Aprender haciendo

Experiencias fuera del termo, laborales y/o lúdicas

Luciano Rodriguez

Experiencias

Desarrollo de un sistema online en Django (Framework de Python) para una fundación que trabaja con personas con discapacidad.

- Tratar con el cliente
- Noches de desvelo
- Sacrificar fechas de finales (no necesariamente)
- Mucho aprendizaje (HTML, CSS, Python, scripts en servidor corriendo Linux)

Desarrollo de videojuego en C un verano que estaba al "vicio" (o casi).

- Diversión
- Mucho aprendizaje

¿Que es un juego? (a grandes rasgos)

- Renderizado (dibujar en pantalla)
- Capturar eventos del jugador
- Lógica del juego (que hacer cuando sucede algo)
- Arte visual y sonido

Mi idea

Hacer un juego en 2d con una herramienta que me provea un punto medio entre darme buen control de lo que ejecuta mi juego/programa para tener buen rendimiento, no programar en OpenGL/DirectX pero tampoco arrastrar y soltar.

¿Qué herramientas existen?

Hay de todo, según el nivel de dificultad que uno esté dispuesto a enfrentar o a cuan bajo o alto nivel uno decida programar, a grosso modo las dividí en dos:

Frameworks completos con entorno de desarrollo

Un par de ejemplos de los más conocidos:

- Unity
- Unreal Engine

Puntos a favor:

- La mayoría de la programación está hecha, como ser las físicas, y agregar personajes al juego (aunque no tan a favor si nos interesa hacer estas cosas nosotros)
- Muy buena portabilidad entre plataformas
- Quizás más adecuado si se quiere hacer algo profesional en un tiempo más corto que el de crear nuestro propio motor.

Frameworks completos con entorno de desarrollo

Un par de ejemplos de los más conocidos:

- Unity
- Unreal Engine

Puntos en contra:

- La dinámica del desarrollo es arbitraria según la pensaron los desarrolladores de cada uno y no suele ser transferible a otros frameworks
- Puede haber incompatibilidades con código más antiguo al salir una nueva versión si uno quiere actualizar para utilizar cosas nuevas.

APIs

Nos proveen procedimientos que se encargan de hacer las cosas de muy bajo nivel, como ser la carga de imágenes y sonido en memoria desde archivos, renderizado de imágenes y texto en pantalla, creación de ventanas, manejo de captura de eventos del teclado/gamepad/mouse, nos brindan funciones a las que llamamos, les damos los parámetros adecuados y hacen las cosas por nosotros.

Algunos de los más usados en 2d:

- SDL (C o C++)
- Allegro (C o C++)
- Cocos2d (Python), Cocos2d-x (C++)
- PyGame (Python)

Mi elección: Allegro

Después de probar SDL y Allegro me quedé con el último, me gustó más el estilo pero son muy similares, elegí entre esos dos porque sentía que eran los más adecuado para el tiempo que tenía y el nivel en el que quería programar.

Por detrás Allegro usa OpenGL, así que la performance es bastante buena también.

