



TECHNISCHE UNIVERSITÄT
CHEMNITZ



Agent-Based Modelling with Group Behaviour and Social Distancing

Master Thesis

Submitted in Fulfillment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Software Engineering

Submitted by: Shivam Shrivastava

Student ID: [REDACTED]

Date: 25.02.2021

University Supervisor : Prof. Dr.-Ing. Janet Siegmund & Arooba Aqeel

Company Supervisors : Dr. Mirjam Fehling-Kaschek & Dr. Corinna Köpke

Acknowledgement

First of all, I would like to express my sincere gratitude to my industry supervisors Dr. Mirjam Fehling-Kaschek & Dr. Corinna Köpke , without whom I would not have been able to complete this research thesis. With their theoretical and technical skills, they led me in the right direction whenever they felt I needed it.

I would also like to thank my academic supervisor Prof. Dr.-Ing. Janet Siegmund & Arooba Aqeel for the continuous support for my master thesis work with their persistence, inspiration and tremendous expertise.

Pursuing a thesis at Fraunhofer EMI, motivated me to explore the interactive domains of the industry, especially due to their strong work ethics and supportive environment. I would also like to thank the experts and my colleagues for their moral and ethical support during my time at Fraunhofer EMI. Without their passionate participation and input, the thesis could not have been successfully conducted.

Finally, I would like to thank my friends for their great support. I am grateful to my family's diligent efforts to provide me with the opportunities I have had in my career. I'm dedicating this work to them.

Freiburg in Breisgau, March, 2021

Shivam Shrivastava

Abstract

Agent-Based Modelling (ABM) is a modern way of modelling structures made up of autonomous, interacting agents. ABM is helpful in modelling systems whose behaviour is driven by the interaction of different entities such as groups of agents. Interaction can be designed as communication between agents. To achieve the communication between groups of agents, in this master thesis, a "leader and follower" model has been implemented in an existing ABM that has been developed by Fraunhofer EMI in the course of the EU H2020 project SATIE (Security of Air Transport Infrastructure of Europe) to represent passenger behaviour in airports. In the "leader follower model", all the followers stick to their leader in order to reach a specific target in a two dimensional environment. In this thesis, the group performance with different group size has been investigated. In the current pandemic situation, different distance rules have been proposed in order to restrict the spread of corona virus. In this thesis, the impact of different distance rules on airport performance has been analysed whereby breaking of rules by impatient passengers has been taken into account.

Keywords Agent Based Modelling, Groups of Agents, Leader, Follower, Distance Rule Violations.

Contents

Contents	4
List of Figures	6
List of Tables	9
Listing	10
List of Abbreviations	11
1 Introduction	12
1.1 Project Context	12
1.2 Motivation & Objectives	13
1.3 Research Question	15
1.4 Thesis Contribution	15
1.5 Thesis Outline	16
2 Background And Code Basis	17
2.1 Agent Based Modelling	17
2.1.1 What is an Agent?	18
2.1.2 Benefits of Agent-Based Modelling	20
2.1.3 Areas of Application	21
2.1.4 Existing Toolkit Available	22

CONTENTS

2.2	Existing Code	22
2.2.1	Path Finding Algorithm	24
2.2.2	Collision Avoidance Algorithm	27
2.2.3	Simulation Output	31
2.2.4	Simulation Layout	32
3	Implementation	35
3.1	Group Dynamics	35
3.1.1	Structure of Groups of Agents	35
3.1.2	Agent ID & Group ID	37
3.1.3	Group Behaviour	37
3.2	Global variables	45
3.3	Distance Rules	47
3.3.1	Introduction	47
3.3.2	Distance Rule Violations	48
3.3.3	Implementation Of Different Distance Rules	49
3.3.4	Plotting The Simulation Output	49
3.3.5	Automate The Simulation Process	51
4	Results	53
4.1	Group Dynamics	53
4.2	Distance Rules	55
4.3	Layout modifications	58
5	Conclusion	64
5.1	Summary	64
5.2	Outlook	65
	Bibliography	66

List of Figures

1.1	Compiled view on some of IPS functionalities with network model (bottom left) and ABM (bottom right) provided by colleagues at Fraunhofer EMI.	13
1.2	Motivation summary. *ABM: existing code developed in SATIE. The boxes 4 and 6 (light blue color) i.e. groups dynamics and distance rules are part of research topics. The boxes 1 and 5 (dark blue color) i.e. covid and airport are external sources which are modelled. The boxes 2 and 3 (slight dark blue color) i.e. infrastructure design and ABM are the basis for this master thesis work.	14
2.1	Representation of a human agent, the upper part of the image is the representation of agents and their behaviour where as the lower part is the representation of agents and their behaviour in the computer code[13].	18
2.2	Artificial world populated by agents and other objects.	19
2.3	Schematic representation of flight information attributes of agents where a ticket is a reduced set of flight information.	23
2.4	Creation of nodes and adjacent child nodes form the current agent position.	25
2.5	Path finding algorithm design.	25
2.6	Path finding algorithm with object obstruction.	26

LIST OF FIGURES

2.7	The movement of an agent from source to destination in 4 time steps. The speed of an agent is 5 meter per time steps.	27
2.8	The movement of an agent from source to destination in 4 time steps. The speed of an agent is 2.5 meter per time steps.	27
2.9	Agent scenario. Orange: agent 1 wants to go to the target and Green: agent 2 wants to go to his target without intersecting each other. Here the distance between the center point of both the agents is $x > 2 * R$ + D where R is radius of an agent, D is the distance rule 2 meter. . .	28
2.10	Agent collision scenario. Orange: agent 1 wants to go the target and Green: agent 2 wants to go to the his target but due to distance between the agent is less they are intersecting each other. Here the distance between the center point of both the agents is $x < 2 * R +$ D where R is radius of an agent, D is the distance rule 2 meter. . . .	29
2.11	Agent collision scenario. Orange: agent 1 wants to go to his and Green: agent 2 wants to go to his target but due to collision with walls and agent 1, agent 2 creates third potential angel on his right. .	30
2.12	Agent collision scenario. Orange: agent 1 wants to go to his target and Green: agent 2 wants to go to his target but agent 2 is intersecting will agent 1 as well as with the walls.	31
2.13	Airport layout with doors (blue), FIDS (red), check-in (yellow), secu- rity (green), gates (orange), dots (black) represents the queue points where agent will stand in order to do the check-in and security check processing.	33
3.1	Structure of an example group of agents; 1 in red denotes the leader and 2-6 in black are the followers.	36
3.2	Simulation of passenger movement in the airport with groups of agents presented at a specific time step. Leaders are visualized as red and followers as black dots.	39

LIST OF FIGURES

3.3	Simulation of passengers movement in the airport for an example group of agents. The leader is presented by a red and the followers with a black dotted line.	40
4.1	Box plots for each case of Table 4.1 and 100 simulations show the average duration that agents 'exist' in the airport. The black line above and below the box are called as whiskers it show the spread of data, yellow line in the box represents the median of the data, an outlier (black small circle) is defined as a data point that is located outside the whiskers of the box plot.	54
4.2	Results of 100 repeated simulations with a distance rule of 1 meter. Graphs shows the number of violations that happened over the 2000 time steps. On the y-axis, the number of violations can be observed for a particular time step. Right: The red line shows the mean and the blue lines across the border of the grey area is the +/- standard deviation.	56
4.3	Mean & Deviation graphs of distance rule violations for 100 simulation runs. The red line shows the mean and the blue lines across the border of the grey area is the +/- standard deviation for the distance rule of 1.5 meter (left) and 2 meter (right).	57
4.4	Comparison between the different distance rules. The black line above and below the box are called as whiskers it show the spread of data, yellow line in the box represents the median of the data, an outlier (black small circle) is defined as a data point that is located outside the whiskers of the box plot.	58
4.5	Comparison of distance rule violations between simple and complex layout at 1.5 meter distance.	60
4.6	Complex layout with more obstacles (yellow rectangles) which is called 'shop layout'.	61

LIST OF FIGURES

4.7 Layout comparison between simple, FIDS and shop layout. The median is given as black dotted line. The black line above and below the box are called as whiskers it show the spread of data, yellow line in the box represents the median of the data, an outlier (black small circle) is defined as a data point that is located outside the whiskers of the box plot. 62

4.8 Different movement of groups of agents across the complex airport layout. 63

List of Tables

2.1	List of different toolkit [17],[22], GPL: General Public License.	22
2.2	Attributes of the objects 'agent' and 'airport'.	23
2.3	List of objects in all three airport areas i.e. outside area, landside area and airside area. *FIDS: Flight Information Display System monitors.	33
3.1	Follower spawn positions relative to the leaders x-position X and y- position Y.	37
3.2	An example of the airport performance data set.	50
4.1	Comparison between three simulation cases with different group sizes. The average group duration is a mean value over 100 simulations and all groups.	54

Listing

3.1	C++ code snippet at the exit point.	41
3.2	Data structure of groups in C++.	43
3.3	C++ code snippet that represents the creation of a leader.	43
3.4	C++ code snippet that represents the creation of followers.	44
3.5	Python code snippet to extract information from csv all the files and generate the graphs.	49
3.6	Python code snippet shows the implementation of triggering the exe file.	51
3.7	C++ code snippet shows the implementation of capturing the com- mand line argument in the C++ code.	52

List of Abbreviations

ABM Agent Based Modelling

CA Cellular Automata

GPL General Public License

XML Extensible Markup Language

BDI Belief Desire Intention

SATIE Security of Air Transport
Infrastructure of Europe

EIS Environmental Information
System

CSV Comma Separated Value

ID Identification Number

NA Not Applicable

IPS Impact Propagation Simulation

COV Corona Virus

FIDS Flight Information Display
System

ABMS Agent-Based Modelling and
Simulation

1 Introduction

Agent based modelling (ABM) is used to model and simulate interacting agents [1]. ABM based models are capable of simulating large number of agents or groups of agents. It allows us to understand and analyse the behaviour of specific agents at any point in time [35]. A system is a collection of individual agents that interact with the environment [21]. The interactions between the agents are not sequential because agents behave individually in parallel with each other [8]. An agent interacts in many ways with the environment like e.g. path finding in a building. An agent must decide its behaviour quickly according to the current environment and situation, like e.g. avoiding obstacles and choosing the nearest doors. ABM is an empirical tool which helps to design processes and environments [34]. In this thesis work, ABM is used to model the group behaviour of agents and to analyse the impact of social distance rules in the airport. This chapter outlines the project background, problem statement, and major research contributions along with the report structure.

1.1 Project Context

This master thesis is embedded into and funded by the EU-H2020 project Security of Air Transport Infrastructure of Europe (SATIE)¹ [11]. In this project a toolkit is developed to strengthen air traffic management and airport operations against cyber-physical threats. Threat scenarios are defined in airports of three different

¹This work has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 832969. This output reflects the views of author(s), and the European Union cannot be held responsible for any use which may be made of the information contained therein. For more information on the project see: <http://satie-h2020.eu/>.

countries of Europe i.e. Croatia, Italy and Greece in order to evaluate the toolkit in operational conditions. One of the tools is the Impact Propagation Simulation (IPS) developed by Fraunhofer EMI which enables to estimate the impact of certain cyber-physical threats on the airport infrastructure. IPS is a hybrid tool that consists of two models, i.e. a network model and an ABM representing the airport infrastructure. The screenshot of IPS tool mentioned in Fig 1.1 is provided by Fraunhofer EMI colleague.

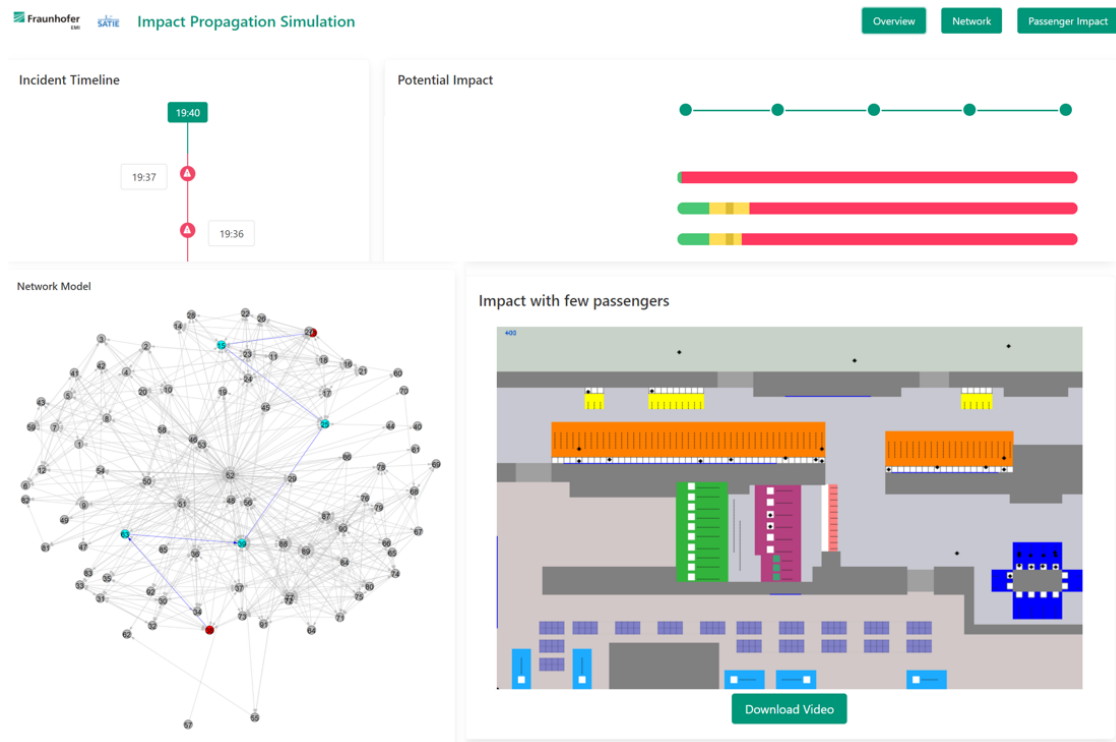


Figure 1.1: Compiled view on some of IPS functionalities with network model (bottom left) and ABM (bottom right) provided by colleagues at Fraunhofer EMI.

1.2 Motivation & Objectives

The main objective is to contribute to the ongoing project SATIE and especially to the ABM in IPS. In the existing simulation, the behavior of individual agents is implemented. The main task of this master thesis is to implement the behaviour

of groups of agents in IPS-ABM to simulate the behaviour of groups such as e.g. friends and families. It give us a realistic view of queuing and how they wait for each other in the simulation. Apart from that, in the current pandemic situation, across the globe, every country is trying to restrict the spread of corona virus (COV) by applying different distance measures, which is an area of research. In some crowded situations, people do not have the chance or patience to respect the prescribed distance rules. In this thesis research, the impact of distance rules on an example airport infrastructure has been analysed, taking into account the limited patience or time resources of people in an airport.

