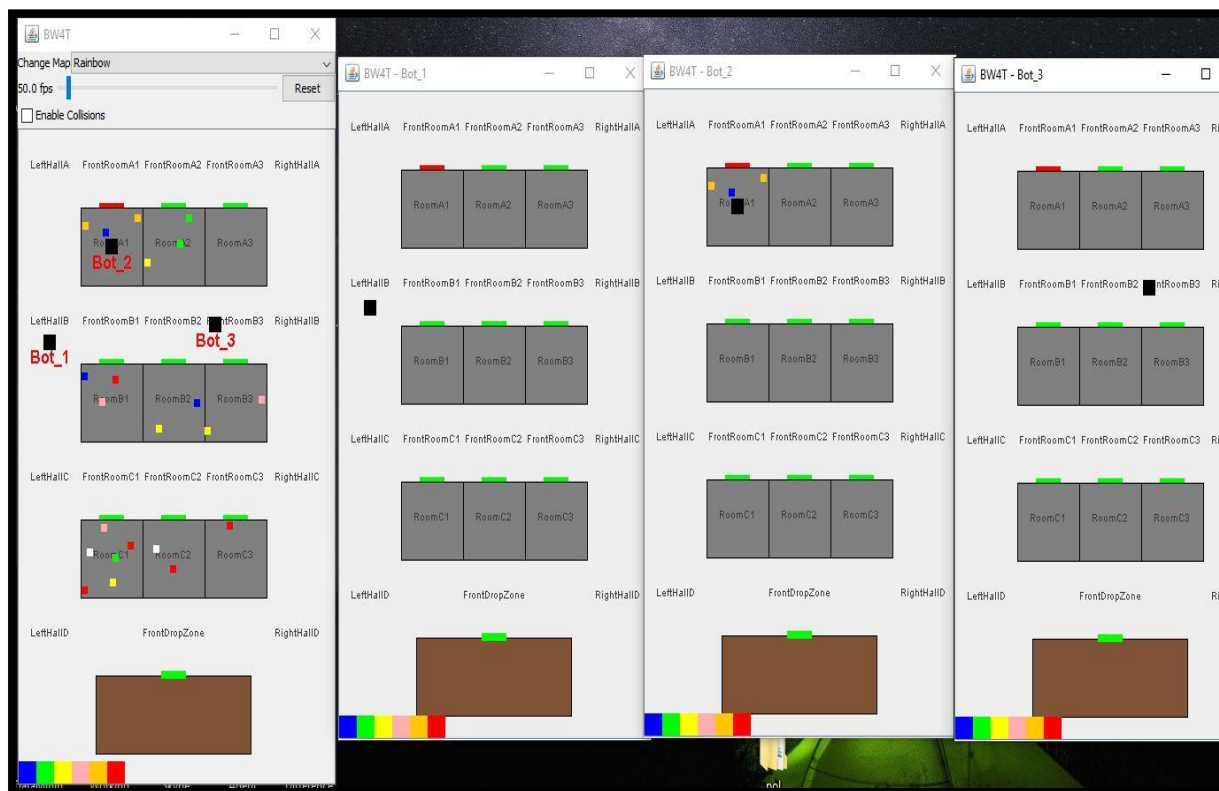


User	Status	CPU	Memory	Disk	Network
prashant (47)		3%	46%	0%	0%
eclipse		0%	15.3%	0 MB/s	0 Mbps
Java(TM) Platform SE binary		0%	2.9%	0 MB/s	0 Mbps
Microsoft Word (32 bit)		0.3%	2.5%	0 MB/s	0 Mbps
Windows Explorer		0.7%	0.9%	0 MB/s	0 Mbps
Desktop Window Manager		0.4%	0.6%	0 MB/s	0 Mbps
Task Manager		1.0%	0.3%	0 MB/s	0 Mbps
Service Host: Unistack Serv...		0%	0.2%	0 MB/s	0 Mbps

