Universitat Autònoma de Barcelona

TAXIWAY MANAGEMENT THROUGH MULTI AGENT SYSTEM

Memòria del Treball Fi de Grau

en

Gestió Aeronàutica

realitzat per

Carolina Valderrama Rafegas

i didirigit per

Miquel Àngel Piera Eroles

Escola d'Enginyeria Sabadell, *Juliol* de *2013*

El sotasignat, ***Miquel Àngel Piera Eroles***,
professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memoria
ha estat realitzat sota la seva direcció per

Carolina Valderrama Rafegas.

I per a que consti firma la present.

Sabadell, ***Juliol*** de ***2013***

Firmat:

Signat: ***Dr. Miquel Àngel Piera Eroles***

FULL DE RESUM – TREBALL FI DE GRAU DE L'ESCOLA D'ENGINYERIA

Títol del Projecte: TAXI WAY MANAGEMENT THROUGH MULTI AGENT SYSTEM	
Autor[a]: CAROLINA VALDERRAMA RAFEGAS	**Data:** ***JULIOL 2013***
Tutor[a]/s[es]: Dr. Miquel Àngel Piera Eroles	
Titulació: GRAU EN GESTIÓ AERONÀUTICA	

Paraules clau (mínim 3)

- Castellà: sistema multi agente, proceso de escala, calles de rodaje, puertas de embarque, pista de aterrizaje, aeropuerto, gestión del tráfico aéreo, simulación, agente, NetLogo, sistema de apoyo en la toma de decisiones, rodaje autónomo, técnicas orientadas a agentes, área de terminal, superficie del aeropuerto.

- Anglès: muli-agent system, turnaround, taxiway, gate, runway, airport, air traffic management, simulation, ground controllers, agent, NetLogo, decision support system, autonomous taxiing, agent-oriented techniques, terminal-area, airport surface.

Resum del Projecte (extensió máxima 100 paraules)

- Català:
  Aquest projecte estudiarà la introducció de Sistemes Multi-Agent (MAS) entre aeronaus per millorar la fluïdesa de les operacions en terra. L'objectiu del projecte és analitzar l'eficàcia de MAS en la gestió dels carrers de rodatge a causa del important paper que representen els carrers de rodatge, com a connectors de les diferents fases d'un procés d'escala. L'objectiu principal és analitzar l'eficàcia de MAS millorant la fluïdesa de les operacions en terra.

- Castellà:
  Este proyecto estudiará la introducción de Sistemas Multi-Agente (MAS) entre aeronaves para mejorar la fluidez de las operaciones en tierra. El objetivo del proyecto es analizar la eficacia de MAS en la gestión de las calles de rodaje debido al importante papel que representan las calles de rodaje como conector de las diferentes fases de un proceso de escala. El objetivo principal es analizar la eficacia de MAS mejorando la fluidez de las operaciones en tierra.

- Anglès:
  This project will explore the introduction of Multi-Agent Systems (MAS) between aircrafts to improve the fluidity of ground operations. The project's purpose is to analyse the effectiveness of MAS in taxiways management due to the significant paper which taxiways represent as a connector of the different phases of a turnaround. The main objective is to analyse the effectiveness of MAS enhancing the fluidity in ground operations.

Dedicated to my family and friends,

INDEX

Abstract

The expected rate of growth in air traffic demand is imbalanced with the rate of growth in capacity of airport infrastructures. Ground operations in terminal areas are highly constrained due to the limitations of accommodating safe and efficient traffic flow under pre-defined terminal trajectories. The gradual growth of air traffic operations creates a challenge when balancing the capacity of ground infrastructures with the demand of air traffic (arrivals and departures) in presence of limited ground configurations including runways, taxiways and gates.

Terminal areas must be considered a key part of air traffic management systems to enhance the overall traffic flow. An airport represents a connecting process between two consecutive flights. The preparation process of an inbound aircraft for a following outbound flight that is scheduled for the same aircraft is known as a turnaround. Therefore the efficiency of ground operations in airport management involves the proper execution of turnarounds which have a direct impact on delays and furthermore on the efficiency of the overall air traffic flow.

As a result, airports are expected to be the bottleneck of the air traffic process so it is necessary to increase their capacity. Airport capacity can grow mainly in two different ways. First of all, the capacity of ground operations can be increased by developing plans for airport expansions such as building new terminals or runways, which is a long and especially expensive process. The challenges of this option are making more evident the need to make more of the available capacity. So the other possibility is to improve the efficiency of the existing terminals through the gradual introduction of advanced air traffic management (ATM) procedures to enhance the fluidity of ground operations.

This thesis will explore the introduction of Multi-Agent Systems (MAS) between aircrafts to improve the fluidity of ground operations. The main purpose is to analyse the effectiveness of MAS in taxiways management due to the

significant paper which taxiways represent as a connector of the different phases of a turnaround.

The main objective of this thesis is to analyse the effectiveness of MAS enhancing the fluidity in ground operations. MAS will allow real-time communication between aircrafts to determinate which is the best trajectory according to a set of initial conditions predefined by the airline company. This system will provide paths in 3D, where the thitrd dimension is the time in which these actions will be executed, taking into consideration the pre-defined flight schedule. Furthermore, it will reduce the workload of air traffic controllers contributing to the improvement in the overall air traffic management.

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Background

The European air traffic network contains about 117,000 links between airports and a foundation layer of 2,000 airports. The top 35 of the largest airports generates 50% of all flights. The total number of flights in Europe in 2011 was 9.8 million(segmented as seen in Figure 1), which represented an increase of 3.1% compared to 2010. (*EUROCONTROL, 2012*)

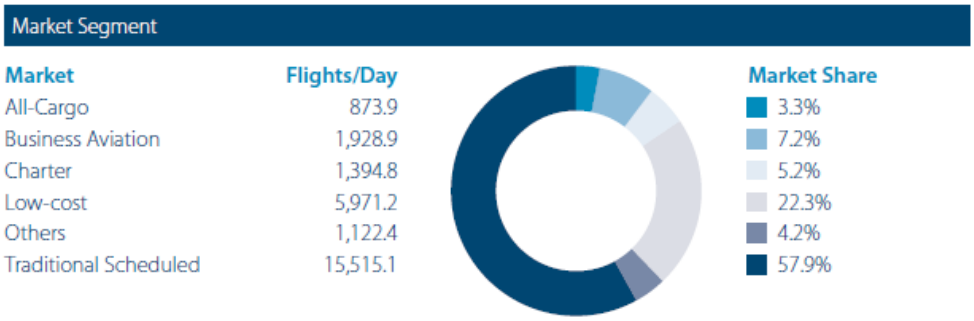

Figure 1: Number of flights in Europe in 2011

In more detail, from 2009 to 2011, as it can be seen in Figure 2, there has been a steadily growth of all flights during these 3 years. However, this growth has levelled off significantly during 2011 in comparison to the end of 2009. (*EUROCONTROL, 2012*)

Figure 2: Flight growth compared to previous year

According to CODA (Central Office for Delay Analysis) publications, by 2012 airline delays fell slightly showing an average delay per delayed flight of 27 minutes, this was a decrease around 1% in comparison to the delays observed in 2011.

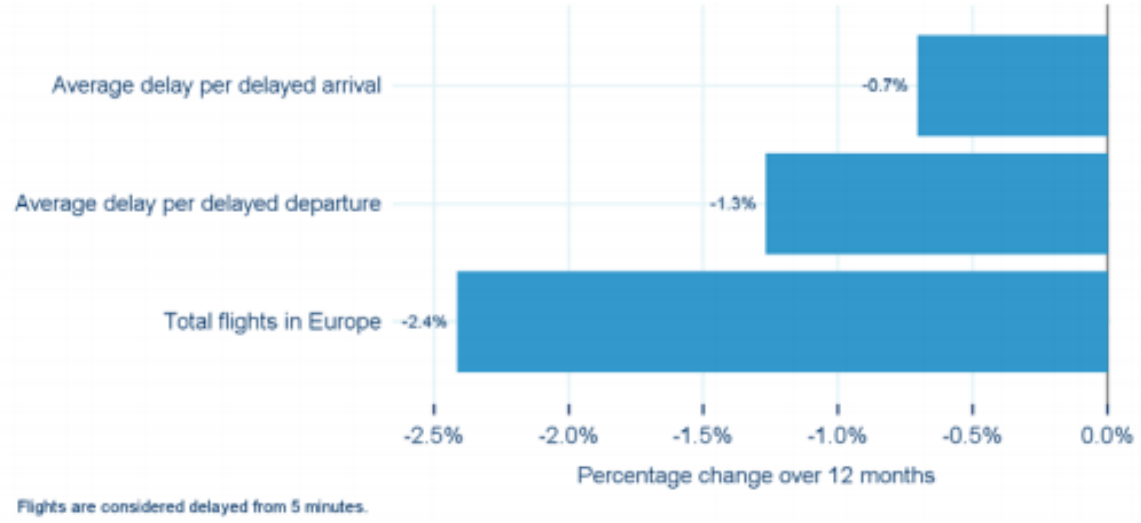

Figure 3: Delays comparing 2012 with 2011

Even though the statistics show that the average delay of flights has decreased in the last year, this average remains high. Moreover, it is well known that the air transport demand has been experiencing an enormous growth and it will continue to grow for the years. This trend demonstrates that current air traffic management (ATM) systems come dangerously close to the limit and are starting to get over-passed.

EUROCONTROL defines ATM as a system about the process, procedures and resources which come into play to make sure that aircraft are safely guided in the skies and on the ground.

The current ATM system in Europe is highly fragmented according to the different phases of a flight trip. Thus, different air traffic controllers are responsible at different stages; namely, approach controllers are responsible for inbound traffic, departure controllers for outbound traffic and ground controllers for ground movements in the terminal area. (*Weinjing Zao, 2012*)

As is seen in Figure 4, each flight goes through the following phases to be completed: push back from gate, taxiing into the runway, taking off, following a departure route, cruising along an airway, following an arrival route, landing, taxiing off the runway and stopping at the gate.

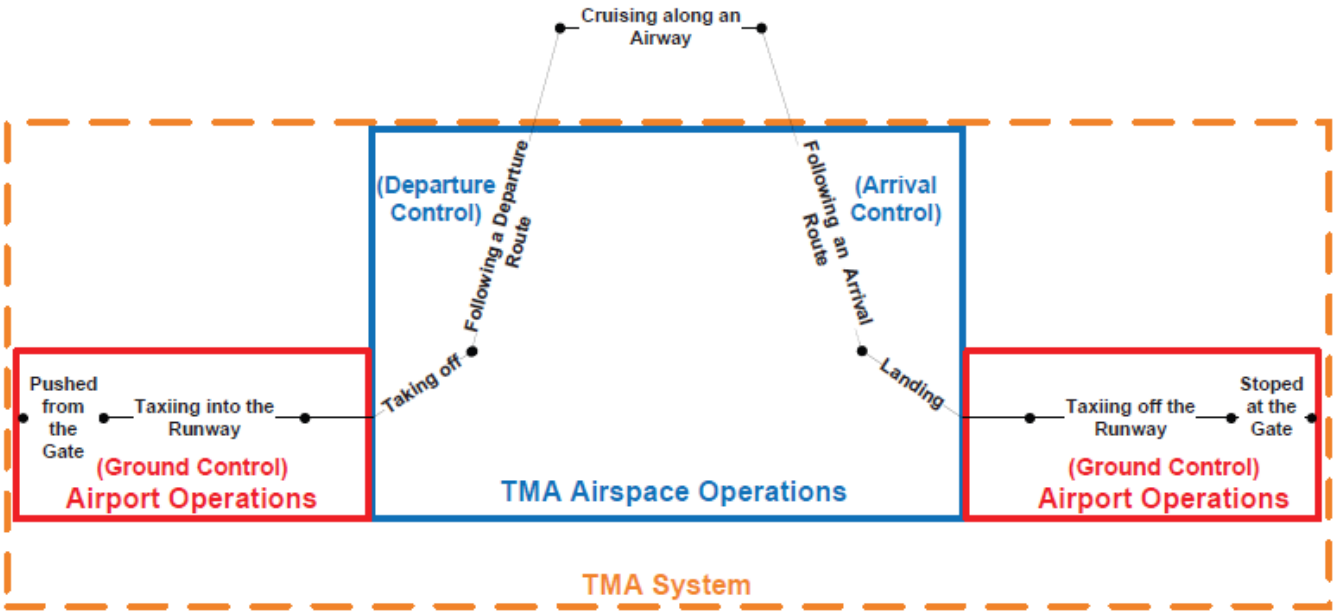

Figure 4: Phases of a typical flight trip

Except for the cruise phase, all other flight phases operate in the airport and its terminal airspace. The terminal area system is a highly complicated environment involving static elements, such as airport infrastructure, and dynamic events, such as arrivals, departures and ground events. Hence, airports form a critical part of the air transportation process.

Thus, the management of aircraft movements on the airport surface is subject to a variety of operational constraints, such as the compliance with safety regulations on surface operations, the required time-window for incoming arrivals and outgoing departures, the inherent uncertainty of the availability of ground resources, and the precedence constraints from a first come first served sequence. The challenge lies in how to accomplish system optimization considering the complex interactions among all parts of the operational environment.

Currently, many major airports are operating close to their maximum throughput capacity. A large number of flights are delayed due to the congestion on the airport surface. To increase airport capacity, runways and taxiways systems are being expanded. However, as the density and complexity of airport surface increases, safety concerns and the available capacities of an airport cannot be used effectively.

This project will be focused on terminal areas specifically on the effectiveness of ground operations. As mentioned above, the control of the ground movement of aircraft and vehicles rest with air traffic control (ATC), so that ground operations are managed by ground controllers (GC) from the control tower. Ground procedures currently being used are the Surface Movement Guidance and Control System (SMGCS) which are mainly based on the principle of 'to see and be seen' as the primary mean to ensure the safety of surface movements. Visual observations are used to estimate the relative positions of aircrafts and vehicles on ground operations.

SMGCS consists on visual aids such as lighting, markings and signage to guide pilots and vehicle along their assigned routes. Furthermore, this system allows the identification of intersections and holding points. SMGCS provide guidance to aircraft for movements on the entire aerodrome surface, including movements from runway to parking position on the apron and vice versa, as well as movements from one apron to another one. (*ICAO, 2004*)

However, the main principle of 'to see and be seen' is not fulfilled when adverse conditions occur, such as low visibility conditions. Therefore, advanced capabilities are needed to ensure the safety of surface movements when visual means are not adequate. For low visibility operations, SMGCS prescribes the operational procedures that must be followed during surface movements depending on the regulations and policies of Air Traffic Services (ATS) and the aerodrome configuration and facilities. Nevertheless, SMGCS cannot ensure the safety of operations in all visibility conditions. (*SESAR, 2008*)

Furthermore, the ATM system in this section is based on the principle 'first in first out' leading to a lack of optimization on the fluidity of the overall air traffic.

Currently in ground operations, ‘stop-and-go’ operations occur wasting resources, such as fuel and time, due to the lack of global vision that the system involves and the difficulty of predicting the consequences in the decision-making process.

ATM operations in each individual phase have an impact on the ATM effectiveness of the other phases. Having air traffic controllers with different responsibilities and objectives, make it difficult to achieve truly high efficiency for air transportation. This fragmentation, from a system perspective, implies the achievement of local objectives instead of the accomplishment of the global objective of the entire system.

This project will explore the introduction of Multi-Agent Systems (MAS) in ground operations to avoid the separately treat of ground movements of the overall system optimizing the air traffic fluidity and the usage of resources.

1.2. Motivation

Air traffic demand is expected to heavily grow in coming years. According to EUROCONTROL Long-Term Forecast 2010-2030 (LFT10, 2010), in the 'most-likely' scenario, there will be 16.9 million IFR movements in Europe in 2030, 1.8 times more than in 2009. The air traffic will grow an average of 2.8% annually in the 'most-likely' scenario and it will be faster in the early years.

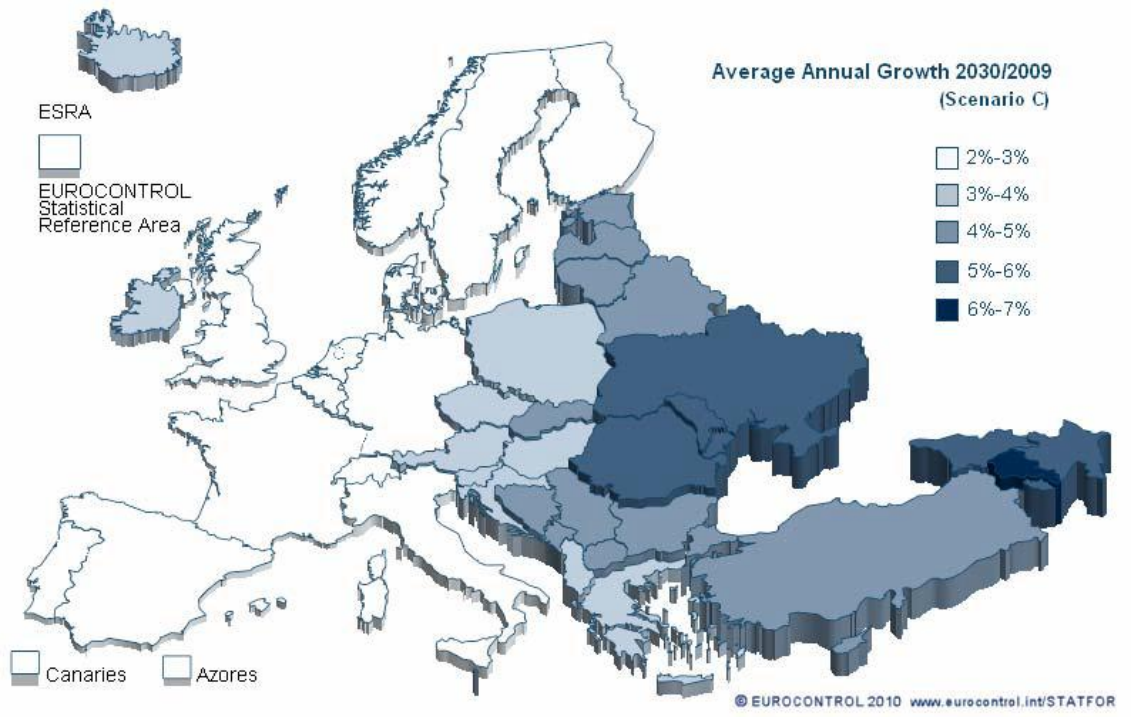

Figure 5: Average annual growth 2030-2009

Future air traffic will be limited by capacity at the airports, 0.7-5.0 million flights will not be accommodated in 2030; this represents between 5%-19% of the demand. Besides it is expected that in 2030 there will be 13-34 airports as big as the top 7 in 2010, congestion in airports will grow and it will create pressure on operation's fluidity generating delays and inefficiency on air traffic flow.