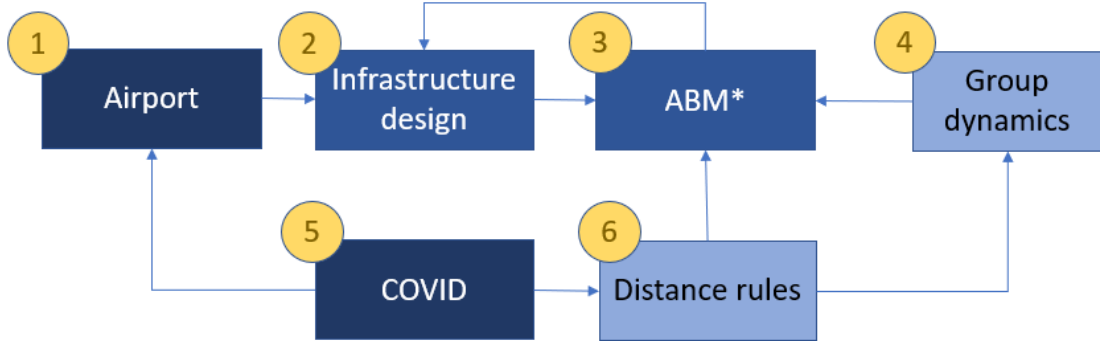


Figure 1.2: Motivation summary. *ABM: existing code developed in SATIE. The boxes 4 and 6 (light blue color) i.e. groups dynamics and distance rules are part of research topics. The boxes 1 and 5 (dark blue color) i.e. covid and airport are external sources which are modelled. The boxes 2 and 3 (slight dark blue color) i.e. infrastructure design and ABM are the basis for this master thesis work.

The motivation for this thesis is further summarized in Fig 1.2. Due to the project context, the infrastructure that is considered in this thesis is an airport (1). To improve existing or design new airports (2) models and simulation environments are needed (3), here represented by the ABM. The existing airport layout needs to be incorporated in the simulation and the passenger behaviour needs to be represented realistically. In this thesis work, group dynamics (4) have been researched in the literature and implemented in the existing ABM code ². Further, due to the COV

²The existing ABM code is developed by Fraunhofer EMI.

pandemic (5) the processes of airports in the SATIE project have been highly impacted also by distance rules (6). These distance rules have been researched in the literature and implemented in ABM between the groups of agents to analyse the impact on the performance of the example airport.



1.3 Research Question

The implementation of groups of agents in ABM comes with research questions, i.e. how groups of agents behave collectively, how the agents walk together, how the members of the group wait for each other or how to make the groups look realistic. Especially the COV distance rules, requires an effort to understand the impact on the airport and the needed balance between safety and impact on the performance of the airport.



1.4 Thesis Contribution

In the past, many researchers have contributed to ABM. However, literature for groups dynamics and COV distance rules is very sparse. In this thesis, group dynamics and distance rules in airports is discussed, quantified and the results are presented. The primary contributions of this thesis are summarized as follows:

1. Implementation of groups of agents in the existing ABM developed in SATIE.
2. A class of *agent group* is introduced to dynamically generate groups of agents with different sizes.
3. Implementation of realistic behaviour of agents in the simulation i.e. how they wait for each other after different processes in the simulation.
4. Implementation of different distance rules between the groups of agents and individual agents.

5. Introduction of patience count as an attribute of an agent is introduced.
6. Distance rules and the patience count of each agent is used to analyse the impact on the performance of the example airport.
7. The layout of the airport is modified to compare the performance of the airport under varying situations.

1.5 Thesis Outline

The remainder of this thesis is structured as follows. First, an introduction to the background of ABM and the existing ABM code given in Chapter 2. Chapter 3 presents the literature survey on group dynamics and the implementation of it in the existing code. Further, the literature survey and the implementation of distance rule violations is included. The results of the group dynamics and distance rule violation implementation are represented in Chapter 4. Finally, the report concludes with a summary of this thesis.

2 Background And Code Basis

2.1 Agent Based Modelling

Agent based modelling (ABM) is the computational study of social agents as evolving systems of autonomous interacting agents. ABM allows to test different hypothesis related to the attributes of agents, their behavioural rules and their interaction [12]. In ABM, the relationship of the agents is not totally parallel but rather it is reciprocal i.e. agents can change their behaviours according to the behaviour of their surroundings [4].


The main strength of ABM is that the individual agents have full control over their future decisions. ABM first began to be used in the academic research in the 1960s and 1970s. Before this, the most common model used for the simulation was spatial interactions or diffusion models. In spatial models, large diverse groups of people were considered as one group and each of them were given the same behaviour and movements [14].

ABM has evolved from Cellular Automata (CA), which has been developed in the 1960s. Each CA is essentially a computer grid. In the grid, each cell has a set of possible states. It runs through a number of iterations and at each iteration each cell looks at the state of the neighbouring cell and changes its state accordingly [15].

In ABM, the theoretical hypothesis of agents behaving like passengers in an airport is converted into micro-specifications, i.e. a set of rules that specify how an agent behaves and responds to the environment. Once the environment is properly populated with agents, the micro-specification can be implemented and the

simulation to evaluate the result can be performed [28].

2.1.1 What is an Agent?

From an ABM perspective, there are certain features that are common to most of the agents, which are explained briefly in the following [13]: 

- **Autonomy:** Agents are autonomous units, but they are capable of sharing information with other agents in order to make decisions independently and collectively.
- **Heterogeneity:** An agent is the representation of a human or some other entity such as e.g. cars and insects. They have attributes such as e.g. age, gender and size. These attributes are drawn from distributions.
- **Perception:** Agents can perceive their environment as well as other agents in that environment [28].
- **Memory:** agents have a memory in which they keep track of their current and previous states [28].

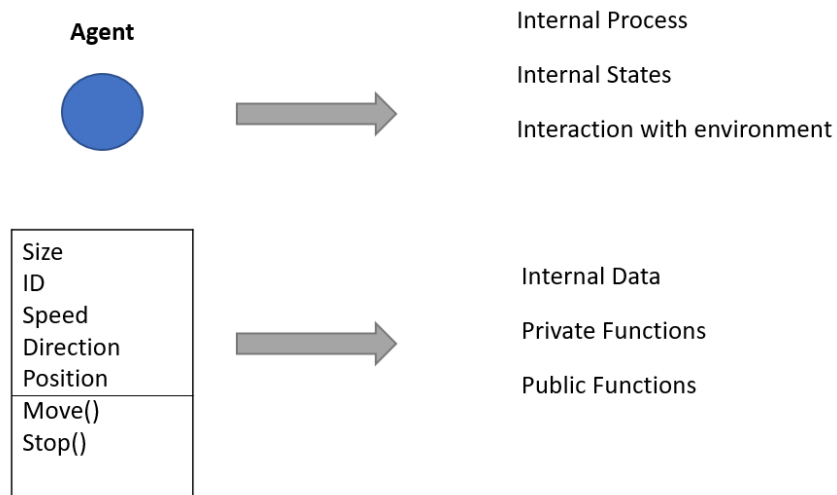



Figure 2.1: Representation of a human agent, the  upper part of the image is the representation of agents and their behaviour whereas the lower part is the representation of agents and their behaviour in the computer code[13].

Fig 2.1 represents human agent. The attributes given here are examples and can vary. Thus, different simulations can have different attributes. The Move() function generates a route from source to target. A collection of multiple agents interacting with each other or interacting within the environment is termed as ABM.

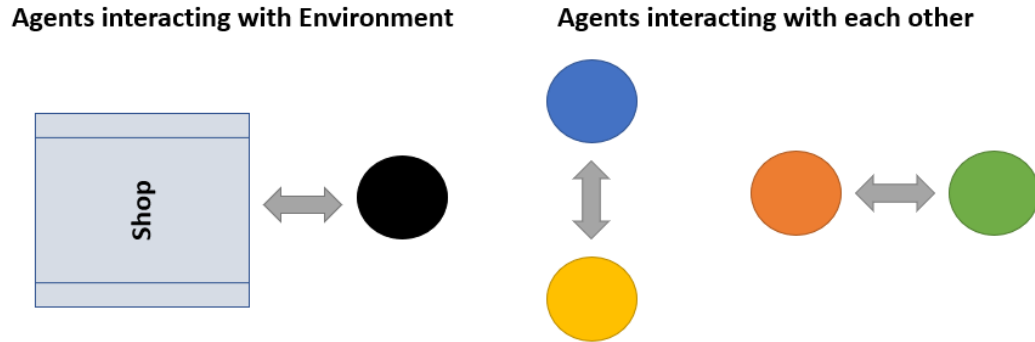


Figure 2.2: Artificial world populated by agents and other objects.

ABM is composed of three main components besides the agents which are given in the following:


- Rules
- Environment
- Time

Rules

The agents presented in Fig 2.2 possess rules that affect their behaviour and relationship with other agents and their surrounding environment. The rules are typically derived from numerical calculation or data analysis. They can be applicable to all the agents or agents can have individual rules. An example for a general rule would be if an agent in an airport, i.e. a passenger, wants to board a flight, he first needs

to pass the security check. Rules are typically implemented around if-else conditions. Once a condition is satisfied the agent will carry out a certain functionality, otherwise the agent will perform an alternative specified action [13].

Environment

The environment in terms of ABM is the space in which agents and other objects exist. An environment can be abstract or can have objects in the space. A simulation can have multiple environments. Further, an environment can be static or dynamic where the latter means that it can change over the simulation time steps [3]. An agent within an environment is spatially explicit, i.e. the agent has a location in the geometrical space. The latter is important as an agent is required to have a specific location to generate a route [13]. 

Time

The third component of ABM simulations is time. A simulation runs for a specific number of time steps. In each time step, each agent updates its location in the environment. A time step represents e.g. a millisecond, a second, a day or a year [3].

2.1.2 Benefits of Agent-Based Modelling

The benefits of ABM compared to other modelling techniques are described below

- **ABM captures emergent phenomena:** ABM can be used when individual behaviour is non-linear, be classified by thresholds and if-then rules [7].
- **ABM provides a natural description of the system:** ABM is suitable for simulating a system composed of behavioural entities. ABM can be used when activities describe a system better than processes.
- **ABM is flexible:** Flexibility makes ABM multidimensional i.e. it is easy to add more agents and entities in the environment.

2.1.3 Areas of Application

ABM is a powerful modelling and simulation technique that has a wide number of applications. Some example applications are given in the following:

- **Evacuation Flow** Life threatening situation like fires or terrorist attacks in buildings trigger panic and cause crowd stampedes which leads to fatalities and injuries. To understand and avoid these kind of scenarios, ABM is appropriate to incorporate human behaviour. It helps to analyse if the design of a building is safe [27].
- **Traffic Flow** ABM is used to simulate traffic systems and to identify potential bottlenecks of the system. ABM allows to model pedestrians and traffic control systems to study e.g. traffic signal priorities, to visualize results and to predict the potential issues [25].
- **Stock Market Simulation** ABM allows to understand how market trends dynamically evolve to an equilibrium or how they drifts away from it. ABM helps to analyse how a small change in a initial parameter could change complex financial markets [33].
- **Safety** ABM allows to model the safety operations from disturbances or hazardous scenarios. There are several scenarios which can be modelled such as radar is not working, an human agent (pilot) is affected by usage of alcohol, pilot gives the wrong position of the aircraft etc [31].
- **Archaeology** Archaeology uses ABM to analyse and understand the societal change, human-environment interaction. ABM allows to model human evolution, culture evolution in contemporary archaeological simulation[23].
- **Biological Process** ABM allows to model cancer and immune cells individually. Each cell is represented as an individual agent and can have individual attributes like e.g. birth, death rates, position, state and immune characteristics.

Varying model parameters and rules allow to understand the characteristics of successful and unsuccessful treatments[26].

2.1.4 Existing Toolkit Available

There are multiple open source packages and licensed versions available for the development of ABM environments. Table 2.1 contains some example ABM toolkit.

Toolkit	Programming Language	License
Swarm	Objective-C, Java	GPL
RePast	Java, Python	GPL
Mason	Java	GPL
NetLogo	NetLogo	Free, but not open
JADE	Java	open source
JAMES II	Java	open source
MaDKit	Java, C++	Paid
SeSAm	Java	Paid
Jadex BDI	Java, XML	Open Source
JIAC	Java	Open Source

Table 2.1: List of different toolkit [17],[22], GPL: General Public License.

In the following the ABM code especially designed by Fraunhofer EMI in the project SATIE to represent the passenger movement in an airport is presented.

2.2 Existing Code

The ABM SATIE simulation is written in the C++ programming language. In the airport simulation, different functionalities are implemented like the creation of an agent, agent rules and the environment for the agents. To this end, the code is divided into different classes and each class is responsible only for the specified functionality. Note, the existing code only handles individual agents and does not yet take group dynamics into account.

	Name	Type	Value	
Agent	Unique ID	Integer	-	
	Ticket	Object	-	
	Flight Information	Object	-	
	Size	Double	0.8 m Diameter	
	Speed Average	Double	1.5 m per time step	
	Speed Divergence	Double	0.5 m per time step	
	Space To Walls	Double	(Size/2) + 0.5 m	
	Space Between Agent	Double	Size + 0.25 m	
	Position	Object	-	
	Direction	Object	-	
	Side	Enum	Outside (at the time of spawning)	
	Airport	Side	Enum	Landside, Outside, Airside
		Flight Information	Object	-

Table 2.2: Attributes of the objects 'agent' and 'airport'.

An agent has multiple attributes mentioned in Table 2.2. The unique ID attribute is assigned to the agents as they get created and it is incremented by one for every new agent. The size of each agent is kept at 0.4 meter radius in the existing code. The speed of each agent is generated randomly with the speed average and speed divergence variable. The space between the agents to the wall is calculated from the radius of an agent. The airport side is an attribute of an agent that will be changed as an agent moves from one target to another. In the existing code, the airport side is stored in the enum which has three sides, i.e. outside, airside, landside.

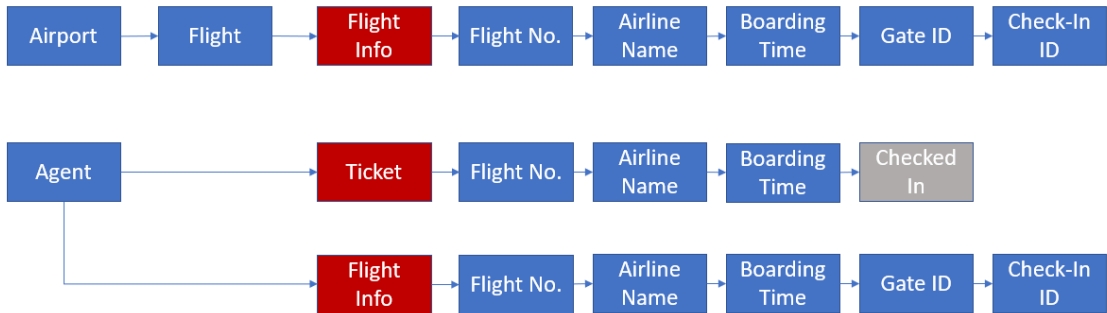


Figure 2.3: Schematic representation of flight information attributes of agents where a ticket is a reduced set of flight information.