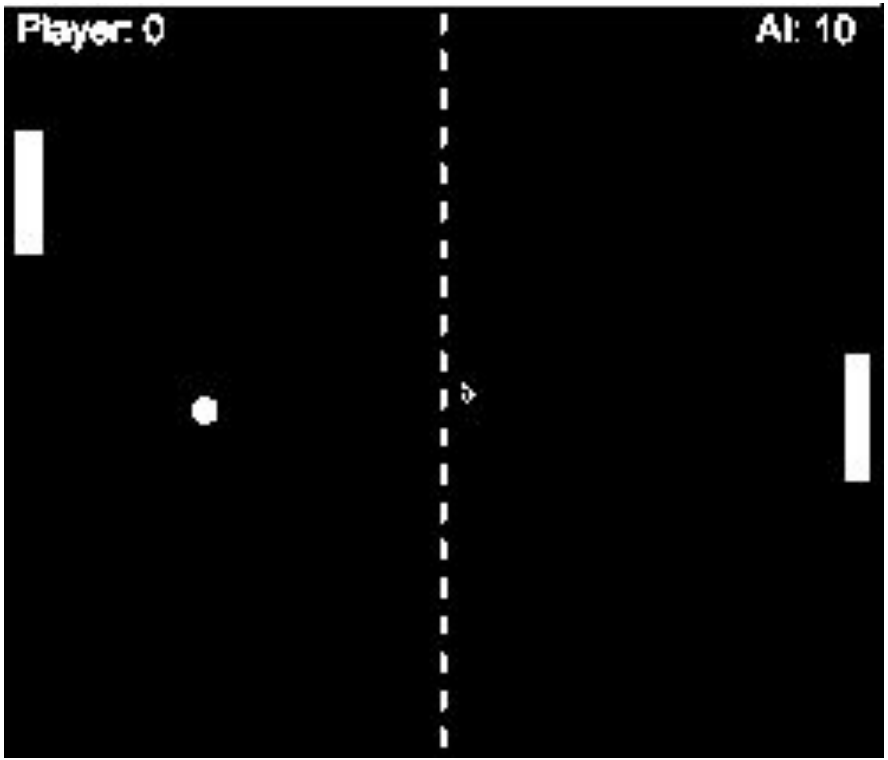
El Hello World de Allegro

Crear una ventana y rellenar con color negro:

```c
#include <stdio.h>
#include <allegro5/allegro.h>

int main(int argc, char **argv){

   ALLEGRO_DISPLAY *display = NULL;

   if(!al_init()) {
      fprintf(stderr, "failed to initialize allegro!\n");
      return -1;
   }

   display = al_create_display(640, 480);
   if(!display) {
      fprintf(stderr, "failed to create display!\n");
      return -1;
   }

   al_clear_to_color(al_map_rgb(0,0,0));

   al_flip_display();

   al_rest(10.0);

   al_destroy_display(display);

   return 0;
}
```

Siempre se recomienda empezar por juegos simples para agarrarle la mano a las APIs, por lo tanto elegí...

Pong!



Pong 1.2

DEMO

Cosas básicas de un juego:

- Renderizado
- Manejo de eventos de input (teclado, mouse, etc)
- Lógica del juego
- Colisiones

Renderizado

Básicamente dibujar en pantalla, lo hacemos cada cierto tiempo para generar la ilusión de animación, por ejemplo 60 FPS serian 60 "dibujadas" por segundo.

Para dibujar un objeto tenemos una función bastante directa:

void al_draw_bitmap(ALLEGRO_BITMAP *bitmap, float dx, float dy, int flags)

Ejemplo en Pong

```
// Creamos una de las "paletas"
ALLEGRO_BITMAP *p1_pad = NULL;
p1_pad = al_create_bitmap(pad_SIZE_W, pad_SIZE_H);
// p1_pad variable u objeto tipo paleta
// p1_pad_x coordenada x, p1_pad_y coordenada y donde queremos que se empiece a dibujar
al_draw_bitmap(p1_pad, p1_pad_x, p1_pad_y, 0);

*
*
*
// Cuando no usamos mas el recurso lo liberamos
al_destroy_bitmap(p1_pad);
```

Mismo estilo que hacer malloc/calloc, y luego al final del programa hacer free() del recurso, lo cual es lo que en realidad está haciendo la API por debajo, aparte de otras cosas.

Manejo de eventos de input

Un evento, como lo dice la palabra, es algo que sucede, nos interesan los eventos referidos a si el usuario apretó alguna tecla para modificar la posición de la paleta.

```
event_queue = al_create_event_queue();
al_register_event_source(event_queue, al_get_keyboard_event_source());

while(!exitgame) {
    *
    *
    *
     // Variable tipo evento que nos provee Allegro
     ALLEGRO_EVENT ev;

     // Nos quedamos a la espera que entre algun evento a la cola de eventos
     al_wait_for_event(event_queue, &ev);

     // Si el evento que entra es que se presionó una tecla
     if(ev.type == ALLEGRO_EVENT_KEY_DOWN) {
         // Chequeamos que tecla se presionó viendo el keycode
         switch(ev.keyboard.keycode) {
             case ALLEGRO_KEY_UP:
                 moverLaPaletaParaArriba(p1_pad);
                 break;
             case ALLEGRO_KEY_DOWN:
                 moverLaPaletaParaAbajo(p1_pad);
                 break;
         }
     }
    *
    *
    *
}
```