However, since 2010, traffic has not grown as it was expected as the 'most-likely' scenario. The economic and traffic downturn has emerged making the expected growth in demand go down about 6% in 2030. (*EUROCONTROL, 2010*)

Nevertheless, according to EUROCONTROL 7-year Forecast in 2011 it is expected that this downturn will reverse from 2013. As it can be seen in Figure 6, the growth of business aviation is expected to be reduced by 2% in 2012 compared to 2011 but in medium-term it is expected to continue its growth.

Figure 6: EUROCONTROL 7-year forecast

However, air transport demand continues to grow faster than the available air transport system capacity. Consequently appears the challenge of accommodating air transport under the limited airspace configuration and highly constrained ground operations under pre-defined terminal trajectories. (*EUROCONTROL, 2012*)

As a result, congestion delays will increase unless new ATM solutions are developed and implemented. Therefore, it is becoming crucial to enhance future ATM systems to ensure efficiency and safety on air traffic procedures in order to meet the expected demand. Research efforts are being carried out by Europe (Single European Sky ATM Research, SESAR) and the United States (Next Generation Air Transportation System, NextGen).

SESAR was created to change the architecture of European ATM. This programme is the technological and operational part of the Single European Sky (SES) initiative that aims to get support of airspace capacity and safety needs. The main purpose is to increase the overall efficiency of the air traffic

management system. This future ATM system will be a trajectory based programme involving advanced information sharing environment, automated tools, and improved surveillance capabilities.

Regarding ground operations, SESAR's concept intends to enhance the management of terminal resources in order to facilitate airport surface movements in a safe and cost effective way.

SESAR's programme proposes the introduction of Advanced Surface Movement Guidance and Control System (A-SMGCS) which will provide enhanced information to controllers whilst Cockpit Display of Traffic Information (CDTI) technology will provide aircraft and vehicle drivers with guidance traffic information. (*EUROCONTROL, 2012*)

A-SMGCS will be developed to improve the currently SMGCS system. The main benefits of A-SMGCS will be associated with low visibility surface operations. The following goals are some of the improvements for surface movement operations that are involved in this concept: (*ICAO, 2004*)

- It should provide more precise guidance and control for all aircraft and vehicles on the movement area and should be able to ensure spacing between them in all weather conditions.
- It should provide situation awareness not only to ATC, but also to aircrafts and vehicles that are liable to come in proximity to each other.
- It will reduce voice communications, improve surface guidance aids and increase reliance on avionics in the cockpit to help guide the pilot to and from the runway.
- Automated data communications will be introduced providing information between users to assist in monitoring surface operations.
- Improved ATC surveillance will provide accurate information on the position and identify all aircraft and vehicles operating on the movement area.

- Automated functions will include the monitoring of conformance with taxi instructions and the detection of potential conflicts and their resolution.
- Surface traffic planning automation functions will be integrated with approach and departure operations.

In the same way, the US NextGen aims to adjust ATM's system capability to satisfy forecast demand. The key objective is to increase the capacity of the air traffic in a safety way by monitoring airplanes location. This new technology will allow airplanes to know the precise locations of other aircrafts around them both in the air and in the ground procedures.

The US NextGen also highlights the need for modern airports ground movement procedures integrating the terminal area with the en-route constraints in order to achieve gate-to-gate trajectory-based operations. (*FAA, 2012*)

It is well recognized that challenges in ATM cannot be treated separately. Air transport is a continuous sequence of arrival, turnaround, departure and flight route events, and the airport process has a direct effect in all of these phases of the trajectory.

Airport operations involve the real-time, safe and efficient management of aircrafts on the airport platform. The airport throughput is one of the main processes that determine the on-time performance of the overall air traffic flow. Due to this factor, the efficiency of airport surface movements determines whether delays in air traffic can increase or be recovered.

Moreover, if delays are necessary, then they must occur on the ground rather than on the air, where they would be more expensive and worse for the environment. With the application of a ground delay program it is possible to save considerable amounts of fuel having the delayed flights on their departing airport with the engines off. Therefore, optimum management of surface traffic flow will not only increase efficiency and predictability during the ground movement phase but will also have a positive impact on the environment.

Then, the efficiency of future terminal areas is essential to be able to accommodate the projected increase on air traffic demand. The complexity of managing operations in terminal areas is given by the challenge of balancing capacity and demand in consideration with dynamic constrained ground resources through the integration of arrival and departure operations. (*Weinjing Zao, 2012*)

As a result, this project will be focused on the optimization of ground operations, more specifically, on the managing of taxiways. It will explore the introduction of Multi-Agent Systems(MAS) for a taxi management tool that aims to optimize the sequencing and scheduling of traffic, as well as to minimize delays and to reduce the number of stop-and-go during taxiing.

Reducing the workload of controllers through the implementation of MAS airports capacity can be potentially incremented. The combination of these systems with enhanced taxi display systems in the cockpit that may enable the flight crew to operate independent from the ATC, so that autonomous taxiing could be implemented.

1.3. Objectives

As mentioned above, this thesis will explore the introduction of Multi-Agent Systems (MAS) to improve the fluidity of ground operations, specifically the main purpose is to analyse the effectiveness of MAS in taxiways management.

The introduction of MAS as a taxi management tool aims to optimize the sequencing and scheduling of traffic, as well as to minimize delays and to reduce the number of stop-and-go during taxiing. The proposed concept will provide aircrafts with a high precision taxi capability to precisely follow the de-conflicted movements' plans produced by the taxi planning tool.

Due to the significant paper which taxiways represent as a connector of the different phases of a turnaround, it is expected that the introduction of MAS in taxiway management will help to enhance the fluidity of the overall ground operations and furthermore, the efficiency of air traffic flow.

MAS will allow real-time communication between aircrafts to determinate which is the best trajectory according to a set of initial conditions predefined by the airline company. This system will provide paths in 4D, where the fourth dimension is the time in which these actions will be executed, taking into consideration the pre-defined flight schedule. Furthermore, it will reduce the workload of air traffic controllers contributing to the improvement in the overall air traffic management.

Reducing the workload of controllers through the implementation of MAS airports capacity can be potentially incremented. The combination of these systems with enhanced taxi display systems in the cockpit that may enable the flight crew to operate independent of the ATC, so that autonomous taxiing could be implemented. Hence, autonomous taxiing would allow upgrading the actual structure of ATC. Reducing the workload on ground controllers, and having a shared information environment with the implementation of MAS tools, it would be possible to follow the objectives of SESAR and NextGen unifying the current ATC structure.

To summarize the high level goals of this thesis are the following:

- Analyze whether the introduction of MAS systems as a taxiways management tool enhance the effectiveness of ground operations.
- Explore different scenarios with a different set of previous conditions each one to determine the optimum configuration for each situation.
- Prove the introduction of MAS to minimize delays and to reduce the number of stop-and-go during taxiing, thereby reducing the environmental impact and fuel costs.
- Reduce controllers' workload and fluidity on ground operations to increment airport's capacity.
- Consider the possibility of autonomous taxiing through the implementation of MAS tools.

Chapter 2

Concept

2.1. Multi-Agent System MAS

Agent-oriented techniques (AOT) enable analysis, design, building and implementation of complex systems by providing abstraction levels that make it simpler to deal with. AOT aims to handle these complex systems with a uniform concept: agents. Systems modelled by a collection of agents are called multi-agent systems (MAS).

MAS can be used for simulation and planning purposes, enhancing the autonomy of single systems components and realizing new system solutions. AOT are designed to have significant impact in domains where subsystems are functionally or geographically distributed, where a dynamic environment exists and subsystems have to interact with more flexibility. The main benefit of using MAS techniques is the reduction of complexity in systems due to the concise modelling of the problem domain. (*B. Burmeister, A. Haddadi, G. Matylis, 1997*)

Air traffic domain contains the very characteristics for which AOT has been developed. MAS' techniques should improve existing air traffic guidance and control systems by increasing the autonomy of single systems and enhancing their interaction capabilities. Agents can cooperate between them in order to increase their performance and coordinate their activities so as to collaboratively adapt to the dynamics of ground operations.

Therefore, aircrafts can be modelled as agents who coordinate their movements on taxiways. This could be a significant improvement of the current systems where ground operations are fully controlled from ATC. MAS' tools will provide information technology and techniques to make air traffic management more efficient, resource saving and ecological friendly.

2.1.1. What is an Agent?

Adopting an agent-oriented approach means decomposing the problem into multiple interacting components (agents) that have particular objectives to achieve. Agents are considered problem solvers that operate in an environment over which they only have partial control and observation. Agent-oriented interactions occur through a high-level communication language in terms of which goals should be followed, at what time and by whom.

At present, there is a deal when introducing the concept of “Agent” in the field of Multi-agent Systems (MAS). However, several researchers find appropriate the following definition: *“An agent is an encapsulated computer system that is simulated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives”*. (*Wooldridge, 1997*) Even though it is a fairly complete and comprehensive definition, there are some characteristics and properties of agents that are important to go in depth to understand the concept of agent in MAS domain.

The main characteristic that defines an agent is its autonomy; they have control both over their internal state and over their own behaviour to act in response of perceived changes in the environment. Therefore, agents have the ability of interact with the environment where the agent is situated; they receive inputs related to the state of that particular environment through their sensors and they act on it through their effectors. (*Adina Magda Florea, 1998*)

Agents are designed to fulfil a specific role. They need to be flexible to achieve their particular objectives, so that they need to be reactive to respond to changes that occur in their environment, and pro-actives to adopt new goals and take the initiative, in order to satisfy their objectives. (*Nicholas R. Jennings, Michael Wooldridge, 1999*)

When adopting an agent-oriented view to complex systems, multiple agents are needed to represent the multiple perspectives or the competing interests. Thus, agents need social ability to interact with one another via some kind of agent-communication language.

Moreover, when agents have the capability to learn then they go from computational agents to intelligent agents. This occurs when methods and techniques specific to artificial intelligence are used by agents to select the best operations necessary to achieve their goals.

To summarize, agents are characterized by being autonomous, reactive, proactive and social. Furthermore, agents are commonly modelled using mental notions, such as knowledge, belief and obligation. Then, agents are able to decide whether or not to execute an action after receiving a request.

2.1.2. Decision Support System DSS

Introducing MAS in ground operations management paves the way to evaluate the interactions between system components from the perspective of identifying inefficiencies. By evaluating these interactions and the possible configurations to reveal improvement opportunities, MAS techniques will significantly improve understanding and decision making. (*Dejan Lavbic, Rok Rupnik, 2009*)

In such a complex environment decisions must be made quickly, analysing and sharing large amounts of information with multiple actors. A taxiways management system includes several entities: the different airline companies involved and, for each airline company, different aircrafts specialized in the accomplishment of a specific flight schedule. Such scenario makes urgent the realization of new tools for effectively using, sharing and retrieving the information flowing in the system.

MAS' techniques represent a very appropriate platform for integrative decision support systems due to the similarities between the agents and the human actors in terms of their characteristics and coordination. Thus, intelligent agents represent actors in human organizations.

The management of ground operations on taxiways are constantly subject to unpredictable system dynamics. In this thesis, MAS tools are implemented to

automatically organize taxiways management maximizing the utility of the whole system. The success of the parties involved depends on the capability of each scheduled flight and of the whole group of aircrafts to adapt to the changes in the environment. Adapting their strategies and behaviours, the whole group of partners is able to exploit new solutions and configurations that can ensure the fluidity on ground operations reducing delays and stop-and-go operations to guarantee a profit for each partner and optimise the whole system.

2.1.3. Agent-Oriented Simulation MASS

Following the enormous evolution in the number of flights appears the necessity of an efficient supervision system supporting complexities, distributions and dynamicity. The objectives of a taxiway management system are to fluidize and to secure the traffic and to anticipate reacting on events in a real-time traffic administration, what brings a better request management and a safer displacement through the airport platform.

The simulation is a way of testing models used to confirm the efficiency of this model and to refine it so that it will respect real world constraints. It facilitates understanding and representing entities' behaviours in a determinate environment. (*Mohamed Habib Kammoun, Ilhem Kallel, Mohamed Adel Alimi, 2005*)

Therefore, aircraft must be seen as an autonomous and intelligent subsystem, namely an agent. The advantages of using an agent-oriented approach to manage ground operations in taxiways are a high-level description of aircraft behaviour, which is easier to edit, change, modify and maintain; and the possibility to easily model explicit communication and coordination among aircrafts.

The taxiway management system based on multi-agent approach has the following objectives: first of all, to find the best network exploitation and

administration; secondly, to react to the users' needs dealing with the dynamics of the system and the best path attempting his destination.

Each agent follows an iterative process based on reception-deliberation-action. Reception represents the observation and interpretation according to the agent internal beliefs of the information about the environment. The deliberation is the process in which the agent determinates its action according to its internal rules while taking into account their static and dynamic knowledge. In the action phase the agent executes the action previously chosen in order to update its dynamic knowledge or act in the environment.

2.1.4. Agent Modelling

The MAS technology considers that is possible to model not only the individuals (aircrafts) involved and their behaviours, but also the interactions between them in the environment. The macroscopic level of global system dynamics results directly from the interactions between individuals that compose the microscopic level. Therefore, multi-agent approach directly models the interactions generated by individual behaviours. (*Adina Magda Florea, 1998*)

Agents' architecture has different modules, namely: actuators, sensors, motivation, communication and cognition. The cognition module controls and monitors the individual agents' behaviour by relating its perception through its sensors to the execution of appropriate reactions through its actuators to accomplish the objective contemplated in the motivation module. (*Mohamed Habib Kammoun, Ilhem Kallel, Mohamed Adel Alimi, 2005*)

An agent has beliefs about itself such as position, velocity and lane number; about other agents such as position of the surrounding aircraft and their speed; and about the environment such as lanes configuration, cross-points and pre-defined flight scheduled. Different states are defined such as ongoing, stopped in cross-point or stopped in gate/remote position. When a state change occurs,

the agent deliberates its response according to its internal rules while taking into account static and dynamic knowledge. An action represents the operation that an agent executes in order to be able to upgrade its dynamic knowledge, to send a message to other agents, or to act in the environment. (*Birgit Burmeister, Afsaneh Haddadi, Guido Matylis, 1997*)

2.2. MAS Tools

Currently, there is a huge variety of agent tools on the market to develop and construct intelligent software agents. Agent-based modelling tools range from general agent development platforms to highly specialized tools. The most important issue for agent-based software engineering is the understanding of the situations in which agent solutions are appropriate.

An agent tool is defined as any software package, application or development environment that provides agent builders with a sufficient level of abstraction to allow them to implement intelligent agents with desired attributes, features and rules. (*Nicholas R. Jennings, Michael Wooldridge, 1999*)

Some agent tools only offer a platform for agent development, whereas others also provide an environment for running, monitoring, analyzing and testing agents through features for visual programming. Multi-agent tools have to deal with the interaction that involves the coordination of actions between agents and the adaptation of agent behaviour in response to the environment. The following table provides a summary of the more popular multi-agent tools.

Multi-agent tools	Domain	Technical caracteriscs
Agent Sheets	• Computational science • Games	• Conversational programming (Visual AgentTalk) • Exportable to Java
AnyLogic	• Supply chain and logistics • Transportation and warehousing • Markets and competition • Health • Manufacturing and production • Strategic planning and management • Business processes and service systems	• Simulation tool for: system dynamics, discrete-event and agent-based modelling • Java environment • 3D view

Multi-agent tools	Domain	Technical characteristics
Breve	• Computational science	• Python programming language or Steve scripting language • 3D view
Cougaar	• Computational science • Large scale agent systems • Distributed logistic planning	• Cognitive agent architecture • Java environment
JADE	• Computational science • Distributed applications	• Java Agent Development Framework • Java environment • WADE extension to workflows
MASON	• Multi-agent Simulation Of Neighbourhoods (or Networks) • Discrete-event • Large-custom purposes	• Java environment • 3D view
NetLogo	• Art • Biology • Chemistry and physics • Computational science • Earth science • Games • Mathematics • Networks • Social science • System Dynamics	• Logo dialect of Logo Language • 3D view
Repast Symphony	• Social science • Consumer products • Supply chains • Others	• ReLogo dialect of Logo Language • Flowcharts • Groovy • Java • NetLogo models can be imported

Table 1: Multi-agent tools

2.2.1. NetLogo

To perform the simulation of the scenario considered in this project it has been chosen the NetLogo software as it is well suited for modelling complex systems. NetLogo is a free programmable modelling environment used by students and researchers worldwide.

Figure 7: NetLogo interface

Modellers can give instructions to a huge amount of agents all operating independently. Exploring the behaviour of the individuals and the patterns that emerge from their interaction under various conditions, allows modellers to investigate complex systems easily changing variables in the environment to see and analyse the reaction both at the individual and at the macro-level of the overall system.

NetLogo has a friendly use environment with three main tabs, namely, interface tab, info tab and programming tab. The interface tab is where the model visually runs. It has different tools to see what's happening inside the model and alter its behaviour by changing different commands, such as changing values of global variables or switching on and off some policies defined in the model. In this tab, relevant information of the system can also be monitored to easily analyse its behaviour, such as monitors that automatically report the updated value of a global variable or plots to show the data that the model is generating.

This interface can be modelled in 2D and 3D views. The simulation designed for this project is represented in 2D due to the irrelevance of the third dimension, which represents the height, to model the taxiway management as a

sub-system of the overall airport system. The visual representation of the model shows the world defined by agents.

In NetLogo, there are four types of agents: turtles, patches, links and the observer. Turtles are agents that move around the world which is divided into a grid of patches. Each patch is a square of the ground over which turtles can move. Links are agents that connect two turtles. Finally, the observer gives instructions over the world of turtles and patches. Furthermore, agent sets can be defined to contain a group of agents either turtles, patches or links, but not more than one type at once, to efficiently organize the characteristics and patterns between turtles.

Moreover, in the interface tab there is a command centre to introduce commands directly without adding them to the model's procedures. This is a useful tool for inspecting and manipulating agents on the fly.

