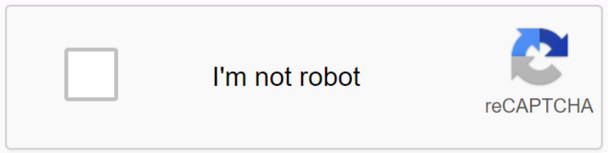

Continue

How to code gravity in scratch

For the past two weeks we have learned about gravity and building a moon module project to simulate how gravity works in the real world. What is gravity? Gravity is what keeps us down on earth, as we jump into the air, gravity will withdraw us. If you want to travel free of earth gravity you have to travel very quickly and achieve escape velocity it is about 25,000 mph. The space station, even if it travels at 17,150 mph (it is about 5 miles per second!) has not escaped the earth gravity while trying to travel away from Earth at 17.150 mph gravity pulls it off, making it travel around the earth in a Orbital rat. So we set out to build a gravitational simulation game. In order to make it a little easier we decided to use the moon as our base as the gravity is less and there is no atmosphere to delay us when we move. Some of the rules we had to code were: Our spacecraft should not move if it is on the ground. The higher it should move faster – Less gravity. If it starts moving left or right, it should keep moving until it is stopped – No Atmosphere. The Sprite needs 4 Costumes: Landed - This is the default costume where nothing happens. Up - This costume has a flame coming from the bottom. Left - This costume has a flame coming out on the right. Right - This costume has a flame coming out on the left. Once we built the sprite we started on the code. We first started the left and right code, basically when you clicked on the left or right button, we will change the costume and also lower a variable stage or decrease to indicate how far we need to move along the X axis (this is the one that goes from left to right). We used a variable for the LeftRight movement to make it simpler. It ended with some code that looked like this: And for the Right arrow We put a small delay in as well, it took the effect of pushing on and off that looks more realistic. The left legal movement was taken care of by a separate piece of code that just kept moving the Sprite. It also had a little code to stop everything if we touched a particular color, it was the color of my landing path. For the Up Code we did something similar, but we also had to add in the movement. Calculating the Up/Down movement was a little more complicated than we needed to adjust how much we moved up depending on how high we were, basically the Y value. Scratching doesn't help here, as the Y value goes from -180 to 0 and then to +180, we needed to always be positive, so we had to add a variable use and 180 to it. The other thing we had to do was find a number that was when we were divided by the Y value, it gave us a reasonable to move the Sprite through trial and errors it came out on 200. What ends up with some code like this: this: code also has all the initialization code, where we set all the variables we use to their initial values. Note that the UpDown default is 1, it is to ensure that the first time you press the Up key that the spacecraft actually moves. And if you want all the code, here it is and finally that's what my project looks like on end Simulating Gravity, or at least the effect of gravity in Scratch can be difficult, but due to trigonometric blocks in Scratch, it is possible. Tip: You should always adjust the scripts shown in this tutorial as needed to match your project. Creating gravity using Trigonometry This section is designed to teach you how to simulate gravity's draws on an object around a center point. It is similar to how the gravity of a star influences the movement of a planet. The script below is based on a simple trigonometric identity that states that sin2x + cos2y = 1. Below are the scripts you need to put into the object you want to turn: when flag clicked set [Distance v] to [100] set [Rotation v] to repeat [0] forever (360) go to x: ((([probe v] of (Rotation)) * (Distance: variables)* (Distance: variables)* (Distance: variables)* + ([x ranking v] of [Sprite2 v])) y: ((([food v] of (Rotation)) * (Distance:variables)) + ([y position v] of [Sprite2 v])) points in direction ([direction v] from [Sprite2 v])+(90)) if you then set [Rotation v] to [0] end as (Rotation) > then set [Rotation v] to [0] end as > Then change [Distance v] by (-1.5) end if you want the person who plays the game to control the rotation you can use this script : when you want to control the person who plays the game to control the rotation you can use this script: when you want to control the person who plays the game to control the rotation you can use this script: when you want to play the person the game to control the rotation you can use this script : when you want to control the person who plays the game to control the rotation you can use this script: when you want to control the person who plays the game to control the rotation you can use this script: when you want to control the person who plays the game to the <[359]></[359]> < (rotation)=> <[-359]>rotation you can use this <not></not> <touching [sprite2= v]?