FIFA Distributed Game

Functional Programming in Concurrent and Distributed Systems - Ben Gurion Universityend

By: Moshe Dahan and Yuval Assayag

Instructors: Dr. Yehuda Ben-Shimol and Mr. David Leon

Table of Content:

Table of Content: 2

Overview 3

Characters 5

Wx Server Responsibilities 5

Monitors (Regular Servers) 6

Who Is Entitled Of A Process?! 6

Strategies 6

Project Objective 7

Design 7

Files Description 8

Servers Crash Handling 9

Game Over 12

Main Components FSM 13

- Computer Player FSM 13
- Computer Player Events Table 14
- Controlled Player FSM 14
- Controlled Player Events Table 15
- Ball FSM 15
- Ball Events Table 16

Statistics 16

Coding Techniques 16

Main Difficulties and Obstacles 18

Work division 18

Conclusions 18

TimeLine 18

User Manual 20

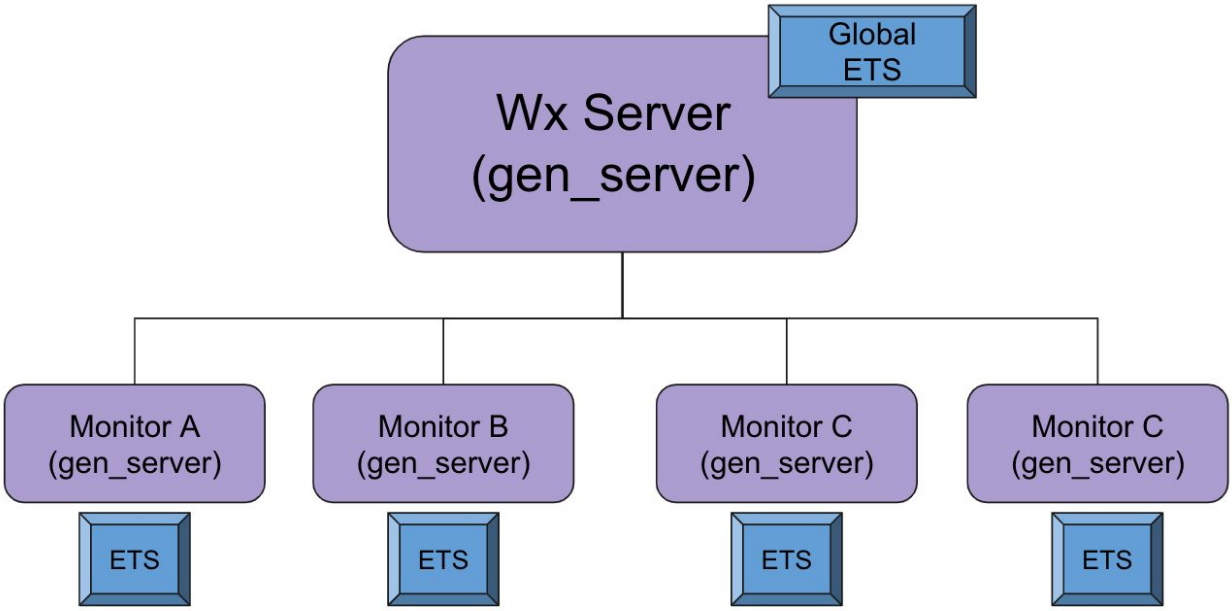
- Local Display 20
- Distributed Display 22

Links 24

Overview

Our project simulates a FIFA game. It is based on Erlang, OTP interface, and graphics by WxWidgets.

The game includes two rival teams that work according to two different strategies, a referee, and a ball. Each character moves across the soccer field.



Characters

There are 5 types of characters:

1. Computer players - divided into two teams, moves according to its team strategy. When owning the ball the player would try to move forward to the rival team's net and to kick the ball to the direction of the net. The player's kick destination is randomized.
2. Two goalies - one for each team. The goalie movement is restricted to the area of their team's net, the job is to protect the net.
3. A ball - the ball is an important component of the game it can be attached to a running player in case it fetches it. Besides, it can be static if none possesses it or kicked if the player applies the kick action to it. The ball is implemented by FSM - using gen_statem. If the ball coordinates are inside the net then the relevant group gains a point. The ball location is set according to the location of the player that possesses it.
4. A referee - the referee watches the ball from a close distance.

