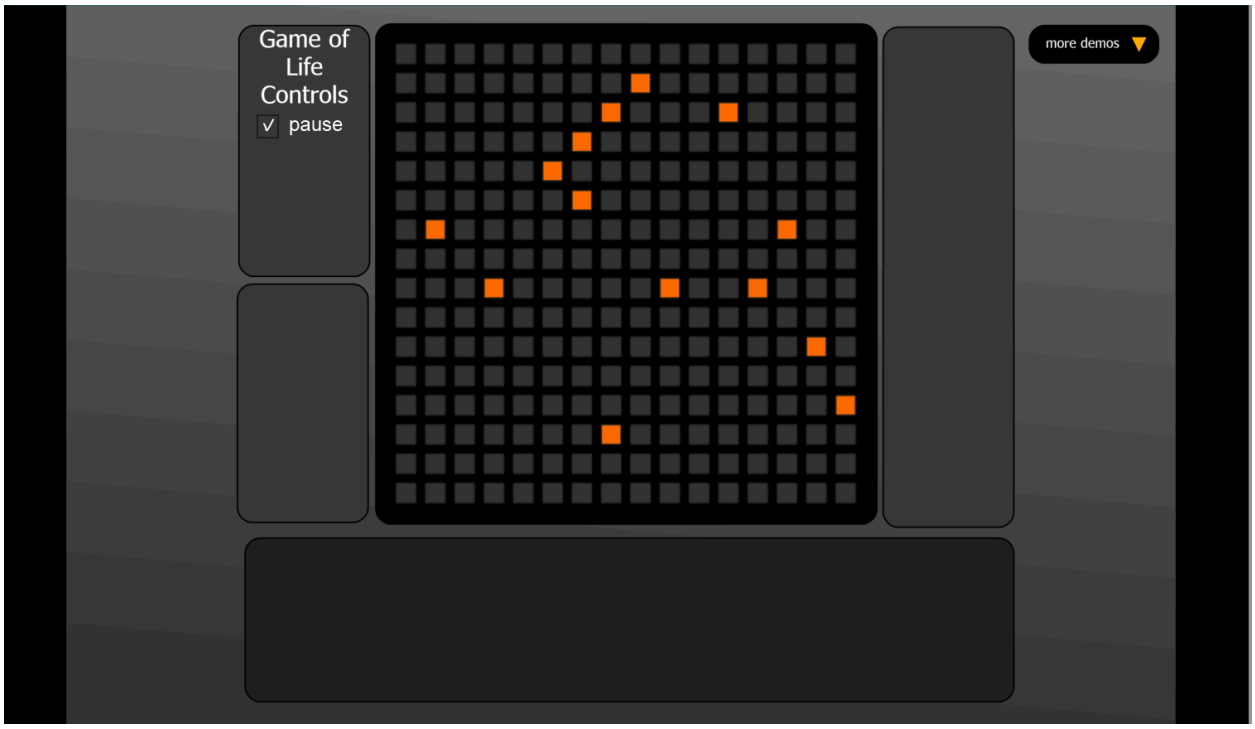
Zadanie z funkcyjnego programowania reaktywnego

Rafał Łukaszewski
3 lutego 2013

Rozwiązania proszę nadsyłać na adres solarplexus6@gmail.com. Termin: 17.02.2014.

1. Treść zadania

Zadanie polega na zaimplementowaniu w języku Elm wizualizacji Gry w życie. Interesuje nas wersja ograniczona do stałej siatki, np. rozmiaru 16 x 16. Można przyjąć dla uproszczenia, że poza widocznym polem komórki od razu umierają. Program powinien mieć zaszyty dowolnie przez was wybrany domyślny wzorzec i automatycznie uruchamiać symulacje po włączeniu. Polecam użycie jednego z charakterystycznych dla GoL wzorców, aby łatwiej było zweryfikować, że symulacja działa poprawnie (np Pulsar, albo Glider Gun).
Przykładowo taka wizualizacja może wyglądać tak jak na obrazku poniżej:



Oczywiście stylizacja interfejsu nie jest wymagana. Ważna jest tylko siatka wyświetlająca GoL. Rozwiązanie musi natomiast spełniać poniższe warunki:

1. Na ocenę 4:
    a. wyświetlanie symulacji GoL
    b. obecny musi być **`checkbox`**, dzięki któremu będziemy mogli pauzować symulacje
    c. przynajmniej jeden element interfejsu musi reagować w widoczny sposób na zmianę rozmiarów okienka roboczego (np. siatka wyświetlająca GoL może się skalować zgodnie z rozmiarami okna)
2. Na ocenę 5:
    a. musi być dostępna kontrolka, np. **`dropDown`** lub pole tekstowe, dzięki któremu możliwa będzie kontrola prędkości symulacji
    b. klikanie: możliwość aktywowania lub zabijania komórek przez kliknięcie w wybrane pole siatki (ta funkcjonalność powinna działać podczas gdy symulacja jest aktywna, jak również gdy jest zatrzymana)

Uwaga: bardzo bym prosił o przesyłanie w taki sposób przygotowanego rozwiązania, aby działało ono z marszu w http://elm-lang.org/try, aby można je było szybko sprawdzić.

2. Materiały

Wszystkie potrzebne materiały i dokumentację znajdziecie na http://elm-lang.org/. Znajduję się tam również wyczerpująca kolekcja przykładów, a także linki do grupy dyskusyjnej i namiary na irca. Szczególnie polecam dokumentację: Signal, Time, Graphics.Collage, Graphics.Input. Dostępny jest oddzielny kompilator języka, instalowany przez haskellowe narzędzie cabal, jeśli ktoś woli korzystać ze swojego ulubionego edytora - szczegóły na stronie. Niestety cywilizowany debugger nie jest jeszcze skończony, więc w większości przypadków trzeba się posiłkować przykładami i metodą prób i błędów.