Fig. 2.3 presents the flight information attributes. In the code, the flight infor-

mation attribute is also part of the airport object. In the agent class, every agent has a ticket and at the time of creation this ticket contains only parts of the general flight information object such as flight number, airline name, boarding time but also the additional checked-in variable. The checked-in variable is a boolean variable to check if an agent has checked-in or not. Apart from that, when an agent goes to the FIDS a flight information object is assigned containing flight number, airline name, boarding time, gate id and check-in id. The flight number is generated by a random integer generator. In the existing code, the airline has an enum class which contains three values, i.e. Lufthansa, Ryanair, Eurowings. In the existing code, there is a probability where agents do not have luggage and they have done the web check-in, in this case, the checked-in attribute of a ticket is set to true.

2.2.1 Path Finding Algorithm

To move from the current position to a specific target, an agent needs a route or path. Prior to moving, the path finding algorithm of the existing code generated a route of child-nodes and nodes.

Fig 2.4 shows the creation of child nodes and nodes. From the nodes (represented by blue color), 5 child nodes (represented by green color) are created in different random angles. The space between all child nodes and nodes is defined as 5 meter. The reason behind 5 meter distance is to have less computational efforts i.e. if the distance is less it will create more nodes and child nodes which will impact the computation processing of the simulation.

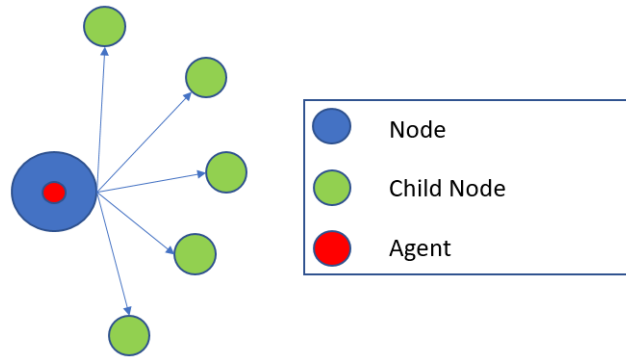


Figure 2.4: Creation of nodes and adjacent child nodes form the current agent position.

An agent creates its path from source to target by creating nodes and child nodes as described in Fig 2.5. The selection of a child node is based on the shortest distance between the source and the target. If one of the child nodes has a shorter distance to the target compared to the rest of the four child nodes then that node will be selected.

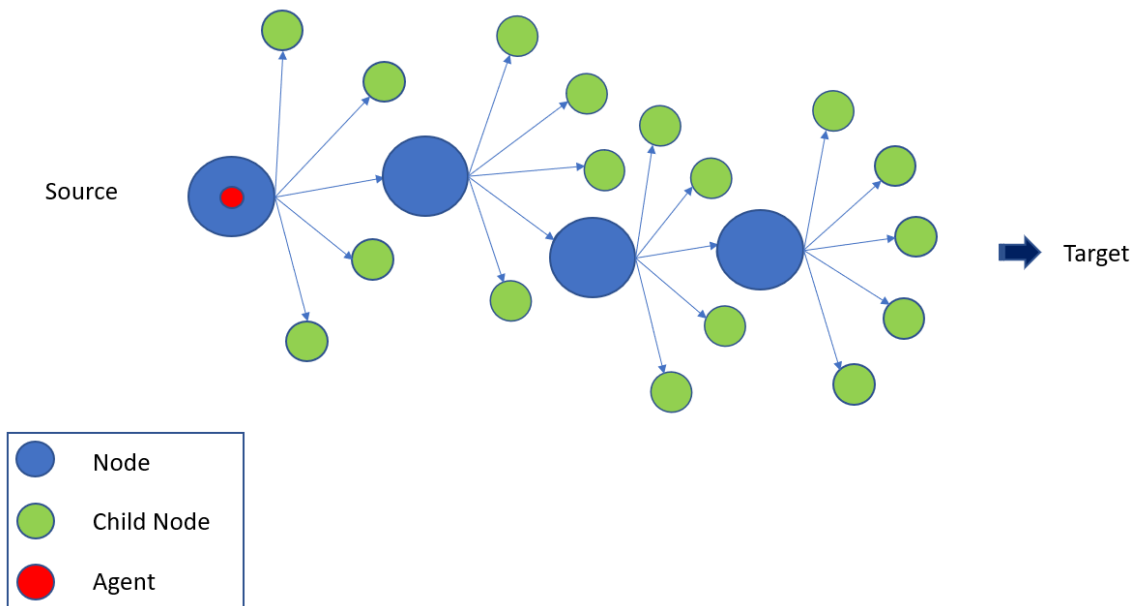


Figure 2.5: Path finding algorithm design.

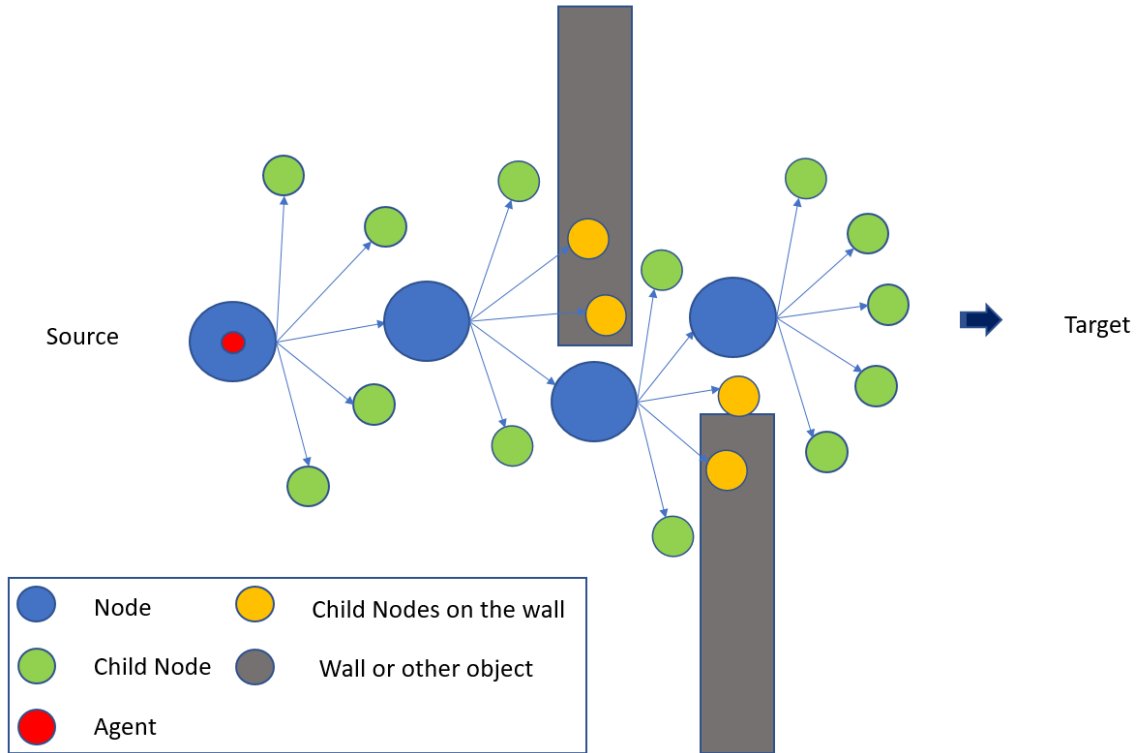


Figure 2.6: Path finding algorithm with object obstruction.

There are chances while creating child nodes that they may fall onto an object like e.g. a wall. In that case, an agent would bypass the object which would not look realistic and thus the algorithm avoids selecting these nodes. In Fig 2.6, walls are present between the source and the target. Some child nodes overlap with the walls. To extract the right path, the algorithm avoids the child nodes which are on the walls. As we can see from Fig 2.6, the child nodes with an orange color and a path intersecting with the object's surfaces are avoided even if they are closer to the target. One of the neighbouring child nodes are selected to reach the target.

Note, the path finding is performed prior to actual walking, agents calculates the path from source to the interim targets and once it reaches to the interim target it calculate the path to next interim target, this process continues until it reaches to the main target, but agent does not calculate path from source to main target beforehand. Only if the route has been successfully created using the algorithm the agent can walk to the target using the route consisting of nodes.

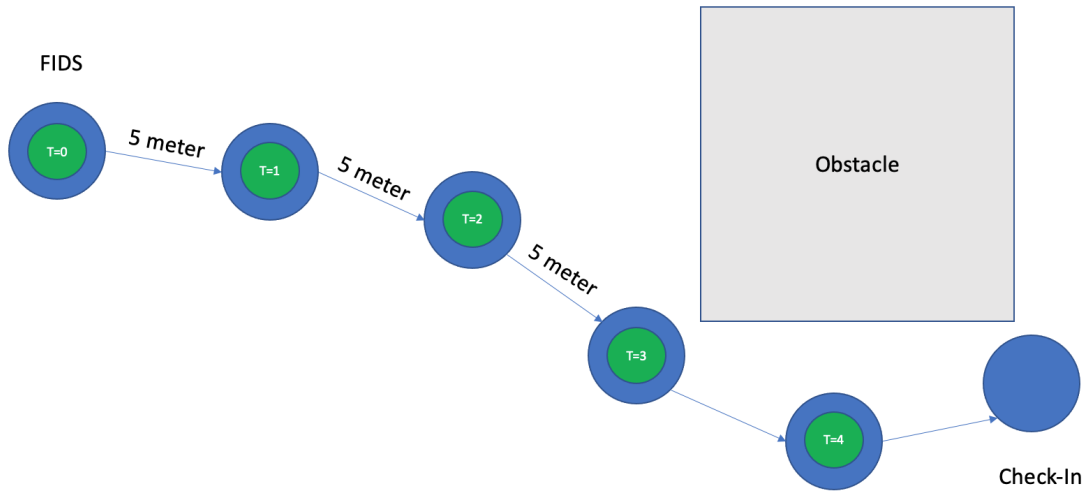


Figure 2.7: The movement of an agent from source to destination in 4 time steps. The speed of an agent is 5 meter per time steps.

Fig 2.7 shows the movement of an agent from the example source FIDS to the example target check-in. The path consisting of nodes with 5 meter distance is presented along with the steps the agent will take.

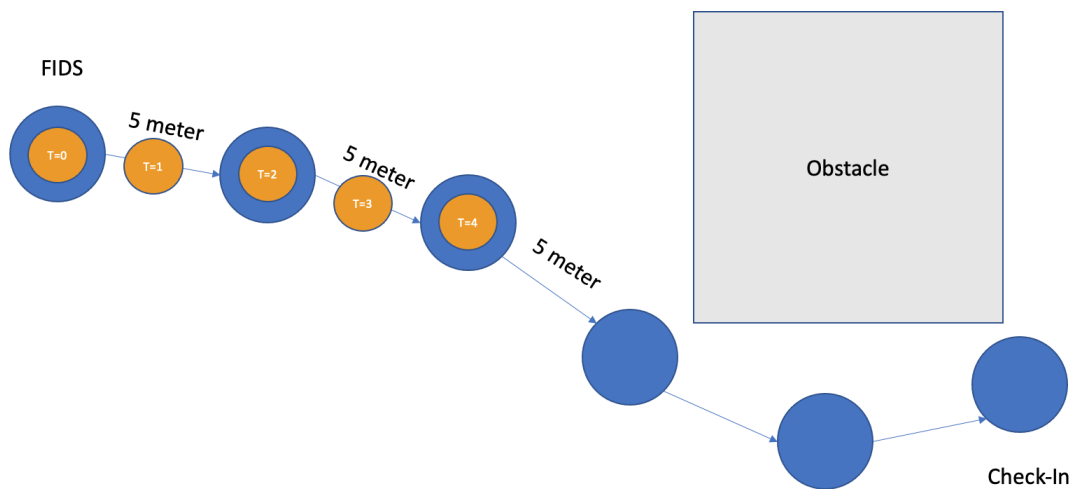


Figure 2.8: The movement of an agent from source to destination in 4 time steps. The speed of an agent is 2.5 meter per time steps.

Fig 2.8 shows the movement of an agnt on the same path as in Fig 2.7. However this agent moves with a smaller velocity and thus covers less nodes in the same

amount of time.

Even if a suitable route has been created that leads to the target, the agent taking that path might collide with other agents on its way.

2.2.2 Collision Avoidance Algorithm

In the Fig 2.9 two agents with the same velocity and same agent radius are moving towards their targets. The distance between both the agent's center is large and they are not intersecting each other. Hence there will be no collision and in this case the potential next position on the path can be selected.

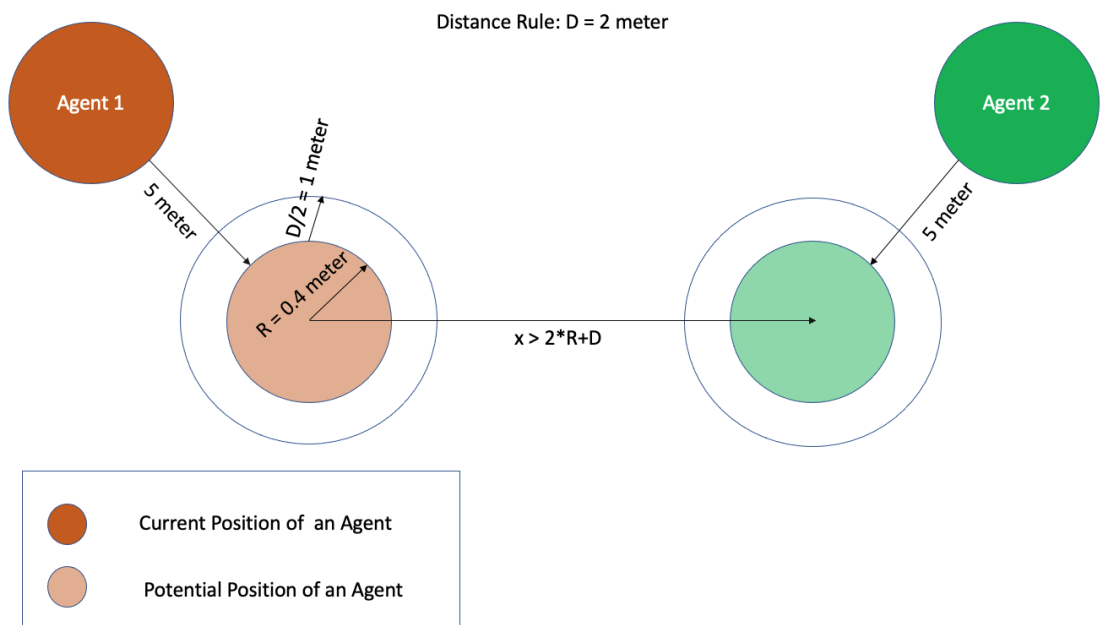


Figure 2.9: Agent scenario. Orange: agent 1 wants to go to the target and Green: agent 2 wants to go to his target without intersecting each other. Here the distance between the center point of both the agents is $x > 2 * R + D$ where R is radius of an agent, D is the distance rule 2 meter.

