Proyecto 2025-26:

Recreativa 1978

Space Invaders, Taito Corp.

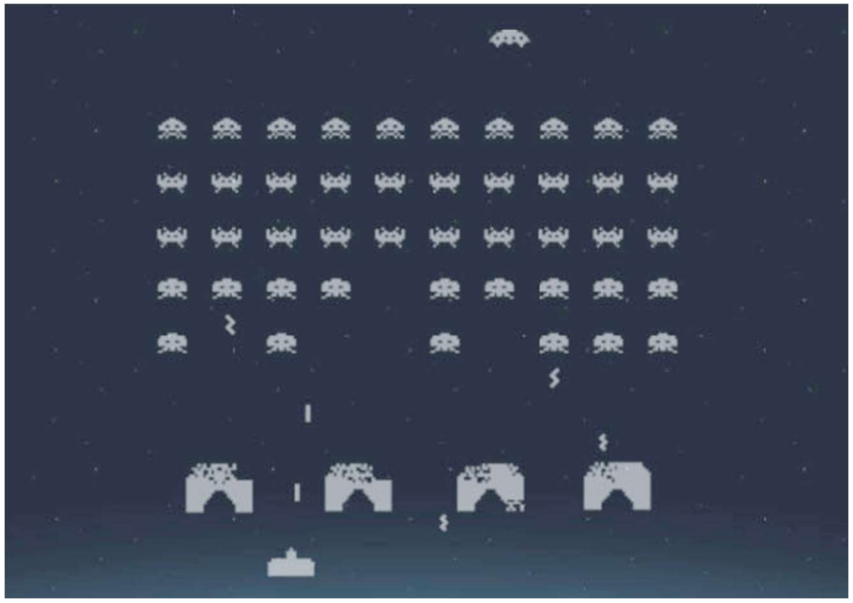
Programación de Sistemas y Dispositivos

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Proyecto

curso 2025-26

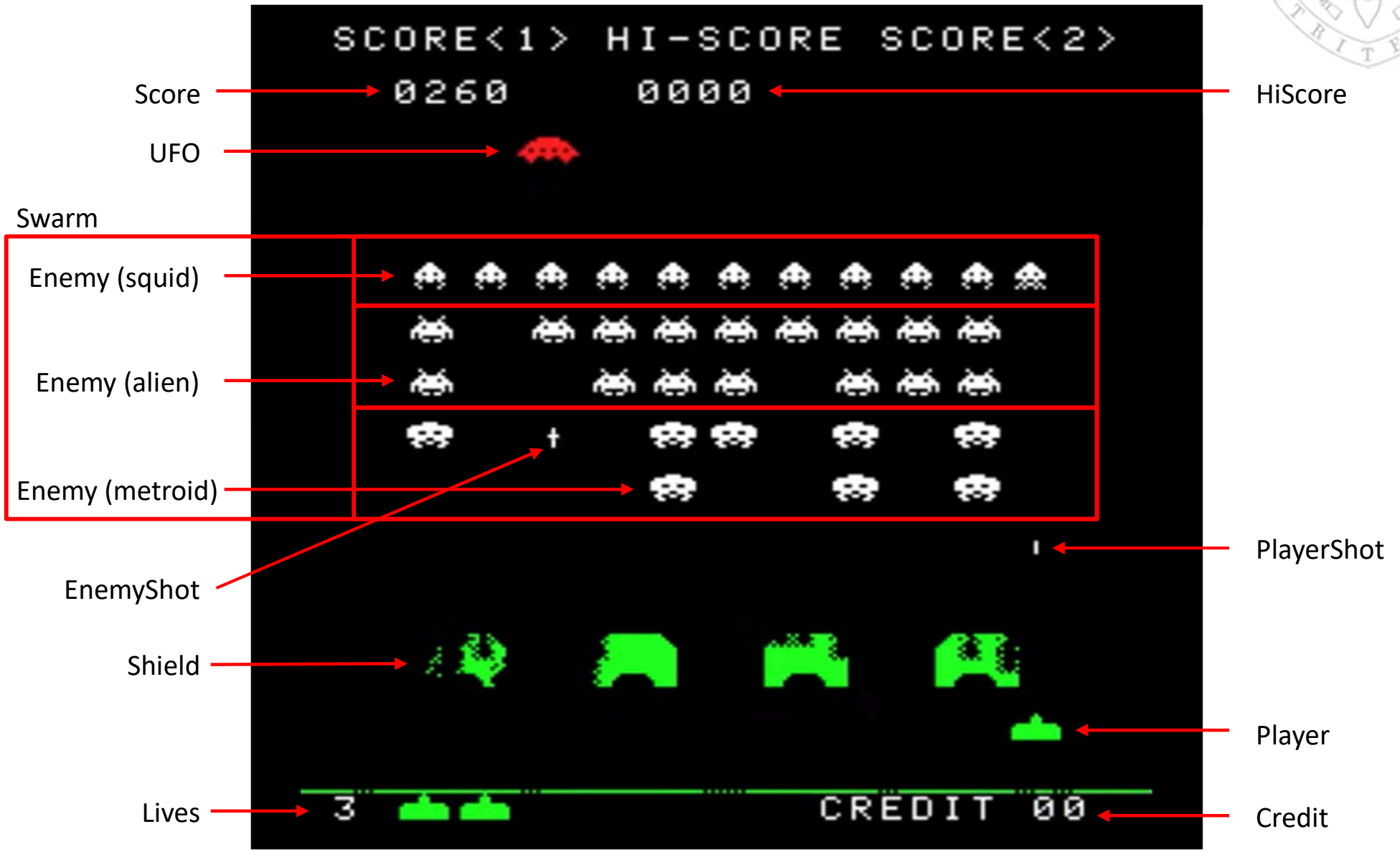
- Recreativa 1978 (Space Invaders, Taito Corp.)
  - Aplicación multihebra baremetal.
  - Visualizar sprites en un LCD.
  - Interactuar a través de keypad.
  - Reproducir sonidos a través de un Audio Codec.
  - Tener un mínimo footprint (unos 100 KB).



https://www.youtube.com/watch?v=MU4psw3ccUI

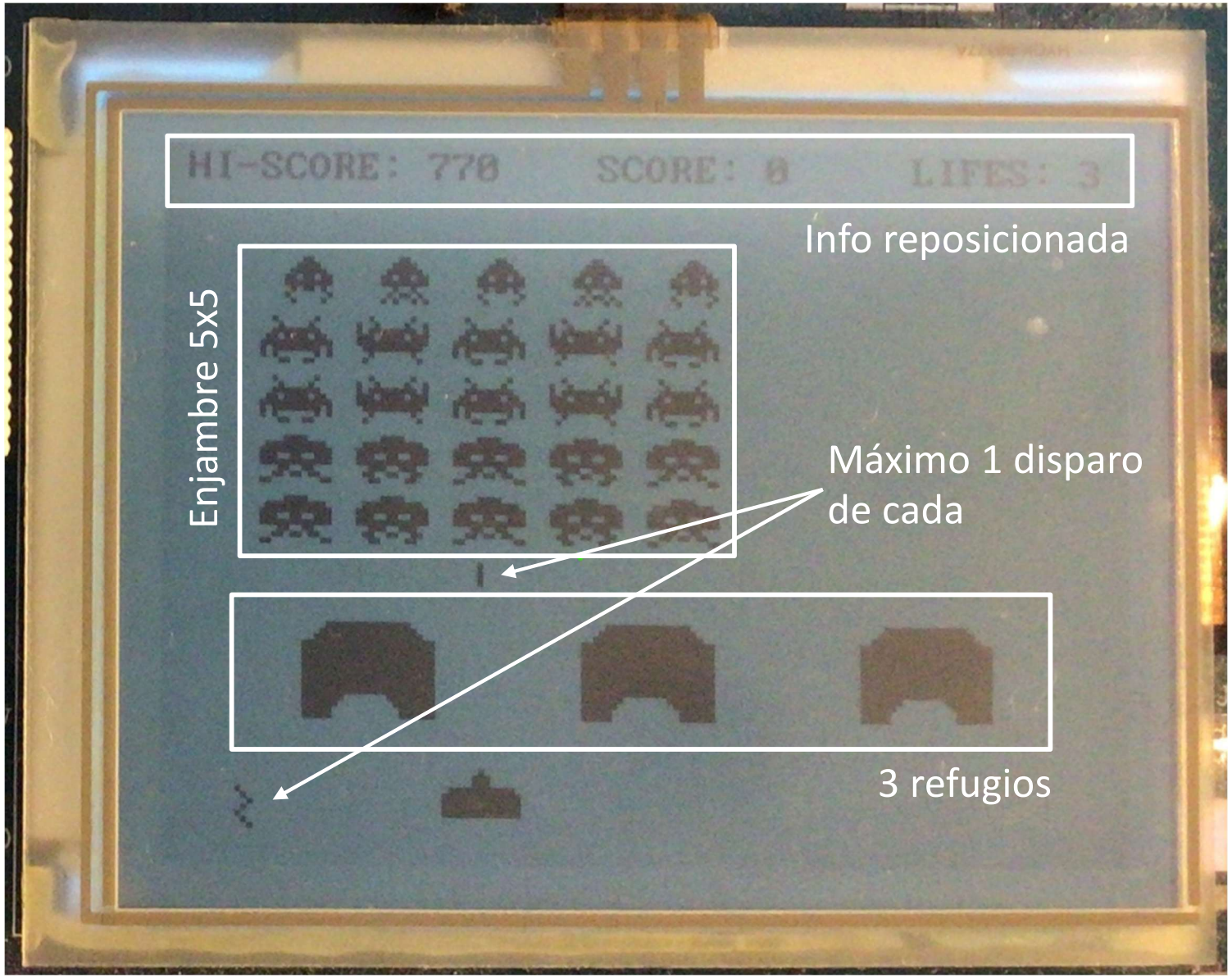
J.M. Mendías
2025

Space Invaders



J.M. Mendías
2025

Adaptación



J.M. Mendías 2025

Sprites

- Un *sprite* es la representación gráfica de una entidad, tiene:
  - Anchura, altura y *pixmap* (array monodimensional de pixeles organizados por filas).

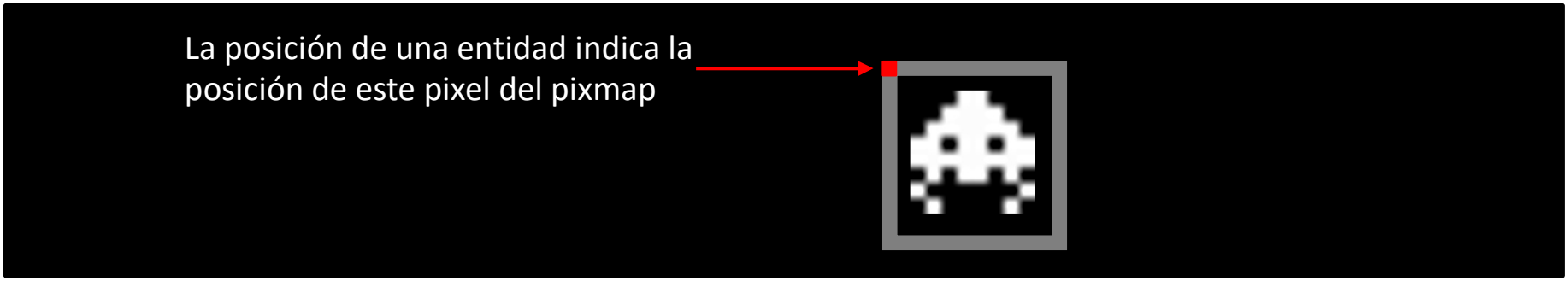
```
#define ENEMY_WIDTH  (16)
#define ENEMY_HEIGHT (10)

uint8 squidPixMap_1[] =
{
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
```

Pixeles visibles

J.M. Mendías
2025

Sprites

- Dado que el *sprite* ocupa varios píxeles, es necesario escoger uno para representar la posición de la entidad que representa.
  - Elegiremos el extremo superior izquierdo para facilitar su visualización.



