λPP Paradigme de Programare

Racket: Flappy Bird

- Responsabili:
  - Vlad Neculae [mailto:neculae.vlad@gmail.com]
  - Teodor Szente [mailto:teodor98sz@gmail.com]
  - Mihnea Muraru [mailto:mmihnea@gmail.com]
  - Andrei Medar
  - Șerban-Ioan Ciofu
  - Gabriel Dănuț Matei

- Deadline soft: **03.04.2020**
- Deadline hard: 08.04.2020
- Data publicării: 18.03.2020
- Data ultimei modificări: 23.03.2020 changelog
- Data tester-ului: 24.03.2020
- Tema se va încărca pe **vmechecker [https://vmchecker.cs.pub.ro/ui/#PP]**
- Forum temă [https://acs.curs.pub.ro/2019/mod/forum/view.php?id=13581]

Obiective

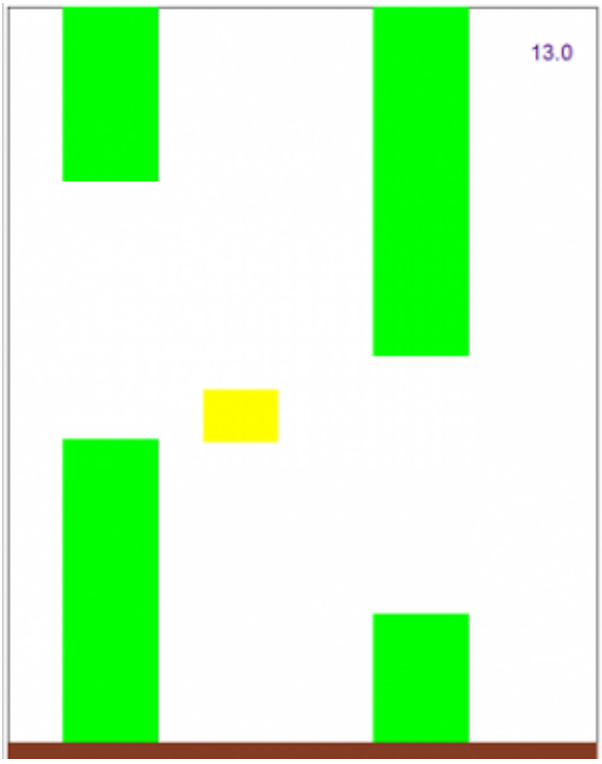
- Utilizarea mecanismelor **funcționale** din limbajul Racket pentru implementarea unei aplicații vizuale și interactive

Descriere

Tema urmărește implementarea unei variante simplificate a jocului *Flappy Bird [https://flappybird.io/]*, utilizând biblioteca *universe [https://docs.racket-lang.org/teachpack/2htdpuniverse.html]* a limbajului Racket. Aceasta permite realizarea de aplicații vizuale și interactive, implementate în stil **funcțional**.

Flappy Bird

Jocul presupune ghidarea unei **păsări**, reprezentate de dreptunghiul galben, pentru a putea trece prin fantele dintre **obstacolele** verzi, fără a se lovi de acestea sau de solul maro:



Gravitația acționează constant asupra păsării, iar utilizatorul îi poate imprima un **impuls în sus**, prin apăsarea unei taste. De asemenea, pasărea poate dobândi anumite **abilități** pe parcursul jocului, în momentul intersectării cu anumite obiecte.

Structuri în Racket

Deși veți avea libertate în alegerea modalității de reprezentare a entităților, putând recurge, spre exemplu, la listele standard, este puternic încurajată utilizarea **structurilor [https://docs.racket-lang.org/guide/define-struct.html]** din Racket. Acestea sunt similare structurilor din alte limbaje (vezi `struct` din C), și simplifică destul de mult manipularea construcțiilor complexe.

Găsiți un **tutorial** pentru utilizarea structurilor la sfârșitul acestei secțiuni.