Wx Server Responsibilities

1. Initiation and monitor of the 4 regular servers
2. Rearrange all servers' responsibilities in case of a crash.
3. Manages the game's graphics.
4. Possesses the statistic of the entire game (ball possessions, points, etc).
5. Refreshing the locations of all the players.
6. Acting upon the mouse movement using WX functions (wx_object).
7. Announce when the game is over.

Monitors (Regular Servers)

Each server responsible for a certain slice of the screen - depending on how many servers are running at the moment.
In initialization time there are 4 regular servers running each of them responsible on a vertical rectangle which represents a quarter of the soccer field.
Each screen monitors the objects within its boundaries.

Who Is Entitled Of A Process?!

- ➔ Each player.
- ➔ The ball.
- ➔ Each goalie.
- ➔ The controlled player

All these processes run simultaneously and managed by the wx Widget server.

Strategies

1. The first team strategy is to work randomly across the soccer field trying to block and score goals.
2. The second team strategy is a more defensive approach at all times half of the group players are on the defense trying to block the rival team to score a goal. The rest of the players can move more freely across the field (both attack and defense).

Project Objective

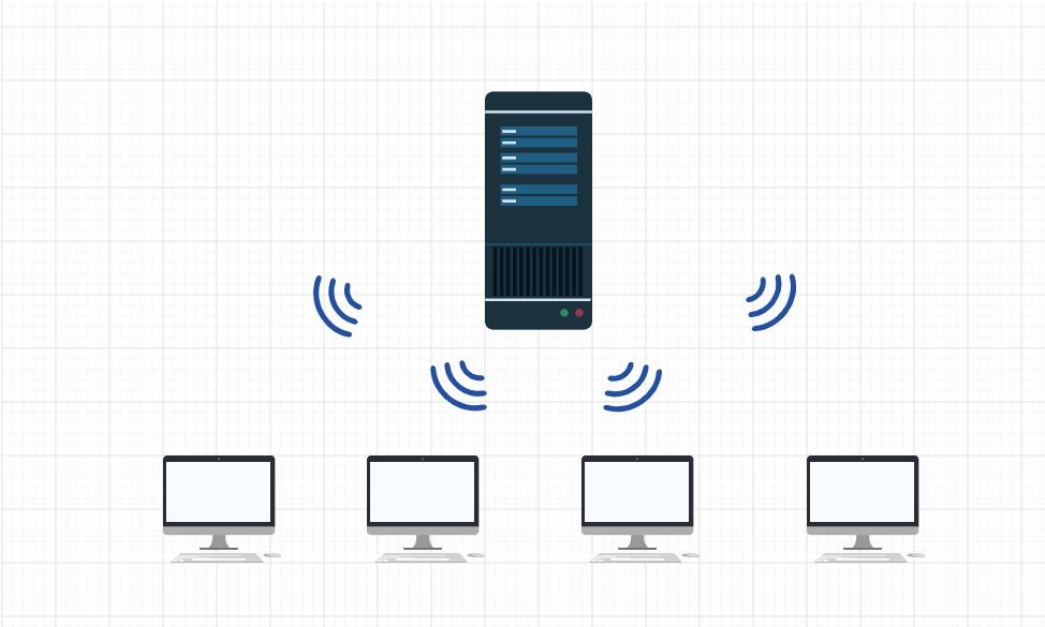
Our goal in this project was to create a distributed FIFA game using the Erlang language and Functional Programming knowledge we have gained along the course.

While building the simulation, we tried to:

1. Maximize usage of OTP and native Erlang modules, messaging, capabilities, etc.
2. Isolate the game components and enable them to run concurrently.
3. Resiliency to monitor crashes.

Design

The design is based on OTP in a master-slave behavior. The structure and hierarchy are shown in the figure below. Following this OTP model simplified the management of the process and servers. To monitor the existence of the processes we used start_link and monitor Erlang's functions. In the case of a crash the master (wxWidget server) is responsible for dividing the work to its remaining slaves (the other servers).



Files Description

`wxserver.erl` - contains the main logic of our project. It is implementing gen_server behavior and uses WxWidget capabilities for the UI. Contains ~780 code lines.
monitor.erl - contains the code for the four servers each server should be responsible for a quarter of the soccer field. It is implementing gen_server behavior. Contains ~260 code lines.

