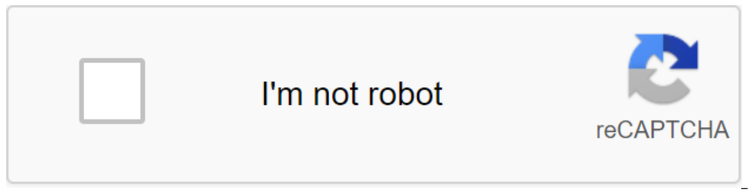
# City building games 2020 android




Continue

Creating a fully working game for Android is much easier than you think. The key to successful Android development - or any development - is to know what you want to achieve and find the tools and skills you need to do so. If you follow the path of least resistance, and keep in mind a clear goal. When it comes to creating games, the best tool in my opinion is unity. Yes, you can make the game in Android Studio, but if you're not experienced with Java and Android SDK it will be an uphill struggle. You need to understand what classes are doing. You will need to use custom views. You will rely on some additional libraries. The list can be sing. Unity is a highly professional tool that powers the vast majority of the best-selling titles on the Play Store.Unity, on the other hand, does most of the work for you. It's a game engine, which means that all the physics and many other features that you can use are already taken care of. It's a cross-platform and it's designed to be very rookie friendly for hobbyists and indie developers. At the same time, Unity is a highly professional tool that feeds the vast majority of the best-selling titles on the Play Store. There are no restrictions and there is no good reason to make life harder for yourself. It's free too! To demonstrate how easy it is to develop a game with Unity, I'm going to show you how to make your first Android game in just 7 minutes. No, I'm not going to explain how to do it in seven minutes. I'll do it in seven minutes. If you follow along too, you'll be able to do the same! Disclaimer: Before we started, I just want to point out that I'm a bit of a hoax. While the process of creating the game will take 7 minutes, it assumes that you have already installed Unity and got everything set up. But I won't leave you hanging: you can find a complete tutorial on how to do it over on Android Authority.Adding sprites and physicsstart, twice clicking on unity to run it. Even the longest journey begins with one step. Now create a new project and make sure you choose '2D'. Once you're in, you'll be met with several different windows. They do things. We don't have time to explain, so just follow my instructions and you'll pick it up as we go. The first thing you want to do is create a sprite to be your character. The easiest way to do this is to draw a square. We'll give him a couple of eyes. If you want to be even faster yet, you can just grab a sprite you like from somewhere. Save this sprite and then just drag and throw it into the scene by placing it in the biggest window. You'll notice that it also pops up on the left in the hierarchy. Now we want to create several platforms. Again, we do with a simple square, and we'll be able to change this free hand to the walls, platforms and You have. Come on, it's beautiful. Drop it just like you. We already have something that looks like a game. Click the play button and you should see the static scene at the moment. We can change this by clicking on our sprite player and looking at the right box called Inspector. Here we change properties for our GameObjects.Choose 'Add Component' and then choose Physics 2D zgt; RigidBody2D. You just added physics to your player! It would be incredibly difficult for us to do on our own and really highlights the usefulness of Unity.We also want to correct our orientation to prevent the character spinning and freewheeling around. Find limitations in the inspector with the selected player and mark the field to freeze the rotation. Now click the play button again and you have to find your player now falling from the sky on his endless doom. For a moment to reflect on how easy it was: just by using this script called 'RigidBody2D', we have fully functional physics. If we applied the same scenario to the round form, it would also roll and even bounce. Imagine coding what you yourself are and how involved that would be! To stop our character falling through the floor, you need to add a collider. It's basically a solid shape. To apply this, select the player, click add the component and this time select Physics 2D zgt; BoxCollider2D. For a moment to reflect on how easy it was: just by using this script called 'RigidBody2D', we have fully functional physics. Do the same with the platform, click the play button and then your character has to fall to solid ground. It's easy! Another thing: to make sure that the camera follows our player whether they are falling or moving, we want to drag the object of the camera that is in the scene (this was created when you started the new project) on top of the player. Now in the hierarchy (GameObjects list on the left) you're going to drag the camera so that it backs down under the player. The camera is now a child of Player GameObject, which means that when the player moves, so will the camera. Your first scenario We're going to make a basic endless runner, which means our character has to move right across the screen until they hit the obstacle. To do this, we need a script. So click the right button in the Assets folder at the bottom and create a new folder called Scripts. Now click again and select Create a C Script. Call it PlayerControls. For the most part, the scripts we create will determine the specific behavior for our GameObjects.Now double-click on your new script, and it will open in Visual Studio if you set it right. There is already a code that is the boiler plate code. This means that this is the code that you will use almost every script, so its ready-made you to save time. Now we'll add a new object with this line above the Start void () :p Lic Rigidbody2D rb; Then place the next line of code in the Start method to find the hard body. It basically tells Unity to find the physics attached to GameObject that this scenario will be associated with (our player, of course). Start is a method