UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,

RAČUNALNIŠTVO IN INFORMATIKO

Tomaž Piko

**Statična in dinamična analiza**

KPO – Vaja 8

Črna na Koroškem, april 2022

Kazalo vsebine

[1 Diagram poteka 3](#_Toc101940270)

[2 Statična analiza 4](#_Toc101940271)

[2.1 Dvakratna iteracija čez tabelo. 4](#_Toc101940272)

[2.2 Intervali simulacije 5](#_Toc101940273)

[2.3 Generacija začetnih entitet 6](#_Toc101940274)

[2.4 Iskanje poti 7](#_Toc101940275)

[3 Dinamična analiza 8](#_Toc101940276)

[3.1 Dvakratna iteracija čez tabelo 8](#_Toc101940277)

[3.2 Intervali simulacije 8](#_Toc101940278)

[3.3 Premiki živali 8](#_Toc101940279)

[3.4 Generacija začetnih entitet 9](#_Toc101940280)

Kazalo slik

[Slika 1.1: Diagram poteka 1. 4](#_Toc101941000)

[Slika 1.2: Diagram poteka 2. 4](#_Toc101941001)

[Slika 2.1: Funkcija obnovitve tabele prikaza. 5](#_Toc101941002)

[Slika 2.2: Funkcija simulacijskega koraka. 6](#_Toc101941003)

[Slika 2.3: Funkcija začetne generacije hrane. 7](#_Toc101941004)

[Slika 2.4: Funkcija iskanja naključne poti. 8](#_Toc101941005)

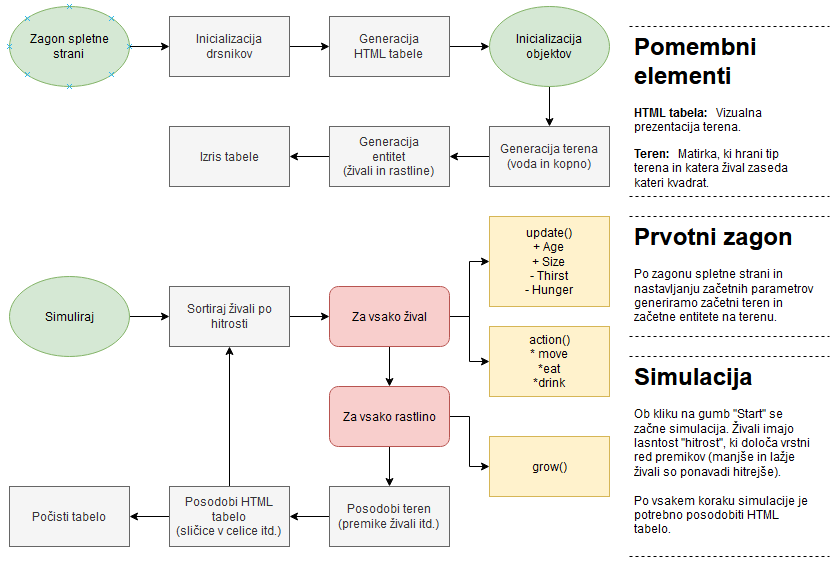
Kazalo tabel

[Tabela 3.1: Merjenje izboljšane časovne zahtevnosti. 9](#_Toc101941054)

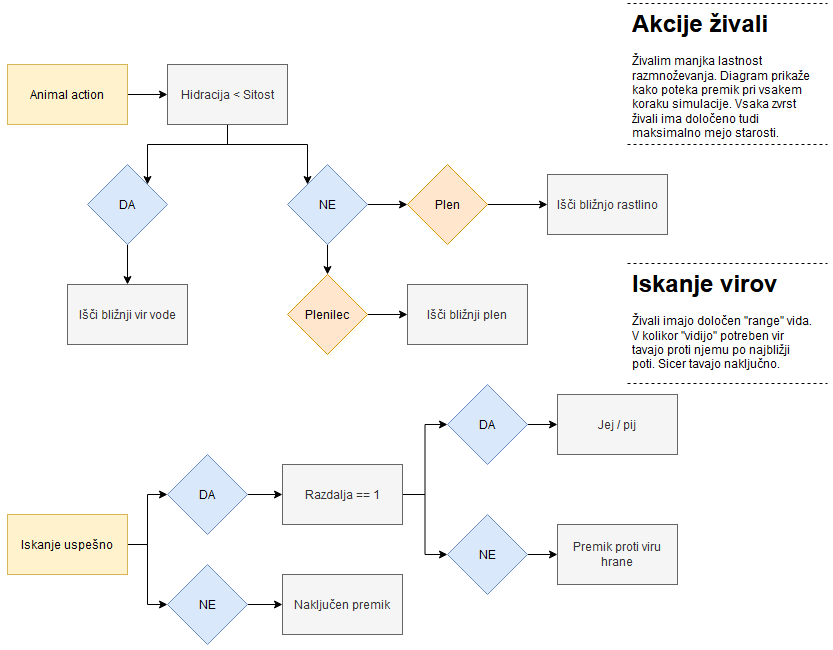
[Tabela 3.2: Časovna zahtevnost generacije začetnih entitet. 10](#_Toc101941055)

[Tabela 3.3: Časovna zahtevnost izboljšane generacije začetnih entitet. 10](#_Toc101941056)

# Diagrami poteka



Slika 1.1: Diagram poteka 1.



