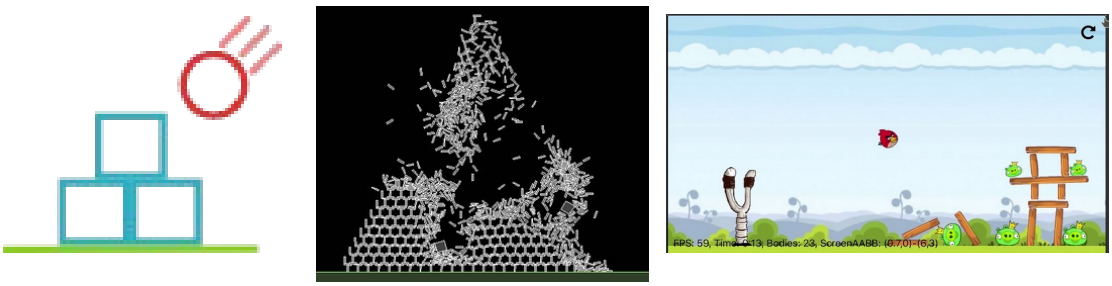
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Revisión de un ejemplo de uso de Box2D. Caso de uso en plataformas 3DS, Switch y PC

Apellidos, nombre	**Agustí i Melchor, Manuel** (magusti@disca.upv.es)
Departamento	**Departamento de Informática de Sistemas y Computadores (DISCA)**
Centro	Universitat Politècnica de València

1 Resumen de las ideas clave

Box2D [1] es una biblioteca que implementa un motor de físicas 2D que se emplea en videojuegos fue escrita en C++ por Erin Catto y publicado como *open source* con licencia MIT. Es multiplataforma y, posiblemente, lo más conocido sea su uso por el juego *Angry Birds* [1], véase la Figura 1.



a) *b)* *c)*

Figura 1: Box2D: (a) Logo[1], (b) Wiki <https://es.wikipedia.org/wiki/Box2D> y (c) el juego de "Angry Birds" que la utiliza <https://www.youtube.com/watch?v=GvA9bUhtjIw>.

Esta biblioteca nos permite simular las físicas de objetos 2D, como las colisiones y la gravedad. Está disponible tanto para los distintos SDK de desarrollo en los sistemas operativos habituales del computador de escritorio, como para el SDK *homebrew* de desarrollo para las plataformas de Nintendo 3DS y *Switch*. Este artículo presenta la reescritura de un ejemplo de uso de Box2D sobre 3DS de forma que sea portable a otras plataformas, lo que vamos a comprobar que es posible para *Switch* y para el PC sin grandes cambios. Para portar el ejemplo hay que implementar dos partes: por un lado, el uso de Box2D que se encargará del cálculo de las posiciones de los objetos y, por otro, cómo llevar a cabo la representación gráfica de la escena. La disponibilidad de Box2D en las tres plataformas enunciadas nos asegura que se puede hacer la simulación física. La parte de visualización hay que reescribirla para que no sea una aplicación nativa de 3DS y, tras buscar alternativas para la implementación, en las tres plataformas, se ha optado por reescribirlo con *Simple Directmedia Layer* (SDL) [2]. Un detalle: en 3DS disponemos de la versión SDL 1.2.15, en *Switch* la SDL 2.0 y, en el computador de escritorio, cualquiera de las dos. A la fuerza, habrá pequeñas diferencias en el código para cada plataforma. Lo veremos.

2 Objetivos

Una vez que el alumno haya leído el documento y explorado la aplicación de ejemplo que se expone aquí, será capaz de:

- Describir los elementos más básicos en una aplicación que utiliza la biblioteca Box2D.
- Utilizar SDL para dar una interfaz gráfica a las simulaciones que calcula Box2D.

- Comprobar que es posible trasladar la aplicación entre las plataformas 3DS, *Switch* y el computador, con mínimos cambios.

3 Introducción

Examinemos los elementos que participan en esta pequeña aplicación para poder encajar cada uno de los elementos de que está compuesta nuestra solución.

