Easing and Splines for UI Animations

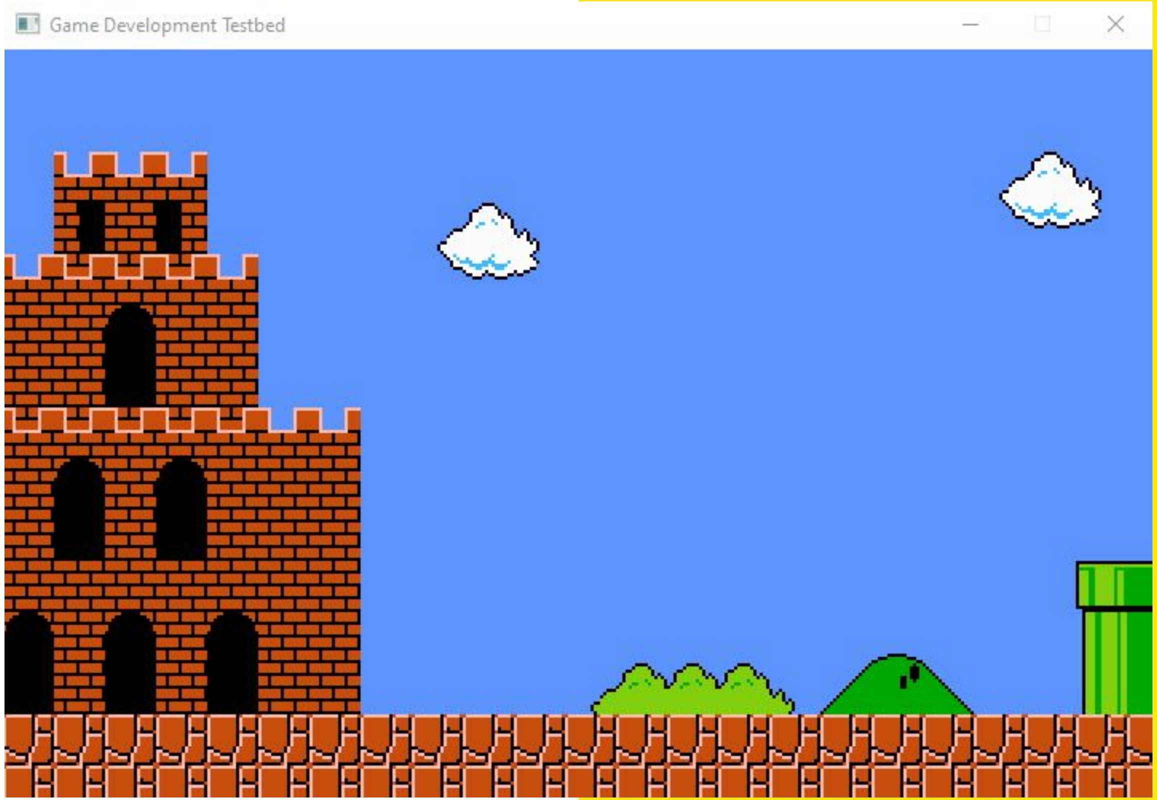
By Aitor Álvarez.

Easing and Splines

- When creating animations for the UI in a video game, we can follow a series of methods in order to create a decent and good looking environment.

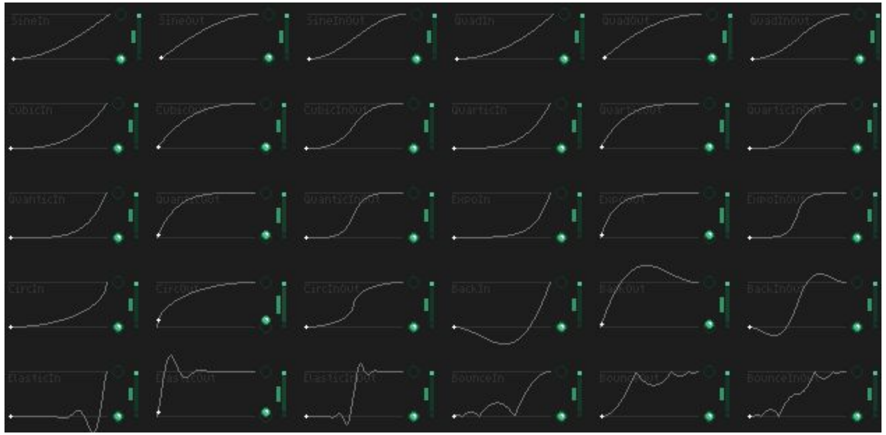
- We will be using a method called Easing used in animation. And in the implementation of this method we will be using a series of mathematical functions that are called splines.

