OULU UNIVERSITY OF
APPLIED SCIENCES

Lal Jung Gurung

IOS GAME DEVELOPMENT USING SPRITEKIT FRAMEWORK WITH SWIFT PROGRAMMING LANGUAGE

IOS GAME DEVELOPMENT USING SPRITEKIT FRAMEWORK WITH SWIFT PROGRAMMING LANGUAGE

Lal Jung Gurung
Bachelor's Thesis
Spring 2016
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Lal Jung Gurung
Title of the bachelor's thesis: iOS Game Development using SpriteKit Framework with Swift Programming Language
Supervisor: Kari Laitinen
Term and year of completion: 2016 Number of pages: 42

iOS is a mobile operating system for Apple manufactured phones and tablets. Mobile Gaming Industries are growing very fast, and compatibility with iOS is becoming very popular among game developers. The aim of this Bachelor's thesis was to find the best available game development tools for iOS platform.

The 2D game named Lapland was developed using Apple's own native framework, SpriteKit. The game was written with the Swift programming language. The combination of SpriteKit and Swift help developers to make game development easy and fast, focusing more time on gameplay than other tedious tasks.

The result was the game running successfully on iOS devices. SpriteKit provides all the necessary tools for creating a 2D-game for the iOS platform.

Keywords: iOS, Game Development, SpriteKit, Swift Programming Language.

PREFACE

During my training program in Oulu Game Lab [1], I developed a game using a Unity game engine for android platform [2]. I became familiar with the Unity and android platform. In this thesis, I try to find best game development tools to develop a 2D-game for the iOS platform.

I would like to thank my thesis supervisor Kari Laitinen for his support and advice during my thesis. We had meeting every two weeks. I would also like to thank my language teacher Kaija Posio. GameArt2D provides all the 2D game art assets for the game.

May 2016, Oulu
Lal Jung Gurung

CONTENTS

VOCABULARY

TERM	MEANING
iOS	iPhone Operating System
2D	2-dimensional
3D	3-dimensional
App	Application
WWDC	Apple Worldwide Developers Conference
iPAD	iOS-based line of tablet computer
Mac	Macintosh
Apple TV	A digital media player and a micro console
Apple Watch	Smartwatch developed by Apple
GPU	Graphics Procressing Unit
OpenGL	Open Graphics Library
IDE	Integrated Development Environment
LLVM	Low Level Virtual Machine
Lapland	Game Title
Cocao Touch	User Interface framework for building software programs to run on iOS

1 INTRODUCTION

Games are the most popular app category in mobile stores like Apple App store and Google Play. App downloaders are also most willing to pay for games. The increase in popularity of mobile games has allowed an increased distribution on gaming platforms such as Android, iOS, Windows Phone and other Multiplatform support.

The aim of this thesis was to find suitable game development tools for beginners to make a 2D-game on the iOS platform. Apple's own native game framework SpriteKit was chosen to develop the game with the Swift programming language. The decision was made on the basis of a native performance, a platform integration, a future proof development and a developer friendliness.

This thesis is divided into four chapters; Evaluating iOS game development platform, An Introduction to SpriteKit Framework, Introduction to Lapland and Working with SpriteKit Framework.

2 EVALUATING IOS GAME DEVELOPMENT PLATFORM

2.1 iOS Game Development Platform Introduction

iOS has become a great platform for developers to develop games [3]. In WWDC 2013, Apple finally understood the significance of games for their platforms and announced Sprite Kit followed by Scene Kit and Metal in coming. iOS games are better every year with the advancements in game development frameworks. iOS 9 provides powerful gaming technologies to build better quality games.

2.2 iOS Game Development History

In early days, game developers used OpenGL API to make 2D and 3D graphics on the screen [4]. Foundation and CocoaTouch was on the upper level to manipulate UIKit objects. The development features like Sprite, partical emitters, maps, bounding boxes requires lower level structuring. In 2011, Apple introduced GLKit framework to reduces the effort provided by earlier version of OpenGL. Third-party frameworks, such as Unity, GameMaker, Unreal Engine, Cocos2D also made the game development easier for the developers by keeping the design aspect of the game more focus. Since none of these third-framework were supported by Apple, there would be integration problem with new version of iOS. Finally, Apple introduced SpriteKit framework for making 2D games in 2013.

2.3 Comparing with Other Platform

iOS Games can be made in other third party platform engines like Unity, Unreal, Cocos2D and so on. Third party platform engines are especially popular among the indie developers. Third party platforms are closed source and mostly cost money for full version.

Before the launch of SpriteKit, Cocos2D was a popular third party framework for creating iOS games. Being Apple's native game development engine for iOS platform, SpriteKit has emerged as a serious challenger to Cocos2D. Gaming Platforms are compared in Table 1.

TABLE 1. Comparing SpriteKit, Cocos2D and Unity2D [5]

Platform	Pros	Cons
SpriteKit	• Native framewrok, supported by Apple • Built in to Xcode • Easier for beginners	• New framework, still more upgrades to come.
Cocos2D	• Perfect fo casual games • Free and open source • Hardware accelerated graphics and good performance	• No large commercial entity support and bug fixes • APIs are somewhat unorthodox
Unity2D	• Cross platform • strong community of asset and plugin creators	• Need to pay for full licence. • Collaboration is difficult • Performance is not great • The engine source code is not available.

2.4 Tools, Technologies and Capabilities.

2.4.1 Xcode

The Xcode is an integrated development environment (IDE) for developing applications for Mac, iPhone, iPad, Apple Watch, and Apple TV [6]. Xcode is integrated with the Cocoa and Cocoa Touch frameworks. Xcode comes with a new interactive environment called Playground. This allows developers to interactively try out content as well as seeing both the final results and intermediate calculations, leading to some impressive possibilities. Figure 1 shows a window showing code in action.