Colisiones

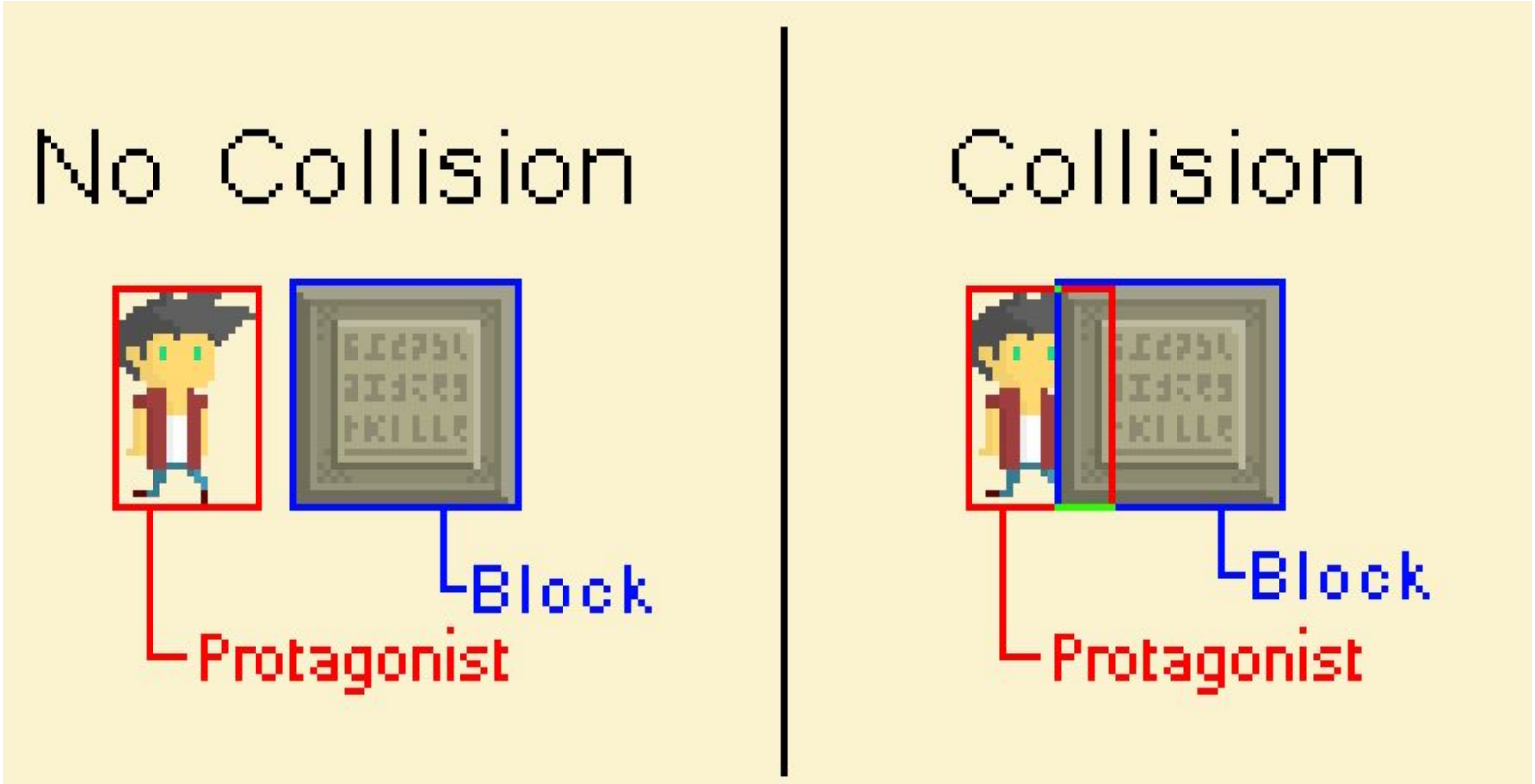
- Nos interesa saber cuando dos objetos del juego interactúan, en este caso la pelota y alguna de las paletas.
- Se llama colisión a la situación donde dos objetos "chocan" o, en la programación, cuando se superponen sus píxeles
- Hay varias formas de chequear por colisiones:

Tipos de chequeo de colisiones

- Pixel Perfect: se fija en cada pixel de cada objeto si está en superposición con cualquier otro pixel de alguno de los demás objetos
  - Este algoritmo detecta toda las colisiones perfectamente pero es super ineficiente, ya que hay que llevar seguimiento de cada pixel de cada objeto en pantalla (ni hablar en 3d)

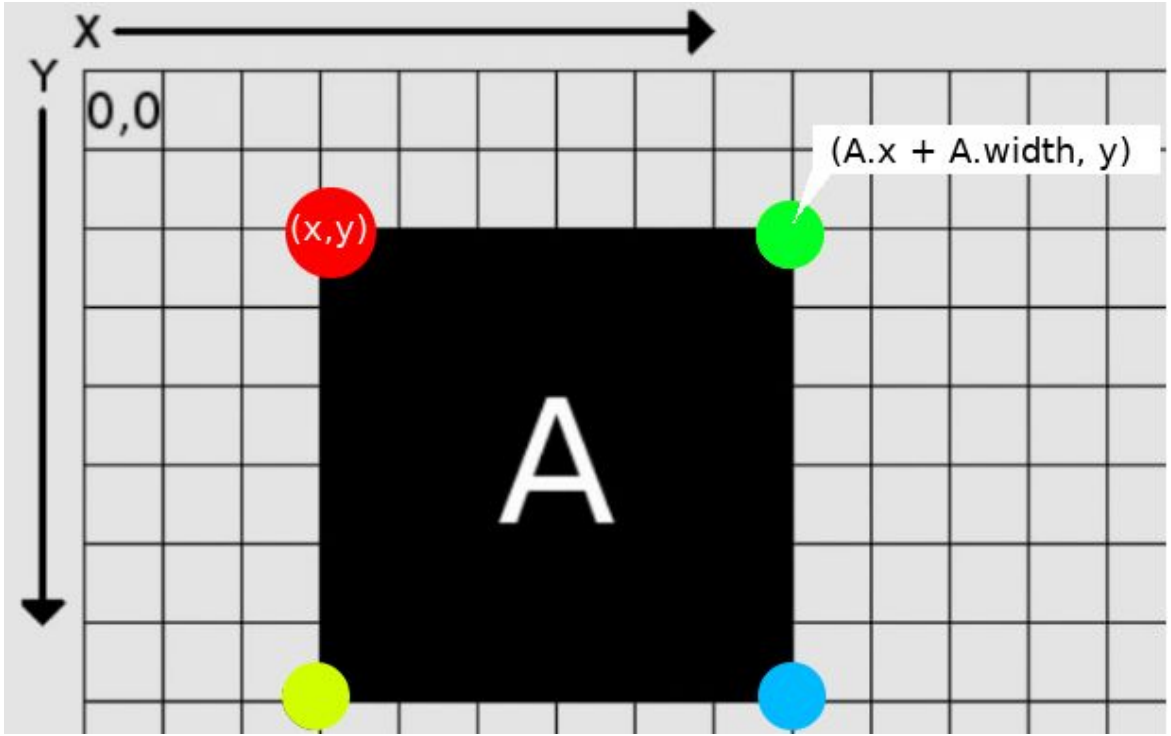
- Bounding Box Collision: de los más utilizados, se dibuja uno o varios rectángulos sobre el personaje y objetos y sólo se chequea si se superponen sus rectángulos, ejemplo visual:

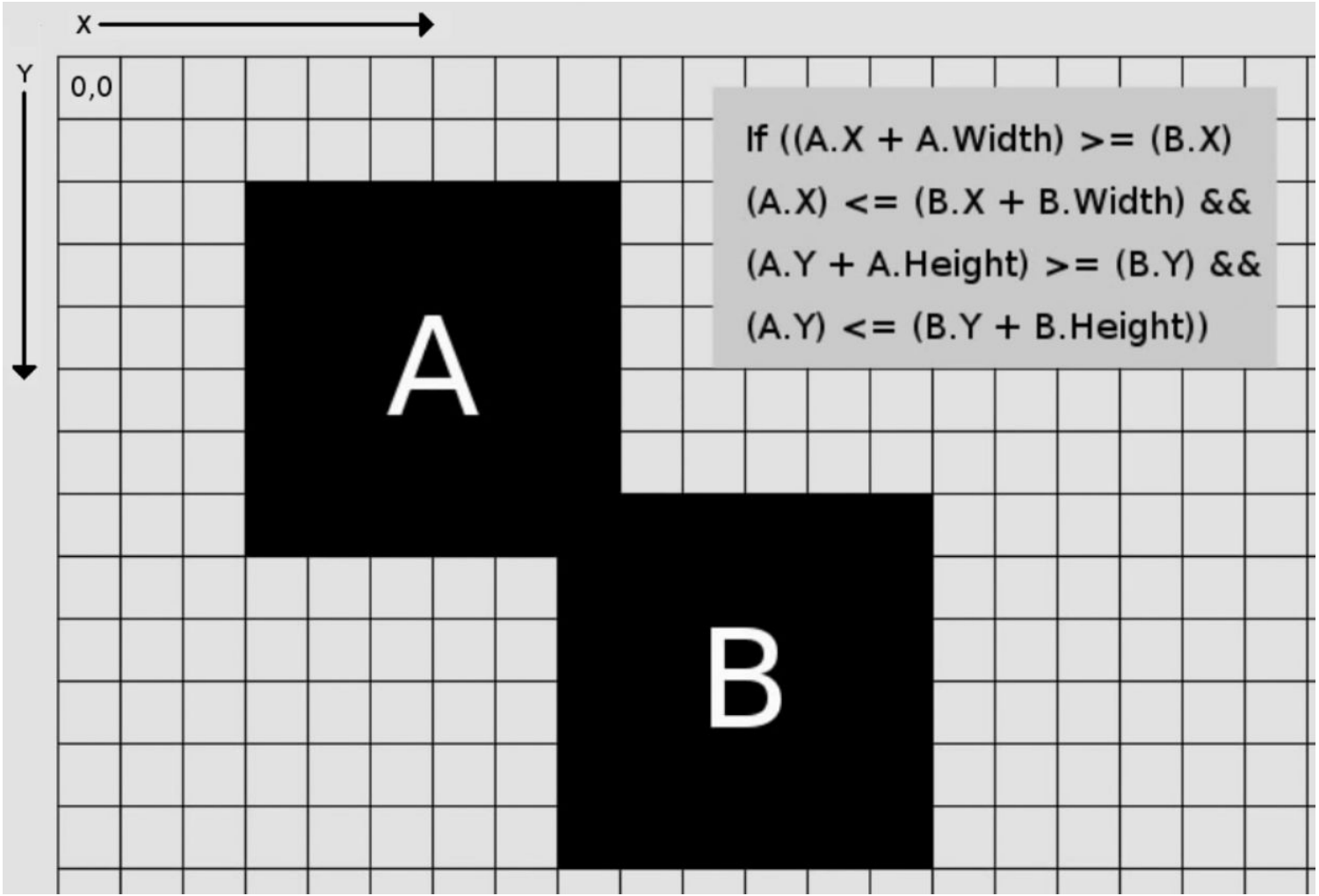
Bounding Box Collision con 1 rectangulo



Bounding Box Collision con 1 rectángulo

Solo necesitamos chequear 4 puntos




X
Y
0,0
A
B
If ((A.X + A.Width) >= (B.X)
(A.X) <= (B.X + B.Width) &&
(A.Y + A.Height) >= (B.Y) &&
(A.Y) <= (B.Y + B.Height))