3.1 Ejemplo de partida en 3DS

Lo que nos muestra esta aplicación [Error: no se encontró el origen de la referencia6] es el "Hello Box2D" [1] que aparece en el propio sitio web de Box2D, con un interfaz gráfico que utiliza el API de Citro2D (nativo de 3DS), para representarlo de forma gráfica en pantalla. Así que la aplicación tiene dos partes: una la que ejecuta la simulación de un mundo con gravedad en el que hay un objeto fijo (a modo de suelo) y un objeto dinámico que se deja caer libremente y, la otra, la que toma estos resultados de posición de los objetos en el mundo para representarlos en pantalla.

Podemos ver el resultado que se obtiene al ejecutar la aplicación en la Figura 2: la pantalla superior no se utiliza; en la inferior se observa un rectángulo verde (el suelo) y un cuadrado rojo (el objeto dinámico) del que se puede cambiar el punto de partida, tocando en la pantalla táctil. Los mensajes muestran la variación de los valores que definen el mundo.

a) *b)*

Figura 2: Captura de dos instantes de la ejecución del ejemplo physics/box2d de 3DS: (a) inicial y (b) final.

Ahora que ya hemos presentado la aplicación en plataforma 3DS, vamos a plantear una reimplementación que permita la portabilidad de esta aplicación a *Switch* o al computador de escritorio.

3.2 ¿Qué es la biblioteca Box2D?

Box2D está compuesta [3] por tres módulos, como se muestra en la Figura 3a que indica las relaciones entre ellos y que son:

- *Common*. Es el básico que contiene las operaciones de gestión de memoria, operaciones matemáticas y configuración del sistema.
- *Collision*. Es el que define diferentes tipos de polígonos y cómo se calculan las colisiones.
- *Dynamics*. Es el que calcula la simulación utilizando entidades que modelan el sistema o la escena (*world*), los objetos (*bodies*) y sus características (*fixtures* y *joints)*.
    - *Fixtures* son propiedades que van asociadas a los objetos, definiendo para estos la geometría de los polígonos que describen su forma exterior (pudiendo ser más de una) y otros datos no geométricos, como la fricción, o para filtrar cuándo han de dar indicación de que han "tocado" o colisionado con otro objeto. En otras aplicaciones se las denomina *colliders*.
    - *Joints* son las juntas o conexiones entre dos objetos para indicar que existe una relación entre ellos y que el movimiento de uno, implica el del otro.

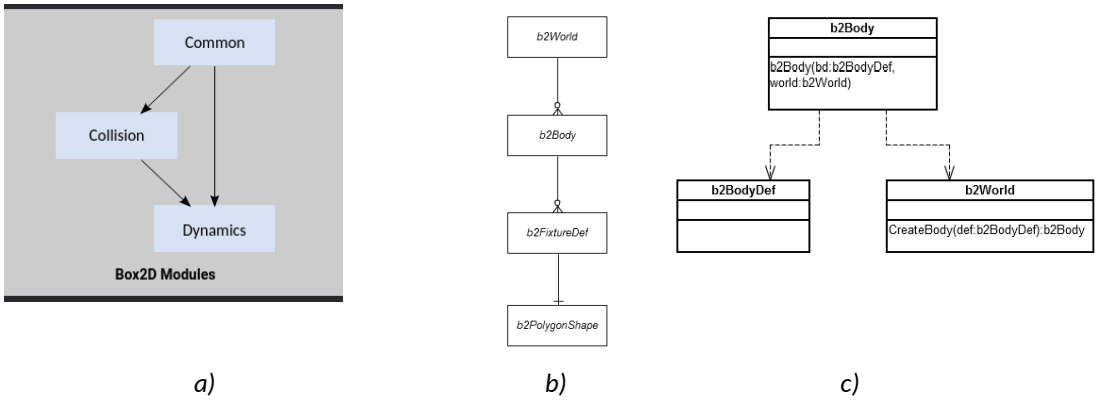


Figura 3: Elementos que componen Box2D: (a) Módulos [3], (b) relaciones entre los objetos del módulo "Dinamics" y (c) relaciones entre las clases.
Imágenes de <http://devmag.org.za/2011/05/27/box2dflash-tutorial/> .