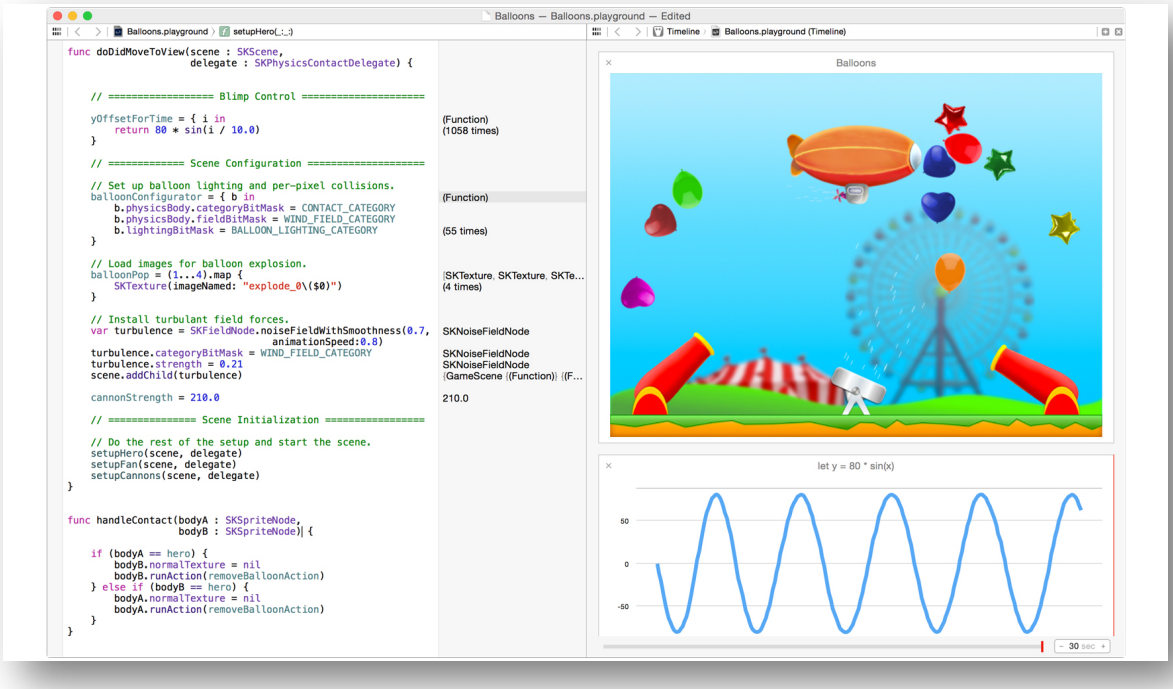

FIGURE 1. A 3 panel window showing code in action.

2.4.2 Apple Game Development Framework

Apple provides three game development frameworks to make games for iOS platform [7]. They are SpriteKit, SceneKit and Metal. iOS games are better than ever with the latest advancements in Sprite Kit, Scene Kit and Metal. iOS 9 provides more powerful and easy to use gaming technologies like GameplayKit, ReplayKit, and Model I/O. Table 2 shows the differences between SpriteKit, SceneKit and Metal.

TABLE 2. Differences between SpriteKit, SceneKit and Metal.

Sprite Kit	Scene Kit	Metal
• Designed for 2D games • Focus on simplicity and automation • Fast to learn • Can show 3D objects from scenekit	• Create 3D Games • Similar style to spritekit • Requires higher resources • A great next step from spritekit	• Full control over the GPU to do whatever you can imagine • Not a beginner friendly • Making games takes a lot longer • Possibly less resources available

SpriteKit is the best framework to start learning and developing games. After SpriteKit, we can surely start developing games with SceneKit and Metal in future.

2.4.3 Programming Language

SpriteKit games can be written in two programming languages – Objective-C and Swift.

Objective-C is not a friendly language to learn. It has an insane amount of history and it adds a lot of complexity to beginners. Swift is defined as a modern language which was designed to run smoothly and be efficient and friendly with newcomers.

Advandage of Swift Programming Language [8]:

- Error handling model and syntax improvements.
- Swift is open source.
- Swift is new programming language, combined with years of experience building Apple platforms.
- Playgrounds make writing swift code very easy.
- Swift was designed to be safe.
- Swift is very fast. It uses Using high-performance LLVM compiler.
- Interoperability with Objective-C.

3 INTRODUCTION TO SPRITE KIT FRAMEWORK

3.1 Description

Sprite Kit is a framework for developing 2D games for iOS devices [9]. It is easy to learn, powerful and fully supported by Apple, which makes it more reliable to use than third-party game development engines. As Sprite Kit is a native framework of iOS, it has an in-build support for using the particle effects, texture effects, and physics simulations. The performance of Sprite Kit is better than, that of other third-party frameworks or engines.

3.2 Elements of SpriteKit

SpriteKit game consists of many scenes which are made of nodes, and the functioning of nodes in a scene is determined by actions.

3.2.1 Scene

A level in a game is called as a Scene [10]. It holds all the contents, i.e. nodes and sprites that are to be rendered. A scene in Sprite Kit is represented by SKScene objects [11].

3.2.1 Nodes

The basic building blocks for all the content in a scene is called Node. The SKNode acts like a blueprint for all other nodes. The SKNode have attributes like position, rotation, scale and alpha. The Figure 2 shows the properties of SKNode and the link with other nodes.

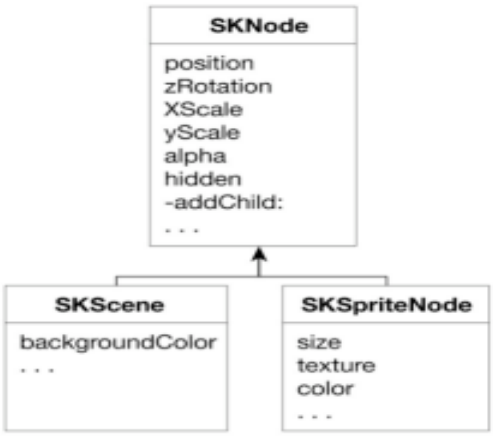

FIGURE 2. SKNode properties.

The SKNode class does not draw anything on a scene by itself, but applies its properties to its descendants. There are node subclasses as follows [12]:

- SKSpriteNode: This can be used for drawing textured sprites and playing video content.
- SK3DNode: This can be used for rendering a SceneKit scene as a 2D textured image.
- SKVideoNode: This can be used for playing video content.
- SKLabelNode: This can be used for rendering a text string.
- SKShapeNode: This can be used for rendering a shape, based on a core graphics path.
- SKEmitterNode: This can be used for creating and rendering particles.
- SKCropNode: This can be used for cropping child nodes using a mask.
- SKEffectNode: This can be used for applying a core image filter to its child node.
- SKLightNode: This can be used for applying lighting and shadows to a scene.
- SKFieldNode: This can be used for applying physics effects to a specific portion of the scene.

3.2.3 Actions

In SpriteKit, Action is used to animate scenes. An action is an object which is used to change the structure of the node in the scene. All actions are implemented by the SKAction class. The most common things that an action can do are as follows [13]:

- Changing a node's position and orientation.
- Changing a node's size or scaling properties.
- Changing a node's visibility or making it translucent.
- Changing a sprite node's contents so that it animates through a series of textures.
- Colorizing a sprite node.
- Playing simple sounds.
- Removing a node from the node tree.
- Calling a block.
- Invoking a selector on an object.

3.3 Features of Sprite Kit

SpriteKit provides many features to facilitate the development of a game. These features can be used for enhancing the experience as well as the performance of the game.

