Multimedia-Programmierung
Übung 5

Ludwig-Maximilians-Universität München

Sommersemester 2017

Today

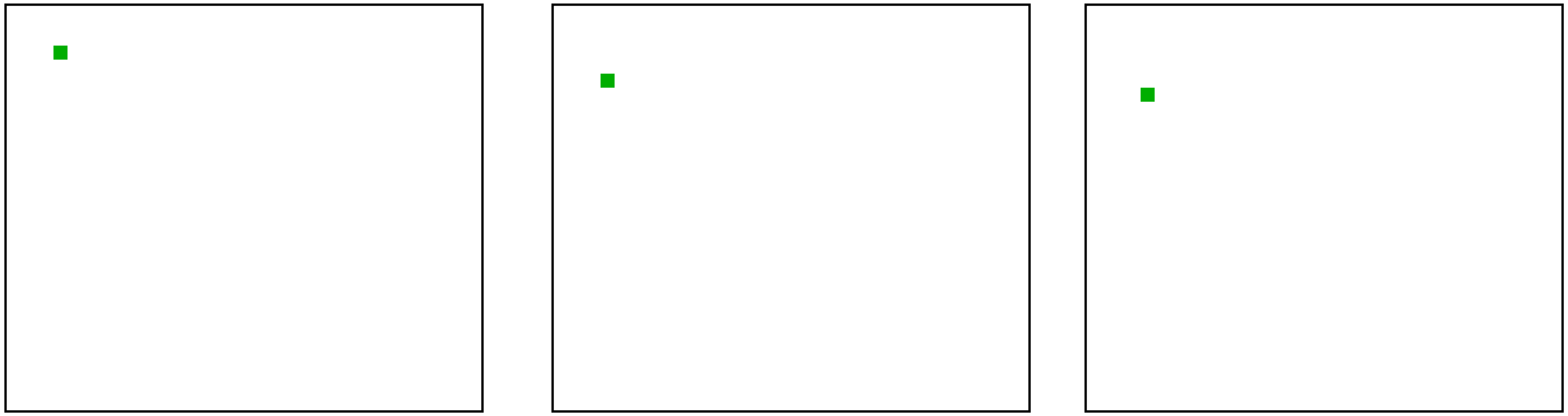
- Animations
- Illustrated with pygame + COCOS2DX

Literature:

W. McGugan, Beginning Game Development with Python and Pygame, Apress 2007
ENGELBERT, Roger. Cocos2d-x by Example: Beginner's Guide. Packt Publishing Ltd, 2015.

Objects on the Screen don’t actually move

- Basically, only the colours of pixels are changed
- Everytime something changes, the whole screen is repainted
- Framerate defines the appearance of the animation (the higher, the better)
- Possible framerate depends on the hardware (e.g. hertz of the monitor)



Moving an object in a straight line

```
import pygame
from pygame.locals import *
from sys import exit

player_image = 'head.jpg'
pygame.init()

screen = pygame.display.set_mode((640, 280), 0, 32)
pygame.display.set_caption("Animate X!")
mouse_cursor = pygame.image.load(player_image).convert_alpha()

x = 0 - mouse_cursor.get_width()
y = 10

while True:                    ←  event loop
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((255,255,255))
    if x > screen.get_width():         ←  if the object left the screen, reset x
            x = 0 - mouse_cursor.get_width()
    screen.blit(mouse_cursor, (x, y))
    x+=10                              ←  animated in steps of 10 pixels
    pygame.display.update()
```

Timing and Framerate

- Problem: The previous example creates an animation that runs in different speed depending on the power of the cpu
- Solution: time-based animations
- pygame.time.Clock() provides an appropriate tool for time-based animations
- Clock.tick() returns the time that passed since its last call

```
clock = pygame.time.Clock()
clock.tick()
```

Moving an object time-based

```
...

clock = pygame.time.Clock()

speed = 300.0

x = 0 - mouse_cursor.get_width()
y = 10
while True:
    time_passed = clock.tick() / 1000.0
    moved_distance = time_passed * speed
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((255,255,255))
    if x > screen.get_width():
            x = 0 - mouse_cursor.get_width()
    screen.blit(mouse_cursor, (x, y))
    x+=moved_distance
    pygame.display.update()
```