On the other hand, the code tab is where the programming code of the model is introduced. NetLogo's programming language is Logo dialect extended to support agents. This software has its own dictionary with a large vocabulary of built-in language primitives and functions that allows running mobile agents (turtles) over a grid of stationary agents (patches).

Finally the info tab provides an interface to the modeller to explain which system is being modelled, how it was created and how to use it. (*NetLogo Web Site, 2013*)

Chapter 3

Implemented Model

3.1. Considered scenario

In this project, MAS techniques are proposed to implement the concept of a taxiway management tool that allows the surface traffic automation. The main purpose is to prove that MAS techniques could be a powerful tool to support air traffic procedures such as taxiway movements on the airport surface, to alleviate the controllers' workload with a particular focus on the primary objective: increasing the capacity of current airports to improve the overall air transport network. More specifically, the developed model demonstrates that autonomous taxing simulation is possible through the implementation of multi-agent systems.

Besides, the model implemented in this project is an academic tool developed for a simple hypothetical airport, the proposed taxiway management concept is able to provide a platform where autonomous communication between aircraft is achieved and safety surface movement is assured. The scenario contains five gates and one runway interconnected by taxiways.

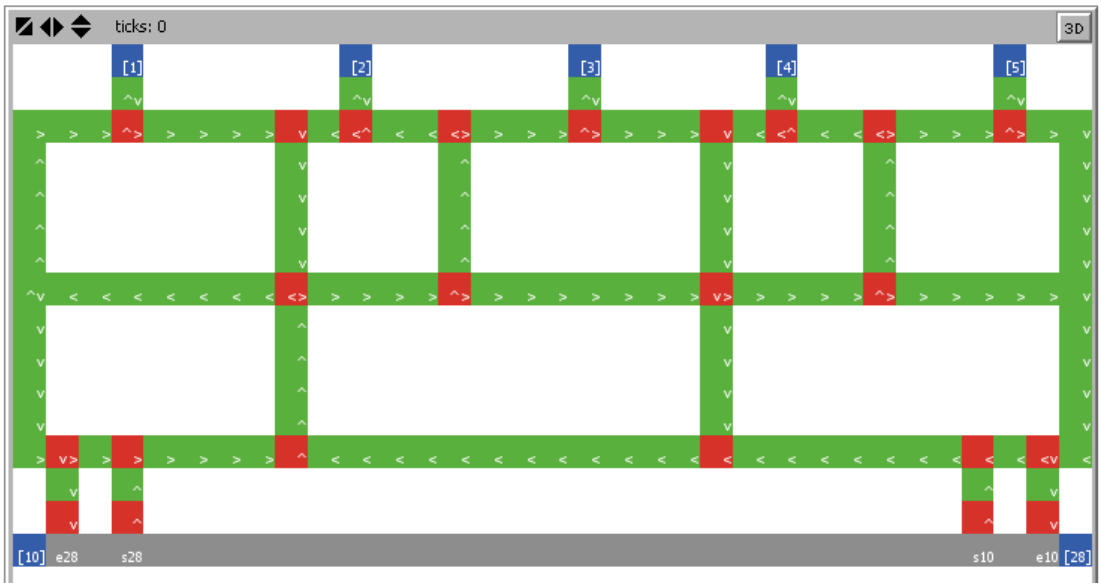

Figure 8: Considered Scenario

3.2. Agents description

To better organize the proposed system, different kinds of agents are defined to take advantage of their cooperative characters while minimizing the risk of objective conflict. In the proposed scenario, agents are mainly divided in turtles and patches. Turtles represent airplanes moving on the airport platform and patches conform the taxiways, the runway and the gates.

Moreover, turtles and patches are divided into agent sets to easily group agents according to their characteristics and to facilitate the internal patterns and communications between agents.

Therefore, the scenario considered involves the following kinds of agent sets:

- Aircraft
- Cross-points
- Taxiways
- Gates
- Runway
- Entry and exit of the runway
- Landing / Take-off

Aircraft agents are represented by turtles and contain information about the specific flight which they represent. This information is individually saved and updated during the simulation for each particular turtle.

The program defines a set of default variables for each turtle that is created. Worth noting that each one contains a unique identifier, the *who* number. The *colour* is pre-defined for all turtles as white. The *heading* is an important variable for this model since it determines the direction of the aircraft in 360 degrees, where 0 points upwards. The *shape* is an airplane as it can be seen in the following screenshot.

Furthermore, for this particular model other variables have been defined. The *n-stop-and-go* is a counter that stores the number of stops and goes that this concrete airplane has done during its life in the scenario. The boolean variable *low-cost?* determines whether this airplane belongs to a low cost airline company or not. The *origin* and *destination* variables contain the name of the specific gate or runway header of the origin and destination of its trajectory. The *n-dec-point* is a counter that stores the number of decision points that the airplane has passed. The *turnaround* is a reversed counter of the time remaining to complete the turnaround operation and it starts when the airplane arrives to its destination gate. The boolean variable *stopped?* shows if the airplane is currently stopped due to a collapse or an stop-and-go operation. The *time-stopped* stores the total amount of time that the airplane has been stopped. Finally, the boolean variable *g-stopped?* shows if the airplane is currently stopped waiting to enter to the assigned gate.

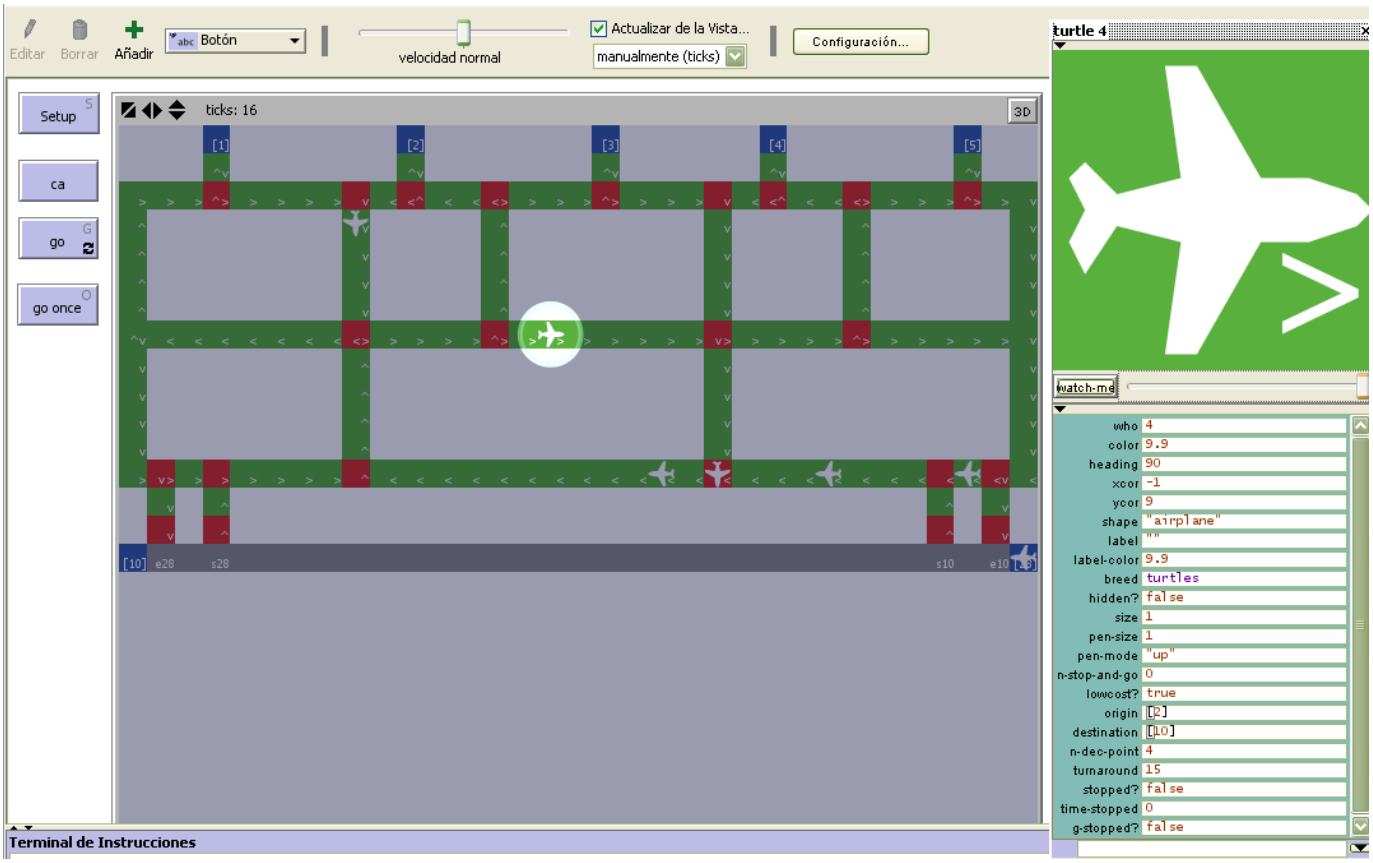

Figure 9: Aircraft agents

The rest of agents are patches divided in different agent sets. All patches have the same variables, what distinguishes one group from others is the value adopting these variables. All patches have some variables predefined by the program, and those are: *pxcor* and *pycor* that show the value of the x and y coordinates where the patch is situated, *pcolor* determines the colour of the patch, *plabel* stores what is printed in this patch and *plabel-color* determines the colour of this label.

Taxiways are defined as an agent set containing all the patches that conform the taxiways that interconnect the gates to the runway. Patches also have variables defined specially for this model. The boolean variables *left?*, *right?*, *up?* And *down?* as their name suggests, determine the direction of the patch. The *direction-point?* variable determines that the current patch is a decision point when two or more directions are possible. The *crosspoint?* determines if the patch pertains to an intersection between two taxiways. The boolean variable *available?* determines the availability of the runway. The *rw?* shows whether the current patch is part of the runway or not. The *g-crosspoint?* indicates that this specific cross-point is the entry/exit of one of the gates.

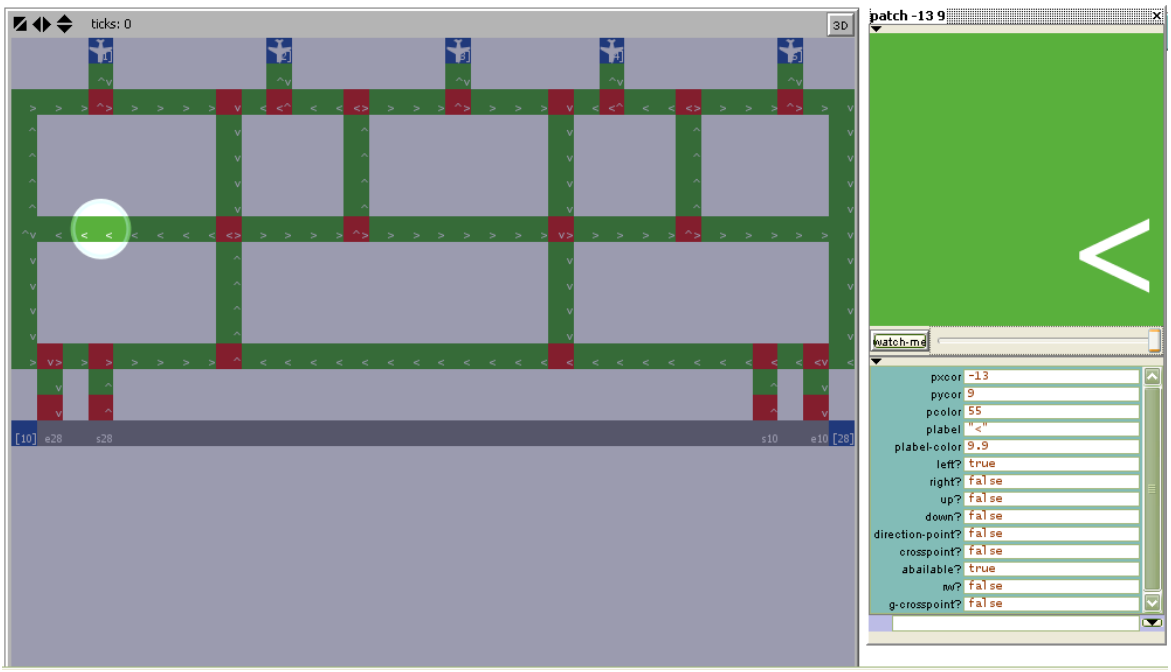

Figure 10: Taxiways patches

Cross-points are represented as an agent set of patches defined as taxiways containing the intersections between them. Therefore, the boolean variable *crosspoint?* will be true. The *pcolor* value for cross-points is red to distinguish them from the rest of taxiways patches.

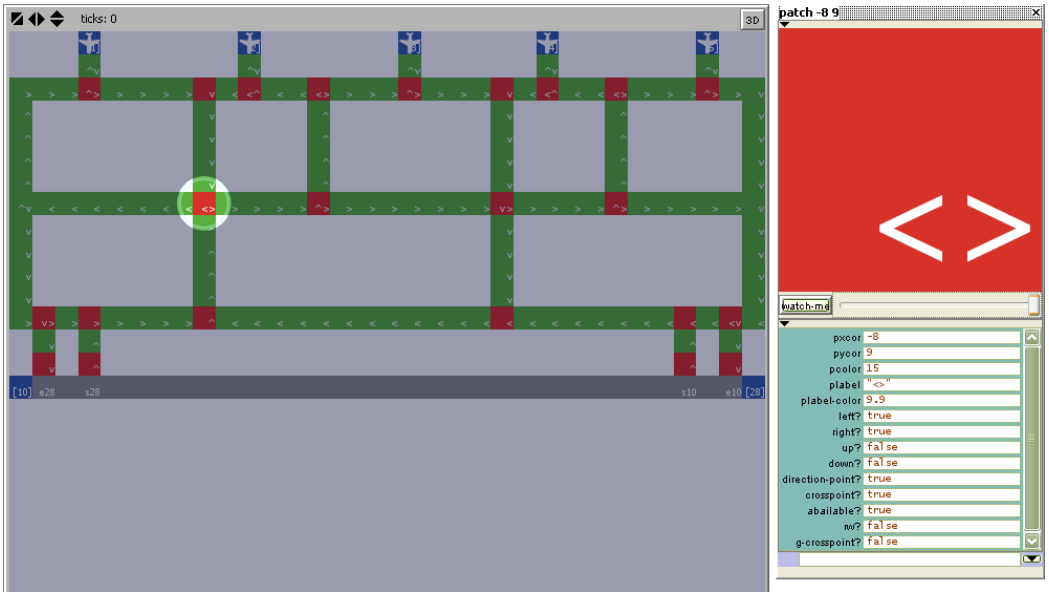

Figure 11: Cross-points patches

Gates are defined as an agent set independent of the *Taxiways* agent set containing the patches that represent the gates and where the turnaround of aircraft takes place. Here, only the variables predefined by the program have relevance.

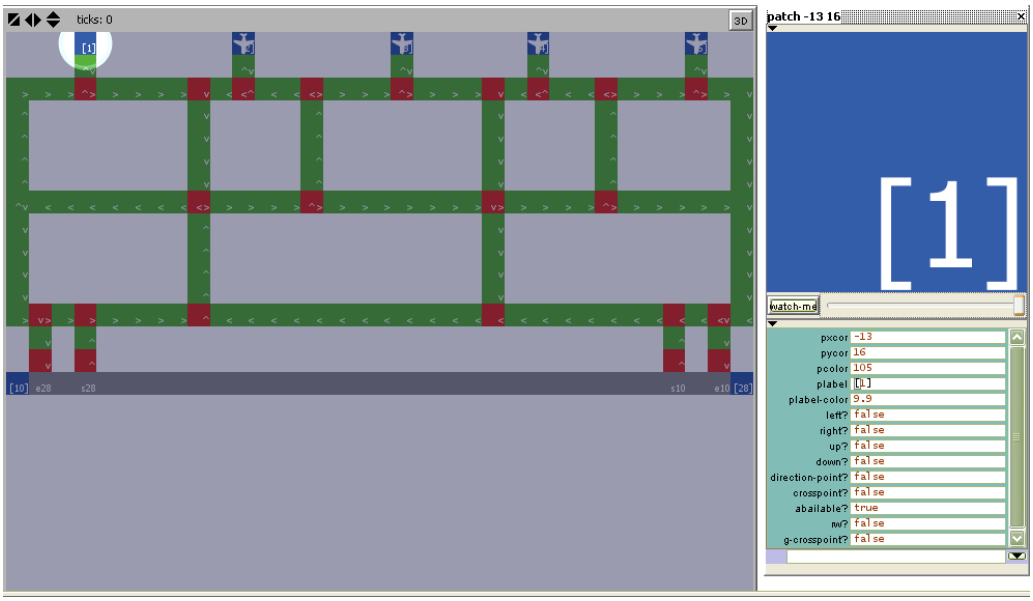

Figure 12: Gates patches

Entry and exit of the runway agent set is also independent from *Taxiways* and represents the interconnection between the taxiways and the runway in both directions. Here, only the predefined variables and the direction ones have relevance.

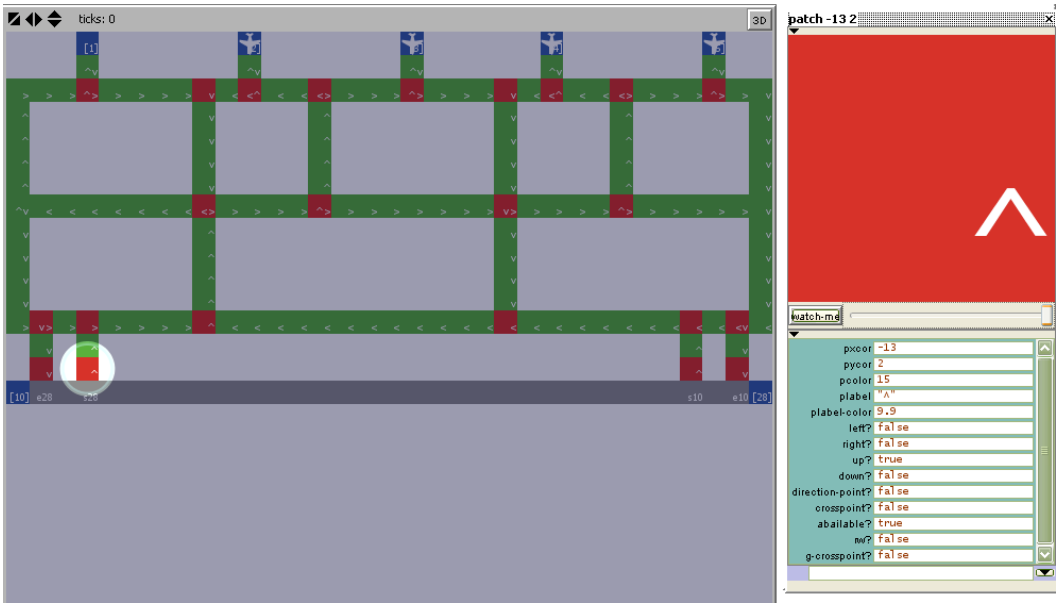

Figure 13: Entry and exit of the runway patches

Runway is another agent set which, as its name suggests, contains the patches that conform the runway, then the variable *rw?* will be true for this agent set. Here, labels take an important role because they will determinate the direction of the aircraft.