En la Figura 3b se muestra el diagrama mínimo de una simulación Box2D basada en la creación de un mundo (*b2World*) y un objeto (*b2Body*), este a su vez se define en base a otros elementos definidos por sus características (*b2FixtureDef* y *b2PolygonShape*). Para definir los objetos, tanto dinámico como estático, se sigue la misma estrategia: primero definimos la posición, el tipo, etc. en un *b2BodyDef*, luego creamos un cuerpo con las definiciones realizadas anteriormente y, finalmente, se crea un elemento con las dimensiones y demás atributos adicionales como la densidad del objeto, fricción, forma, elasticidad... Para crear la simulación es necesario ejecutar una serie de instrucciones que llamarán a los métodos que implementan las clases que definen estos elementos, como se indica en la Figura 3c.

4 Desarrollo

Como se ha indicado, hay dos partes claramente diferenciadas en la aplicación que estamos observando: por un lado una simulación de las características físicas de unos objetos en un mundo (con bastante parecido al real) y, por otro, una interfaz gráfica que visualiza esa escena, para que sea más fácil de entender e interactuar con ella.

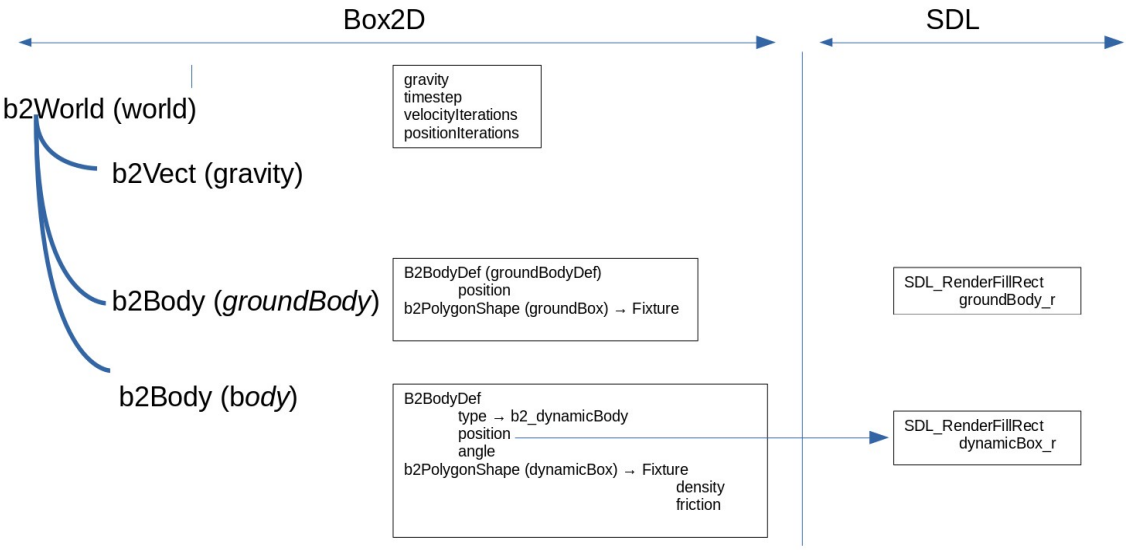


Figura 4: Esquema de desarrollo de la aplicación, utilizando SDL para hacer totalmente portable el resultado.

De las dos partes de la aplicación, Figura 4, Box2D está disponible tanto en 3DS, como en *Switch* y en Linux (va a ser esta nuestra plataforma en el computador, sin pérdida de generalidad). La base para crear una aplicación con Box2D es crear un mundo con, al menos, un valor para la propiedad de gravedad que gobernará el movimiento básico de todos los objetos que se definan dentro de este. ¿Recuerdas la Figura 2, estimado lector? En este caso, tenemos un objeto estático ("el suelo") y un objeto dinámico (un cuadrado rojo) que cae hasta alcanzar, recuerde la Figura 2, el suelo.