=>script to control: when you want to flag forever <key [right= arrow= v]= pressed?=>clicked as then change [Rotation v] by (2) convert costume to [costume3 v] end when flag forever <key [up= arrow= v]= pressed?=>clicked as then repeat (10) change [Distance v] by (5) end waiting to end when flag for <touching [sprite2= v]?=>clicked if then change <key [left= arrow= v]= pressed?=>[Rotation v] by (-2) switch costume to [costumes4 v] end For the Planet or other object rotated over you, this script will be rotated: when flag points forever to [planet v] Make sure you set the Planet or other object rotation to turn around. Using trigonometry is a smooth and effective way for more experienced Scratches to mimic gravity. With some more advanced scripts, you can even turn over non-circular objects. An example of trigonometric gravity can be seen here. Creating gravity using rotation Another method of simulating gravity is to change the center of a sprite so that when you turn it, it turns out to be drawn by gravity. This method is much more simple, but much harder to achieve. Change the sprite's center to the middle of the planet. Repeat Step 1 for every single sprite of that character. Write scripts for every sprite for every different situation: when flag clicked show goes </key > </touching > </key > </key > </touching > </[-359]> </[-359]> x: (0) y: (0) point in direction (0 v) forever <not></not> <key [space= v]= pressed?=>axis or >>> then if <key [right= arrow= v]= pressed?=>the <key [left= arrow= v]= pressed?=> <touching [sprite1= v]?=>costume to [costumes1 v] end when flag waits forever <key [up= arrow= v]= pressed?=>until play sound [jump v] repeat (20) switch costume to [costume4 v] move (1.5) steps end repeat (10) steps ends repeat (10) convert costume to [costumes4 v] move (1.5) steps end repeat (10) steps end repeat (10) convert costume to [costumes4 v] move (1.5) steps end (1.5) steps ends repeat (10) convert costume to [costumes4 v] move (1.5) steps end (1.5) steps end repeat (10) convert costume to [costumes4 v] move (1.5) steps end (1.5) steps ends repeat (10) convert costume to [costumes4 v] move (1.5) steps end (10) steps end (10) convert costume to [costumes4 v] move (1.5) steps end (10) steps end repeat (10) convert costume to [costumes4 v] move (1.5) steps end (10) steps end repeat (10)) switch costume to [costume 4 v] move (1) steps end repeat (10) switch costume to [costume v] move (-1) steps end (10) switch costume to [costumes4 v] move (-1.5) steps end switch costume to [costumes 1 v] go to x : (0) y: (0) eindig wanneer vlag vir ewig <key [right= arrow= v]= pressed?=>gekliek het indien en >> draai dan <not></not> <key [up= arrow= v]= pressed?=>cw (5) grade skakel kostuum na [kostuums2 v] wag (0.05) sekte draai cw (5) grade skakel kostuum na [kostuums3 v] wag (0.05) secs eindig indien <key [right= arrow= v]= pressed?=>en > draai dan <key [up= arrow= v]= pressed?=>cw (5) grade skakel kostuum na [kostuums4 v] wag (0.05) sekte draai cw (5) grade skakel kostuum na [kostuums4 v] wag (0.05) sekse eindig wanneer <key [left= arrow= v]= pressed?=> <not></not> <key [up= arrow= v]= pressed?=>gevlag vir ewig as en >> dan draai cw (- 5) grade skakel kostuum na [kostuums5 v] wag (0.05) pieke draai cw (-5) grade skakel kostuum na [kostuum 6 v] wag (0.05) seekatuur eindig <key [left= arrow= v]= pressed?=>axis and > Then turn <key [up= arrow= v]= pressed?