- Los enemigos y la explosión de la nave del jugador tienen animaciones:
  - Estos objetos tienen 2 *sprites* cuya visualización se va alternando.
  - Los enemigos alternan *sprite* a cada movimiento, la explosión cada vez que transcurren cierto número de refrescos.

Sprites

- En el archivo pixmaps.c están definidos los pixmaps de todas las entidades, en el archivo pixmaps.h sus dimensiones.

`playerPixMap`	nave del jugador
`ufoPixMap`	UFO
`alienPixMap_0` `alienPixMap_1`	visualizar alternadamente a cada movimiento del *alien*
`metroidPixMap_0` `metroidPixMap_1`	visualizar alternadamente a cada movimiento del *metroid*
`squidPixMap_0` `squidPixMap_1`	visualizar alternadamente a cada movimiento del *squid*
`shieldPixMap`	refugio
`playerShotPixMap`	disparo del jugador
`enemyShotPixMap`	disparo del enemigo

Sprites

- En el archivo pixmaps.c están definidos los pixmaps de todas las entidades, en el archivo pixmaps.h sus dimensiones.

`playerShotExplosionPixMap`	explosión del disparo del jugador tras impactar contra el techo o contra un refugio
`enemyShotExplosionPixMap`	explosión del disparo enemigo cuando es alcanzado o tras impactar contra el suelo o contra un refugio.
`playerExplosionPixMap_0` `playerExplosionPixMap_1`	visualizar alternadamente mientras la nave explota
`ufoExplosionPixMap`	explosión del UFO
`enemyExplosionPixMap`	explosión de cualquier enemigo

J.M. Mendías
2025

Sonidos

- Un sonido es un fichero WAV que deberemos cargar en memoria desde consola del depurador.

- En el archivo wavs.h están definidos sus posiciones de carga.

`INTRO_MUSIC`	en bucle hasta que la partida comience

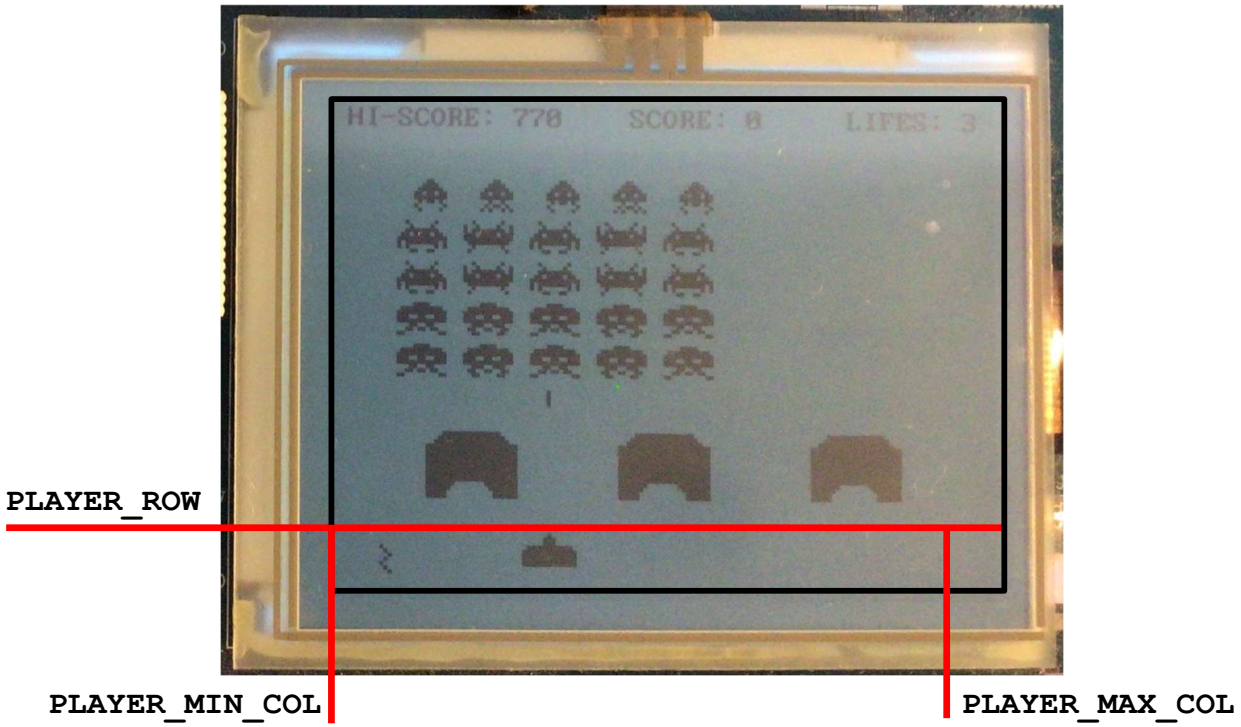
`SWARM_MOVE1`
`SWARM_MOVE2`
`SWARM_MOVE3`
`SWARM_MOVE4`
	en secuencia cada vez que se mueve un enemigo del enjambre

`UFO_LAUNCH`	cuando aparece un UFO
`PLAYERSHOOT_LAUNCH`	cuando el jugador dispara
`PLAYER_EXPLOSION`	cuando la nave del jugador es alcanzada
`ENEMY_EXPLOSION`	cuando un enemigo o un UFO es alcanzado

Configuración

- En el archivo config.h están definidos múltiples parámetros del juego:
  - Posiciones máximas y mínimas de cada objeto, tiempos de actualización, duración de efectos, puntuaciones, etc.

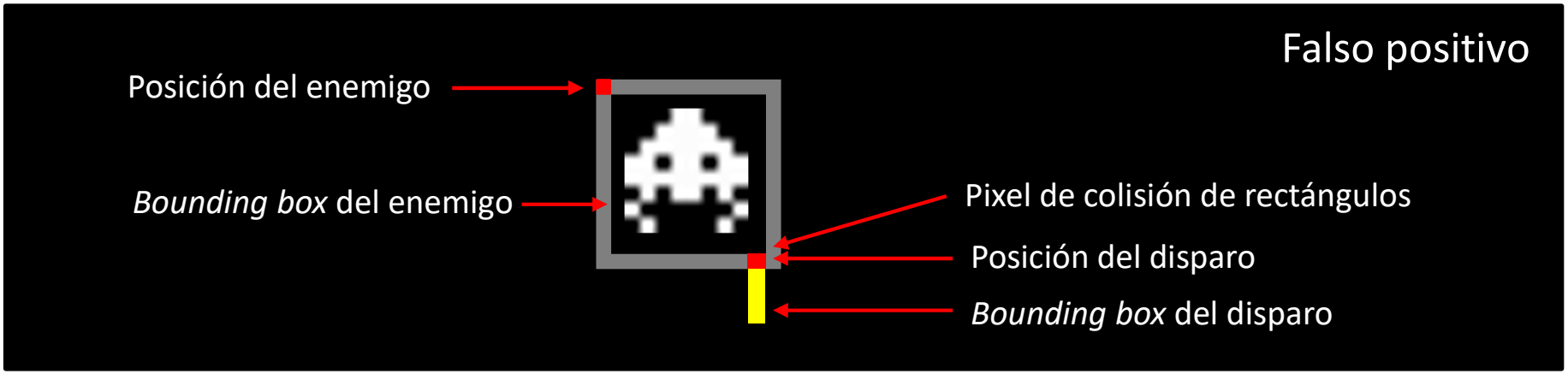
```
#define PLAYER_ROW     (GAME_HEIGHT - PLAYER_HEIGHT - ENEMYSHOT_EXPLOSION_HEIGHT)
#define PLAYER_MIN_COL (0)
#define PLAYER_MAX_COL (GAME_WIDTH - PLAYER_WIDTH)
```



Detección de colisiones

- *Bounding boxes* es un algoritmo elemental de detección de colisiones:
  - Cada entidad se considera como un rectángulo definido por su posición y por la anchura y altura de su *sprite*.
  - Dos entidades colisionan si sus rectángulos se superponen.
  - Muy eficiente (solo comparaciones de coordenadas).

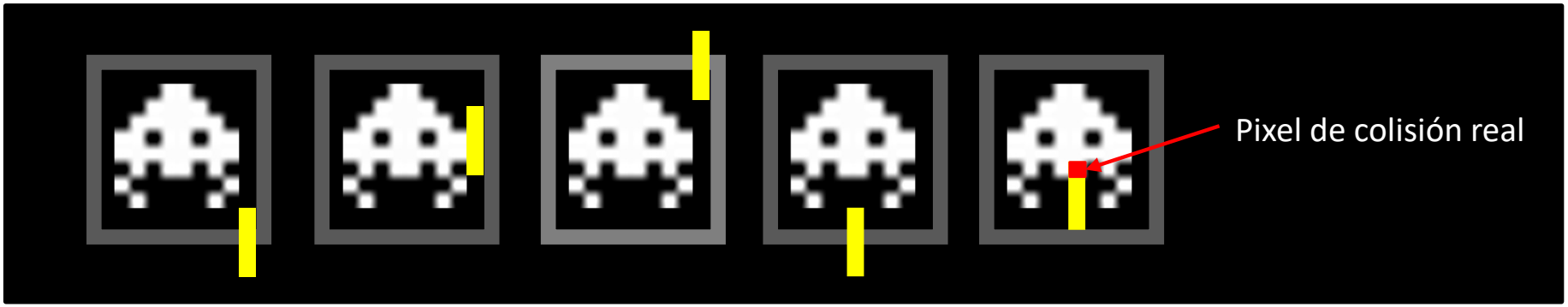
- Genera falsos positivos porque los *sprites* tienen formas irregulares:
  - Los rectángulos pueden colisionar, aunque no lo hagan sus píxeles visibles.



