CS 116 – Action Script Character Behavior

Elie Abi Chahine

Character Behavior

- In this chapter, we are going to add extra behaviors on our main character(ship) like translation using velocity, rotation, shooting bullets.
- By adding those behaviors, I want you to learn how to move objects on the screen, rotate and shoot bullets according to the direction he is facing.
- Also I'll cover, not in depth, how to add text on the screen and update it at runtime.

Movement

- Let's start with movement.
- We want to move our object. Every object has an X and Y coordinate on the screen which can be altered by changing the values of the x and y variables inside the MovieClip class.
- So start with creating the "Ship" class with an "Initialize" function that accepts two integers representing the object's position on the screen

```
function Initialize(iPosX_:int , iPosY_:int)
{
        x = iPosX_;
        y = iPosY_;
}
```

Note: The Y axis in Flash is downwards. So y = 0 would be the window's top and as y increases the object will go down.

Movement

- Add an instance from the Ship class to the stage.

```
var mcShip:Ship = new Ship();
mcShip.Initialize(100,200);
stage.addChild(mcShip);
```

- Next, let’s add the variables and functions needed to move our Ship. This time we will implement a smoother movement (though it won’t be the smoothest possible).

Movement

- Variables to add to the ship class:

– **nVelocityX**	**which will tell the character how much to move on the X axis**
– **nVelocityY**	**which will tell the character how much to move on the Y axis**
– **nMaxVelocity**	**which will set a maximum speed that the character can have**
– **nSpeed**	**which will set the increment and decrement step for the velocity every time the user presses a key.**

NB: Don't forget to adjust the Initialize function accordingly.

Movement

```
var nMaxVelocity:Number;
var nSpeed:Number;
var nVelocityX:Number;
var nVelocityY:Number;

function Initialize(iPosX_:int , iPosY_:int , nMaxVelocity_:Number , nSpeed_:Number)
{
        x = iPosX_;
        y = iPosY_;
        nMaxVelocity = nMaxVelocity_;
        nSpeed = nSpeed_;
        nVelocityX = 0;
        nVelocityY = 0;
}
```

- **nVelocityX and nVelocityY are set to 0 since we don't want the ship to start moving immediately (when we first run the game) without us pressing anything.**
- **Don't forget to update in the Layer's code the way you are calling the Initialize function.**

Movement

- Functions to add to the ship class:
  - Two function are going to be added to our class
    - The "Input" function that will be checking which key the user pressed and will update the ship's variables accordingly. The Input function will work according to a KeyboardEvent. It will also needs to check if we reached the max velocity value so that we limit our character's speed.
    - The "Update" that will run every frame and update the ship's position according to the current velocity values.

Movement

```
function Input(ke_:KeyboardEvent)
{
    if(ke_.keyCode == Keyboard.UP  &&  nVelocityY > -nMaxVelocity)
    {
            nVelocityY -= nSpeed;
    }

    if(ke_.keyCode == Keyboard.DOWN  &&  nVelocityY < nMaxVelocity)
    {
            nVelocityY += nSpeed;
    }

    if(ke_.keyCode == Keyboard.LEFT  &&  nVelocityX > -nMaxVelocity)
    {
            nVelocityX -= nSpeed;
    }

    if(ke_.keyCode == Keyboard.RIGHT  &&  nVelocityX < nMaxVelocity)
    {
            nVelocityX += nSpeed;
    }
}
```

Movement

!!! Don’t forget to add a listener for the stage that is waiting for a key to be pressed !!!

```
stage.addEventListener(KeyboardEvent.KEY_DOWN , UserInput);

function UserInput(ke_:KeyboardEvent)
{
    mcShip.Input(ke_);
}
```

Movement

```
function Update(e_:Event)
{
        x += nVelocityX;
        y += nVelocityY;
}
```

- **As you can see, the "x" and "y" variables are being updated every frame. It is up to the nVelocityX and nVelocityY values to say if the ship will move or not.**

NB: Don't forget to add a listener in the Initialize function

```
addEventListener(Event.ENTER_FRAME , Update);
```

Rotation

- Now let's see how we can rotate our object while moving.
- In order to update the Object's rotation you need to change the "rotation" variable in it.
- In code:

```
rotation += 2; /* will rotate the ship 2 degrees clockwise */
rotation -= 2; /* will rotate the ship 2 degrees counter clockwise */
```

- **Let's add that behavior in our code. We rotate clockwise when we press the "Left" key and counter clockwise when we press the "Right" key.**

Rotation

```
if(ke_.keyCode == Keyboard.LEFT)
{
        /* Rotate the Ship counter clockwise */
        rotation -= 7;
}

if(ke_.keyCode == Keyboard.RIGHT)
{
        /* Move the Ship clockwise */
        rotation += 7;
}
```

***Note:** We can always add a variable for the rotation step (like we did with speed)*

- **Run the code**

Movement & Rotation

- **Now that we have our ship rotate, movement became weird.**
- **We need to make some changes to our class so that the ship moves according to the direction it is moving.**
- **Let's see… what to remove and what to add to our main character's class?**

Movement & Rotation

Our current variables:

```
var nMaxVelocity:Number;
var nSpeed:Number;
var nVelocityX:Number;
var nVelocityY:Number;
```

- **Remove:**
  - We don’t need 2 variables for Velocity(X,Y) anymore… since we are going to move according to the direction the ship currently has. So one velocity variable is enough.
- **Add:**
  - 2 variables for the direction which will change every time you rotate.
  - 1 variable for the velocity (nVelocity)

Movement & Rotation

- **New Variables:**

```
var nMaxVelocity:Number;
var nSpeed:Number;
var nVelocity:Number;
var nDirectionX:Number;
var nDirectionY:Number;
```

- **Run the code… Yes you have a lot of errors… Try to update the functions inside your class in order to remove those errors.**

Movement & Rotation

A little bit of Math ☹ ! How do we get the direction from the angle?