`controlledplayer.erl` - contains the code for the FSM of the controlled player. Implement using gen_Statem behavior. Contains ~80 code lines.

`computerplayer.erl` - contains the code for the FSM of the computer player this is a regular player which the user has no control over. This namespace handles the movement of both the regular player and the goalies. Contains ~150 code lines.

`ball.erl` - contains the code for the FSM of the ball. Contains ~90 code lines.

`utils.erl` - contains a function that can help display photos and lines on the screen. Contains ~150 code lines.

`etsutils.erl` - contains all the ETS of our code which is the equivalent of a DB with easy access to it. Contains ~90 code lines.

`Common.erl` - all the common functions for all the described above namespace. A common namespace could help us keep the code concise. Contains ~140 code lines.

`params.hrl` - holds all the Macros (constants). Contains ~55 code lines.

Servers Crash Handling

As already explained the game is distributed over 5 computers (or terminals).

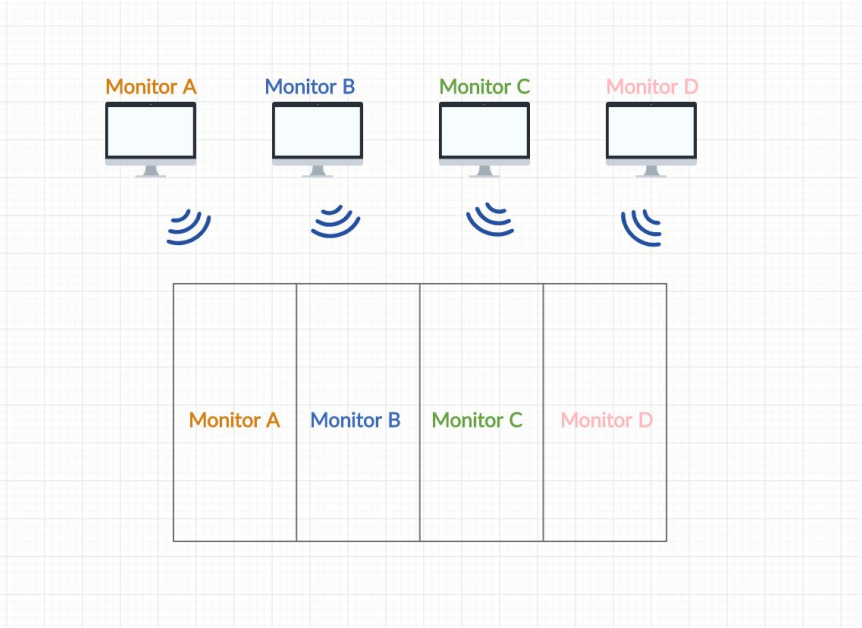
We decided to support servers crash handling under the assumption that several servers might lose connection during the game or crash for any other reason. This service has high resiliency and can survive with one functioning monitor.

How does it work?

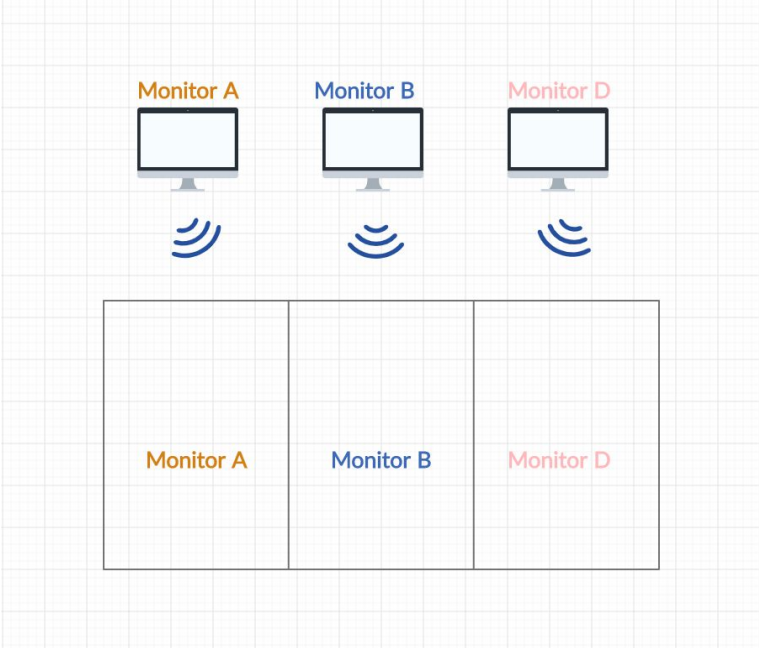
In initiation time there are 4 monitors under the master (Wx Server). When a server is losing the connection with its master, the master takes charge and assigns the dead server's job to the remaining servers and divides the field to the other servers. In this way, all the components that once was supervised by the lost server now have a new supervisor server to approach with messages. A dead server can reestablish the connection with the master and regain his power over a new slice of the field.

