Writing A VideoGame with IOS Spritekit - AstroKnot Documentation

Llewellyn Flauta

Masters of Computer Science

Depaul University

June 1, 2017

Writing A SpaceGame with SpriteKit: Abstract/Introduction

Games are one of the cornerstones of Smartphone usage. Being able to develop them is a good strategy if a developer wants to be on one of the cutting edges of mobile-development. I believe that if you have the concepts of making a great game and can mix in the ability to code them, a developer can be a powerhouse for the company that they work for.

I propose to create a Game app that demonstrates the different concepts or design patterns commonly used in game-development. If not one app, maybe a series of mini-apps that showcases the different things that can be done with the SpriteKit Framework. I will try to find things that are unique to swift and Spritekit, or things made more convenient with Spritekit. I think once the app and paper are completed, it will be a nice reference manual and showcase for a portfolio. The app and paper will show a knowledge of programming best-practices, a knowledge of the application of design-patterns, as well as showcasing the ability to create a good game.

I also planned to execute the tenets of Agile Development methodologies in the development. Each Phase will be a sprint, I think it's a little more important to deliver on each phase, so I will probably stop most of the work on a Phase once the time is up. If time allows, I will go back and modify the code. Agile Targets will be stated at the beginning of each Phase, and they will be crossed-off the list as they are achieved. It is just me working on the project, so I don't think i really need

Purpose or Hypothesis - Requirements Documentation

The purpose or hypothesis of the research. This is a more general overview of your research.

Requirements of App

- Name of App: AstroKnot
- Overview - a suite of apps rolled into one. Ideally I would create 5 or 6 Games based on Arcade Classics. Example: Pong, Asteroids, SpaceInvaders, 1941.
    - Pong - could show some early physics in games
    - Asteroids - begins to show more advanced collision detection
    - Shooters - show pathing and movement of enemies in games.
    - BossBattle - Particle Effects and Physics

Method of Approach/Work Plan/Agile

I will take an approach of Coding daily and then logging the results nightly. I will use Agile Techniques where possible to achieve my goals. Here are some of the phases that will take place in my project:

- Brainstorming
- Research possibilities
- Agile Planning
- Start Coding!
- Code Phases 1- 4 with approximately one week allotted for each phase
- Final week for write up of paper

SPECIAL CONSIDERATIONS

- I know I personally tend to work well with a 2 hour heavy coding/learning, and then take a light break or light-studying for an hour or so.
- I will start coding on a set time every day with a set schedule, this will eliminate indecision.
- I will take a sandbox approach to coding in the beginning, but will go toward requirements so that forward progress is always made.
- I will make sure to use source-control for my project….commit often on github.
- Toward the end of the project, I may run out of time in implementing all the effects into gameplay, I may need to just show some proof of concept

AGILE PROCESS

I am the only person working on the project, so many things from Agile probably won't make sense to use. I will use Stories though and weigh them according to their difficulty/complexity in order to have a gauge of my work velocity. During each Phase in this paper, I will list all the stories i need fulfilled. I will assess the tasks based on priority for us to narrow down what I should be working on, as well as the amount of work needed to get to completion

CODE

I will try to make my code self-documenting by using good names for objects and functions. I will also remark where possible, but since there is a companion research paper, I will explain things more in-depth in the paper compared to relying on remark statements in code.

Phase 0 Research

In the beginning, I didn't know what was really possible for me to code up. Especially within about 4-5 weeks. Of course anything is possible, but I was trying to find a good fit for a nice game, while at the same time showing some Good Programming Practices. As a spoiler, I think I succeeded in the former, but not so much in the latter. Eventually one of our homework assignments was to make a game that used the accelerometer as a control mechanism. This turned out a really nice Application compared to what I would have done hand-calculating every shape, and collision detection item. With this foundation of building blocks I knew I had the makings of multiple games.

AGILE DEVELOPMENT

So, as stated in earlier portions of this document, I will try to create a series of games out of the baseline of an asteroids type shooter game. I will try to hit the various features of game development. In this stage I laid out the general stories for each section in the game. I weighed out how much each task should take. Based on actual time available, the work will be divided into 3 day sprints. At about 5 hours of coding a day each section should take about 10-15 hours of codetime including research. Code time should end up at about 50+ hours which is about double what I spent on our previous homework assignments.