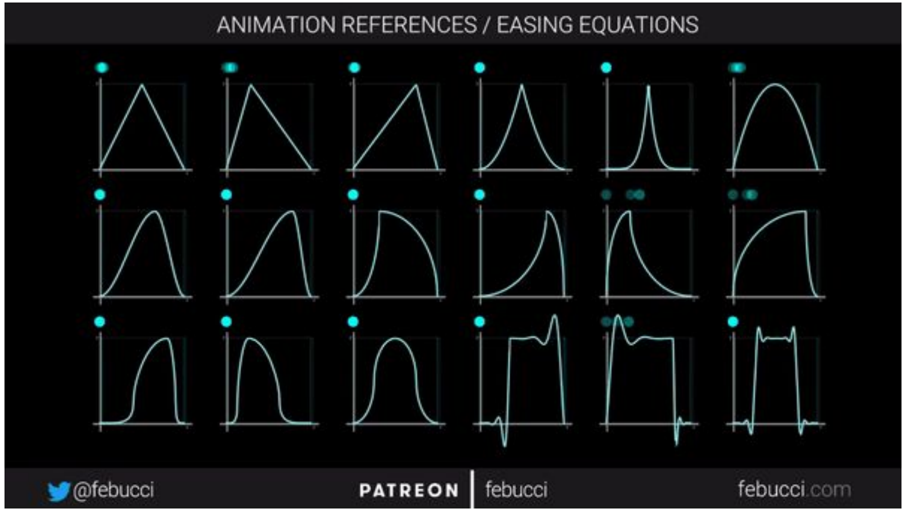
Easing and Splines



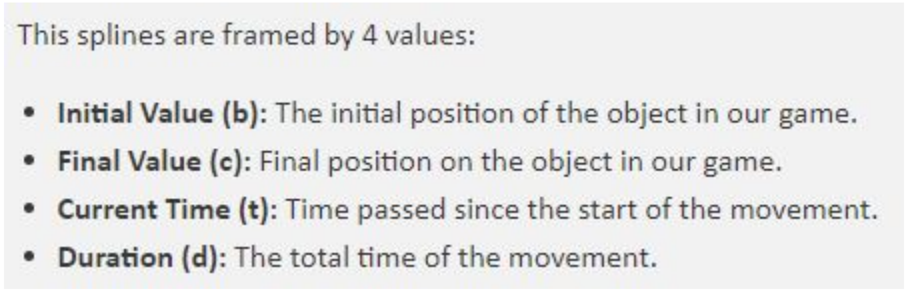
Ease
Spline

Easing and Splines

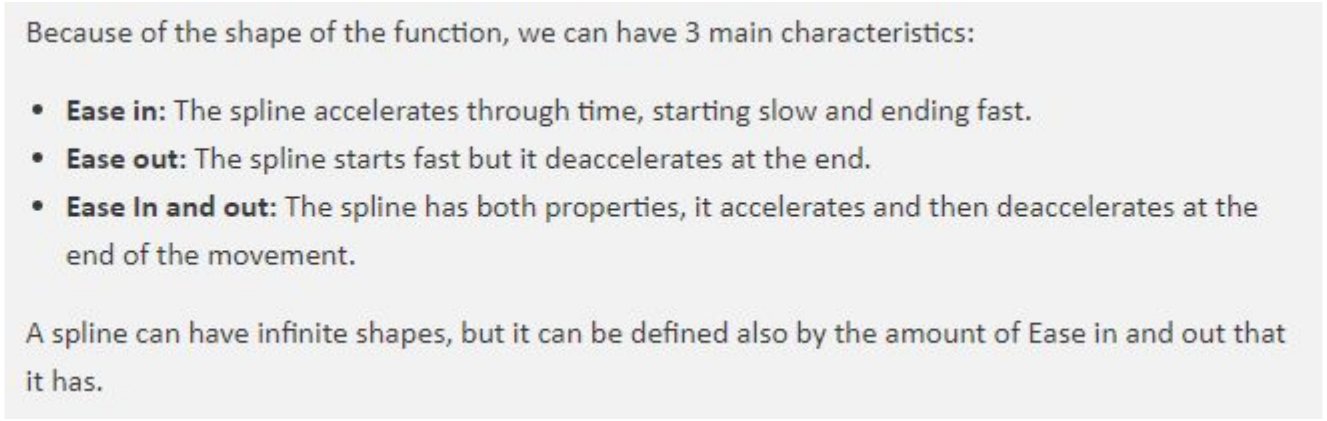




Spline values


This splines are framed by 4 values:

- **Initial Value (b)**: The initial position of the object in our game.
- **Final Value (c)**: Final position on the object in our game.
- **Current Time (t)**: Time passed since the start of the movement.
- **Duration (d)**: The total time of the movement.