Simulation:

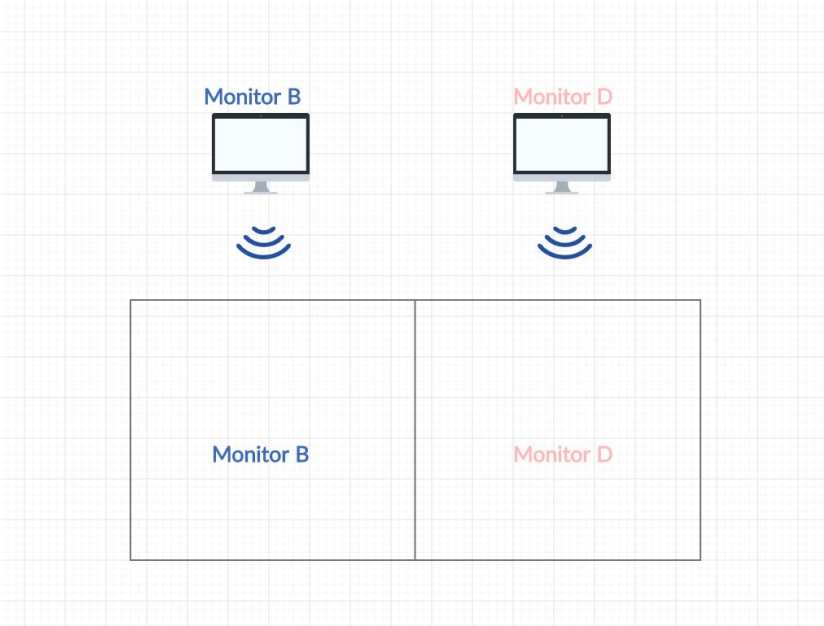
In initiation time the field is divided into 4 vertical rectangles:



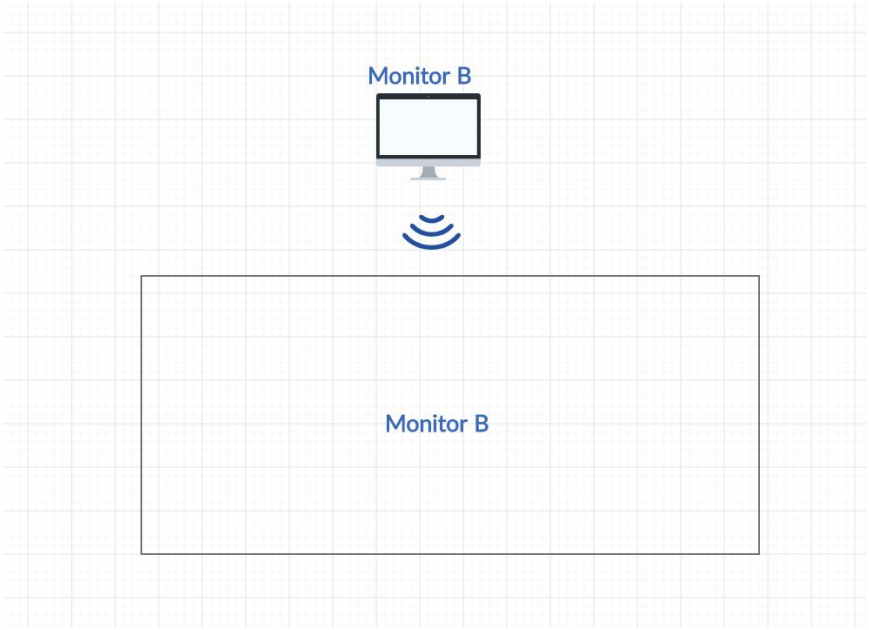
When server C suddenly Crushes:



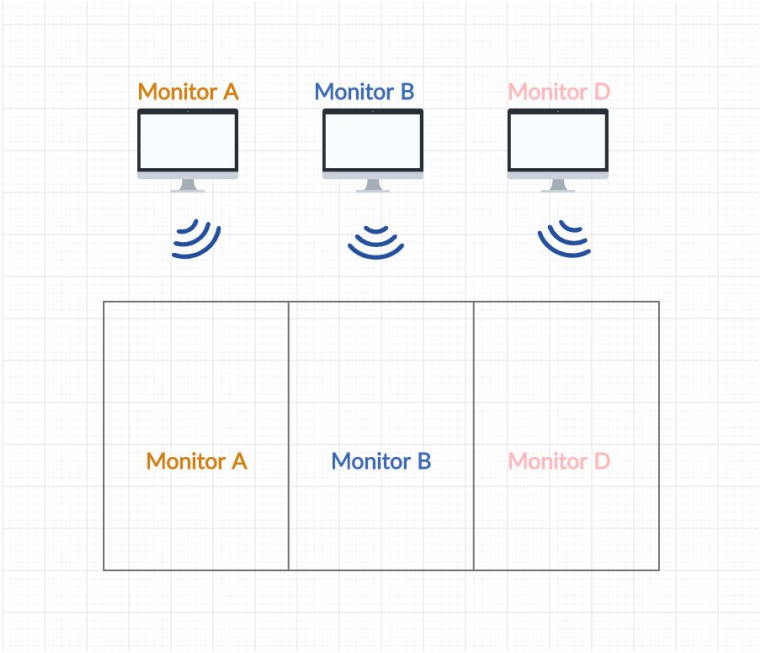
And the game continues after wx server reorganized the responsibilities of the three remaining servers but an unfortunate thing happens and now monitor A is down:



And now again - another one is DOWN! But the game still continues with only one monitor:



There is also support for reestablishing connection for the server that died so we can now restart them and the wx server will reconnect them into the game:



Game Over

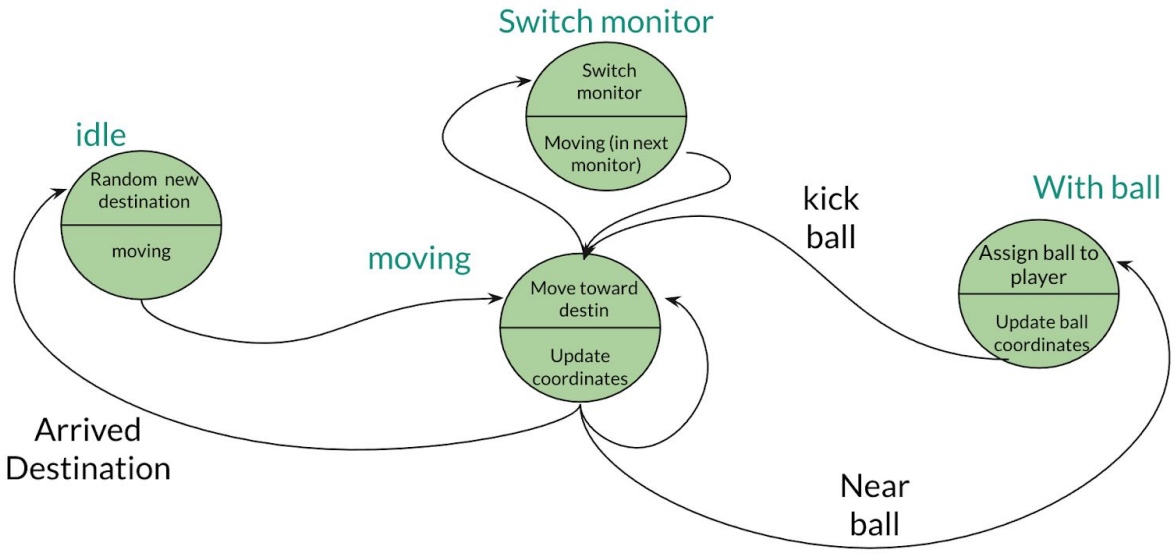
The game is over when one of the teams gains 3 points.

