

Space Race

PROGRAMACIÓN DECLARATIVA
4º CURSO PRIMER CUATRIMESTRE
INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE CÓRDOBA

PEDRO JESÚS GÓMEZ MOLINA
CÓRDOBA, 15 DE DICIEMBRE DE 2020

Índice

- ❑ [Introducción](#)
- ❑ [Fundamentos teóricos](#)
- ❑ [Descripción del código](#)
- ❑ [Resultados](#)
- ❑ [Conclusiones](#)
- ❑ [Bibliografía](#)

Introducción

El juego en el que se ha basado este trabajo se trata de **Space Race**, un juego de Atari lanzado en el año 1973.

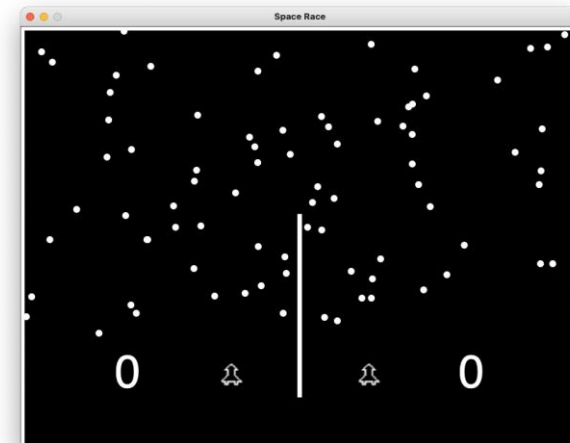
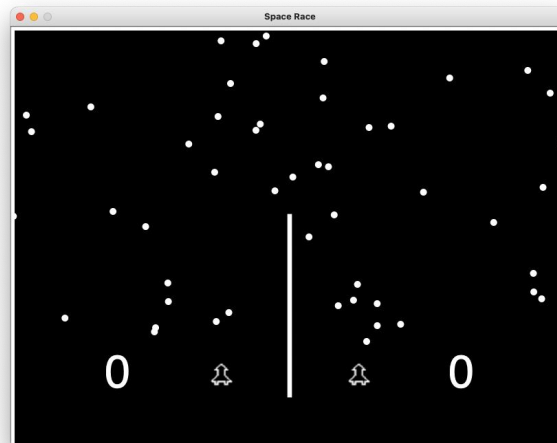
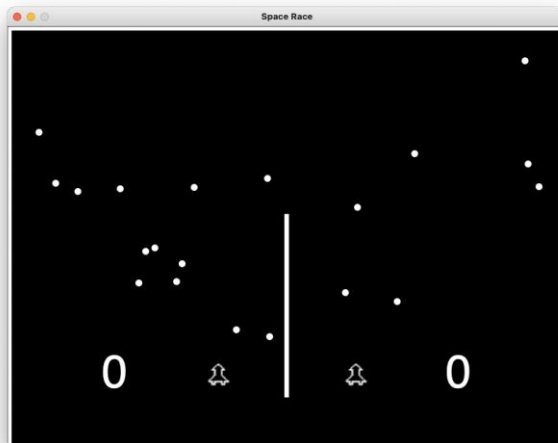


Es un juego para **2 jugadores**, en el que cada jugador controla una nave que debe llevar hasta el borde superior de la pantalla **esquivando los meteoritos** que van apareciendo.

Los jugadores sólo pueden moverse **verticalmente**, pudiendo avanzar o retroceder.

Cada vez que se llega al borde superior de la pantalla, se suma un punto al marcador del jugador. Gana el jugador que antes consiga la **puntuación máxima**.

Al juego se le han añadido niveles de dificultad, que cambian el **número de meteoritos total** en pantalla y la **velocidad de estos**.



Fundamentos Teóricos

Para programar el juego, se ha utilizado el lenguaje **Scheme** en su dialecto “Muy Grande”.

A fin de no reinventar la rueda, se decidió hacer uso de la librería *2HTDP* para las mecánicas básicas del juego. Gracias a esta librería se puede:

- Dibujar imágenes en pantalla
- Detectar pulsaciones de teclas en todo momento
- Actualizar el estado del juego (y la salida por pantalla) un número determinado de veces por segundo
- Determinar cuando se debe finalizar el juego

El menú principal se creó haciendo uso de las funciones y elementos básicos de *racket/base*.

Para los elementos del juego que varían, se crearon los **tipos abstractos de datos** *World*, *Player* y *Meteor*, junto a sus funciones de acceso.

Descripción del código


El código del juego se encuentra dividido en dos archivos .rkt:

- juego.rkt
 - Constantes
 - World
 - Player
 - Meteor
 - Movimiento
 - Imagen
 - 2htdp

- main.rkt
 - Ventana principal
 - Título
 - Controles
 - Dificultad
 - Resolución
 - Score
 - Botones principales

Constantes

```
(require 2htdp/universe 2htdp/image)
(require racket/base)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  CONSTANTS
;;
(define SHIP ) ;; Imagen del jugador

(define WIDTH 800) ;; Ancho de la pantalla
(define HEIGHT 600) ;; Alto de la pantalla
(define SCALE (/ HEIGHT 900)) ;; Escala en función del tamaño de la pantalla
(define INITIAL-Y (- HEIGHT (/ HEIGHT 6))) ;; Posición inicial de los jugadores en el eje Y
(define PLAYER1-INITIAL-X (+ 100 (/ WIDTH 4))) ;; Posición inicial del jugador 1 en el eje X
(define PLAYER2-INITIAL-X (- WIDTH PLAYER1-INITIAL-X)) ;; Posición inicial del jugador 2 en el eje X
(define Y-DELTA 20) ;; Velocidad de movimiento
(define MAX-METEORS 50) ;; Número máximo de meteoritos simultáneos
(define MIN-METEOR-SPEED 2) ;; Velocidad mínima de los meteoritos
(define MAX-SCORE 10) ;; Puntuación máxima
```