J.M. Mendías 2025

Detección de colisiones

- Para colisiones usaremos el algoritmo de sobreescritura de píxeles:
  - Al pintar cada pixel visible del *sprite* se chequea si ya existe en dicha posición un pixel dibujado. Si lo hay, es una colisión real.
  - Aprovecha que el *buffer* de vídeo refleja los objetos vivos que existen.
  - Muy eficiente (una sola pasada por los píxeles del *sprite*)



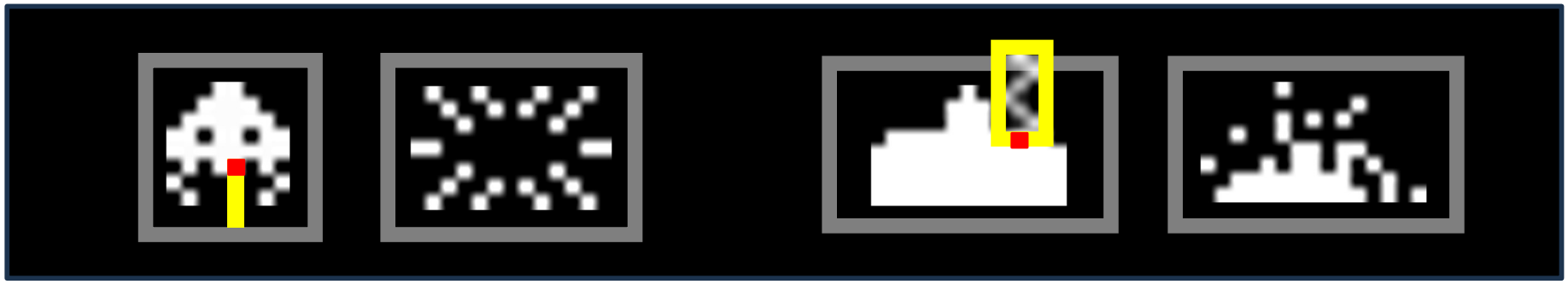
- Aunque detecta colisiones reales, no determina con quién.
  - Para identificar la entidad colisionada, usaremos bounding boxes.

```
#define OVERLAP( a, b, b_len ) ( a >= b && a < b+b_len )
```

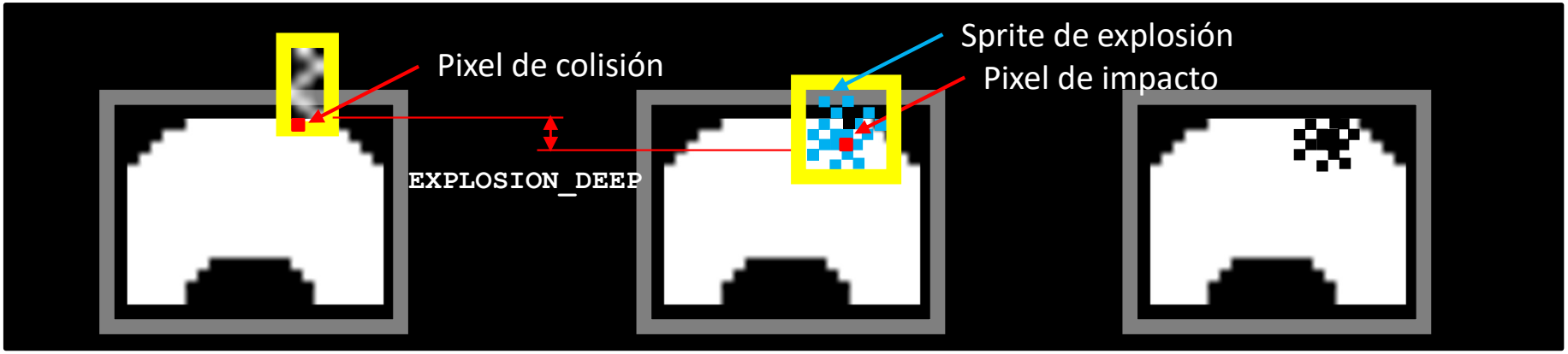
J.M. Mendías 2025

Erosión del refugio

- Cuando un disparo colisiona con una nave, esta explota:



- Pero cuando un disparo colisiona con un refugio, este se erosiona:
  - Nuevamente aprovecharemos que el *buffer* de vídeo refleja los objetos que existen.
  - Al colisionar, movemos ligeramente el punto de impacto y pintamos la explosión.
  - Al borrar la explosión, parte del refugio de la pantalla (no del sprite) desaparece.



Elección de actores

- Cuando colisionan 2 entidades, debemos elegir a una de ellas que:
  - Detecte la colisión.
  - Visualice la explosión.
  - Genere el sonido cuando corresponda.

- Los disparos detectan la colisión con otras entidades:
  - El disparo del jugador colisiona con: refugio, enemigo, ufo, disparo enemigo, techo.
  - El disparo del enemigo colisiona con: refugio, nave del jugador, suelo.
- La explosión es visualizada por la entidad alcanzada:
  - Visualizan su explosión: nave del jugador, enemigo, ufo y disparo del enemigo.
  - En el caso del techo, suelo y refugio, la visualiza el correspondiente disparo.
- El sonido de la explosión es generado por la entidad alcanzada:
  - Generan sonido al explotar: nave del jugador, enemigo y ufo.
  - El resto de las explosiones son silenciosas.

J.M. Mendías 2025

Movimiento de enjambre

- Uno de los elementos más característicos del juego es el movimiento del enjambre.
- Los enemigos en formación se mueven repetitivamente:
  - De izquierda a derecha hasta que todos los enemigos del extremo derecho alcanzan el borde derecho, entonces bajan.
  - De derecha a izquierda hasta que todos los enemigos del extremo izquierdo alcanzan el borde izquierdo, entonces, vuelven a bajar.
- Sin embargo, el movimiento no es en bloque sino enemigo a enemigo.
  - En cada momento, por turno cíclico, se mueve un único enemigo.
- Conforme los enemigos van muriendo:
  - Los extremos del enjambre cambian.
  - La velocidad del enjambre aumenta al reducirse el número de enemigos a desplazar.
- Para realizar este movimiento, será necesario tener en cuenta:
  - El último enemigo que se movió.
  - Un enemigo que determine cada uno de los extremos del enjambre.
  - Si uno de ellos muere, debe ser reemplazado por otro vivo.

J.M. Mendías 2025

C orientado a objetos

- La programación orientada a objetos permite organizar el código de una manera más clara y fácil de mantener.

- Aunque C no tiene soporte nativo para clases y objetos, si es posible organizar el código con cierta orientación a objetos.
  - Modularidad: Cada clase se organiza en módulos independientes (archivos .c y .h).
  - Encapsulación: Los atributos de la clase se agrupan en *struct* y todas las funciones operan sobre punteros a esa estructura (*self*).
  - Métodos simulados: Las funciones que reciben el puntero a la estructura actúan como métodos de un objeto.
  - Estado interno: Cada objeto (instancia del *struct*) mantiene su propio estado.

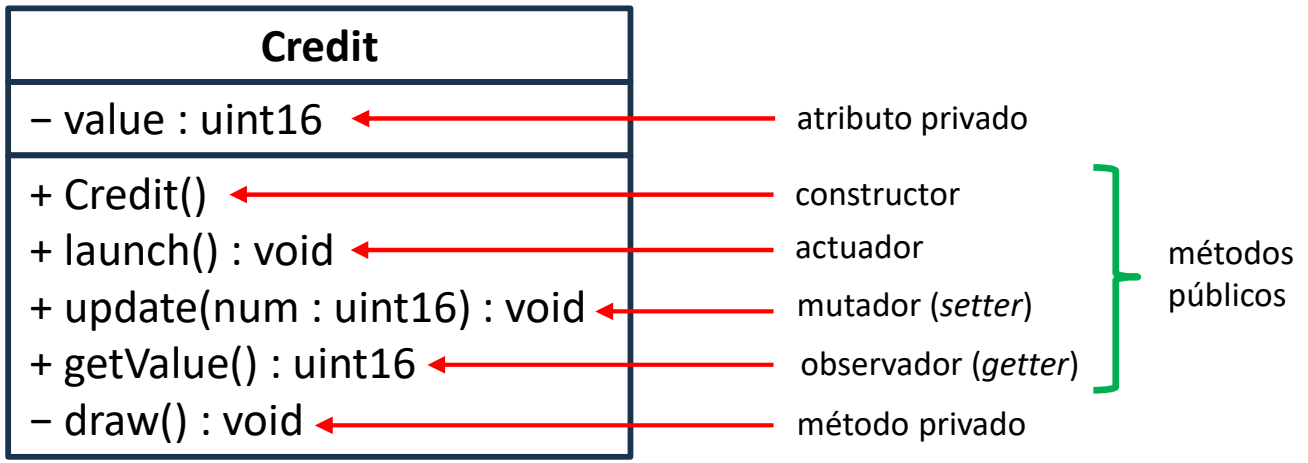
- El uso de OO-like C una práctica común en bibliotecas de firmware y en software empotrado moderno.
  - Es una disciplina de programador que se puede seguir con distintos grados de rigor.

C orientado a objetos

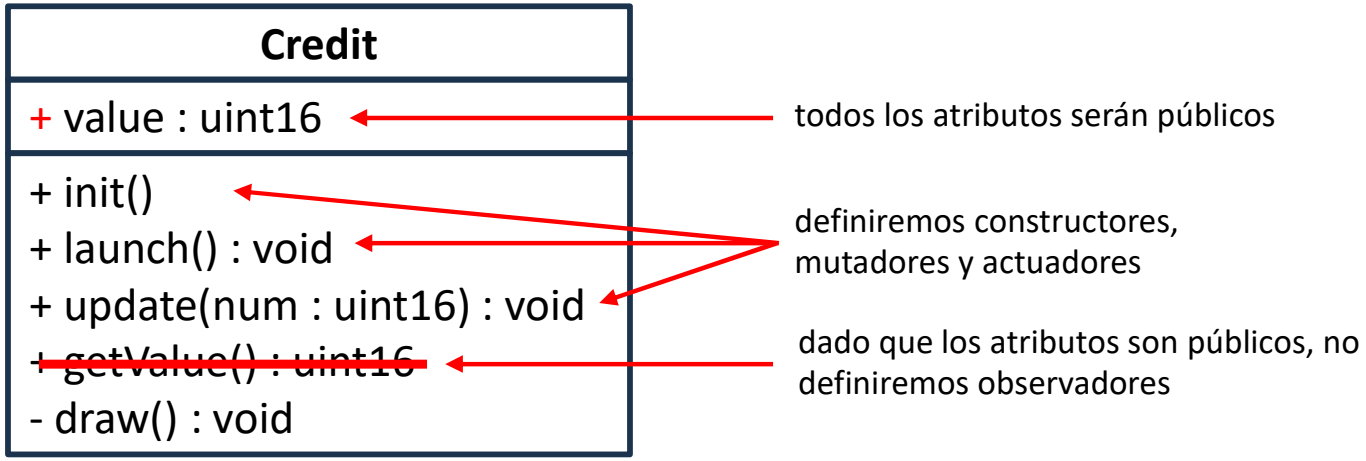
clase *Credit*

- Gestiona el crédito (número de partidas posible) del juego.

Clase original



Clase adaptada a OO-like C elemental



C orientado a objetos

clase *Credit*: credit.h

```
#ifndef CREDIT_H
#define CREDIT_H

#include <common_types.h>

typedef struct {
  uint16 value;
} Credit;

void credit_init( Credit *self );
void credit_launch( Credit *self );
void credit_update( Credit *self, uint16 num );

#endif
```

todos los métodos reciben como primer argumento un puntero ***self*** a la estructura que representa al objeto, equivale al ***this*** C++/java

atributo del objeto

estructura que representa a un objeto de la clase *Credit*

métodos públicos del objeto

Fichero cabecera

```
...
Credit credit;
...
credit_init( &credit );
if ( credit.value > 0 )
  ...
```

Fichero principal

J.M. Mendías
2025

C orientado a objetos

clase *Credit*: credit.c

```
#include <segs.h>
#include "credit.h"

static void credit_draw( Credit *self );

void credit_draw( Credit *self ) {
  segs_putchar( self->value );
}

void credit_init( Credit *self ) {
  self->value = 0;
}

void credit_launch( Credit *self ) {
  credit_draw( self );
}

void credit_update( Credit *self, uint16 num ) {
  self->value += num;
  credit_draw( self );
}
```

método privado del objeto

J.M. Mendías 2025

C orientado a objetos

- Cada entidad será un objeto perteneciente a cierta clase.
  - Que se modelará usando un autómata de estados finitos.