$\leftarrow$ speed in pixels per second

$\leftarrow$ time passed since last tick() in seconds

$\leftarrow$ distance moved since last call

$\leftarrow$ move the sprite the calculated distance

Moving an object high level

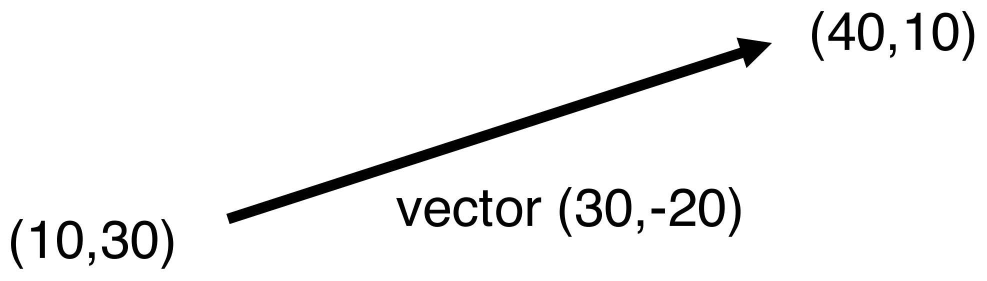
- In Cocos2d-x, each Action has a *By* and *To* version
  - **By** is relative to the current state of the Node.
  - **To** is absolute, meaning the current state of the Node is irrelevant.

```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setPosition(Vec2(200, 256));

// MoveBy - lets move the sprite by 500 on the x axis over 2 seconds
// MoveBy is relative - since x = 200 + 200 move = x is now 400 after the move
auto moveBy = MoveBy::create(2, Vec2(500, mySprite->getPositionY()));

// MoveTo - lets move the new sprite to 300 x 256 over 2 seconds
// MoveTo is absolute - The sprite gets moved to 300 x 256 regardless of
// where it is located now.
auto moveTo = MoveTo::create(2, Vec2(300, mySprite->getPositionY()));
```

Diagonal Movement

or: Vectors, yeah!

- Moving a sprite to a specific coordinate requires movement on the x- and y-axis
- Best achieved using vectors
- E.g. a vector of (10,30) means move 10 pixels on the x- and 30 on the y-axis
- Store vectors as tuples, lists or create a class



Vectors I

- Example class

```
class Vector(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "vector (%s,%s)"%(self.x, self.y)

    @classmethod
    def vector_from_points(cls,from_p,to_p):
        return cls(to_p[0]-from_p[0],to_p[1]-
from_p[1])
```

Use:

```
vector1 = Vector(10.0,20.0)
print vector1

print
Vector.vector_from_points((10,10),
(30,10))
```

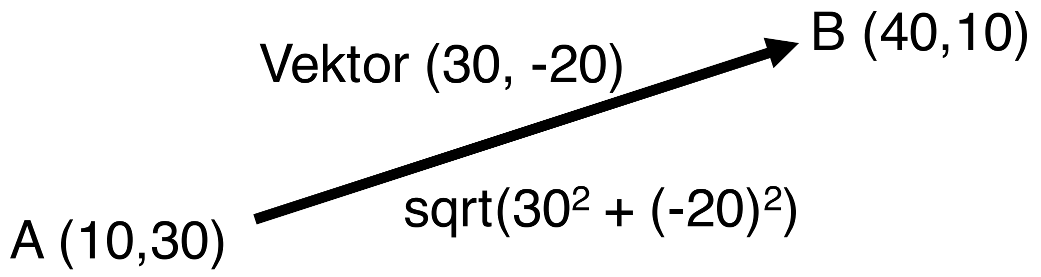
Output:

```
vector (10.0,20.0)
vector (20,0)
```