3.3.1 Particle Emitter Editor

Particle emitters are used to create special effects like rain, fire, snow that change over the time [14]. Emitters controls the position and motion of the particle. The following emitter items can be controlled in SpriteKit:

- The location and duration of the particle.
- The amount of particles.
- The size and color of the particle, throughout its lifetime.
- The direction and rotation of the particle from its origin point.

3.3.2 Texture Atlas Generator

Texture atlas is a large image which consists of atlases. Texture atlas is used to enhance the performance of the game [15]. The performance is improved by drawing multiple mages with a single draw call. While building, the compiler search for the name.atlas format files in the folder and all the images within those folders are combined to form a large image files.Figure 3 shows the Artwork folder.

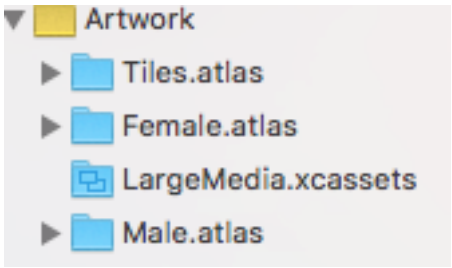

FIGURE 3. Artwork folder

3.3.3 Shaders

Shaders are used to produce a variety of special effects. They calculate rendering effects on a graphic hardware with a high degree of flexibility. In Sprite Kit, shaders are represented by the SKShaderNode class object [16].

3.3.4 Lighting and Shadows

Lighting and shaders effects are produced using the SKLightNode class object [17]. The SKLightNode object can:

- spread a lighting effect at any desirable position on the scene
- add lightning in any sprite
- support colors and shadows

3.3.5 Simulating Physics

In order to imitate the physics, we need to add physics bodies to the scenes [18]. A physics body property uses the SKPhysicsBody class object. In the life cycle of the frame, the didSimulatePhysics function is called just after actions are evaluated. The work of this function is to calculate the physical properties, such as gravity, velocity, friction, restitution, collision. There are three kinds of physics bodies. They are Dynamic, Static and Edge.

TABLE 3. Differences between physics bodies.

Dynamic Volume	Static Volume	Edge
• Physical objects with volume and mass which can be affected by forces and collisions.	• It is unaffected by forces or collisions.	• It is a static volume-less body.
• It is used to move and collide physics bodies.	• It can be used to take up space in the scene.	• It can be used to represent the boundries in the scene.

Figure 4 shows the standard shapes provided by SpriteKit.

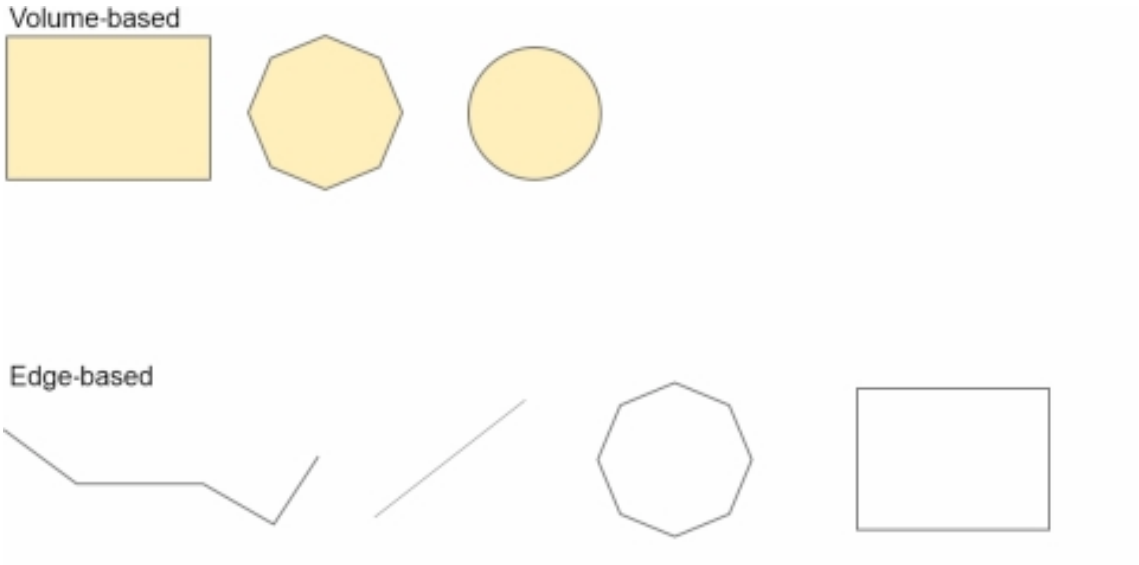

FIGURE 4. Volume-based and Edge-based physics bodies

3.3.6 The Game Loop

The game loop is a important part of every game. The game loop allows the game to run efficiently. Figure 5 shows the rendering loop, which is used by SpriteKit.

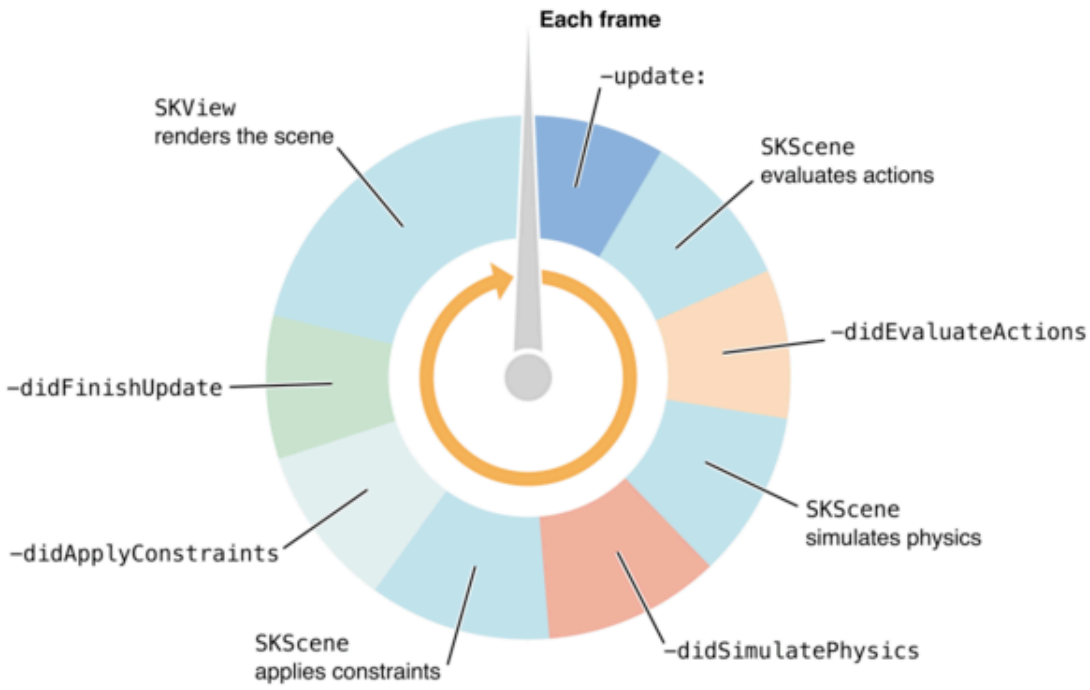

FIGURE 5. The Game Loop

The following game loop steps are explained in SpriteKit Programming Guide provided by Apple [19]:

1. *The update function is the place to implement in-game simulation like running nodes, input handling, artificial intelligence, game scripting, and other game logic.*
2. *The scene processes actions on all the nodes in the tree. It finds all running actions and applies those changes to the tree.*
3. *The scene's didEvaluateActions method is called after all actions for the frame have been processed.*
4. *The scene simulates physics on nodes in the tree that have physics bodies.*
5. *The scene's didSimulatePhysics method is called after all physics for the frame has been simulated.*
6. *The scene applies any contraints associated with nodes in the scene. Contraints are used to establish relationships in the scene.*
7. *The scene calls its didApplyConstraints method.*

8. *The scene calls its didFinishUpdate method. This is the last chance to make changes to the scene.*
9. *Finally, SKView renders the scene. The frame is complete and it continues 60 times per second.*

4 INTRODUCTION TO LAPLAND

4.1 Game Description

Lapland is a beautiful looking platformer game in a cold and icy winter tileset. The free running player task is to jump between suspended platforms and obstacles to advance the game. The player also has to collect gems to unlock the next level.

FIGURE 6. Start Game

4.2 Art and Sound

Art and sound are legally downloaded and purchased from gameart2d [20]. The art and sound are under public domain [21]. We can use them for personal and commercial use.

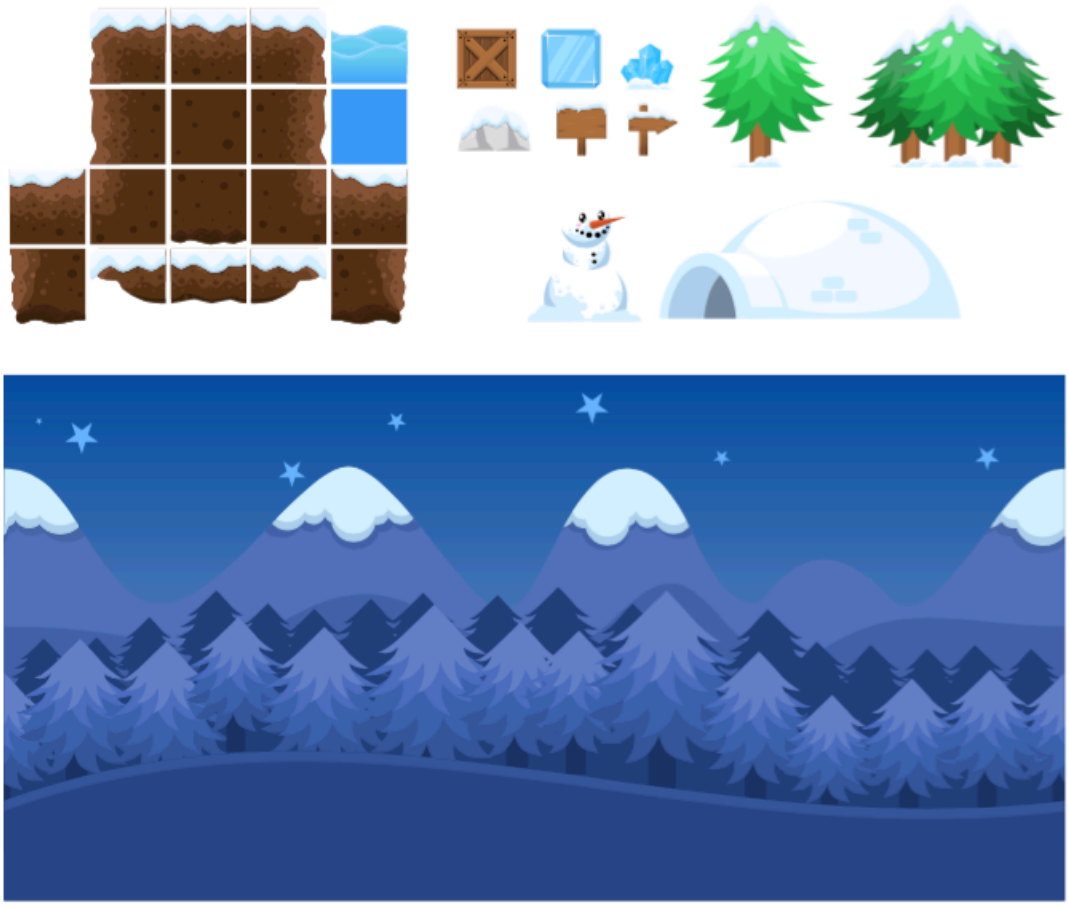

FIGURE 7. Game's background, tiles and objects

FIGURE 8. Female and Male Characters

4.3 Result

The end result is a landscape mode arcade game made with SpriteKit and Swift 2. The game is sucessfully tested and running in iPad air and iPhone 5s. The gameplay trailer can be found in youTube - https://www.youtube.com/watch?v=za0bcRH-DZ0

FIGURE 9. Game running in iPad Air

FIGURE 10. Game running in iPhone 5s

4.4 Future Development

The game demo was created. The game currently consists of two complete levels. More levels, enemy characters, tiles and objects in the game will be added in future. Also, a new interested game developer mainly an artist and a programmer will be recruited.

FIGURE 11. Level Select Screen

5 WORKING WITH SPRITEKIT FRAMEWORK

5.1 Description

With the release of iOS 7.0, Apple introduced its own native 2D game framework called SpriteKit. SpriteKit is a great 2D game engine which has e.g. support for sprite, animations, filters, masking, physics and many more.

The Xcode IDE makes it easier for everyone to build apps and run them directly on their Apple devices. The Swift programming language is becoming faster and easier to write with every new update. Apple also provides the SpriteKit Programming Guide [22] and the official Apple developer documentation for the developers.