Because of the shape of the function, we can have 3 main characteristics:

- **Ease in:** The spline accelerates through time, starting slow and ending fast.
- **Ease out**: The spline starts fast but it deaccelerates at the end.
- **Ease In and out:** The spline has both properties, it accelerates and then deaccelerates at the end of the movement.

A spline can have infinite shapes, but it can be defined also by the amount of Ease in and out that it has.

Spline formulas

```
simple linear tweening - no easing, no acceleration
```

```
Math.linearTween = function (t, b, c, d) {
        return c*t/d + b;
};
```

```
cubic easing in - accelerating from zero velocity
```

```
Math.easeInCubic = function (t, b, c, d) {
        t /= d;
        return c*t*t*t + b;
};
```

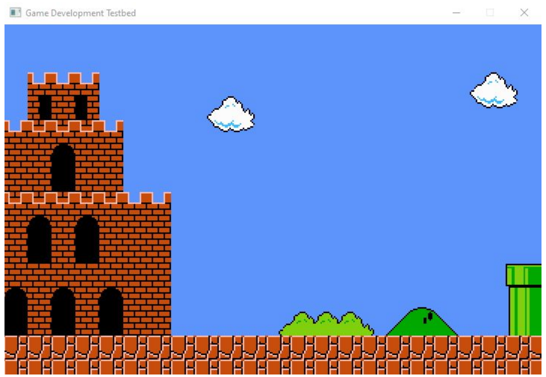
```
exponential easing in/out - accelerating until halfway, then decelerating
```

```
Math.easeInOutExpo = function (t, b, c, d) {
        t /= d/2;
        if (t < 1) return c/2 * Math.pow( 2, 10 * (t - 1) ) + b;
        t--;
        return c/2 * ( -Math.pow( 2, -10 * t) + 2 ) + b;
};
```

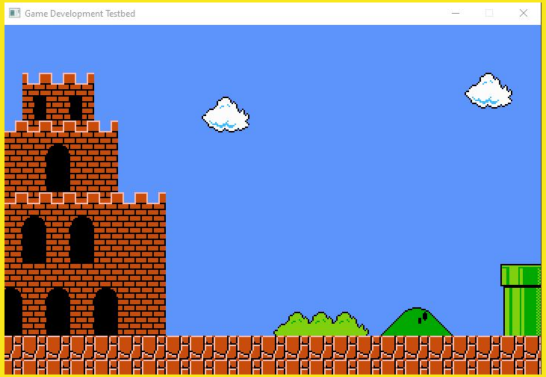
```
quadratic easing in - accelerating from zero velocity
```

```
Math.easeInQuad = function (t, b, c, d) {
        t /= d;
        return c*t*t + b;
};
```