- Atributos de clase principales:
  - state: estado en el que se encuentra el objeto.
  - col, row: fila y columna en la que se encuentra el objeto.
  - sprite: *sprite* asociado al objeto (pueden ser varios).
  - sound: sonido asociado al objeto (pueden ser varios).
- Métodos de clase principales:
  - init: inicializa los atributos del objeto.
  - launch: visualiza por primera vez el objeto.
  - update: actualiza la posición del objeto y visualiza en consecuencia.
  - hit: señaliza al objeto que ha sido alcanzado para que cambie su estado.
  - left/rigth/down/stop: señaliza al objeto su sentido de movimiento.
  - on*Object*: si la colisión es con *Object*, actúa en consecuencia.
  - draw: visualiza el objeto a la vez que detecta si hay colisión con otro.
  - clear: borra el objeto.

C orientado a objetos

clase *Ufo*

Ufo	
+ state : { noUfo | movingLeft | movingRight | exploding | scoring } + countDown : uint8 + score : uint8 + col : int16 + row : int16 + sprite : Sprite + explosionSprite : Sprite + lauchSound : Sound + explosionSound : Sound	} composición de clases
+ init() + launch() : void + update() : void	+ hit() : void − draw() : void − clear() : void

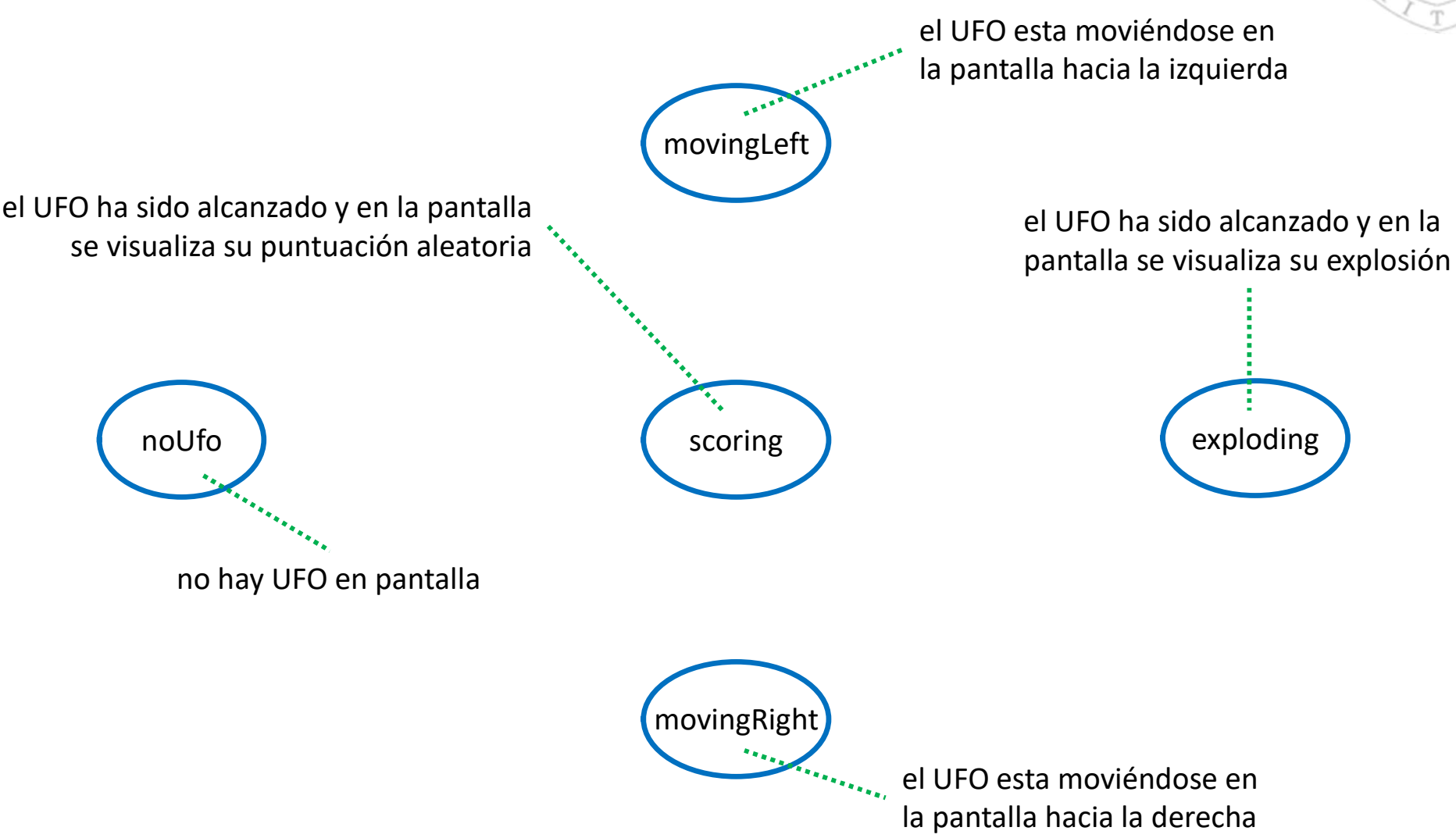
C orientado a objetos

clase *Ufo*: ufo.h

```
#ifndef UFO_H
#define UFO_H

#include <common_types.h>
#include "sprite.h"
#include "sound.h"

typedef enum {noUfo,ufoMovingLeft,ufoMovingRight,ufoExploding,ufoScoring} ufo_state_t;

typedef struct {
    ufo_state_t state;
    uint8  countDown;
    uint8  score;
    int16  col;
    int16  row;
    Sprite sprite;
    Sprite explosionSprite;
    Sound  launchSound;
    Sound  explosionSound;
} Ufo;

void ufo_init( Ufo *self );
void ufo_launch( Ufo *self );
void ufo_update( Ufo *self );
void ufo_hit( Ufo *self );

#endif
```