Biblioteca universe

Biblioteca *universe* permite realizarea de aplicații vizuale și interactive, implementate în stil **funcțional**. Astfel, centrală este noțiunea de *stare* a aplicației, care poate fi **desenată** în forma unei imagini, sau **transformată** în starea următoare, în urma trecerii timpului sau a acțiunilor utilizatorului. Aceste prelucrări sunt surprinse, așa cum era de așteptat, prin **funcții**, care iau starea curentă ca parametru, și calculează rezultatele necesare.

Aveți la dispoziție un **tutorial** complet, în forma unei mici aplicații, care utilizează atât **structuri** Racket, cât și biblioteca *universe*. Parcurgeți-l cu atenție și observați valorile diverselor expresii din program.

Cerințe

Pornind la scheletul din secțiunea Resurse, implementați funcțiile din fișierele `main.rkt` (cerințele de bază) și `abilities.rkt` (pentru bonus), respectând indicațiile de mai jos și comentariile din fișiere, în **ordinea** dată de secvențele `TODO` (`TODO 1`, `TODO 2` etc.).

Gravitație și momentum

Gravitația acționează constant asupra păsării. Cu alte cuvinte, la fiecare cadru care trece, vitezei pe y a păsării i se adaugă valoarea gravității.

Atunci când vrem să imprimăm un impuls păsării, viteza pe y a păsării va fi înlocuită complet cu o valoare dată.

Sistemul de referință

Fiecare obiect din cadrul jocului (de la Flappy Bird, la pereți, abilități, chiar și pământ) are 2 coordonate - x și y - care îi descriu poziția pe ecran.

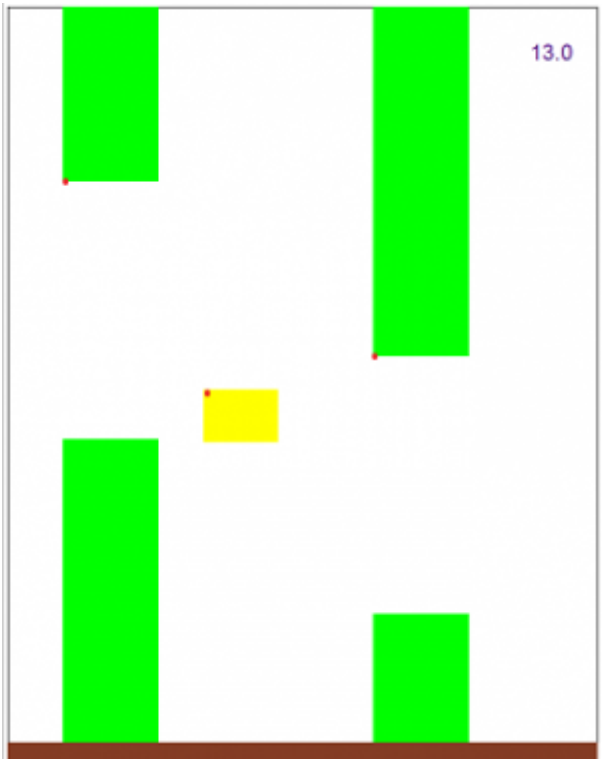
Evident, acestea depind de sistemul de referință ales, iar pentru cazul nostru, acesta este poziționat și orientat ca în următoarea imagine:

Astfel, creșterea valorii coordantei x va duce la apropierea de marginea dreaptă a ecranului, iar a coordonatei y-la apropierea de marginea de jos a ecranului.

Punctul în care ambele coordonate au valoarea 0 este colțul din stânga-sus al ecranului.

