- C se transforma in G
- G se transforma in C
- T se transforma in A

2. Avand la dispozitie un text si un cuvant numarati aparitiile cuvantului in acel text.

3. Acelasi lucru ca la 2, doar ca cuvintele se compara case insensitive ('a' este la fel ca 'A').

4. Ca la 2 si 3 doar ca se doreste un map care numara aparitiile fiecarui cuvant din text:

   Daca textul este `"Ana are mere, si uneori are si pere"` rezultatul este: `%{"Ana" => 1, "are" => 2, "mere" => 1, "si" => 2, "uneori" => 1, "pere" => 1}`

   Puteti folosi orice functie doriti din standard library String sau Enum

Tema 2 - Laborator 7 - 9 - Procese in Elixir

Deadline ~~11 Mai~~ 18 Mai 23:59.

Tema va reprezenta ansamblul de exercitii din laboratoarele 7, 8 si 9. Veti primi punctaj partial daca rezolvi numai pana la un punct.

Vom implementa acest laborator folosind elixir/erlang cu procese (actori).

Regatul este inca o data asaltat de dragon.

Eroii nostri au reusit sa-l ucida in laboratorul 5, insa folosind magie neagra cineva l-a reinviat, iar acum dragonul este un zombie. Regele a hotarat astfel sa trimita pe cineva cu cunostiinte despre zombie sa il infranga: un `necromancer`.

Necromancer-ul are un cantitate de viata `10 000`, iar dragonul o cantitate de viata `1 000 000`.

Fiecare din cei doi va reprezenta un proces separat.

Laborator 7:

1. Necromancer-ul stie o singura vraja: `Anti zombie bolt`, care loveste dragonul pentru o valoare aleatoare intre `0` si `1 000` cantitate de viata. Dragonul stie si el un singur atac: `Whiptail`, care loveste un adversar de-al sau pentru o valoare aleatoare intre `50` si `100` cantitate de viata.

   Fiecare din cei doi simuleaza aceste atacuri la infinit.

   Simulati lupta dintre cei doi. Afisati la final concluzia:

   - A castigat necromancer-ul si a ramas cu X cantitate de viata

- A castigat dragonul si a ramas cu Y cantitate de viata.

Spoiler alert: deocamdata castiga dragonul.

Pentru implementare se recomanda sa folositi 2 procese suplimentare: NecromancerStrategy si DragonStrategy care simuleaza strategia lor, iar Necromancer si Dragon ca procese vor implementa logica operatiilor.

2. Realitate e ca orice atac dureaza putin pentru a se intampla. `Anti zombie bolt` dureaza 12 milisecunde, iar `Whiptail` 5 milisecunde.

   Observati concluzia in situatia actuala (spoiler alert: inca castiga dragonul).

3. Necromancer-ul este un expert in zombie. Stie sa isi faca proprii lui `Zombie Knight` care pot incasa `Whiptail` in locul lui.

   Practic Necromancer mai cunoaste si vraja `Summon: Zombie Knight` care dureaza 20 de milisecunde si invoca un `Zombie Knight` cu `600` cantitate de viata. Fiecare `Zombie Knight` are un singur atac:

   - `Sword Slash` care dureaza 5 milisecunde si loveste dragonul pentru o valoare aleatoare intre `20` si `50` cantiate de viata.

   Acuma cand dragonul foloseste `Whiptail`, va lovi aleator unul dintre `Zombie Knight`s daca exista vreunul, sau pe `Necromancer` altfel.

4. Necromancer-ul si-a mai amintit o vraja: `Summon: Zombie Archer` care dureaza tot 20 de milisecunde si invoca un `Zombie Archer` cu `100` cantitate de viata. Arcasul are un singur atac: `Shot` care dureaza 10 milisecunde si loveste dragonul pentru o valoare aleatoare intre `100` si `200` cantitate de viata.

   Cand dragonul nu are niciun `Zombie Knight` sa atace, va ataca aleator fie pe `Necromancer` fie pe vreun `Zombie Archer`.

5. Dragon-ul devine mai puternic. La fiecare atac are 20% sansa ca in loc sa foloseasca `Whiptail`, sa foloseasca in schimb `Dragon Breath`.

   `Dragon Breath` loveste toti adversarii acestuia pentru o valoare aleatorie intre `50` si `150`.

Daca va plictisiti, adaptati-va strategia `Necromancer`-ului sa puteti bate un dragon cu cat mai multa cantitate de viata.

6. Practic pare ca atat `Necromancer`-ul, `Dragonul` cat si `Zombie` actioneaza ca niste servere. Ei sunt interogati de alte procese client (strategiilke corespunzatoare) si in functie de

operatie raspund inapoi cu ceva sau doar executa o actiune. Implementati in schimb aceste procese cu `GenServer`.

Pana acum daca exista erori probabil ati pornit procesele cu `spawn` / `GenServer.start` si acestea ar fi ignorate. De exemplu daca adaugati o eroare in strategia dragonului:

```
if :rand.uniform() < 0.1 do
    raise "Hopa, s-a stricat ceva"
end
```

Atunci la un moment dat dragonul s-ar opri din executat actiuni si nu am sti de ce.

Daca de exemplu le-ati fi pornit in schimb cu `spawn_link` / `GenServer.start_link` s-ar fi intamplat altceva: tunci cand se arunca o eroare, tot procesul s-ar opri. Evident niciuna din cele 2 variante nu pare perfecta. Cel mai bine ar fi daca atunci cand apare o eroare in vreun proces care exe uta strategia unui personaj, acesta sa se reporneasca automat.

Laborator 8:

7. Folositi un supervisor care sa urmareasca strategia `Necromancer`-ului si a `Dragonului` si sa le reporneasca automat daca acestea intampina o problema.

8. Acelasi principiu trebuie aplicat si proceselor personajelor `Zombie`. Prolema e ca `Supervisor` trebuie sa stie de la bun inceput ce copii urmareste pentru a-i restarta.

   `Zombie` sunt procese dinamice, si deci au nevoie de un `DynamicSupervisor`. Testati ca totul functioneaza corect, de exemplu folosind erori cu o anumite probabilitate in strategia unui `Zombie`.

Laborator 9:

9. Pana acum era responsabilitatea unui `Zombie` sa anunte pe `Dragon` ca a murit (pentru ca acesta sa il scoate din lista de "dusmani"). Daca insa se intampla o eroare pe undeva dragonul nu va fi notificat de moartea acestora si poate ii va ataca degeaba.

   Folositi `Process.monitor` pentru a urmari procesele `Zombie`. Veti primi mesaje pe care le puteti observa cu callback-ul `handle_info`. Atunci cand un process moare (din diferite motive) scoateti-l din lista de dusmani al dragonului.

10. Nu mai notificati `Dragon`-ul de creearea unui `Zombie`. De-acuma dragon-ul va lua un zombie ca pe un "adversar" la primul atac. Cu aceasta am decuplat zombie ca fiind responsabili de notificarea dragonului. Singura lor grija va fi sa il loveasca pe acesta.

11. Rulati dragon-ul si necromancer-ul (si aferentii acestuia) din procese de sistem separate.