Distributed Airport Project

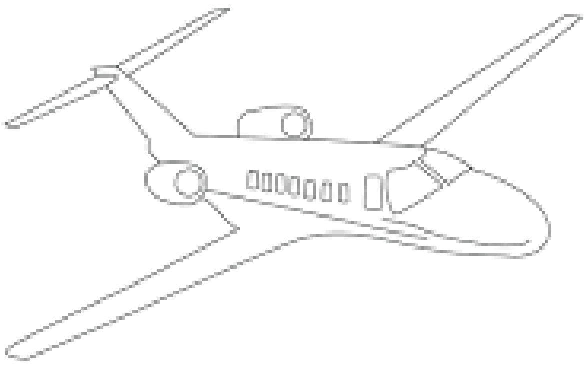
Final Project Report

Ben Gurion University of the Negev

Functional Programming in Concurrent and Distributed System

August 2023

Instructors – Dr Yehuda Ben Shimol, Mr David Leon.



Written by

Shahaf Zohar - 205978000

Danny Belozerov - 315306464

Dean Elimelech – 206323206

Abstract

In this project we implemented an airplane network routing system based on a given airfield map. The map includes one way landing pads with lanes over which planes move and respond to a control tower. The system is comprised of several main components.

- Planes.
- Control towers.
- Main controller.
- Graphics module.

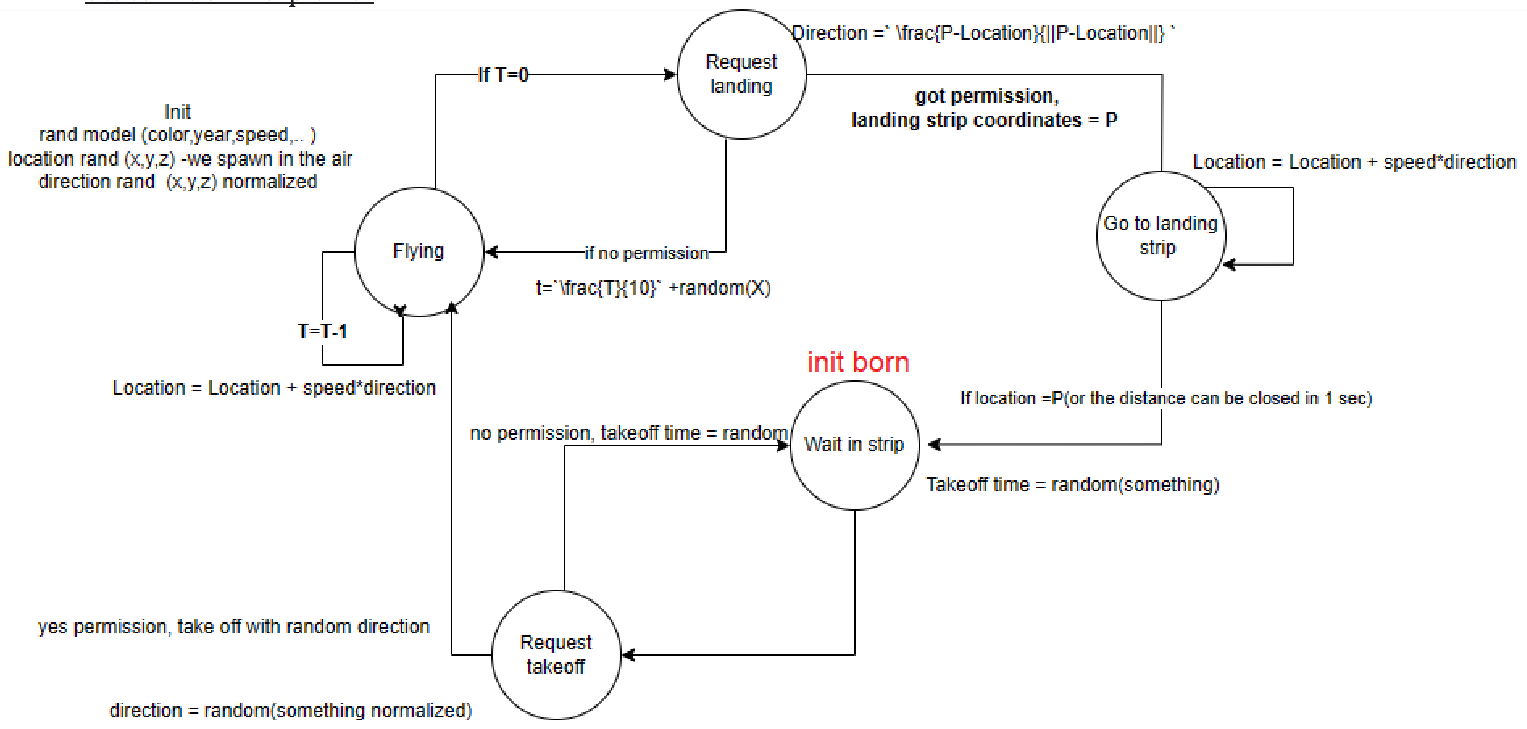
Using communication towers along the map, the plane transfers information the control towers, which make decisions regarding the continuation of plane's journey. The map is divided into four regions, so that a each control tower is responsible for its region and for routing the planes in its region.