Veremos cómo en el bucle principal se va ejecutando la simulación y, a cada paso, habremos de obtener los valores de posición del objeto dinámico, el que se mueve, para actualizar la visualización en pantalla. Habrá que actualizarlos desde Box2D y recogerlos desde la parte de SDL. Es por este motivo que la información ha de fluir desde la parte de Box2D a la de SDL, como se muestra en la Figura 4, básicamente para el objeto dinámico (*body*) puesto que el suelo (*groundBody*) no se mueve en este ejemplo. Vamos a entrar ya a detallar la representación gráfica basada en el uso de la biblioteca SDL que hemos realizado:

4.1 Interfaz gráfico realizado para *Switch*

En el caso de la plataforma *Switch* lo que corresponde hacer es mantener el código del "Hello World" [3] y utilizar las primitivas gráficas de SDL 2.0 para mostrar el resultado en pantalla. Vamos a adaptar el código de la 3DS cambiando la interfaz gráfica a SDL que está disponible en las tres plataformas que nos hemos propuesto comprobar que es viable el desarrollo.

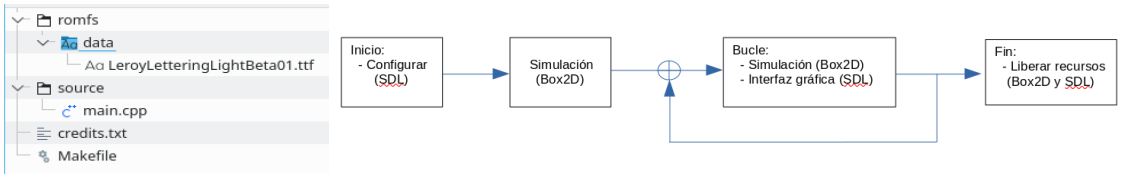


Figura 5: El proyecto: (a) estructura en disco y (b) esquema de bloques del código a realizar.

Vamos a ver el código para ir comentando dónde hemos ido interviniendo. La Figura 5a muestra el contenido del proyecto que estamos generando. Vamos a ir detallando el contenido del fichero de código *main.cpp* aunque no lo veremos aquí completamente por brevedad en la exposición; el código completo está disponible en Github [7]. Su contenido se esquematiza en la Figura 4b como una serie de bloques que van encadenados en secuencia y de los que el tercero puede ejecutarse varias veces, tantas como dure la simulación y como lo decida el usuario.

El primer bloque se recoge en el Listado 1 y muestra la inicialización del código: las cabeceras y las declaraciones de variables, junto con una función (líneas 15 a la 25) que encapsula las acciones para poner un texto en la ventana de SDL. Como lo haremos utilizando un tipo de letra, tenemos el fichero *LeroyLetteringLightBeta01.ttf* (reaprovechando que aparece en otro ejemplo de *devkitPro* para *Switch*) en el subdirectorio *data* dentro de *romfs*.

El siguiente bloque de instrucciones es el contenido de la inicialización del "Hello World" [3] de Box2D, así que no veremos el detalle del código por brevedad en este artículo. En el código en Github [7] sí que está todo y se han mantenido los comentarios del código original. El Listado 2 muestra la inicialización de SDL (líneas 51 a 53) y la carga del fichero del tipo de letra *True Type* que permite mostrar el primer mensaje en pantalla (líneas 54 a 59), junto con la activación del sistema de entrada para recoger los eventos del usuario (líneas 61 a 67) sobre uno de los mandos (*Joy-con*) y la pantalla (que es táctil). Esta interacción no está hecha con SDL para que sea lo más parecido al código original de la 3DS, pero en la versión del computador veremos cómo se utilizan los eventos de SDL para esta acción.