Slika 1.2: Diagram poteka 2.

# Statična analiza

## Dvakratna iteracija čez tabelo.

**Opis problema:**

Po vsakem koraku simulacije je potrebno posodobiti izrise v tabeli. Se pravi si koraki sledijo, izbris trenutnih sličic v celicah tabele po celotni tabeli in ponovna iteracija čez celotno tabelo, da se glede na teren vstavijo sličice v tabelo.

**Možna rešitev:**

Združitev obeh funkcij. Lahko nastopi problem pri mejah iteracije, saj je lahko teren večji od tabele, ki služi samo prikazu določenega »okenca« terena.



Slika 2.1: Funkcija obnovitve tabele prikaza.

## Intervali simulacije

**Opis problema:**

Ob kliku na gumb za začetek simulacije, se glede na določen interval (1500ms) kliče funkcija doSimulationStep(). Takšna implementacija je namenjena spremljanju dogajanja simulacije na tabeli. Saj interval omogoči, da so premiki uporabniku vidni ob hitrem delovanju računalnika. Problem nastopi ob dovolj nizkem času intervala in velikem številu entitet ter velikosti terena. Zgodi se lahko, da korak simulacije traja dlje časa kot sam interval in se nov korak simulacije kliče preden se prejšnji konča.

**Možne rešitve:**

1. Po vsakem zaključenem koraku simulacije, se izvede rekurzivni klic za nov korak vendar ta klic zakasnimo z funkcijo sleep().

+… Odpravimo možnost napake.

1. Simulacije se izvede do konca, korake pa prikazujemo naknadno. Dodatno lahko implementiramo premik levo in desno po korakih ter tako omogočimo uporabniku večji nadzor nad ogledom.

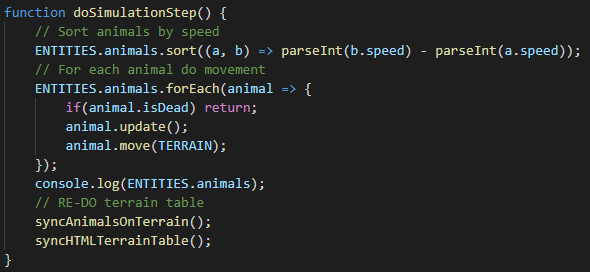
+… Odpravimo možnost napake.

+… Večji nadzor uporabnika nad ogledom simulacije

+… Pohitrimo samo simulacijo

-… Znatno povišana poraba pomnilnika za shranjevanje simulacije

-… Večja količina dodatnega dela



Slika 2.2: Funkcija simulacijskega koraka.

## Generacija začetnih entitet

**Opis problema:**

Funkcija poizkuša naljučno določiti pozicije n-entitet. V kolikor naljučna pozicija ni ustrezna (voda ali že zasedena pozicija) se šteje kot en poizkus. V primeru petih neuspelih poizkusov se funkcija prekine. Robni primer, do katerega načeloma nikoli ne pride, v primeru, da se zgodi pa simulacija z danimi parametri tako ni izvedljiva ampak se načeloma podaljša čas izvajanja.

**Možna rešitev:**

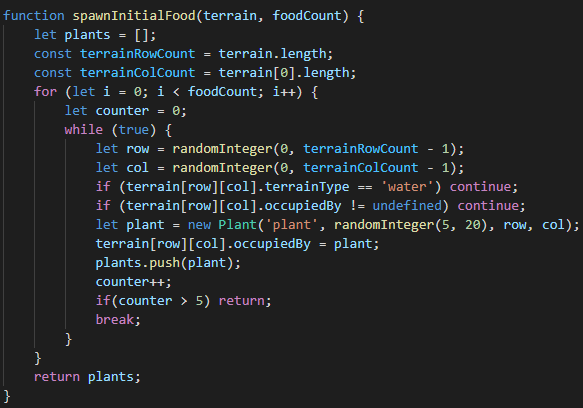
Beleženje »zemeljnih« pozicij. V kolikor velja (število plenilcov + plena + rastlin > števila pozicij) se funkcija zaustavi ali pa se števila prilagodijo. Po generaciji enega objekta na določeno pozicijo to pozicijo odstranimo iz nabora.

+… Hitrejše dodeljevanje pozicij. Nikoli ni potrebno naključno iskati ustrezno pozicijo saj so vse ustrezne.

+… Funkcija se prej zakluči, če simulacija ni mogoča.

-… Večja poraba (sicer zanemarljiva) pomnilnika v času generacije

-… V optimalnih pogojih in »srečo« pri iskanju naključnih pozicij, je trenutna implementacija še vedno hitrejša. (Testi v dinamični analizi)



Slika 2.3: Funkcija začetne generacije hrane.

