

# OOP Techniques in a Simple Game

Radu Albastroiu



# What game?



# Where can I see the code for the final version?

This project was developed for this course and you are encouraged to continue working on it or to build a new game based on this foundation.

You can find the repository on GitHub at:

<https://github.com/RaduAlbastroiu/Minotaur>

There are instruction on how to clone and run it, how to start a new project and few tutorials that I found helpful.

For other questions: [radu.albastroiu@caphyon.com](mailto:radu.albastroiu@caphyon.com)

# Technologies and platforms

## Technologies:

- C++
- Cocos2d-x game engine

## Platforms:

- Windows
- MacOS
- Linux
- Android
- iOS

# Cocos2d-x

For this project we will use the Cocos2d-x game engine.

Cocos2d-x because:

- Smaller
- Easy to use
- Cross-platform



# **Coding - Stage 1**

# OOP Concepts

These are also called as four pillars of Object Oriented Programming.

- Inheritance
- Encapsulation
- Abstraction
- Polymorphism

# Inheritance

Inheritance expresses is-a/has-a relationship between two objects.

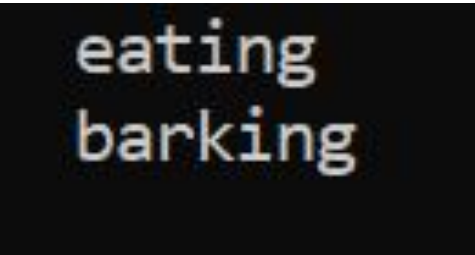
```
class Animal
{
public:
    void eat() { std::cout << "  eating  \n"; }
};

class Dog : public Animal
{
public:
    void bark() { std::cout << "  barking  \n"; }
};

int main()
{
    auto dog = Dog();

    dog.eat();
    dog.bark();
}
```

Output:



```
eating
barking
```

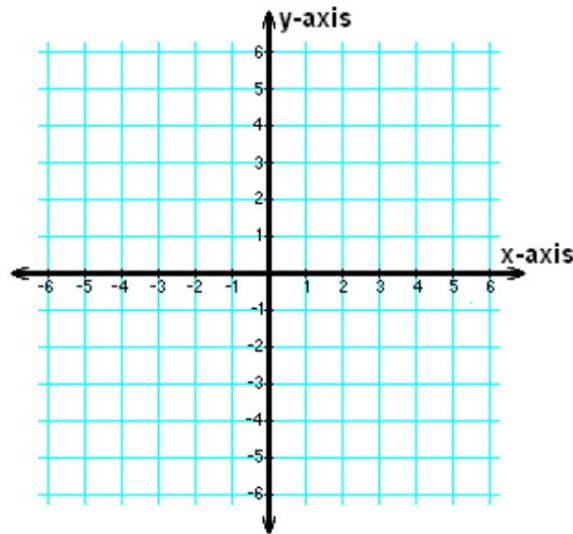


# Cocos2d::Scene

Cocos2d game engine is based on nodes.

Scene is a node with the property that its anchor point by default is in the center of the screen.

Cocos2d coordinate system origin is in the bottom left corner of the screen.



# Cocos2d::Sprite

Sprite just like the Scene is another Node than can be created using an image using formats like PNG, JPEG, TIFF and others.

Unlike the Scene we need to set its position on the screen.

On a sprite we can do actions like scaling, modifying opacity, changing rotation, changing position, running animations and many other.



## **Coding - Stage 2**

# SOLID Principles

SOLID is an acronym for 5 important design principles when doing Object Oriented Programming.

- S - Single responsibility principle
- O - Open/Closed principle
- L - Liskov substitution principle
- I - Interface segregation principles
- D - Dependency inversion principle

## **S - Single responsibility principle**

In programming, the Single Responsibility Principle states that every module or class should have responsibility over a single part of the functionality provided by the software.

# KISS Principle

**Keep It Simple, Stupid**

**Programming languages are for humans to understand, so keep coding simple.**

**Keep your methods small each method should never be more than 30-40 lines.**

**Each method should only solve one small problem.**

# OOP Concepts

These are also called as four pillars of Object Oriented Programming.

- Inheritance
- Encapsulation
- Abstraction
- Polymorphism

# Encapsulation

Encapsulation is the mechanism of hiding of data implementation by restricting access to public methods.

This Concept is also often used to hide the internal representation, or state, of an object from the outside. This is called information hiding.



# Public, Protected, Private

Public class members and functions can be used from outside of a class by any function or other classes.

Protected class members and functions can be used only inside a class or its derived classes.

Private class members and functions can be used only inside of class.