El tercer bloque de la aplicación es el bucle principal, el código del cual está repartido entre los listados 2 y 3. El Listado 2 contiene la parte del bucle de actualizar la simulación desde el punto de vista de Box2D, línea 73, y la consulta las acciones del usuario que pueden llevar a reiniciar la simulación; porque si este toca la pantalla (lo que se comprueba entre las líneas 75 a la 81), entonces hay que resituar el objeto dinámico y volver a empezar la simulación desde ese punto, lo que se hace destruyendo el objeto actual y creando otro (líneas 82 a 85) con las mismas características. Hemos guardado (líneas 75 a 78) las pulsaciones de botones por el usuario. Aunque no se utilizan de momento, dentro de poco lo abordaremos en un ejercicio. No olvide este punto. La parte del bucle de actualizar la representación gráfica utilizando SDL la tenemos en el Listado 3. Este contiene las instrucciones encargadas de comprobar, en cada iteración, "cómo está el mundo"; para ello, se hace uso de las funciones *GetPosition* y *GetAngle* (lineas 89 y 90) que recogen la posición y el ángulo del objeto, respectivamente. Aunque, de momento, no vamos a entrar a ver cómo influye la variación del ángulo, pero por respetar el ejemplo original, lo hemos dejado ahí.

```
1.  #include <stdio.h>
2.  #include <box2d/box2d.h>
3.  #include <memory>
4.  #include <math.h>
5.
6.  #include <SDL.h>
7.  #include <SDL_mixer.h>
8.  #include <SDL_image.h>
9.  #include <SDL_ttf.h>
10. #include <switch.h>
11.
12. #define SCREEN_WIDTH  1280
13. #define SCREEN_HEIGHT 720
14.
15. SDL_Texture * render_text(SDL_Renderer *renderer, const char* text,
16.                     TTF_Font *font, SDL_Color color, SDL_Rect *rect) {
17.     SDL_Surface *surface;
18.     SDL_Texture *texture;
19.     surface = TTF_RenderText_Solid(font, text, color);
20.     texture = SDL_CreateTextureFromSurface(renderer, surface);
21.     rect->w = surface->w;
22.     rect->h = surface->h;
23.     SDL_FreeSurface(surface);
24.     return texture;
25. }
26.
27. int main(int argc, char **argv) {
28.     int exit_requested = 0, wait = 25;
29.     SDL_Texture *dynamicBuffer_tex = NULL, *groundBody_tex = NULL;
30.     SDL_Rect groundBody_r, dynamicBox_r;
31. #define TAMANY_LLETRA 36
32.     SDL_Rect dynamicBuffer_rect = { 0, 0, SCREEN_WIDTH-TAMANY_LLETRA,
33.                                     TAMANY_LLETRA };
34.     char buffer[160];
35.     SDL_Color colors[] = {
36.         { 128, 128, 128, 0 }, // gray
37.         { 255, 255, 255, 0 }, // white
38.         { 255, 0, 0, 0 },     // red
39.         { 0, 255, 0, 0 },     // green
40.         { 0, 0, 255, 0 },     // blue
41.         { 255, 255, 0, 0 },   // brown
42.         { 0, 255, 255, 0 },   // cyan
43.         { 255, 0, 255, 0 },   // purple
44.     };
45.     float gravetat = 100.0f;
46.     b2Vec2 position;
47.     float angle = 0.0f;
48.
49.     romfsInit();
50.     chdir("romfs:/");
...
```

Listado 1: .Versión del código de box2D_para Switch.c (1° parte).

Borramos la pantalla (líneas 97 y 98) y actualizamos el mensaje que queremos darle al usuario (líneas 100 a 102). El resto del bucle se encarga de pintar el objeto estático (líneas 103 a 107), porque recuerde que hemos borrado la pantalla totalmente, y el dinámico (lí-