Vectors II

- Vector magnitude

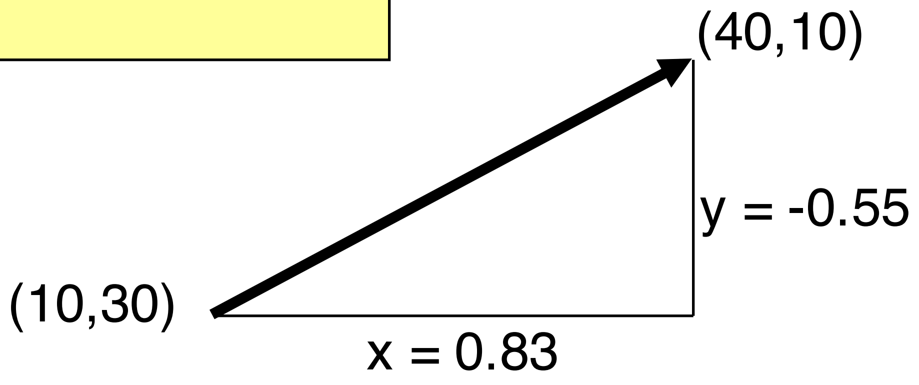
```
Import math
class Vector(object):


...


   def get_magnitude(self):
      return math.sqrt(self.x**2 + self.y**2)
```



Vectors 3

- Normalizing a vector

```
class Vector(object):
...

    def normalize(self):
        magnitude = self.get_magnitude()
        self.x /= magnitude
        self.y /= magnitude
```



Diagonal movement using vectors 1

```
import pygame
from pygame.locals import *
from sys import exit
import math

class Vector(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __str__(self):
        return "(%s, %s)"%(self.x, self.y)

    def get_magnitude(self):
        return math.sqrt(self.x**2 + self.y**2)

    def normalize(self):
        magnitude = self.get_magnitude()
        self.x /= magnitude
        self.y /= magnitude

    @classmethod
    def vector_from_points(cls,from_p,to_p):
        return cls(to_p[0]-from_p[0],to_p[1]-from_p[1])
```

$\leftarrow$ needed for magnitude calculation

Diagonal movement using vectors 2

```
...
mpos = (0.0,0.0)
destination  = (500,430)
player_image  = 'head.jpg'
pygame.init()

screen = pygame.display.set_mode((640,  640), 0, 32)
pygame.display.set_caption("Animate  X!")

mouse_cursor = pygame.image.load(player_image).convert_alpha()
clock = pygame.time.Clock()

speed = 300.0 # pixels per second
heading  = Vector.vector_from_points(mpos,  destination)
heading.normalize()

...
```

$\leftarrow$ start and end positions

$\leftarrow$ calculate the vector and normalize it

Diagonal movement using vectors 3

```
...


while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((255,255,255))

    time_passed = clock.tick() / 1000.0
    moved_distance = time_passed * speed

    screen.blit(mouse_cursor,mpos)
    mpos= (mpos[0]+heading.x  * moved_distance,mpos[1]  + heading.y  *
moved_distance)
    pygame.display.update()
```