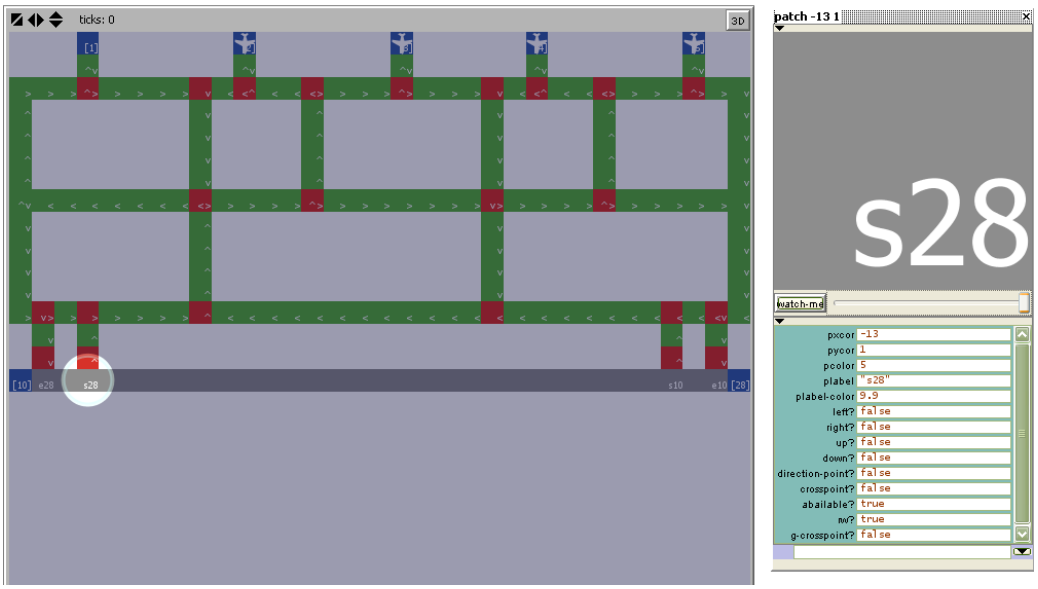

Figure 14: Runway patches

Landing / Take-off is an agent set of *Runway* agents from where the aircrafts land and take-off the runway. When the origin of an aircraft is one of the headings of the runway it is created in the corresponding patch. On the other hand, when the destination is one of the headings, which means that it will be the last patch that the aircraft will touch before leave the runway, the aircraft agent dies.

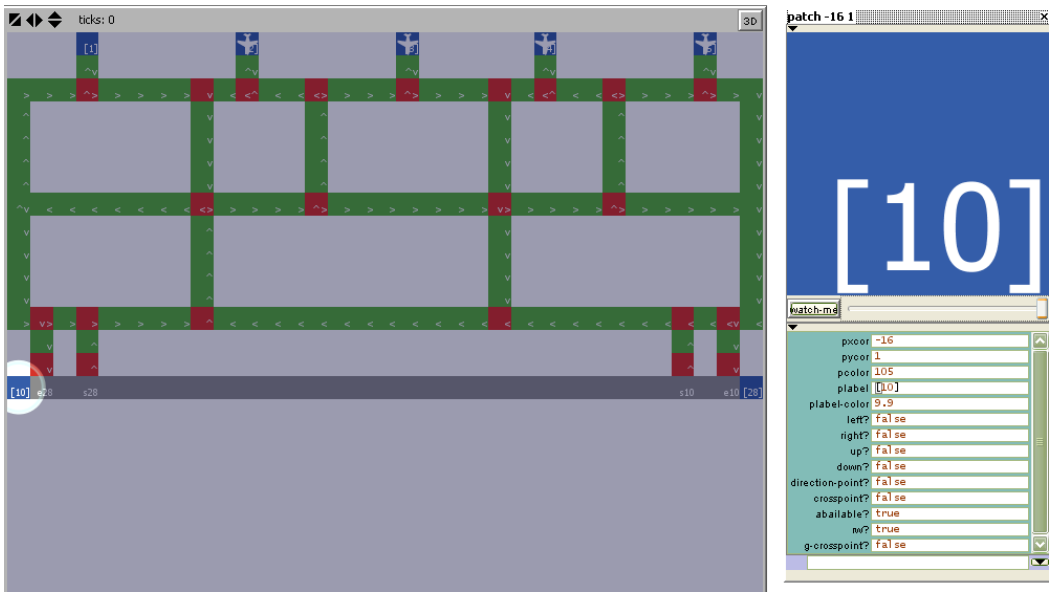

Figure 15: Landing / Take-off patches

3.3. Model flow

The developed model is an academic scenario which contains five gates with one runway, where each gate represents one finger of the airport terminal where the airplanes do the turnaround. The time of turnaround is predefined to all aircrafts equally; however, random variation of the turnaround can be implemented to make it proper to reality. The runway is configured in both directions and it depends on a random variable which determinates the origin and the destination of each aircraft.

The scheduling of flights is not predefined; aircrafts are created and appear into the world depending on the frequency established and the availability of the resources, thus, for an agent to be created there must be availability in the origin position. The frequency in which aircrafts appear in the world can be altered during the simulation to explore the affectations that could represent peaks in the demand during a real scheduling of a real airport.

This model is capable to coordinate arrivals, departures and turnarounds with a predefined taxiway's trajectory given by the established origin and destination of the specific flight.

These trajectories have been defined considering the shortest path for each route. Each independent aircraft contains information about its origin and its destination and the global system contains the information about the trajectory that must follow according to this data.

In NetLogo, time passes in discrete steps called ticks. This is a counter that shows the current number of ticks that the procedure has done. When the simulation starts, ticks are reset and each time the model goes on the *go* procedure the number of ticks increases by one unit.

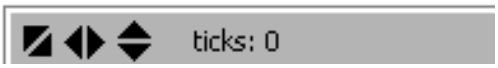

Figure 16: Ticks counter

Therefore, the frequency in which aircrafts appear on the scenario depends on the number of ticks and is defined by default every 5 ticks, but it can be modified during the simulation to see the impact that they have on the scenario.

Figure 17: Frequency slider

The setup situation simulates an airport at the early hours of the day, all gates are occupied and the aircrafts are ready to go.

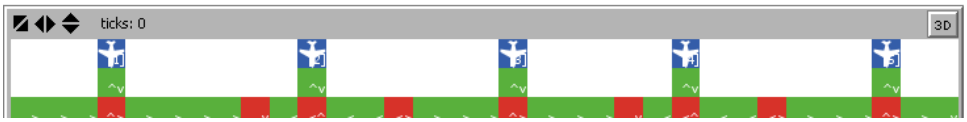

Figure 18: Setup situation

Then, aircrafts are created according to the frequency established. During the first 100 ticks new aircrafts can be created both on gates and on runways. For every two arrivals one departure aircraft appears. As it is shown in the following code, it depends also on the location. The function *setup-location* checks the availability of the corresponding resource, such that, if at the current tick one aircraft is going to be created at gates, it ascertains that there is at least one gate with no aircrafts entering/leaving the gate or on it, and the same for the runway.

```
if (ticks > (frequency - 1) and ticks mod frequency = 0) [create-agents]
```

```
to create-agents
  ifelse time = 0
  [
    crt 5 [
    setup-agent
    setup-location
    ]
    set total-number-flights 5
  ]
  [
     ifelse (not all? p-gates [any? turtles-on self]) and
     (time < 100 and (time > (frequency - 1) and time mod (frequency * 2) = 0))
     [
        if ((not any? turtles-on patch -13 14 and not any? turtles-on patch -13 15 and not any? turtles-on patch -13 16) or
       (not any? turtles-on patch -6 14 and not any? turtles-on patch -6 15 and not any? turtles-on patch -6 16) or
       (not any? turtles-on patch 1 14 and not any? turtles-on patch 1 15 and not any? turtles-on patch 1 16) or
       (not any? turtles-on patch 7 14 and not any? turtles-on patch 7 15 and not any? turtles-on patch 7 16) or
       (not any? turtles-on patch 14 14 and not any? turtles-on patch 14 15 and not any? turtles-on patch 14 16))


       [
         crt 1 [setup-agent
                 setup-location
                 set total-number-flights total-number-flights + 1]
       ]

     ]

     [
       if not any? turtles-on p-runway and (not any? turtles-on patch 13 2 or not any? turtles-on patch -13 2)
       [ crt 1 [setup-agent
         setup-location
          set total-number-flights total-number-flights + 1
         ]
       ]
     ]

   ]

  set-default-shape turtles "airplane"
end




to setup-location

 ifelse (time < 100 and (time > (frequency - 1) and time mod (frequency * 2) = 0)) or time = 0
 [
       if (not any? turtles-on patch -13 14 and not any? turtles-on patch -13 15 and not any? turtles-on patch -13 16)
       [
         move-to patch -13 16
         ask patch-at 0 0 [set set-origin plabel]
         set origin set-origin
         set heading 180
         ask one-of p-landing-takeoff [set set-destination plabel]
         set destination set-destination
         set created? true
       ]


[
     ifelse not any? turtles-on patch -13 2
  [ move-to patch 16 1
    ask patch-at 0 0 [set set-origin plabel]
    set origin set-origin
    ifelse set-origin = [10] [set heading 90][set heading 270]
    ask one-of p-gates [set set-destination plabel]
    set destination set-destination
  ]

  [ move-to patch -16 1
    ask patch-at 0 0 [set set-origin plabel]
    set origin set-origin
    ifelse set-origin = [10] [set heading 90][set heading 270]
    ask one-of p-gates [set set-destination plabel]
    set destination set-destination
  ]

]

end
```

This implies, as mentioned before, that scheduling of arrivals and departures and their location are not predefined. Furthermore, if there is no availability on location, this aircraft is not created, hence delays on new incoming arrivals or departures are not contemplated. Moreover, this configuration priories the creation of agents with origin in runway header [10] rather than [28]. The same happens with gates, the priority goes from gate 1 to 5 when depending on their availability.

Once an agent is created, the origin variable is assigned to the location where the agent has been created; if there is more than one available location to be created it is defined randomly. The same happens with the destination but here the availability is not considered.

When an aircraft is created, it is ready to go to its destination. Predefined trajectories indicate the path to follow through taxiways to get from origin to destination. These trajectories are defined in the model as a matrix, as it can be seen in the following table, the matrix contains for each column the indications to follow for each combination of origin/destination. The row level represents the decision point where the aircraft is situated.

A decision point is defined in the model as a patch where the aircraft is able to go to more than one direction. The numbers in each matrix position represent the action that the aircraft must do to follow the route, namely, 0 means go straight, 1 turn left and 2 turn right. Positions with -1 are not used because the aircraft has arrived to its destination.

dec-	1	1	2	2	3	3	4	4	5	5	10	10	10	10	10	28	28	28	28	28
point	28	10	28	10	28	10	28	10	28	10	1	2	3	4	5	1	2	3	4	5
0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	1	2	2	2	2
1	1	1	2	2	1	1	2	2	1	1	2	1	1	0	0	2	1	1	0	0
2	2	1	2	1	0	1	0	1	0	1	1	1	2	0	0	1	1	2	0	0
3	1	0	1	0	1	0	1	0	1	-1	0	2	1	1	1	0	2	1	1	1
4	2	0	2	0	1	1	1	1	1	-1	-1	0	0	1	2	-1	0	0	1	2
5	-1	0	-1	0	2	-1	2	-1	2	-1	-1	-1	-1	2	1	-1	-1	-1	2	1
6	-1	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	-1	-1	0	0

Table 2: Trajectories matrix

The following code shows how the airplane decides its direction each time it is in a decision point.

```
to look-m-trajectory

  if not stopped?
  [
    if (origin = [1] and destination = [28]) [decide-direction n-dec-point 0]
    if (origin = [1] and destination = [10]) [decide-direction n-dec-point 1]
    if (origin = [2] and destination = [28]) [decide-direction n-dec-point 2]
    if (origin = [2] and destination = [10]) [decide-direction n-dec-point 3]
    if (origin = [3] and destination = [28]) [decide-direction n-dec-point 4]
    if (origin = [3] and destination = [10]) [decide-direction n-dec-point 5]
    if (origin = [4] and destination = [28]) [decide-direction n-dec-point 6]
    if (origin = [4] and destination = [10]) [decide-direction n-dec-point 7]
    if (origin = [5] and destination = [28]) [decide-direction n-dec-point 8]
    if (origin = [5] and destination = [10]) [decide-direction n-dec-point 9]
    if (origin = [28] and destination = [1]) [decide-direction n-dec-point 10]
    if (origin = [28] and destination = [2]) [decide-direction n-dec-point 11]
    if (origin = [28] and destination = [3]) [decide-direction n-dec-point 12]
    if (origin = [28] and destination = [4]) [decide-direction n-dec-point 13]
    if (origin = [28] and destination = [5]) [decide-direction n-dec-point 14]
    if (origin = [10] and destination = [1]) [decide-direction n-dec-point 15]
    if (origin = [10] and destination = [2]) [decide-direction n-dec-point 16]
    if (origin = [10] and destination = [3]) [decide-direction n-dec-point 17]
    if (origin = [10] and destination = [4]) [decide-direction n-dec-point 18]
    if (origin = [10] and destination = [5]) [decide-direction n-dec-point 19]
    set n-dec-point n-dec-point + 1
  ]
end


to decide-direction [row column]

  if(( matrix:get m-trajectory row column) = 0) [move-fd who]
  if(( matrix:get m-trajectory row column) = 1) [set heading heading - 90   move-fd who]
  if(( matrix:get m-trajectory row column) = 2) [set heading heading + 90   move-fd who]

end
```

When the assigned destination of the aircraft is one of the gates, once the aircraft arrives to this gate, the turnaround is executed. By default, the time of each turnaround is 15 ticks, but in the interface tab it can be added randomness to this variable by a switch button which makes that the 50% of turnarounds have a variation between 0 and 15 ticks more besides the 15 predefined ticks. When the turnaround is finished, the origin and destination variables are updated to the current position and one of the runway headers respectively. On the other hand, when the assigned destination of the aircraft is one of the runway headers, the aircraft takes-off and disappears of the scenario, on other words, the agent dies.

Figure 19: Switch button to add randomness to turnarounds

Furthermore, all the process is conflict free in the sense that aircraft does not collide. The observer agent knows the position of each aircraft at all times. At each step on the *go* procedure each aircraft checks if the next patch is available to continue its trajectory.

- When this patch is a cross-point and there is more than one aircraft waiting to enter, the first that runs the go procedure is the first to pass through.

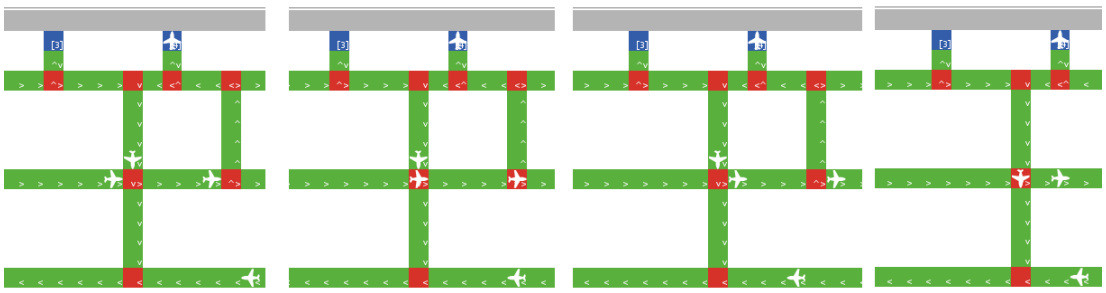

Figure 20: Avoiding collision on cross-points

- When the next step is entering into the runway, this must be empty, ergo there cannot be any aircraft in any of the patches that conforms the runway.

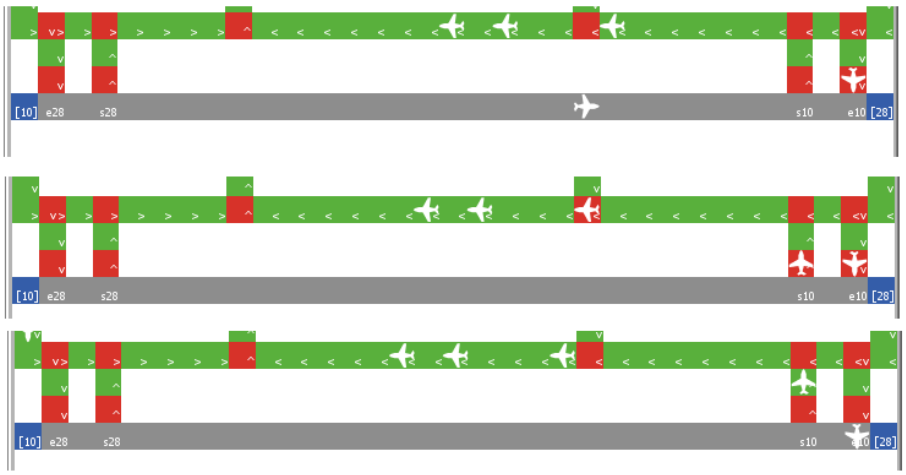

Figure 21: Checking runway availability

- When the destination is one of the gates and the next patch is the decision and cross-point that will drive the aircraft to its destination, this proves that there is no aircraft occupying the region corresponding to this gate. If another airplane is doing the turnaround

or the pushback to get out of there, it has preference to leave before the coming aircraft enters to avoid collapses.