that is performed as soon as a new object or script is created. Find the object of physics: rb - GetComponent Add this inside the update (:rb.velocity - new Vector2 (3, rb.velocity.y); Update () is updated again and so any code here will work again and again until the object is destroyed. you'll probably be using 'FixedUpdate' in the future. Save it and return to unity. Click on your player character and then inspector select Add a component of the scripts and then your new script. Click play, and boom! Now your character has to move to the edge of the ledge like a lemming. Note: If it all sounds confusing, just watch the video to see it all done - it will help! Very simple player inputIf we want to add a jump feature, we can do it very simply with just one extra bit of code: if (Input.GetMouseButtonDownDown (0)) - rb.velocity - the new Vector2 (rb.velocity.x, 5); This happens inside the upgrade method, and he says that if a player clicks, add speed on the y axis (with a value of 5). When we use, if, anything that follows inside a bracket is used as a kind of true or false test. If the logic inside said the bracket is true, then the code in the following curly brackets will work. In this case, if the player clicks on the mouse, the speed is added. Android reads the left click of a mouse like clicking anywhere on the screen! So now your game has basic crane controls. Finding your baseIt's basically enough to make a Flappy Birds clone. Throw in some obstacles and learn how to destroy the player when he touches them. Add score on top of that. If you get this down, no problem will be too great in the future But we have a bit more time so we can get more ambitious and make an endless runner-type game instead. The only thing that is wrong with what we have at the moment is that clicking the jump will jump even if the player is not touching the floor so that he can essentially fly. Fixing this gets a little tricky, but it's about as complicated as unity gets. If you get this down, no problem will be too great in the future. Add the following code to your script over the method public Transform groundCheck; Public conversion startPosition; public swimming groundCheckRadius; Public LayerMask whatIsGround; Private bull on the ground; Add this line to the update method above zlt;/Rigidbody2D/zlt;/Rigidbody2D Physics2D.OverlapCircle (EarthCheck.position, groundCheckRadius, whatIsGround); Finally, change the following line so that it includes on earth: if (Input.GetMouseButtonDownDown (0) ) All this should look like this: public class PlayerControls : MonoBehaviour public Rigidbody2D rb; Public Transformation GroundCheck; Public conversion startPosition; public swimming groundCheckRadius; Public LayerMask whatIsGround; Private bull on the ground; void start () - rb - GetComponent - invalid update () - rb.velocity - new Vector2 (3, rb.velocity.y); onGround - Physics2D.ShreddCircle (landCheck.position, groundCheckRadius, whatIsGround); If (Input.GetMouseButtonDown (0) - on the ground) - rb.velocity - the new Vector2 (rb.velocity.x, 5); What we're doing here is creating a new transformation - a position in space - then we set its radius and ask if it overlaps a layer called earth. We then change the value of Boolean (which may be true or false) depending on whether it is true or not. Thus, onGround is true if a conversion called groundCheck overlaps the layer of land. If you click save and then head back into unity, now you have to see that you have more options available in your inspector when you choose the player. These publicly available variables can be seen from within the Oneness itself, which means that we can install them as we like. Tap the right button in the hierarchy on the left to create a new empty object and then drag it so that it is right below the player in the scene window where you want to discover the floor. Rename 'Check Ground' and then make it a player's child just like you did with the camera. Now he has to follow the player, checking the floor under him, as he does. Select the player again and, according to the inspector, drag the new Check Ground object into the groundCheck space. Transformation (position) will now be equal to the position of the new object. While you're here, enter 0.1 where he says radius. Finally, we have to define our ground layer. To do this, select the terrain you created earlier and then in the top right to the right in the inspector, find where it says: Layer: Default. Click on this drop box and select 'Add a layer'. Now click back and this time select the ground as a layer for your platform (repeat this for any other platforms you have floating around). Finally, where your player says What is the earth, choose the ground layer. Now you tell your player script to check if a small dot on the screen overlaps everything that matches that layer. Thanks to this line we've added earlier, the character will now only jump to the zlt;/Rigidbody2D/gt; That's true. And with that, if you click to play, you can enjoy a fairly simple game requiring you to click to jump at the right time. With that, if you click to play you can enjoy a fairly simple game requiring you to click to jump at the right time. If you set your unity correctly with Android SDK, then you should be able to build and run this and then play on your smartphone by tapping the screen to jump. Road forwardEvidably there is a lot more to add to make this a complete game. The player must be able to die and turnip. We would like to add additional levels and more. My goal here was to show you how fast you can get something basic up and running. By following these instructions, you had to be able to build your endless runner over time, simply by letting Unity handle tough things like physics. If you know what you want to build and do your research, you don't need to be a coding master to create a decent game! Game! best city building games android 2020