Project Report
Erl of Thrones

Adir Ben Azarya 203904610

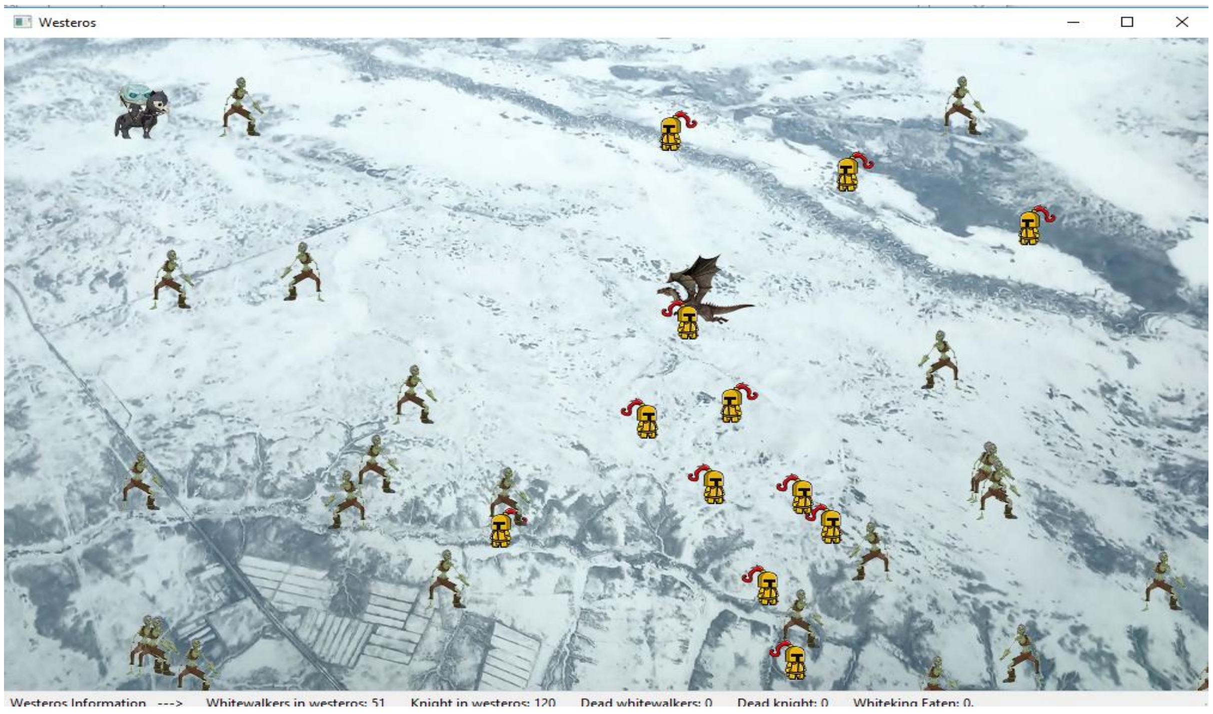
Kobi Eini 201553245

Table of Contents

Simulation overview

Our project is a simulation of last battle of Game of Thrones series. It is based on Erlang, OTP interface and graphics by WxWidgets.

