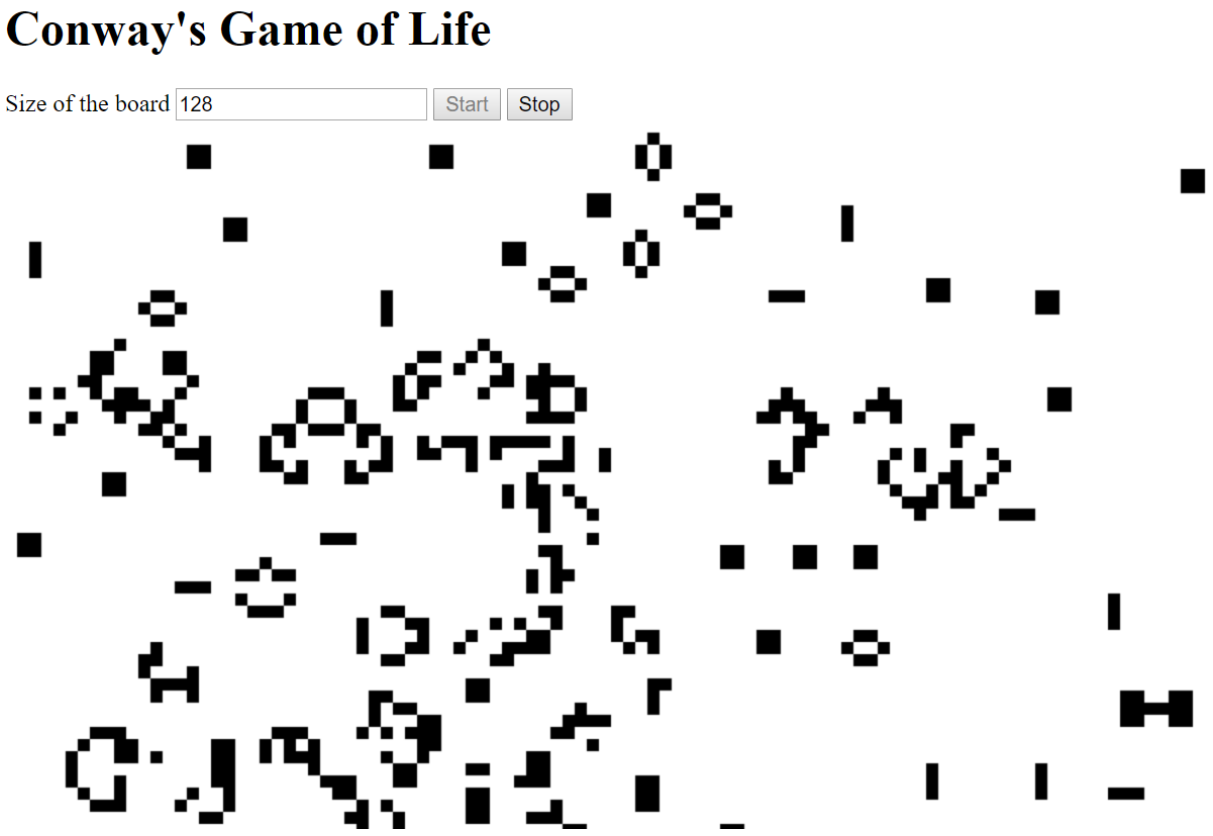
Teemu Heinämäki 503129

Conwayn elämäpelin toteutus

Yhteenveto

Työssäni toteutin selaimessa suoritettavan Conwayn elämäpelin. Toteutuksessa käytin F#-ohjelmointikieltä, joka muunnettiin JavaScript-kielelle käyttäen Fable nimistä muuntajaa.



Rajattu kuva ohjelmasta

Ohjelma pyörii Canvas-elementissä ja käyttäjä voi valita kuinka montaa solua simulaatiossa käytetään

- Lähdekoodi: https://github.com/Hippu/gameoflife
- Linkki peliin: https://hippuu.fi/life
- Fable-muuntaja: https://fable.io/

Conwayn Elämäpeli

Conwayn elämäpeli on nollan pelaajan soluautomaatti (cellular automaton). Peli sijoittuu n * m kokoiseen neliöruudukkoon, missä jokaisella ruudulla tai solulla on kaksi mahdollista tilaa: elävä tai kuollut. Peli etenee askel kerrallaan ja jokaisen ruudun seuraava tila määrittyy sen perusteella, kuinka monta viereistä solua on tällä hetkellä elossa. Pelin pointtina on osoittaa, kuinka yksinkertaisilla säännöillä voidaan rakentaa monimutkaista tai ainakin monimutkaiselta näyttävää toimintaa.