Fig 2.10, both the agents move towards their target and the distance between the agent's center is smaller than $2 * \text{agent radius} + \text{distance}$. In this case, they would collide in the next step. Thus, both the agents will try to create at a random angle to the right a new position to walk to. The distance between current position and new potential position to avoid the collision depends on the agent velocity.

2 Background And Code Basis

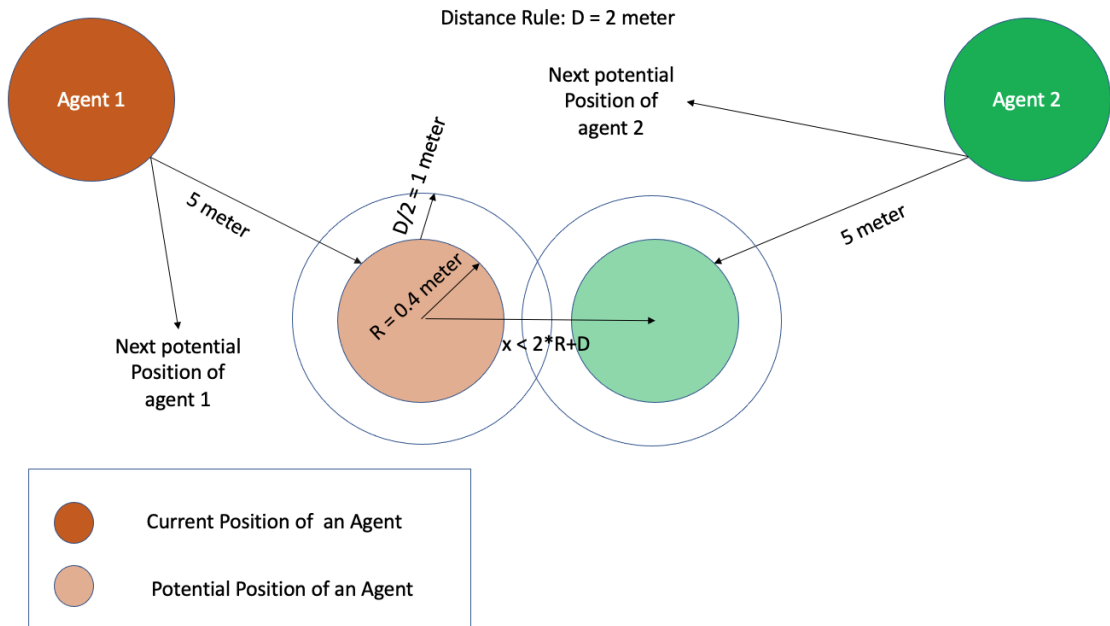


Figure 2.10: Agent collision scenario. Orange: agent 1 wants to go to the target and Green: agent 2 wants to go to the his target but due to distance between the agent is less they are intersecting each other. Here the distance between the center point of both the agents is $x < 2 * R + D$ where R is radius of an agent, D is the distance rule 2 meter.

Fig 2.11 shows the collision of two agents with the same velocity when the next potential position of an agent falls on the wall. In that case, agent 2 creates another random angel on the right to avoid the collision with the wall.

2 Background And Code Basis

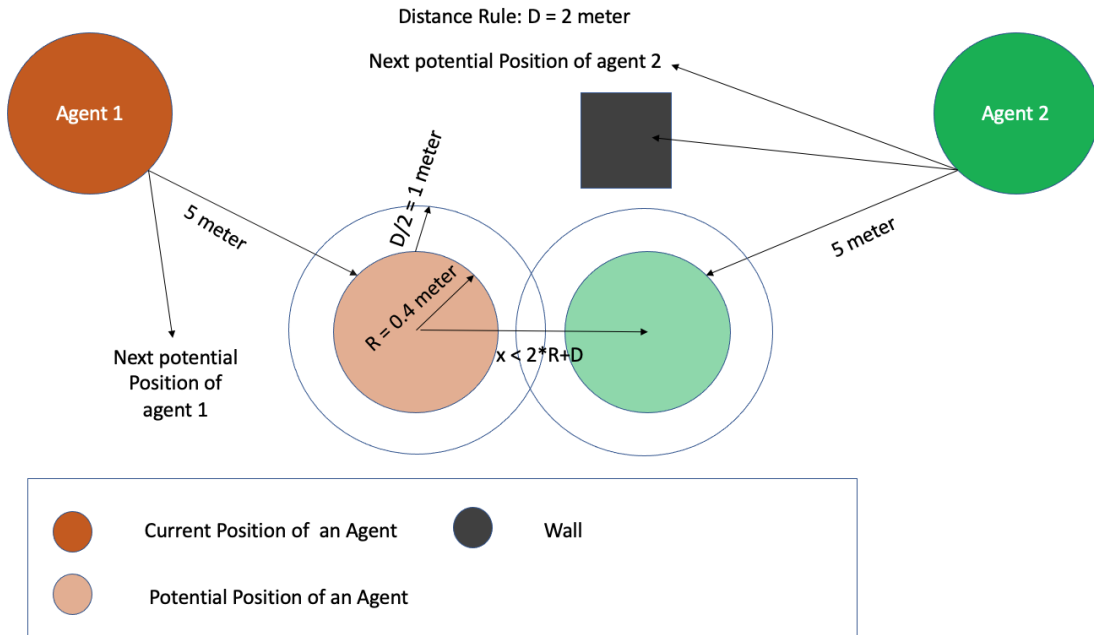


Figure 2.11: Agent collision scenario. Orange: agent 1 wants to go to his and Green: agent 2 wants to go to his target but due to collision with walls and agent 1, agent 2 creates third potential angel on his right.

In Fig 2.12, agent 2 tries to avoid the collision with another agent 1 and creates two potential angels on the right side . As we can see from Fig 2.12, all the potential next positions are either intersecting with a wall or with agents. In this scenario agent 2 cannot move as he could not find a suitable potential next position to walk to. Thus, this agent needs to wait for one time steps to try moving again.

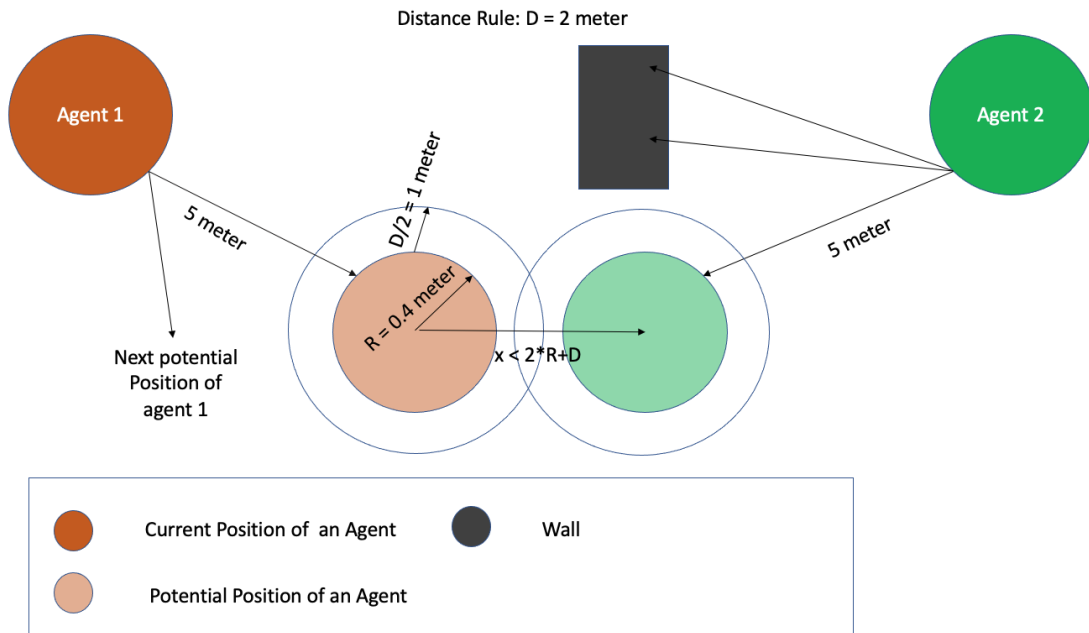


Figure 2.12: Agent collision scenario. Orange: agent 1 wants to go to his target and Green: agent 2 wants to go to his target but agent 2 is intersecting will agent 1 as well as with the walls.

2.2.3 Simulation Output

Each simulation run generates data sets about the agents. The data can be used to analyze the performance of the airport and further in the existing code it is used for generating the user interface. In the simulation, two data sets are generated:

- Agent Data
- Airport Performance

Both the data sets are written in CSV files. The agent data file contains the information about each time step, agent ID of all existing agents in the respective time step, agent position x and position y at each time step. The airport performance data set contains the information about each time step, the number of agents in the outside area, the number of agents in the landside area, the number of agents in the airside area and a boolean value to check if the airport is evacuating.

The position of an agent i.e. x and y coordinates are used to generate the user interface by python scripting. In the C++ code, the positions of all the objects and areas are defined i.e. walls, check-in groups, security check area, gates, outside, landside, airside area. These coordinates are used in the python script to design the airport layout. From the agent data CSV file, the position of an agent is extracted and used to map them on the layout. A python script generates an image at each time step. This image contains the positions of all the agents present in that particular time step. Once the python script finishes to write the images, a video is created by combination of all the images. In this way, the simulation results are summarized in a video.

2.2.4 Simulation Layout

In the airport simulation, the layout is divided into three different sections (see Fig 2.13):

- Outside area,
- Landside area and
- Airside area.

All these areas contain different objects and each object has certain functionalities which are given in Table 2.3.

Area	Objects	Number
Outside Area	Doors	2
	Spawning of agent area	1
Landside Area	Check-in group	2
	Check-in counter	20 (10 counters in each check-in group)
	FIDS*	3
	Security queue	2 vertical and 5 horizontal queue
Airside Area	Gates	3
	FIDS*	2

Table 2.3: List of objects in all three airport areas i.e. outside area, landside area and airside area. *FIDS: Flight Information Display System monitors.

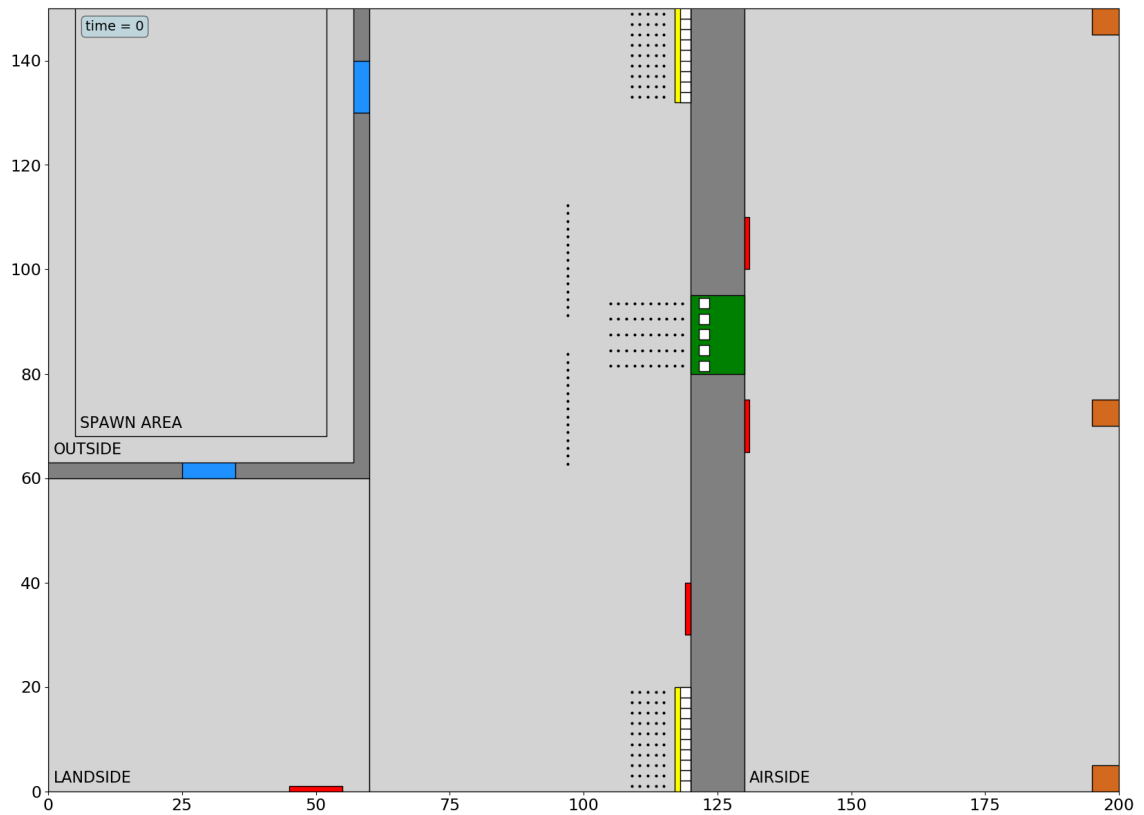


Figure 2.13: Airport layout with doors (blue), FIDS (red), check-in (yellow), security (green), gates (orange), dots (black) represents the queue points where agent will stand in order to do the check-in and security check processing.

Outside Area

The outside area is responsible for the spawning of an agent. It is linked to the landside area through two doors, the agents can move from outside to landside through the doors. At the time of spawning of an agent, they get a random position in the outside area. An agent will get the nearest door as the next target which leads an agent to the landside area.

Landside Area

The landside area in the simulation has different functionalities. It contains two check-in groups which have 10 counters each, corresponding to each counter there is a queue that means each check-in group contains 10 queues. In each queue, 5 queue points are defined for the agents to wait. The landside area has three Flight Information Display System monitors (FIDS). These FIDS set the flight information of an agent like gate ID, airline and flight number. The security check has 5 security check counters and two types of queues, i.e. horizontal and vertical queues. There are 2 vertical queues placed next to each other, in each queue 15 queue points are defined to wait. There are 5 horizontal queues placed next to each other with a specified distance. In each of these horizontal queues, there are 10 queue items for the agents to wait. After passing through the vertical and horizontal queues, the agents choose the least crowded queue to go to the airside area.

Airside Area

The airside area contains three gates. Once the agents have passed the security and as soon as they reach to the gate they get deleted from the simulation. The airside area is also equipped with two FIDS.

3 Implementation

3.1 Group Dynamics

As discussed in Chapter 2, the existing C++ simulation environment only processes individual agents but does not take into account group behaviour. One of the main goals of this work is to introduce group dynamics by adding groups of agents.