neas 109 a 111). Al terminar las operaciones de dibujado, pedimos al sistema que actualice el contenido de la pantalla (líneas 112 y 113).

```
...
51.     SDL_Init(SDL_INIT_VIDEO|SDL_INIT_TIMER);
52.     SDL_Window* window = SDL_CreateWindow("Box2D + SDL 2.0",
                        SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                        SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);
53.     SDL_Renderer* renderer = SDL_CreateRenderer(window, -1,
                                                SDL_RENDERER_ACCELERATED);
54.     TTF_Init();
55.     // load font from romfs
56.     TTF_Font* font = TTF_OpenFont("data/LeroyLetteringLightBeta01.ttf",
    TAMANY_LLETRA );
57.     // render text as texture
58.     dynamicBuffer_tex = render_text(renderer, "Switch: Box2D + SDL
59.  2.0", font, colors[1], &dynamicBuffer_rect);
60.       // Configure our supported input layout: a single player ...
61.     padConfigureInput(1, HidNpadStyleSet_NpadStandard);
62.     // Initialize the default gamepad (which reads handheld mode inputs
63.  as well as the first connected controller)
64.     PadState pad;
65.     padInitializeDefault(&pad);
66.     HidTouchScreenState state={0};
67.     hidInitializeTouchScreen();
68.     // Main loop
69.     while (!exit_requested && appletMainLoop() ) {
70.         // Instruct the world to perform a single step of simulation.
71.         // It is generally best to keep the time step and iterations
72.         // fixed.
73.         world->Step(timeStep, velocityIterations, positionIterations);
74.         // Scan the gamepad. This should be done once for each frame
75.         padUpdate(&pad);
76.         // padGetButtonsDown returns the set of buttons that have been
77.         // newly pressed in this frame compared to the previous one
78.         u64 kDown = padGetButtonsDown(&pad);
79.         // Una pulsació reinicialitza la posició de l'objecte
80.         hidGetTouchScreenStates(&state, 1);
81.         if (state.count > 0) { //!haCanviat && ) {
82.           world->DestroyBody(body);
83.           bodyDef.position.Set(state.touches[0].x,
                                  state.touches[0].y );
84.           body = world->CreateBody(&bodyDef);
85.           body->CreateFixture(&fixtureDef);
86.         }
87. ...
```

Listado 2: Versión del código de box2D_para Switch.c (2º parte).

Y así, hasta que decida el usuario. Para terminar, liberamos recursos (líneas 116 a 119) y cerramos ordenadamente. Si ejecuta el código propuesto verá algo similar al contenido de la Figura 6, que muestra en cada fila la captura correspondiente al mismo momento de la versión para 3DS (a la izquierda) y para *Switch* (a la derecha).

Ejercicio . Explore el código e introduzca algunas pulsaciones de teclas que permitan observar cómo afectan al movimiento del objecto la gravedad, la densidad y la fricción. Tiene la solución para modificar la gravedad en el repositorio en Github [7]. Así que tendrá que des-

cargarlo para comprobarlo. Una pista: en el Listado 2 se ha dejado, en la línea 78, actualizados los valores de los botones que ha podido accionar el usuario, aunque no se han utilizado... Piense en qué variable contiene el valor de la gravedad y busque en la documentación [3] cómo actualizar esa característica del mundo de Box2D.

