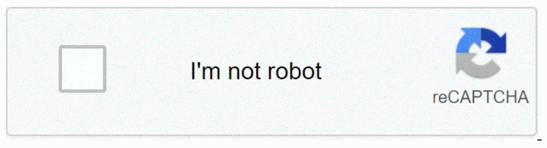

Continue

Top empire building games android

This article first appeared in issue 223 of .net magazine - the world's best-selling magazine for web designers and developers. Taking a look at some of today's top mobile games, it's easy to see that physics-based games are more popular than ever. And it turns out that Corona SDK happens to be the perfect tool for producing these hits, as games like Bubble Ball and Blast Monkeys have already shown, earning the top spot in the iTunes App Store and Android Marketplace in 2011. Corona SDK features high-performance graphics and includes Box2D physics built in, which is why it's perfect for creating these incredibly popular physics-based titles, or a 2D game for that matter. Corona is without a doubt the easiest, fastest way to produce high-quality games and apps for both iOS and Android mobile platforms. In this tutorial we investigate how it works by letting you run through the creation of a simple 2D physics-based game. Make sure you download all the tutorial files and look at the source code as we go, because not everything will be shown here. We have quite a bit of ground to cover, but at the end of it all, you'll probably be surprised at how quickly we were able to create the beginning of what would be the next mobile hit! Get started Before we go ahead, you'll need to create a free Corona Developer account and download the SDK (compatible with both Windows and Mac). Once you've downloaded it, I recommend taking a look at the Quick-Start Guide to familiarize yourself with Corona basics, and to set up your development environment as well. Dino Bounce is a fun, 2D physics-based game that makes gameplay reminiscent of some best selling App Store hits Once you've finished getting things set up, let's dive right into the main concept of our game. Here is a brief description of how it will work: the game begins with a character, platform and target on the screen, with the physics simulation interrupted. The player then drags and positions the platform to the location and rotation of his choice and presses the play button to start the physics simulation. If the platform is positioned correctly, the character will fall, roll and eventually reach the target (level complete). If the player is unsuccessful (the character does not reach the target), the player must try again. Stars are collected optionally for a higher score. In this game there will be very few objects that we will work with to include: a background, the character, a platform, a goal, some stars, the ground and two buttons (stop and play). Now that we have all the preparatory work out of the way, let's start putting everything together. Create a projectWith Corona you use the same project (code, images, etc.) and build the button with the click of a button iOS or Android platforms All that's needed in a Corona project is a single text file folder (main.lua). However, if you have looked at the source code for our project, you will find that, apart from the picture folder, you will find that you are two other files apart from the main.lua: config.lua and build.settings. These two files are only needed if you want to deviate from the default settings when it comes to things like orientation, content scaling, and so on. For a more in-depth explanation of config.lua and build.settings, see the documentation of configuration options. For this project we focus on the iPhone (320x480 resolution) in landscape orientation. However, we'll also want to take advantage of the super-sharp 640x960 screens of newer iPhones (and also sharp Android screens), and therefore - in case you haven't noticed - there are duplicate elements in the image folder, with one double the size with a @2x suffix (specified in config.lua). When you open the main module.lua module from the source files, you immediately see that this is where all the code for our game goes. Lines 14-35 are external libraries (such as widget, line 17 and physics, line 20), some initial settings and local variables used in our game. For our first object, we start by creating the sky, which will serve as the background of the game (lines 38-40).local bg = display.newImageRect(images/background.jpg, 480, 320)bg:setReferencePoint(display. TopLeftReferencePoint)bg.x, bg.y = 0, 0Dank to Corona's simple API, will probably tell you what's going on just by reading the code, but I'll walk you through it anyway. The first line creates a new display object of an image (background.jpg), which is 480x320 (pixels) in size. The second line sets the object's reference point (the point at which an object is placed and rotated) in the upper-left corner. Ghosts Vs Monsters is a full-featured, open-source Angry Birds clone created in just 36 hours with the Corona SDK Finally we do the actual positioning on the last line by changing the object x and y properties. Next, we create a flag (target), the ground, a platform and the drawing objects in the same way:local flag = display.newImageRect(images/flag.png, 50, 102)flag.x, flag.y = 420, 200local ground = display.newImageRect(images/ground.png, 292, 110)ground:setRePointference(display. BottomRightReferencePoint (ground.x, ground.y = display.contentWidth, display.contentHeightlocal platform = display.newImageRect.png, 160, 24)platform.x, platform.y = 260, 75kar = display.newImageRect(images/critter.png, 80, 80)character.x, character.y = 75, 80You may have noticed that the character creation doesn't start with local like the rest of the objects. This is due to the fact that we have already declared it as local on line 35, so that other features (above our character creation code) can easily access the object. This is known as a forward declaration in Lua. Another new thing you might notice is the use of display.contentWidth and These are simply the current width and height of the screen (as specified in in or the standard width/height of the device, if not specified in config.lua), taking into account the orientation of the device. If you were to load the project into the Corona Simulator at this point, you'll see that our project is already starting to come together – pretty easy so far, right? You have to see our character in the sky, a platform, the ground and the target, all placed on the sky background. Adding physics Now