Universitatea din București
Facultatea de Matematica și Informatica

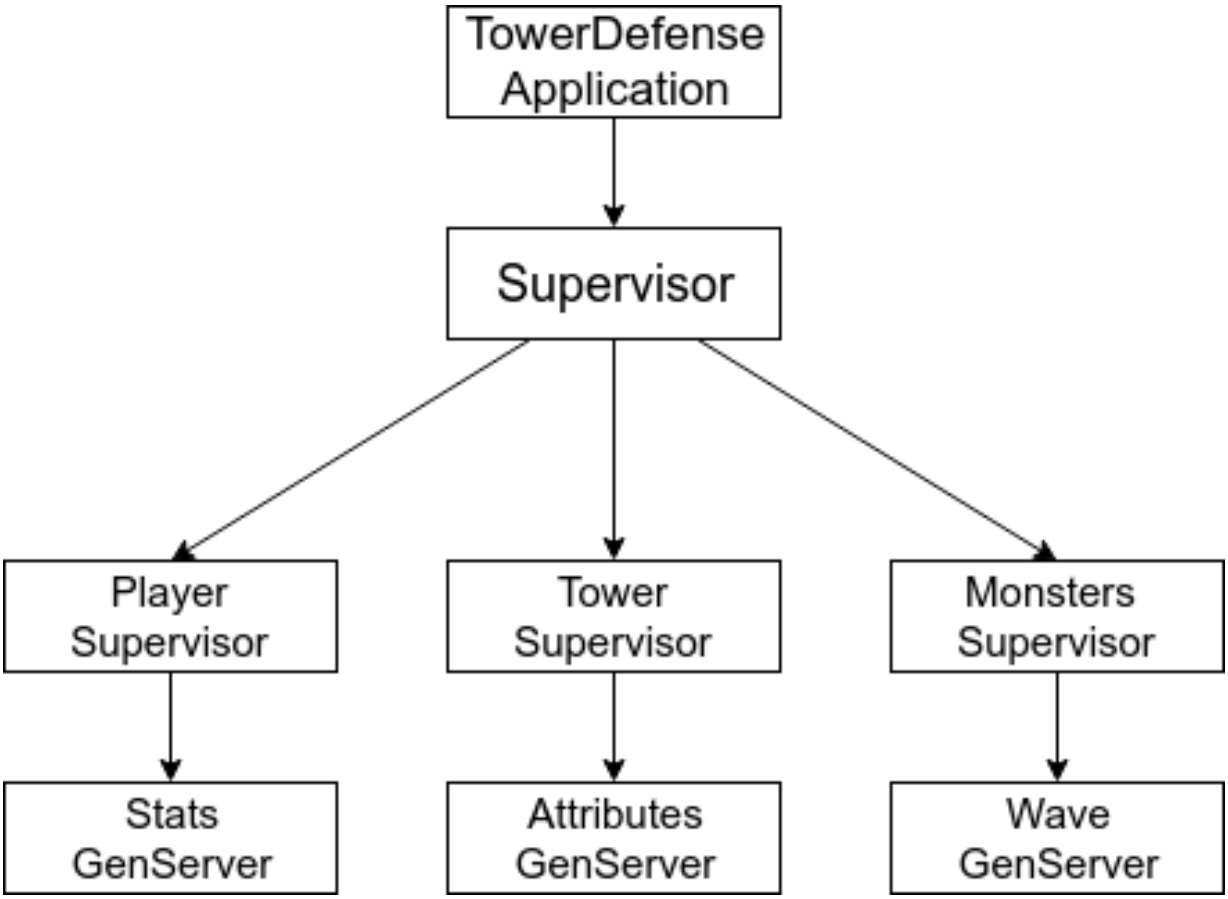
TowerDefense

Ilieș Tatiana, grupa 506
Sandu Nicușor, grupa 506

București 2018

I. Introducere

O aplicație scrisa în Elixir este formata din mai multe module. Aceste module sunt organizate sub forma unei structuri arborescente, ce ofera aplicației caractaristica de fault-tolerant (rezistenta la erori). Fiecare modul din acest arbore este supravegheat de un modul aflat cu un nivel mai sus. În cazul în care un modul întâlnește o eroare și se blochează, supervisor-ul sau îl poate restarta.



II. Structura aplicatiei

Aplicație este formata din 8 actori, distribuiti astfel:
- un actor de tip Application
- patru actori de tip Supervisor (dintre care trei secundari, și unul principal care are rolul de a-i supraveghea pe primii trei)
- trei actori de tip GenServer

1) TowerDefense Application

Desemneaza rădăcina structurii arborescente a aplicației. Modulul ce joaca rol de Application este initializat în momentul pornirii aplicației. Desemnarea acestei functionalitati se face în fișierul de configurare mix.exs.

```
def application do
  [
    mod: {TowerDefense, []},
    extra_applications: [:logger]
  ]
end
```