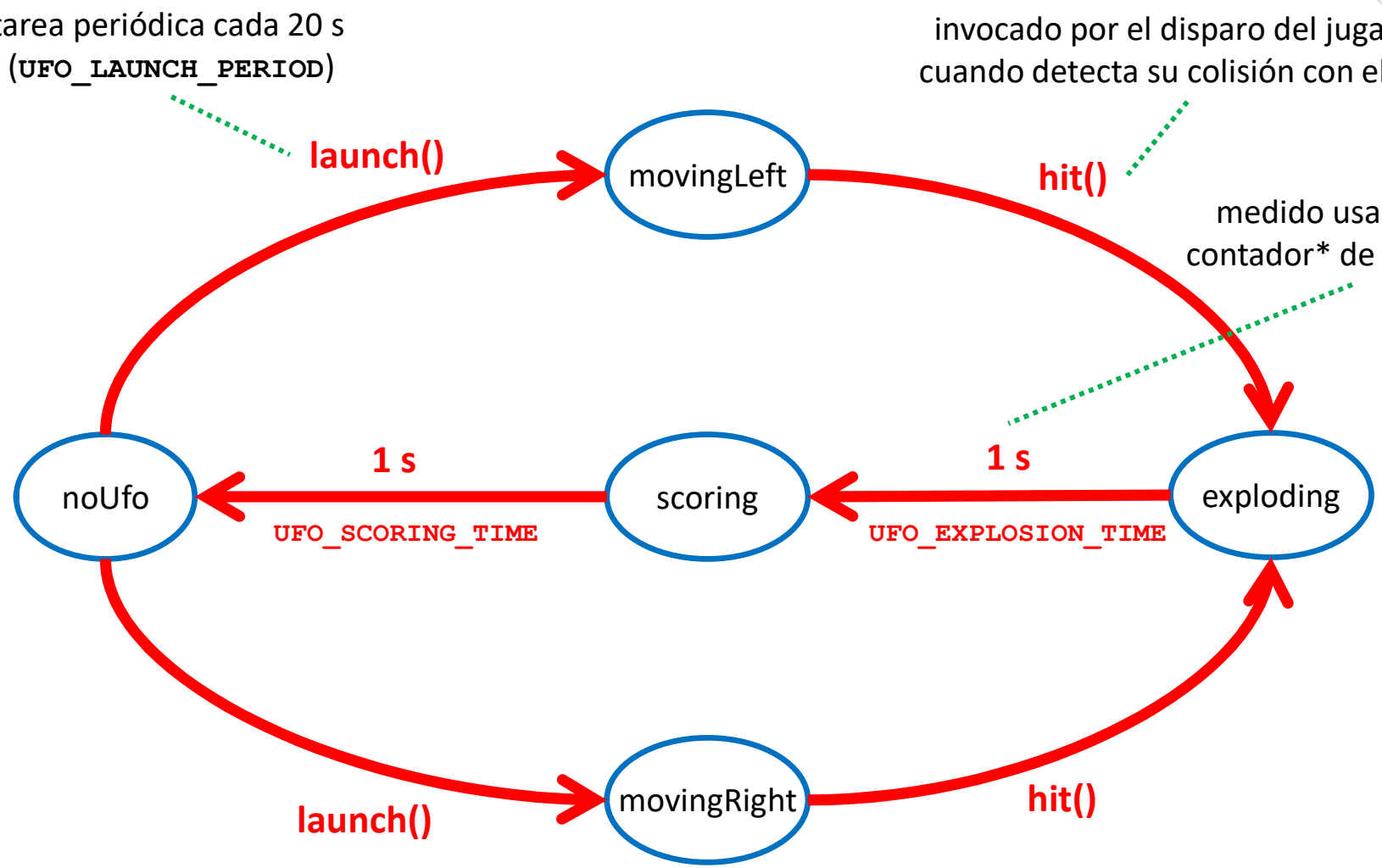
C orientado a objetos

clase *Ufo:* autómata



C orientado a objetos

clase *Ufo:* autómata

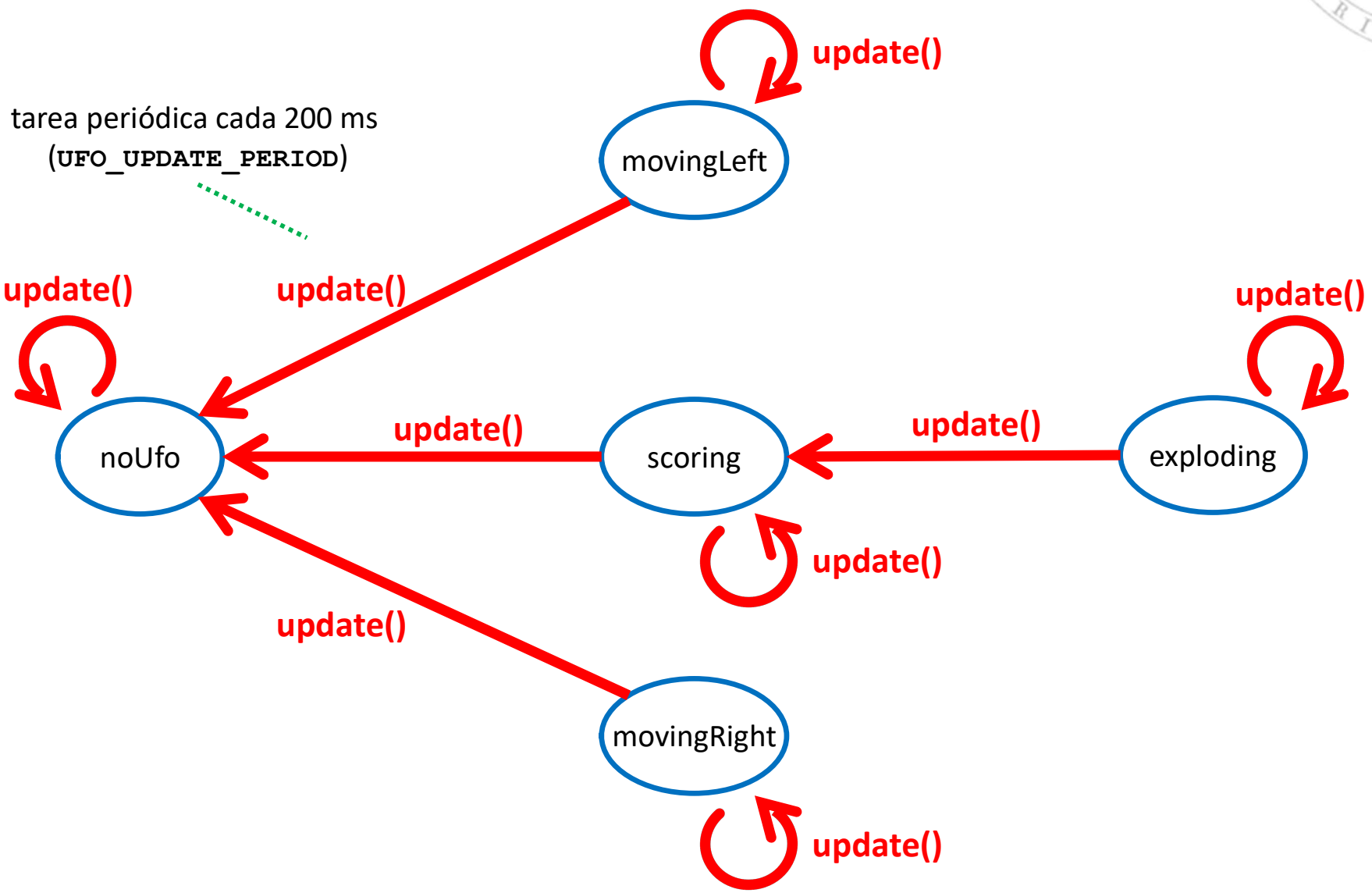


(*) `UFO_EXPLODING_TIME/UFO_UPDATE_PERIOD` = 1s / 200 ms = 5

J.M. Mendías 2025

C orientado a objetos

clase *Ufo:* autómata



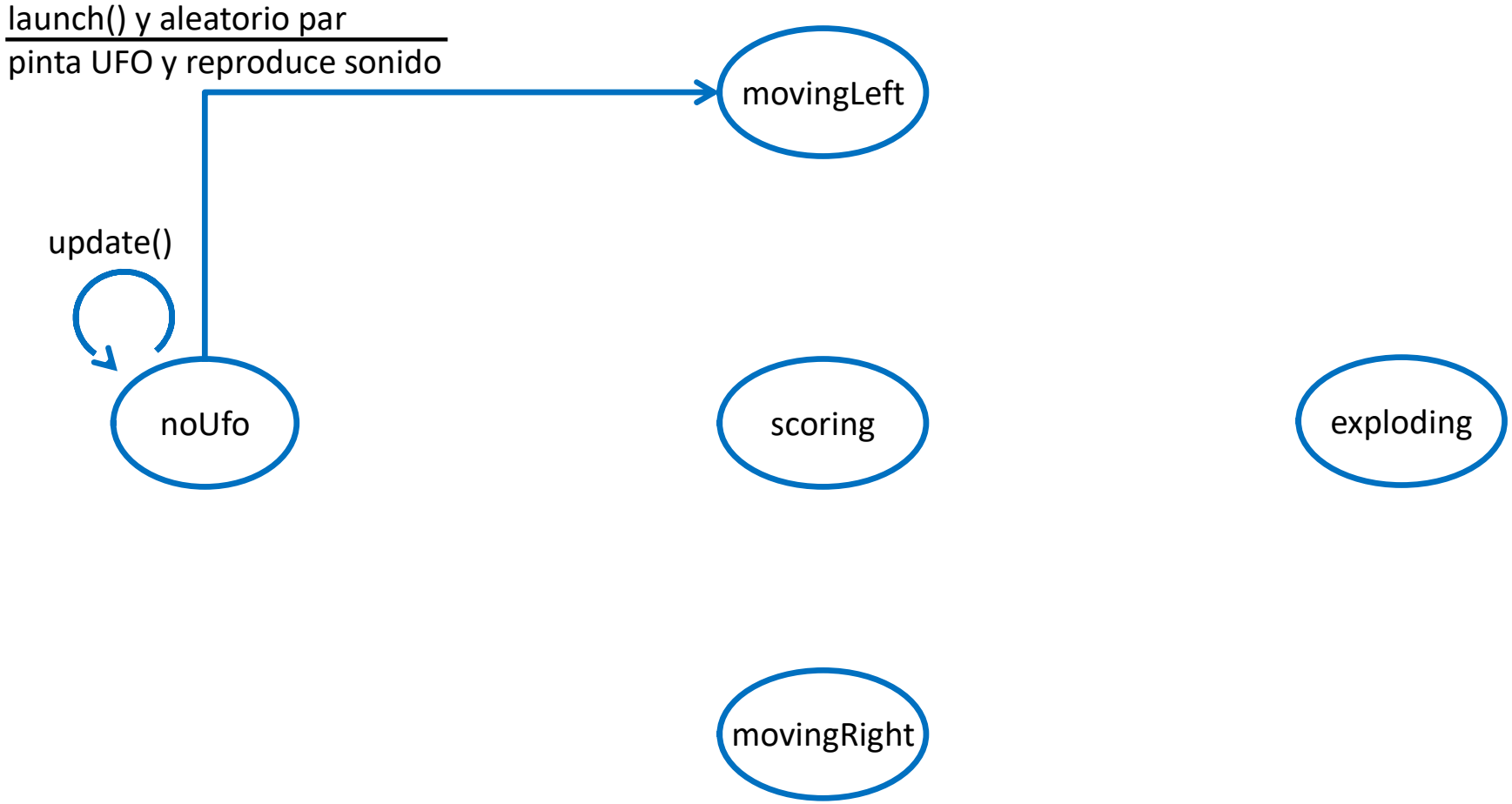
J.M. Mendías
2025

C orientado a objetos

clase *Ufo:* autómata

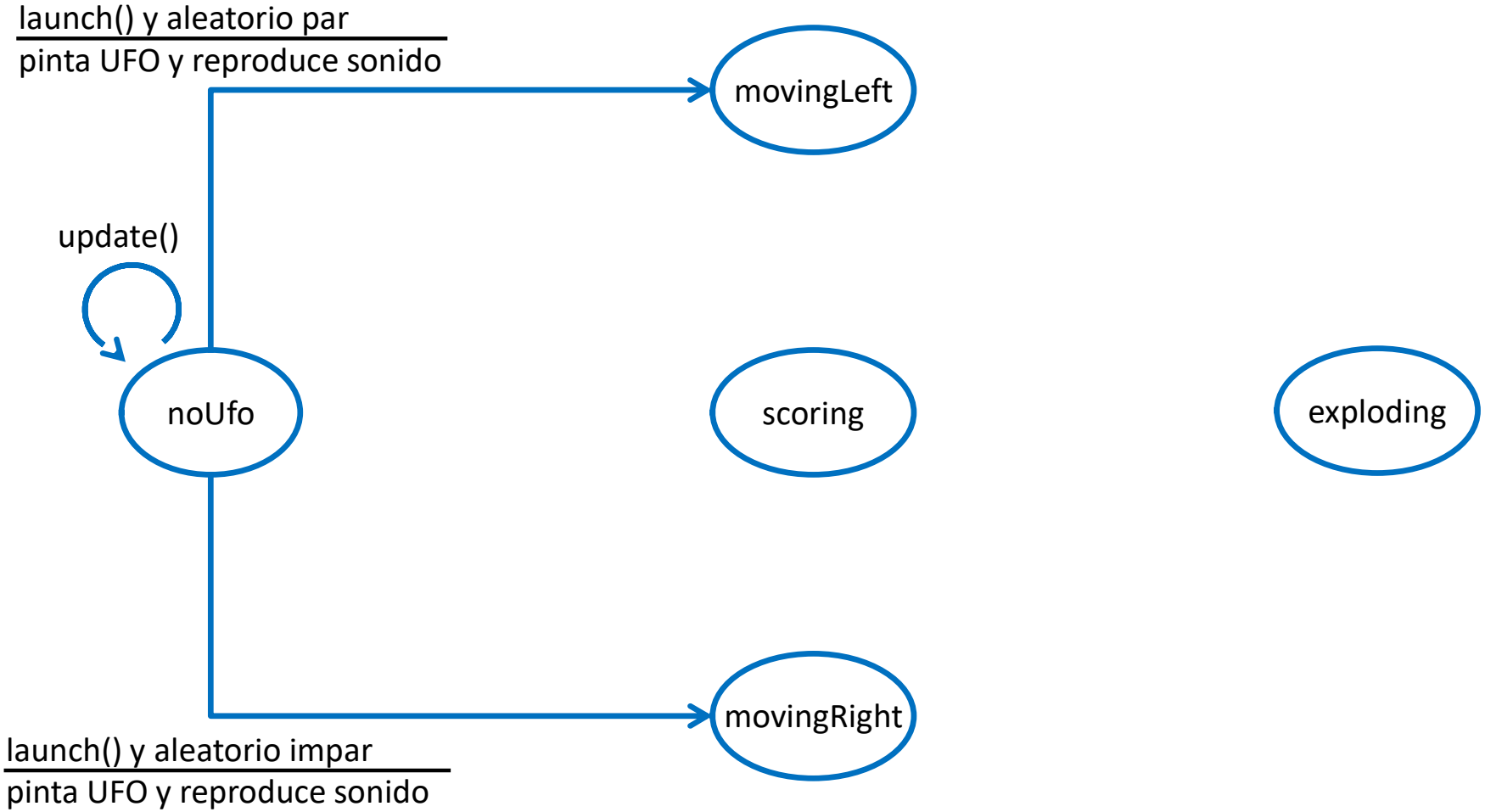
C orientado a objetos

clase *Ufo:* autómata



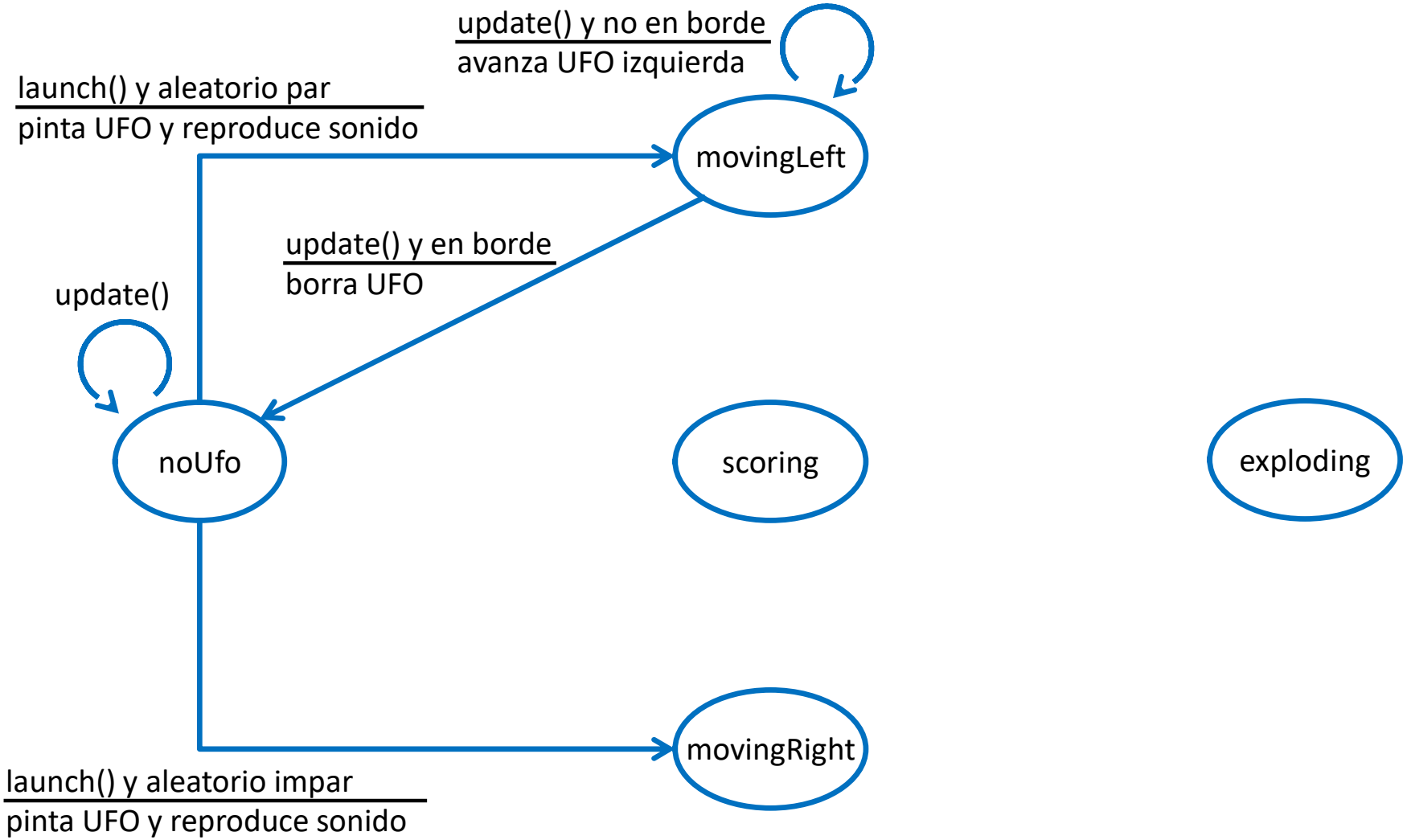
C orientado a objetos

clase *Ufo:* autómata



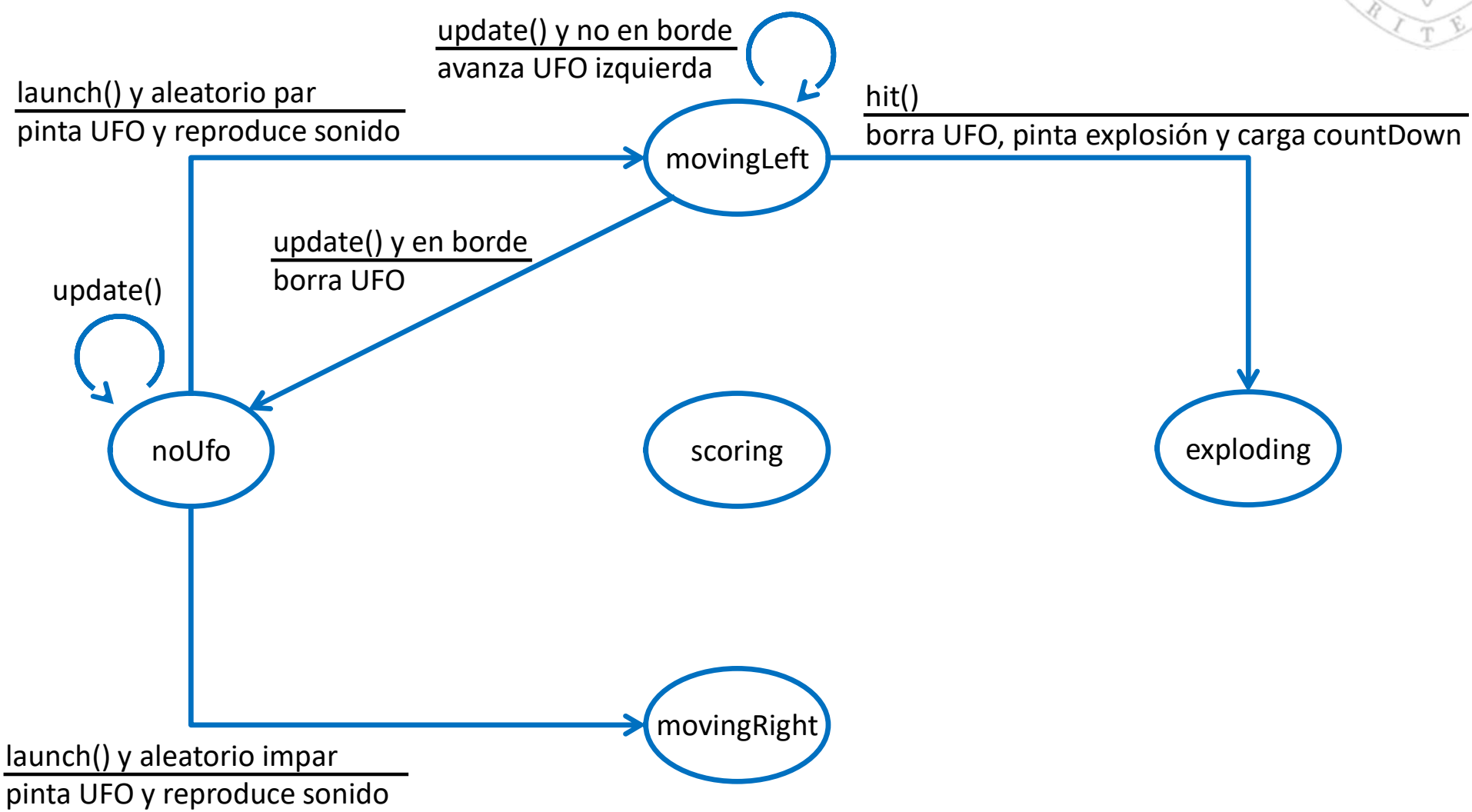
C orientado a objetos

clase *Ufo:* autómata



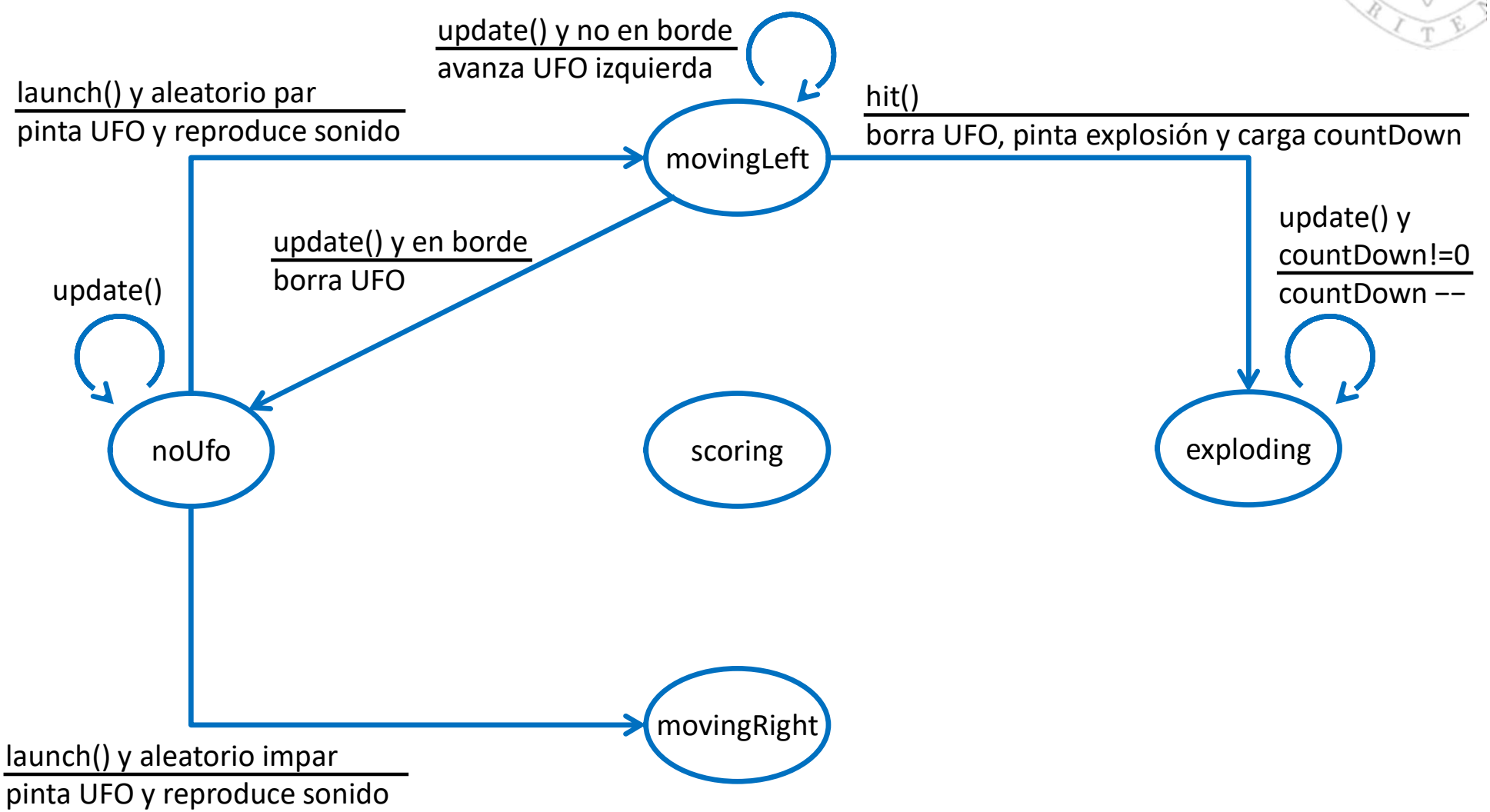
C orientado a objetos

clase *Ufo:* autómata



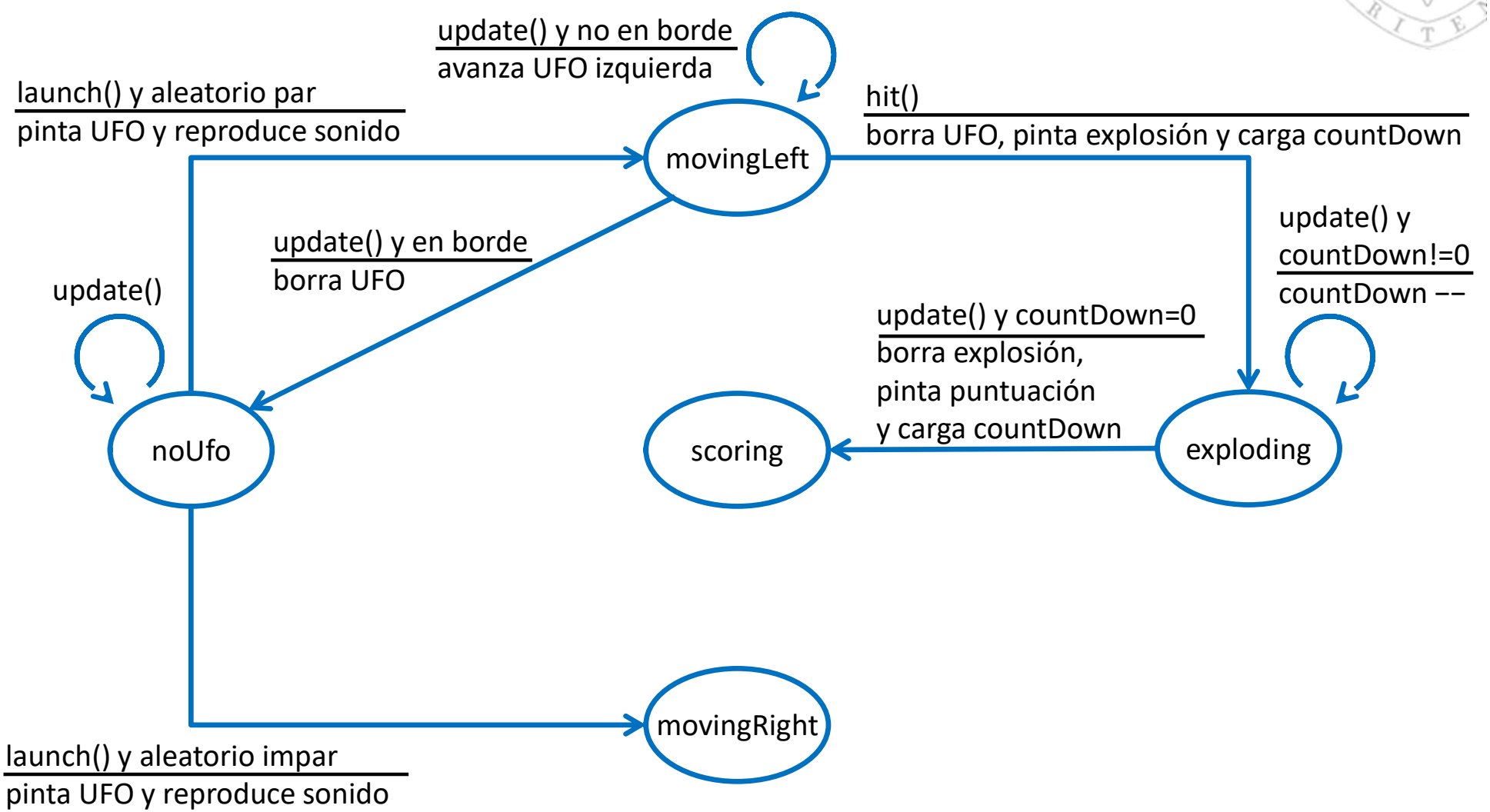
C orientado a objetos

clase *Ufo:* autómata



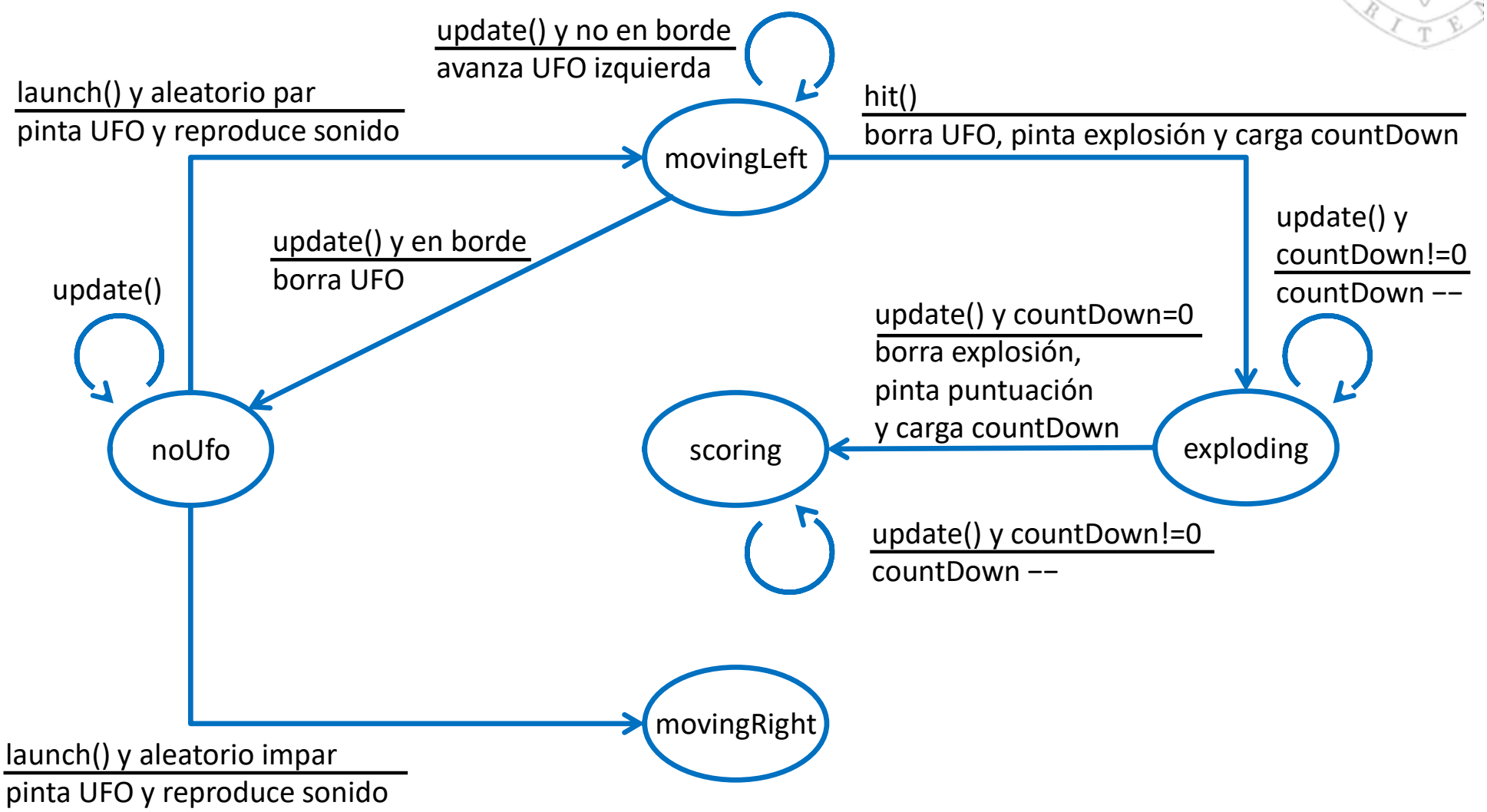
C orientado a objetos

clase *Ufo:* autómata



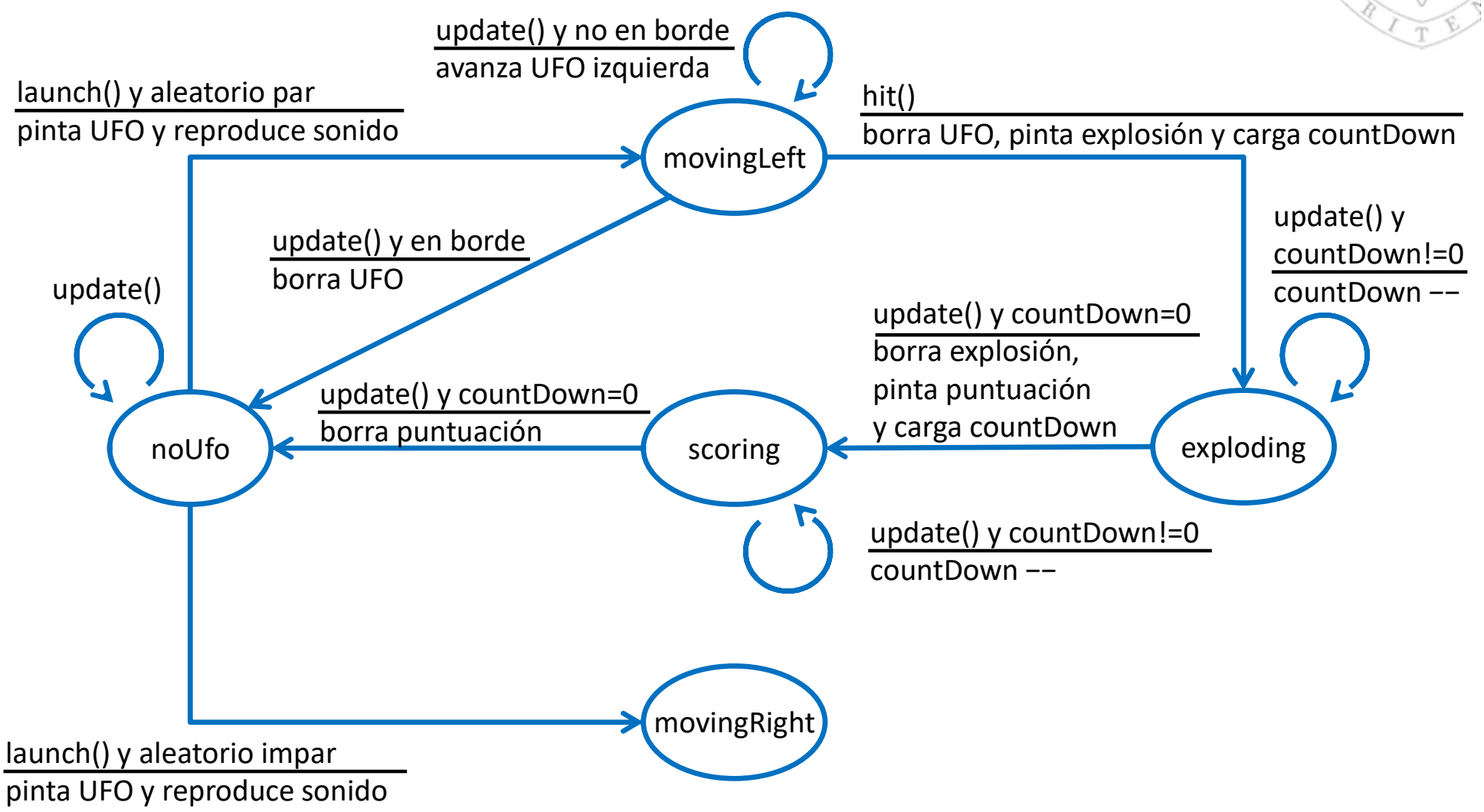
C orientado a objetos

clase *Ufo:* autómata



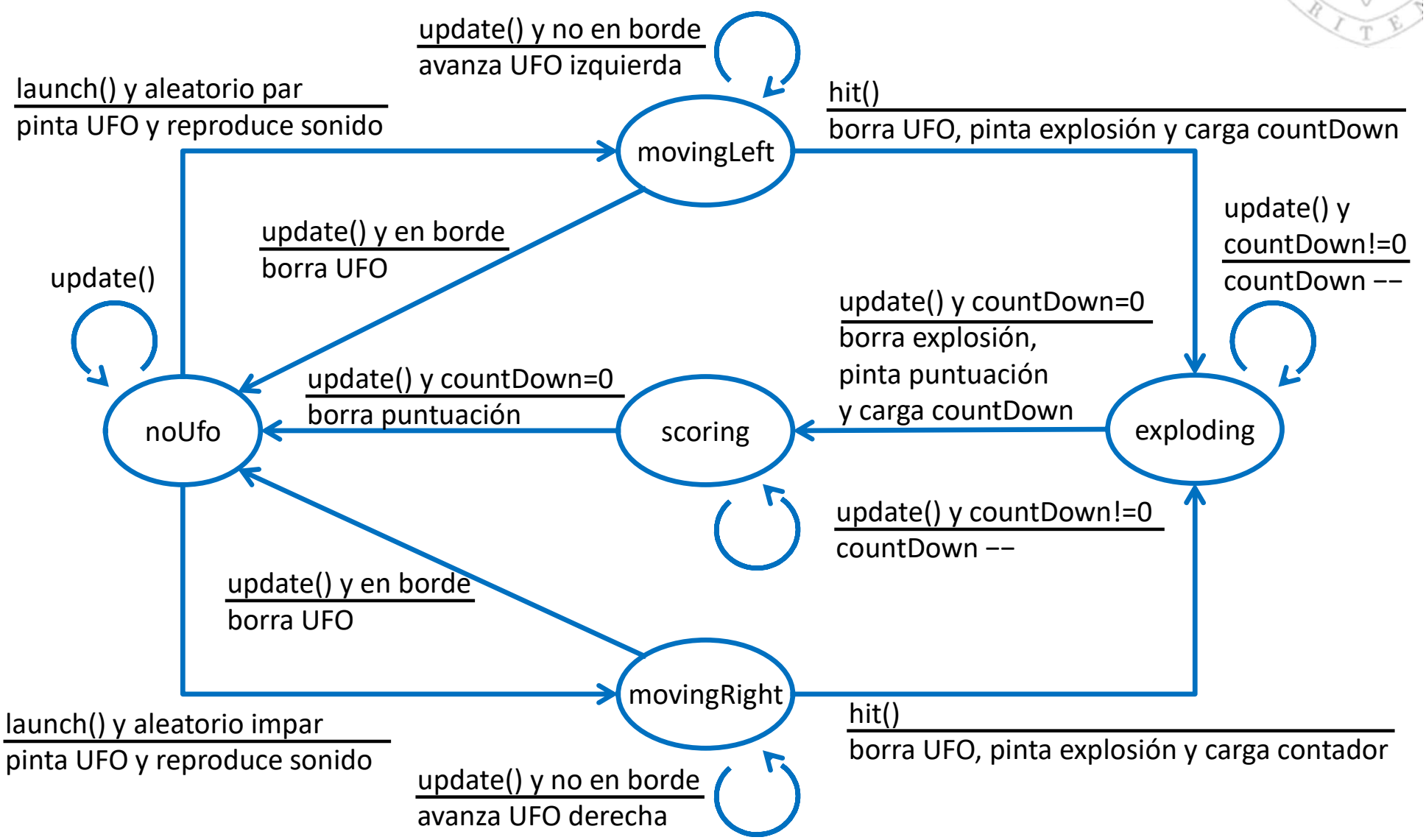
C orientado a objetos

clase *Ufo:* autómata



C orientado a objetos

clase *Ufo:* autómata



J.M. Mendías
2025

C orientado a objetos

clase *Ufo*: ufo.c

```
static void ufo_draw( Ufo *self )
{
  switch (self->state)
  {
    case ufoMovingLeft:
    case ufoMovingRight:
      sprite_draw( &self->sprite, self->col, self->row );
      break;
    case ufoExploding:
      sprite_draw( &self->explosionSprite, self->col, self->row );
      break;
    ...
  }
}

static void ufo_clear( Ufo *self )
{
  switch (self->state)
  {
    case ufoMovingLeft:
    case ufoMovingRight:
      sprite_clear( &self->sprite, self->col, self->row );
      break;
    ...
  }
}
```

J.M. Mendías
2025

C orientado a objetos

clase *Ufo*: ufo.c


```
void ufo_launch( Ufo *self )
{
  if ( self->state == noUfo )
  {
    if (random_get() & 0x1)
    {
      self->state = ufoMovingLeft;
      self->col   = UFO_MAX_COL;
    }
    else
    {
      self->state = ufoMovingRight;
      self->col   = UFO_MIN_COL;
    }
    self->score = scoreTable[random_get()&0xF];
    ufo_draw( self );
    sound_play( &self->launchSound );
  }
}
```

C orientado a objetos

clase *Ufo*: ufo.c

```