# Design patterns

In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design.

Design patterns can speed up the development process by providing tested, proven development paradigms.

## **Coding - Stage 3**

# Game Loop

Pretty much every game ever made has a game loop, even if it's hidden by the game engine.

A Cocos2d-x game is no exception, although it might not be immediately obvious.

If the target frame rate is 60 FPS, then every 16,667 ms the game loop will run once.

# Game Loop

```
void GameLoop()  
{  
    while (game != DONE)  
    {  
        getInput();  
        physicsEngine.stepForward();  
        updateWorld();  
        render();  
    }  
    cleanup();  
}
```

---

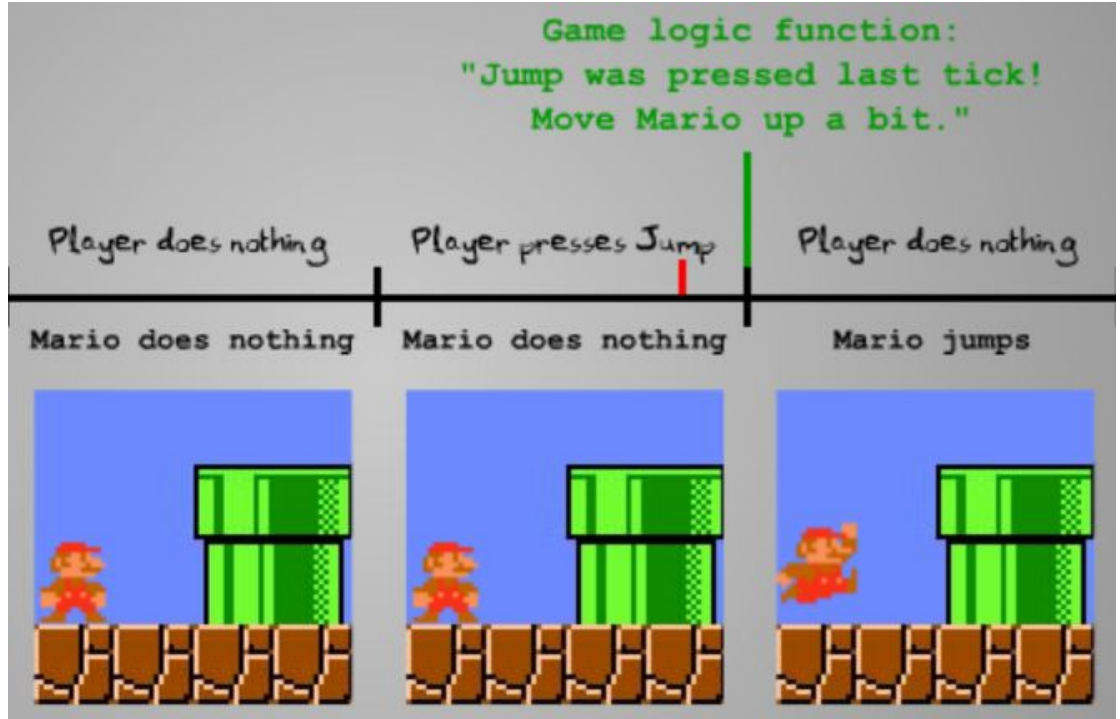
**As you can see, it's quite literally a loop that calls various functions that make you game a game.**

**This is obviously a primitive example of game loop, but most of them end up looking very similar to this.**

# Game Loop

Simple example of a game loop:

- Player does nothing, Mario does nothing
- Player presses jump, Mario does nothing
- Player does nothing, Mario jumps



## **Coding - Stage 4**

# Auto Keyword

Type inference refers to automatic deduction of the data type of an expression in a programming language.

Before C++ 11, each data type needs to be explicitly declared at compile time.

```
auto x = 4;
```