These servers are located on different computers. Furthermore, there is a fifth computer which is the main controller and the graphics.

Plane

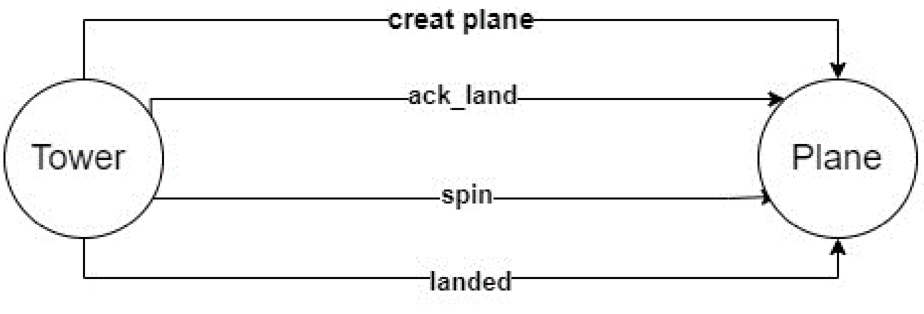
A plane is a state machine implemented with gen_statem. The planes fly along the map with direction and speed using the information, which is passed onto them by a control tower, whose domain is their current location.

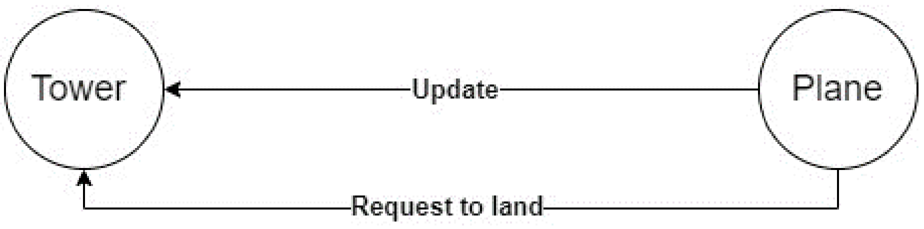
<u>State machine-plane-</u>



Direction = ` \frac{P-Location}{||P-Location||} `
If T=0
Request landing
got permission, landing strip coordinates = P
Location = Location + speed*direction
Init
rand model (color,year,speed,..)
location rand (x,y,z) -we spawn in the air
direction rand (x,y,z) normalized
Go to landing strip
Flying
if no permission
t=`\frac{T}{10}` +random(X)
T=T-1
init born
Location = Location + speed*direction
If location =P(or the distance can be closed in 1 sec)
no permission, takeoff time = random
Wait in strip
Takeoff time = random(something)
yes permission, take off with random direction
Request takeoff
direction = random(something normalized)

<u>Messages passed –</u>





<u>Tower → plane:</u>

- <u>Create plane</u> – tower create a plane with and initializes all its initial data
- <u>Spin</u> – when plane get near to a wall, tower send this whit angel, and the plane know to turn around whit this angel .
- <u>Ack_land</u> – respond about land request message.
- <u>Landed</u> – when plane landed the tower send message and the plane die.

<u>plane → Tower:</u>

- <u>Update</u> – update about new plane location.
- <u>Request to land</u> – after some time the plane ask permission to land.

Tower

The communication towers are servers implemented with gen_server. These towers are initiated by the main controller, using RPC:call on 4 different computers.

Each tower has its own rectangle of coordinates, and any plane inside that rectangle communicates with this server.

The messages that are passed can be shown in the diagrams below and above in the Plain and the Controller sections.

Periodically, the control tower sends updates to the main controller about it's ETS which includes all the planes in it's domain, so that the controller knows and can update the graphics.