THE UPDATE LOOP

```
    override func update(_ currentTime: TimeInterval)
{        //this function smoothes out the movment updates
        if lastUpdateTime > 0 {
            dt = currentTime - lastUpdateTime
        } else {
            dt = 0
        }
        lastUpdateTime = currentTime
        //move the ship
        move(sprite: hero, velocity: velocity)
        boundsCheckHero()
        //rotate the ship in direction of the move on
touches
        rotate(sprite: hero, direction: velocity)
        checkCollisions()
        //print (hero.position)
        moveConga()}
```

The update loop is one of the foundations of game development. This is where the magic happens. The update loop can be likened to the main thread of the app. For this app in the beginning, it checks what it should 'move'…the spaceship. Then it does a bounds check on the spaceship to see if it hit any boundaries of the playfield, if a boundary is hit, the ship is reflected into an opposite direction. After that it rotates the ship toward the correct angle of direction.

Phase 1 Sprites/Animation (GameScene.swift)

AGILE GOALS

- ~~I want to put a sprite on the screen - 1~~
- ~~setup the framework/underlying foundation of the game-8~~
- ~~I want to manipulate the sprite in different ways to show what they can do - 1~~

CREATING AND MANIPULATING A SPRITE

A sprite node is an object that spritekit is programmed to manipulate. Here is the snippet of code for creating the astronaut in GameScene.swift. astronaut.run(SKAction.sequence[…. takes an array of actions, configured by the let statements, and strings them into a sort of macro of manipulations. Some of the different actions that can be taken are movement, fade-in/out, colorize, pause, reverse, scale-up/down, resize.

Sprites are basically the building blocks of games. Spritekit allows for these manipulations without having to self-hardcode a lot of functions needed for making games. Meanwhile, there is still a lot of customization possible, so that programmatically many things are possible.

```
    func spawnAstronaut() {
        let astronaut = SKSpriteNode(imageNamed:
"astronaut")
        astronaut.name = "astronaut"
        astronaut.position = CGPoint(
            x: CGFloat.random(
                min: playableRect.minY +
obstacle.size.height/2,
                max: playableRect.maxY -
obstacle.size.height/2),
            y: CGFloat.random(
                min: playableRect.minY +
obstacle.size.height/2,
                max: playableRect.maxY -
obstacle.size.height/2))
        //astronaut.setScale(2.0)
        addChild(astronaut)

        let actionMove = SKAction.moveBy(x: CGFloat.random(
            min: playableRect.minX + obstacle.size.height/
2,
            max: playableRect.maxX - obstacle.size.height/
2), y:CGFloat.random(min: playableRect.minY +
obstacle.size.height/2,

max: playableRect.maxY - obstacle.size.height/2), duration:
2.0)

        let actionRotate = SKAction.rotate(byAngle: 1.0,
duration: 5.0)
        let actionReverseRotate = SKAction.rotate(byAngle:
-1.0, duration: 5.0)
        let actionRemove = SKAction.removeFromParent()
        let wait = SKAction.wait(forDuration: 1.0)

        astronaut.run(SKAction.sequence([actionRotate,
wait,actionReverseRotate, actionRemove]))

    }
```

ANIMATION

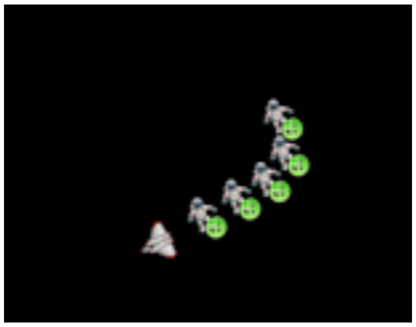
One of the things that is possible with a sprite is to animate it, unfortunately, I couldn't find an appropriate set of images that matched my app. By attaching 4 or 5 different images into a sprite and showing them in sequence, the illusion of movement is formed.

ADVANCED SPRITE ACTIONS

Sprites can be combined so that one Sprite consists of a multitude of sprites. Then each point where it attaches can be moved and rotated so as to simulate movement within the sprite. Another example of htis is when there is a Role-Playing-Character that can equip different weapons by attaching a new sprite to his arm. Sprite borders can also be used to calculate something called a PhysicsBody, which allows the sprite to be affected by such things as game-gravity or bumping into other phsyics bodies. Another thing that can be done is the shape of the sprite can be customized. On a large scale the sprite could be formed into a circle so that the corners of the sprite where there is no image will not trigger events. On a more advanced scale, a sprite can be custom mapped to the shape of an image.