x is of type int

```
auto y = 3.37;
```

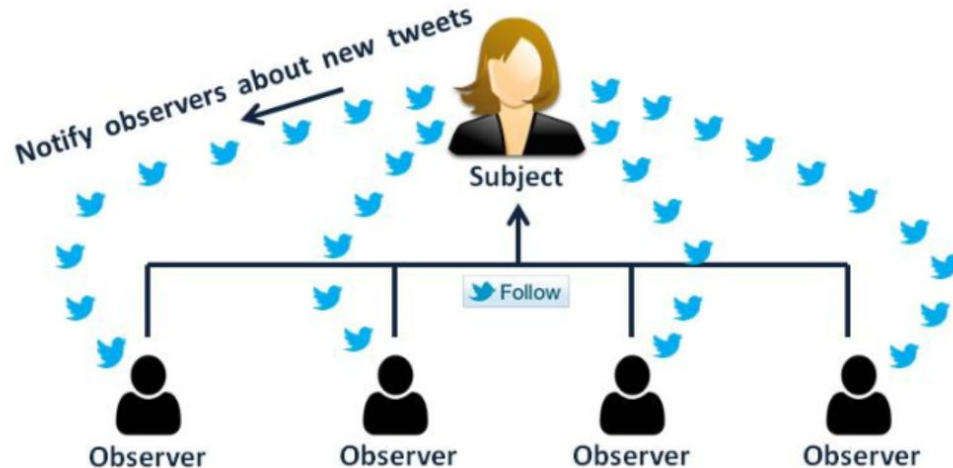
y is of type double



# Observer Pattern

Define a one to many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

## Observer Design Pattern



# Input

Cocos2d supported event types:

- Mouse
- Touch
- Keyboard

# Keyboard Events

Cocos2d lets you to create an event listener that can receive be registered to receive keyboard events.

This process is similar with the Game Loop Update registration, using Observer pattern.

# Keyboard Events

What if you wanted to check to see if the spacebar was pressed at any given time?

You can't, Cocos2d-x is entirely event based.

There is a work around:

- When a key is pressed we store it in a `std::map` using `KeyCode` and current time
- When the key is released, we remove the released key from the map.

Therefore at any given time, we know which keys are pressed and for how long.

# Std::Map

Map is a sorted associative container that contains key-value pairs with unique keys.

Operations:

- Insert a pair of values (key, value) in a map
- Find a pair of values using only the key.
- Iterate through the map
- Erase a pair of values using only the key.

# Register Keyboard Events

To register key pressed we have to provide implementation for the event listener for the `OnKeyPressed`.

The same for key released using `OnKeyReleased`.

How we can provide implementation? Using a lambda function.

# Lambda Function

Simply put a lambda function is a function without a name, an inline function, which can be used for short snippets of code that are not going to be reuse and not worth naming. Lambda expression can be defined as:

```
[ capture ] (parameters) -> return type  
{  
  // logic goes here  
}
```

## **Coding - Stage 5 - 6**



# Questions

**End of part 1**

# Hello Again

Radu Albastroiu



# What we learned so far?

- What is Inheritance
- S from SOLID principles
- KISS - Keep It Simple, Stupid
- Encapsulation
- Observer Pattern
- Lambda function

# What we built so far?

- Static background, health label, static score
- Hero sprite
- Keyboard listener
- Move Hero using the keyboard listener, without going outside the screen

# Refactoring

Code refactoring is the process of restructuring existing computer code.

Benefits:

- Maintainability, easier to solve bugs
- Extensibility, it is easier to extend the capabilities of the application

# DRY - Principle

Don't Repeat Yourself

Just Don't

## **Coding - Stage 7 - 10**



# Stage 6 vs Stage 10

What's the difference ?

It does the same thing almost.

Let's see the roles of objects in stage 6 and after the refactor in stage 9.

# Stage 6 vs Stage 10

AppDelegate - Create Scene

MainScene - Init Hero, Background,  
HealthBar, ScoreLabel, Keyboard

MainScene - Update Hero

MainScene - Move Hero

MainScene - Interogate keyboard

MainScene - Update Health Bar

AppDelegate - Create Scene

MainScene - Init Hero, Background,  
HealthBar, ScoreLabel, Keyboard

MainScene - Update Hero

# **Coding - Stage 11**

# OOP Concepts

These are also called as four pillars of Object Oriented Programming.

- Inheritance
- Encapsulation
- **Abstraction**
- Polymorphism

# Abstraction

- A concept or an Idea which is not associated with any particular instance.
- Abstraction is achieved in C++ using Pure Abstract Classes.
- A pure abstract class is a class without implementation or members from which another object will inherit.
- A frequently used term in other languages is **Interface**.

# OOP Concepts

These are also called as four pillars of Object Oriented Programming.