Additionally , some events require assistance from the main controller:

- Tower crash – When a control tower crashes, the main controller knows about it by monitoring the process, and initiates a recovery protocol by finding an available node that can run a new process, using the ETS from the last time slot the current process was alive.
- Spin\Transfer a plane – When a plane reaches the end of the rectangle , I.E leaving the jurisdiction of the control tower, the control tower needs to know whether the plane will be going to a new rectangle, of some other control tower , or spin because it is the edge of the map. Because the control tower does not know the limits of the map, it sends a request to the main controller, asking what to do, the controller may respond with either a 'destroy' – meaning the plane will be spawned in another control tower's code, or spin – which means it's the edge of the map, and so the control tower merely updates the plane about it's new direction, using Snell's law(+ some randomness).

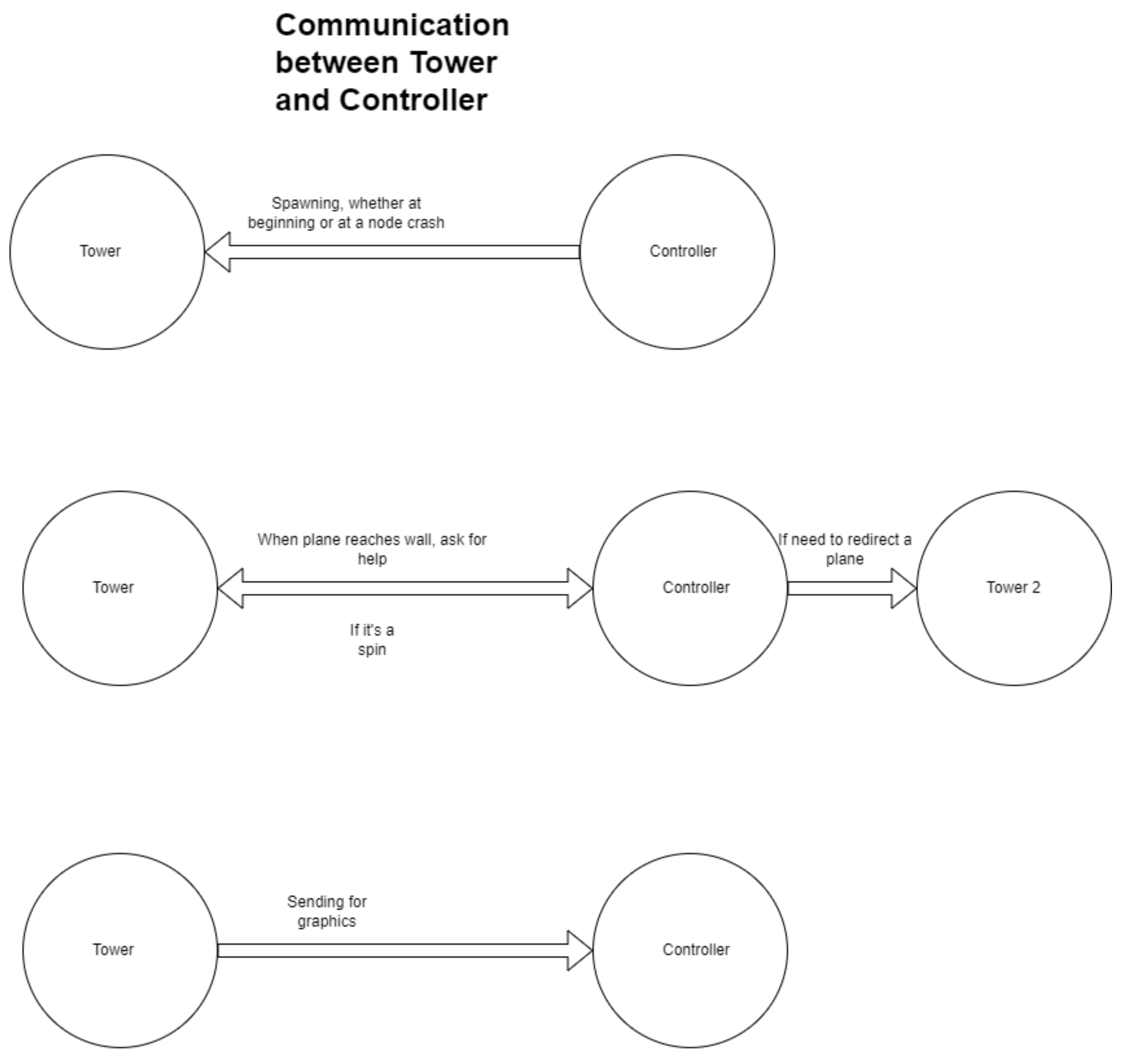
Controller

The controller is a gen_server that spawns the control towers, monitors them throughout the experiment, gathers information from them and sends it to the graphics module , so that the graphics can show all the planes in the airfield.