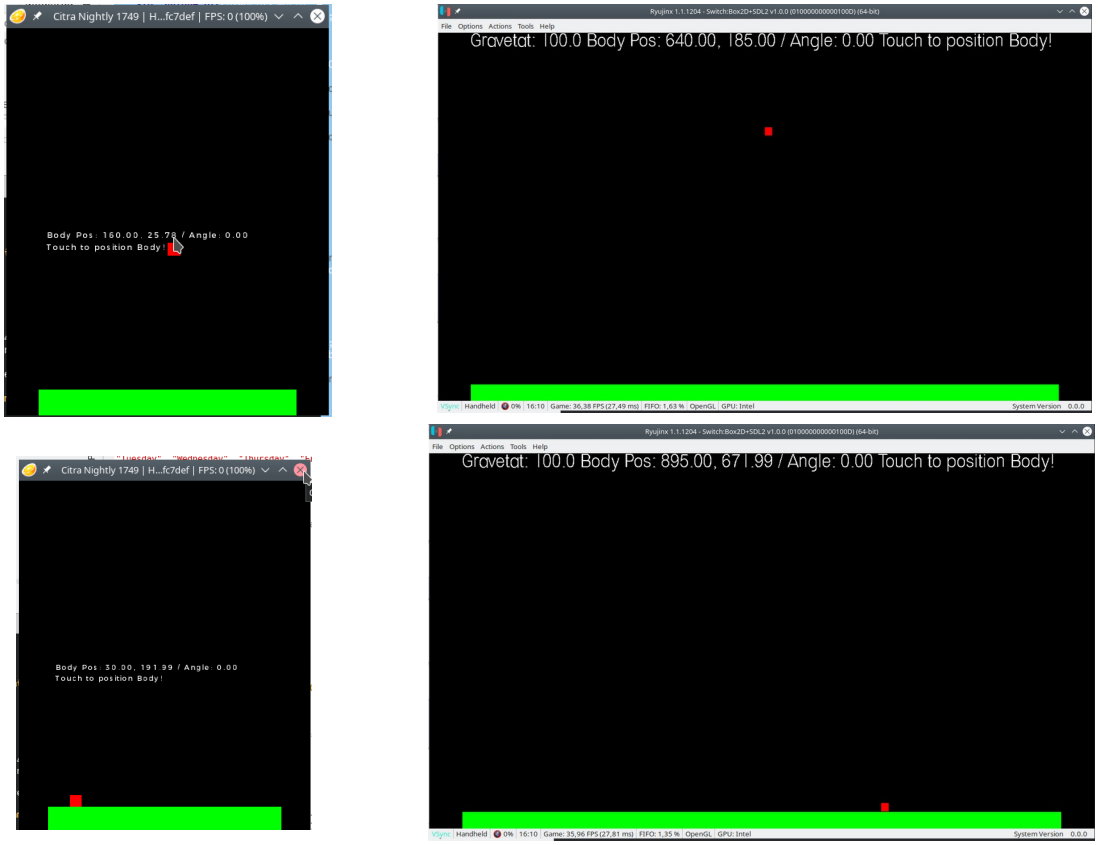
```
    ...
88.         // Now print the position and angle of the body.
89.         position = body->GetPosition();
90.         angle = body->GetAngle();
91.         snprintf(buffer, sizeof(buffer), "Gravetat: %3.1f Body Pos:
92.                %4.2f, %4.2f / Angle: %4.2f\nTouch to position Body!",
93.                                 gravetat, position.x, position.y, angle);
94.         dynamicBuffer_tex = render_text(renderer, buffer, font,
95.                                         colors[1], &dynamicBuffer_rect);
96.         /* Render the scene, clear to black */
97.         SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0xFF);
98.         SDL_RenderClear(renderer);
99.         // put text on screen
100.        if (dynamicBuffer_tex)
101.            SDL_RenderCopy(renderer, dynamicBuffer_tex, NULL,
102.                           &dynamicBuffer_rect);
103.        b2Vec2 groundPos = groundBody->GetPosition();
104.        SDL_SetRenderDrawColor(renderer, 0, 255, 0, 255);
105.        groundBody_r = {(int)groundPos.x, (int)groundPos.y - 32,
106.                        SCREEN_WIDTH, 64};
107.        SDL_RenderFillRect(renderer, &groundBody_r);
108.        SDL_RenderCopy(renderer, groundBody_tex, NULL, &groundBody_r);
109.        /* draw the dynamic box */
110.        SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
111.        dynamicBox_r  = {(int)position.x, (int)position.y, 16, 16};
112.        SDL_RenderFillRect(renderer, &dynamicBox_r);
113.        SDL_RenderPresent(renderer);
114.        SDL_Delay(wait);
115.    } //     while (!exit_requested && appletMainLoop() ) {
116.    TTF_CloseFont(font);
117.    TTF_Quit();
118.    SDL_Quit();
119.    romfsExit();
120.    return 0;
121.}
```

Listado 3: Versión del código de box2D_para Switch.c (3º parte).

Una reflexión sobre los valores de este ejercicio propuesto y lo que sucede en el mundo real: la gravedad[1] es la aceleración con la que un cuerpo, que se encuentra en el campo gravitatorio de otro cuerpo, es atraído; y es proporcional a la fuerza (F=ma) de gravedad. Por lo tanto, las unidades de la gravedad son m/s^2, aunque también puede expresarse en N/kg^2. En la Tierra la gravedad es de 9,81 N/Kg ó m/s^2, en la Luna es de 1,63$m/s^{2,}$ pero deje que el usuario pueda escoger valores de gravedad entre, p. ej., -20 y +20 ó entre -100 y +100 para exagerar el efecto de la gravedad y que sea fácilmente visible.

[1] Sobre la definición, cálculo y unidades de la gravedad consulta en <https://www.ingenierizando.com/dinamica/gravedad/>.