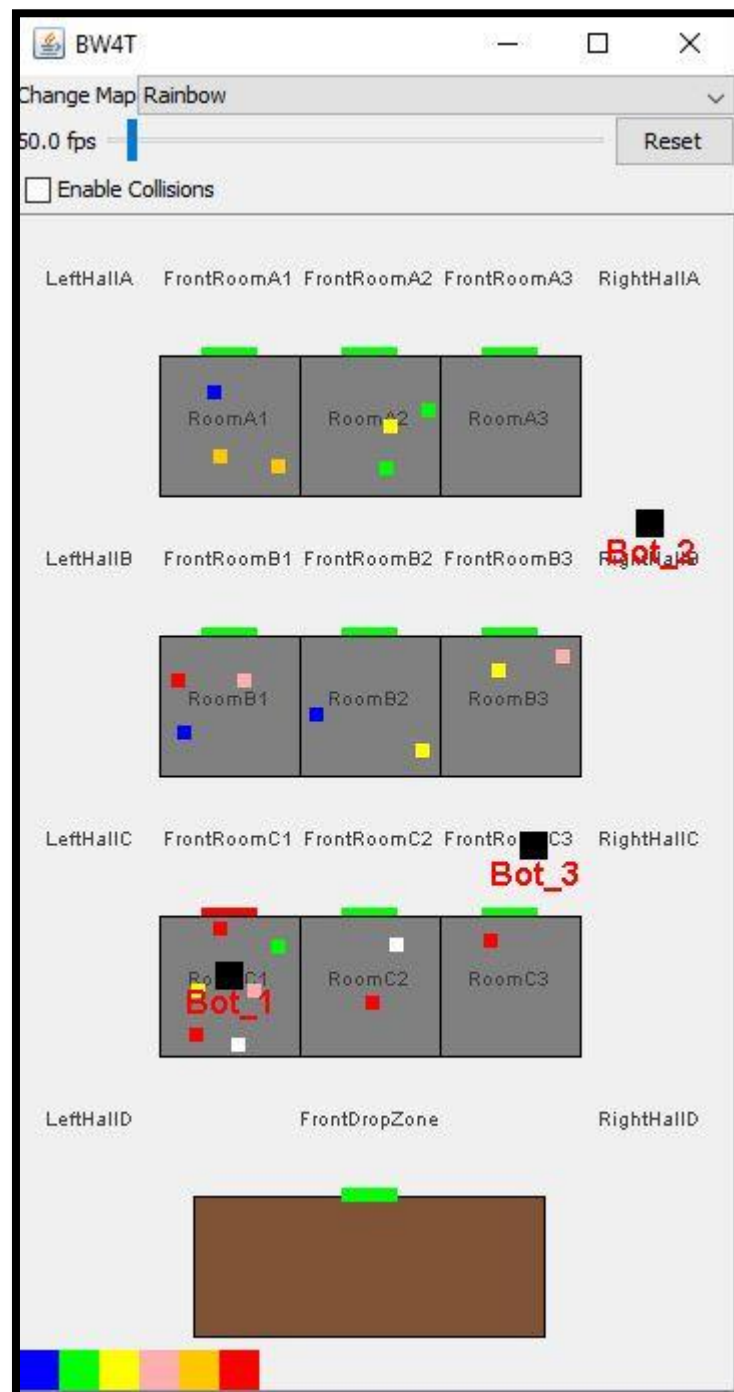
- The separate GUI for the bots was opened which led to the inconsistent view for the whole system.

The separate GUI for the bots was not launched, and the consistent view was made available.

Previous View:



New View:



4. For the implementation of the BW4T system, with multiple bots a strategy was adopted in which only a single bot was given the responsibility of picking up the blocks, while the responsibility of finding the blocks was coordinated among all the bots. This prevented the collision between the bots in the system.
5. No separate ASTRA agent files for child agent creation are used. For multiple agents creation, a generic ASTRA agent file *ChildBots.astra* is used.

Strengths and Weaknesses of the implemented solution

Weakness:

1. Concurrent access of blocks is not available in the implemented system.

Note: For few scenarios, there was a clash between the different bots while concurrently fetching the blocks. Hence the system crashed in those scenario's. To avoid this, only the master bot is allowed to retrieve the blocks.

Strengths:

1. Each Bot can visit each room and adapt to different maps
2. Consistent GUI for the system
3. Proper termination of the system
4. The system is able to locate and retrieve the next block in the specified sequence.
5. Adaptable to multiple bots

Experience with ASTRA

1. The familiarity of ASTRA syntaxes to that of Java did help to adapt to the MAS programming language quickly.
2. The ease with which it connects to the environment is really recommendable.
3. With Search Ahead automatic syntax recommendations and by pointing all the compile-time errors at same, would enable faster development of MAS systems using ASTRA.