that we have all our objects in place, we need to add physics to it. At the top of the head.lua, you will notice that we have already prepared the physics library for use under the physics namespace (main.lua, line 20). By default, no objects respond to the physical simulation when it starts. In order for one of our objects to exist in the physical 'world' in a Corona app, we need to give them a body. This is done with the physics.addBody() function. Let's start by adding a physical body to the flag object, to give you an idea of how it is achieved (main.lua, line 47):p hysics.addBody(flag, static, { isSensor = where })The above rule will add a body to our flag object. The second argument indicates the type of body (dynamic, static or kinematic) that our object has. Since the flag is a fixed object, we set it to static. The feature isSensor allows other objects to pass through the flag, but it will still receive collision events so we can detect when something touches it. In short, this will behave the flag as a non-physical object, but still allows us to detect when the character reaches the target. Next, I'm about adding a body to the character, because it's not a fixed object (main.lua, line 136).physics.addBody(character, { friction = 0.3, bounce =0.5, radius = radius 30 })Since the standard body type in Corona is dynamic, we don't need to specify it in this call to physics.addBody(). If you continue with the properties table that is now passed as a second argument, the friction trait determines a body's behavior when it moves along another physical object (such as the ground). The bounce property determines the bounciness of an object when it comes into contact with another physical object. Finally, the radius property ensures that this is a circular housing and corresponds to the pixel radius of the object on the screen. Now that you understand how to add physical bodies to display objects, the rest of the body-creation code in main.lua should be easy to understand. For a more in-depth explanation of the physics bodies in Corona, see the Physics Bodies documentation. Ansca mobile anscamobile.com is a one-stop resource for Corona users to communicate with other developers through the forums, reference APIs and even share code with others Touches and event uses what is known as event-driven programming. Events occur based on numerous different factors, but are usually a result of user input. With our target platform being iOS and Android touch-based mobile devices, the user touching the screen corresponds to a touch event. Corona allows you to these events per object and then respond accordingly. Since the entirety of our gameplay consists of the player dragging a platform to the right location, we need to assign a touch event listener to our platform object. An event listener is simply a function that runs when a specific event is detected. Here's what our touch listener for our platform object looks like (main.lua, line 89):function platform:touch(event) if not isPlaying then as event.phase == then - finger just touched display.getCurrentStage(): setFocus(self) self.isFocus = true self.markX, self.markY = self.x, self.yelseif self.isFocus dan local moveX = math.abs(event.x - event.xStart) local moveY = math.abs(event.y - event.yStart) local moveThresh = 10 if event.phase == then moved -- finger drags object -- sleep object self.x = event.x - event.xStart + self.markX self.y = event.y - event.yStart + self.markY elseif event.phase == ended then -- finger is released -- object rotate if not dragged past threshold as moveX < moveThresh or moveY < moveThresh dan self.rotation = self.rotation + 10 end -- allows touches to display other display objects.getCurrentStage():setFocus(nihil) self.isFocus = false end return trueendplatform:addEventListener(touch, platform)At the beginning of the touch (the initial stage), we place the app's focus on this object and also mark the current location. When the object is dragged, we change the location when the player's finger moves. Once the player's finger is lifted, we determine whether the object is dragged past the rotation threshold, and let the object respond accordingly. The last line assigns the listener to the platform object, and from that moment on, listening to player will become more player-like. Looking at your game in Debug or Hybrid physics mode you will see exactly how bodies are attracted to help you make the necessary adjustments Adding the latest detailsNow we need to add a collision listener to the flag object to detect when the character of the player is touching (main.lua, line 50). Collision listeners behave in the same way that listeners are touched, but act instead when two physical objects collide. In our game, we will notify the user that the level is complete and restart the game when the character object collides with the flag. For more in-depth information about collision listeners, see collision detection documentation. While we're at it, let's add a star to the level so that the player can earn extra points when the character collects it. To do this make and position we just add the star object, add a body, and a collision listener that will remove the star and increase the player's score upon collision with it Finally, we need to create two widget buttons for Play and Stop. The player presses play when they think the platform is positioned correctly, and once the simulation is active, they can touch the stop button at any time to try again in case they in leading the character to the goal. If you take the project and run it in the Corona Simulator, you'll see that we have a pretty completely small physics-based game. Of course, this game really only serves as a level, but it's enough to show you how fast and easy it is to create fun, visually-pleasing games using the Corona SDK. If we were to add more levels and obstacles, it's easy to see that this little creation could potentially have what it takes to stand up to the big guys in the App Store. If you feel inspired to create your own mobile games and apps for iOS or Android, go to Ansca to sign up for a developer account, download the Corona SDK, read more tutorials, and connect with numerous friendly developers willing to help you with any issues you may have, no matter what your experience level is at the moment. Is.