[2] Véase <https://www.ingenierizando.com/dinamica/gravedad/?utm_content=cmp-true> àra ampliar estos detalles.



a) *b)*

Figura 6: Captura de un instante de la ejecución del ejemplo physics/box2d en la versión para: (a) 3DS y (b) Switch.

¿Qué debe suceder? Que si está cayendo el objeto y bajamos el valor, el objeto cae con menos velocidad y si llega a valores negativos... ¡empezará a subir!. Si la gravedad vale cero, el objeto continua con el movimiento que lleva de forma indefinida, pero si pusiéramos un elemento en su camino que lo frene y lo quitamos... ¿Qué pasará? Haga una prueba rápida, modifique el valor de la gravedad hasta que llegue a cero y, entonces, pulse en un punto de la pantalla: ¿qué sucede? ¡El objeto no cae porque no hay gravedad y no hay ninguna fuerza que lo empuje!

4.2 En el computador y en plataforma 3DS

Es importante destacar que el *Makefile* del ejemplo original define el uso de la biblioteca Box2D para la N3DS con las líneas:

```
LIBS    := -lbox2d -lcitro2d -lcitro3d -lctru -lm
```

y

```
LIBDIRS := $(CTRULIB) $(PORTLIBS)
```

En el repositorio [7] puede encontrar las versiones para cada una de estas plataformas, por brevedad de la exposición sólo se detalla aquí lo que respecta a la compilación de los proyectos. Observe como esas líneas han cambiado en la versión para 3DS con SDL y cómo los

cambios son menores para obtener una versión ejecutable en el computador, que podría utilizar tanto SDL 1.2 como SDL 2.0, con una línea del estilo de :

```
$ gcc box2D_linux.c -o box2D_linux `sdl2-config --cflags --libs` -lSDL_ttf
$ box2D_linux
```

Los resultados de ejecutar estas dos versiones deben ser equivalentes a los vistos en la Figura 6, así que por brevedad en la exposición no se muestran nuevas capturas. Lo haremos en el Github [7] donde se publiquen estos contenidos.

5 Conclusión y cierre

A lo largo de este objeto de aprendizaje hemos visto cómo se ha planteado la actualización de un ejemplo que utiliza Box2D para calcular la evolución de objetos en un "mundo" donde existen fuerzas (como la gravedad), que actúan sobre los cuerpos que se definen en la escena, atendiendo a las características deseadas del mundo y de los objetos que se definen en él. Para representarlo gráficamente se ha buscado una solución independiente de la plataforma: SDL ha permitido realizar la versión para la videoconsola *Switch* (con SDL 2.0); al mismo tiempo que se ha utilizado este tándem, Box2D+SDL, para generar las versiones en la videoconsola 3DS (con SDL 1.2) y para el computador de escritorio (donde se ha optado, por ser más actual, por la SDL 2.0). Esto ha permitido comprobar que es posible trasladar la aplicación entre las plataformas 3DS, *Switch* y el computador con mínimos cambios.

Como ha podido comprobar el lector, las diferencias entre en las tres plataformas comparadas en este estudio son obligadas por la diferencia que hay entre el hardware de las tres. En el caso de plataformas más similares en hardware, los cambios se reducen y, si además se utiliza la misma versión de SDL en ambas, el número de cambios puede llegar a ser muy próximo a cero. Espero que haya probado a modificar el código con la propuesta que se ha hecho en un ejercicio y reconstruir las tres versiones. Hay unas cuantas operaciones interesantes del módulo *Collisions* que servirían para extender este contenido, como aplicar fuerzas y detectar colisiones... ¿Te animas, estimado lector?

6 Bibliografía y referencias

[1] Box2D. Sitio web. <http://box2d.org>.

[2] SDL. Sitio web. <https://www.libsdl.org/>.

[3] Box2D Overview. <https://box2d.org/documentation/index.html>.

[4] Nelson M. Moreno. Sitio web. <https://github.com/nelsonmoreno/Box2d/wiki>.

[6] Ejemplo de Box2D del SDK para 3DS. <https://github.com/devkitPro/3ds-examples/tree/master/physics/box2d>.

[7] Repositorio de los ejemplos de este artículo <https://github.com/magusti/3DS/portabilidadSDLBox2D_PC_3DS_Switch>.