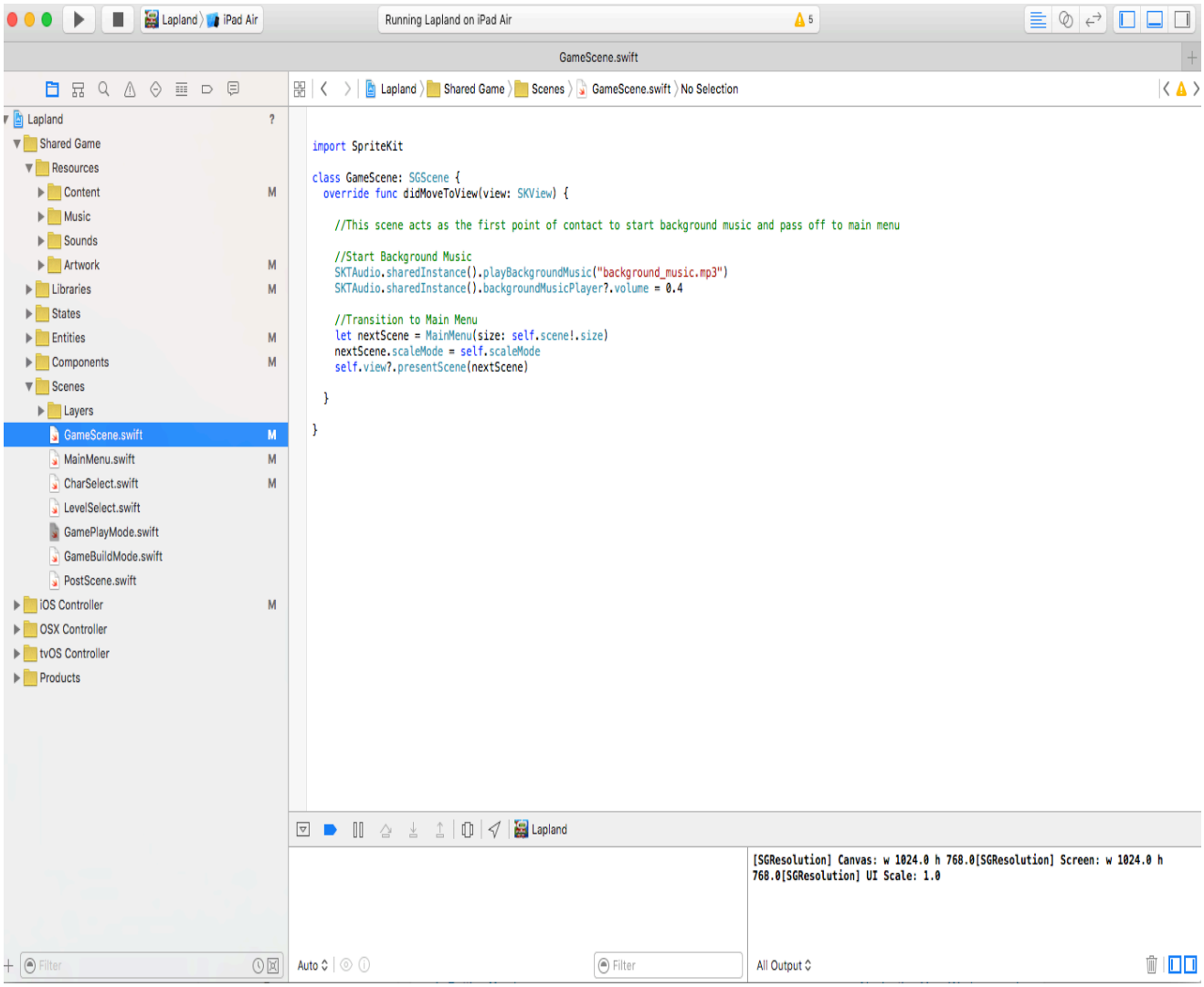

FIGURE 12. Workspace Window Overview in Xcode 7.2

5.2 Scenes in SpriteKit

Scene is a the content that is rendered by SKView objects. The Scene is a root node which runs the action and simulates physics. We need to present it from SKView objects to display a scene. In Figure 5 i.e. Game loop, each frame is processed by scene.

In the game, different scenes are created for each interface. For example, creating a scene for the main menu and other scene for the gameplay. Figure 13 shows the different scenes file in the scene folder.

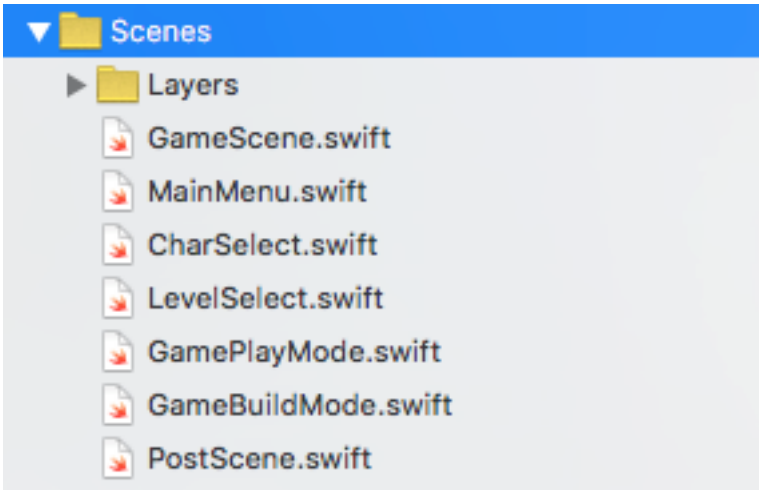

FIGURE 13. Scenes in the project

A transition is used to perform animation while shifting from one scene to another. An object called SKTransition [23] is used to perform this action. Scenes are the basic building blocks of the game so the transitioning from one scene to another is necessary in the game.

For example, the GameScene.swift file of the project looks as the following:

```
import SpriteKit

class GameScene: SGScene {
  override func didMoveToView(view: SKView) {

    //This scene acts as the first point of contact to start background music and pass off to main menu

    //Start Background Music
    SKTAudio.sharedInstance().playBackgroundMusic("background_music.mp3")
    SKTAudio.sharedInstance().backgroundMusicPlayer?.volume = 0.4

    //Transition to Main Menu
    let nextScene = MainMenu(size: self.scene!.size)
    nextScene.scaleMode = self.scaleMode
    self.view?.presentScene(nextScene)

  }

}
```

The MainMenu.swift file begins in the following way:

```
class MainMenu: SGScene {

  //Sounds
  let sndTitleDrop = SKAction.playSoundFileNamed("title_drop.wav", waitForCompletion: false)
  let sndButtonClick = SKAction.playSoundFileNamed("button_click.wav", waitForCompletion: false)

  override func didMoveToView(view: SKView) {

    let background = SKSpriteNode(imageNamed: "BG")
    background.posByCanvas(0.5, y: 0.5)
    background.xScale = 1.2
    background.yScale = 1.2
    background.zPosition = -1
    addChild(background)
```

5.3 Working with Sprites

Sprites are the basic building blocks used to create the majority of our scene's content. Sprites are represented by SKSpriteNode [24] objects. The SKSpriteNode class is a root node class which is used to draw texture images with many customizations.

