Introduction to Game Programming

OffTopic Session – Gravity and mouse

Nacho Cabanes
Nacho Iborra

IES San Vicente

Index of contents

1. Introduction

In this off-topic session about game programming with SDL, we are going to deal with two additional concepts:

- The first one is how to apply gravity to our video games, and also how to make our characters jump and stay over some platforms.
- The second one is how to use some additional peripherals in our video game, such as the mouse.

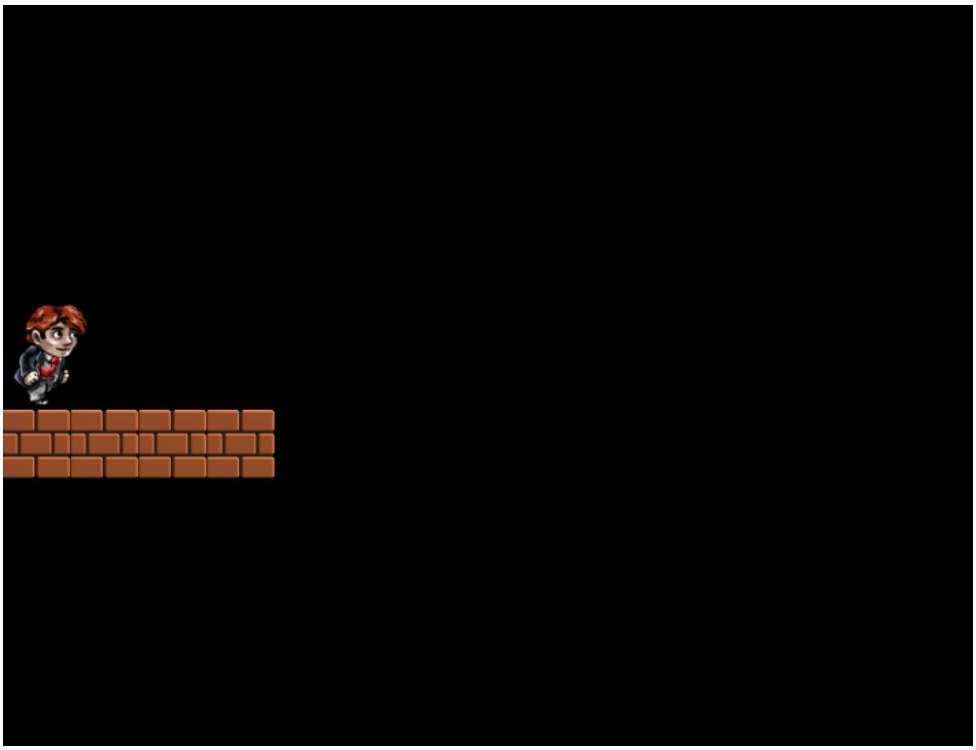
In order to practice these basic concepts of gravity and mouse usage, you will be provided with a basic Tao SDL with some basic game already implemented. You will be asked to add some improvements to this game.

2. Adding gravity to a video game

Adding gravity to a video game means basically that everything that goes up (or is already up) must go down again if nothing is holding it. In other words:

- If no floor is under an element, this element must go down
- If an element (character, enemy...) is jumping, then there is a force that lifts it up. When this force is weaker than gravity, then this element must go down again.

Let's start by the beginning: making an element go down if no floor is under it. Then, we will learn how to jump and go down. To apply all these concepts, we are going to work with a project called *TestJump*, that has a set of bricks and a character over them.



If you try to move the character to the left or to the right, you can see that he "flies" when no brick is under him. We are going to change this behaviour in next subsections, making the character fall down when no brick supports him, and letting him jump when he is over a brick or in the bottom border.

2.1. Making things fall down

If we want to make an element fall down, we should first create a method that makes this element go down (i.e. increase its Y coordinate) until it reaches a floor or platform. Assuming that we have an *Image* or *Sprite* class to represent every drawable object, we can add a *Fall* method like this one:

```
public void Fall()
{
    this.y++;
}
```

2.1.1. Detecting a floor or platform

When will we apply this method? When no floor is under the object. To detect this, we can add a method to the *Image* class to detect if *this* image is over a given image.

```
public bool IsOver(Image img)
{
    return (this.CollidesWith(img, this.GetImageWidth(), this.GetImageHeight(),
        img.GetImageWidth(), img.GetImageHeight()) &&
        img.GetY() >= this.GetY() + this.GetImageHeight() * 0.9);
}
```

What we do is detect if both images collide, and if so, check if *img* is right under *this* image (at least, under the 90% of *this* image)

Now, we can apply these methods in our main program. First of all, we are going to declare a boolean variable to determine if character is currently falling or not.

```
bool left, right, isFalling = false;
```

Then, in step #4 of game loop (collision detection), we detect if character is going to fall after each movement, and then, set the appropriate value for *isFalling* variable:

```
// 4. Collision detection
isFalling = true;
for (int i = 0; i < bricks.Length; i++)
{
    if (character.IsOver(bricks[i]))
    {
        character.MoveTo(character.GetX(), bricks[i].GetY() -
            CHARACTER_SPRITESHEET_HEIGHT);
        isFalling = false;
    }
}
if (isFalling && character.GetY() ==
    SCREEN_HEIGHT - CHARACTER_SPRITESHEET_HEIGHT)
{
    character.MoveTo(character.GetX(), SCREEN_HEIGHT -
        CHARACTER_SPRITESHEET_HEIGHT);
    isFalling = false;
}
```

The main character will be falling if no brick is under him (we check this with the *for* structure), or if he is not in the lower border of the screen (we check this with the last *if*). In both cases, we move the main character to be just over the brick or floor (it may be some pixels down, depending on when the collision is detected).