Phase 2 Movement (GameScene1.swift)

AGILE GOALS
~~I want to add a hero-spaceship to the scene. 2~~
~~I want to be able to have the ship move toward the direction that I touch on the screen. 4~~
~~I want the ship to rotate towards the direction that the ship is moving 2~~
~~If the ship hits a border of the playing field I want it to bounce off like a billiards ball. 4~~
~~I want to have bonus items that spawn. 4~~
~~I want to be able to pick up the bonus items so that they follow my ship in a conga line. 6~~



ACCELEROMETER CONTROL

I do not use accelerometer control in this app, but I did use it in the past similar homework assignment. In this code snippet I gave the hero (the Spaceship) a physics body, which is basically a sprite that becomes affected by in-game virtual physics, things like gravity, hitting objects, etc. motionManager is initialized, and motionManager uses the accelerometer data to move the ship toward a point depending on how far tilted the device is. The code is actually remarked out in GameScene.swift. By giving the spaceship a physics body

MOVEMENT

This is the core of movement functions in the beginning phases of the app. It uses Vectors and Points to calculate the direction of movement. A MathUtils Swift file was made to store common overloaded Vector calculations for movement

```
//MARK: MOVEMENT
```

```
    func move (sprite: SKSpriteNode, velocity: CGPoint) {

        let amountToMove = CGPoint(x: velocity.x * 
CGFloat(dt),
                                   y: velocity.y * 
CGFloat(dt))
        //print("Amount to move: \(amountToMove)")
        sprite.position += amountToMove
    }

    func MoveHeroToward(location: CGPoint) {
        let offset = location - hero.position
        let direction = offset.normalized()
        velocity = direction * heroMovePointsPerSec
        //physicsWorld.gravity =  CGVector (dx: 0, dy: 0)
        //use some type of friction here
    }

    func moveConga(){
        var targetPosition = hero.position

        enumerateChildNodes(withName: "conga") { node, stop 
in
            if !node.hasActions() {
                let actionDuration = 0.3
                let offset = targetPosition - node.position
                let direction = offset.normalized()
                let amountToMovePerSec = direction * 
self.movePointsPerSecond
                let amountToMove = amountToMovePerSec * 
CGFloat(actionDuration)
                let moveAction = SKAction.moveBy(x: 
amountToMove.x, y: amountToMove.y, duration: 
actionDuration)
                node.run(moveAction)
            }
            targetPosition = node.position
        }
    }
```

BOUNDS CHECK
This function is called from the update function. It checks if the hero hits one of the

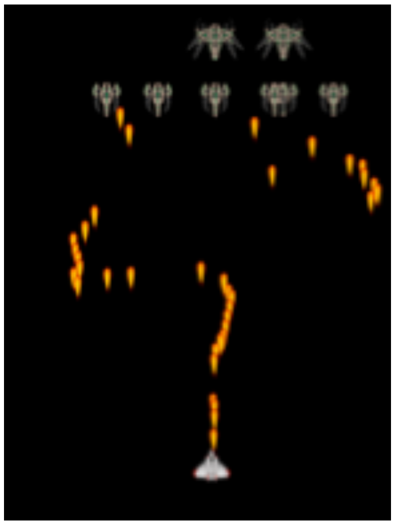
boundaries of the play-field. If a boundary is hit, the velocity of the hero is reversed including the angle of the spaceship as it bounces off into the opposite direction

```
    func boundsCheckHero() {}    func boundsCheckHero() {
        let bottomLeft = CGPoint(x: 0, y: 
playableRect.minY)
        let topRight = CGPoint(x: size.width, y: 
playableRect.maxY)

        //reverses the velocity of the hero when a bound is 
hit

        if hero.position.x <= bottomLeft.x {
            hero.position.x = bottomLeft.x
            velocity.x = -velocity.x
        }
        if hero.position.x >= topRight.x {
            hero.position.x = topRight.x
            velocity.x = -velocity.x
        }
        if hero.position.y <= bottomLeft.y {
            hero.position.y = bottomLeft.y
            velocity.y = -velocity.y
        }
        if hero.position.y >= topRight.y {
            hero.position.y = topRight.y
            velocity.y = -velocity.y
        }
    }
```

Phase 3 Collision Detection (GameScene2.swift)

AGILE Goals

- ~~I want to change the movement style of my ship to a galaga-style where the ship is bound to the bottom portion of the screen - 2~~
- ~~I want the ship to always point upwards instead of rotating - 1~~
- ~~I want to spawn enemy ships - 4~~
- ~~I want the enemy ships to move in different directions -8~~
- ~~I want be able to fire weapons at the enemy ships -4~~
- I want to be able to destroy the enemy ships on a collision - 8 (note below)