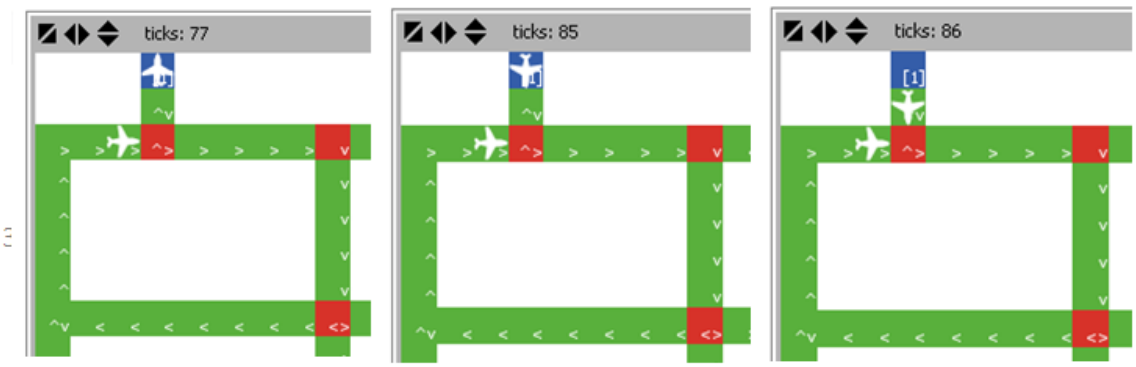

Figure 22: Checking gate availability

All these casuistic to avoid collision are controlled by the next function:

```
to move-fd [x]

  ask patch-at-heading-and-distance heading 1
  [
    if rw?
    [
      if abailable?
      [
        ask turtle x [fd 1]
        ask p-runway [set abailable? false]
      ]
    ]

    if g-crosspoint?
    [
      ask turtle x
      [
        if xcor = -14 and ycor = 14 and destination = [1]
        [
          ifelse (any? turtles-on patch -13 14 or any? turtles-on patch -13 15 or any? turtles-on patch -13 16)
          [
            if not stopped?
            [set n-stop-and-go n-stop-and-go + 1 set stopped? true set g-stopped? true set total-stop-go total-stop-go + 1]
          ]
          [
            ask turtle x [fd 1 set stopped? false set g-stopped? false]
          ]
        ]
```

```
    if not rw?
    [
      ifelse (any? turtles-on self)
      [ask turtle x
       [ if not stopped?

         [set n-stop-and-go n-stop-and-go + 1 set stopped? true set total-stop-go total-stop-go + 1]
       ]
      ]
      [  ask turtle x [if not g-stopped? [fd 1 set stopped? false]   ]
      ]
    ]
  ]

end
```

The switch button *traditional-first?* activates a function that priories in cross points aircraft from traditional airlines rather than low cost ones. Low cost aircraft are printed in yellow to differentiate them from the traditional ones. Also when an aircraft is stopped it turns black either because it is waiting to get through a cross point or it is stopped in a queue of aircraft.

Figure 23: Switch button to prioritize traditional flights

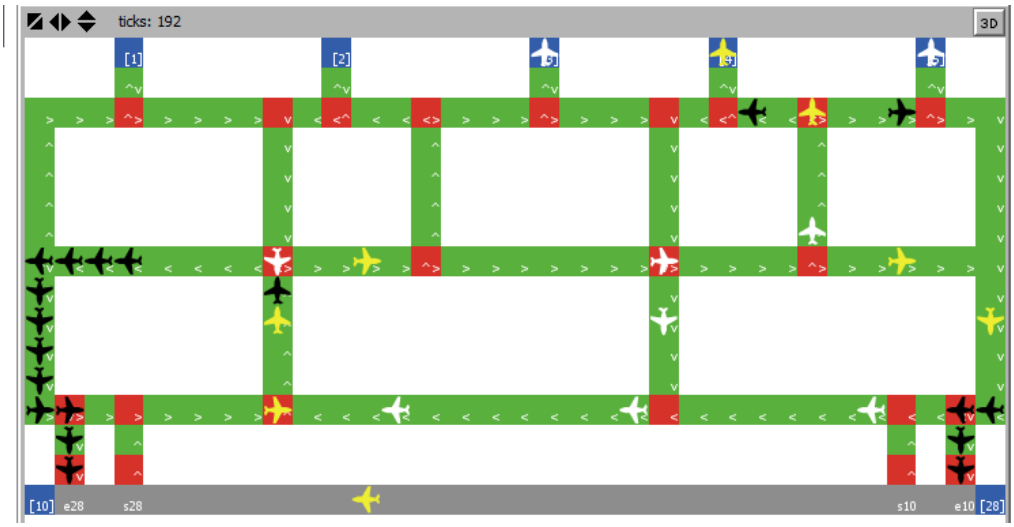

Figure 24: Simulation with traditional-first? button on

The implemented scenario demonstrates that safe autonomous taxiway management could be modelled and if hazard research is done to apply constrains and situations to achieve a real world simulation it could be a really powerful system to look for the optimization of taxiway management.

There are two main buttons to simulate the model, namely, the *setup* button and the *go* button. The *setup* button clear all the data, initialize the global variables, creates the scenario initializing all patches variables and creating all patches agents, create the initial turtle agents (one aircraft in each gate) and finally reset the number of ticks.

The following flowchart shows the algorithm implemented on the *go* procedure:

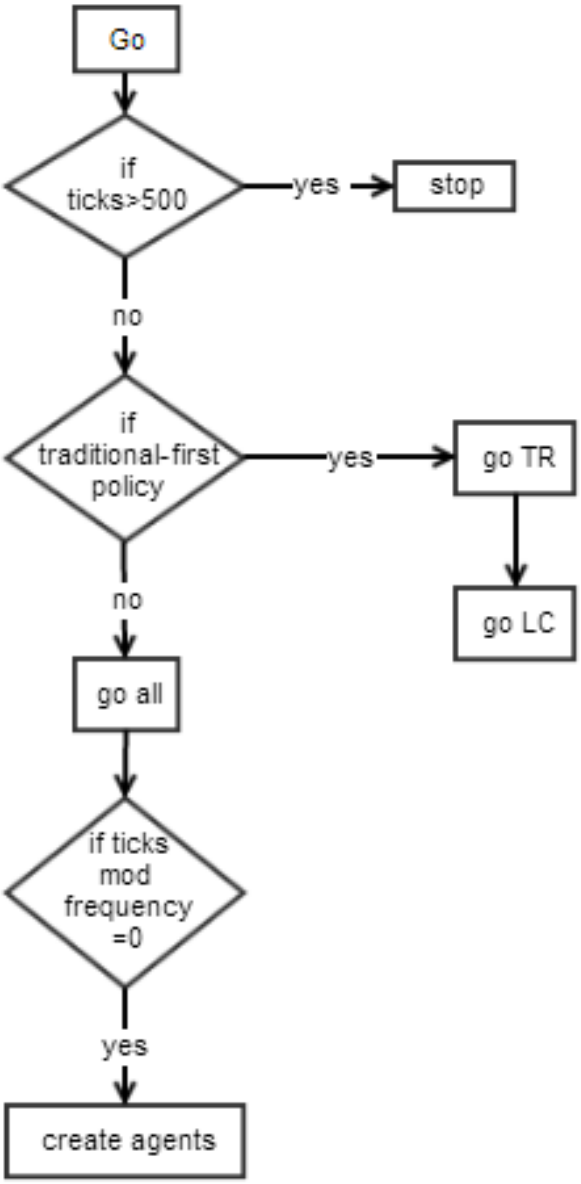

Figure 25: Go procedure

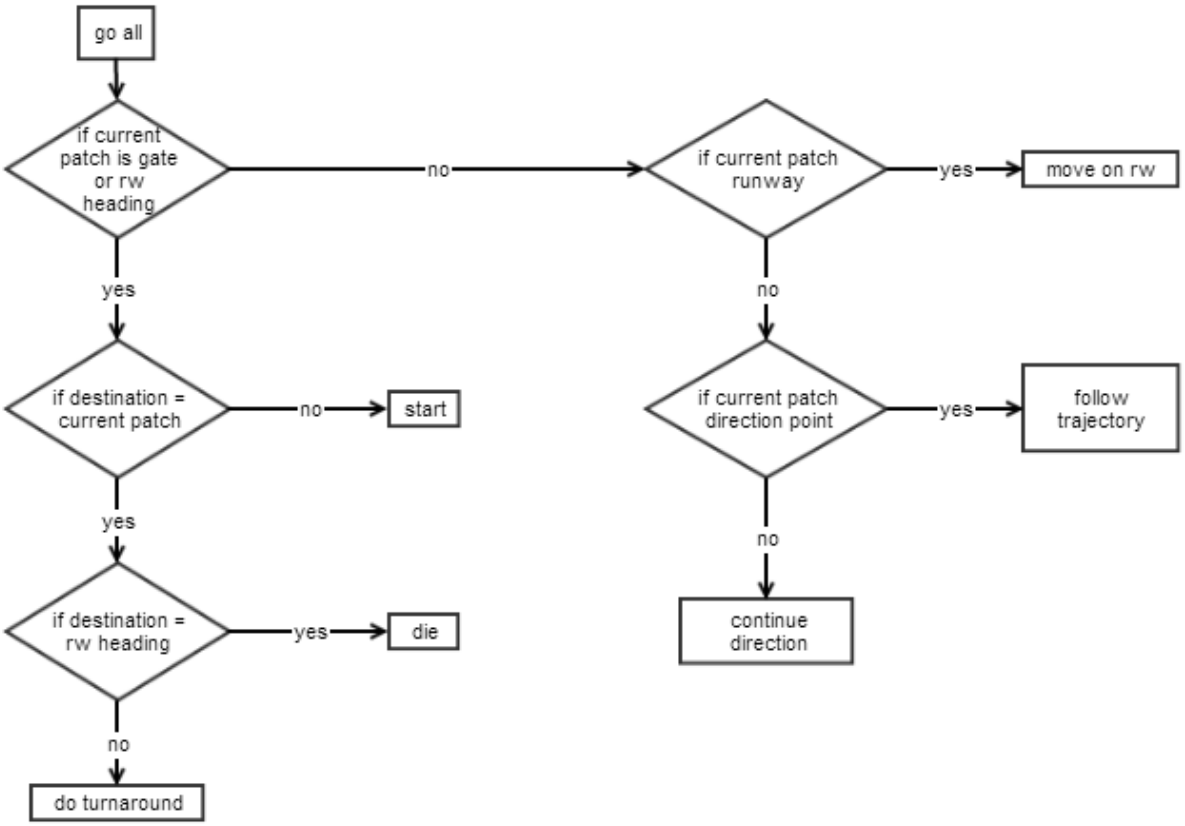

Figure 26: Go all procedure

Chapter 4

Obtained results

The model developed in this project has explored the introduction of Multi-Agent Systems to improve the fluidity of ground operations through the automation of taxiway management. It has been demonstrated that it is possible to model complex systems such as an airport platform as an agent-based system. Therefore, the introduction of MAS systems as a taxiway management tool is feasible.

This academic model simulates a simple hypothetical airport which verifies that MAS techniques allow communication between aircrafts and conflict free movements on the airport surface. Aircraft position is known by the system at every moment and aircrafts are able to know their positions with each other. Therefore, conflict detection in cross points is feasible without the need of ground controllers.

Furthermore, this model distinguishes between aircrafts pertaining to low cost companies and aircrafts of traditional airlines. This allows the application of policies to give priority to flights of traditional airlines in cross points in order to benefit this type of airlines that pay more airport fees.

However, this model does not include the implementation of delays when the aircraft is doing the turnaround with the engines off in order to decrease the number of stops and goes during its trajectory and consequently to reduce fuel costs and environmental impact.

To analyze output data from the model the *stop-simulation?* button is activated. This will allow a fair comparison as the model will always run the *go* procedure the same number of times, set to 500 ticks.

Here, data obtained from applying different policies and altering the values of variables will be analyzed.

Scenario 1

The first scenario runs the model without randomness applied to turnarounds and without distinguishing between low cost flights and traditional airline ones. With the predefined frequency it is seen that the number of total stops and goes are nearly an average of 4 stops and goes per flight. The plots show that when the scenario arrives reaches 14 aircraft moving on the platform the number of stops and goes exceeds the number of total flights.

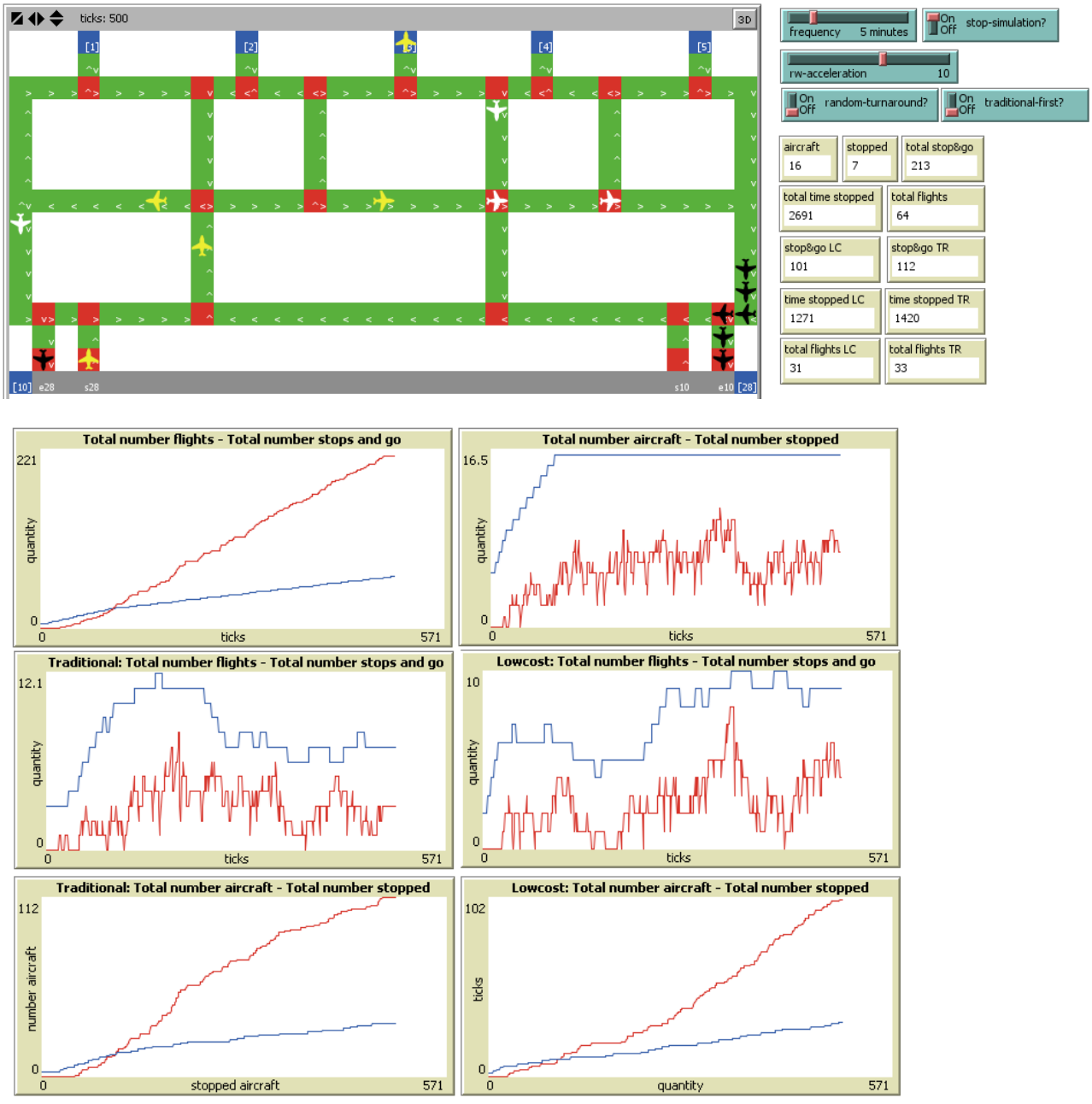

Figure 27: Obtained results Scenario 1

Scenario 2

The change made in this scenario is the application of randomness to turnarounds. With the predefined frequency it is seen that the number of total stops and goes are nearly an average of 3 stops and goes per flight. This amount has been reduced but this is because in this simulation the maximum number of simultaneous aircrafts per platform has been 14 instead of 16 due to the randomness applied on delays in turnarounds.

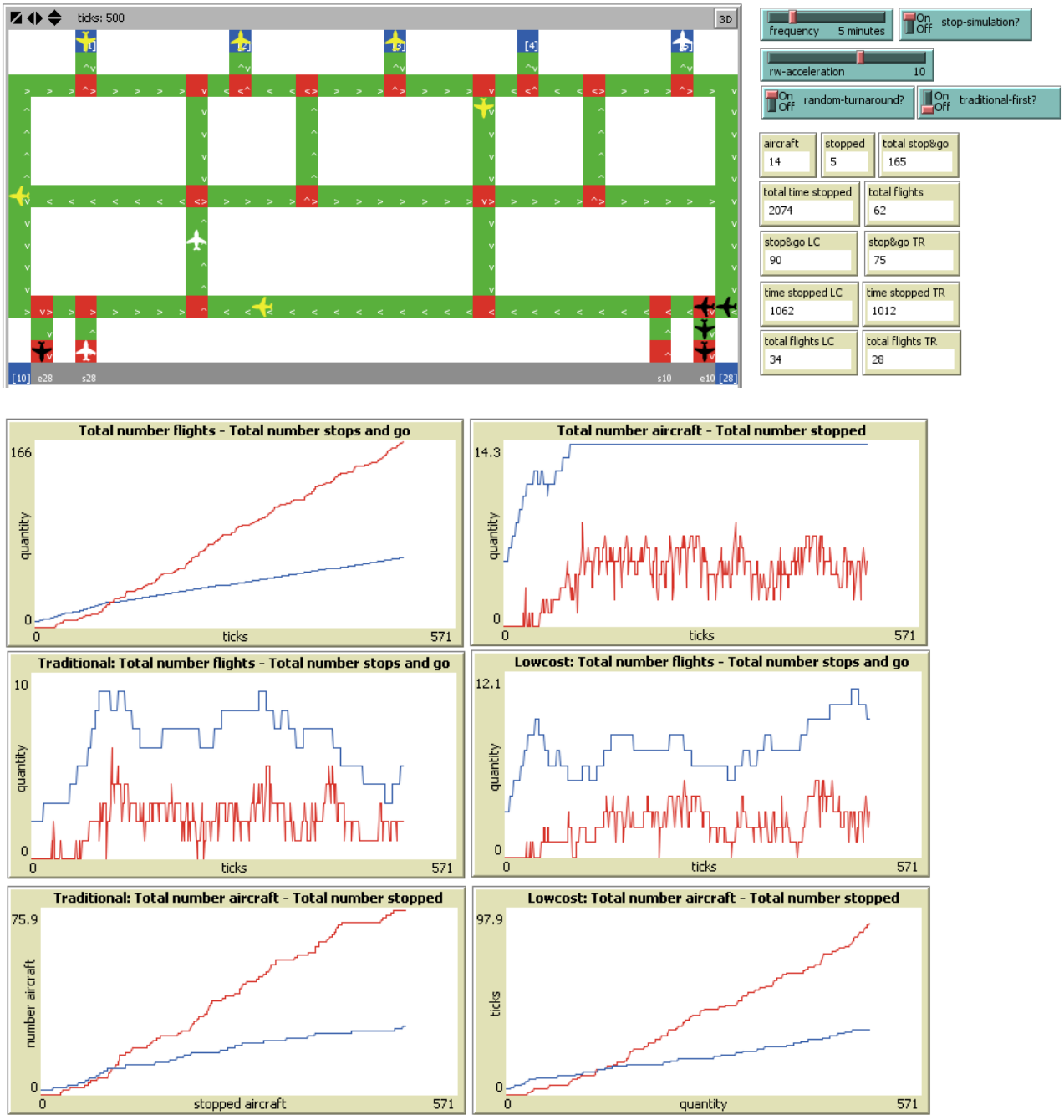

Figure 28: Obtained results Scenario 2

Scenario 3

This scenario applies the policy of preference based on the type of the company. The number of total stops and goes are nearly an average 3 stops and goes per flight, a little bit higher in average for low cost flights. It should be noted that the number of stops and goes does not only consider the stops at cross points but also considers the tail stops.