void ufo_update( Ufo *self )
{
  switch (self->state)
  {
    case ufoMovingLeft:
      ufo_clear( self );
      if ((self->col - UFO_ADVANCE_COL) >= UFO_MIN_COL)
      {
         self->col -= UFO_ADVANCE_COL;
         ufo_draw( self );
      }
      else
         self->state  = noUfo;
      break;
      ...
  }
}
```

```
void ufo_hit( Ufo *self )
{
  ufo_clear( self );
  self->countDown = UFO_EXPLODING_TIME/UFO_UPDATE_PERIOD;
  self->state = ufoExploding;
  ufo_draw( self );
  sound_play( &self->explosionSound );
}
```

J.M. Mendías
2025

C orientado a objetos

otras clases

Lives
+ value : uint16 + col : int16 + row : int16
+ init() + launch() : void + update(num : uint16) : void – draw() : void

Score
+ value : uint16 + col : int16 + row : int16
+ init() + launch() : void + update(num : uint16) : void – draw() : void

Credit
+ value : uint16
+ init() + launch() : void + update(num : uint16) : void – draw() : void

HiScore
+ value : uint16 + col : int16 + row : int16
+ init() + launch() : void + update(num : uint16) : void – draw() : void – load() : void – store() : void

J.M. Mendías
2025

C orientado a objetos

otras clases

Sprite
− width : uint16 − height : uint16 − pixMap : uint8*
+ draw(col : uint16, row : uint16) : boolean + clear(col : uint16, row : uint16) : void

Sound
− wav : int16*
+ play() : void + loop() : void + stop() : void + isPlaying() : boolean

Shield
+ col : int16 + row : int16 + sprite : Sprite
+ init(col : uint16) + launch() : void

C orientado a objetos

otras clases

Player	
+ state : { stopped | movingLeft | movingRight | exploding | dead }	
+ countDown : uint8	
+ times : uint8	
+ col : int16	
+ row : int16	
+ sprite : Sprite	
+ explosionSpriteSet : uint8	
+ explosionSprite[2] : Sprite	
+ explosionSound : Sound	
+ lives : Lives	
+ score : Score	
+ init()	+ stop() : void
+ launch() : void	+ hit() : void
+ update() : void	+ invaded() : void
+ left() : void	− draw() : void
+ right() : void	− clear() : void

C orientado a objetos

otras clases

Enemy
+ state : { alive | exploding | dead } + type : { alien | metroid | squid } + score : uint8 + col : int16 + row : int16 + spriteSet : uint8 + sprite[2] : Sprite + explosionSprite : Sprite + explosionSound : Sound