Solun tila	Elossa olevia naapureita	Seuraava tila
Elossa	0-1	Kuollut
Elossa	2-3	Elossa
Elossa	Enemmän kuin 3	Kuollut
Kuollut	3	Elossa
Kuollut	Muu kuin 3	Kuollut

F# JA FABLE-MUUNTAJA

Peli on toteutettu F#-ohjelmointikielellä, joka on vahvasti tyypitetty, ”ensisijaisesti funktionaalinen”, moniparadigmainen kieli. Tämä tarkoittaa, että vaikka ohjelmointikielellä on mahdollista ohjelmoida proseduraalisesti tai käyttää olioita ja luokkia, niin ohjelmointikieli ja sen rakenteet on kuitenkin suunniteltu funktionaalinen ohjelmointi mielessä pitäen. Myös peli on toteutettu tällä periaatteella, eli pääsijassa funktionaalisesti, mutta joitain osia on joko optimoitu tekemällä niistä enemmän proseduraalisia, tai mukautuen ulkoisten ohjelmarajapintojen (verkkoselaimen API:t) käyttämiin paradigmoihin.

Peli on kuitenkin tarkoitettu käytettäväksi verkkoselaimilla, jotka eivät suoraan tue F# ohjelmointikieltä tai sen käyttämää .NET-pohjaista ajoympäristöä, joten F#-koodi täytyy muuttaa jotenkin verkkoselaimien hyväksymään muotoon. Tähän käytän Fable-muuntajaa, joka muuntaa F#-koodin JavaScriptiksi. Tämä toimii hyvin itse kirjoitetun koodin osalta, mutta kaikkia .NET-rajapintoja, se ei pysty hyödyntämään. Tämän seurauksena jouduin kirjoittaman muutaman kaksiulotteisia taulukoita käsittelevän funktion.

Fable tuo myös verkkoselaimien rajapinnat, kuten tässä pelissä käytetyn Canvas-API:n käytettäväksi F#-kielesssä.

KOODILISTAUS

```
1   module ArrayF
2   /// Simple and naive operations for two-dimensional arrays
3   /// since Fable doesn't support the default multidimensional arrays
4
5   /// Initialize the two dimensional array with the given initializer function
6   let init (length1: int) (length2: int) (initializer: int → int → 'T) = // int → int → (int → int → 'T) → 'T [] []
7       [[ for x in 0 .. length1 →
8           [[ for y in 0 .. length2 → (initializer x y) ]]
9       ]]
10
11  /// Applies the function to the elements of the source array and puts the result in the target array
12  let mapiTo (transform: int → int → 'T → 'T) (source: 'T[][]) (target: 'T[][]) = // (int → int → 'T → 'T) → 'T [] [] → 'T [] [] → unit
13      for x in 0 .. source.Length - 1 do
14          let length = source.[x].Length
15          for y in 0 .. length - 1 do
16              target.[x].[y] ← transform x y source.[x].[y]
17
18  /// Sets all the values in the array
19  let setAll (value: 'T) (array: 'T[][]) = // 'T → 'T [] [] → unit
20      for x in 0 .. array.Length - 1 do
21          let length = array.[x].Length
22          for y in 0 .. length - 1 do
23              array.[x].[y] ← value
24
25  /// Iterate through all the cells in an array and give them as parameters to the 'action' function
26  let iteri (action: int → int → 'T → unit) (array: 'T[][]) = // (int → int → 'T → unit) → 'T [] [] → unit
27      for x in 0 .. array.Length - 1 do
28          for y in 0 .. array.[x].Length - 1 do
29              action x y array.[x].[y]
```

Ensimmäiseksi määritetään muutama apufunktio kaksiulotteiden taulukoiden käsittelemistä varten. F#:ssa taulukoita, jotka luodaan operaattorilla [| |] voidaan muuttaa ajon aikana. Peli käyttää

muutettavia tauluja, koska näin voidaan välttää ylimääräiset muistivaraukset, jotka tapahtuisivat, jos käytettäisiin ei-muutettavia listoja. Tämä nopeuttaa pelin suoritusta ja tekee siitä sulavamman.

```
type cellState =
    | Alive
    | Dead

let randomGen = System.Random() // System.Random

let emptyBoard arrayDimension = // int → cellState [] []
    ArrayF.init arrayDimension arrayDimension (fun _ _ → Dead)

let randomBoard arrayDimension = // int → cellState [] []
    ArrayF.init arrayDimension arrayDimension (fun _ _ →
        match randomGen.Next 5 with
        | 0  → Alive
        | _ → Dead )
```