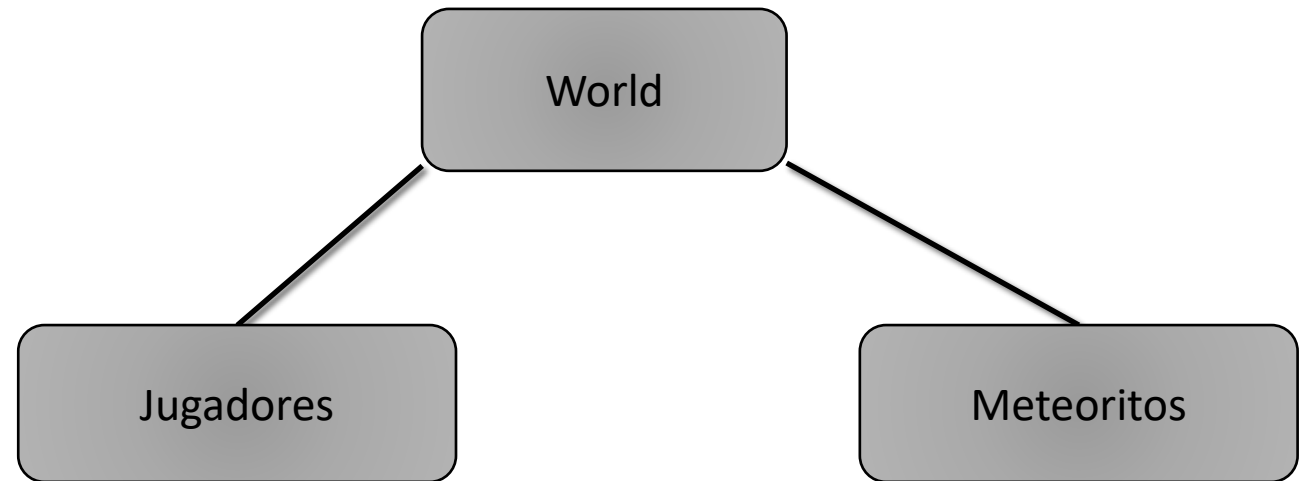
World

El uso de las librerías *2htdp* nos obliga a tener un elemento *World* (Mundo), en el que se deben contener todos los elementos que van cambiando en cada fotograma.

Se optó por representar este elemento como una **lista asociativa** que contiene dos sublistas.

- Jugadores
- Meteoritos

Así, en cada actualización de fotograma se genera un nuevo elemento *World* que sustituye al anterior.



juego.rkt

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Nombre: world
;; Objetivo: tipo abstracto de datos world
;; Descripción:
;;     Se crea el tipo abstracto de datos world usando listas de asociación, junto a sus funciones de acceso
;; Parámetros:
;;     player-1: primer jugador
;;     player-2: segundo jugador
;;     meteors: lista de meteoritos que hay actualmente
;;
;; Función de creación
(define (create-world player-1 player-2 meteors)
  (list (list 'player-1 player-1) (list 'player-2 player-2) (list 'meteors meteors)))
|
;; Funciones de acceso

(define (get-player-1 world)
  (cadr (assoc 'player-1 world)))

(define (get-player-2 world)
  (cadr (assoc 'player-2 world)))

(define (get-meteors world)
  (cadr (assoc 'meteors world)))