Bounding Box Collision usando más rectángulos

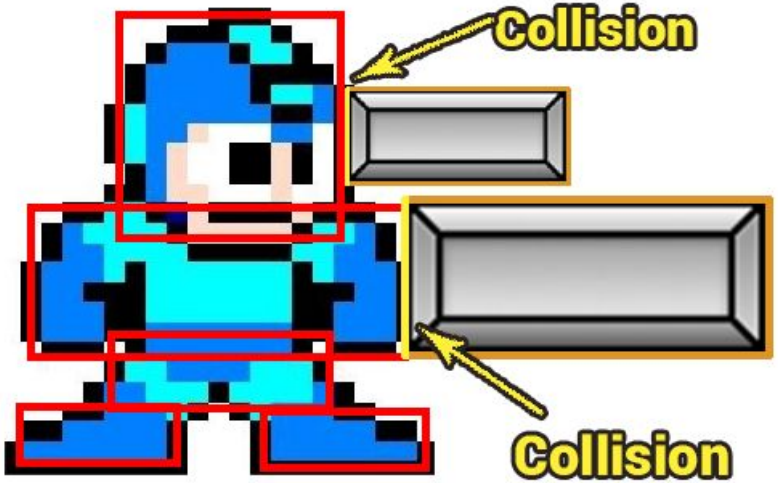
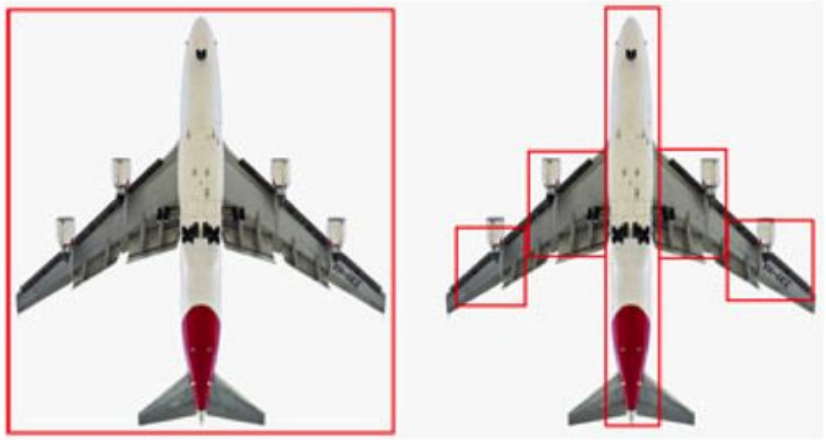
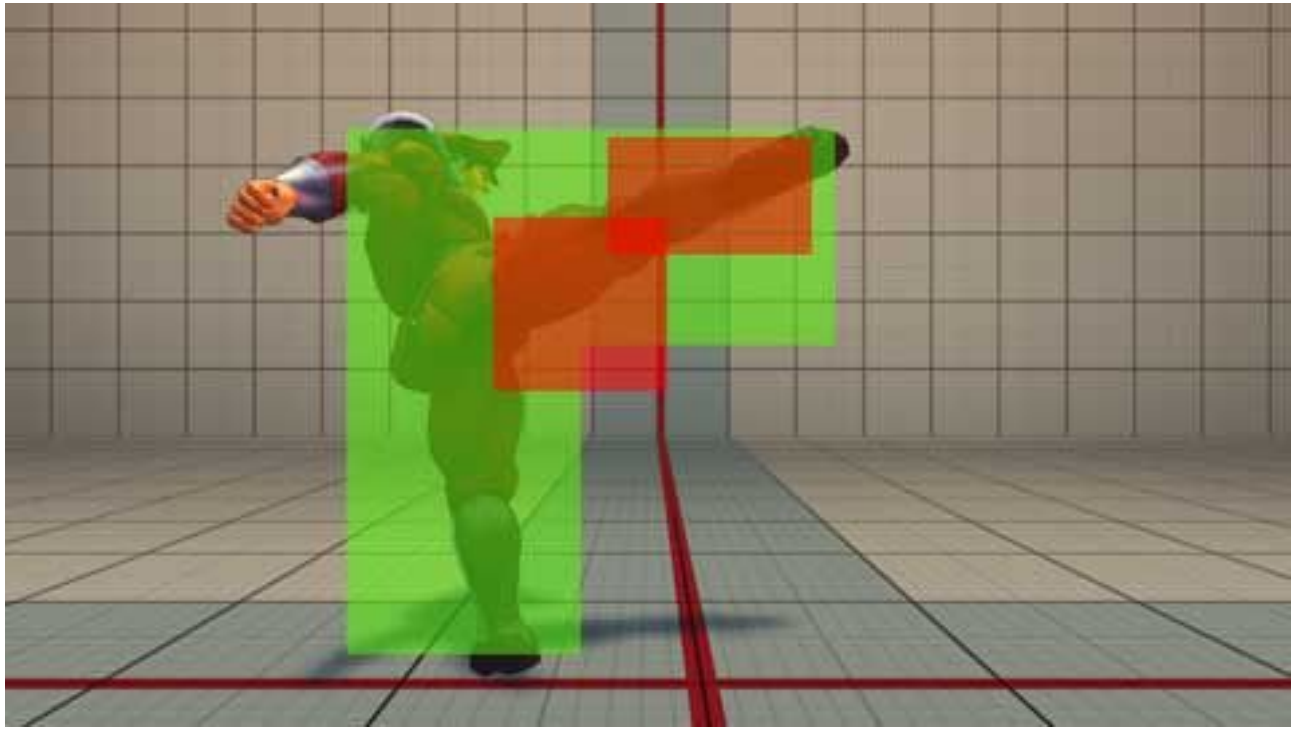