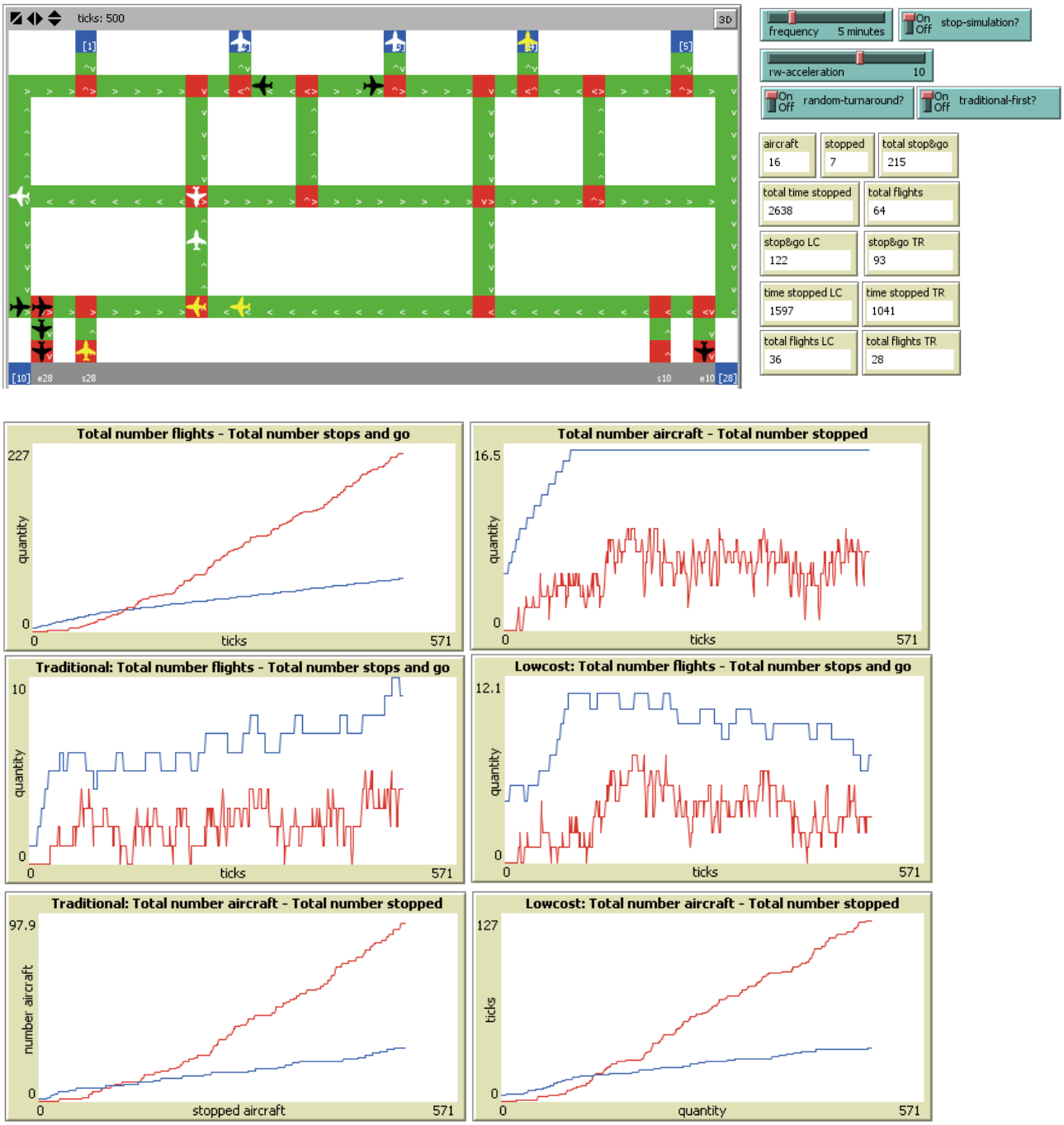

Figure 29: Obtained results Scenario 3

Scenario 4

This scenario retains the randomness and the priority policies but changes the frequency decreasing it by one unit. The results denote that this scenario does not support such a high frequency with this configuration since the number of stops and goes extremely increases and the airport surface platform is collapsed with long queues of aircrafts.

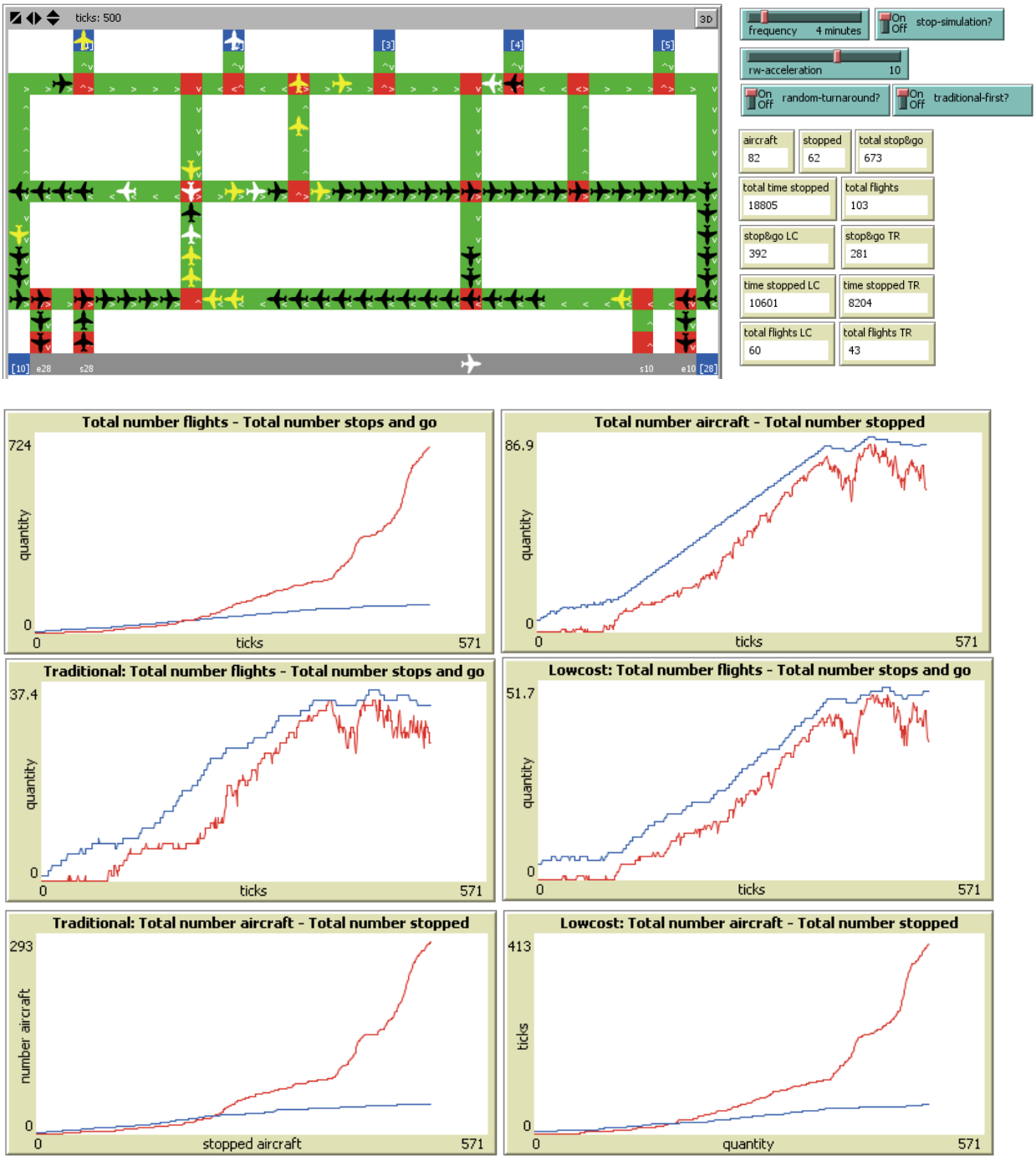

Figure 30: Obtained results Scenario 4

Scenario 5

This scenario retains the randomness and the priority policies but changes the frequency increasing it by one unit according to the predefined value. The average of stops and goes per aircraft is drastically reduced to near 1.3 although the number of simultaneous aircrafts in the platform has been the most elevated of all reaching the value of 18.

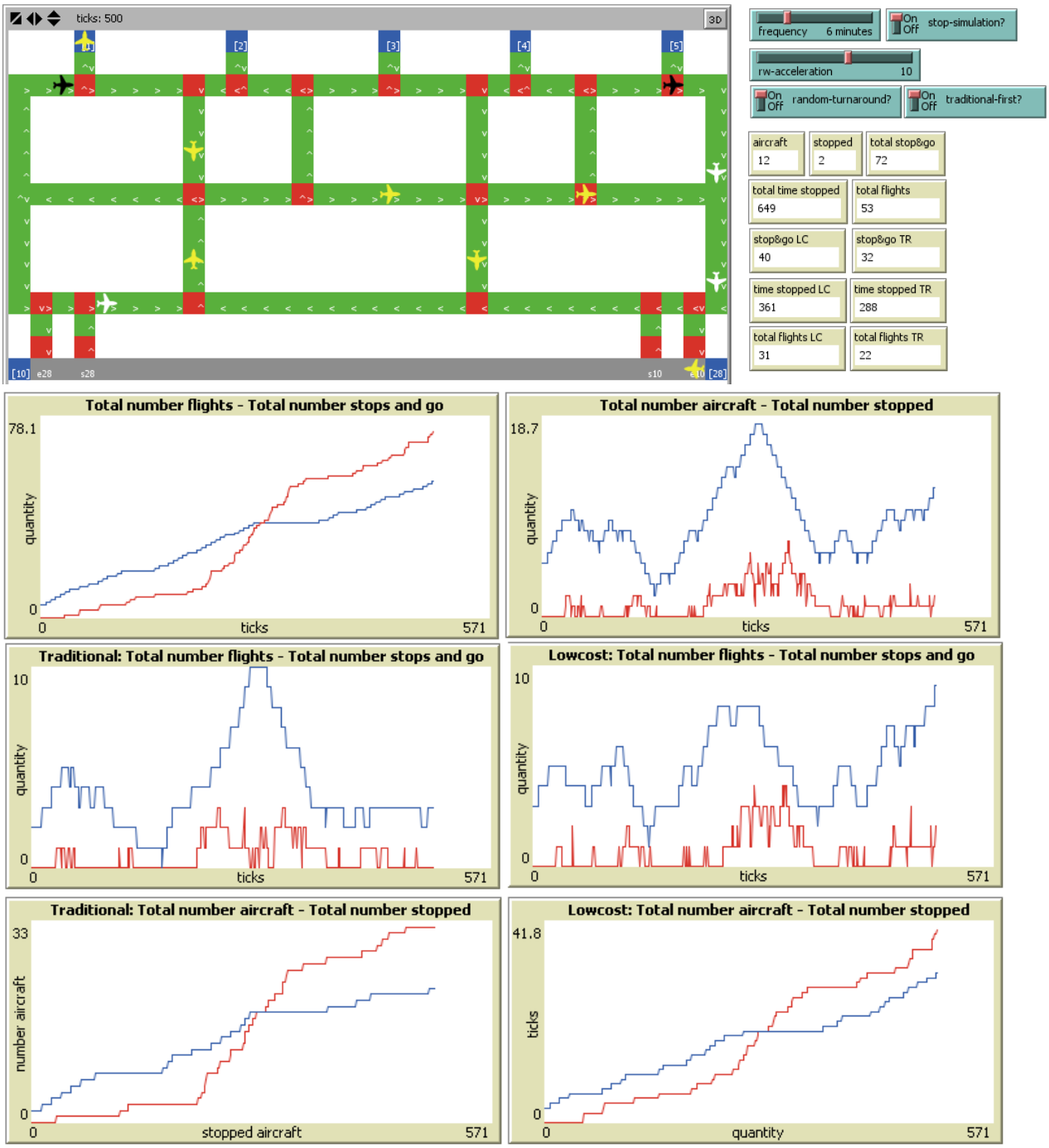

Figure 31: Obtained results Scenario 5

Chapter 5

Conclusion

5.1. Summary Results

- The developed model in this project corroborates that ground operations can be modelled and simulated as a Multi-Agent based system taking advantage of the benefits of MAS techniques.
- Communication between aircrafts in real time allows conflict free ground operations.
- Different policies can be added such as priority in cross-points depending on the type of the company.
- The implementation of MAS as a taxiway management tool is feasible to enhance the fluidity of ground movements in the airport platform.

5.2. Future Work

There are many areas of improvement to optimize this model and make it proper to reality.

The first point to note is the scenario, which in this case is a simple hypothetical airport where predefined trajectories have been established. Nevertheless, it is possible to model any scenario where the taxiway configuration optimizes the fluidity of ground operations. Moreover, trajectories from origin to destination do not have to be predefined if the primary purpose is to reduce the number of stops and goes. It is possible that an alternative route even longer in distance could be conflict free. This alternative must be considered to reduce costs.

Another important issue is the introduction of scheduled flights into the system and gate assignments. The current model has a lot of inefficiencies due

to the randomness on gates and runway header assignments. With scheduled flights incorporated into the system, it could be possible to optimise the resources. Furthermore, reassignment policies should be implemented when delays alter the scheduled planning.

Policies such as giving priority to arrival flights on a holding point could be implemented due to the elevate cost that supposes to have an airplane waiting in the air in comparison with having one waiting to take-off. Moreover, aircrafts should be able to anticipate the situation having information about the current state of other aircraft and the future actions expected. A central system, like the observer in NetLogo, should be able to get all the information and process it to determine whether or not it could be better to delay a specific flight.

On the other hand, to approach the model as much as possible to reality, this problem of taxiway management should be represented in a vertical hierarchical architecture:

Taxiways Supervisor Agent (TSA) has full information about the environment plan and state. This agent knows the configuration of taxiways and the position and the state of each aircraft situated on the scenario. The role of TSA is to look for the best path for each aircraft at each determinate moment.

Taxiways Management Agent (TMA) has information about the pre-defined flights scheduling that represents the static knowledge of the system and the information about delays and the current position of each aircraft. The role of TMA is to help each aircraft to decide whether to start or not their route through taxiways taking into account the estimated time arrival/departure (ETA/ETD) and the number of ‘stop-and-go’ that the aircraft will do.

Airline Company Agent (ACA) has information about aircrafts of their own company but also about aircrafts of other companies. The role of ACA is to negotiate with ACA of other companies in order to benefit aircrafts of their own company. Therefore, it is necessary to have knowledge about the pre-defined flight schedule of all their flights and the final result of the flights executed to compete with other agents taking into account the policies defined at a higher level about, such as equity policies.

Aircraft Agent (AA) has information about its own estimated time arrival/departure (ETA/ETD) and about its destination, whether it is a gate or a runway threshold. Its role consists on competing with other AA to achieve the best path in order to go to its destination minimizing the number of ‘stop-and-go’ according to its internal rules and the achievements of the other AA of their own airline company in terms of number of ‘stop-and-go’ and delays accumulated.

Cross-point Agent (CA) has information about the configuration of taxiways. Its role is to negotiate with ACA and AA to decide which agent has permission to pass through the cross-point in order to satisfy high-level goals imposed by TMA and TSA.