How does a team gain a point?
By scoring 3 Goals to the opponent team.

Reasoning:
The presentation time is limited.

Main Components FSM

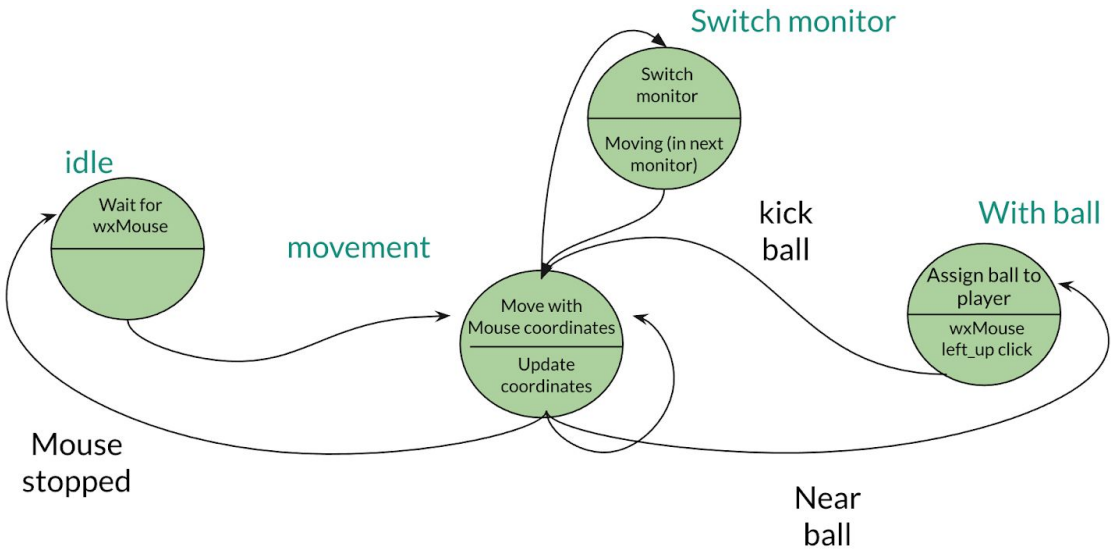
Computer Player FSM



Computer Player Events Table

State	Event	Condition	Action	Next State
waitForStartGame	Mouse click	-	Component init	idle
idle	Enter state	-	Random new destination	moving
moving	Move to next location	Location /= detination	Update next location in monitor	moving
moving	Move to next location	Ball is Near	Update ball to move with player	moving
moving	kick	Player with ball	Set destination for ball	moving
moving	Move to next location	Arrive Destination	Exit state	idle
moving	Move to next location	Crossed monitor borders	Switch monitor	Moving (in next monitor)

