Acting Shooting Star: Documentation

Version 7.6

Olivier Nguyen,
Hicham Zghari,
Augustin Gauchet,
and Imad Boudroua

May 13, 2020

Contents

1 Actor

1.1 Structure

```
(struct actor (position mailbox name category)
    #:extra-constructor-name make-actor)
  position : list?
  mailbox : list?
  name : any?
  category : any?
```

La structure qui décrit les acteurs.

- position est une liste de deux éléments qui contient les coordonnées x et y
- mailbox est une liste de messages
- name? contient la représentation raart de l'acteur
- category est le type de l'acteur (joueur, ennemi, projectile)

message est une liste dont le contenu varie selon le type de message : Pour les messages de mouvement "message" prend la forme (move '(position)) avec position le vecteur coordonnée qui va faire bouger l'acteur qui reçoit le message Pour les messages de création d'acteur "message" prend la forme (create '(position)) avec position les coordonnées où est créé l'acteur

1.2 Fonctions et prédicats

```
(name-of-actors actor) → name
  actor : (actor?)
```

Retourne le nom de l'acteur

```
(actor-location actor) → (listof number)
  actor : (actor?)
```

retourne la liste de coordonnées de l'acteur

```
(actor-send actor-send message) → actor
  actor-send : (actor?)
  message : (message?)
```

renvoie une copie de l'actor avec le message ajouté à sa mailbox

```
(x-pos-top-mail actor) → entier
  actor : actor?
```

Si le message en première position de la mailbox de l'actor est un message de mouvement, renvoie la composante horizontale du mouvement en question

```
(y-pos-top-mail actor) → entier
  actor : actor?
```

Si le message en première position de la mailbox de l'actor est un message de mouvement, renvoie la composante verticale du mouvement en question

```
(actor-update actor) → (listof actor)
  actor : actor?
```

Retourne une liste dont le premier élément contenant à la fois une copie de l'acteur initial qui a effectué toutes les instructions contenues dans sa mailbox, et les actors créés par les messages de création.

```
(colliding? actor-1 actor-2) → boolean
  actor-1 : actor?
  actor-2 : actor?
```

Retourne `#t` si les deux actors ont les mêmes coordonnées, `#f` sinon

```
(collisions? actor actors) → boolean
  actor : actor?
  actors : (listof actors?)
```

Retourne `#t` s'il existe un actor dans la liste qui a les mêmes coordonnées que l'acteur passé en paramètre, `#t` sinon

2 World

2.1 Structure

```
(struct world (actors)
    #:extra-constructor-name make-world)
  actors : (listof actors?)
```

actors est la liste des acteurs intervenants dans le monde.

2.2 Fonctions et prédicats

```
(send-world world message category) → world
  world : world?
  message : message?
  category : any?
```

renvoie le monde dont chaque acteur de la catégorie spécifiée reçoit le nouveau message dans leur mailbox

```
(update-world world) → updated-world
  world : world?
```

renvoie une copie du monde où tous les acteurs ont été mis à jour

```
(save-world world) → void
  world : world?
```

la liste contenant les derniers worlds est modifié par effet de bord, le monde est sauvegardé par ordre du plus récent au moins récent de gauche à droite. Si la taille de la liste est égale à 20, le monde le plus ancien de la liste est effacé.

```
(world-travel n current-world) → world?
  n : number?
  current-world : world?
```

le monde qui correspond au n-ième dernier monde parcouru. Si n >= 20 ou n < 0, on renvoie *current-world*.

```
(actor-alive? x w) → boolean
  x : actor?
  w : world
```

renvoie `#t` si l'acteur x n'entre pas en collision avec d'autres acteurs, `#f` sinon.

```
(player-dead? w) → booleen
  w : world
```

renvoie `#t` si il n'y a aucun `actor` de catégorie "player" dans le `world` w, `#f` sinon.

```
(world-alive world) → world
  world : world?
```

Renvoie un monde qui n'a que des acteurs vivants

```
(execute-msg world) → world
  world : world?
```

Renvoie un monde dont les acteurs ont été mis à jour, le monde ne contenant que des acteurs vivant

```
(world-filter world filter) → world
  world : world?
  filter : procedure?
```

Renvoie un monde dont les acteurs vérifient les conditions imposées par la fonction filtre.

```
(shoot world) → world
  world : world?
```

Renvoie un monde avec un nouveau acteur de type "missile" créé par le player. Generate?

```
(generate tick) → list
  tick : any?
```

Renvoie une liste d'acteurs de type "enemy". Selon la congruance de tick modulo 3, les acteurs sont générés aléatoirement.

3 Runtime

```
(struct runtime (world-fps world-event world-output world-tick)
    #:extra-constructor-name make-runtime)
  world-fps : number?
  world-event : any?
  world-output : any?
  world-tick : number?
```

- world-fps initialise le nombre d'images par seconde que le jeu est censé afficher
- world-event gère les événements qui se produisent et les interactionsavec les périphériques extérieurs
- world-output permet d'afficher les éléments de la bibliothèque raart sur le terminal
- world-tick s'occupe de mettre à jour la nouvelle runtime après chaque unité de temps

Le jeu se lance avec la fonction `(start-application)`.