Modulul TowerDefense are dimensiuni reduse. Acesta este desemnat ca fiind Application prin folosirea macro-ului „use Application”. Metoda start are rolul de a afisa un mesaj în consola și de a porni supervisor-ul principal.

```
defmodule TowerDefense do
  use Application

  def start(_type, _args) do
    IO.puts "Run Game.help for info"
    TowerDefense.Supervisor.start_link
  end
end
```

2) Supervisor-ul principal

Acest supervisor se ocupa cu monitorizarea celor trei supervisori secundari. Prin apelarea macro-ului „use Supervisor” acesta primește functionalitati de proces Supervisor.
Supervisor-ul este alcătuit din 2 metode. Metoda de pornire „start_link” creeaza procesul aferent. În metoda „init” este definita lista de copii(children). Acești copii sunt nodurile ce urmează să fie supravegheate.

```elixir
defmodule TowerDefense.Supervisor do
  use Supervisor

  def start_link do
    Supervisor.start_link(__MODULE__, [])
  end

  def init(_) do
    children = [
      supervisor(TowerDefense.Player.Supervisor, []),
      supervisor(TowerDefense.Monsters.Supervisor, []),
      supervisor(TowerDefense.Tower.Supervisor, [])
    ]
    supervise(children, strategy: :one_for_all)
  end
end
```

Un copil poate sa primeasca și un parametru „restart”. Acest parametru poate avea 3 valori:

- :permanent
Când se blochează, procesul copil va fi restartat, indiferent cum a fost terminat procesul.

- :transient
Când se blochează, procesul copil va fi restartat doar dacă procesul a fost terminat fără eroare.

- :temporary
Când se blocheaa, procesul copil nu va fi restartat, indiferent de modul în care a fost terminat.

Comanda „supervise” primește doi parametri. Primul parametru este lista de copii. Cel de-al doilea reprezintă strategia de supraveghere, care poate fi de 3 tipuri:

- :one_for_one
Atunci când un proces se blochează, supervisor-ul îl reporneste.

- :one_for_all
Atunci când un proces se blochează, supervisor-ul îl reporneste pe acesta, dar și restul proceselor supravegheate.

- :rest_for_one
Atunci când un proces se blochează, supervisor-ul îl reporneste pe acesta, și toate pricesele ce se afla sub el.

- :simple_one_for_one
Funcţionează în aceeași maniera ca :one_for_one. Diferența este ca aceasta strategie primește în lista de children un singur copil. Procesul copil poate fi creat de mai multe ori, apeland metoda „start_child”. Aceasta strategie este folosită atunci când dorim sa supraveghem un numar de procese de același tip.

3) Player Supervisor

Acest Supervisor supravegheaza modulul TowerDefense.Player.Stats. Macro-ul „alias” ne permite să facem referire la acest modul într-o forma mai scurta: Stats.

```elixir
defmodule TowerDefense.Player.Supervisor do
  use Supervisor
  alias TowerDefense.Player.Stats

  def start_link do
    Supervisor.start_link(__MODULE__, [])
  end

  def init(_) do
    children = [
      worker(Stats, [])
    ]
    supervise(children, strategy: :one_for_one)
  end
end
```

4) Tower Supervisor

Acest Supervisor supravegheaza modulul TowerDefense.Tower.Attributes.

```elixir
defmodule TowerDefense.Tower.Supervisor do
  use Supervisor
  alias TowerDefense.Tower.Attributes

  def start_link do
    Supervisor.start_link(__MODULE__, [])
  end

  def init(_) do
    children = [
      worker(Attributes, [])
    ]
    supervise(children, strategy: :one_for_one)
  end
end
```

5) Monsters Supervisor

Acest Supervisor supravegheaza modulul TowerDefense.Monsters.Wave.

```elixir
defmodule TowerDefense.Monsters.Supervisor do
  use Supervisor
  alias TowerDefense.Monsters.Wave

  def start_link do
    Supervisor.start_link(__MODULE__, [])
  end

  def init(_) do
    children = [
      worker(Wave, [])
    ]
    supervise(children, strategy: :one_for_one)
  end
end
```

6) Stats GenServer

Acest modul păstrează datele jucatorului. State-ul sau intern păstrează un map cu doua atribute: life și gold. Life reprezintă viața jucatorului. Acest atribut va scadea cu 10 pentru fiecare monstru rămas în viața la finalul unei lupte. Gold reprezintă moneda din joc. De fiecare data când omoară un monstru, jucătorul va fi rasplatit cu o anumita suma.

```elixir
defmodule TowerDefense.Player.Stats do
  use GenServer

  def start_link do
    GenServer.start_link(__MODULE__, [], name: :player_process)
  end

  def init(_initial_data) do
    IO.puts "Started player module!"
    stats = %{life: 100, gold: 100}
    {:ok, stats}
  end
```

Modulul este prevăzut cu 3 metode importante:
- get_my_state – returneaza starea curenta a state-ului

```elixir
def get_my_stats() do
  GenServer.call(:player_process, {:get_the_stats})
end
```