Reprezentarea obiectelor

Pentru a simplifica reprezentarea obiectelor în starea jocului, vom salva poziția *colțului din stânga sus* a fiecărui obiect. După cum știți, când suprapunem o imagine peste o alta, *coordonatele* pe care le specificăm drept poziția imaginii reprezintă *centrul* acesteia. Pentru a face translația dintre reprezentarea noastră și imaginea actuală, va trebui să translatăm coordonatele salvate în jos și la dreapta (adică crescător atât pe axa x, cât și pe y), cu jumătate din înălțimea, respectiv lățimea obiectului. Întrucât fiecare pipe este alcătuit din două componente, cea superioară și cea inferioară, fiind despărțite de acea fanta, este suficient să salvăm coordonatele fantei, urmând ulterior să adăugăm la afișare și la verificarea coliziunilor cele două componente.



Reprezentarea stării

- **bird**
  - **y** : poziția pe verticală
  - **v-y** : viteza pe verticală
- **pipe** (puteți alege o altă variantă în care rețineți cele doua pipe-uri și nu gap-ul)
  - **gap-y**: poziția gap-ului pe verticală
  - **x**: poziția pe orizontală
- **variables** (valorile care pot varia în funcție de abilități)
  - **gravity**: cât de mare este forța cu care pasărea este atrasă în jos
  - **momentum**: cât de mare este impulsul pe care îl primește pasărea cand se apasa space
  - **scroll-speed**: cât de repede se mișcă restul obiectelor pe scenă
- **abilities** (bonus)
  - **visible**: abilitățile care sunt vizibile pe scena (nu neapărat pe ecran, pot să fie la o distanța mai mare astfel încât să nu fie momentan vizibile pe ecran)
  - **active**: abilitățile active, a căror compunere modifică variabilele state-ului

Starea descrisă mai sus poate fi reprezentată oricum, însă este necesar să implementați gettere pentru fiecare element descris în stare, în vederea testării automate.

Exemplu:

```
(get-bird state)
; -> va întoarce o reprezentare internă aleasă de voi
(get-bird-y (get-bird state))
; -> va întoarce un număr reprezentând poziția păsării
```

Random

Oriunde folosiți funcția random trebuie sa includeți (require „random.rkt“) dacă nu este inclus deja, pentru a putea testa codul în checker.

Reprezentarea punctelor cu posn

Biblioteca de imagini din Racket folosește o structura pentru a reprezenta punctele numită `posn`. Aceasta contine doua campuri: x și y.

```
> (require lang/posn)
> (make-posn 2 3)
(posn 2 3)
> (posn-x (make-posn 2 3))
2
> (posn-y (make-posn 2 3))
3
```

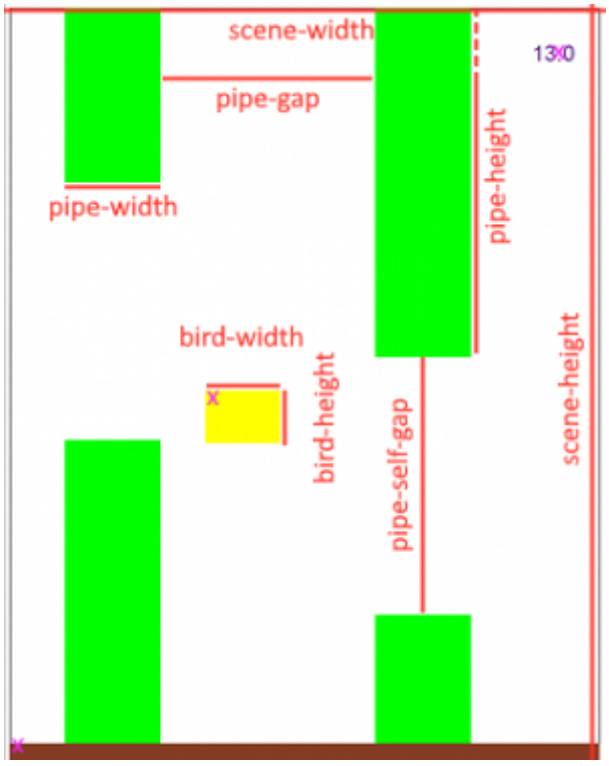
Această structură va apărea în schelet și în documentația pentru imagini, este recomandat sa o folosiți acolo unde este posibil.