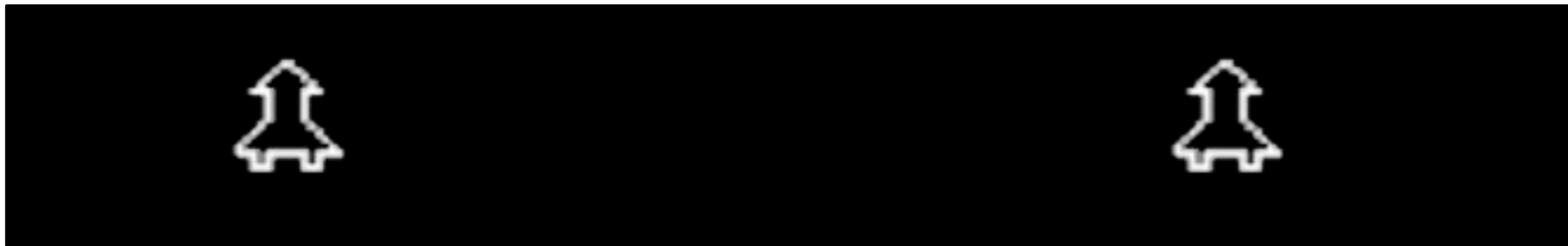
```

Jugadores

Cada jugador tiene tres atributos:

- Posición en el eje X
- Posición en el eje Y
- Puntuación actual

Para simplificar la solución final, se eligió codificar un tipo de datos abstracto *Player* donde se almacenasen los atributos de cada jugador.



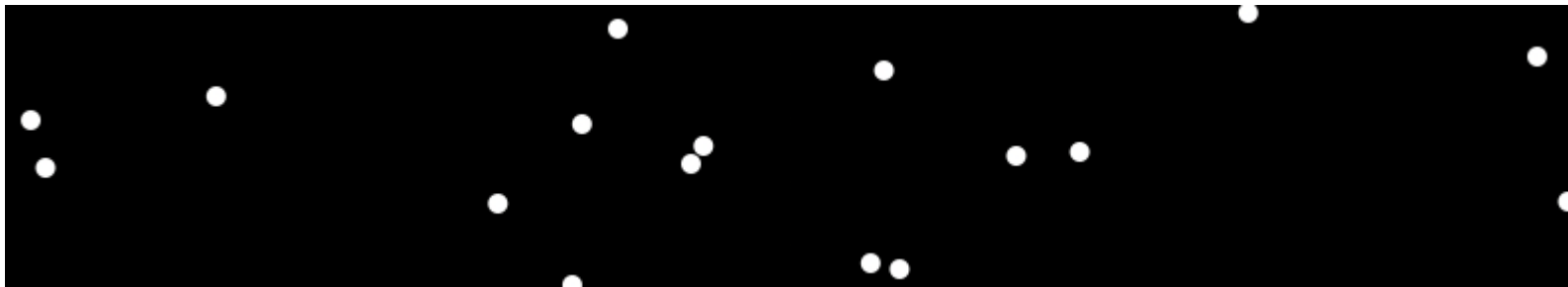
```
;;;;;;;;;;;;;;  
;;  
;; Nombre: player  
;; Objetivo: tipo abstracto de datos player  
;; Descripción:  
;;     Se crea el tipo abstracto de datos player usando listas de asociación, junto a sus funciones de acceso  
;; Parámetros:  
;;     x: posición del jugador en el eje x  
;;     y: posición del jugador en el eje y  
;;     score: puntuación del jugador  
;;  
  
;; Función de creación  
(define (create-player x y score)  
  (list (list 'x x) (list 'y y) (list 'score score)))  
  
;; Funciones de acceso  
  
(define (get-x player)  
  (cadr (assoc 'x player)))  
  
(define (get-y player)  
  (cadr (assoc 'y player)))  
  
(define (get-score player)  
  (cadr (assoc 'score player)))
```

Meteoritos

En el caso de los meteoritos también nos encontramos con tres atributos principales:

- Posición en el eje X
- Posición en el eje Y
- Velocidad (es constante a partir de su creación)

De forma que, al igual que con los jugadores, se creó el tipo abstracto *Meteor* junto a sus funciones de acceso correspondientes.



```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Nombre: meteor
;; Objetivo: tipo abstracto de datos meteor
;; Descripción:
;;   Se crea el tipo abstracto de datos meteor usando listas de asociación, junto a sus funciones de acceso
;; Parámetros:
;;   x: posición del meteorito en el eje x
;;   y: posición del meteorito en el eje y
;;   direction: dirección del meteorito en el eje x
;;   speed: velocidad del meteorito en el eje x
;;

;; Función de creación
(define (create-meteor x y direction speed)
  (list (list 'x x) (list 'y y) (list 'direction direction) (list 'speed speed)))

;; Función de creación con valores aleatorios
(define (create-random-meteor)
  (if (= (random 2) 0)
      (create-meteor 0 (random 0 (- INITIAL-Y 40)) "right" (+ (random) MIN-METEOR-SPEED))
      (create-meteor WIDTH (random 0 (- INITIAL-Y 40)) "left" (+ (random) MIN-METEOR-SPEED))))

;; Función de creación de meteoritos iniciales
(define (create-random-initial-meteor)
  (if (= (random 2) 0)
      (create-meteor (random 0 WIDTH) (random 0 (- INITIAL-Y 40)) "right" (+ (random) MIN-METEOR-SPEED))
      (create-meteor (random 0 WIDTH) (random 0 (- INITIAL-Y 40)) "left" (+ (random) MIN-METEOR-SPEED))))

;; Función de creación de la lista de meteoritos inicial
(define (create-initial-meteors meteors)
  (if (= (length meteors) MAX-METEORS)
      meteors
      (create-initial-meteors (cons (create-random-initial-meteor) meteors))))

;; Funciones de acceso

(define (get-x meteor)
  (cadr (assoc 'x meteor)))

(define (get-y meteor)
  (cadr (assoc 'y meteor)))

(define (get-direction meteor)
  (cadr (assoc 'direction meteor)))