In order to implement and analyse the group behaviour of agents, communication between the agents in the groups plays a vital role [24][30]. The agents must be able to share information in the group. Real-time simulation of groups of agents comes with enormous amount of complexity. We need to control the overall action of groups of agents[10]. To achieve this, a leader and follower model has been designed based on [32] where the idea of agents being connected to each other is presented. Each group has a leader and a certain amount of followers. The leader receives a route from target to target (e.g. from door to check-in counter) and the rest of the followers copies the same path.

A leader with group of agents could contribute not only to our understanding of human social phenomena but also the development of teams of agents with practical applications [16]. Practical applications are in spread in different fields, such as monitoring, surveillance and entertainment [6], [2].

3.1.1 Structure of Groups of Agents

In this thesis work, groups of agents are designed with the following structure:

- Leader

3 Implementation

- Follower
- Agent ID
- Group ID

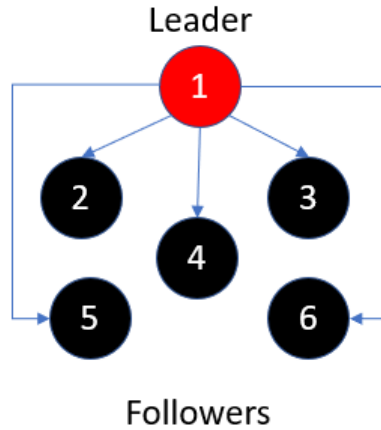


Figure 3.1: Structure of an example group of agents; 1 in red denotes the leader and 2-6 in black are the followers.

Fig 3.1, shows the creation of one example group of agents. The first element is the leader of the group whereas the remaining are the followers. All groups are based on the same structure but different group sizes are possible to be created in the simulation environment. A group with only one agent, i.e. the leader, represents an individual agent - in the case of an airport this would be e.g. a passenger that travels alone. The structure presented in Fig 3.1 is used to spawn groups of agents. The position of the leader is generated randomly in a specified spawn area where all the individual agents, as well as groups of agents, will get created. The position of the followers are relative to the leader's position of that group.

Table 3.1 gives the position of the group members.

3 Implementation

Followers	x-Position	y-Position
2	X - 0.5	Y - 0.5
3	X + 0.5	Y - 0.5
4	X - 0	Y - 1
5	X - 0.5	Y - 1.5
6	X + 0.5	Y - 1.5

Table 3.1: Follower spawn positions relative to the leaders x-position X and y-position Y.

The relative positions in the group will change as they start moving and the followers copy the route of the leader.

3.1.2 Agent ID & Group ID

Every agent has a unique ID to distinguish him from the other agents. When an agent is created the counter for this agent ID is incremented so that every agent gets a unique ID. The agent ID helps also to create groups of agents and to keep track of the agents belonging to a particular group.

All agents have a group ID identifying the group they belong to. The group size can vary at the time of the spawning of the agents. At the time of creation, all the group members belonging to one group get the same group ID. Following this approach, the agents belonging to the same group can be tracked. In order to maintain groups of agents in this work, MultiMap has been used to maintain all groups of agents, which is part of the Standard Template Library of C++. It allows to have multiple values with the same key, which helped to assign the same group ID to agents belonging to the same group.

3.1.3 Group Behaviour

With the help of these attributes, i.e. agent ID and group ID, groups of agents with random sizes that perform actions collectively were formed. Once a leader gets a new target, the path finding to the target is performed which means generating the

3 Implementation

child nodes and nodes to reach to the destination. The members of the group copy the route and nodes from the leader to reach the same target.

There are a few scenarios where groups get separated and they should reunite after the process. In the implementation two scenarios have been encountered, which are

- after the check-in and
- after the security check.

At check-in each member of the group creates a separate route in the queue which is called simple route. Once the check-in process is done, all agents move to the exit point that is defined after the check-in. At this point, all the members of one group wait for each other. Once all the members of the group gather at the exit point the new target gets set for the leader and again the followers get the same route as the leader. After the check-in exit point, the leader gets the target e.g. the FIDS or the nearest queue of the security. After the security, the other scenario takes place and in this work an additional exit point has been implemented where all the members of the group gather and then proceed e.g. to the gate in the air side. Fig 3.2 illustrates groups of agents at different parts of the airport.

3 Implementation

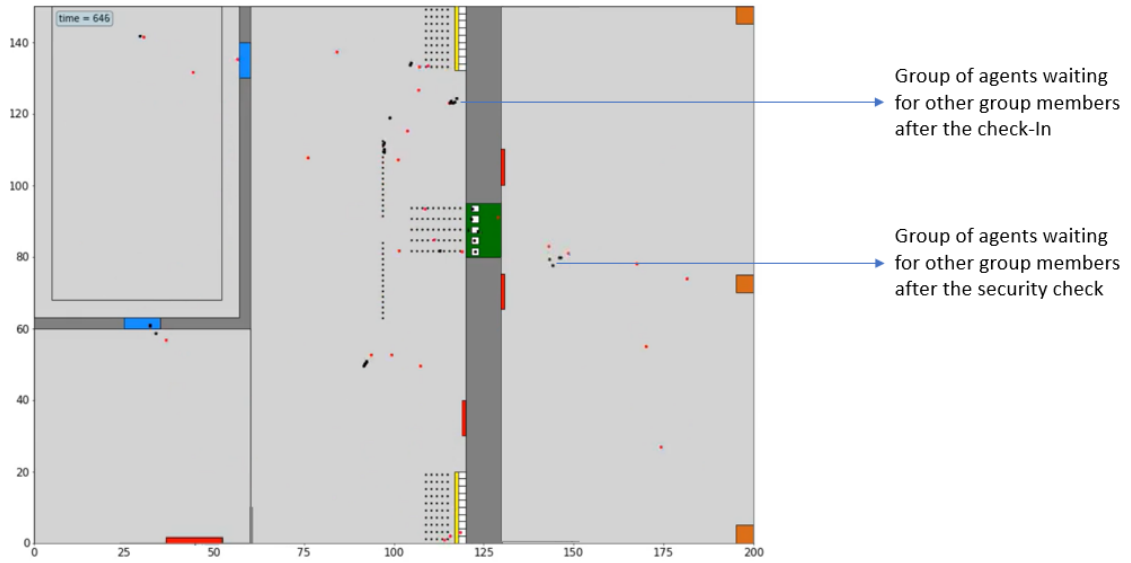


Figure 3.2: Simulation of passenger movement in the airport with groups of agents presented at a specific time step. Leaders are visualized as red and followers as black dots.

In Fig 3.3, the movement of one example group of agents is tracked in the airport layout. As we can see from Fig 3.3, the black dotted lines represent the movement of group members and the red dotted line represents the movement of a leader. The group started from the spawn area and chooses the nearest door. When the group crosses the doors and enters the landside area we can see that they stick together till the check-in area. Once the group member are done with the check-in, they wait at the exit point for all the group member. Once all the group members have arrived, they proceed towards the security queues. At the security, the group members are distributed in different security queues and process through the different security counters. Once the security is done, again the agents in the group wait at the exit point. Once all agents gather at the exit point, they walk together towards the gate and finally de-spawn (deleting from the simulation).

3 Implementation

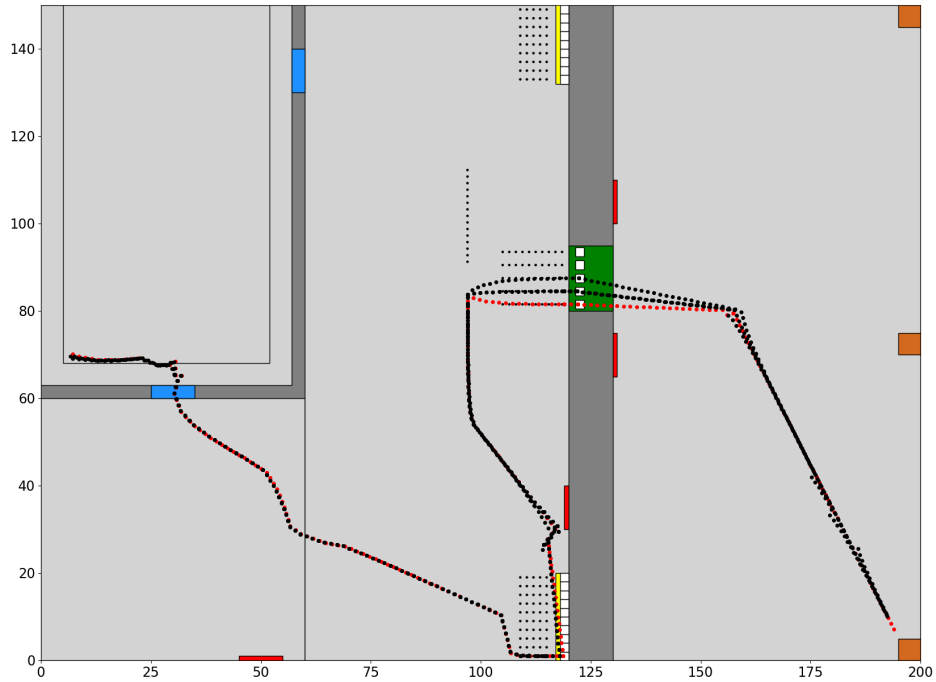


Figure 3.3: Simulation of passengers movement in the airport for an example group of agents. The leader is presented by a red and the followers with a black dotted line.

Listing 3.1 shows the code snippet written to perform the waiting of group members for each other after the security check. Line number 1 in the code snippet is the lambda expression at the exit point of the security. Once the agent reaches the security exit point, it triggers this lambda expression. The exit point captures the airport object, `tempExitSecurity` and the route object. The lambda expression accepts the agent object as parameter. Once the agent reaches the exit point, the side attribute of an agent is changed to airside area described in line number 2. In line number 4, the size of the group is received with the help of an agent object.

Listing 3.2 shows the data structure of the groups. Multimaps has been used to store the groups of agents. As the multimaps is a key-value pair data structure, the leader's unique ID has been added as the key and the follower agent as the value.

3 Implementation

A group can have multiple agents, so all the follower agents are inserted with the same key i.e. the respective leader ID. Line number 1 of the Listing 3.2 gives the declaration of the mapping of followers with the respective leader. In line number 2, the key and corresponding values are inserted in the multimap.

Following this approach, the number of agents belonging to the same group can be extracted which refers to line number 4 of Listing 3.1. In line number 5, a condition is implemented to check if an agent is an individual. If the latter case is true, i.e. the group has no followers then the `newTarget` function in line number 6 is called directly. The `newTarget` function sets the new object of a particular airport side as the next target. For example in the airside area, an agent may either go to the FIDS or the gates. In line number 7, the `isInQueue` attribute of an agent is set to false. In line number 9, the simulation iterates over the vector of leaders and again checks the size of the group (see line number 10).

In line number 11, the simulation iterates over all the followers in the multimap and checks if the leader ID matches with the key of the multimap. If the latter is true, it is checked if all the group members including the leader have reached the target by calling the `isTargetReached` method. This method accepts two parameters. First, it takes the position of the leader and compares it with the target. In the same way it is checked for each follower if their targets have been reached. If both, leader and followers, have reached the exit point, the local variable `agentsAtSecurityExit` is incremented by one. Second in line number 19, it is checked if the size of the group equals the local variable `agentsAtSecurityExit`. If the latter is true, the leader's airport side attribute is assigned to airside and the new target function is called. As soon as a leader gets a new target, the rest of the follower agents of that particular group will follow the path of the leader.

```
1  exitSecurity.setCallback([airport, tempExitSecurity, route](Agent
   * agent) {
2    agent->setAirside(AirportObject::AirportSide::AIRSIDE);
3
```

3 Implementation

```
4     int sizeofGroup = airport->getLeaderWithFollowerMap()->count(
agent->getUniqueId());
5     if (agent->isLeader() && sizeofGroup == 0) {
6         agent->newTarget();
7         agent->isInQueue(false);
8     }
9     for (std::vector<Agent*>::iterator itLeader = airport->
getLeaderAgents()->begin(); itLeader != airport->getLeaderAgents
()->end(); itLeader++) {
10         sizeofGroup = airport->getLeaderWithFollowerMap()->count((*
itLeader)->getUniqueId());
11         for (std::multimap<int, Agent*>::iterator itrMap = airport->
getLeaderWithFollowerMap()->begin(); itrMap != airport->
getLeaderWithFollowerMap()->end(); itrMap++) {
12             (*itrMap).second->setAirside(AirportObject::AirportSide::
AIRSIDE);
13             if ((*itLeader)->getUniqueId() == itrMap->first) {
14                 if ((route->isTargetReached(itrMap->second->getPosition()
, new Target(tempExitSecurity->getArea()->getShape(), 10)))
&& (route->isTargetReached((*itLeader)->getPosition(),
new Target(tempExitSecurity->getArea()->getShape(), 10))))
15                     {
16                         {
17
18                             agentsAtSecurityExit++;
19
20                         }
21                         if (sizeofGroup == agentsAtSecurityExit) {
22                             (*itLeader)->setAirside(AirportObject::AirportSide::
AIRSIDE);
23
24                             (*itLeader)->newTarget();
25                             agentsAtSecurityExit = 0;
26                         }
27                     }
28                 }
```

3 Implementation

```
29     agentsAtSecurityExit = 0;
30 }
31 });
```

Listing 3.1: C++ code snippet at the exit point.

```
1 std::multimap<int, Agent*> leaderWithFollowerMap;
2 airport->getLeaderWithFollowerMap()->insert(std::pair<int, Agent*>(
    leaderAgent->getUniqueId(), groupAgent));
```

Listing 3.2: Data structure of groups in C++.

In section 3.1.1, the creation of groups is described and in Listing 3.3 the corresponding implementation is presented. In line number 1, the group constructor accepts the airport object, airport side, spawn area and size of the group as parameters. In line number 7 and 8, the x- and y-position of the leader are generated using the spawn area dimensions and a random number generator. Once the position of the leader is created, the corresponding new agent object gets created in line number 10. The agent constructor accepts the airport object, x- and y-position of the agent, direction of an agent (initially 0,0), ticket object, boolean value to check if an agent is a leader and group ID. Once the agent gets created, it is pushed into a vector of agents. If the size of the group is more than one, the agents is added it to a specific vector that contains only leaders.

```
1 Group::Group(Airport* airport, AirportObject::AirportSide
    airportSide, Shapes::Rectangle spawnArea, int size) {
2     this->shape = spawnArea;
3     this->groupID = latestGroupID;
4     this->size = size;
5
6     //Spawn leaders
7     double leaderX = shape.getPointMin().getX() + shape.getWidth() *
```

3 Implementation

```
    Auxillary::genDouble();
8  double leaderY = shape.getPointMin().getY() + shape.getHeight() *
    Auxillary::genDouble();
9
10 Agent* leaderAgent = new Agent(airport, Point(leaderX, leaderY),
    Direction(0, 0), airport->generateTicket(), true, latestGroupID)
    ;
11 airport->getAgents()->push_back(leaderAgent);
12 if (size > 1) {           // only add leader if they have a group.
13     airport->getLeaderAgents()->push_back(leaderAgent);
14 }
15 leaderAgent->newTarget();
```

Listing 3.3: C++ code snippet that represents the creation of a leader.