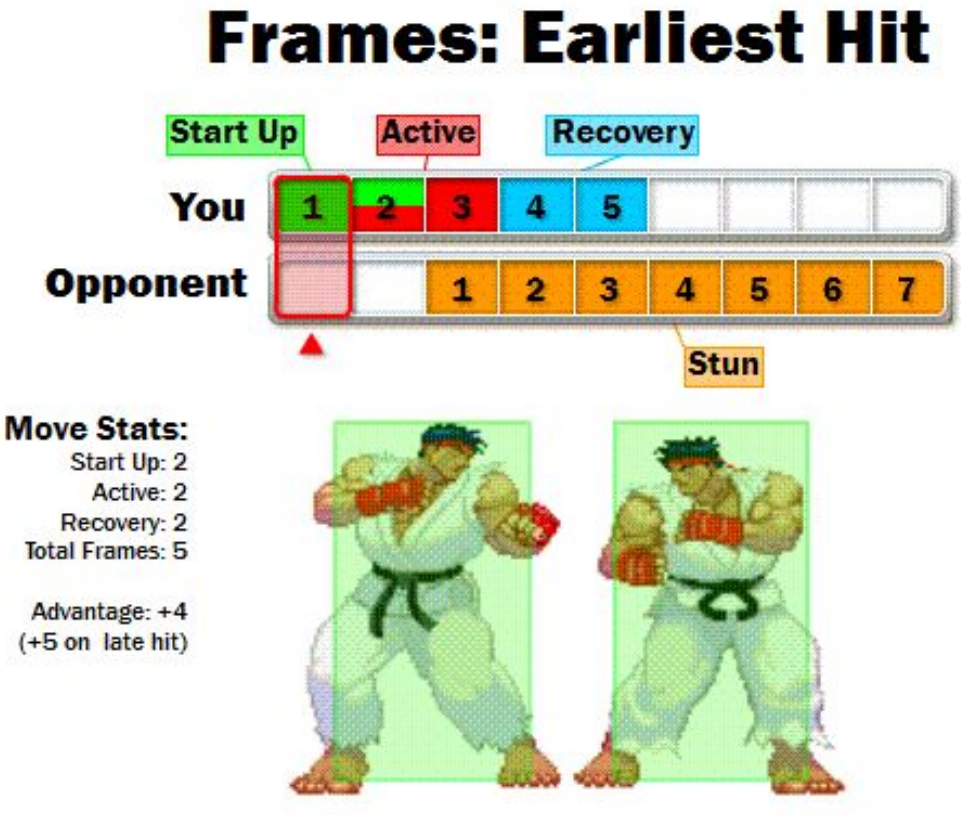
Figure 4-5. Airplane with single bounding box (left) and multiple bounding boxes (right)



Bounding Box Collision, hit boxes

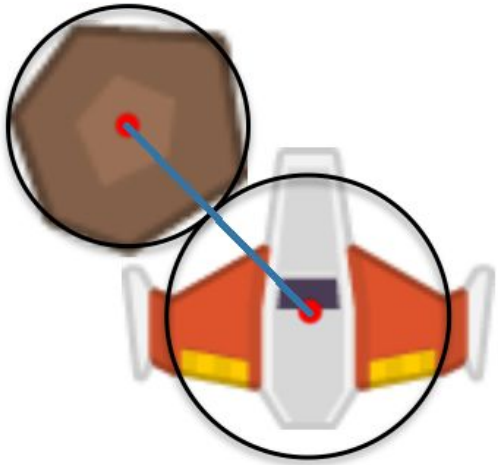


Bounding Box Collision, hit boxes y hurt boxes



Bounding Circle Collision

Similar a Bounding Box Collision, pero es más útil para objetos más redondeados, en vez de chequear los 4 puntos se trabaja con la posición del círculo y el radio que nos daría el dato de la distancia de la posición hasta los límites del círculo. Si la distancia es menor a la suma de los radios es que se produjo una colisión.



```
distance = sqrt((circle2.x - circle1.x)^2 +
                (circle2.y - circle1.y)^2);
if (distance < (circle1.radius + circle2.radius)){
    collision(circle2, circle1);
}
```