(define (get-speed meteor)
  (cadr (assoc 'speed meteor)))
```

Movimiento

Durante el juego, hay dos elementos que están en movimiento:

- Jugadores
- Meteoritos

El movimiento de todos los elementos se apoya en las funcionalidades de la librería *2htdp*; así, las funciones de movimiento de los jugadores se ejecutarán cuando se detecten **pulsaciones de teclado**, y las de los meteoritos en cada **actualización de fotograma**.

Para realizar los movimientos, se varía la posición del elemento en una **cantidad fija** en el caso de los jugadores y **variable** para cada meteorito del juego, y **se vuelve a generar** ese elemento con su posición alterada.

Los meteoritos sólo se desplazan en el eje X y los jugadores en el eje Y.

Movimiento jugadores

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Funciones de movimiento

(define (move-player1-up world)
  ;; Se comprueba si el jugador ha llegado al límite superior
  (if (<= (get-y (get-player-1 world)) 0)
      ;; En caso afirmativo, se suma 1 a la puntuación del jugador y se coloca en la posición inicial
      (create-world (create-player PLAYER1-INITIAL-X INITIAL-Y (+ (get-score (get-player-1 world)) 1)) (get-player-2 world) (get-meteors world))
      ;; En el caso contrario, se resta Y-DELTA a su posición en el eje Y (se mueve hacia arriba)
      (create-world (create-player PLAYER1-INITIAL-X (- (get-y (get-player-1 world)) Y-DELTA) (get-score (get-player-1 world))) (get-player-2 world) (get-meteors world))))

(define (move-player1-down world)
  ;; Se comprueba si el jugador ha llegado al límite inferior
  (if (>= (get-y (get-player-1 world)) INITIAL-Y)
      ;; En caso afirmativo, se mantiene en la misma posición
      (create-world (create-player PLAYER1-INITIAL-X INITIAL-Y (get-score (get-player-1 world))) (get-player-2 world) (get-meteors world))
      ;; En el caso contrario, se suma Y-DELTA a su posición en el eje Y (se mueve hacia abajo)
      (create-world (create-player PLAYER1-INITIAL-X (+ (get-y (get-player-1 world)) Y-DELTA) (get-score (get-player-1 world))) (get-player-2 world) (get-meteors world))))

(define (move-player2-up world)
  ;; Se comprueba si el jugador ha llegado al límite superior
  (if (<= (get-y (get-player-2 world)) 0)
      ;; En caso afirmativo, se suma 1 a la puntuación del jugador y se coloca en la posición inicial
      (create-world (get-player-1 world) (create-player PLAYER2-INITIAL-X INITIAL-Y (+ (get-score (get-player-2 world)) 1)) (get-meteors world))
      ;; En el caso contrario, se resta Y-DELTA a su posición en el eje Y (se mueve hacia arriba)
      (create-world (get-player-1 world) (create-player PLAYER2-INITIAL-X (- (get-y (get-player-2 world)) Y-DELTA) (get-score (get-player-2 world))) (get-meteors world))))

(define (move-player2-down world)
  ;; Se comprueba si el jugador ha llegado al límite inferior
  (if (>= (get-y (get-player-2 world)) INITIAL-Y)
      ;; En caso afirmativo, se mantiene en la misma posición
      (create-world (get-player-1 world) (create-player PLAYER2-INITIAL-X INITIAL-Y (get-score (get-player-2 world))) (get-meteors world))
      ;; En el caso contrario, se suma Y-DELTA a su posición en el eje Y (se mueve hacia abajo)
      (create-world (get-player-1 world) (create-player PLAYER2-INITIAL-X (+ (get-y (get-player-2 world)) Y-DELTA) (get-score (get-player-2 world))) (get-meteors world))))

```

Movimiento meteoritos

```
;; Función para mover un meteorito
(define (move-meteor meteor)
  (cond
    ;; Si el meteorito ha salido de los límites de la pantalla, se elimina
    ((or (> (get-x meteor) WIDTH) (< (get-x meteor) 0)) null)
    ;; Si la dirección del meteorito es izquierda, se le resta 1 a su posición en el eje X (se mueve hacia la izquierda)
    ((string=? (get-direction meteor) "left") (create-meteor (- (get-x meteor) (get-speed meteor)) (get-y meteor) (get-direction meteor) (get-speed meteor)))
    ;; En otro caso se le suma 1 a su posición en el eje X (se mueve hacia la derecha)
    (else (create-meteor (+ (get-x meteor) (get-speed meteor)) (get-y meteor) (get-direction meteor) (get-speed meteor))))))

;; Función para mover todos los meteoritos de la lista
(define (move-meteors meteors)
  ;; Se comprueba si hay actualmente algún meteorito
  (if (null? meteors)
      meteors
      (let (
        ;; Declaración de variables
        (new-meteor (move-meteor (car meteors)))) ;; Meteorito tras moverse
        ;; Cuerpo del let
        (if (null? new-meteor)
            ;; Si el primer meteorito de la lista se elimina al moverse, creamos uno nuevo y lo unimos al resto
            (cons (create-random-meteor) (move-meteors (cdr meteors)))
            ;; En otro caso, unimos el meteorito ya movido al resto de la lista
            (cons new-meteor (move-meteors (cdr meteors)))))))

;; Función para conseguir la distancia Euclídea de un meteorito a un jugador
(define (get-distance player meteor)
  (sqrt (+ (expt (- (get-x meteor) (get-x player)) 2) (expt (- (get-y meteor) (get-y player)) 2))))

;; Función para detectar si algún meteorito de la lista ocupa la misma posición que un jugador (colisionan)
(define (check-collisions player1 player2 meteors)
  (cond
    ;; Si no queda ningún meteorito, devuelve 0
    ((null? meteors) 0)
    ;; Si el primer jugador ha colisionado con el meteorito actual, devuelve 1
    ((< (get-distance player1 (car meteors)) 15) 1)
    ;; Si el segundo jugador ha colisionado con el meteorito actual, devuelve 2
    ((< (get-distance player2 (car meteors)) 15) 2)
    ;; En otro caso, comprueba las colisiones del resto de meteoritos de forma recursiva
    (else (check-collisions player1 player2 (cdr meteors)))))
```

En cada actualización de fotograma, se comprueba si existe colisión entre algún meteorito y uno de los jugadores

Imagen

Una vez definidos los elementos básicos y sus interacciones, es el momento de concretar como serán **mostrados por pantalla**.