The following is an example of creating a textured sprite.

```
let background = SKSpriteNode(imageNamed: "BG")
background.posByCanvas(0.5, y: 0.5)
background.xScale = 1.2
background.yScale = 1.2
background.zPosition = -1
addChild(background)
```

5.4 Physics in Sprite Kit

SKPhysicsBody [25] object is used to add physics body in the game. The properties like volume-based, egde-based, force and shape are defined in each class. The following is an example of using a SKPhysicsBody object.

```
class PhysicsComponent: GKComponent {

  var physicsBody = SKPhysicsBody()

  init(entity: GKEntity, bodySize: CGSize, bodyShape: PhysicsBodyShape, rotation: Bool) {

    switch bodyShape {
    case.square:
      physicsBody = SKPhysicsBody(rectangleOfSize: bodySize)
      break
    case.squareOffset:
      physicsBody = SKPhysicsBody(rectangleOfSize: bodySize, center: CGPoint(x: 0, y: bodySize.height/2 + 2))
      break
    case .circle:
      physicsBody = SKPhysicsBody(circleOfRadius: bodySize.width / 2)
      break
    case .topOutline:
      physicsBody = SKPhysicsBody(edgeFromPoint: CGPoint(x: (bodySize.width/2) * -1, y: bodySize.height/2), toPoint: CGPoint(x: bodySize.width/2, y: bodySize.height/2))
      break
    case .bottomOutline:
      physicsBody = SKPhysicsBody(edgeFromPoint: CGPoint(x: (bodySize.width/2) * -1, y: (bodySize.height/2) * -1), toPoint: CGPoint(x: bodySize.width/2, y: (bodySize.height/2) * -1
      break
    }
```

The following is the list of physical properties of the physics body [26]:

- `mass` : It is the mass of the body in kilograms.
- `density` : It is the density of the body in kilograms per square meter. The density and mass properties are interrelated. One property is recalculated every time the other is changed. The default value is 1.0.
- `area` : It is the area covered by the body. This is a read-only property and is used to define the mass of the physics body with the help of the `density` property.
- `friction` : It is used to determine how much friction force should be applied to the other physics body in contact with the current body. This property has a value between 0.0 and 1.0. The default value is 0.2.
- `restitution` : It is used to determine the bounciness of the physics body. This property has a value between 0.0 and 1.0. The default value is 0.2.
- `linearDamping` : It is used to reduce the linear velocity of a physics body. This property has a value between 0.0 and 1.0. The default value is 0.1.
- `angularDamping` : It is used to reduce the angular velocity of a physics body. This property has a value between 0.0 and 1.0. The default value is 0.1.
- `affectedByGravity` : This is a Boolean value. It determines if a physics body will be affected by gravity in the scene. Edge-based physics bodies simply ignore this property as they are not affected by gravity. The default value is `true` .
- `allowsRotation` : This is also a Boolean value. It determines if a physics body will be affected by angular forces and impulses applied to it in the scene. An edge-based physics body simply ignores this property. The default value is `true` .
- `dynamic` : This is a Boolean value too. It determines if a physics body will be moved by the physics simulation in the scene. Edge-based physics bodies simply ignore this property. The default value is `true` .

SpriteKit uses two kinds of interactions between physics bodies, Contact and Collision. Contact is used to find out if two bodies are in contact or not.

Whereas, Collision is used to avoid two objects from hitting each other. The following are the list of collision control properties [27]:

- `categoryBitMask` : This is a mask which defines the category of the physics body. We can have up to 32 different categories. With the help of a category bitmask, you can define which physics bodies should interact with each other. This property is used along with `contactTestBitMark` .
- `collisionBitMask` : This property is used to define the categories of physics bodies which could collide. It is used to determine whether a collision occurs using an AND operation with the other physics body. If the result is a nonzero value, this body will be affected by the collision, otherwise not. This helps you skip collision calculations in case of a minute velocity change.
- `usesPreciseCollisionDetection: Bool` : If `true` , this body will be affected by the collision, otherwise it will pass through the other body in a single frame. A `true` value on either of the bodies will lead to a collision, which means that more computation power will be used by Sprite Kit to detect collisions and perform precise calculations. For very small and fast moving objects, this property can be set to `true` , otherwise the default value is `false` .
- `contactTestBitMask` : This property defines which category a `BitMask` physics body should notify the intersection with the receiving physics body through an AND gate operation. If the value is non-zero, the `SKPhysicsContact` object is created and passed to the physics world delegate.
- `allContactedBodies() -> [AnyObject]` : This is the function which is used to determine if one or more bodies is in contact with the receiving physics body. It simply returns an array of all physics body objects that are in contact with the receiving physics body, that is, the body on which this function is applied.

5.5 Animation and Texture

All the images in the same atlas folder are animated multiple times to make animation. Texture atlases is used to collect all the images together. The SKTexture [28] object is attached with the sprites and loads all the texture data when sprite node is in the scene. Some texture data used in the game is listed in Figure 14.

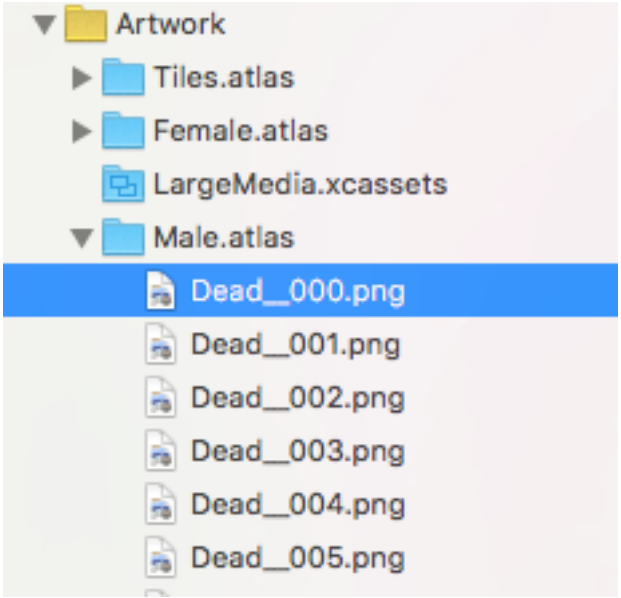

FIGURE 14. Texture atlas in Artwork folder

The following is an example of animating a texture:

```
func loadAnimations(textureAtlas:SKTextureAtlas) -> [AnimationState: Animation] {
  var animations = [AnimationState: Animation]()

  animations[.Jump] = AnimationComponent.animationFromAtlas(textureAtlas,
    withImageIdentifier: AnimationState.Jump.rawValue,
    forAnimationState: .Jump, repeatTexturesForever: false, textureSize: CGSize(width: 23.5, height: 48.0))
  animations[.Run] = AnimationComponent.animationFromAtlas(textureAtlas,
    withImageIdentifier: AnimationState.Run.rawValue,
    forAnimationState: .Run, repeatTexturesForever: true, textureSize: CGSize(width: 37.93, height: 48.0))
  animations[.IdleThrow] = AnimationComponent.animationFromAtlas(textureAtlas,
    withImageIdentifier: AnimationState.IdleThrow.rawValue,
    forAnimationState: .IdleThrow, repeatTexturesForever: false, textureSize: CGSize(width: 40.1, height: 48.0))
```

Also, a parallax scrolling background is added to slower the movement of background images than foreground images. Parallax scrolling is especially used in 2D game to make a smooth dynamic game environment [29].