Bounding Circle Collision

Pero como calcular raiz cuadrada es más costoso que multiplicar por sí mismo y vamos a estar chequeando colisiones todo el tiempo podemos hacer una optimización llevando la raíz cuadrada al otro miembro como potencia

```
// distance = sqrt((circle2.x - circle1.x)^2 +
//                 (circle2.y - circle1.y)^2);
//
// if (distance < (circle1.radius + circle2.radius)){
//     collision(circle2, circle1);
// }

squareDistance = (circle2.x - circle1.x)^2 + (circle2.y - circle1.y)^2;
if (squareDistance < (circle1.radius + circle2.radius)^2){
    collision(circle2, circle1);
}
```

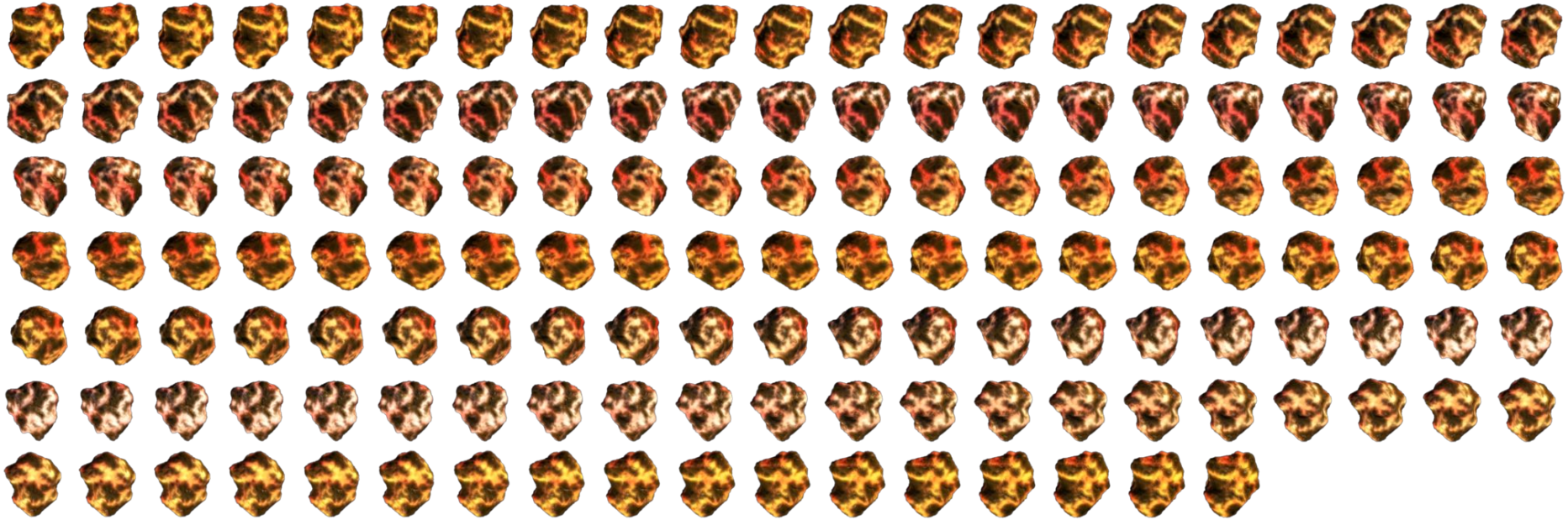
Con estos conceptos simples podemos crear cosas más complejas

Juego 2: Side Shooter

Demo

Manejo de sprites

Animación



Gracias!

Código de Pong, SpaceShooter e instaladores:

- https://github.com/lucianoinso