- modify_my_life (amount) – modifica valoarea vieții jucatorului

```elixir
def modify_my_life(amount) do
  GenServer.call(:player_process, {:modify_the_life, amount})
end
```

- modify_my_gold(amount) – modifica valoare totala a aurului jucatorului

Aceste metode au în spate un set de handlere ce ajuta la modificarea state-ului:

```
  def handle_call({:get_the_stats}, _from, my_stats) do
    {:reply, my_stats, my_stats}
  end

  def handle_call({:modify_the_life, amount}, _from, my_stats) do
    %{life: old_life} = my_stats
    new_stats = Map.put(my_stats, :life, old_life + amount)
    if old_life + amount > 0 do
      {:reply, new_stats, new_stats}
    else
      IO.puts "Oh no, an unexpected bug ):"
      {:something_stupid}
    end

  end

  def handle_call({:modify_the_gold, amount}, _from, my_stats) do
    %{gold: old_gold} = my_stats
    new_stats = Map.put(my_stats, :gold, old_gold + amount)
    {:reply, new_stats, new_stats}
  end
end
```

7) Attributes GenServer

Acest modul păstrează informațiile legate de turn (tower). Turnul este cel care lupta împotriva valurilor de monstri. El ataca la fiecare secunda. După aparitia unui val, turnul are un timp limita de 7 secunde pentru a omorî toți monștrii. Acesta modul păstrează în state-ul sau nivelul, atacul și costul de upgrade al turnului.

```
defmodule TowerDefense.Tower.Attributes do
  use GenServer
  alias TowerDefense.Player

  def start_link do
    GenServer.start_link(__MODULE__, [], name: :tower_process)
  end

  def init(_initial_data) do
    IO.puts "Started tower module!"
    level = 1
    damage = 5 + 5*level
    upgrade_cost = 50*level
    attributes = %{level: level, damage: damage, upgrade_cost: upgrade_cost}
    {:ok, attributes}
  end
```

Modulul conține o metoda ce furnizeaza informații legate de turn și una care sa permita evoluarea acestuia, contra unei sume de aur.

```
def get_tower_attributes() do
  GenServer.call(:tower_process, {:get_tower_attributes})
end

def upgrade_tower() do
  player_gold = Player.Stats.get_my_stats.gold
  upgrade_cost = get_tower_attributes().upgrade_cost
  case player_gold < upgrade_cost do
    true -> IO.puts "Not enough gold!"
    false -> upgrade(upgrade_cost)
  end
end
```

8) Wave GenServer

Acest modul conține informații legate de valul de monstri

```
defmodule TowerDefense.Monsters.Wave do
  use GenServer
  alias TowerDefense.Player.Stats

  def start_link do
    GenServer.start_link(__MODULE__, [], name: :wave_process)
  end

  def init(_initial_data) do
    IO.puts "Started monsters module!"
    level = 1
    monster = %{life: 10 + 5*level, reward: 10 + 5*level}
    monsters = [monster, monster, monster, monster, monster]
    wave = %{level: level, monsters: monsters}

    {:ok, wave}
  end
```

Modulul conține 3 metode importante:

- get_wave – întoarce informații legate de wave

- increase_level – crește dificultatea monstrilor

- decrease_life(amount) – reduce viața primului monstru

```elixir
def get_wave() do
  GenServer.call(:wave_process, {:get_my_wave})
end

def increase_level() do
  GenServer.call(:wave_process, {:increase_level})
end

def decrease_life(amount) do
  GenServer.call(:wave_process, {:decrease_life, amount})
end
```

9) Game module

Acest modul joaca rol de interfata. Prin intermediul acestuia utilizatorul poate interactiona cu jocul. Modulul pune la dispoziția utilizatorului 5 comenzi.

```elixir
defmodule TowerDefense.Game do
  alias TowerDefense.Player.Stats
  alias TowerDefense.Tower.Attributes
  alias TowerDefense.Monsters.Wave

  def help do
    IO.puts "
    Commands:
      .player - returns player info
      .tower - returns tower info
      .wave - return next wave info

      .upgrade - upgrades tower
      .send_wave - sends next wave
    "
  end
```

10) mix.exs

Acest fisier este alcătuit din 3 componente principale:

a) project - aici sunt pastrate diverse informaţii generale despre proiect (precum numele și versiunea proiectului, versiunea de elixir folosită)

```
def project do
  [
    app: :tower_defense,
    version: "0.1.0",
    elixir: "~> 1.6",
    start_permanent: Mix.env() == :prod,
    deps: deps()
  ]
end
```

b) application – aici este definit modulul ce va juca rol de Application

```
def application do
  [
    mod: {TowerDefense, []},
    extra_applications: [:logger]
  ]
end
```

c) deps – aici sunt definite dependintele proiectului nostru

```
defp deps do
  [
    # {:dep_from_hexpm, "~> 0.3.0"},
    # {:dep_from_git, git: "https://github.com/elixir-lang/my_dep.git", tag: "0.1.0"},
  ]
end
```

Repo github:

https://github.com/3x3cu7or/TowerDefense