Constante

Fișierul constants.rkt conține majoritatea constantelor globale folosite în construcția jocului. Mai jos se află o imagine care acopera ce reprezintă unele dintre acestea.

Observăm 3 X-uri roz, în colțurile din stânga sus ale pământului și păsări, și în centrul textului scorului. Punctul (**`bird-x`**, **`bird-initial-y`**) reprezintă coordonatele colțului din stânga sus al păsării, punctul (0, **`ground-y`**), coordonatele colțului din stânga sus ale pământului, iar (**`text-x`**, **`text-y`**), coordonatele *centrului* textului. Nu va fi nevoie să translatati imaginea textului.

De asemenea, avem o constantă pentru înălțimea unui pipe, **`pipe-height`** (aceeași atât pentru pipe-urile superioare, cât și pentru cele inferioare, chiar dacă depășesc dimensiunile scenei), înălțimea pământului, **`ground-height`**, o constantă pentru gravitație, **`gravity`**, una pentru momentum, cu acest nume, și una pentru viteza obiectelor, **`initial-scroll-speed`**.



Tratarea coliziunilor

În cadrul implementării temei vă va fi utilă o funcție care să verifice dacă două dreptunghiuri se intersectează. O idee de implementare pleacă de la *următoarea simulare [https://silentmatt.com/rectangle-intersection/]*. Oferim o implementare a acestei idei in functia **`check-collision-rectangles`**, care primeste ca parametri cele 4 colturi ale dreptunghiurilor.

Abilitați (bonus, 20p)

Pentru bonus ne dorim să implementăm un sistem generic de a introduce diverse abilități în joc. O abilitate va trebui să conțină:

- **image**: o imagine care va fi afișată în joc
- **time**: cat timp durează abilitatea
- **pos**: poziția abilității pe scenă
- **next**: o funcție care primește variables din state și le modifica într-un anumit fel.

La fiecare pas jocul trebuie sa aplica funcția next pentru fiecare abilitate activă și astfel să obțină variables. O abilitate este activă dacă a avut in trecut o coliziune cu pasărea și încă nu a expirat.

Abilitățile își vor produce efectul începând cu următorul cadru după ce au devenit active.

Pentru a poziționa abilitățile pe scenă puteți folosi funcția (random-position POSITION_RANGE) care întoarce **centrul** abilității.

Precizări

- Încercați să exploatați la maxim construcțiile de limbaj învățate: **funcții** ca valori, **funcționale**, construcții `let`. Un nivel foarte redus sau inexistent de utilizare a **funcționalelor** va conduce la o **depunctare** de 10p din 100.
- Nu aveți voie sa folosiți **set!** sau alte proceduri care au efecte laterale.

Resurse

- **tutorial**
- **demo (linux)**
- **demo (windows)**
- **schelet+checker**

Changelog

- 04.04.2020 - Prelungire deadline
- 24.03.2020 - Clarificare enunț abilități
- 25.03.2020 - Fix test 7 checker + cerință
- 25.03.2020 - Checker+reference images update
- 24.03.2020 - Checker update+comentarii pentru bonus reperate
- 24.03.2020 - Adăugare checker+schelet nou
- 21.03.2020 - În **main.rkt**, redenumit **bird** în **bird-image**
  - În **main.rkt**, redenumit **ground** în **ground-image**
  - Adăugat mențiuni despre efecte laterale
- 22.03.2020 - Modificat TODO-ul corespunzător lui **next-state-bird**. Această funcție va modifica atât v-y, cât și y-ul păsării.
- 23.03.2020 - Modificat TODO-urile 6 si 11. Viteza pe y a pasarii va fi inlocuita cu -momentum, si move-pipes va scadea din x-ul fiecarui pipe scroll-speed-ul dat.

20/teme/racket-flappy-bird.txt · Ultima modificare: 2020/04/17 14:56 de către Mihnea Muraru