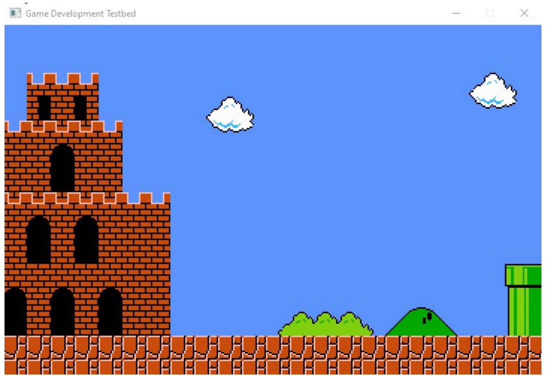
Examples



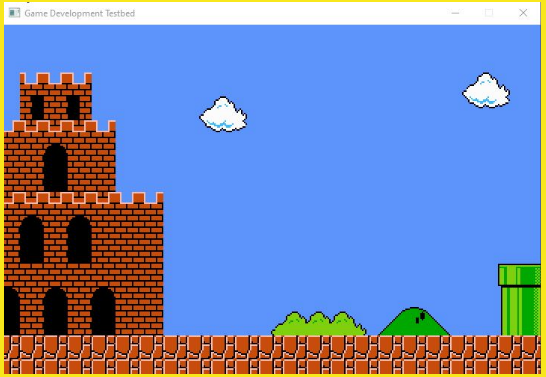
Ease Spline



Cubic Spline

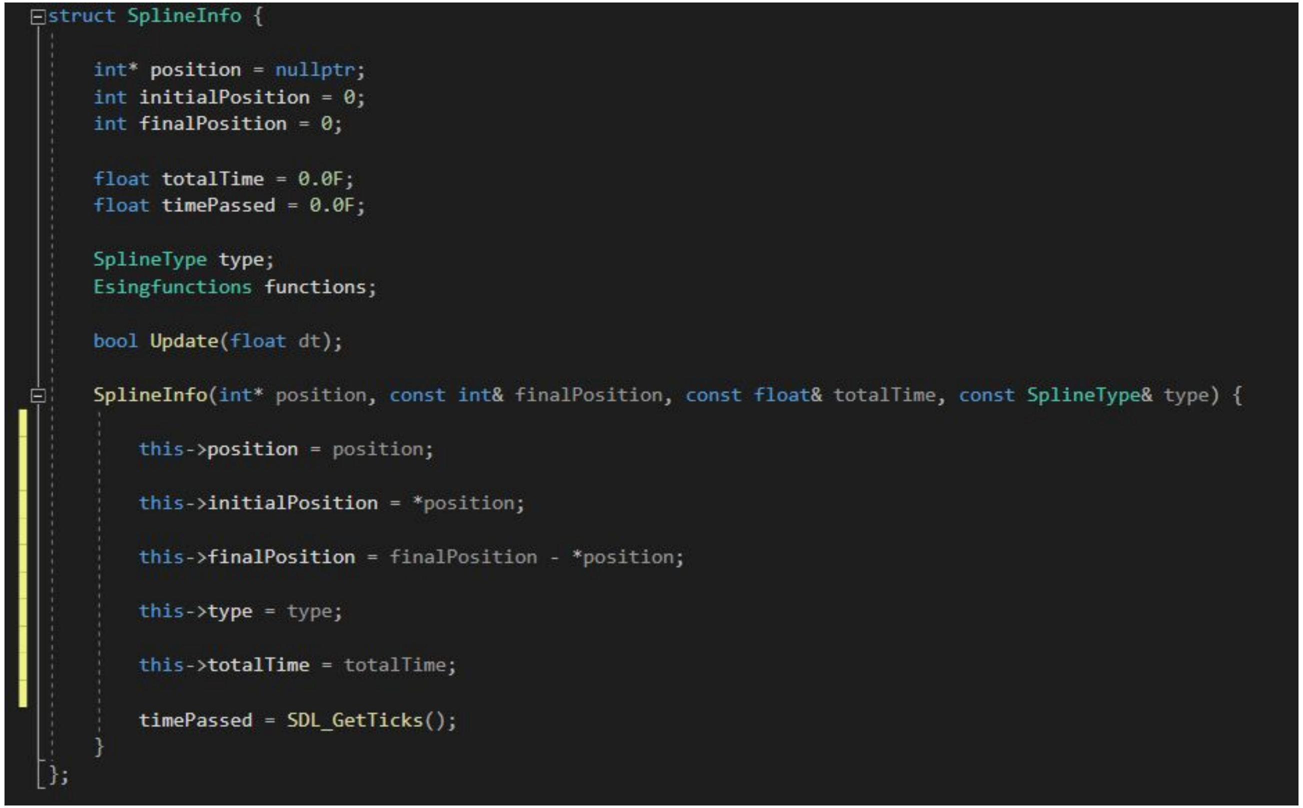


Expo Spline



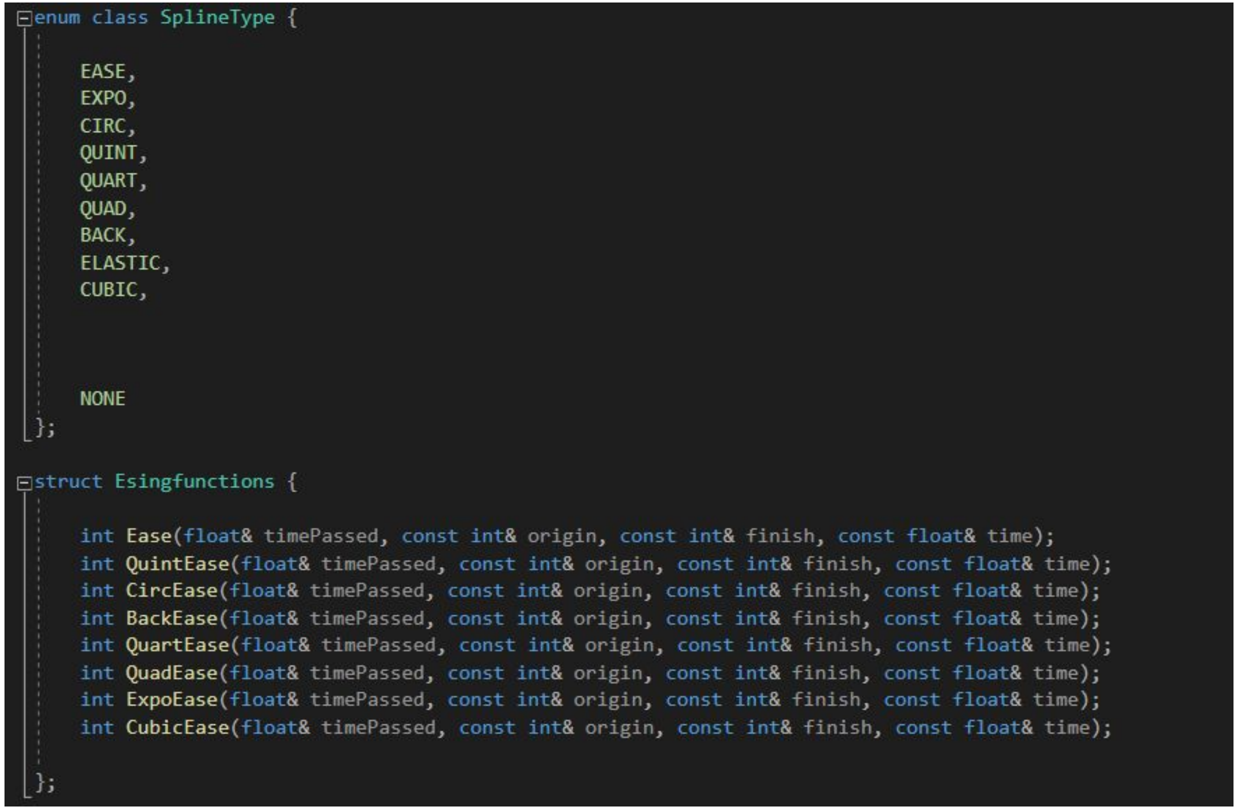
Quad Spline

Code



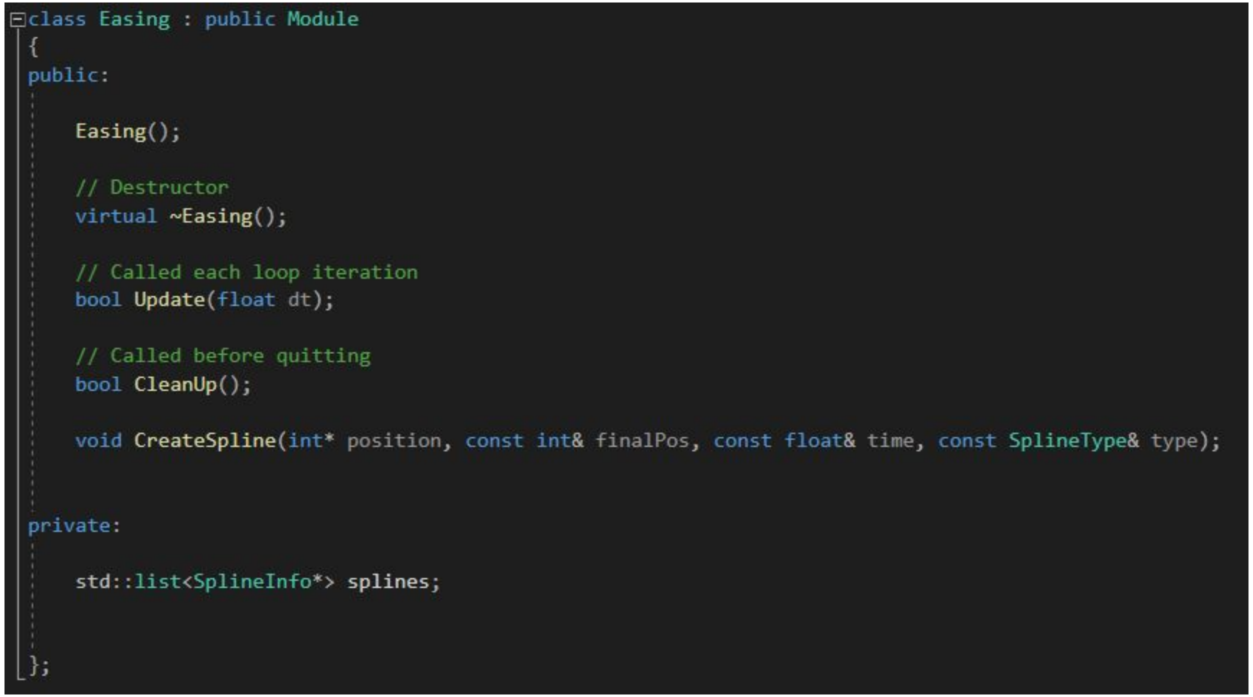
```
struct SplineInfo {

    int* position = nullptr;
    int initialPosition = 0;
    int finalPosition = 0;

    float totalTime = 0.0F;
    float timePassed = 0.0F;

    SplineType type;
    Esingfunctions functions;

    bool Update(float dt);

    SplineInfo(int* position, const int& finalPosition, const float& totalTime, const SplineType& type) {

        this->position = position;

        this->initialPosition = *position;

        this->finalPosition = finalPosition - *position;

        this->type = type;

        this->totalTime = totalTime;

        timePassed = SDL_GetTicks();
    }
};
```