## Iskanje poti

**Opis problema:**

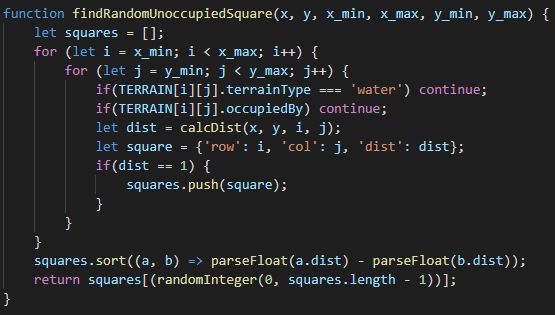
Ko plen ali plenilec išče vir hrane oziroma vode v svoji okolici najde vse možne vire hrane, v kolikor jih ne najde pa naključno mesto kamor se lahko premakne, vedno preišče vse možnosti in v primeru več enakovrednih rešitev naključno izbere eno.

**Možna rešitev:**

Odstranitev naključnega elementa. Premiki živali bodo 100 odstotno predvidljivi in odvisni od orientacije v katere pregledujemo okolico.

+… Pospešena simulacija

-… Predvidljiva simulacija, brez naključnosti.



Slika 2.4: Funkcija iskanja naključne poti.

# Dinamična analiza

## Dvakratna iteracija čez tabelo

Tabela 3.1: Merjenje izboljšane časovne zahtevnosti.

|  |  |
| --- | --- |
| **Dvakratna iteracija** | **Enkratna iteracija** |
| 5 ms | 4 ms |
| 6 ms | 4 ms |
| 5 ms | 5 ms |
| 5 ms | 5 ms |
| 6 ms | 4 ms |

**Rezultat:**

Zgornja tabela prikazuje merjenja časa izvajanja funkcije *syncHTMLTerrainTable()*, ki iz tabele prikaza izbriše in ponovno doda sličice živali oziroma rastlin ter barve ozadja. Z optimizacijo funkcije smo dosegli, z človeškega stališča, minimalno časovno izboljšavo.

Realistično gledano, pri tako majhnem število iteracij 25x25, kar je enako kot dimenzije prikazovalne tabele, optimizacija za enojno merjenje ni tako pomembna zaradi hitrosti trenutnih računalnikov. V kolikor pa upoštevamo, da se funkcija kliče na vsakem koraku simulacije in predpostavimo povprečno izboljšavo za 1 ms, postane izboljšava nekoliko bolj izrazita. V primeru tisoč korakov iteracije bi tako pridobili dobro sekundo kalkulacijskega časa, kar je za eno funkcijo zelo izrazito.

## Intervali simulacije

**Rezultat:**

Z omejenimi parametri, ki jih ponuja aplikacija, ni mogoče preseči 1500ms izvajanja funkcije *doSimulationStep()*.

## Premiki živali

Merjenje časovne zahtevnosti funkcije premikanja živali ni smiselno, saj je trajanje preveč odvisno od velikega števila drugih dejavnikov, kot so tip živali, njihove naključno določene lastnosti, kaj se nahaja v njihovi okolici itd.

## Generacija začetnih entitet

Generacija začetnih entitet je gnezdena funkcija treh praktično identičnih funkcij. Ena za generacijo plenilce, druga za plen ter tretja za rastline. Ker funkcije sledijo enaki logiki in je razlika v njihovi časovni odvisnosti odvisna le od števila določenih entitet, smo merjenje opravili kar na glavno funkcijo, ki zajema vse tri.

**Rezultat:**

Tabela 3.2: Časovna zahtevnost generacije začetnih entitet.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Časovna**  **zahtevnost** | **< 1ms** | **< 1ms** | **< 1ms** | **1 ms** | **1 ms** |

Časovna odvisnost pri tej funkciji lahko vidimo, da ni problem. Pri dodatnih testih smo ugotovil, da funkcija pade kadar ni dovolj mest za vse živali oziroma rastline. Zaradi tega razloga je bila implementirana izboljšava omenjena v poglavju 2.3 pri statični analizi. Pri spodnji tabeli vidimo, da po izboljšavah funkcija deluje isto hitro ampak ne pade ob velikih parametrih. Pri implementaciji smo žrtvovali minimalno povečano porabo pomnilnika.

Tabela 3.3: Časovna zahtevnost izboljšane generacije začetnih entitet.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Časovna**  **zahtevnost** | **< 1ms** | **1ms** | **< 1ms** | **1 ms** | **1 ms** |

# Zaključek

Aplikacijo smo uspeli dodelati s pomočjo statične in dinamične analize. Pri dinamični analizi smo potrdili vsa domnevanja predpostavljena pri statični analizi. Hkrati smo ugotovili, da nekatere izboljšave omenjene pri statični analizi niso nujno potrebne saj pridejo v poštev le pri skaliranju aplikacije.