The initial positions of follower relative to the leader position are given in Table 3.1. The Listing 3.4 describes the implementation of the creation of followers of the group. While creating the group, the size of the group is passed to the function that creates the position of the followers relative to the leader. In line number 6 and 7, a local variable is created which stores the value relative to the position of the leader. In the same way, the positions for all the members of the group are created. Once the position of a follower is created, the agent object constructor is called with the leader boolean value as false because to generate the follower agents.

```
1  for (int i = 0; i < size -1; i++) {
2      double tempX = 0;
3      double tempY = 0;
4
5      if (i == 0) {
6          tempX = leaderX - 0.5;
7          tempY = leaderY - 0.5;
8      }else if (i == 1) {
9          tempX = leaderX + 0.5;
10         tempY = leaderY - 0.5;
```

```

11     }else if (i == 2) {
12         tempX = leaderX + 0;
13         tempY = leaderY - 1;
14     }else if (i == 3) {
15         tempX = leaderX - 0.5;
16         tempY = leaderY - 1.5;
17     }else if (i == 4) {
18         tempX = leaderX + 0.5;
19         tempY = leaderY - 1.5;
20     }else if (i == 5) {
21         tempX = leaderX + 0;
22         tempY = leaderY - 2;
23     }else if (i == 6) {
24         tempX = leaderX + 1;
25         tempY = leaderY - 1;
26     }else if (i == 7) {
27         tempX = leaderX - 1;
28         tempY = leaderY - 1;
29     }
30 }

```

Listing 3.4: C++ code snippet that represents the creation of followers.

3.2 Global variables

During the implementation of group dynamics, variables needed to be modified and added. However, in the existing code, most of the variables which can be customized or can be used as a configuration parameter were hard coded. To make the code more flexible, a class which contains all the global variables has been introduced which can be customized before the start of the simulation. The variables listed below can be customized:

- **Time step** This variable defines the length of one time step. By default the

3 Implementation

time step is 1 second.

- **Spawn rate** To run the simulation, agents need to be spawned at particular time intervals. By default the spawn rate value is kept at 10 which means an agent spawns every 10 time steps.
- **Airport width & airport height** The airport height and width allows the user to configure the airport infrastructure. In the code by default the airport height and width is set to 150 and 200 meter respectively.
- **Check-in & security counter processing speed** The processing speed of the check-in and security counters can be customized by the user. To make the simulation process fast these parameters can be changed. By default value is kept is 5 and 10 time steps respectively.
- **Time steps to run the simulation** The user can specify the amount of time to run the simulation. By default the simulation runs for 2000 time steps in the code.
- **Space between agents in groups** The space between agents in groups can be customized. This variable enables the agents to maintain a certain distance within the group. By default the value is kept at 0.25 meter.
- **Space between individual agents** The space between individual agents can also be customized which also defines the space agents of different groups have to keep to each other. By default the value is kept at 1.5 meter.
- **Queue item distance** The queues at check-in and security counters have queue item positions which enable the agents to stand at specified distance. By default in the code the value is kept at 1.5 meter.

3.3 Distance Rules

3.3.1 Introduction

In Chapter 2 and 3, the agent-based simulation and the implementation of group dynamics was introduced. Further, distance variables were described between agents and obstacles, between agents in a group and between agents of different groups. These already in place distance rules and the current pandemic situation motivated the remainder of this work.

During the COV pandemic, physical distancing is an important part of measures to control the spread but exactly how much distance is required is unclear. COV virus proliferates when it gets into the body through the eyes, nose or mouth [29]. The virus usually spreads through surfaces or more commonly through the air where it is released by coughs and sneezes. Thus the physical distancing comes into picture to avoid the contagious from spreading. According to research done by Well [20], the origin of distance rules began in the 19th century where initially scientists were collecting samples of visible droplets containing pathogens on glass and proposed a 1-2 meter distance as safe. Further, the 1-2 meter rule is based on the distinction of respiratory droplets into two sizes, large and small. The emitted large droplets can travel in the air more quickly than they evaporate and land within a 1-2 meter range whereas small droplets can evaporate more quickly than they fall.

According to health officials from the UK, 2 meter distance from each other is recommended. Countries like Canada and Spain opted for the same distance rule as the UK. But different countries have their own distance rules. In the US, the common distance rule is 1.8 meter whereas countries like Germany, Italy, Greece, Australia, Netherlands a distance rule of 1.5 meter is implemented. South Korea has a distance rule of 1.4 meter. In some countries like France, Denmark, China, Hong Kong and Singapore distance is kept to 1 meter [29].

The Lancet study instructed by the World Health Organization observed that the spread of virus without any protection such as mask or face shield comes down

drastically with the distance. At the area of 1 meter, the risk is 12.8%, 2.6% at the distance of 1.5 meter, 1.3% at 2 meter[9].

3.3.2 Distance Rule Violations

An increased distance rule in terms of ABM means that the agents have a larger radius around themselves in which no other agent is allowed to enter. Thus, the chance for collisions between agents increases and also the chance that agents need to wait for some time steps to find a suitable path without collisions. The possibility for agents to violate the distance rule when they lose their patience has been introduced. A patience threshold attribute has been implemented as a uniform random number between 2 and 5. The patience threshold attribute of an agent is initialized at the time of the creation of an agent. Agents try to maintain the specified distance with the other agents. However, if there is no other way to avoid the collision than waiting, the agent's waiting counter attribute gets increased. If an agent has to wait more than one time step, the waiting counter increases and once it reaches the patience threshold value, the agent will violate the distance rule. In the code this is reflected by a small distance value which means that this agent does no longer respect the prescribed distance rule and simply squeezes through the other agents to reach the target.

The information about the agents which have violated the distance rule is written in an output CSV file generated by the C++ code. More information on the existing data sets can be found in section 2.2.3. Here, a boolean value to track if an agent has violated the distance rule has been added to the output data. In the data set which stores the information related to airport performance, a new field named **Distance violation count** has been introduced which stores the violation count as a function of time.

3.3.3 Implementation Of Different Distance Rules

In the implementation, five different rules have been introduced to observe the impact of distances in the airport, i.e. $d = (1, 1.25, 1.5, 1.75, 2)$ meter. These distances are used between the different individual agents and between different groups. The distances can be configured in the global variables. Once the parameter is set, all the agents maintain the distance everywhere in the airport i.e. in the queue and the rest of the airport area. Note that the distance between the agents in the same group is kept at 0.25 meter.

3.3.4 Plotting The Simulation Output

To analyze the performance of the airport like distance violation count, data sets need to be plotted. To achieve this, a python script has been written to generate the graphs based on standard libraries like matplotlib and numpy. Matplotlib is a library created for static, animated visualizations for python. It's an open-source library hosted in github[19]. Numpy is a python library that provides the multidimensional array object and basic statistical operations [18]. The C++ code generates the airport performance data set mentioned in Table 3.2. To take uncertainties in the simulations into account, 100 repeated data sets have been generated. For the data analysis, the 100 data sets have been stored in a list. In Listing 3.5, line from 19 to 22, it is presented how the distance rule violation count is extracted from the CSV file for 2000 time steps. There is a separate list that contains the time steps extracted from the same CSV file as the violation count (line number 14 to line number 17). Using both lists, the graph for distance violation count with respect to the time steps has been plotted.

```

1 countToReadTimeOnlyOnce = 0
2 counterForListIndex = 0
3 # iterating over all the files and storing the time steps in
4 # r_time list for once and distance violation count in

```

3 Implementation

Variable Name	t = 1	t = 2	-	t = 1999	t = 2000
agentDensityOutside	5	5		10	5
agentDensityLandside	0	0		0	5
agentDensityAirside	0	0		32	26
isEvacuating	0	0		0	0
agentsInSecurityQueue	0	0		15	10
distanceRuleViolationCount	0	0		20	22

Table 3.2: An example of the airport performance data set.

```
5 # different list.
6 for i in file_paths:
7     openFile = open(i, 'r')
8     next(openFile)
9     lines = openFile.readlines()
10    if len(lines) < 2000:
11        print('error opening the file.')
12    else:
13        if countToReadTimeOnlyOnce == 0:
14            for x in lines:
15                c = x.strip('\n') # remove line break at the end
16                a = int(c.split(';')[0])
17                r_time.append(a)
18                countToReadTimeOnlyOnce = countToReadTimeOnlyOnce + 1
19            # counter to read the time steps only once.
20            for y in lines:
21                d = y.strip('\n')
22                a = int(d.split(';')[6])
23                accumulatedList[counterForListIndex].append(a) #
24                # appending distance violation count to the list.
25                counterForListIndex = counterForListIndex + 1 #
26                # accumulated list contains the list of list,
27                # counter is used
```

```
25 # to refer the next inner list.
```

Listing 3.5: Python code snippet to extract information from csv all the files and generate the graphs.

To plot the mean and standard deviation graph from the data set, the distance rule count list has been converted to a numpy array because the numpy array provides the in-built function `mean()` and `std()` that calculate mean and standard deviation respectively. In the code, the sum and the difference of mean and standard deviation have been calculated and stored in a different array. Some negative values appeared when subtracting the standard deviation from the mean. To eliminate these values, as the graph should not go to negative values, a condition has been added that replaces them with zero.

3.3.5 Automate The Simulation Process

To reduce manual effort, a python script that triggers an .exe file generated by ABM C++ code has been written. In Listing 3.6, line number 6, the source of the .exe file has been passed with spawn rate and distance rule as parameters. The argument passed from the python script is captured as command-line arguments in the C++ code. In Listing 3.7, line number 3 and 4, the command-line arguments are used to assign them to the global variables.

```
1 import subprocess
2 # runs for the 100 times and triggers the .exe file
3 # with specified arguments, first argument is the number of
4 # timesteps and other is distance rule.
5 for i in range(1):
6     exe_command = "..\\source\\repos\\SATIEABM\\
7         AgentSimulationSATIE\\x64\\Debug" \
            "\\AgentSimulationSATIE.exe 2000 1.5 "
```

3 Implementation

```
8 subprocess.run(exe_command)
```

Listing 3.6: Python code snippet shows the implementation of triggering the exe file.

```
1 //capturing the command line arguments and assinging it to the  
   global variable.  
2  
3 GlobalVariables::totalTimeStepsToRunSimulation = std::stoi(argv  
   [1]);  
4 GlobalVariables::spaceBetweenIndividualAgents = std::stof(argv  
   [2]);
```

Listing 3.7: C++ code snippet shows the implementation of capturing the command line argument in the C++ code.

4 Results

Chapter 3, shows the implementation of group dynamics and different distance rules. This chapter describes the results obtained from the implementation of earlier mentioned features. The first section contains the result of group dynamics i.e. how different group sizes take time to process all the required functionalities. The second section describes the impact of different distance rules on airport performance. The last section contains a modification of layout and how the changes in the layout of the airport impact the distance rule violations.

4.1 Group Dynamics

The implementation of group dynamics has been motivated in Chapter 3 and presented in Section 3.1 of this work. In this Section, the results are visualized and simulations without groups are compared with simulations with different sizes of groups (S_{groups}). To this end, a measure to compare those different simulations has been introduced which is the time it takes a specific individual agent or group of agents to move from the outside area to the gate in the example airport layout presented in Section 3.1.3. To take uncertainties into account, 100 repeated simulations with $t = 2000$ time steps each have been performed with different group sizes. The duration that the individual agents or groups spent in the airport has been averaged over the number of individual agents or groups of agent which leads to 100 average duration values per case. The different cases are presented in Table 4.1. The spawn rate (R_{spawn}), the time interval between the spawning of agents, has been adjusted

4 Results

Case	1	2	3
Group size	1	4	8
Spawn rate	20	80	160
Number of groups formed	100	25	12
Average group duration	275	295	335

Table 4.1: Comparison between three simulation cases with different group sizes. The average group duration is a mean value over 100 simulations and all groups.

for the three cases to ensure a comparable total amount of agents in the airport during all simulations. The respective number of groups (N_{groups}) can be estimated using the following equations:

$$N_{groups} = \frac{t}{R_{spawn}} \quad \text{with} \quad R_{spawn} = 20 * (S_{groups}) \quad (4.1)$$

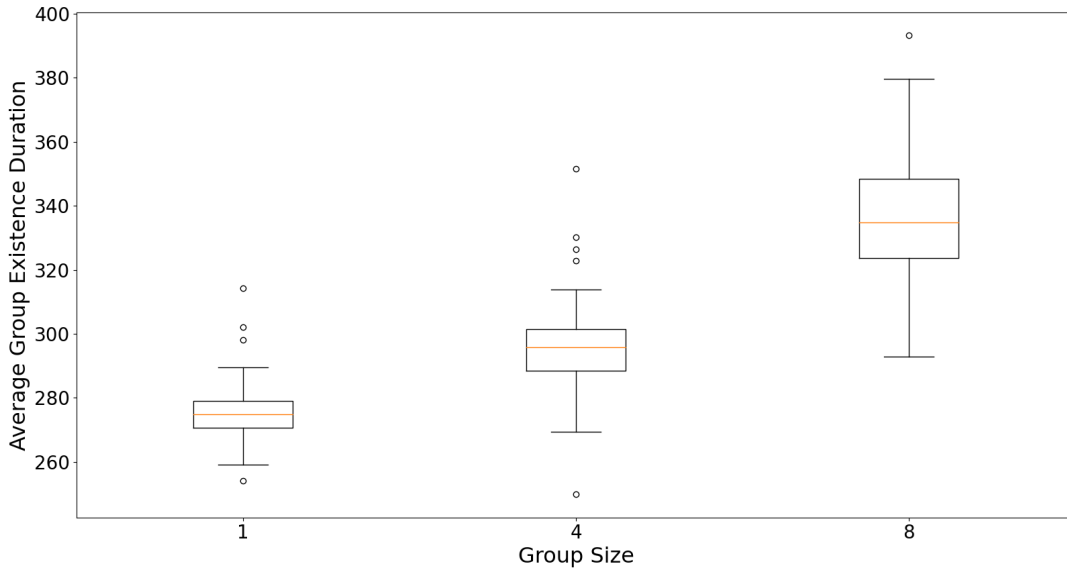


Figure 4.1: Box plots for each case of Table 4.1 and 100 simulations show the average duration that agents 'exist' in the airport. The black line above and below the box are called as whiskers it show the spread of data, yellow line in the box represents the median of the data, an outlier (black small circle) is defined as a data point that is located outside the whiskers of the box plot.