Appendix

```
extensions [matrix]

globals
[
number-of-gates-up      ;; total number of gates on the upper side of the runway
p-between-gates-up
list-gates              ;; names of gates
n                       ;; variable auxiliar nomenclar gates
set-origin
set-destination
time
heading-dp
total-stop-go           ;; total number of stops-and-go during the simulation
total-time-stopped
total-number-flights
total-lc
total-tr
created?
stop-go-lc
time-stop-lc
stop-go-tr
time-stop-tr


  ;;path trajectory matrix

m-trajectory
col

  ;;patch agentsets

p-crosspoints       ;; agentset containing the patches that are crosspoints
p-taxiways          ;; agentset containing the patches that are taxiways
p-taxiways-up       ;; direction up-down
p-taxiways-down     ;; direction down-up
p-taxiways-left     ;; direction left-right
p-taxiways-right    ;; direction right-left
p-runway            ;; agentset containing the patches that form the runway
p-gates             ;; agentset containing the patches that are gates or remote
position
p-entry-rw          ;; agentset containing the patches that are entry/exit of the
runway
  p-landing-takeoff  ;; agentset containing the patches that are for landing and
taking off from the rw
]

turtles-own
[
```

```
n-stop-and-go     ;; the number of stop-and-go that the agent has done
lowcost?          ;; true if the aircraft is heavy false if it is medium or light
origin            ;; from
destination       ;; to
n-dec-point       ;; number of decision points passed trough
turnaround        ;; time of the turnaround
stopped?          ;; true if the airplane is doing an stop and go
time-stopped
g-stopped?

]

patches-own
[
left?             ;; true if the path goes in this direction
right?            ;; true if the path goes in this direction
up?               ;; true if the path goes in this direction
down?             ;; true if the path goes in this direction
direction-point?  ;; true if there are two or more possibilities

crosspoint?       ;; true if the patch is at the intersection of two roads
abailable?        ;; true if the green light is above the intersection.  otherwise,
false. false for a non-intersection patches.
rw?
g-crosspoint?
]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to setup
  ca                ;; borramos el estado actual
setup-globals       ;; inicialize global variables
setup-patches       ;; First we ask the patches to draw themselves and set up a
few variables
create-agents
reset-ticks         ;; reseteamos el numero de ticks a 0
end

to setup-globals
set number-of-gates-up 5
set p-between-gates-up (world-width / number-of-gates-up)
set list-gates [1 2 3 4 5 10 28]
set time 0
set col 0
set heading-dp 0
set total-stop-go 0
set total-time-stopped 0
set total-number-flights 0
```

```
set total-lc 0
set total-tr 0
set created? false
set stop-go-lc 0
set time-stop-lc 0
set stop-go-tr 0
set time-stop-tr 0
setup-trajectory-matrix

end

to setup-patches
ask patches                ;; initialize the patch-owned variables and color of the
patches
[ set crosspoint? false
set abailable? true
set pcolor white
set up? false
set down? false
set left? false
set right? false
set direction-point? false
set rw? false
set g-crosspoint? false]

setup-runway
setup-gates
setup-landing-takeoff
setup-taxiways
setup-ent-rw
setup-crosspoints
setup-taxiways-directions

end


to setup-trajectory-matrix
;; 0 forward 1 right 2 left
set m-trajectory matrix:from-row-list [

[0  0  0  0  0  0  0  0  0  0  1  2  2  2  2  1  2  2  2  2]
[1  1  2  2  1  1  2  2  1  1  2  1  1  0  0  2  1  1  0  0]
[2  1  2  1  0  1  0  1  0  1  1  1  2  0  0  1  1  2  0  0]
[1  0  1  0  1  0  1  0  1  -1  0  2  1  1  1  0  2  1  1  1]
[2  0  2  0  1  1  1  1  1  -1  -1  0  0  1  2  -1  0  0  1  2]
[-1  0  -1  0  2  -1  2  -1  2  -1  -1  -1  -1  2  1  -1  -1  -1  2  1]
[-1  1  -1  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  0  0  -1  -1  -1  0  0]

]
```

```
end

to create-agents
ifelse time = 0
  [
crt 5 [
setup-agent
setup-location
    ]
set total-number-flights 5
  ]
  [
ifelse (not all? p-gates [any? turtles-on self]) and
     (time< 100 and (time > (frequency - 1) and time mod (frequency * 2) = 0))
     [
if ((not any? turtles-on patch -13 14 and not any? turtles-on patch -13 15 and
not any? turtles-on patch -13 16) or
       (not any? turtles-on patch -6 14 and not any? turtles-on patch -6 15 and not
any? turtles-on patch -6 16) or
       (not any? turtles-on patch 1 14 and not any? turtles-on patch 1 15 and not
any? turtles-on patch 1 16) or
       (not any? turtles-on patch 7 14 and not any? turtles-on patch 7 15 and not
any? turtles-on patch 7 16) or
       (not any? turtles-on patch 14 14 and not any? turtles-on patch 14 15 and
not any? turtles-on patch 14 16))


       [
crt 1 [setup-agent
setup-location
set total-number-flights total-number-flights + 1]
       ]

     ]

     [
if not any? turtles-on p-runway and (not any? turtles-on patch 13 2 or not any?
turtles-on patch -13 2)
[ crt 1 [setup-agent
setup-location
set total-number-flights total-number-flights + 1
         ]
       ]
     ]

   ]

set-default-shape turtles "airplane"
end
```

```
to setup-matrix
;set m matrix:make-constant 1 1 0

end

to setup-agent
set n-stop-and-go 0
set n-dec-point 0
ifelse random-turnaround?
  [
ifelse random 2 = 0
    [set turnaround 15]
    [set turnaround 15 + random 16]
  ]
  [set turnaround 15]

set stopped? false
set g-stopped? false
ifelse random 2 = 0
  [set lowcost? false set color yellow set total-tr total-tr + 1]
  [set lowcost? true set color white set total-lc total-lc + 1]
;set color white

end

to setup-location

ifelse (time < 100 and (time > (frequency - 1) and time mod (frequency * 2) = 0))
or time = 0
 [
if (not any? turtles-on patch -13 14 and not any? turtles-on patch -13 15 and not
any? turtles-on patch -13 16)
       [
move-to patch -13 16
ask patch-at 0 0 [set set-origin plabel]
set origin set-origin
set heading 180
ask one-of p-landing-takeoff [set set-destination plabel]
set destination set-destination
set created? true
       ]
if (not any? turtles-on patch -6 14 and not any? turtles-on patch -6 15 and not
any? turtles-on patch -6 16)
       [
move-to patch -6 16
ask patch-at 0 0 [set set-origin plabel]
set origin set-origin
set heading 180
ask one-of p-landing-takeoff [set set-destination plabel]
set destination set-destination
```

```
set created? true
      ]
if (not any? turtles-on patch 1 14 and not any? turtles-on patch 1 15 and not
any? turtles-on patch 1 16)
      [
move-to patch 1 16
ask patch-at 0 0 [set set-origin plabel]
set origin set-origin
set heading 180
ask one-of p-landing-takeoff [set set-destination plabel]
set destination set-destination
set created? true
      ]
if (not any? turtles-on patch 7 14 and not any? turtles-on patch 7 15 and not
any? turtles-on patch 7 16)
      [
move-to patch 7 16
ask patch-at 0 0 [set set-origin plabel]
set origin set-origin
set heading 180
ask one-of p-landing-takeoff [set set-destination plabel]
set destination set-destination
set created? true
      ]
if (not any? turtles-on patch 14 14 and not any? turtles-on patch 14 15 and not
any? turtles-on patch 14 16)
      [
move-to patch 14 16
ask patch-at 0 0 [set set-origin plabel]
set origin set-origin
set heading 180
ask one-of p-landing-takeoff [set set-destination plabel]
set destination set-destination
set created? true
      ]
 ]
 [
ifelse not any? turtles-on patch -13 2
[ move-to patch 16 1
ask patch-at 0 0 [set set-origin plabel]
set origin set-origin
ifelse set-origin = [10] [set heading 90][set heading 270]
ask one-of p-gates [set set-destination plabel]
set destination set-destination
  ]

[ move-to patch -16 1
ask patch-at 0 0 [set set-origin plabel]
set origin set-origin
ifelse set-origin = [10] [set heading 90][set heading 270]
```

```
ask one-of p-gates [set set-destination plabel]
set destination set-destination
   ]

 ]

end


.....................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; PROCESS ;;;;;;;;;;;;;
.....................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


to go

if stop-simulation?[if ticks >= 500 [stop]]

set time ticks

ifelse any? turtles-on p-runway [ask p-runway [set abailable? false]][ask p-
runway [set abailable? true]]  ;compruebo si la rw esta ocupada o no

ifelse traditional-first?
[ go-tr  go-lc ]
 [go-all]

if (ticks > (frequency - 1) and ticks mod frequency = 0) [create-agents]
tick

end

to go-all
ask turtles  ; para todas las tortugas en el escenario
 [
ask patch-at 0 0 ; i en el patch en el que se encuentran
   [
ifelse (plabel = [28] or plabel = [1] or plabel = [2] or plabel = [3] or plabel = [4] or
plabel = [5] or plabel = [10])      ;if estan en algun gate o cabecera de pista
[
ask turtles-here
       [
ifelse (destination = plabel)
[
ifelse (destination = [28] or destination = [10]) ; si han llegado a su destino en la
pista
          [die] ; mueren, desaparecen porque han despegado
          [do-turnaround] ;sino,  han llegado al gate de destino pero hay que
restablecer valores y hacer que esperen el tiempo minimo de embarque
         ]
```

```
        ;;else se encuentran en el origen algun gate o cabecera listos para salir
        [start]      ;; AÑADIR CODIGO DELAY O NO????
       ]
      ]

    ;;else   no estan en gate o cabecera de pista     --> on-route
[
ifelse (rw?)   ;; estan on-route i estan en el rw
      [
ask turtles-here
[ move-on-rw ]
      ]
;; else on-route i no estan en el rw
[
ask turtles-here
        [
ifelse (direction-point?)  ;;si esta en ruta i esta en un direction point    ;; va a
buscar los parametros correspondientes a la matriz de trayectorias

          [follow-trajectory]
          ;;else esta en ruta pero no esta en un direction point
[
ifelse ((xcor = 16 or xcor = -16) and ycor = 4)
[ if xcor = 16[set heading 270 move-fd who]
if xcor = -16[set heading 90 move-fd who]]

            [continue-direction]

            ]

         ]
       ]
    ]
  ]



ifelse stopped?

 [set total-time-stopped total-time-stopped + 1 set time-stopped time-stopped +
1 set color black ifelse lowcost? [set time-stop-lc time-stop-lc + 1][set time-stop-
tr time-stop-tr + 1]]

 [ifelse lowcost? [set color yellow][set color white]]
]
end

to go-lc
ask turtles with [lowcost?] ; para todas las tortugas en el escenario
 [
```

```
ask patch-at 0 0 ; i en el patch en el que se encuentran
   [
ifelse (plabel = [28] or plabel = [1] or plabel = [2] or plabel = [3] or plabel = [4] or
plabel = [5] or plabel = [10])      ;if estan en algun gate o cabecera de pista
[
ask turtles-here
      [
ifelse (destination = plabel)
[
ifelse (destination = [28] or destination = [10]) ; si han llegado a su destino en la
pista
         [die] ; mueren, desaparecen porque han despegado
         [do-turnaround] ;sino,  han llegado al gate de destino pero hay que
restablecer valores y hacer que esperen el tiempo minimo de embarque
        ]
        ;;else se encuentran en el origen algun gate o cabecera listos para salir
        [start]      ;; AÑADIR CODIGO DELAY O NO????
       ]
     ]

    ;;else   no estan en gate o cabecera de pista     --> on-route
[
ifelse (rw?)   ;; estan on-route i estan en el rw
      [
ask turtles-here
[ move-on-rw ]
      ]
;; else on-route i no estan en el rw
[
ask turtles-here
        [
ifelse (direction-point?)  ;;si esta en ruta i esta en un direction point    ;; va a
buscar los parametros correspondientes a la matriz de trayectorias

          [follow-trajectory]
          ;;else esta en ruta pero no esta en un direction point
[
ifelse ((xcor = 16 or xcor = -16) and ycor = 4)
[ if xcor = 16[set heading 270 move-fd who]
if xcor = -16[set heading 90 move-fd who]]

            [continue-direction]

            ]

         ]
       ]
    ]
  ]
```

```
ifelse stopped?

 [set total-time-stopped total-time-stopped + 1 set time-stopped time-stopped +
1 set color black ifelse lowcost? [set time-stop-lc time-stop-lc + 1][set time-stop-
tr time-stop-tr + 1]]

 [ifelse lowcost? [set color yellow][set color white]]

; ifelse stopped? [set total-time-stopped total-time-stopped + 1 set time-stopped
time-stopped + 1 set color black] [ifelse lowcost? [set color yellow set time-stop-
lc time-stop-lc + 1][set color white set time-stop-tr time-stop-tr + 1]]
]
end

to go-tr
ask turtles with [not lowcost?]  ; para todas las tortugas en el escenario
 [
ask patch-at 0 0 ; i en el patch en el que se encuentran
   [
ifelse (plabel = [28] or plabel = [1] or plabel = [2] or plabel = [3] or plabel = [4] or
plabel = [5] or plabel = [10])      ;if estan en algun gate o cabecera de pista
[
ask turtles-here
       [
ifelse (destination = plabel)
[
ifelse (destination = [28] or destination = [10]) ; si han llegado a su destino en la
pista
          [die] ; mueren, desaparecen porque han despegado
          [do-turnaround] ;sino,  han llegado al gate de destino pero hay que
restablecer valores y hacer que esperen el tiempo minimo de embarque
         ]
         ;;else se encuentran en el origen algun gate o cabecera listos para salir
         [start]      ;; AÑADIR CODIGO DELAY O NO????
        ]
      ]

     ;;else   no estan en gate o cabecera de pista     --> on-route
[
ifelse (rw?)   ;; estan on-route i estan en el rw
       [
ask turtles-here
[ move-on-rw ]
       ]
;; else on-route i no estan en el rw
[
ask turtles-here
         [
```

```
ifelse (direction-point?)  ;;si esta en ruta i esta en un direction point    ;; va a
buscar los parametros correspondientes a la matriz de trayectorias

          [follow-trajectory]
          ;;else esta en ruta pero no esta en un direction point
[
ifelse ((xcor = 16 or xcor = -16) and ycor = 4)
[ if xcor = 16[set heading 270 move-fd who]
if xcor = -16[set heading 90 move-fd who]]

            [continue-direction]

            ]

         ]
       ]
    ]
  ]

ifelse stopped?

 [set total-time-stopped total-time-stopped + 1 set time-stopped time-stopped +
1 set color black ifelse lowcost? [set time-stop-lc time-stop-lc + 1][set time-stop-
tr time-stop-tr + 1]]

 [ifelse lowcost? [set color yellow][set color white]]
]
end



to start

fd 1

end

to do-turnaround

set turnaround turnaround - 1
if turnaround = 0
  [
set heading 180
set turnaround 10
set n-stop-and-go 0
set n-dec-point 0
set origin plabel
ifelse random 2 = 0
    [set destination [10]]
    [set destination [28]]
```

```
  ]
end

to move-on-rw
repeat rw-acceleration [
if (destination = [28] and heading = 180) [set heading 90]
if (destination = [10] and heading = 180) [set heading 270]
if (origin = [10] and plabel = "s10") [set heading 0]
if (origin = [28] and plabel = "s28") [set heading 0]
ask patch-at 0 0 [if rw? [ ask turtles-here [fd 1]]]
  ]
end

to look-direction [row column]

if(( matrix:get m-trajectory row column) = 0) [set heading-dp heading]
if(( matrix:get m-trajectory row column) = 1) [set heading-dp heading - 90 ]
if(( matrix:get m-trajectory row column) = 2) [set heading-dp heading + 90 ]

end

to follow-trajectory

if (origin = [1] and destination = [28]) [look-direction n-dec-point 0]
if (origin = [1] and destination = [10]) [look-direction n-dec-point 1]
if (origin = [2] and destination = [28]) [look-direction n-dec-point 2]
if (origin = [2] and destination = [10]) [look-direction n-dec-point 3]
if (origin = [3] and destination = [28]) [look-direction n-dec-point 4]
if (origin = [3] and destination = [10]) [look-direction n-dec-point 5]
if (origin = [4] and destination = [28]) [look-direction n-dec-point 6]
if (origin = [4] and destination = [10]) [look-direction n-dec-point 7]
if (origin = [5] and destination = [28]) [look-direction n-dec-point 8]
if (origin = [5] and destination = [10]) [look-direction n-dec-point 9]
if (origin = [28] and destination = [1]) [look-direction n-dec-point 10]
if (origin = [28] and destination = [2]) [look-direction n-dec-point 11]
if (origin = [28] and destination = [3]) [look-direction n-dec-point 12]
if (origin = [28] and destination = [4]) [look-direction n-dec-point 13]
if (origin = [28] and destination = [5]) [look-direction n-dec-point 14]
if (origin = [10] and destination = [1]) [look-direction n-dec-point 15]
if (origin = [10] and destination = [2]) [look-direction n-dec-point 16]
if (origin = [10] and destination = [3]) [look-direction n-dec-point 17]
if (origin = [10] and destination = [4]) [look-direction n-dec-point 18]
if (origin = [10] and destination = [5]) [look-direction n-dec-point 19]

;; ahora tengo en heading-dp la direccion en la que ira el avion en este direction
point

look-stopped-dp who
```

```
look-m-trajectory

end

to look-stopped-dp [x]
ask patch-at-heading-and-distance heading-dp 1  ;; miramos el siguiente patch
donde debe moverse el avion
[
ifelse (any? turtles-on self) ; si el siguiente patch no se trata de un rw i tiene
algun otro avion ocupandolo
[ask turtle x
[ if not stopped?         ; miramos que el avion que quiere moverse no este
haciendo un stop-and-go

        [set n-stop-and-go n-stop-and-go + 1 set stopped? true set total-stop-go
total-stop-go + 1 ifelse lowcost? [set stop-go-lc stop-go-lc + 1][set stop-go-tr
stop-go-tr + 1]]  ; si no lo esta haciendo, lo ponemos en parado porque no
puede avanzar y sumamos uno al numero de stops&go
]
     ]
[  ask turtle x [ set stopped? false]
] ; si el siguiente patch no es un rw i no esta ocupado avanzamos una posicion i
ponemos stoped en falso
]
end

to look-m-trajectory

if not stopped?
  [
if (origin = [1] and destination = [28]) [decide-direction n-dec-point 0]
if (origin = [1] and destination = [10]) [decide-direction n-dec-point 1]
if (origin = [2] and destination = [28]) [decide-direction n-dec-point 2]
if (origin = [2] and destination = [10]) [decide-direction n-dec-point 3]
if (origin = [3] and destination = [28]) [decide-direction n-dec-point 4]
if (origin = [3] and destination = [10]) [decide-direction n-dec-point 5]
if (origin = [4] and destination = [28]) [decide-direction n-dec-point 6]
if (origin = [4] and destination = [10]) [decide-direction n-dec-point 7]
if (origin = [5] and destination = [28]) [decide-direction n-dec-point 8]
if (origin = [5] and destination = [10]) [decide-direction n-dec-point 9]
if (origin = [28] and destination = [1]) [decide-direction n-dec-point 10]
if (origin = [28] and destination = [2]) [decide-direction n-dec-point 11]
if (origin = [28] and destination = [3]) [decide-direction n-dec-point 12]
if (origin = [28] and destination = [4]) [decide-direction n-dec-point 13]
if (origin = [28] and destination = [5]) [decide-direction n-dec-point 14]
if (origin = [10] and destination = [1]) [decide-direction n-dec-point 15]
if (origin = [10] and destination = [2]) [decide-direction n-dec-point 16]
if (origin = [10] and destination = [3]) [decide-direction n-dec-point 17]
if (origin = [10] and destination = [4]) [decide-direction n-dec-point 18]
if (origin = [10] and destination = [5]) [decide-direction n-dec-point 19]
```

```
set n-dec-point n-dec-point + 1
  ]
end


to decide-direction [row column]

if(( matrix:get m-trajectory row column) = 0) [move-fd who]
if(( matrix:get m-trajectory row column) = 1) [set heading heading - 90   move-fd
who]
if(( matrix:get m-trajectory row column) = 2) [set heading heading + 90   move-fd
who]

end


to continue-direction

ifelse ((heading = 270 and left?) or (heading = 90 and right?) or (heading = 0
and up?) or (heading = 180 and down?)) [move-fd who]
;else
  [
if (heading != 270 and left?) [set heading 270 move-fd who]
if (heading != 90 and right?) [set heading 90  move-fd who]
if (heading != 0 and up?)     [set heading 0   move-fd who]
if (heading != 180 and down?) [set heading 180 move-fd who]
  ]

end

to move-fd [x]

ask patch-at-heading-and-distance heading 1  ;; miramos el siguiente patch
donde debe moverse el avion
[
if rw? ; si corresponde al rw
    [
ifelse abailable? ; entonces miramos si esta disponible para poder entrar
      [
ask turtle x [fd 1 set stopped? false]                    ; si esta disponible entramos
ask p-runway [set abailable? false]       ; y marcamos que ya no esta disponible
porque acabamos de entrar
]
      [
ask turtle x [set stopped? true]
      ]
    ]

if g-crosspoint?
    [
```

```
ask turtle x
     [
if xcor = -14 and ycor = 14 and destination = [1]
       [
ifelse (any? turtles-on patch -13 14 or any? turtles-on patch -13 15 or any?
turtles-on patch -13 16)
[
if not stopped?         ; miramos que el avion que quiere moverse no este
haciendo un stop-and-go
[set n-stop-and-go n-stop-and-go + 1 set stopped? true set g-stopped? true set
total-stop-go total-stop-go + 1 ifelse lowcost? [set stop-go-lc stop-go-lc + 1][set
stop-go-tr stop-go-tr + 1]]  ; si no lo esta haciendo, lo ponemos en parado
porque no puede avanzar y sumamos uno al numero de stops&go
]
        [
ask turtle x [fd 1 set stopped? false set g-stopped? false]
        ]
       ]
if xcor = -5 and ycor = 14 and destination = [2]
       [
ifelse (any? turtles-on patch -6 14 or any? turtles-on patch -6 15 or any? turtles-
on patch -6 16)
[
if not stopped?         ; miramos que el avion que quiere moverse no este
haciendo un stop-and-go
[set n-stop-and-go n-stop-and-go + 1 set stopped? true set g-stopped? true set
total-stop-go total-stop-go + 1 ifelse lowcost? [set stop-go-lc stop-go-lc + 1][set
stop-go-tr stop-go-tr + 1]]  ; si no lo esta haciendo, lo ponemos en parado
porque no puede avanzar y sumamos uno al numero de stops&go
]
        [
ask turtle x [fd 1 set stopped? false set g-stopped? false]
        ]
       ]
if xcor = 0 and ycor = 14 and destination = [3]
       [
ifelse (any? turtles-on patch 1 14 or any? turtles-on patch 1 15 or any? turtles-
on patch 1 16)
[
if not stopped?         ; miramos que el avion que quiere moverse no este
haciendo un stop-and-go
[set n-stop-and-go n-stop-and-go + 1 set stopped? true set g-stopped? true set
total-stop-go total-stop-go + 1 ifelse lowcost? [set stop-go-lc stop-go-lc + 1][set
stop-go-tr stop-go-tr + 1]]  ; si no lo esta haciendo, lo ponemos en parado
porque no puede avanzar y sumamos uno al numero de stops&go
]
        [
ask turtle x [fd 1 set stopped? false set g-stopped? false]
        ]
       ]
```

```
if xcor = 8 and ycor = 14 and destination = [4]
       [
ifelse (any? turtles-on patch 7 14 or any? turtles-on patch 7 15 or any? turtles-
on patch 7 16)
[
if not stopped?         ; miramos que el avion que quiere moverse no este
haciendo un stop-and-go
[set n-stop-and-go n-stop-and-go + 1 set stopped? true set g-stopped? true set
total-stop-go total-stop-go + 1 ifelse lowcost? [set stop-go-lc stop-go-lc + 1][set
stop-go-tr stop-go-tr + 1]]  ; si no lo esta haciendo, lo ponemos en parado
porque no puede avanzar y sumamos uno al numero de stops&go
]
        [
ask turtle x [fd 1 set stopped? false set g-stopped? false]
        ]
       ]
if xcor = 13 and ycor = 14 and destination = [5]
       [
ifelse (any? turtles-on patch 14 14 or any? turtles-on patch 14 15 or any?
turtles-on patch 14 16)
[
if not stopped?         ; miramos que el avion que quiere moverse no este
haciendo un stop-and-go
[set n-stop-and-go n-stop-and-go + 1 set stopped? true set g-stopped? true set
total-stop-go total-stop-go + 1 ifelse lowcost? [set stop-go-lc stop-go-lc + 1][set
stop-go-tr stop-go-tr + 1]]  ; si no lo esta haciendo, lo ponemos en parado
porque no puede avanzar y sumamos uno al numero de stops&go
]
        [
ask turtle x [fd 1 set stopped? false set g-stopped? false]
]
       ]
     ]
   ]

if not rw?
   [
ifelse (any? turtles-on self) ; si el siguiente patch no se trata de un rw i tiene
algun otro avion ocupandolo
[ask turtle x
[ if not stopped?         ; miramos que el avion que quiere moverse no este
haciendo un stop-and-go

        [set n-stop-and-go n-stop-and-go + 1 set stopped? true set total-stop-go
total-stop-go + 1 ifelse lowcost? [set stop-go-lc stop-go-lc + 1][set stop-go-tr
stop-go-tr + 1]]  ; si no lo esta haciendo, lo ponemos en parado porque no
puede avanzar y sumamos uno al numero de stops&go
]
     ]
[  ask turtle x [if not g-stopped? [fd 1 set stopped? false]   ]
```

```
     ] ; si el siguiente patch no es un rw i no esta ocupado avanzamos una
posicion i ponemos stoped en falso
]
  ]

end




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;   PATCHES  ;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



  ;;;;;;;;  RUNWAY  ;;;;;;;;

to setup-runway
set p-runway patches with [pycor = 1]
ask p-runway
[ set pcolor gray
set rw? true
if pxcor = 13 [set plabel "s10"]
if pxcor = -13 [set plabel "s28"]
if pxcor = 15 [set plabel "e10"]
if pxcor = -15 [set plabel "e28"]
   ]

end

  ;;;;;;;;  GATES  ;;;;;;;;

to setup-gates
set p-gates patches with
  [((floor((pxcor + max-pxcor - floor(p-between-gates-up - 3)) mod p-between-
gates-up) = 0) and (pycor = max-pycor))]
ask p-gates [set pcolor blue ]
set n 1
foreach sort p-gates [
ask ? [
set plabel filter [? = n] list-gates
set n n + 1
     ]
   ]
end

  ;;;;;;;;  TAXIWAYS  ;;;;;;;;

to setup-taxiways
```

```
set p-taxiways patches with
[ (pycor = 4) or (pycor = 9) or (pycor = 14) or
;; horizontal lines
    (pxcor = min-pxcor and pycor < 14 and pycor > 4) or
;; vertical lines left & right
    (pxcor = max-pxcor and pycor < 14 and pycor > 4) or
    ((floor((pxcor + max-pxcor - floor(p-between-gates-up - 3)) mod p-between-
gates-up) = 0)    and pycor > 14 and pycor < max-pycor) or         ;; from gates to
taxiway
    (pxcor = -15 and (pycor = 3 or pycor = 2)) or
;; entrance exit runway
    (pxcor = -13 and (pycor = 3 or pycor = 2)) or
    (pxcor = 15 and (pycor = 3 or pycor = 2)) or
    (pxcor = 13 and (pycor = 3 or pycor = 2)) or
    ((floor((pxcor + max-pxcor - 2 - floor(p-between-gates-up * 3)) mod p-
between-gates-up * 3) = 0) and (pycor < 14 and pycor > 4)) or           ;; between
taxiways
    (pxcor = 10 and pycor < 14 and pycor > 9) or
    (pxcor = -3 and pycor < 14 and pycor > 9) ]
ask p-taxiways [set pcolor green]
end

to setup-taxiways-directions
set p-taxiways-down p-taxiways with
[ (pycor < 10 and pycor > 4 and pxcor = min-pxcor) or
    (pycor< 15 and pycor > 4 and pxcor = 5) or
    (pycor< 15 and pycor > 4 and pxcor = max-pxcor) or
    (pycor< 15 and pycor > 9 and pxcor = -8) or
    (pycor< max-pycor  and pycor > 14 and pxcor = -13) or
    (pycor< max-pycor  and pycor > 14 and pxcor = -6) or
    (pycor< max-pycor  and pycor > 14 and pxcor = 1) or
    (pycor< max-pycor  and pycor > 14 and pxcor = 7) or
    (pycor< max-pycor  and pycor > 14 and pxcor = 14) or
    (pycor< 5  and pycor > 1 and pxcor = 15) or
    (pycor< 5  and pycor > 1 and pxcor = -15)
    ]
ask p-taxiways-down [set down? true]

set p-taxiways-up p-taxiways with
[ (pycor < 14 and pycor > 8 and pxcor = min-pxcor) or
    (pycor< 9 and pycor > 3 and pxcor = -8) or
    (pycor< 14 and pycor > 8 and pxcor = -3) or
    (pycor< 14 and pycor > 8 and pxcor = 10)  or
    (pycor< max-pycor  and pycor > 13 and pxcor = -13) or
    (pycor< max-pycor  and pycor > 13 and pxcor = -6) or
    (pycor< max-pycor  and pycor > 13 and pxcor = 1) or
    (pycor< max-pycor  and pycor > 13 and pxcor = 7) or
    (pycor< max-pycor  and pycor > 13 and pxcor = 14) or
    (pycor< 4  and pycor > 1 and pxcor = 13) or
    (pycor <4  and pycor > 1 and pxcor = -13) ]
```

```
ask p-taxiways-up [set up? true]

set p-taxiways-right p-taxiways with
[  (pxcor > min-pxcor - 1 and pxcor < -8 and pycor = 14) or
     (pxcor> -9 and pxcor < max-pxcor and pycor = 9) or
     (pxcor> -4 and pxcor < 5 and pycor = 14) or
     (pxcor> 9 and pxcor < 16 and pycor = 14) or
     (pxcor > -17 and pxcor < -8 and pycor = 4)    ]
ask p-taxiways-right [set right? true]

set p-taxiways-left p-taxiways with
[ (pxcor > -8 and pxcor < max-pxcor + 1 and pycor = 4) or
    (pxcor> min-pxcor and pxcor < -7 and pycor = 9) or
    (pxcor> 5 and pxcor < 11 and pycor = 14) or
    (pxcor > -8 and pxcor < -2 and pycor = 14)     ]
ask p-taxiways-left [set left? true]

ask p-taxiways[
if left? [set plabel "<"]
if right? [set plabel ">"]
if up? [set plabel "^"]
if down? [set plabel "v"]
if left? and right? [set plabel "<>" set direction-point? true]
if left? and up? [set plabel "<^" set direction-point? true]
if left? and down? [set plabel "<v" set direction-point? true]
if right? and up? [set plabel "^>" set direction-point? true]
if right? and down? [set plabel "v>" set direction-point? true]
if up? and down? [set plabel "^v" set direction-point? true]
   ]

end

;;;;;;;;   LANDING-TAKEOFF  ;;;;;;;;

to setup-landing-takeoff
set p-landing-takeoff p-runway with
[ (pycor = 1) and ((pxcor = min-pxcor) or (pxcor = max-pxcor)) ]
ask p-landing-takeoff [set pcolor blue]
set n 10
foreach sort p-landing-takeoff [
ask ? [
set plabel filter [? = n] list-gates
set n 28
     ]
   ]

end

  ;;;;;;;;  ENTRANCE-EXIT RW  ;;;;;;;;
```

```
to setup-ent-rw
set p-entry-rw patches with
  [
    (pxcor = -15 and pycor = 2) or
    (pxcor = -13 and pycor = 2) or
    (pxcor = 15 and pycor = 2) or
    (pxcor = 13 and pycor = 2) ]
ask p-entry-rw [set pcolor red]
end

  ;;;;;;;;  CROSSPOINTS  ;;;;;;;;

to setup-crosspoints
set p-crosspoints p-taxiways with
[ (pxcor = 15 and pycor = 4) or
    (pxcor = -15 and pycor = 4) or
    (pxcor = 13 and pycor = 4) or
    (pxcor = -13 and pycor = 4) or
    ((floor((pxcor + max-pxcor - floor(p-between-gates-up - 3)) mod p-between-
gates-up) = 0) and pycor = 14) or
    ((floor((pxcor + max-pxcor - 2 - floor(p-between-gates-up * 3)) mod p-
between-gates-up * 3) = 0) and pycor = 14) or
    ((floor((pxcor + max-pxcor - 2 - floor(p-between-gates-up * 3)) mod p-
between-gates-up * 3) = 0) and pycor = 9) or
    ((floor((pxcor + max-pxcor - 2 - floor(p-between-gates-up * 3)) mod p-
between-gates-up * 3) = 0) and pycor = 4) or
    (pxcor = 10 and pycor = 14) or
    (pxcor = 10 and pycor = 9) or
    (pxcor = -3 and pycor = 14) or
    (pxcor = -3 and pycor = 9) ]
ask p-crosspoints
    [set pcolor red
set crosspoint? true
if ((floor((pxcor + max-pxcor - floor(p-between-gates-up - 3)) mod p-between-
gates-up) = 0) and pycor = 14) [set g-crosspoint? true]

     ]
end
```