COLLISION DETECTION

checkCollisions() is called from the Update function. It enumerates the astronaut sprites and checks each one against the position of the spaceship. For this part of the code, once an astronaut is hit, it changes it's label to "conga" where a conga function attaches it to the spaceship, leaving a trail of astronauts in the ship's wake.

```
    func heroHit(obstacle: SKSpriteNode){
        obstacle.removeFromParent()
    }

    func heroHitAstronaut(astronaut: SKSpriteNode){
        //print ("hero hit astronaut")
        astronaut.name = "conga"
        astronaut.removeAllActions()
        astronaut.zRotation = 0

        /*        astronautsSaved = astronautsSaved + 1
         if (astronautsSaved > 7) {
         gameOver = true
         print ("you win!")
         let gameOverScene = GameOver(size: size, won:
true)
         gameOverScene.scaleMode = scaleMode
         let reveal =
SKTransition.flipHorizontal(withDuration: 1.5)
         view?.presentScene(gameOverScene, transition:
reveal)

         }*/
        //astronaut.removeFromParent()
    }

    func heroHitObstacle(){
        //print ("you lose!")
        //let gameOverScene = GameOver(size: size, won:
false)
        //gameOverScene.scaleMode = scaleMode
        //let reveal =
SKTransition.flipHorizontal(withDuration: 1.5)
        //view?.presentScene(gameOverScene, transition:
reveal)
    }

    func checkCollisions(){
        //var hitObstacles: [SKSpriteNode] = []
        var hitAstronaut: [SKSpriteNode] = []

        /*enumerateChildNodes(withName: "obstacle"){node, _
in
         let obstacle = node as! SKSpriteNode
         if node.frame.insetBy(dx: 20, dy:
30).intersects(self.hero.frame){
         hitObstacles.append(obstacle)
         }
         }

         for obstacle in hitObstacles {
         heroHitObstacle()


         } */

        enumerateChildNodes(withName: "astronaut"){node, _
in
            let astronaut = node as! SKSpriteNode
            if node.frame.insetBy(dx: 20, dy:
30).intersects(self.hero.frame){
                hitAstronaut.append(astronaut)
            }
        }

        for astronaut in hitAstronaut {
            //print ("hit astronaut")
            heroHitAstronaut(astronaut: astronaut)
        }
    }
```

SHOOT ENERGY BOLTS/I WANT TO BE ABLE TO DESTROY ENEMY SHIPS ON A COLLISION DETECTION

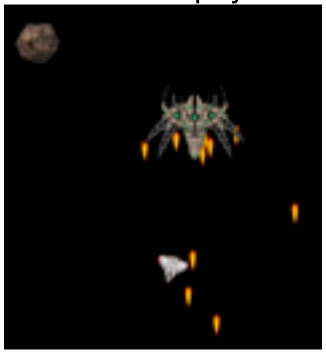
I found a source of how I could implement shooting firebolts at the enemies. I tried implementing the above type of function for collision detection, and I found that performance slowed to an utter crawl making the game unplayable. I believe it's because of enumerating every energy bolt through every enemy ship results in an exponential runtime and the system cannot handle it. I believe the way to do this is to utilize Physics bodies which I will experiment with in the next Phase

```
    //MARK: TOUCH EVENTS
    func sceneTouched (touchLocation:CGPoint){
        MoveHeroToward(location: touchLocation)
        spawnEnergyBolt(touchLocation: touchLocation)
    }

    func spawnEnergyBolt (touchLocation: CGPoint){
        let energyBolt = SKSpriteNode(imageNamed:
"energyBolt")
        energyBolt.position = hero.position
        let offset = touchLocation - energyBolt.position
        addChild(energyBolt)
        let direction = offset.normalized()
        let shootAmount = direction * 2000
        let realDest = shootAmount + energyBolt.position
        let actionMove = SKAction.move(to:realDest,
duration: 2.0)
        let actionMoveDone = SKAction.removeFromParent()

energyBolt.run(SKAction.sequence([actionMove,actionMoveDone
]))
    }
```

Phase 4 Physics/Advanced Nodes Boss (GameScene3.swift)

AGILE Goals

- ~~I want to spawn a boss enemy ship 2~~
- I want the ship to take up most of the screen and to have multiple node points for weapon attachments 8 (didn't really have time or the asset files to do this)
- ~~I want physics to affect the gameplay 16~~