Code



```
enum class SplineType {

    EASE,
    EXPO,
    CIRC,
    QUINT,
    QUART,
    QUAD,
    BACK,
    ELASTIC,
    CUBIC,



    NONE
};

struct Esingfunctions {

    int Ease(float& timePassed, const int& origin, const int& finish, const float& time);
    int QuintEase(float& timePassed, const int& origin, const int& finish, const float& time);
    int CircEase(float& timePassed, const int& origin, const int& finish, const float& time);
    int BackEase(float& timePassed, const int& origin, const int& finish, const float& time);
    int QuartEase(float& timePassed, const int& origin, const int& finish, const float& time);
    int QuadEase(float& timePassed, const int& origin, const int& finish, const float& time);
    int ExpoEase(float& timePassed, const int& origin, const int& finish, const float& time);
    int CubicEase(float& timePassed, const int& origin, const int& finish, const float& time);

};
```

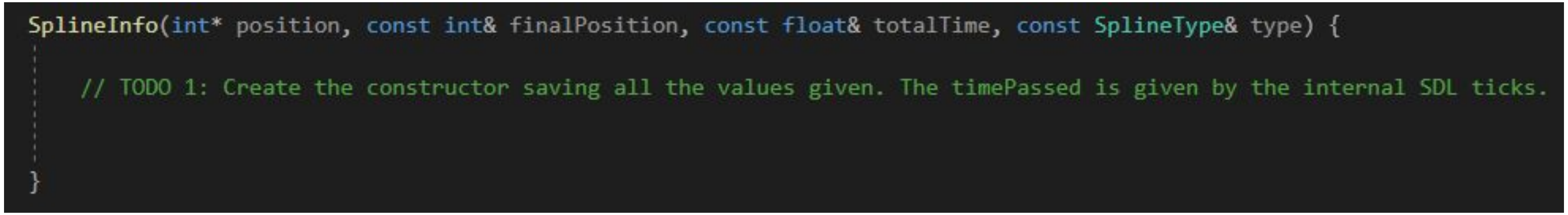
Code



```
class Easing : public Module
{
public:

    Easing();

    // Destructor
    virtual ~Easing();

    // Called each loop iteration
    bool Update(float dt);

    // Called before quitting
    bool CleanUp();

    void CreateSpline(int* position, const int& finalPos, const float& time, const SplineType& type);


private:

    std::list<SplineInfo*> splines;


};
```

TODO 1

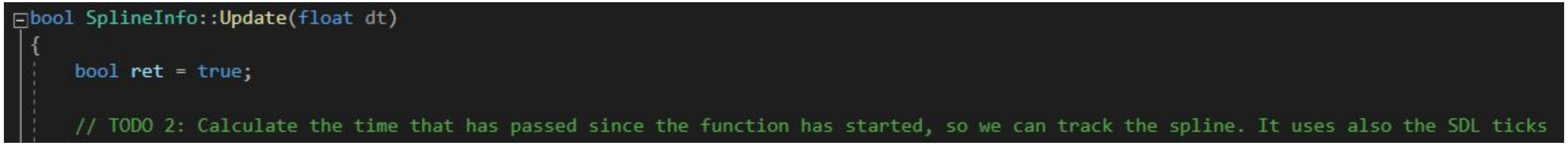
Easing.h



```
SplineInfo(int* position, const int& finalPosition, const float& totalTime, const SplineType& type) {

    // TODO 1: Create the constructor saving all the values given. The timePassed is given by the internal SDL ticks.


}
```

TODO 2

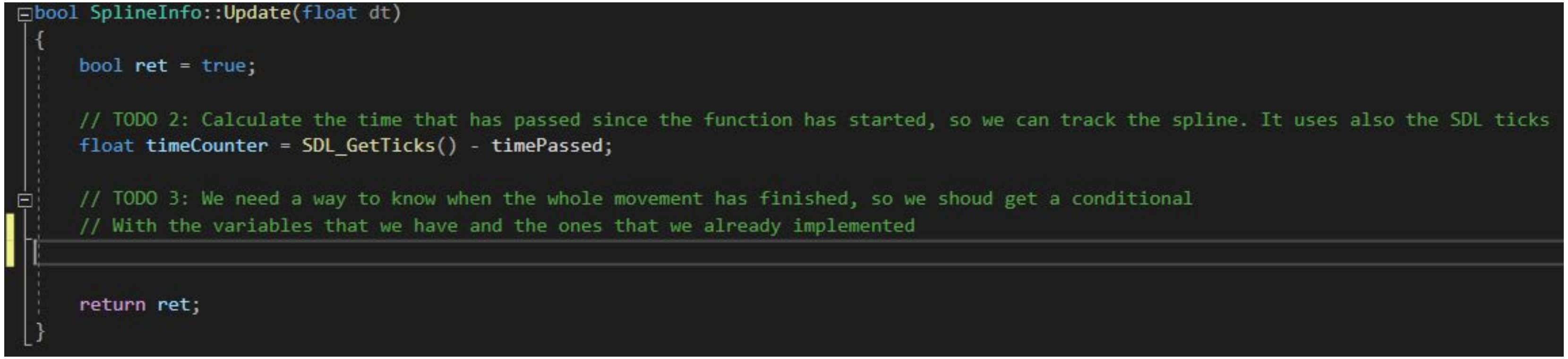
Easing.cpp



```
bool SplineInfo::Update(float dt)
{
    bool ret = true;

    // TODO 2: Calculate the time that has passed since the function has started, so we can track the spline. It uses also the SDL ticks
```