Pelin käynnistyessä ruudukon tila määritetään satunnaisesti siten, että jokaisella ruudulla on 25% todennäkyisyys olla elossa. ”randomBoard”-funktio luo valitun kokoisen taulun satunnaisella sisällöllä.

```
/// Compute the next state of a cell in a location
let cellNextState arrayDimension x y (state: cellState[][]) = // int → int → int → cellState [] [] → cellState
    let cellState = state.[x].[y]
    match cellState with
    | Alive →
        match countAliveNeighboursOf arrayDimension x y state with
        | 0 | 1 → Dead
        | 2 | 3 → Alive
        | _ → Dead
    | Dead →
        match countAliveNeighboursOf arrayDimension x y state with
        | 3 → Alive
        | _ → Dead
```

Itse pelin säännöt on toteutettu ”cellNextState”-funktiossa, mikä ottaa argumentteina kaksiulotteisen cellState-taulun, taulun koon ja solun koordinaatit ja palauttaa kyseisen solun seuraavan tilan.

```
/// Scratch array for countAliveNeighbours of function
let neighbourArray = Array.init 8 (fun _ → (0, 0)) // (int * int) []
/// Scratch array for countAliveNeighbours of function
let targetArray = Array.init 8 (fun _ → Dead) // cellState []

/// Count the amount of alive neighbours of a cell in the given location
/// Uses the neighbourArray and targetArray arrays as scratch memory.
/// This is to optimize the program by minimizing memory allocations
let countAliveNeighboursOf arrayDimension x y (state: cellState[][]) = // int → int → int → cellState [] [] → int
    match (x, y) with
    | (0, _) | (_, 0) → 0
    | (x, y) when x = arrayDimension || y = arrayDimension → 0
    | _ →
      Array.set neighbourArray 0 (x-1, y-1)
      Array.set neighbourArray 1 (x-1, y)
      Array.set neighbourArray 2 (x-1, y+1)
      Array.set neighbourArray 3 (x, y-1)
      Array.set neighbourArray 4 (x, y+1)
      Array.set neighbourArray 5 (x+1, y-1)
      Array.set neighbourArray 6 (x+1, y)
      Array.set neighbourArray 7 (x+1, y+1)

      neighbourArray
      ▷ Array.iteri (fun i (x,y) → targetArray.[i] ← state.[x].[y])

      targetArray
      ▷ Array.sumBy (fun e →
          match e with
          | Alive → 1
          | Dead → 0
       )
```

Solujen naapurien määrän laskemiseen käytetään ”countAliveNeighboursOf”-funktiota. Kyseinen funktio käyttää välimuistina ”neighourArray” ja ”targetArray”-funktioita. Huomionarvoista on myös se, että mikäli tutkittava solu on ruudukon reunalla, niin naapureiden määräksi palautetaan nolla.

```
/// The app can be in two states:
/// Running with i * i dimensions or not running
type appStatus =
    | RunningWithDimensions of int
    | Stopped

/// The initial state of the app
let mutable app = Stopped // appStatus

/// The function that starts running the app
let runApp () = // unit → unit
    // Get the dimensions for the "board" that the app is running in
    let arrayDimension =
        match app with
        | RunningWithDimensions dimension → dimension
        | _ → 8
    let pixelMultiplier = 1024. / (float arrayDimension)

    let canvas = document.getElementsByTagName_canvas().[0]
    let ctx = canvas.getContext_2d()
    ctx.fillStyle ← !^"rgb(0, 0, 0)"

    // The application puts it states in these two two-dimensional arrays and swaps between them
    let mutable state = randomBoard arrayDimension
    let mutable nextState = emptyBoard arrayDimension

    /// Draw the current state of the application to the rendering context of the canvas
    let draw state (ctx: Browser.CanvasRenderingContext2D) =
        ctx.clearRect(0., 0., 1024., 1024.)
        ArrayF.iteri (fun x y state →
            match state with
            | Alive → ctx.fillRect (float x * pixelMultiplier, float y * pixelMultiplier, pixelMultiplier, pixelMultiplier)
            | Dead → ()
        ) state

    /// Update the state of the program according to the rules of 'Game of Life'
    let updateState () =
        ArrayF.setAll Dead nextState
        ArrayF.mapiTo (fun x y _ →
            cellNextState arrayDimension x y state
        ) state nextState

        let currentState = state
        state ← nextState
        nextState ← currentState

    /// Timer for when the state the app was last updated
    let mutable lastUpdate = 0.

    /// The callback for updating and drawing the state of the app
    let rec updateAndDraw (dt: float) =
        if (dt - lastUpdate) > 33.3 then
            lastUpdate ← dt
            updateState ()
            draw state ctx

        match app with
        | RunningWithDimensions _ → window.requestAnimationFrame updateAndDraw ▷ ignore
        | Stopped → ()

    window.requestAnimationFrame updateAndDraw ▷ ignore
```