I thought it was going to be difficult, but it was pretty easy to add physics bodies to the sprites and then alter the properties. I was changing too many things at once in my own app though. It probably would have been better to go in with a strong idea of how I wanted the physics to work, and then code the paramaters until they complied.

```
    override func didMove(to view: SKView) {

        //initialize objects, and start looping actions. 
References are weak so that when the child objects are 
removed the memory is freed
        backgroundColor = SKColor.black
        physicsWorld.gravity = CGVector(dx: 0.0, dy: -5.0)
        //setupCoreMotion()
        let borderBody = SKPhysicsBody(edgeLoopFrom: 
self.frame)
        borderBody.friction = 0
        self.physicsBody = borderBody

        spawnHero()

        nextIcon.setScale(0.5)
        nextIcon.position = CGPoint(x:(size.width - 100 ), 
y:(size.height - 100))
        addChild(nextIcon)

        let actionSpawnEnemy = 
SKAction.sequence([SKAction.run() { [weak self] in
            self?.spawnEnemy(type: "spike", x: 1200, y: 
2000)
            },SKAction.wait(forDuration: 0.5)])
        run(SKAction.repeat(actionSpawnEnemy, count: 1))

        let actionSpawnSpike =
SKAction.sequence([SKAction.run() { [weak self] in
            self?.spawnEnemy(type: "obstacle", x: 400, y:
1900)
            },SKAction.wait(forDuration: 1.5)])
        run(SKAction.repeat(actionSpawnSpike, count: 2))
    }


    func spawnHero(){
        //create spaceship, give spaceship a physicsBody
        hero = SKSpriteNode(imageNamed: "Spaceship")

        hero.physicsBody = SKPhysicsBody(circleOfRadius:
hero.size.width / 2)
        hero.physicsBody!.allowsRotation = true
        //hero.physicsBody!.linearDamping = 0.0
        hero.physicsBody!.affectedByGravity = false
        hero.position = CGPoint (x: 150, y: 150)
        hero.physicsBody?.friction = 1.0
        hero.physicsBody?.restitution = 1

        hero.physicsBody?.angularDamping = 0
        //        hero.position = CGPoint(x: 96, y: 0)
        hero.setScale(0.33)
        //hero.zRotation = CGFloat(M_PI/Double(2.0))
        hero.name = "hero"
        addChild(hero)
    }


    func spawnEnemy(type: String,x: CGFloat, y: CGFloat) {
        print ("spawn enemies")

        var enemy = SKSpriteNode(imageNamed: type)
        enemy.name = type
        enemy.setScale(1.0)
        enemy.physicsBody = SKPhysicsBody(circleOfRadius:
hero.size.width / 2)
        enemy.physicsBody!.allowsRotation = true
        enemy.physicsBody!.linearDamping = 0.0
        enemy.physicsBody!.restitution = 1.0
        enemy.physicsBody!.affectedByGravity = false
        enemy.physicsBody!.mass = 0.00000001
        enemy.position = CGPoint(x: x,y: y)
        enemy.physicsBody?.friction = 0.0

        if type == "spike" {
            enemy.position = CGPoint(x:400,y:y)
            enemy.physicsBody?.affectedByGravity = true
            enemy.setScale(0.5)
        }

        //if type == "obstacle" {
            enemy.physicsBody!.linearDamping = 0.0
            enemy.physicsBody?.friction = 0.0
            enemy.physicsBody?.linearDamping = 0.1
            enemy.physicsBody?.angularDamping = 0
      // }

        addChild(enemy)
```

Conclusions

One of the things I wanted to accomplish in this app was to use sound Object Oriented Design Patterns in the creation and management of the game. The problem I ran into however, is that I'm not quite comfortable with how things work on their own. Without the fundamental knowledge of how everything interacts once everything is done, it was an impediment to try to design modular code that was elegant and could be reused. In the end, I went to seat-of-pants coding with a big reliance on source-control. It may not have been the best practice, but I did learn a lot. I am sure that in the next iteration of the App or in future versions I will be able to compile more things together as well as design things better out of the box.

I didn't utilize as many of the specific topics taught in our class, but working on the projects in quick-succession really impressed upon me the work needed to get these apps up and running. A lot of research has to be done, and it helps to prototype a bunch of small apps, or to modify completed related apps in order to learn how everything works together.

References

- Rey Wunderlich Tutorial https://www.raywenderlich.com/145318/spritekit-swift-3-tutorial-beginners
- Vector Math http://www.mathsisfun.com/algebra/vectors.html
- Classic Game Design: From Pong to Pac-Man with Unity by Franz Lanzinger
- Swift Design Patterns by CodeWiz RDZ
- Game Programming Patterns - Robert Nystrom
- Level Up! The Guide to Great Video Game Design by Scott Rogers
- Game Project Completed: How Indie Game Developers Finish Their Projects by Thomas Schwarzl