within the event loop …

… the distance is calculated as usual

new position is based on the normalized vector

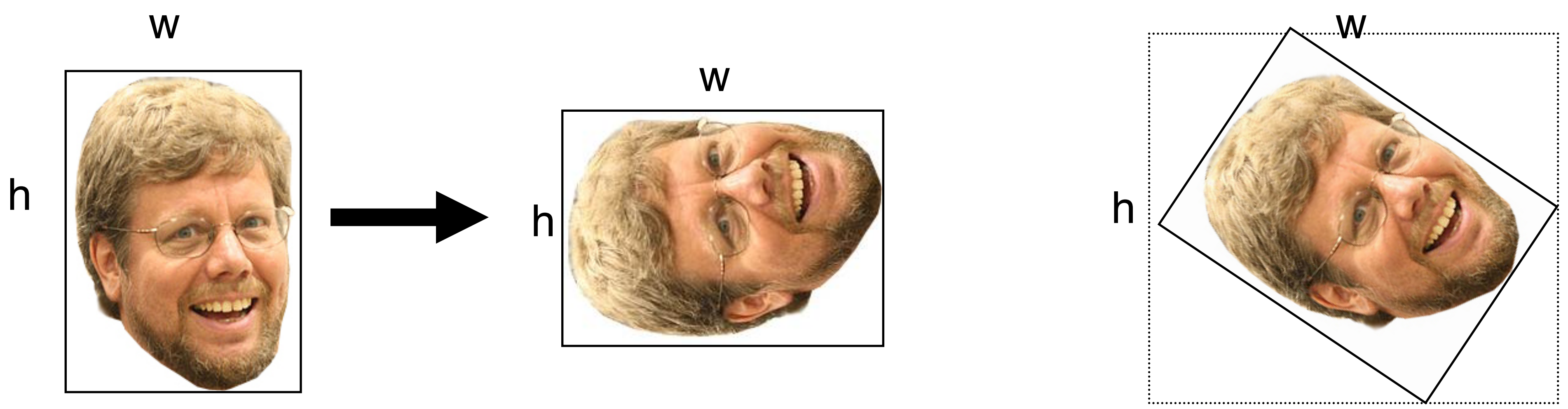
Diagonal movement high level

```
auto mySprite = Sprite::create("mysprite.png");

// Move a sprite to a specific location over 2 seconds.
auto moveTo = MoveTo::create(2, Vec2(50, 50));
mySprite->runAction(moveTo);

// Move a sprite 50 pixels to the right, and 50 pixels to the top over 2 seconds.
auto moveBy = MoveBy::create(2, Vec2(50, 50));
mySprite->runAction(moveBy);
```

Rotating Surfaces

- Use pygame.transform.rotate(surface,angle) to rotate a surface (counterclockwise)
- Returns a new Surface Object
- Attention: the new Surface can have different width and height than the original

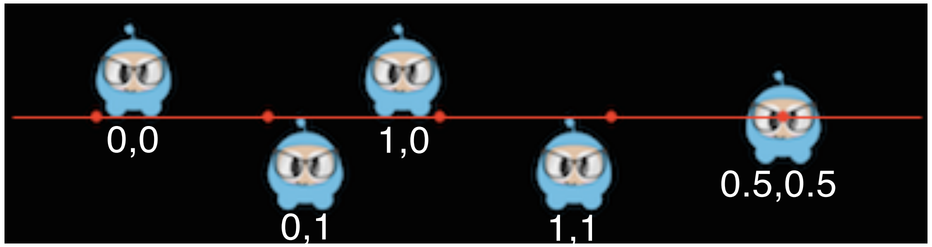
```
rotated_surface = pygame.transform.rotate(old_surface,90)
```



Anchor Points

Anchor point is a point that you set as a way to specify what part of the Sprite will be used when setting its position.
Important for all transformations: e.g. *scale, rotation, skew*.

```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setAnchorPoint(0.5, 0.5); // DEFAULT anchor point
mySprite->setAnchorPoint(0, 0); // bottom left
mySprite->setAnchorPoint(0, 1); // top left
mySprite->setAnchorPoint(1, 0); // bottom right
mySprite->setAnchorPoint(1, 1); // top right
```