VectorX = cos(Angle)

VectorY = sin(Angle)

Eg:

Angle = 90 (PI/2 in radian)

VectorX = 0
VectorY = 1

NB: we need the angle to be in radian:

AngleRadian = AngleDegree * PI/180

Movement & Rotation

Things you should change:

- Updated the “Initialize” function. We need to send the initial direction values.

```
function Initialize(nPosX_:Number, nPosY_:Number, nSpeed_:Number)
{
    x = nPosX_;
    y = nPosY_;
    nSpeed = nSpeed_;
    nVelocity = 0;
    nDirectionX = Math.cos(rotation * PI_OVER_180);
    nDirectionY = Math.sin(rotation * PI_OVER_180);

    addEventListener(Event.ENTER_FRAME, Update);
}
```

Movement & Rotation

Things you should change:

- Change the "Update" function:
  - we only have one Velocity variable now.
  - change the way we update our position

```
x  += nDirectionX * nVelocity;
y  += nDirectionY * nVelocity;
```

- Update the "Input" function inside the ship class since we don't have nVelocityX and nVelocityY anymore.
- Update the code in the layer since we made changes in the "Initialize" function (now it expects different parameters).

Movement & Rotation

Compute our current direction in the ship's Update function :

```
function Update(e_:Event)
{
    nDirectionX = Math.cos(rotation * PI_OVER_180);
    nDirectionY = Math.sin(rotation * PI_OVER_180);

    x += nDirectionX * nVelocity;
    y += nDirectionY * nVelocity;
}
```

Movement & Rotation

- When running the code, you might have the following behavior:
  - Pressing "Up" makes the ship go backwards
  - Pressing "Down" makes the ship go forward.
- Fix it by changing the signs when pressing UP and Down

Movement & Rotation

- How about we add DRAG to the ship so that it stops when we don't press UP or DOWN…
- Add a constant value for the drag value
  - **const DRAG_FACTOR:Number = 0.99;**
- In the Update function add the following command at the end:
  - **nVelocity *= DRAG_FACTOR;**

Shooting Bullets

- Now that we have our ship moving and updating it's direction every frame, we can start thinking about shooting.
- First thing to do is to create a shape for the bullet. For the sake of simplicity, I advice you to just make the bullet as a circle.
- Of course, you are going to convert it to a symbol and export it to ActionScript

NB: Don't forget to delete the bullet that you created on the stage, because as usual, we will create the bullets dynamically (Hopefully you know what that means by now)

Shooting Bullets

So let's see… What do we need to add to our bullet class.

- Variables:
  - nDirectionX
  - nDirectionY
  - nSpeed

- Create an Initialize function for the Bullet class so that we can send different positions, directions and speed for each bullet.

```
function Initialize(nPosX_:int, nPosY_:int, nDirectionX_:Number, nDirectionY_:Number, nSpeed_:Number)
{
      x = nPosX_;
      y = nPosY_;
      nDirectionX = nDirectionX_;
      nDirectionY = nDirectionY_;
      nSpeed = nSpeed_;
}
```

Shooting Bullets

- Now let us add a function that will dynamically create bullets. Where do you think we should add this function????
- We should add it inside the main ship class since it is the ship who is shooting bullets…

```
function Shoot()
{
    var b:Bullet = new Bullet();
    b.Initialize(x, y, 50, nDirectionX, nDirectionY);
    stage.addChild(b);
}
```

Shooting Bullets

Let's add the bullet behavior:

- add an event listener that will happen every frame
- Add an Update function (call it whatever you want) so that the bullet's position will be changed every frame according to the direction variables that we assigned to it.

```
function Update(e_:Event)
{
	x += nDirectionX * nSpeed;
	y += nDirectionY * nSpeed;
}
```

Shooting Bullets

Let's add a key to shoot

- We can add a simple extra check in the ship's Input function in order to shoot a bullet

```
if(ke_.keyCode == Keyboard.SPACE)
{
    Shoot();
}
```

- **Run the code**

Text

- Now that we have movement and shooting up and running, we are going to add a text that will display on the screen the ship’s x and y coordinates on the screen.

- I’m not going to cover a lot on the TextField class, you can check out the help and learn more about it.

Text

- In order to create a text variable, you should use the “TextField” type, which is found under the **flash.text.TextField** library.

- Let’s create a text inside the ship class and add it to the stage.
- Start with adding a TextField variable in the ship class:

```
var tPositionText:TextField;
```

- In the ship’s “Initialize” function add the following:

```
tPositionText = new TextField();
tPositionText.scaleX = 3;
tPositionText.scaleY = 3;
tPositionText.x = 50;
tPositionText.textColor = 0XFF0000;
tPositionText.text = String(x) + " " + String(y);
```

Text

- At this point we have the text set but not added to the stage.
- In order to add something to the stage from inside the Ship's initialize function, you need first to have the Ship instance added to the stage.

```
var mcShip:Ship = new Ship();
stage.addChild(mcShip);
mcShip.Initialize(100,200,5,1,0,-1);
```

- Now we can add the text to the stage inside the "Initialize" function

```
stage.addChild(tPositionText);
```

- Let's run the code. You should have a red text on the screen showing 2 values.

Text

- Now, if we move the ship the text won’t get updated.
- To solve this problem we need to update the text on every frame
- The ship’s “Update” function is a good place to update our text value

```
tPositionText.text = String(x) + " " + String(y);
```

- Now if you run the code, everything should be working.

The End ☺