Para mostrar diferentes elementos por pantalla, la librería *2htdp* nos permite dibujar escenas de **forma apilada** (stack). Como la función *place-image* toma otra imagen como fondo de pantalla, se diseñó el **fondo de pantalla básico** del juego y a partir de ahí se fueron dibujando **elementos encima de este**.

De esta forma nos encontramos con que en cada fotograma primero se dibuja el fondo, encima de este los meteoritos, por encima los jugadores, y una vez que todo eso se ha dibujado se colocan los marcadores en pantalla.

A fin de permitir diferentes resoluciones, se hizo uso de una **escala** para ajustar el tamaño de los elementos al de la pantalla. La **resolución base** elegida es de 1600x900p.

juego.rkt

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Funciones de imagen

;; Fondo de pantalla y rectángulo central
(define BACKGROUND
  (place-image (scale SCALE (rectangle 10 400 "solid" "white"))
    (/ WIDTH 2)
    (- INITIAL-Y 100)
    (empty-scene WIDTH HEIGHT "black")))

;; Función para dibujar los meteoritos de una lista
(define (draw-meteors meteors background)
  (if (or (null? meteors) (null? (car meteors)))
      background
      (place-image (ellipse 10 10 "solid" "white")
        (get-x (car meteors))
        (get-y (car meteors))
        (draw-meteors (cdr meteors) background))))

;; Función para dibujar los jugadores
(define (draw-ships world background)
  (place-image (scale SCALE SHIP)
    (get-x (get-player-1 world))
    (get-y (get-player-1 world))
    (place-image (scale SCALE SHIP)
      (get-x (get-player-2 world))
      (get-y (get-player-2 world))
      background)))

;; Función para dibujar las puntuaciones
(define (draw-scores world background)
  (place-image (scale SCALE (text (number->string (get-score (get-player-1 world))) 100 "white"))
    (- PLAYER1-INITIAL-X 150)
    INITIAL-Y
    (place-image (scale SCALE (text (number->string (get-score (get-player-2 world))) 100 "white"))
      (+ PLAYER2-INITIAL-X 150)
      INITIAL-Y
      background)))

```

Se dibujan recursivamente
los meteoritos

Se dibujan recursivamente los meteoritos

2htdp

```
;;;;;;;;;;;;;;  
;;  
;; Funciones necesarias 2HTDP  
  
;; Función que renderiza las escenas  
(define (render-scene world)  
  (draw-scores world (draw-ships world (draw-meteors (get-meteors world) BACKGROUND)))))  
  
;; Manejador de tecla pulsada  
(define (key-pressed world key)  
  (cond  
    ((key=? key "w") (move-player1-up world))  
    ((key=? key "s") (move-player1-down world))  
    ((key=? key "up") (move-player2-up world))  
    ((key=? key "down") (move-player2-down world))  
    (else world)))  
  
;; Función para actualizar el mundo en cada ciclo de reloj  
(define (update-world world)  
  (let (  
    ;; Declaración de variables  
    (collision (check-collisions (get-player-1 world) (get-player-2 world) (get-meteors world)))) ;; Colisiones producidas al momento de llamar a la función  
    ;; Cuerpo del let  
    (cond  
      ;; Si ningún jugador ha colisionado, se mantienen sus posiciones  
      ((= collision 0) (create-world (get-player-1 world) (get-player-2 world) (move-meteors (get-meteors world))))  
      ;; Si el jugador 1 ha colisionado, se devuelve a su posición inicial  
      ((= collision 1) (create-world (create-player PLAYER1-INITIAL-X INITIAL-Y (get-score (get-player-1 world))) (get-player-2 world) (move-meteors (get-meteors world))))  
      ;; Si el jugador 2 ha colisionado, se devuelve a su posición inicial  
      (else (create-world (get-player-1 world) (create-player PLAYER2-INITIAL-X INITIAL-Y (get-score (get-player-2 world))) (move-meteors (get-meteors world)))))))  
  
;; Función que comprueba si uno de los jugadores ha alcanzado la puntuación máxima  
(define (end-game? world)  
  (or (= (get-score (get-player-1 world)) MAX-SCORE) (= (get-score (get-player-2 world)) MAX-SCORE)))  
  
;; Función principal del juego  
(define (game)  
  (big-bang (create-world (create-player PLAYER1-INITIAL-X INITIAL-Y 0) (create-player PLAYER2-INITIAL-X INITIAL-Y 0) (create-initial-meteors null))  
    (to-draw render-scene)  
    (on-key key-pressed)  
    (on-tick update-world)  
    (stop-when end-game?)  
    (name "Space Race")))
```

Menú del juego

Para mostrar el menú principal del juego se ha hecho uso exclusivamente de la librería *racket/base*, que nos permite crear con facilidad interfaces gráficas con apariencia nativa del sistema operativo.

El menú es completamente funcional y consta de:

- Título
- Controles de los jugadores
- Casillas para seleccionar dificultad
- Checkbox para seleccionar resolución
- Entrada de texto para introducir la puntuación máxima para ganar el juego
- Botones para iniciar la partida o salir del juego

main.rkt

Ventana principal

```
(load "juego.rkt")

;;;;;;;;;;;;;;
;;
;; VENTANA PRINCIPAL

;; Ventana del menú principal
(define main-menu-frame (new frame%
  [label "Space Race"]
  [width 500]
  [height 450]
  [x 800]
  [y 200]
  [style '(no-resize-border)]
  [alignment '(center top)]
  [stretchable-width #f]
  [stretchable-height #f]))