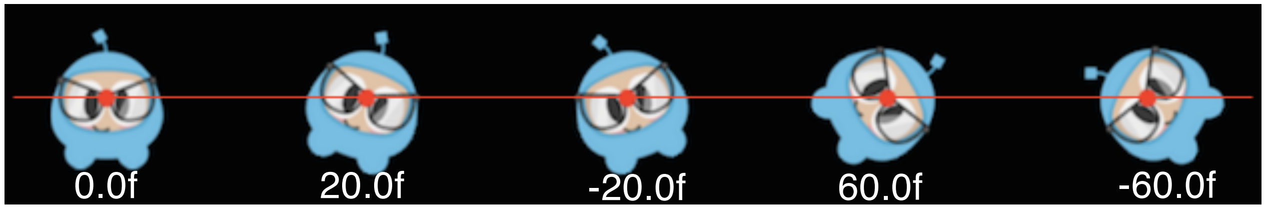


http://www.cocos2d-x.org/docs/programmers-guide/3/index.html

Rotating Surfaces

- Use mySprite->setRotation(angle); to rotate a surface
- Use rotateTo and rotateBy to animate a rotation
- Positive values rotate the Sprite object clockwise, while negative values rotate the Sprite object counter-clockwise.

```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setRotation(20.0f); // rotate sprite by +20 degrees
mySprite->setRotation(-20.0f); // rotate sprite by -20 degrees

auto rotateTo = RotateTo::create(2.0f, 60.0f); // Rotates to a specific angle over 2 seconds
mySprite->runAction(rotateTo);
auto rotateBy = RotateBy::create(2.0f, -60.0f); // Rotates by a specific angle over 2 seconds
mySprite->runAction(rotateBy);
```

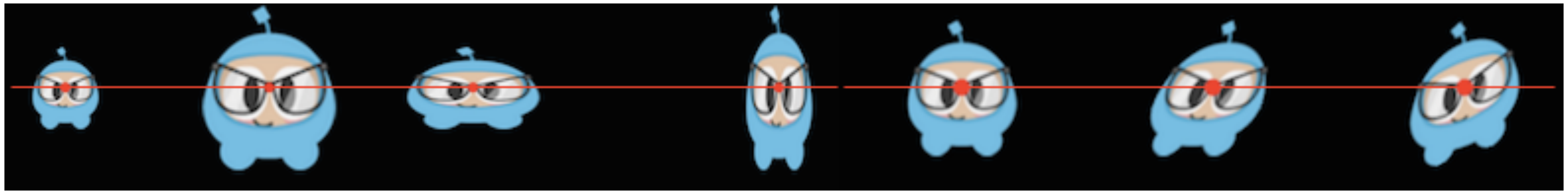


http://www.cocos2d-x.org/docs/programmers-guide/3/index.html

Scaling and Skewing Surfaces

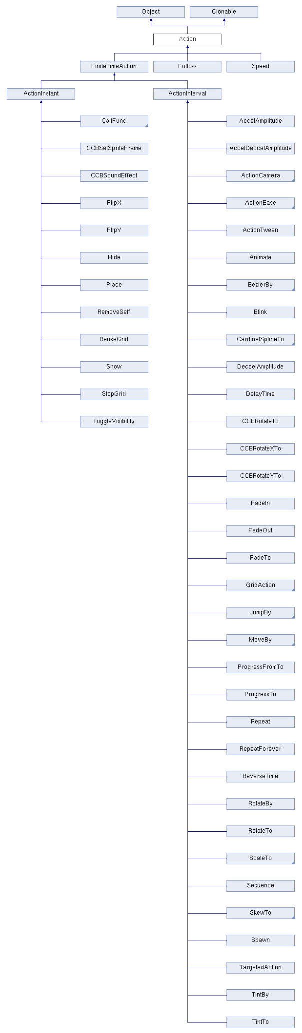
- Use mySprite->setScale(factor); to scale a surface
- Use mySprite->setSkew(position); to skew a surface
- Also: scaleBy, scaleTo, skewBy, skewTo

```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setScale(2.0); // increases X and Y size by 2.0 uniformly
mySprite->setScaleX(2.0); // increases just X scale by 2.0
mySprite->setSkewX(20.0f); // adjusts the X skew by 20.0
mySprite->setSkewY(20.0f); // adjusts the Y skew by 20.0
```