Fig. 4.1 represents the average duration over all groups with different sizes for 100

simulations in each case. The result gives us an idea that when the group size is smaller it takes the agents less time to process at the airport. For the group size 1, 100 individual agents were created over 2000 time steps and it takes the agents in average 275 time steps to move from the spawn area to the gates. For the group size 4, 25 groups were formed and it takes the agents in average 295 time steps to reach the gates. For the group size 8, 12 groups were created and it takes in average 335 time steps to reach the gates. These values can also be found in Table 4.1. The increase in duration happens because when larger groups process through the check-in counters and security check counters, each member waits for other members of the group to arrive at the exit point. Hence it causes larger groups to take more time in the airport simulation as compared to smaller groups.

4.2 Distance Rules

All the above mentioned simulations with different group sizes, are performed with the same distance of 0.25 meter between agents. In the following the group sizes are randomized at spawning between 1 and 8 but the distances between agents are varied between 1.5 and 2 meter. In all cases, the patience count of an agent is randomly generated between 3 and 5.

4 Results

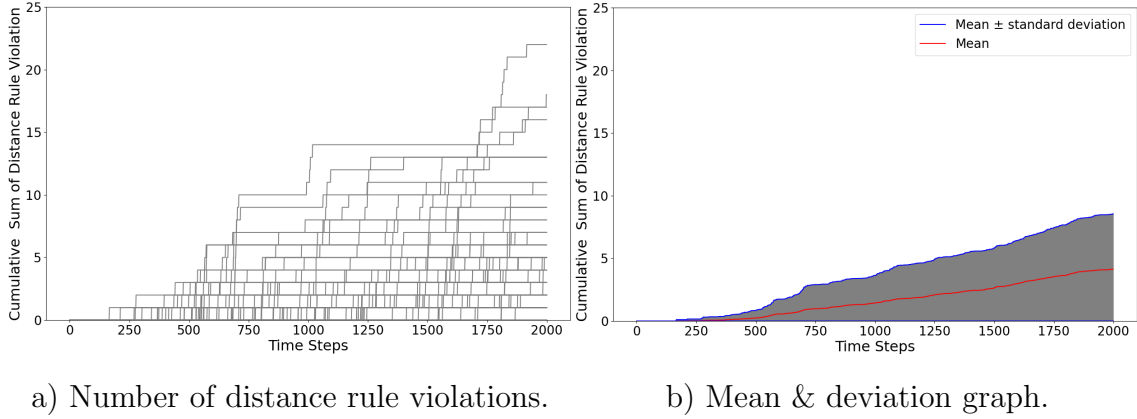


Figure 4.2: Results of 100 repeated simulations with a distance rule of 1 meter. Graphs shows the number of violations that happened over the 2000 time steps. On the y-axis, the number of violations can be observed for a particular time step. Right: The red line shows the mean and the blue lines across the border of the grey area is the \pm standard deviation.

100 simulations with 2000 time steps were performed to observe distance rule violations by all the agents in the 1-meter distance case for the airport layout presented in Fig 2.13. The data can be visualized in a graph (please see Fig 4.2).

In Fig 4.2 b, mean and standard deviation graph for the same data set have been plotted. The standard deviation measures the dispersion of the data set relative to its mean. The standard deviation helps to predict the performance trends.

In Fig 4.3 we can see the results for the 1.5 and 2 meter distance cases for 100 simulation runs. Comparing the graphs, we observe for 1.5 meter distance rule (Fig 4.3 a), that the violation count increased as compared to 1 meter distance (Fig 4.2 b). For the 1.5 meter case, the distance violation count is near 20 whereas for the 1 meter case the distance violation count is less than 10. This shows the trend that as the distance increases, the agents tend to violate the rules more frequently.

As the distance increases from 1.5 meter to 2 meter, the violation count increases subsequently. The mean and standard deviation graph is presented in Fig 4.3 b. In the 2 meter case, the violation count is around 50 which is more as compare to the other 2 cases i.e. 1 meter and 1.5 meter. As the distance increases, an agent tries

4 Results

to maintain the distance of 2 meter, there are chances where an agent is waiting for other agents to move so that they can avoid the collision but due to the large distance rule an agent is finding difficulty to get suitable route and they wait until they find a new route. This ultimately increases their patience value and they are forced to violate the distance rule.

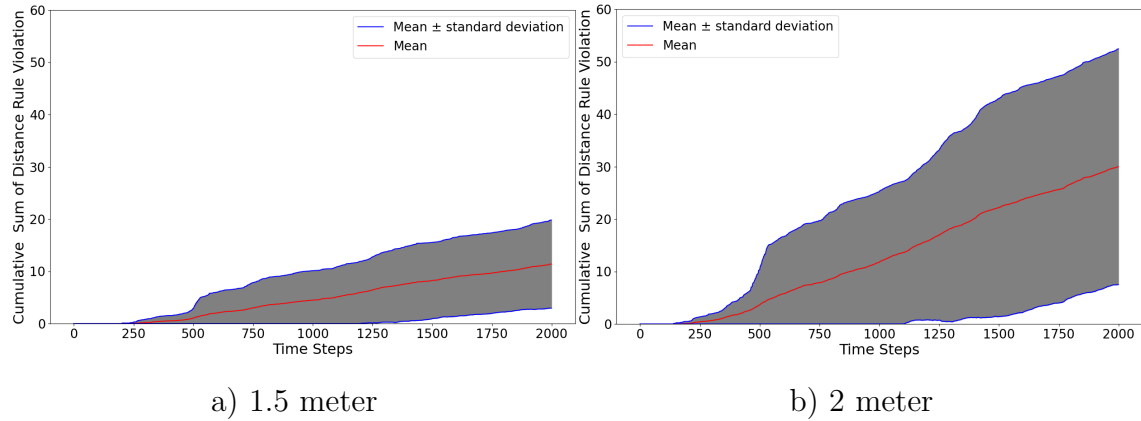


Figure 4.3: Mean & Deviation graphs of distance rule violations for 100 simulation runs. The red line shows the mean and the blue lines across the border of the grey area is the \pm standard deviation for the distance rule of 1.5 meter (left) and 2 meter (right).

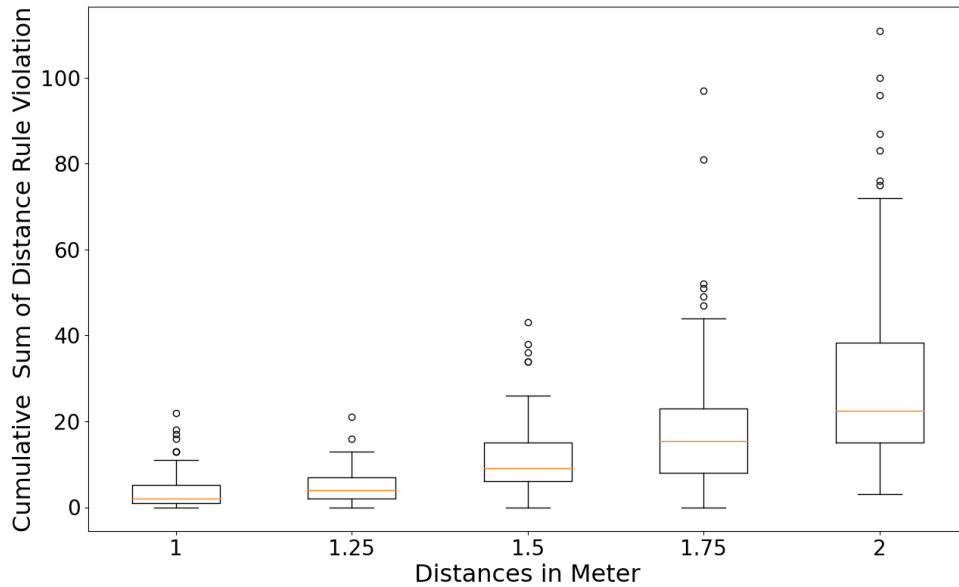


Figure 4.4: Comparison between the different distance rules. The black line above and below the box are called as whiskers it show the spread of data, yellow line in the box represents the median of the data, an outlier (black small circle) is defined as a data point that is located outside the whiskers of the box plot.

We can see the results from the box plot graph for different distance rules in Fig 4.4. To see the results more clearly, two more distance rules have been introduced, i.e. 1.25 meter and 1.75 meter. As the distance increases, the number of violations are also increasing. In each case, the space to be maintained by an agent is getting increased which causes more collisions. Note, from this results it can be concluded that the number of violation increase exponentially as a function of distance.

4.3 Layout modifications

To further test the abilities of the simulation code and to compare distance rule violations under varying situations, layout modifications have been introduced in the master thesis work. Fig 4.5 a), shows the simple layout without any complexity

4 Results

i.e. without any layout modification. In Fig 4.5 b), a layout with walls around the FIDS has been created. As the complexity increases, there are chances of more violations. From Fig 4.5 c) and Fig 4.5 d), we can observe the number of violations over 2000 time steps for the simple layout and the complex layout at 1.5 meter distance between the agents. The graph plotted for the simple layout has a smaller number of violations of around 20-25 whereas for the complex layout, the violation count goes up to 35 approximately.

If the airport layout complexity is increased, it also increases the violation count because the agents have less space to move around the airport and when there are more agents present in a small space at the same time, then there are more chances of getting collisions. To avoid collision, an agent waits and when the patience level increases, an agent may violate the rule.

4 Results

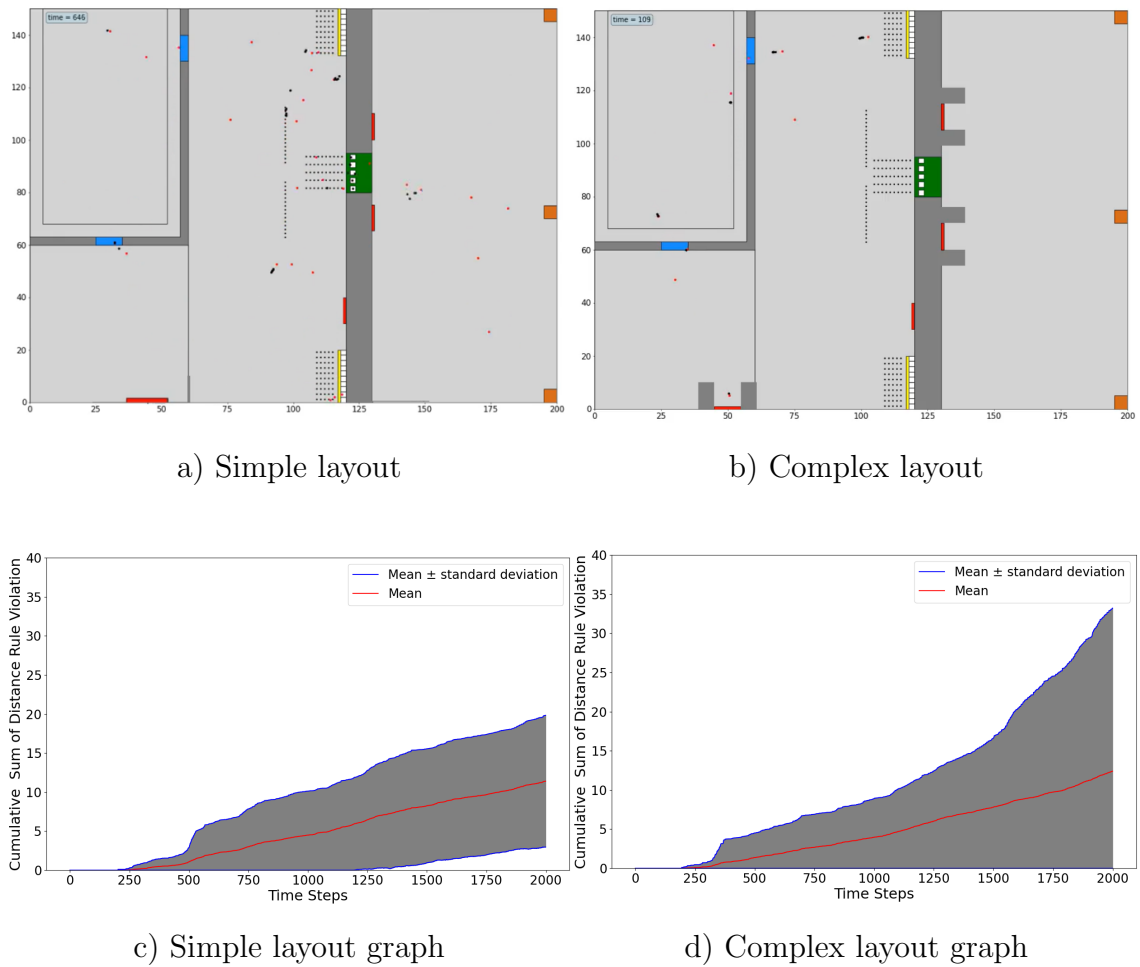


Figure 4.5: Comparison of distance rule violations between simple and complex layout at 1.5 meter distance.

Fig 4.5b), shows quite artificial layout modifications, i.e. it just has walls around the FIDS.

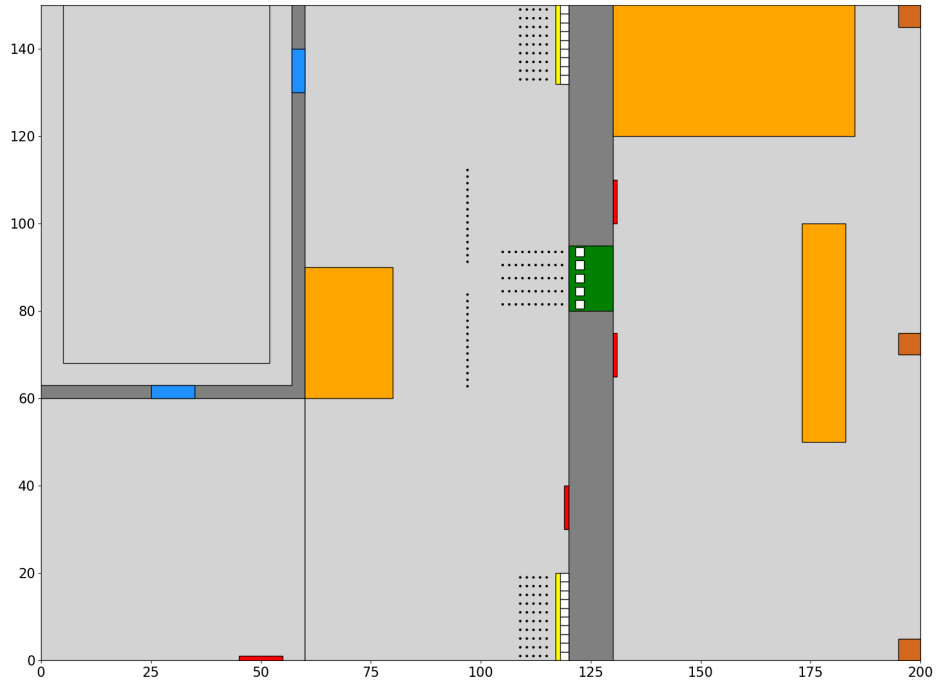


Figure 4.6: Complex layout with more obstacles (yellow rectangles) which is called 'shop layout'.