Itse ohjelman suoritus tapahtuu ”runApp”-funktion sisällä. Ensimmäiseksi se hakee käyttäjän syöttämän koon peliruudukolle ja CanvasRenderingContext2D-instanssin (https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D), johon se voi piirtää pelin tilan. Lisäksi se määrittää kaksi muutettavaa peliruudukkoa ”state” ja ”nextState”. ”draw”-funktio piirtää pelin nykyisen tilan annettuun CanvasRenderingContext2D-instanssiin ja ”updateState” funktio päivittää seuraavan askeleen ”state”-peliruudukkoon. ”updateAndDraw” funktio on rekursiivinen funktio, joka kutsumalla itseään päivittää ja piirtää pelin tilan 33.3 millisekunnin välein. Se käyttää

apunaan ”requestAnimationFrame”-metodia (https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame).

```
/// Read the user inputted size for the board
let getDimensionSetting () = // unit → int
    let dimensionInput : HTMLInputElement = !!document.getElementById "arrayDimension"
    int dimensionInput.value

/// Set up the events for the start and stop button
let setUpEvents () = // unit → unit
    let startButton : HTMLButtonElement = !!document.getElementById "startButton"
    startButton.onclick ← fun _ →
        app ← getDimensionSetting () ▷ RunningWithDimensions
        runApp ()
        startButton.disabled ← true

    let stopButton : HTMLButtonElement = !!document.getElementById "stopButton"
    stopButton.onclick ← fun _ →
        app ← Stopped
        startButton.disabled ← false

setUpEvents ()
```

”getDimensionSetting”-funktio palauttaa käyttäjän syöttämän koon peliruudukolle ja ”setUpEvents”-funktio määrittää mitä tapahtuu, kun käyttäjä painaa Start- tai Stop-painiketta.

```
<!doctype html>
<html>
<head>
  <title>Game of Life</title>
  <meta http-equiv='Content-Type' content='text/html; charset=utf-8'>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="shortcut icon" href="fable.ico" />
</head>
<body>
  <h1>Conway's Game of Life</h1>
  <form onSubmit="return false;">
    <label for="arrayDimension">Size of the board</label> <input type="number" id="arrayDimension" value="128">
    <button id="startButton" type="button">Start</button>
    <button id="stopButton" type="button">Stop</button>
  </form>
  <canvas align="center" id="canvas" width="1024" height="1024"></canvas>
</body>
</html>
```

Itse verkkosivun HTML-koodi on hyvin yksinkertainen sisältäen laatikon, johon käyttäjä voi syöttää haluamansa, sekä Start- ja Stop-nappulat, jotka käynnistävät ja pysättävät pelin.

ASENNUSOHJEET

VAATIMUKSET

- Dotnet SDK 2.0 tai uudempi
- Node.js 6.11 tai uudempi
- Yarn: JS paketinhallintaohjelma
- Mono, jos käytetään Linuxia tai macOS:ää

Mene lähdekoodin juurikansioon ja aja komentorivillä

yarn

Asentaaksesi ohjelman riippuvuudet

Seuraavaksi aja komento

```
yarn build
```

Mikä luo kaikki tarvittavat tiedosto build/ kansioon, mistä ne voidaan siirtää minne tahansa verkkopalvelimelle.

Voit myös käyttää komentoa

```
yarn start
```

Mikä käynnistää paikallisen kehityspalvelimen, joka päivittyy automaattisesti koodia muokatessa.