;; Panel principal
(define main-panel (new vertical-panel%
  [parent main-menu-frame]
  [alignment '(center top)]
  [min-width 500]
  [min-height 450]
  [stretchable-width #f]
  [stretchable-height #f]))
```

Space Race

SPACE RACE

CONTROLES JUGADOR 1	CONTROLES JUGADOR 2
Arriba: W	Arriba: UP-ARROW
Abajo: S	Abajo: DOWN-ARROW

Dificultad: ☐ Fácil ☒ Normal ☐ Díficil

Resolución: ☒ 800x600 ☐ 1280x720 ☐ 1600x900

Si la puntuación escogida es menor que 1 o no es un número, toma 10 como valor por defecto

Puntuación máxima:

INICIAR SALIR

main.rkt

Título

```
;;;;;;;;;;;;;
;;
;; TÍTULO

;; Panel contenedor
(define title-panel (new panel%
  [parent main-panel]
  [alignment '(center center)]
  [min-width 500]
  [min-height 100]
  [stretchable-width #f]
  [stretchable-height #f]))

;; Canvas del título
(define title-canvas (new canvas%
  [parent title-panel]
  [style '(transparent)]
  [min-width 500]
  [min-height 100]
  [stretchable-width #f]
  [stretchable-height #f]
  [paint-callback
   (lambda (canvas dc)
     (send dc set-scale 3 3)
     (send dc set-text-foreground "black")
     (send dc draw-text "SPACE RACE" 50 10))]))
```



Controles

```
;;;;;;;;;;;;;;  
;;  
;; CONTROLES DEL JUGADOR  
  
;; Panel contenedor  
(define controls-panel (new horizontal-panel%  
  [parent main-panel]  
  [alignment '(center center)]  
  [style '(border)]  
  [min-width 500]  
  [stretchable-width #f]  
  [stretchable-height #f]))  
  
;; Panel de los controles del jugador 1  
(define player1-controls-panel (new vertical-panel%  
  [parent controls-panel]  
  [style '(border)]  
  [min-width 250]  
  [alignment '(center center)]  
  [stretchable-width #f]  
  [stretchable-height #f]))  
  
;; Texto de los controles del jugador 1  
(define player1-controls-message (new message%  
  [parent player1-controls-panel]  
  [label "CONTROLES JUGADOR 1\n\nArriba: W\nAbajo: S"]))  
  
;; Panel de los controles del jugador 2  
(define player2-controls-panel (new vertical-panel%  
  [parent controls-panel]  
  [style '(border)]  
  [min-width 250]  
  [alignment '(center center)]  
  [stretchable-width #f]  
  [stretchable-height #f]))  
  
;; Texto de los controles del jugador 2  
(define player2-controls-message (new message%  
  [parent player2-controls-panel]  
  [label "CONTROLES JUGADOR 2\n\nArriba: UP-ARROW\nAbajo: DOWN-ARROW"])))
```

Space Race

SPACE RACE

CONTROLES JUGADOR 1	CONTROLES JUGADOR 2
Arriba: W	Arriba: UP-ARROW
Abajo: S	Abajo: DOWN-ARROW

Dificultad: ☐ Fácil ☒ Normal ☐ Difícil

Resolución: ☒ 800x600 ☐ 1280x720 ☐ 1600x900

Si la puntuación escogida es menor que 1 o no es un número, toma 10 como valor por defecto

Puntuación máxima:

Dificultad

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;
;; DIFICULTAD

;; Panel contenedor
(define difficulty-checkbox-panel (new horizontal-panel%
  [parent main-panel]
  [alignment '(center center)]
  [min-width 500]
  [min-height 50]
  [stretchable-width #f]
  [stretchable-height #f]))

;; Texto de Dificultad:
(define difficulty-message (new message%
  [parent difficulty-checkbox-panel]
  [label "Dificultad: "]))

;; Checkbox para el modo Fácil
(define easy-checkbox (new check-box%
  [parent difficulty-checkbox-panel]
  [label "Fácil"]
  [callback (lambda (checkbox event)
    (if (send easy-checkbox get-value)
      (begin (send medium-checkbox set-value #f)
              (send hard-checkbox set-value #f)
              (set! MAX-METEORS 20)
              (set! MIN-METEOR-SPEED 1))
      (send easy-checkbox set-value #t)))]))

;; Checkbox para el modo Normal
(define medium-checkbox (new check-box%
  [parent difficulty-checkbox-panel]
  [label "Normal"]
  [value #t]
  [callback (lambda (checkbox event)
    (if (send medium-checkbox get-value)
      (begin (send easy-checkbox set-value #f)
              (send hard-checkbox set-value #f)
              (set! MAX-METEORS 50)
              (set! MIN-METEOR-SPEED 2))
      (send medium-checkbox set-value #t)))]))

;; Checkbox para el modo Difícil
(define hard-checkbox (new check-box%
  [parent difficulty-checkbox-panel]
  [label "Difícil"]
  [callback (lambda (checkbox event)
    (if (send hard-checkbox get-value)
      (begin (send easy-checkbox set-value #f)
              (send medium-checkbox set-value #f)
              (set! MAX-METEORS 80)
              (set! MIN-METEOR-SPEED 3))
      (send hard-checkbox set-value #t)))]))

```

Al cambiar la dificultad, se cambia el número de meteoritos y su velocidad mínima