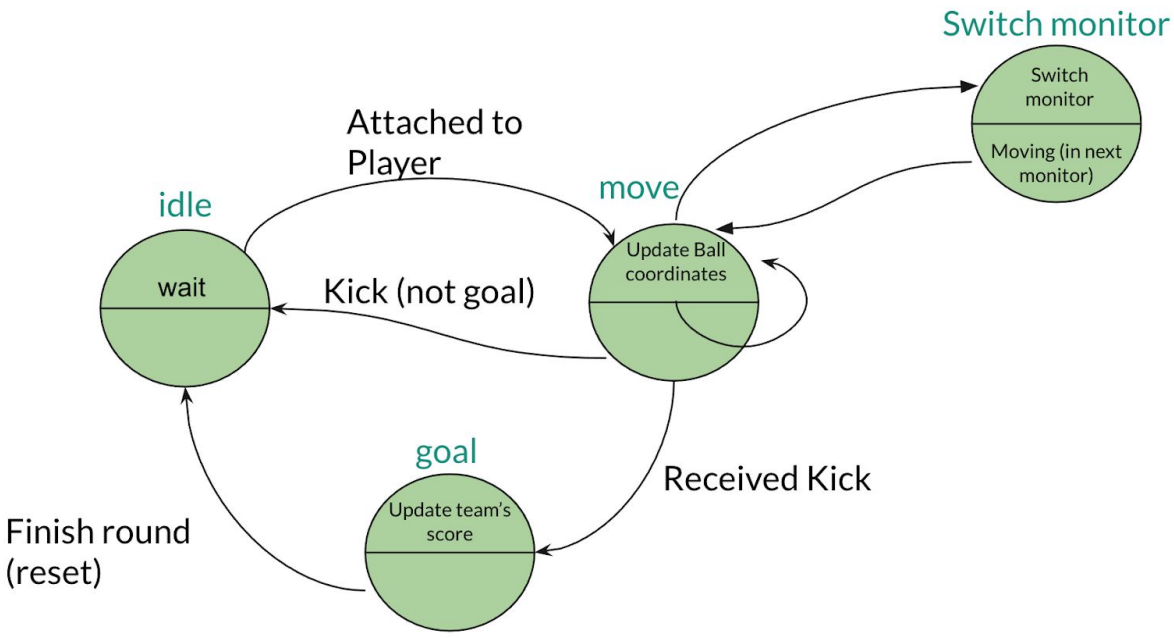
Controlled Player FSM



Controlled Player Events Table

State	Event	Condition	Action	Next State
waitForStartGame	wxMouse click	-	Component init	idle
idle	Enter state	-	Wait for wxMouse	moving
moving	wxMouse moved		Update next location in monitor	idle
moving	wxMouse moved	Ball is Near	Update ball to move with player	idle
moving	wxMouse left_up click	Player with ball	Set destination for ball	idle
moving	wxMouse moved	Crossed monitor borders	Switch monitor	Moving (in next monitor)

Ball FSM



Ball Events Table

State	Event	Condition	Action	Next State
waitForStartGame	wxMouse click	-	Component init	idle
idle	Enter state	Attached to player	Update new Coordinations	idle
moving	kicked		Set new destination	moving
moving	Moved to next location	Location /= destination	Set next location toward destination	moving
moving	Moved to next location	Player is near	Attached to player	idle
moving	Moved to next location	Location == destination	Check if goal	Finish round(goal) / idle (not goal)
moving	Move to next location	Crossed monitor borders	Switch monitor	Moving (in next monitor)

Statistics

The game also support in statistics on the game and on the players, the following statistics are monitored and shown in the end of each round and at the end of the game:

1. Each time points.
2. The total number of ball possessions.
3. The number of ball possessions of each team.
4. The number of ball possessions of each player.

These statistics are maintained by the ETS (DB) and resilient to servers crash as these ETS are maintained by the WX server.

Coding Techniques

1. An exciting fact about the code is that it is very concise and short in contrast to its vast functionalities.
   We were surprised when we measured the number of lines.
   In a non-functional language, the code length would probably have been at least doubled.
2. Pattern matching is functionality that Erlang enables to perform the

code more efficiently and concisely.

3. Recursions are also a strong tool in functional programming.
4. WX interface is full of options and enables us to add graphics to the game without having a lot of knowledge in the UI world.
5. Using gen_server behavior is an easy way to manage async communication.
6. Using gen_statem behavior to implement FSM as we have learned.

Main Difficulties and Obstacles

- ➔ Understanding the WxWidget.
- ➔ Implement the async communication between the servers - using `gen_server`.
- ➔ Distribute the game over 4 monitors (split screens).
- ➔ Trying to implement a complex game with a lot of logic and components that work in parallel.
- ➔ Working together remotely on the Zoom platform.