- Inheritance
- Encapsulation
- Abstraction
- Polymorphism

# Polymorphism

- Greek word for one name many forms
- There are 2 types of polymorphism static and dynamic
- Static achieved by using Method Overloading
- Dynamic achieved by using Method Overriding

# **Coding - Stage 12**



# Updates

**Slides on github**

# **Stages on github**

# Questions

**End of part 2**

# Hello Again

Radu Albastroiu



# What we learned so far?

- What is Inheritance
- Single Responsibility Principle
- KISS - Keep It Simple, Stupid
- Encapsulation
- Observer Pattern
- Lambda function
- DRY - Don't repeat yourself
- Abstraction
- Polymorphism

# What we built so far?

- Static background, health label, static score
- Hero sprite
- Keyboard listener
- Move Hero using the keyboard listener, without going outside the screen
- Keyboard profiles
- Refactored all code, the 2 heros example



**What we have for today**

# Memory management

**But first**



# Animations



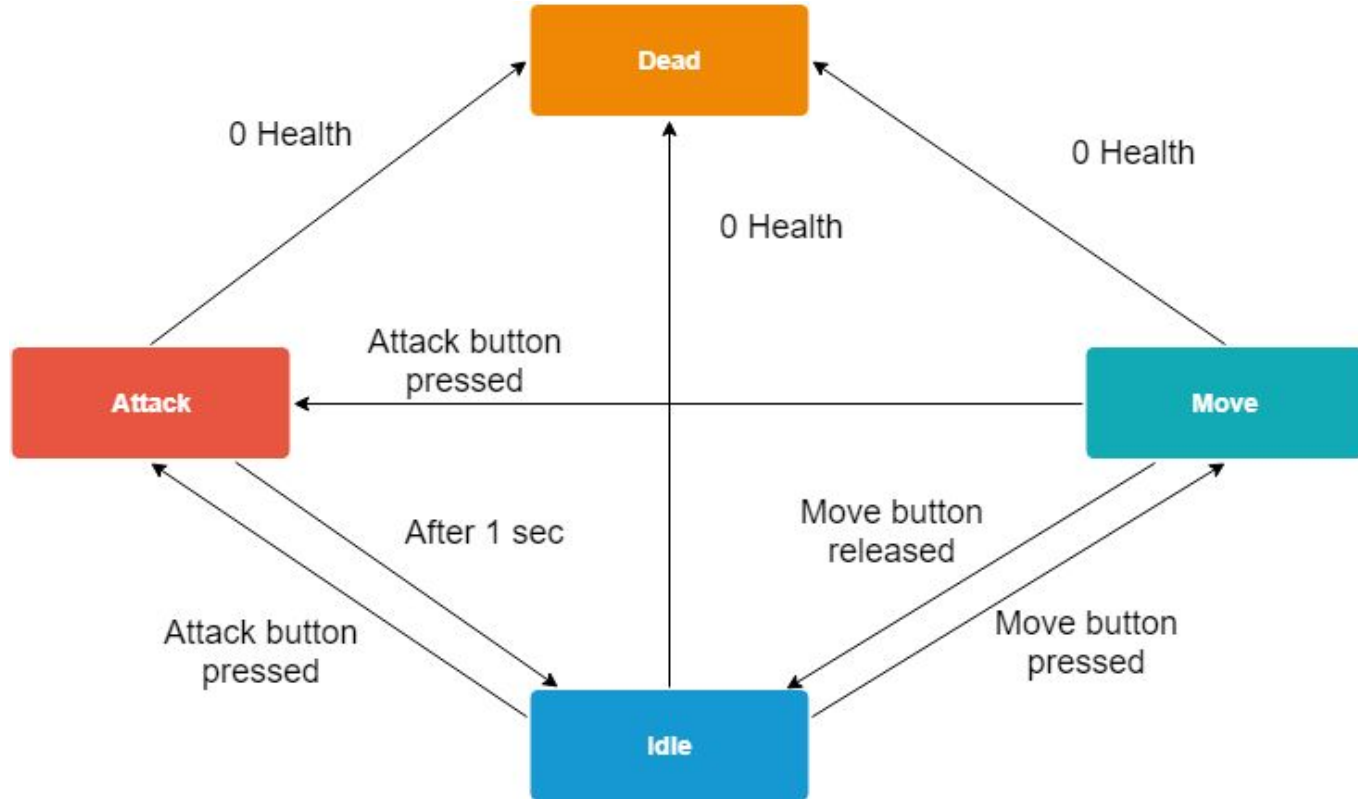
# Sprite Atlas

A set of in a single png file.

A sprite atlas an associated xml or plist file with information about the position of each sprite in that file.



# State machine



## **Coding - Stage 13-17**

# Memory Management



# Further Work

# Create a new enemy type.

- More health
- Faster
- Stronger
- Different spawn locations



# Where I send my project?

Send your project at: [open4tech@caphyon.com](mailto:open4tech@caphyon.com)

You can find the repository on GitHub at:

<https://github.com/RaduAlbastroiu/Minotaur>

There are instruction on how to clone and run it, how to start a new project and few tutorials that I found helpful.

For other questions: [radu.albastroiu@caphyon.com](mailto:radu.albastroiu@caphyon.com)

**Thank you.**