Finally, we go to step #2 of the game loop and we make the character fall if *isFalling* is true. Otherwise, we pay attention to any key input from the user:

```
// 2. Move character from keyboard input
```

```
...
if (isFalling)
    character.Fall();
else if (left || right)
{
    ...
}
```

If you try the game now, you can see how character falls when going to the right.

Exercise 1

Add this code to the project to make character fall when he goes to the right.

2.2. Jumping

When we jump, we need a floor to rely on. So we are going to use part of the previous code. If we are not falling, we are going to jump when we press the space bar. First of all, we need to define a variable to determine if main character is currently jumping. We also need a vertical speed to make the character go up when jumping, and a horizontal speed in case we are moving left or right when jumping.

```
bool left, right, isFalling = false, isJumping = false;
float verticalSpeed = 0.0f, horizontalSpeed = 0.0f;
```

We can also define some constants that might help us tune the appropriate vertical movement.

```
const float MOVEMENT_INCREMENT = 0.3f;
const float MAX_VERTICAL_SPEED = 2.0f;
const float VERTICAL_SPEED_DECREMENT = 0.015f;
```

Then, we go to step #2. If we are not falling, we check if user presses the space bar, and set *isJumping* variable to true. If this variable is already set to *true* previously, then we update current position and decrease vertical speed each time.

```
if (isFalling)
    character.Fall();
else if (isJumping)
{
    character.MoveTo(character.GetX() + horizontalSpeed,
                     character.GetY() + verticalSpeed);
    verticalSpeed += VERTICAL_SPEED_DECREMENT;
    if (verticalSpeed > MAX_VERTICAL_SPEED)
        verticalSpeed = MAX_VERTICAL_SPEED;
}
else if (key == Hardware.KEY_SPACE)
{
    isJumping = true;
    verticalSpeed = -1 * MAX_VERTICAL_SPEED;
```

```
    horizontalSpeed = left ? -1 * MOVEMENT_INCREMENT :
                      right ? MOVEMENT_INCREMENT : 0.0f;
    character.MoveTo(character.GetX() + horizontalSpeed,
                     character.GetY() + verticalSpeed);
}
else if (left || right)
{
    ...
```

Finally, we need to go to step #4 of the game loop (collision detection), and set *isJumping* variable to *false* every time we rely on a brick or the bottom floor.

```
isFalling = !isJumping;
for (int i = 0; i < bricks.Length; i++)
{
    if (character.IsOver(bricks[i]))
    {
        ...
        isJumping = false;
    }
}
if (character.GetY() >= SCREEN_HEIGHT - CHARACTER_SPRITESHEET_HEIGHT)
{
    ...
    isJumping = false;
}
```

Exercise 2

Add this code to our *TestJump* project to make the character jump if he is over the bricks or in the bottom floor.

3. Using the mouse

Tao SDL is a really basic library, but it lets us use some additional peripherals other than the keyboard, such as the mouse, or even a gamepad. We are going to focus on how to use the mouse in our code.

First of all, we need to add some code to check if mouse has been moved or clicked. Everything we do with the keyboard, mouse or even gamepad is an event, and we can (should) treat all these events in a single method. So let's remove our *KeyPressed* method from *Hardware* class and add this new method instead:

```
public void GetEvents(out int key, out int mouseX, out int mouseY)
{
    key = mouseX = mouseY = -1;

    Sdl.SDL_PumpEvents();
    Sdl.SDL_Event evt;
    if (Sdl.SDL_PollEvent(out evt) == 1)
    {
        if (evt.type == Sdl.SDL_KEYDOWN)
        {
            key = evt.key.keysym.sym;
        }
        else if (evt.type == Sdl.SDL_MOUSEMOTION)
        {
            mouseX = evt.motion.x;
            mouseY = evt.motion.y;
        }
    }
}
```

Notice that we define three *out* parameters: one to store the (possible) key pressed, and two more parameters to store the (possible) mouse coordinates whenever it is moved. See how we detect the mouse motiong with the corresponding event type (*Sdl.SDL_MOUSEMOTION*).

We need to try this out, so let's add two more variables to our *Main* method to store mouse X and Y coordinates whenever the mouse is moved:

```
int key = 0, mouseX = 0, mouseY = 0, spriteCount = 0, currentSprite = 1;
```

Then, we replace the old hardware.KeyPressed call with this new call at the beginning of step #2 of the game loop:

```
// 2. Move character from keyboard input


~~key = hardware.KeyPressed();~~
hardware.GetEvents(out key, out mouseX, out mouseY);
```

```
...
```

Finally, at the end of step #2, we are going to move the character to current mouse coordinates whenever mouse is moved:

```
if (mouseX >= 0 && mouseY >= 0)
{
    character.MoveTo(mouseX, mouseY);
}


// 3. Move other elements (automatically)
```

Exercise 3

Add these changes to the video game to see how character is re-located every time we move the mouse. Notice how it falls down as we stop the mouse motion, if no brick is under him.

4. To think a little...

In order to finish with this session, add these improvements to our *TestJump* project:

- Change the character sprite while jumping. Use the first sprite of the sprite sheet. It will always look right, but this is just a test of how to change the sprite sheet while jumping.
- Add more bricks in the scene whenever the user clicks a mouse button. Previously, you may need to remove/comment these lines at the end of step #2 of the game loop:

```
if (mouseX >= 0 && mouseY >= 0)
{
    character.MoveTo(mouseX, mouseY);
}
```

  Then, add/modify the appropriate code to add a new brick to a list of bricks every time we click a button.
- Add some collision detection between the character and the bricks so that character can't go through bricks nor jump when it collides with a brick (unless he is over the brick).