Resolución

```
;; RESOLUCIÓN
;; Función para actualizar los valores relacionados con la resolución de la pantalla
(define (update-resolution width height)
  (set! WIDTH width)
  (set! HEIGHT height)
  (set! SCALE (/ HEIGHT 900))
  (set! INITIAL-Y (- HEIGHT (/ HEIGHT 6)))
  (set! PLAYER1-INITIAL-X (+ 100 (/ WIDTH 4)))
  (set! PLAYER2-INITIAL-X (- WIDTH PLAYER1-INITIAL-X))
  (set! BACKGROUND (place-image (scale SCALE (rectangle 10 400 "solid" "white"))
    (/ WIDTH 2)
    (- INITIAL-Y 100)
    (empty-scene WIDTH HEIGHT "black"))))

;; Panel contenedor
(define resolution-checkbox-panel (new horizontal-panel%
  [parent main-panel]
  [alignment '(center center)]
  [min-width 500]
  [min-height 50]
  [stretchable-width #f]
  [stretchable-height #f]))

;; Texto de Resolución:
(define resolution-message (new message%
  [parent resolution-checkbox-panel]
  [label "Resolución: "]))

;; Checkbox para 800x600
(define res1-checkbox (new check-box%
  [parent resolution-checkbox-panel]
  [label "800x600"]
  [value #t]
  [callback (lambda (checkbox event)
    (if (send res1-checkbox get-value)
      (begin (send res2-checkbox set-value #f)
              (send res3-checkbox set-value #f)
              (update-resolution 800 600))
      (send res1-checkbox set-value #t)))]))

;; Checkbox para 1280x720
(define res2-checkbox (new check-box%
  [parent resolution-checkbox-panel]
  [label "1280x720"]
  [callback (lambda (checkbox event)
    (if (send res2-checkbox get-value)
      (begin (send res1-checkbox set-value #f)
              (send res3-checkbox set-value #f)
              (update-resolution 1280 720))
      (send res2-checkbox set-value #t)))]))

;; Checkbox para 1600x900
(define res3-checkbox (new check-box%
  [parent resolution-checkbox-panel]
  [label "1600x900"]
  [callback (lambda (checkbox event)
    (if (send res3-checkbox get-value)
      (begin (send res1-checkbox set-value #f)
              (send res2-checkbox set-value #f)
              (update-resolution 1600 900))
      (send res3-checkbox set-value #t)))]))
```

Space Race

SPACE RACE

CONTROLES JUGADOR 1	CONTROLES JUGADOR 2
Arriba: W	Arriba: UP-ARROW
Abajo: S	Abajo: DOWN-ARROW

Dificultad:
☐ Fácil
☒ Normal
☐ Difícil

Resolución:
☒ 800x600
☐ 1280x720
☐ 1600x900

Si la puntuación escogida es menor que 1 o no es un número, toma 10 como valor por defecto

Puntuación máxima:

INICIAR

SALIR

Score

```
;;;;;;;;;;;;;;  
;;  
;; SCORE  
  
;; Texto de advertencia  
(define score-message (new message%  
  [parent main-panel]  
  [font small-control-font]  
  [label "Si la puntuación escogida es menor que 1 o no es un número, toma 10 como valor por defecto"]))  
  
;; Panel contenedor  
(define score-text-panel (new horizontal-panel%  
  [parent main-panel]  
  [alignment '(center center)]  
  [min-width 170]  
  [min-height 50]  
  [stretchable-width #f]  
  [stretchable-height #f]))  
  
;; Campo de texto  
(define score-text-field (new text-field%  
  [parent score-text-panel]  
  [label "Puntuación máxima:"]  
  [init-value (number->string 10)]  
  [callback (lambda (text-field event)  
    (if (or (not (string->number (send text-field get-value))) (< (string->number (send text-field get-value)) 1))  
        (set! MAX-SCORE 10)  
        (set! MAX-SCORE (string->number (send text-field get-value))))))])
```

Space Race

SPACE RACE

CONTROLES JUGADOR 1	CONTROLES JUGADOR 2
Arriba: W	Arriba: UP-ARROW
Abajo: S	Abajo: DOWN-ARROW

Dificultad: ☐ Fácil ☒ Normal ☐ Difícil

Resolución: ☒ 800x600 ☐ 1280x720 ☐ 1600x900

Si la puntuación escogida es menor que 1 o no es un número, toma 10 como valor por defecto

Puntuación máxima:

INICIAR SALIR

Botones principales

```
;;;;;;;;;;;;;;  
;;  
;; BOTONES PRINCIPALES  
  
;; Panel contenedor  
(define button-panel (new horizontal-panel%  
  [parent main-panel]  
  [alignment '(center center)]  
  [min-width 500]  
  [min-height 50]  
  [stretchable-width #f]  
  [stretchable-height #f]))  
  
;; Botón de Inicio  
(define start-button (new button%  
  [label "INICIAR"]  
  [parent button-panel]  
  [callback (lambda (button event) (send main-menu-frame show #f) (game) (send main-menu-frame show #t))]))  
  
;; Botón de Salir  
(define exit-button (new button%  
  [label "SALIR"]  
  [parent button-panel]  
  [callback (lambda (button event) (exit))]))  
  
;; Se hace visible la ventana principal  
(send main-menu-frame show #t)
```

Space Race

SPACE RACE

CONTROLES JUGADOR 1	CONTROLES JUGADOR 2
Arriba: W	Arriba: UP-ARROW
Abajo: S	Abajo: DOWN-ARROW