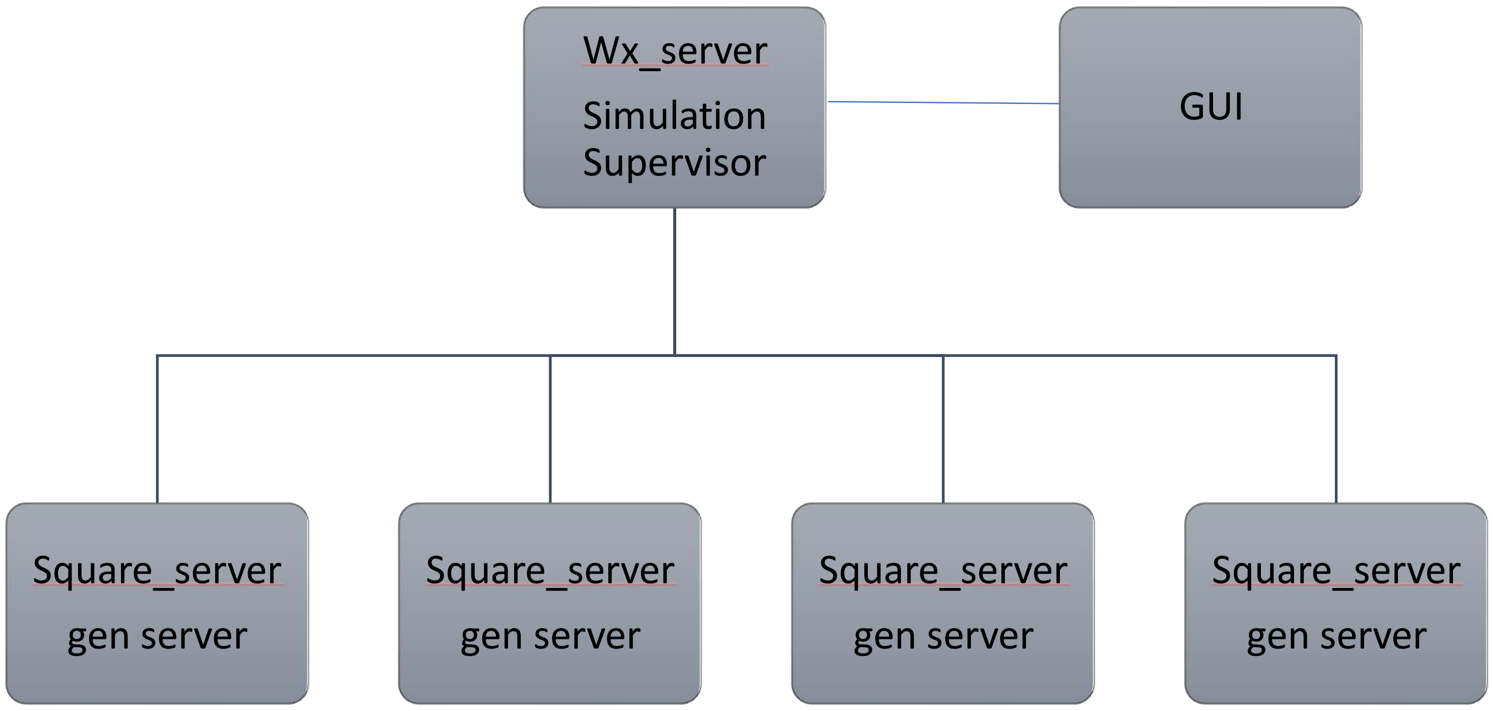
The battle is between 4 characters which ends when the white king gets killed. The simulation begins with an empty board where the user can place 3 characters: knight, dragon and whiteking. Each character moves randomly regarding some basic rules. The 4 character is whitewalker which is created by the whiteking. When the white king gets killed by the knights or dragons, all the whitewalkers are killed as well and the simulation ends.



Westeros Information ---> Whitewalkers in westeros: 51 Knight in westeros: 120 Dead whitewalkers: 0 Dead knight: 0 Whiteking Eaten: 0.

Simulation structure:

Overview:



While building the simulation, we tried to stick to few principles:

1. Maximal usage of the various OTP and native Erlang modules, capabilities, BIF's etc.
2. Design the game elements to be isolated and run concurrently.

After doing some research, we decided to use the OTP behavior of Supervisor and children. The structure and hierarchy are shown in the figure above. Following this OTP model allowed us to greatly simplify the micro-management of ensuring each process is alive and running well, which is normally done with start_link \ monitor BIF's. Instead, the entire lifecycle of any process created in any part\level of the application, is managed by some supervisor; including starting the process, shutting it down, and restarting if necessary due to an unfortunate event.

Concurrency:

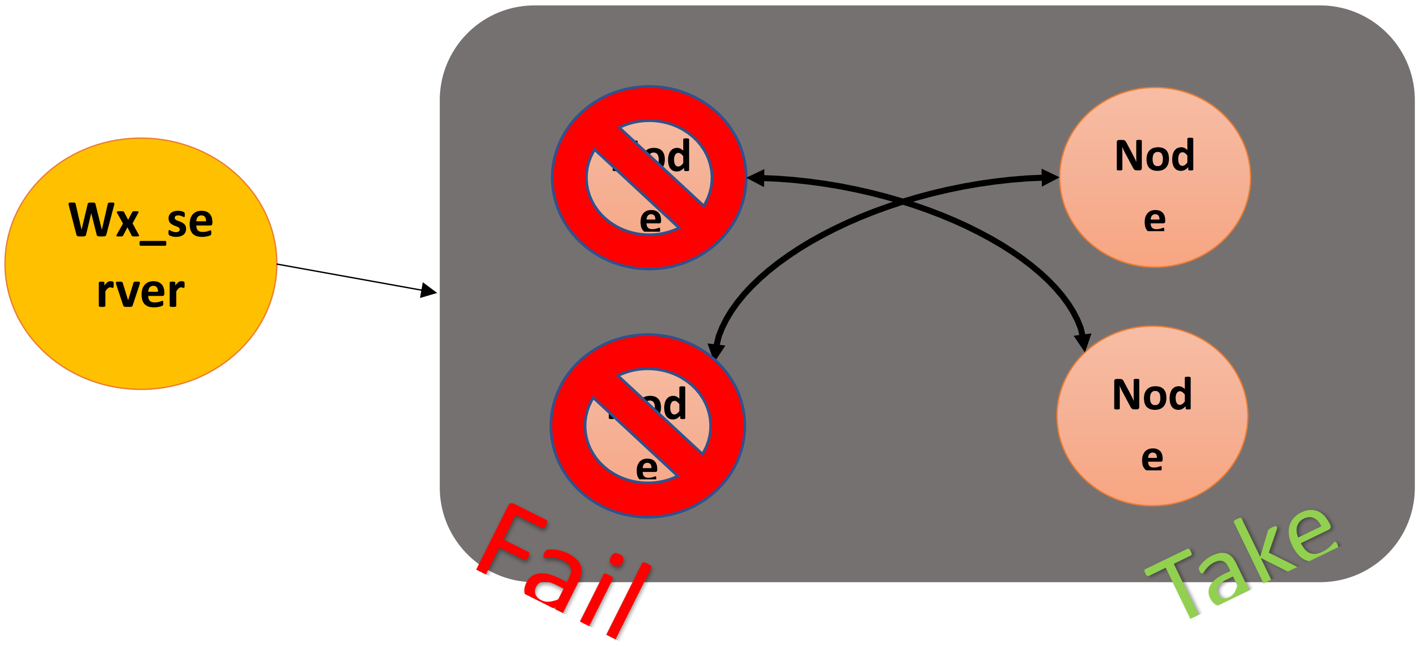
The game is comprised of several main components:

1. Wx server
2. Square servers
3. Characters
4. Shells

The first two entitis are implemented as an OTP gen_server. Some of the required data is stored in the gen_server State, in form of a record, while some of the data is stored in global ETS tables, such as the characters (processes) ID's. This way, each process runs independently and all the interaction between the entities is done via massage passing. We used asynchronous massage passing, using the gen_server:cast\handle_cast functions.

Distribution:

We decided to utilize the built-in capabilities of the Distributed OTP Application. In this way, the simulation may run on a connected network of nodes\machines, and to "jump" to a functioning node in the network if the current machine crashes.
The simulation is distributed by 4 computers (shells) which are responsible of 4 squares (the square servers). Each square server has its own ETS table that includes characters ID’s. Those tables are updated every amount of time (predefined). This gave us the ability to manage the processes even if one of the computers crashes.



Files Description and Organization:

- erlangFinalPro/
- wx_server.erl
- square_server.erl
- knight.erl
- whitewalker.erl
- whiteking.erl
- dragon.erl
- Images/ [contains all the images used in game]

-erlangFinalPro /
This folder contains all the *.erl files.

-wx_server.erl
This Erlang file implements OTP's Simulation behavior. After compilation, enter wx_server:start() in the shell. This function creates the board, the squares division, the gui and database on all participating nodes. Each table in the DB is updated throughout the simulation.
When running on multiple nodes, in case of an unexpected shutdown of the one of the square nodes, a backup node will controlled its area and the one that crashed. Then the backup gets all of the processes that crashed to its ETS table and reconstruct the characters that disappeared. When the crashed node is up again, it takes to its ETS table only the characters which are in its area and the back up erases them from its table.

- square_server.erl

This Erlang file is responsible of a square in the board (predefined). This node concludes ETS table which has the characters ID's. In addition, the node is responsible of updating ETS tables in wx_server and its neighbors, adding new characters or deleting them and use a watchdog that keeps checking the state of the shell and the connectivity with wx_server.

-knight.erl

This Erlang file is responsible of the functionality of the knight's characters. Those characters move randomly according to those rules:

- if there is a whiteking in its radius, the knight will attack the king.
- If there is a white walker in its radius, the knight will speed up and run away.

Each knight has its own lifetime that it disappears when the time gets to 0. Another way of disappearing is when whitewalkers catch the knight.

-whiteking.erl

This Erlang file is responsible of the functionality of the whiteking character. This character stands in one spot and generates whitewalkers. The whiteking can be eaten when:

- Knight succeed to catch it.
- Dragon succeed to catch it.

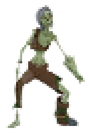
When the whiteking is eaten, all the whitewalkers disappears as well.

-whitewalker.erl

This Erlang file is responsible of the functionality of the whitewalker's characters. Those characters move randomly according to those rules:

- if there is a knight in its radius, the whitewalker will try to catch it.
- When it catches a knight, the knight becomes whitewalker. Means that this whitewalker duplicates itself.

Each whitewalker has its own lifetime that it disappears when the time gets to 0. Another way of disappearing is when the whiteking is dead.



-dragon.erl

This Erlang file is responsible of the functionality of the dragon's characters. Those characters move randomly according to those rules:

- if there is a whiteking in its radius, the dragon will attack the king.

Each dragon has its own lifetime that it disappears when the time gets to 0. The dragon is the fastest character in the simulation and invincible.