=>cw (-5) degrees convert costume to [costumes4 v] wait (0.05) sect turning cw (-5) degrees convert costume to [costumes4 v] wait (0.05) sect one example of rotation gravity can be seen here. Creating gravity using physics One advanced technique for simulating gravity involves Newton's law of universal gravity: Variables and Lists of this comparison and Newton's 2nd Law of Movement (which states that F = mom) can solve us for changing in velocity of an object as: a = (Gm2)/(r2) Object due to the gravity of another (with mass =m2) in the equation above, three variables can be seen to be necessary: G (Newton's universal law of gravity) m2 (mass of the other object) r (the distance that is the objects of each other) G is a universal constant and can often lead to masses and distances that are very unaware To simplify the equation and allow you to easier/friendlier numbers use, we can actually ignore G. When you choose your relative masses you will factor it in. If the project involves very large (e.g. Moon-size) masses and large (e.g. low-orbit level) distances, use 6.67*10-11 for G. Next to the mass of object two and its distance, the speed of the moving object (object one) will have to be stored: Also, to rotate the acceleration in its x and y components, will be used a relationship (it is based on the idea that the x and y forces form a real triangle similar to which is formed by the objects itself) Finally,</key> </key> > </key> > Lt;key> </key> </key> </key > </key> </key> </key> </key> > Lt;/key> variable will be used to exploit through each mass and each object's x and y position: Those masses will be stored in a list: As well as the x and y positions: Script to start, a custom block is necessary that will iterate through each object in the project: define Check Objects / make sure it runs without screen refreshment! set [I v] to (1)/the beginning of the list repeat (length of [Masses v])/each object takes two items in the list . . . this is where the calculations will change [I v] by (1)/move on the next object end to note, the personal block must run without refreshing a screen. After ing the custom block, the distance between the sprite and an object must be calculated: define Check Objects set [I v] to (1)/the start of the list (length of [Masses v]) set [Dist. v] to ([sqrt v] of (((item (I) of [X Positions v]) - (X Position))*(item (I) of [X Positions v]) - (X Position))) + ((((((((((item (i) of [Y Positions v]) - (Y Position))*((item (i) of [Y Positions v]) - (Y Position))) change [I v] by (1) ends Next, the overall acceleration is necessary: define Check Objects set [I v] to repeat (1)//the beginning of the list (length of [Masses v]) set [Dist. v] to ([sqrt v] from (((item (I) from [X Positions v]) - (X Positions v]) - (X Position)*(item (I) of [X Positions v]) - (X Positions v]) - (X Position))*(item (I) of [X Positions v]) - (X Positions v])*(item (I) of [X Positions v]) - (X Positions v])*(item (I) of [X Positions v]) - (X Positions v])*(item (I) of [X Positions v]) - (X Positions v])*(item (I) of [X Positions v]) - (X Positions v]) - (X Position))*(item (I) of [X Positions v]) - (X Positions v]) - (X Position))*(item (I) of [X Position X Positions v]) - (X Positions v]))) - (X Positions)))-(item (I) of [X Positions v]) - (X Positions v]))) - (X Positions)))(item (I) of [X Positions v]) - (X Positions v]))) - (X Position))*(item (I) from [X Positions v])) - (X Position))*(item (I) from [X Positions v])) - (X Positions v]))) - (X Position))*(item (I) from [X Positions v])) - (X Positions v]))) - (X Position))*(item (I) from [X Positions v])) - (X Positions v]))) - (X Positions))*(item(I) from [v]) - (X Positions)))))-(item (I) of [X Positions v]) - (X Position))*(item (I) of [X Positions v]) - (X Positions))))))))))))))))))))-(item (I) of [X Positions v]) - (X Position))*(item (I) of [X Positions v]) - (X Positions)))) + ((item (i) of [X Positions v]) - (X Positions)))) + ((item (i) of [x) of [X Positions v])))) + ((item (i) of [x) Y Positions v]) - (Y Position))*((item (i) of [Y Positions v]) - (Y Position))))) set [Acceleration v] to ((item (i) of [Masses v]) / (dist.) * (Dist.)))/equation we found above change [I v] by (1) end Next, the power is needed to turn into its x and y components. To achieve this, the force will be compared to the distance, and that ratio, when compared to the horizontal/vertical distance between the sprite and an object, will achieve just that: define Check Objects set [I v] to (1)//the beginning of the list repeat (length of [Masses v]) set [Dist. v] to ([sqrt v] of ((((item (I) of [X Positions v]) - (X Position))*((item (I) of [X Positions v]) - (X Position))) + (((((item (i) of [Y Positions v]) - (Y Position))*((item (i) of [Y Positions v]) - (Y Position)))) set [Acceleration v] to ((item (i) of [Masses v]) / ((Dist.) * (Dist.)))