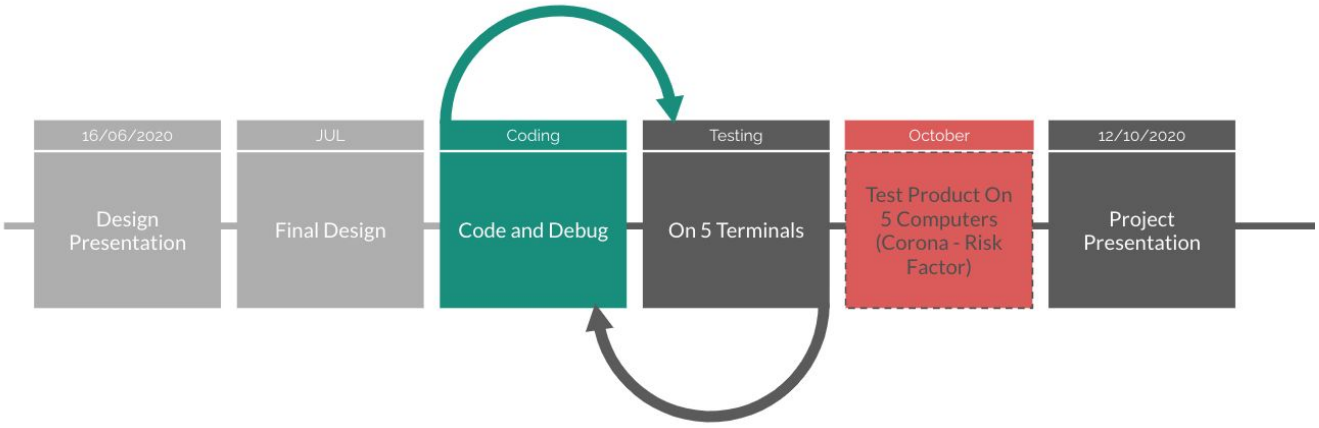
Work division

We worked entirely together on every part of the project.
Pros: we both know to handle all the parts of the project and have the same knowledge regarding all the techniques executed along with it.
Cons: it was probably very time consuming which made the work on it slower in the long term.

Conclusions

- Scalability should be taken care of in the planning stage.
- ETS is a good source for sharing data between processes.
- It is critical to plan the gen_server and gen_statem prior to the coding phase.

TimeLine


16/06/2020
Design Presentation
JUL
Final Design
Coding
Code and Debug
Testing
On 5 Terminals
October
Test Product On 5 Computers (Corona - Risk Factor)
12/10/2020
Project Presentation

User Manual

Local Display

In order to run the project locally (on one computer) you have to open 5 terminals on the path that contains all of the erlang files (.erl).

First go to params.hrl and change the name of the following variables:

```
-define(MONITOR_LONG_NAME_A, 'monitorA@<Your-Computer-Name>').
-define(MONITOR_LONG_NAME_B, 'monitorB@<Your-Computer-Name>').
-define(MONITOR_LONG_NAME_C, 'monitorC@<Your-Computer-Name>').
-define(MONITOR_LONG_NAME_D, 'monitorD@<Your-Computer-Name>').
```

Then you should type the following command in this manner:

On the first terminal:

```
erl -sname monitorA
```

On the second terminal:

```
erl -sname monitorB
```

On the third terminal:

```
erl -sname monitorC
```

On the fourth terminal:

```
erl -sname monitorD
```

In the 5th terminal you should write these two commands:

```
erl -sname main -smp
c(wxserver).
wxserver:startme().
```

Distributed Display

To run the project on five different computers, you have to open 5 terminals each terminal on different computers, in the path that contains all of the erlang files (.erl) and follow the instructions described below:

First go to params.hrl and change the name of the following variables:

```
-define(MONITOR_LONG_NAME_A, 'monitorA@<Computer-1-IP-Address>').
-define(MONITOR_LONG_NAME_B, 'monitorB@<Computer-2-IP-Address>').
-define(MONITOR_LONG_NAME_C, 'monitorC@<Computer-3-IP-Address>').
-define(MONITOR_LONG_NAME_D, 'monitorD@<Computer-4-IP-Address>').
```

Then you should type the following command in this manner:

On the first computer:

```
erl -name monitorA@<Computer-1-IP-Address> -setcookie <cookie-name>
```

On the second computer:

```
erl -name monitorB@<Computer-2-IP-Address> -setcookie <cookie-name>
```

On the third computer:

```
erl -name monitorC@<Computer-3-IP-Address> -setcookie <cookie-name>
```

On the fourth computer:

```
erl -name monitorD@<Computer-4-IP-Address> -setcookie <cookie-name>
```

In the 5th terminal you should write these two commands:

```
erl -name main@<Computer-5-IP-Address> -smp -setcookie
<cookie-name>
c(wxserver).
wxserver:startme().
```

Note: `<cookie-name>` must be identical on all 5 computers.

Links

1. GitHub Repository
2. Youtube Video
3. Presentation