Methods –

- Init – start a process, using known nodes, start 4 control towers remotely, and monitor them.
- Handle call , ask help - Occasionally , the control towers do not know what to do with an airplane that is leaving their jurisdiction , so they ask the main controller what to do in a call function, the controller check's which control tower is responsible for the estimated new coordinates of the plane, and if it's another control tower then it sends the new control tower a message to spawn a new plane with the current details(speed, model etc..)
- Handle cast, ETS update – The control towers send their ETS tables to the main controller periodically in a cast message , the main controller updates his 4 ETS tables, and then forwards them to the graphics module.
- Handle info , control tower crash – We monitor the control towers when we spawn them, in order to recover and assign a new process to be responsible for the area which is now not covered, in the handle info messages we can see which process crashed, and we choose a node that hasn't crashed out of the 4, give it the same name and ETS with all the planes the crashed process had.

The diagram below shows all the links and messages passed between the controller and the towers, and the periodic message to the graphics.


Communication between Tower and Controller
Tower
Spawning, whether at beginning or at a node crash
Controller
Tower
When plane reaches wall, ask for help
If it's a spin
Controller
If need to redirect a plane
Tower 2
Tower
Sending for graphics
Controller

Graphics

- The graphics are implemented with a python process, using the Pyrlang module to link between the erlang and python, and PyGame module to display our planes with images we collected on the internet.
- The graphics receive an update periodically of the 4 ETS's that the main controller possesses at each time slot, and simply display it.

Tools used -

ETS (Erlang Term Storage) is a built-in feature in Erlang for creating and managing in-memory tables that store Erlang terms. ETS tables provide a way to efficiently store and manipulate data in a concurrent environment.

In our experiment, we needed to use ETS tables for fast lookup of planes, where our key was the airplane's Process id, we could extract information such as its location and speed, to assert what we need to do with it(spin for example).

We used ETS tables in both the main controller and each control tower, constantly trying to sync between them.

Gen server –

OTP has a generic server module, which implements the abstraction for many methods of message handling and initializing the objects.

In our case, the main controller and the control towers used the gen server behaviour,

The functions were described above.

Gen state machine –

The airplanes were state machines with 6 states.

We used the state functions callback module to transition between each state and didn't rely too much on events, because our transitions were simple.

The functions were described above in the plane module.

Pyrlang –

Because we decided to implement the graphics with PyGame, we needed a tool to translate erlang to python, and so we used Pyrlang.

Pyrlang uses the Asyncio:Queue module to implement a message box, and thus allowing message passing like we needed.

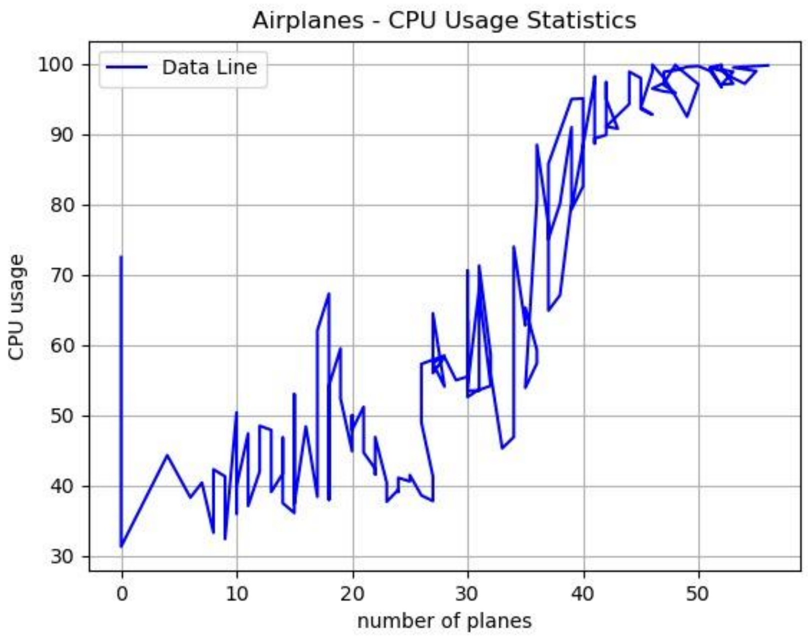
We overrode a few functions to implement a while loop, that is necessary for PyGame to operate.

PyGame –

PyGame is a python library, often used to create simple games, but it can support simple GUI's such as the one we needed here, we used images of a map, and a few airplane models, and using the coordinates we got from the ETS's we could print the planes in their locations, and a small Z axis logo below the planes.

Statistics

Statistics on CPU utilization depending on the number of planes


Airplanes - CPU Usage Statistics
Data Line
CPU usage
number of planes