Fig 4.6 shows a more realistic version of a complex layout. Obstacles such as shops in landside area and airside area have been included in this work labelled in yellow color. By implementing these shops in the layout, curb the space for the agents to move. If there is less space to move, there are chances that the agents will break the distance rule more frequently. To observe the trend of distance rule violations between the simple layout (Fig 4.5 a), the FIDS layout (Fig 4.5 b) and the shop layout (Fig 4.6), box plots have been plotted for 100 repeated simulations of the three cases.

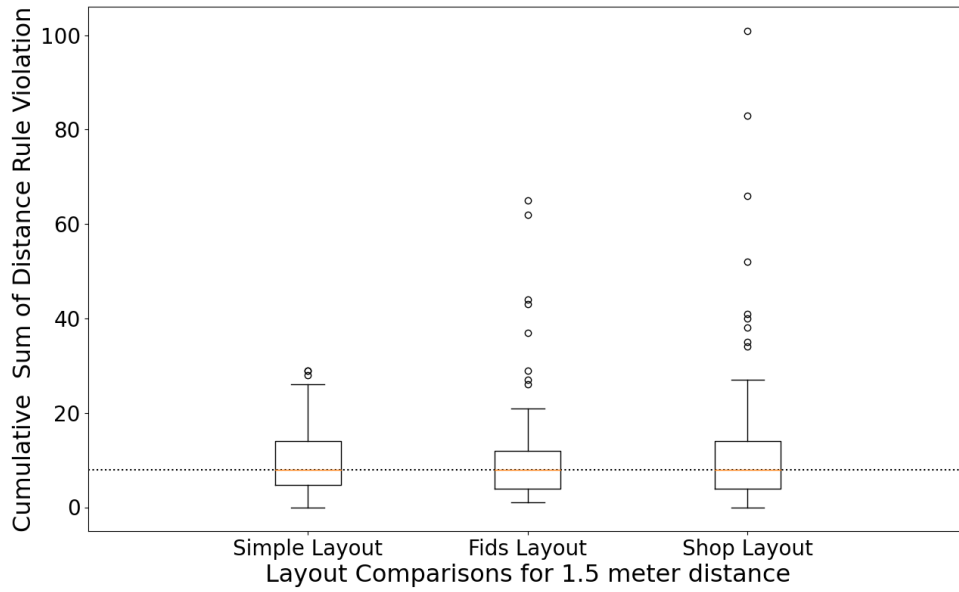
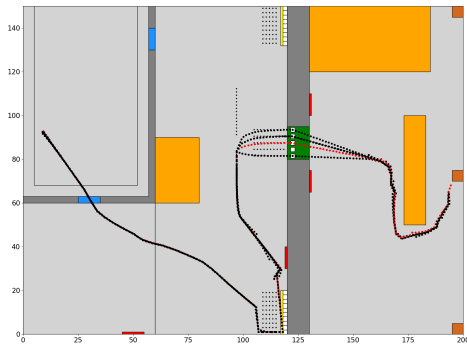
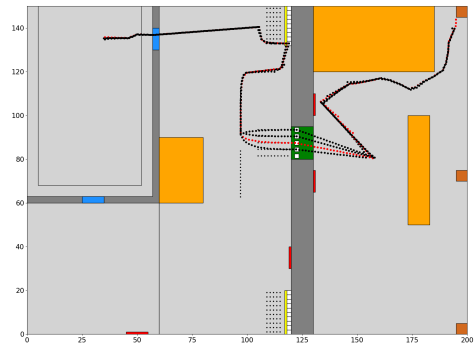


Figure 4.7: Layout comparison between simple, FIDS and shop layout. The median is given as black dotted line. The black line above and below the box are called as whiskers it show the spread of data, yellow line in the box represents the median of the data, an outlier (black small circle) is defined as a data point that is located outside the whiskers of the box plot.

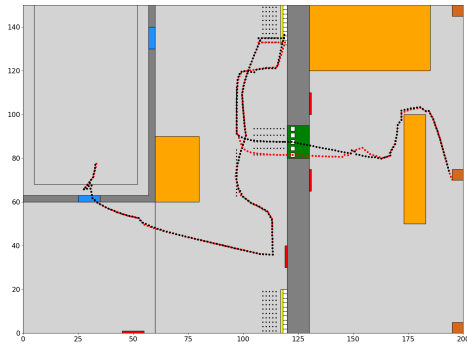
Fig 4.7 shows the comparison between the three cases. In all the three cases, the median value is the same. However, for the FIDS layout and the shop layout, there are some extreme values, which show more distance rule violations compared to the simple layout. These results suggest that the obstacles introduced in the layout do not impact very much on the overall distance rule violations and thus on the performance of the airport. But a trend can be derived that more complexity leads to more extreme cases where agents need to wait for each other.



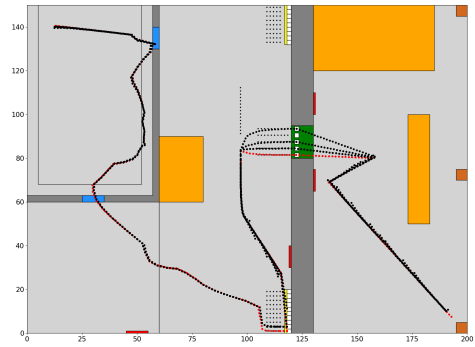
a) Simulation run for group 1



b) Simulation run for group 2



c) Simulation run for group 3



d) Simulation run for group 4

Figure 4.8: Different movement of groups of agents across the complex airport layout.

Finally, Fig 4.8 presents the movement of groups of agents across the complex layout (shop layout) of the airport. As we can observe from each of the four figure that groups of agents choose different paths to reach different gates. For the second gate (Fig 4.8 a) and (Fig 4.8 c), different groups of agents choose different paths to reach to the target. This comparison provides a summary of the main features that have been introduced into the existing code in the course of this master thesis work, i.e. groups of agents, exit points and layout modifications.

5 Conclusion

The following sections in this chapter provide the summary and achievements of this research. Also, it provides an overview of the next steps that can be taken by the team members who are expected to extend this simulation application.

5.1 Summary

The objective of this thesis research was to contribute to the ongoing project SATIE and especially to the ABM in IPS. In this thesis ABM was used to simulate groups of agents, their behaviour and how they stick together in each process at the airport, allows us to understand and predict the behaviour of groups of agents, how different group sizes impacts on the amount of time to complete the all process and formalities at the airport.

Apart from that, this thesis contributes to analyse and study different distance rules across the globe in the current pandemic situation. In a real life situation, many factors can be taken into account. In crowded situations, people may tends to violate this distance rule if the layout of an airport is complex. There are chances that people do not respect the prescribed distance rule. In this thesis, we have found that as the layout gets complex or the distance between the agents is increased then there are more chances of distance rule violations.

This thesis gives an overview and impact of different distance rules. Our proposed model can be customized, i.e. it can be used to analyse the different distance rules and the layout can be enhanced with more complexity to observe the impact in

performance of the airport under varying conditions.

5.2 Outlook

An important aspect, that needs to be addressed in the future development of the ABM presented in this work, is validation and verification. The existing code developed in the project SATIE has been partially verified by airport operators involved in the project. However, the existing code and the group behaviour implementation still need to be validated against data. In order to validate the simulations, the plan was to validate it with real data sets collected in the office infrastructure of Fraunhofer EMI with people moving around the office buildings. But due to COV pandemic we had to change the original plans. Some open source data sets have been reviewed in the course of this work but they did not offer the needed information about group dynamics in airports. In this master thesis work, a manual approach has been followed to analyse the results like by seeing the complete simulations for all time steps and by observing any exceptional cases. The latter could be e.g. an agent being stuck at some place of the airport like e.g. in a queue. But due to the randomness of ABM, it is difficult to analyse the actual root cause, i.e. in each simulation run the behaviour of an agent is different and it is difficult to exactly repeat the simulation run. Hence, the simulation needs to be validated with actual data sets. Still, the COV pandemic motivated the studying of distance rules and the implementation of respective performance measures in ABM to quantify the impact. Also the impacts of distance rules need to be validated against real-world data in the future once they are available. Apart from this, the formation of groups can be improved i.e. the groups of agents can be dynamically formed. If the groups of agents are moving in a certain direction then other agents can adapt and join the groups of agents, this approach is good when agents are evacuating the building where individual agents join the crowd and forms a big group of agents while evacuating [5].

Bibliography

- [1] Allan, R.J., et al.: Survey of agent based modelling and simulation tools. Science & Technology Facilities Council New York (2010)
- [2] Athanasiadis, I.N., Mitkas, P.A.: An agent-based intelligent environmental monitoring system. *Management of Environmental Quality: An International Journal* (2004)
- [3] Augustijn, E.W.: Components of an agent based model, <https://www.futurelearn.com/info/courses/geohealth/0/steps/19290>
- [4] Bai, S., Raskob, W., Müller, T.: Agent based model. *Radioprotection* 55 (05 2020)
- [5] Balboa, A., Cuesta, A., Alvear, D.: Measuring social influence and group formation during evacuation process. *Collective Dynamics* 5, 238–245 (2020), <https://collective-dynamics.eu/index.php/cod/article/view/A56>
- [6] Belsare, A.V., Gompper, M.E., Keller, B., Sumners, J., Hansen, L., Millsbaugh, J.J.: An agent-based framework for improving wildlife disease surveillance: A case study of chronic wasting disease in missouri white-tailed deer. *Ecological modelling* 417, 108919 (2020)
- [7] Bonabeau, E.: Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the national academy of sciences* 99(suppl 3), 7280–7287 (2002)

BIBLIOGRAPHY

- [8] Castiglione, F.: Agent based modeling. Scholarpedia 1(10), 1562 (2006), revision #123888
- [9] Chu, D.K., Akl, E.A., Duda, S., Solo, K., Yaacoub, S., Schünemann, H.J., Elharakeh, A., Bognanni, A., Lotfi, T., Loeb, M., et al.: Physical distancing, face masks, and eye protection to prevent person-to-person transmission of sars-cov-2 and covid-19: a systematic review and meta-analysis. *The Lancet* 395(10242), 1973–1987 (2020)
- [10] Chunlin He, He Xiao, Wen Dong, Liping Deng: Dynamic group behavior for real-time multi-agent crowd simulation. In: 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE). vol. 1, pp. 544–546 (2010)
- [11] Comission, E.: Security of air transport infrastructure of europe (2021), <https://cordis.europa.eu/project/id/832969>
- [12] Crooks, A., Castle, C., Batty, M.: Key challenges in agent-based modelling for geo-spatial simulation. *Computers, Environment and Urban Systems* 32(6), 417–430 (2008)
- [13] Crooks, A., Heppenstall, A.: Introduction to Agent-Based Modelling, pp. 85–105 (01 2012)
- [14] Crooks, A., Heppenstall, A., Malleson, N.: Agent-Based Modeling (12 2017)
- [15] Evans, D.A.: Agent based modelling: Introduction, <http://www.geog.leeds.ac.uk/courses/other/crime/abm/general-modelling/index.html>
- [16] Gigliotta, O., Miglino, O., Parisi, D.: Groups of agents with a leader. *Journal of Artificial Societies and Social Simulation* 10(4), 1 (2007)
- [17] Gilbert, N.: Agent-based models (2008), <https://www.doi.org/10.4135/9781412983259>

BIBLIOGRAPHY

- [18] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., G'érard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. *Nature* 585(7825), 357–362 (Sep 2020), <https://doi.org/10.1038/s41586-020-2649-2>
- [19] Hunter, J.D.: Matplotlib: A 2d graphics environment. *Computing in Science & Engineering* 9(3), 90–95 (2007)
- [20] Jones, N.R., Qureshi, Z.U., Temple, R.J., Larwood, J.P., Greenhalgh, T., Bourouiba, L.: Two metres or one: what is the evidence for physical distancing in covid-19? *bmj* 370 (2020)
- [21] Kasereka, S., Kasoro, N., Kyamakya, K., Goufo, E.F.D., Chokki, A.P., Yengo, M.V.: Agent-based modelling and simulation for evacuation of people from a building in case of fire. *Procedia Computer Science* 130, 10–17 (2018)
- [22] Kravari, K., Bassiliades, N.: A survey of agent platforms. *Journal of Artificial Societies and Social Simulation* 18(1), 11 (2015)
- [23] Lake, M.W.: Trends in archaeological simulation. *Journal of Archaeological Method and Theory* 21(2), 258–287 (2014)
- [24] Mackin, K.J., Tazaki, E.: Evolving intelligent multiagent systems using unsupervised agent communication and behavior training. In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions'* (cat. no.0. vol. 4, pp. 2411–2414 vol.4 (2000)
- [25] Naiem, A., Reda, M., El-Beltagy, M., El-Khodary, I.: An agent based approach for modeling traffic flow. pp. 1 – 6 (04 2010)

BIBLIOGRAPHY

- [26] Ozik, J., Collier, N., Heiland, R., An, G., Macklin, P.: Learning-accelerated discovery of immune-tumour interactions. *Molecular systems design & engineering* 4(4), 747–760 (2019)
- [27] Pluchino, S., Tribulato, C., Caverzan, A., Quillan, A., Cimellaro, G., Mahin, S.: Agent-based model for pedestrians’ evacuation after a blast integrated with a human behavior model (04 2015)
- [28] Salgado, M., Gilbert, N.: Agent Based Modelling, pp. 247–265 (11 2013), https://www.researchgate.net/publication/259335210_Agent_Based_Modelling
- [29] Sample, I.: What is the science behind the UK’s coronavirus distancing rules? (2020), <https://www.theguardian.com/world/2020/jun/10/science-behind-coronavirus-distancing-rules-2-metre>
- [30] Schmidt, B.: The modelling of human behaviour: The PECS reference models. SCS-Europe BVBA Delft (2000)
- [31] Stroeve, S., Bosse, T., Blom, H., Sharpanskykh, A., Everdij, M.: Agent-based modelling for analysis of resilience in atm (01 2013)
- [32] Urban, C., Schmidt, B.: Pecs - agent-based modelling of human behaviour (01 2001)
- [33] Wang, L., Ahn, K., Kim, C., Ha, C.: Agent-based models in financial market studies. *Journal of Physics: Conference Series* 1039, 012022 (jun 2018), <https://doi.org/10.1088/1742-6596/1039/1/012022>
- [34] ZARBOUTIS, N., MARMARAS, N.: Design of formative evacuation plans using agent-based simulation. *Safety Science* 45(9), 920–940 (2007)
- [35] Čertický, M., Drchal, J., Cuchý, M., Jakob, M.: Fully agent-based simulation model of multimodal mobility in european cities. In: 2015 International

BIBLIOGRAPHY

Conference on Models and Technologies for Intelligent Transportation Systems
(MT-ITS). pp. 229–236 (2015)