http://www.cocos2d-x.org/docs/programmers-guide/3/index.html

And many, many more…



TargetedAction, **Animate**, ReverseTime, DelayTime, TintBy, TintTo, FadeTo, FadeOut, FadeIn, Blink, ScaleBy, ScaleTo, BezierTo, BezierBy, JumpTo, JumpBy, **Ease**BackInOut, SkewBy, EaseBackOut, __CCCallFuncO, EaseBackIn, SkewTo, EaseBounceInOut, MoveTo, EaseBounceOut, __CCCallFuncND, MoveBy, EaseBounceIn, CallFuncN, EaseBounce, RotateBy, EaseElasticInOut, EaseElasticOut, RotateTo, SplitCols, CallFunc, EaseElasticIn, CatmullRomBy, SplitRows, ReuseGrid, Spawn, Follow, EaseElastic, Twirl, CCBEaseInstant, StopGrid, JumpTiles3D, CatmullRomTo, EaseSineInOut, CCBRotateYTo, DeccelAmplitude, Waves, EaseSineOut, RepeatForever, WavesTiles3D, CCBRotateXTo, CardinalSplineBy, Place, EaseSineIn, AccelAmplitude, CCBRotateTo, Liquid, Speed, EaseExponentialInOut, TurnOffTiles, FlipY, CCBSoundEffect, CardinalSplineTo, EaseExponentialOut, Shaky3D, FadeOutDownTiles, Repeat, CCBSetSpriteFrame, AccelDeccelAmplitude, EaseExponentialIn, FlipX, Ripple3D, FiniteTimeAction, FadeOutUpTiles, EaseInOut, TiledGrid3DAction, FadeOutBLTiles, RemoveSelf, EaseOut, FadeOutTRTiles, OrbitCamera, Sequence, Lens3D, EaseIn, ToggleVisibility, Grid3DAction, ShuffleTiles, EaseRateAction, FlipY3D, Hide, FlipX3D, ActionInterval, ActionTween, ProgressFromTo, Show, ActionCamera, ShatteredTiles3D, ActionEase, Waves3D, LuaCallFunc, ActionInstant, PageTurn3D, ProgressTo, GridAction, and ShakyTiles3D.

http://www.cocos2d-x.org/reference/native-cpp/V3.0alpha0/db/d61/classcocos2d_1_1_action.html

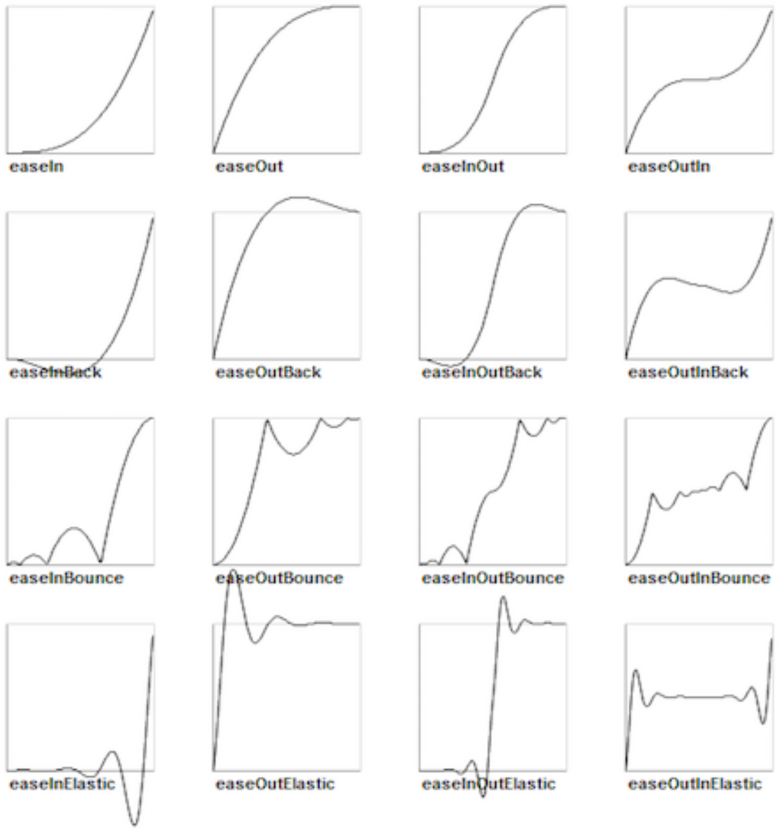
Frame by Frame Animation

```
auto mySprite = Sprite::create("mysprite.png");

// now lets animate the sprite we moved
Vector<SpriteFrame*> animFrames;
animFrames.reserve(6);
animFrames.pushBack(SpriteFrame::create(“anim1.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("anim2.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("anim3.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("anim4.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("anim5.png", Rect(0,0,65,81)));
animFrames.pushBack(SpriteFrame::create("anim6.png", Rect(0,0,65,81)));

// create the animation out of the frames
Animation* animation = Animation::createWithSpriteFrames(animFrames, 0.1f);
Animate* animate = Animate::create(animation);

// run it and repeat it forever
mySprite->runAction(RepeatForever::create(animate));
```