TODO 3

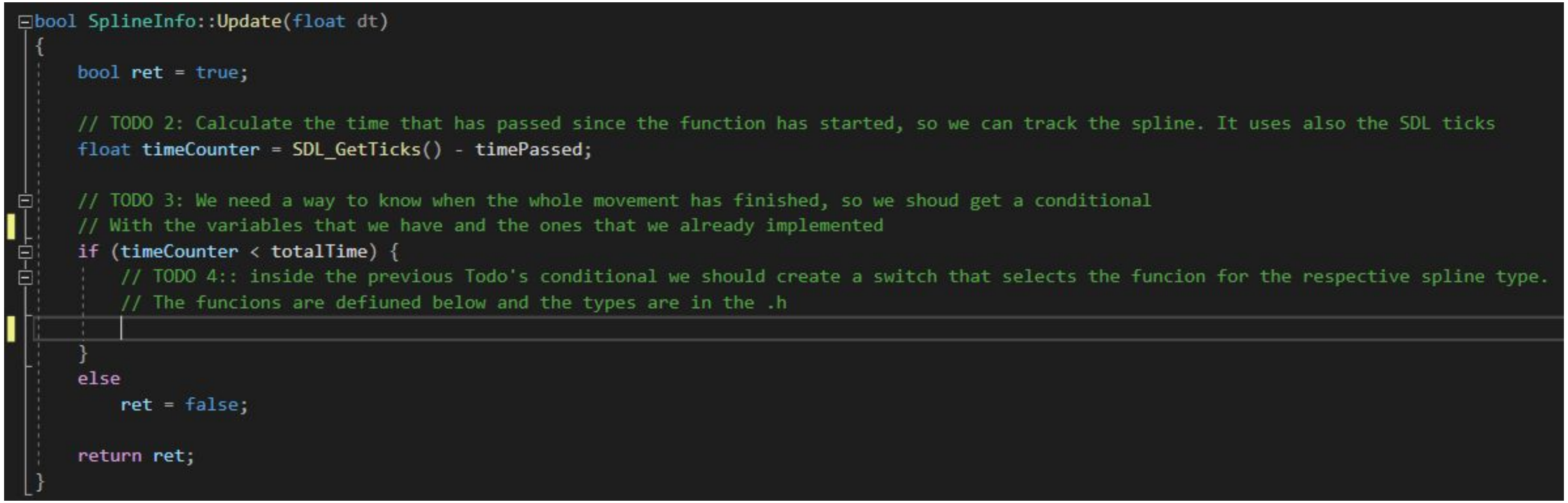
Easing.cpp



```
bool SplineInfo::Update(float dt)
{
    bool ret = true;

    // TODO 2: Calculate the time that has passed since the function has started, so we can track the spline. It uses also the SDL ticks
    float timeCounter = SDL_GetTicks() - timePassed;

    // TODO 3: We need a way to know when the whole movement has finished, so we shoud get a conditional
    // With the variables that we have and the ones that we already implemented


    return ret;
}
```