The following swift class takes care of parallax scrolling:

```
class ParallaxComponent: GKComponent {

  var movementFactor = CGPointZero
  var pointOfOrigin = CGPointZero
  var resetLocation = false

  var spriteComponent: SpriteComponent {
    guard let spriteComponent = entity?.componentForClass(SpriteComponent.self) else { fatalError("SpriteComponent Missing") }
    return spriteComponent
  }

  init(entity: GKEntity, movementFactor factor:CGPoint, spritePosition:CGPoint, reset:Bool) {
    super.init()

    movementFactor = factor
    pointOfOrigin = spritePosition
    resetLocation = reset

  }

  override func updateWithDeltaTime(seconds: NSTimeInterval) {
    super.updateWithDeltaTime(seconds)

    //Move Sprite
    spriteComponent.node.position += CGPoint(x: movementFactor.x, y: movementFactor.y)

    //Check location
    if (spriteComponent.node.position.x <= (spriteComponent.node.size.width * -1)) && resetLocation == true {
      spriteComponent.node.position = CGPoint(x: spriteComponent.node.size.width, y: 0)
    }

    //Add other directions if required.

  }
```

5.6 Controlling the Game

The controls in SpriteKit can be implemented by using methods like Tapping, Gesture recognitions and Moving sprites using the accelerometer. In this game Tapping methods were used. In them the character will jump when the screen is tapped.

There are four override methods for handling touch events with a UIResponder class [30], which is part of UIKit provided by Apple. They are:

- `func touchesBegan(touches:Set<NSObject>, withEvent event:UIEvent)` : This method is called whenever a user touches the view/window
- `func touchesMoved(touches:Set<NSObject>, withEvent event:UIEvent)` : This method is called whenever a user moves his finger on the view/window
- `func touchesEnded(touches:Set<NSObject>, withEvent event:UIEvent)` : This method is called whenever a user removes the finger from view/window
- `func touchesCancelled(touches:Set<NSObject>!, withEvent event:UIEvent!)` : This method is called whenever system events, such as low memory warnings and so on, happen

To implement an action when someone taps on a node on the scene, first the tapped location will be got on the scene. And if the tapped location is within the node's co-ordinates axis points, then the actions for that tap can be defined. This is implemented in the touchesBegan() method. The tapping methods look as the following:

```
override func pressesBegan(presses: Set<UIPress>, withEvent event: UIPressesEvent?) {
  for press in presses {
    switch press.type {
    case .Select:
      control.jumpPressed = true
      break
    case .PlayPause:
      if pauseLoop {
        stateMachine.enterState(GameSceneActiveState.self)
      } else {
        stateMachine.enterState(GameScenePausedState.self)
      }
      break
    default:
      break
    }
  }
}

override func pressesEnded(presses: Set<UIPress>, withEvent event: UIPressesEvent?) {
  control.jumpPressed = false
}

override func pressesCancelled(presses: Set<UIPress>, withEvent event: UIPressesEvent?) {
  control.jumpPressed = false
}

override func pressesChanged(presses: Set<UIPress>, withEvent event: UIPressesEvent?) {
  control.jumpPressed = false
}
```

5.7 GameplayKit

In iOS9, Apple introduced a new framework called GameplayKit. GameplayKit provides tools and technologies for developing complex rule-based games. As Gameplaykit is a high level game engine technology, it can be combined with SpriteKit, SceneKit and other third party game engine too. The following example shows how to import GamePlayKit.

```
import GameplayKit
class GameEntity : GKEntity
```

Figure 15 shows the seven core areas of GameplayKit which can be used in the game.