http://www.cocos2d-x.org/docs/programmers-guide/4/index.html

Easing

Animating with a specified acceleration to make the animations smooth.



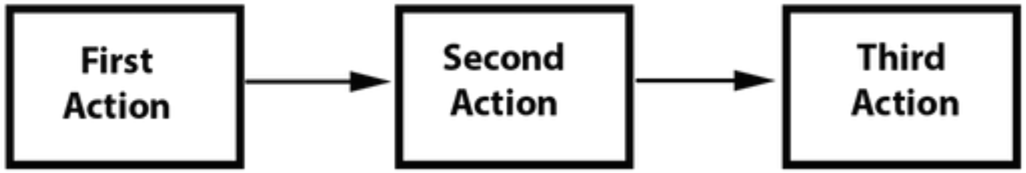
http://www.cocos2d-x.org/docs/programmers-guide/4/index.html

Easing

```
auto mySprite = Sprite::create("mysprite.png");

// create a MoveBy Action to where we want the sprite to drop from.
auto move = MoveBy::create(2, Vec2(200, dirs->getVisibleSize().height - newSprite2->getContentSize().height));
auto move_back = move->reverse();

// create a BounceIn Ease Action
auto move_ease_in = EaseBounceIn::create(move->clone() );

// create a delay that is run in between sequence events
auto delay = DelayTime::create(0.25f);

// create the sequence of actions, in the order we want to run them
auto seq1 = Sequence::create(move_ease_in, delay, move_ease_in_back, delay->clone(), nullptr);

// run the sequence and repeat forever.
mySprite->runAction(RepeatForever::create(seq1));
```

http://www.cocos2d-x.org/docs/programmers-guide/4/index.html

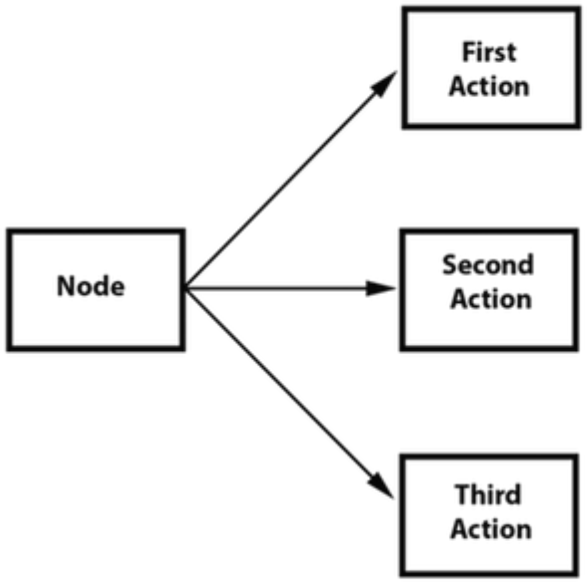
Sequences



```
auto mySprite = Sprite::create("mysprite.png");

auto jump = JumpBy::create(0.5, Vec2(0, 0), 100, 1); // create actions.
auto rotate = RotateTo::create(2.0f, 10);

auto callbackJump = CallFunc::create([](){   // create callbacks.
    log("Jumped!");
});

auto callbackRotate = CallFunc::create([](){
    log("Rotated!");
});

// create a sequence with the actions and callbacks
auto seq = Sequence::create(jump, callbackJump, rotate, callbackRotate, nullptr);

// run it
mySprite->runAction(seq);
```

http://www.cocos2d-x.org/docs/programmers-guide/4/index.html

Spawn



http://www.cocos2d-x.org/docs/programmers-guide/4/index.html