/equation we found above set [Ratio v] to ((Acceleration) / (Dist.)) change [X Velocity v] by ((Ratio) * ((item (I) of [X Positions v]) - (X Position)))//x component of the force vector change [Y Velocity v] by ((Ratio) * ((item (i) of [Y v]) - (Y Position)))/y component of force vector change [I v] by (1) end The script is now done , although the variables X Velocity and Y Velocity must have some use: when gf clicked set [X Velocity v] to (0) set [Y Velocity v] to (0) forever Check Objects change x by (X Velocity)//application of the speeds changing y by (Y Velocity) defines Check Objects . . ./refer above for the coding Final Product Once the steps are followed above, it must be the Encoding: define Check Objects set [I v] to repeat (1)//the beginning of the list (length of [Masses v]) set [Dist. v] to ([sqrt v] of (((item (I) from [X Positions v]) - (X Positions v]) * (item (I) of [X Positions v]) - (X Position))) + ((item (i) of [Y Positions v]) - (Y Position))*(item(i) from [Y Positions v]) - (Y Position)))) set [Acceleration v] to (item (i) of [Masses v]) / (dist.) * (Dist.)))/equation we found above [Relationship v] to ((Acceleration) / (Dist.)) change [X Veloc.)) Change [X Veloc)) change [X Veloc.)) change [ity v] by ((Relationship) * ((item (I) of [X Positions v]) - (X Positions v])/x component of the force vector change [Y Velocity v] by changing [Y Velocity v] by (Relationship) * ((item (i) of [Y Positions v]) - (Y Position)))/y component of power vector change [I v] by (1) end when gf crazy Vlit set [X Velocity v] to (0) set [Y Velocity v] to (0) forever Check Objects change x by (X Velocity)/ /applying the speeds change y By (Y Velocity) Simulating Gravity in Scrollers Gravity can be repeated on Scratch to be used in Scrollers and other projects where an object is forced downwards. Listed below are several methods you can use. Velocity method The Velocity Method is a great method for creating gravity and is highly effective and adaptable for various situations. Here is an example script – please note that this script will have the sprite inside the ground rather than on top of it. Hierdie skrif sal geplaas word in die Sprite wat geraak word deur swaartekrag: wanneer vlag gekliek gaan na x: (0) y: (0) vir ewig verander y deur (Y Snelheid) stel [Y Snelheid v] te (Y Snelheid) * (Y S0.98)) wanneer vlag vir ewig <> <touching [ground= v]?=>gekliek as > dan verander [Y Snelheid v] deur (-0.1) eindig wanneer vlag vir ewig gekliek het as <touching [ground= v]?=>die vlag dan [Y Snelheid v] stel tot <> <touching [ground= v]?=>dat > Klik hier om 'n voorbeeld te sien. Direct Movement Method Direct Movement is more simple than the Velocity Method, but much less efficient at what it does. Not only does it look leap unrealistic, but it is much less practical in almost all situations. Nevertheless, the script works and is a good starting point for beginners: when the flag is forever <not></not> <touching [ground= v]?=>clicked as > Then change y by (-1) end when flag forever <key [up= arrow= v]= pressed?=>clicked as then repeat (10) change y by (10) ends waiting until <touching [ground= v]?=>Examples Simulating Gravity on Flying Objects With the right background and scripts, programming to fly a sprite can be pretty simple, but adding gravity helps with making platforms as well as jetpack games. The following script can help with the gravitational implementation in a flying engine: [problematic] when clicking green flag forever <key [space= v]= pressed?=>if there / You fly slide (0.4) sects to: x: (x position) y: (y position) + (40)) otherwise / You drop glider (0.4) sects to: x: (x position) y: ((y position) - (40)) or you can simply use velocity: when clicking green flag forever&gtt;/key > </touching > </key > </touching > </touching > </touching > </touching > </touching > </key [space= v]= pressed?=>then change [Y Velocity v] by (0.1) otherwise <(Y velocity)=>axis (-8)> then//Maximum fall speed change [Y Velocity v] by (-0.1) /Gravitational end as <touching [ground= v]?=>then / / a ground sprite set [Y Velocity v] to (0) end change y door (Y Velocity) Sample projects The following projects are examples of the above or similar engine: See Also Object Attraction Object Repulsion Physics Gravity Scrolling Scrolling> </(Y></key>