Dificultad: ☐ Fácil ☒ Normal ☐ Difícil

Resolución: ☒ 800x600 ☐ 1280x720 ☐ 1600x900

Si la puntuación escogida es menor que 1 o no es un número, toma 10 como valor por defecto

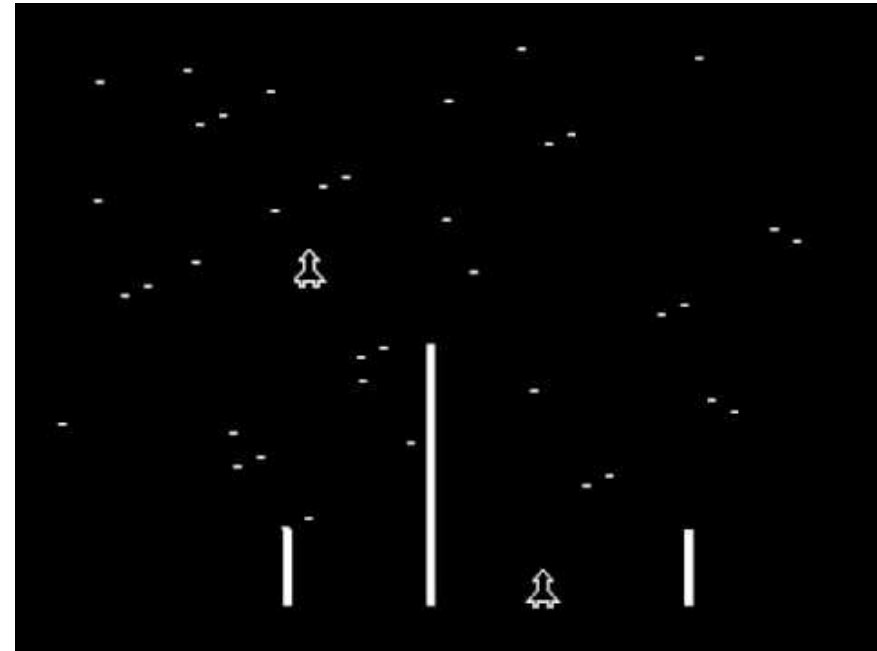
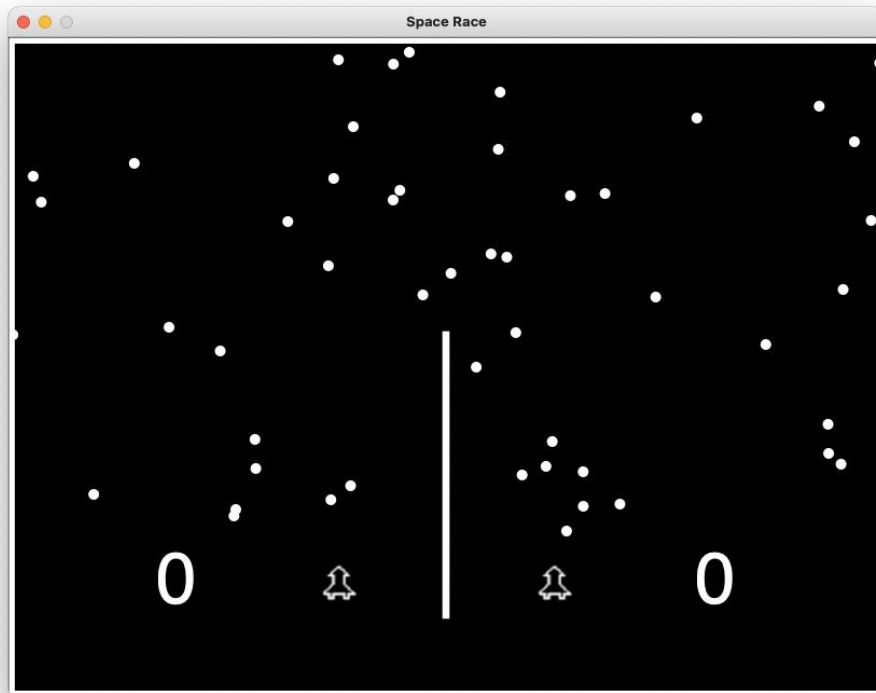
Puntuación máxima:

INICIAR SALIR

Resultados

El resultado de este trabajo es un juego completamente funcional que cuenta además con un menú principal con varias opciones a elegir por el jugador.

Aquí podemos ver una comparación entre el juego original y el desarrollado en Scheme:



Conclusiones

Como se ha podido ver en este trabajo, es posible crear aplicaciones gráficas con cierto grado de complejidad en Scheme y que estas sean completamente funcionales.

A pesar de ser un lenguaje interpretado, con la penalización al rendimiento que eso conlleva, se puede ofrecer una buena experiencia al usuario en aplicaciones 2D con un gran número de elementos en pantalla.

Dicho todo esto, se puede concluir que la Programación Declarativa (Funcional) es una alternativa real y potente que no tiene nada que envidiar a los paradigmas clásicos de programación (Imperativa/OOP), tanto en aplicaciones de línea de comandos/intérprete como para aplicaciones gráficas.

Bibliografía

[https://en.wikipedia.org/wiki/Space_Race_\(video_game\)](https://en.wikipedia.org/wiki/Space_Race_(video_game))

<https://docs.racket-lang.org/style/index.html>

<https://docs.racket-lang.org/gui/index.html>

<https://docs.racket-lang.org/teachpack/2htdp2htdp.html>