+ init(type : enemy_type_t, spriteSet : uint8, col : uint16, row : uint16) + launch() : void + left() : void + right() : void + down() : void + hit() : void + kill() : void − draw() : void − clear() : void

C orientado a objetos

otras clases

Swarm
+ state, stateBeforeExploding : { movingLeft | movingRight | movingDownThenLeft | movingDownThenRight | oneEnemyExploding | dead } + countDown : uint8 + enemies[SWARM_YLEN][SWARM_XLEN] : Enemy + enemiesRemaining : uint8 + toMoveX : uint8 + toMoveY : uint8 + exploding : Enemy * + leftFront : Enemy * + rightFront : Enemy * + downFront : Enemy * + stepSound[4] : sound_t
+ Swarm() + launch() : void + update(player : Player) : void + getShooter() : Enemy * + hit(enemy : Enemy) : void − findNextAlive() : boolean − moveSound() : void − updateLeftFront() : void − updateRightFront() : void − updateDownFront() : void

J.M. Mendías
2025

C orientado a objetos

otras clases

PlayerShot
+ state : { noShot | movingUp | explodingCeiling | explodingShield } + countDown : uint8 + col : int16 + row : int16 + sprite : Sprite + explosionSprite : Sprite + launchSound : Sound + shooter : Player
+ init() + launch(shooter : Player) : void + update(shield : Shield, swarm : Swarm, enemyShot : EnemyShot, ufo : Ufo) : void + onShield(shield : Shield) : void + onUfo(ufo : Ufo) : void + onSwarm(swarm : Swarm) : void + onEnemyShot(enemyShot : EnemyShot) : void − draw() : void − clear() : void

J.M. Mendías
2025

C orientado a objetos

otras clases

EnemyShot
+ state : { noShot | movingDown | explodingFloor | explodingShield | exploding } + countDown : int8 + col : int16 + row : int16 + sprite : Sprite + explosionSprite : Sprite
+ init() + launch(col : uint16, row : uint16) : void + update(shield : Shield, player : Player) : void + onShield(shield : Shield) : void + onPlayer(player : Player) : void + hit() : void − draw() : void − clear() : void

J.M. Mendías
2025

C orientado a objetos

otras clases

Game
+ state : { run | pause } + shield[MAX_SHIELDS] : Shield + player : Player + playerShot : PlayerShot + swarm : Swarm + enemyShot : EnemyShot + ufo : Ufo + hiScore : HiScore + credit : Credit + music : Sound
+ init() + launch() : void + restart() : void

Aplicación multihebra

- La aplicación tendrá una arquitectura multitarea cooperativa bajo un *kernel* de planificación no expropiativo:
  - Un tick de sistema de 100 Hz (10 ms de resolución temporal)

- Existirán las siguientes tareas periódicas:
  - Lectora de *keypad* por *pooling* periódico: 50 ms
  - Lectora de *pushbuttons* por *pooling* periódico: 30 ms
  - Lanzadora del UFO: 20 s
  - Lanzadora del disparo enemigo: 2 s
  - Actualizadora de la posición de la nave espacial del jugador: 20 ms
  - Actualizadora de la posición del enjambre: 20 ms
  - Actualizadora de la posición del UFO: 200 ms
  - Actualizadora de la posición del disparo del jugador: 50 ms
  - Actualizadora de la posición del disparo del enemigo: 50 ms

- El sonido se generará por DMA.

Hitos

- Visualizar elementos estáticos: refugio, vidas, puntuación.
- Visualizar y mover el UFO.
- Visualizar y mover la nave del jugador.
- Visualizar y mover el disparo del jugador.
- Visualizar explosión de disparo del jugador con techo.
- Detectar colisión de disparo del jugador y UFO.
- Visualizar explosión UFO y scoring. Actualización de puntuación.
- Detectar colisión de disparo del jugador y refugio.
- Visualizar erosión de refugio por disparo de jugador.
- Visualizar enjambre.
- Detectar colisión de disparo del jugador y enemigo.
- Visualizar explosión del enemigo. Actualización de puntuación.

Hitos

- Visualizar y mover disparo del enemigo.
- Visualizar explosión de disparo del enemigo con suelo.
- Detectar colisión de disparo del enemigo y refugio.
- Visualizar erosión de refugio por disparo enemigo.
- Detectar colisión de disparo del enemigo y nave del jugador.
- Detectar colisión de disparos (jugador y enemigo).
- Visualizar explosión de disparos.
- Visualizar explosión de nave del jugador. Actualización vidas.
- Mover enjambre en bloque. Gestión de invasión.
- Mover enjambre enemigo a enemigo.
- Gestión del juego: parada, reinicio, pantallas de bienvenida y gameover.
- Extras.

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

    Reconocimiento (*Attribution*):
    En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

    No comercial (*Non commercial*):
    La explotación de la obra queda limitada a usos no comerciales.

    Compartir igual (*Share alike*):
    La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/