Bibliography

Adina Magda Florea, 1998. Introduction to Multi-Agent Systems. *International Summer School on Multi-Agent Systems. Bucharest.*

Birgit Burmeister, Afsaneh Haddadi, Guido Matylis, 1997. Application of Multi-Agent Systems in Traffic and Transportation. *Daimler-Benz Research Systems Technology, Alt-Moabit 96b, D-10559. Berlin.*

Bryan Horlyng, Victor Lesser, Régis Vincent, 2000. Multi-Agent System Simulation Framework. *16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation, EPFL, Lausanne.*

Dejan Lavbic, Rok Rupnik, 2009. Multi-Agent System for Decision Support in Enterprises. *JIOS, Volume 33, Number 2, UDC 005.53:004.8. Ljubljana.*

H.G. Visser, P.C. Roling, 2008. Optimal Airport Surface Traffic Planning using Mixed Integer Linear Programming. *Faculty of Aerospace Engineering, Delft University of Technolog, American Institute of Aeronautics and Astronautics.*

Hanbong Lee, Hamsa Balakrishnan, 2008. A Study of Tradeoffs in Scheduling Terminal-Area Operatoons. *Proceedings of the IEEE, Volume 96, Number 12, 0018-9219.*

Hussein Abbass, Axel Bender, Svetoslav Gaidow, Paul Whitbread, 2011. Computational Red Teaming: Past, Present and Future. *IEEE Computational Intelligence Magazine, 1556-603X.*

International Civil Aviation Organization, 2004. Advanced Surface Movement Guidance and Control Systems (A-SMGCS) Manual. *ICAO.*

Luca Greco, Liliana Lo Prestim Agnese Aufegello, Giuseppe Lo Re, Marco La Cascia, Salvattore Gaglio, A Multi-Agent Decision Support System for Dynamic Supply Chain Organization. *DIGGIM, University of Palermo.*

M. Wooldridge, 1997. Agent-based software engineering. *IEE Proc. on Software Engineering,*

144 (1) 26-37.

Mohamed Habib Kammoun, Ilhem Kallel, Mohamed Adel Alimi, 2005. RoSMAS: Road Supervision based Multi-Agent System Simulation. *International Conference on Machine Intelligente, Tozeur.*

Nicholas R. Jennings, Michael Wooldridge, 1999. Agent-Oriented Software Engineering. *University of London, E1 4NS.*

Philip Butterworth-Hayes, Tim Henderson, 2012. The 2012 Guide to European Air Traffic Management. *EUROCONTROL.*

Rosalind Eveleigh, 2008. SESAR Detailed Operational Description EPISODE 3. *Apron & Taxiways Management – E 2/3, E3-D2.2-025.*

S. Alam, W. Zhao, J. Tang, C. Lokan, H.Abbass, M. Ellejmi, S. Kirbi, 2012. Discovering Delay Patterns in Arrival Traffic with Dynamic Continuous Descent Approaches using Co-Evolutionary Red Teaming. *BTRIS 01444801.*

STATFOR, the EUROCONTROL Statistics and Forecast Service, 2012. Challenges of Growth 2013, Task 1: on the use of the 20-year forecast published in 2010. *EUROCONTROL.*

STATFOR, the EUROCONTROL Statistics and Forecast Service, 2010. EUROCONTROL Long-Term Forecast: IFR Flight Movements 2010-2030, *EUROCONTROL, 10/11/22-134, v1.0.*

STATFOR, the EUROCONTROL Statistics and Forecast Service, 2012. Briefing: Business Aviation in Europe in 2011. *EUROCONTROL.*

STATFOR, the EUROCONTROL Statistics and Forecast Service, 2012. CODA Digest: Delays to Air Transport in Europe Annual 2012. *EUROCONTROL.*

STATFOR, the EUROCONTROL Statistics and Forecast Service, 2007. EUROCONTROL Trends in Air Traffic Volume 3 A Place to Stand: Airports in the European Air Network. *EUROCONTROL.*

United States Government Accountability Office, 2012. Air Traffic Control Modernization: Management Challenges Associated with Program Costs and Schedules Cloud Hinder NextGen Implementation. *GAO.*

Wenjing Zao, 2012. On Dynamic Capacity-Demand Balance in the Terminal Area Using Multi-objective Co-evolutionary Computational Red Teaming. *B.Eng. (Engineering Management), Shandong Institute of Business and Technology. China.*

Web Sites

EUROCONTROL - Statistics *http://www.eurocontrol.int/articles/statistics*

IATA *http://www.iata.org*

NetLogo: http://ccl.northwestern.edu/netlogo/

MAS tools:

http://cougaar.org/wp/

http://www.agentsheets.com/

http://www.anylogic.com/

http://www.spiderland.org/

http://jade.tilab.com/

http://repast.sourceforge.net/index.html

http://cs.gmu.edu/~eclab/projects/mason/