TODO 4

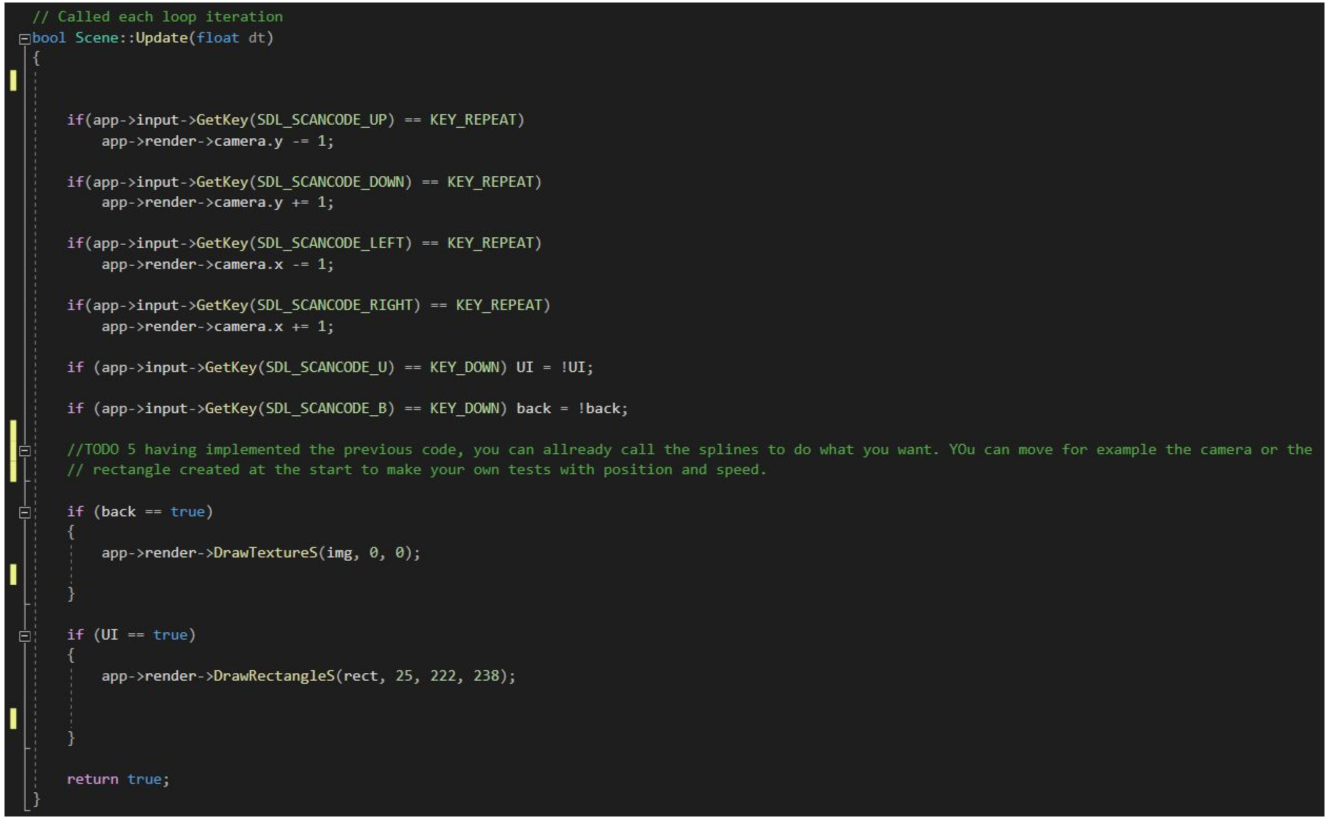
Easing.cpp



```cpp
bool SplineInfo::Update(float dt)
{
    bool ret = true;

    // TODO 2: Calculate the time that has passed since the function has started, so we can track the spline. It uses also the SDL ticks
    float timeCounter = SDL_GetTicks() - timePassed;

    // TODO 3: We need a way to know when the whole movement has finished, so we shoud get a conditional
    // With the variables that we have and the ones that we already implemented
    if (timeCounter < totalTime) {
        // TODO 4:: inside the previous Todo's conditional we should create a switch that selects the funcion for the respective spline type.
        // The funcions are defiuned below and the types are in the .h
        
    }
    else
        ret = false;

    return ret;
}
```

TODO 5

Scene.cpp



```
// Called each loop iteration
bool Scene::Update(float dt)
{


    if(app->input->GetKey(SDL_SCANCODE_UP) == KEY_REPEAT)
        app->render->camera.y -= 1;

    if(app->input->GetKey(SDL_SCANCODE_DOWN) == KEY_REPEAT)
        app->render->camera.y += 1;

    if(app->input->GetKey(SDL_SCANCODE_LEFT) == KEY_REPEAT)
        app->render->camera.x -= 1;

    if(app->input->GetKey(SDL_SCANCODE_RIGHT) == KEY_REPEAT)
        app->render->camera.x += 1;

    if (app->input->GetKey(SDL_SCANCODE_U) == KEY_DOWN) UI = !UI;

    if (app->input->GetKey(SDL_SCANCODE_B) == KEY_DOWN) back = !back;

    //TODO 5 having implemented the previous code, you can allready call the splines to do what you want. YOu can move for example the camera or the
    // rectangle created at the start to make your own tests with position and speed.

    if (back == true)
    {
        app->render->DrawTextureS(img, 0, 0);

    }

    if (UI == true)
    {
        app->render->DrawRectangleS(rect, 25, 222, 238);


    }

    return true;
}
```

Conclusions.

Thanks for listening!