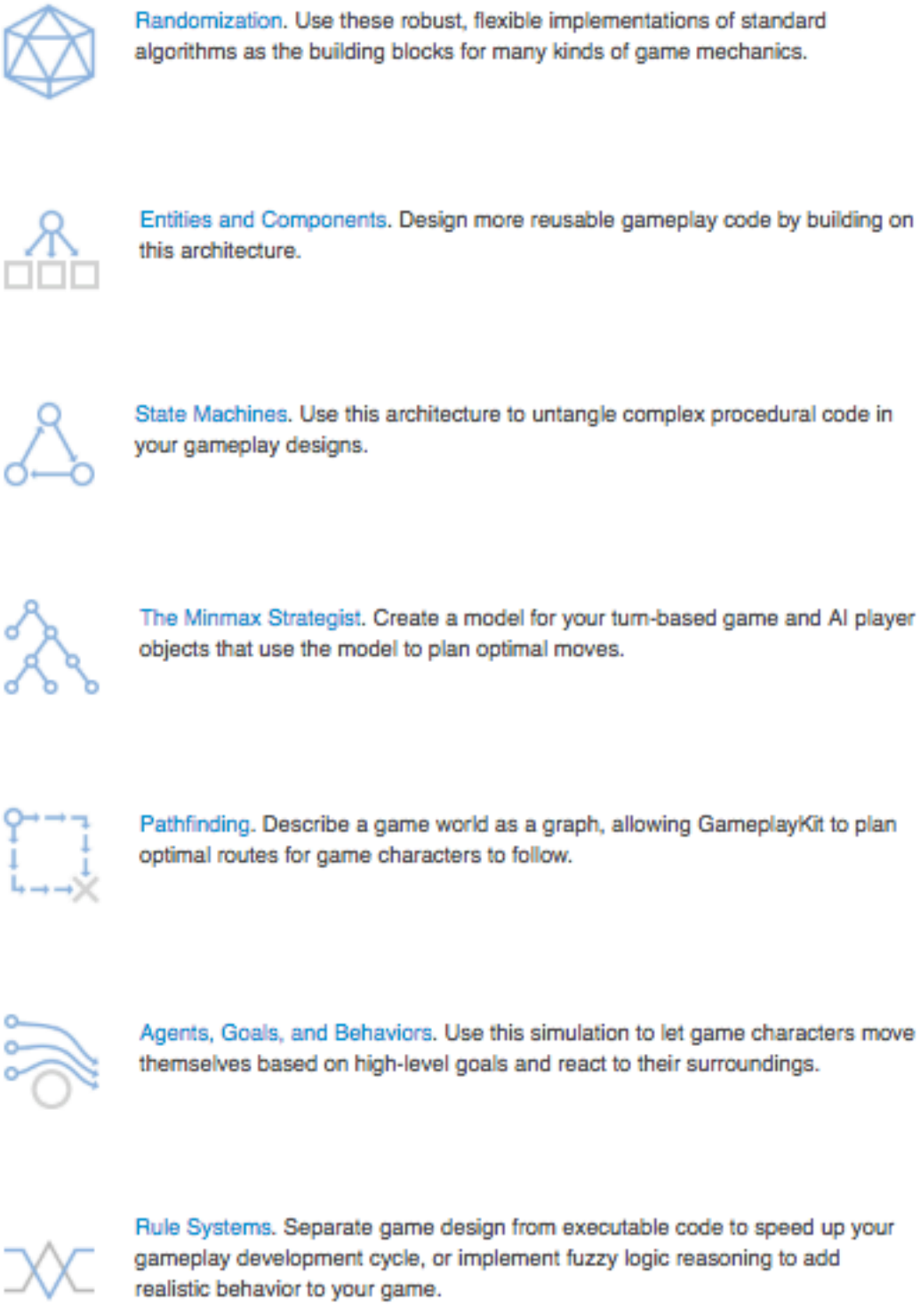

FIGURE 15. Seven core areas of GameplayKit [31]

7 CONCLUSION

The main advantage of SpriteKit is that it is build into iOS. There is no need to download any other third-party libraries or depend on external resources to develop 2D games. It is also written by Apple, so we can be sure that it will be well supported and updated moving forward in future. I think it is the best game engine, especially for beginners who want to develop 2D games. With Sprite Kit, we are going to be locked into the iOS ecosystem. If we want to develop a for cross-platform, Sprite Kit is not the right one.

REFERENCES

1. Oulu Game Lab. Date of retrieval 1.4.2016
   http://www.oulugamelab.net/
2. Android Game Demo. Date of retrieval 20.4.2016
   https://www.youtube.com/watch?v=8e--S3gsCGI
3. Apple Platform. Date of retrieval 3.4.201
   https://developer.apple.com/platforms/
4. iOS Game Development Essentials. By Chuck Gaffney. Date of retrieval 4.4.2016
   https://books.google.fi/books?id=AvWoCwAAQBAJ&pg=PA64&lpg=PA64&dq=ios+game+development+history&source=bl&ots=PqWbWT90vV&sig=4GmPzu4DPfeamn19DcUyJgkxHno&hl=en&sa=X&redir_esc=y#v=onepage&q=ios%20game%20development%20history&f=false
5. Cocos2D VS SpriteKit vs Unity 2D. Date of retrieval 5.4.2016
   https://www.raywenderlich.com/67585/cocos2d-vs-sprite-kit-vs-unity-2d-tech-talk-video
6. Apple Developer Xcode. Date of retrieval 6.4.2016
   https://developer.apple.com/xcode/
7. Developing for iOS. Date of retrieval 6.4.2016
   https://developer.apple.com/ios/
8. Swift. Date of retrieval 6.4.2016
   https://developer.apple.com/swift/
9. Sprite Kit. Date of retrieval 8.4.2016
   https://developer.apple.com/spritekit/
10. Building Your Scene. Date of retrieval 9.4.2016
    https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Nodes/Nodes.html -
    //apple_ref/doc/uid/TP40013043-CH3-SW1
11. SKScene. Date of retrieval 9.4.2016
    https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKScene_Ref/index.html#//apple_ref/occ/cl/SKScene
12. SKNode. Date of retrieval 10.4.2016

https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKNode_Ref/index.html - //apple_ref/occ/cl/SKNode

13. SKAction. Date of retrieval 9.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKAction_Ref/index.html - //apple_ref/occ/cl/SKAction
14. Particle Emitter. Date of retrieval 10.4.2016
https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/xcode_guide-particle_emitter/Introduction/Introduction.html
15. Texture Atlases. Date of retrieval 11.4.2016
https://developer.apple.com/library/ios/recipes/xcode_help-texture_atlas/AboutTextureAtlases/AboutTextureAtlases.html
16. SKShader. Date of retrieval 12.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKShader_Ref/
17. SKLightNode. Date of retrieval 13.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKLightNode_Ref/
18. Simulating Physics. Date of retrieval 14.4.2016
https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Physics/Physics.html#//apple_ref/doc/uid/TP40013043-CH6-SW1
19. Advanced Scene Processing. Date of retrrieval 14.4.2016
https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Actions/Actions.html - //apple_ref/doc/uid/TP40013043-CH4-SW1
20. Game Art 2D. Date of retrieval 14.4.2016
http://www.gameart2d.com/
21. Game Art 2D item license. Date of retrieval 15.4.2016
http://www.gameart2d.com/license.html
22. SpriteKit Programming Guide. Date of retrieval 16.4.2016
https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html
23. SKTransition. Date of retrieval 16.4.2016

https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKTransition_Ref/index.html - //apple_ref/occ/cl/SKTransition

24. SKSpriteNode. Date of retrieval 18.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKSpriteNode_Ref/index.html - //apple_ref/occ/cl/SKSpriteNode
25. SKPhysicsBody. Date of retrieval 19.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKPhysicsBody_Ref/index.html - //apple_ref/occ/cl/SKPhysicsBody
26. Defining a Body's Physical Properties. Date of retrieval 20.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKPhysicsBody_Ref/index.html - //apple_ref/doc/uid/TP40013032-CH1-SW35
27. Working with Collisions and Contacts. Date of retrieval 21.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKPhysicsBody_Ref/index.html - //apple_ref/doc/uid/TP40013032-CH1-SW22
28. SKTexture. Date of retrieval 22.4.2016
https://developer.apple.com/library/ios/documentation/SpriteKit/Reference/SKTexture_Ref/index.html - //apple_ref/occ/cl/SKTexture
29. Parallax Scrolling in SpriteKit. Date of retrieval 23.4.2016
https://digitalleaves.com/blog/2013/12/parallax-scrolling-in-spritekit-made-easy/
30. UIResponder. Date of retrieval 24.4.2016
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIResponder_Class/
31. GameplayKit. Date of retrieval 25.4.2016
https://developer.apple.com/library/ios/documentation/General/Conceptual/GameplayKit_Guide/