Guidelines

In space. Lesson №5

"Scratch is a project of the Scratch Foundation, in collaboration with the Lifelong Kindergarten Group at the MIT Media Lab. It is available for free at https://scratch.mit.edu".

Lesson guidelines

- 3D
- Messages
- Other blocks
- Sprites uploading
- Drawing

Slide №2

Messaging between sprites. And now we start a new game - "In space" (SpaceShooter). This will be 3D space shooter. Scratch offers only 2D, but we will try to create a three-dimensional world. Start a new project. **File** - **New**.

But first, let's repeat the material. Let's remember. Add two familiar sprites. These can be any other sprites. Let's create these scripts. There are two on the left - beetle, the right one - crab. Let's click. Everything is OK? Do you remember everything? Well. Let's delete one of the sprites. Delete the script from the remaining sprite. Create a **List**.

Slide №3

Lists. Create. Display. Repeat the **lists**. Build this script. Everyone shall enter their own data. Click. Watch. Joy. Did you remember the **list** device? OK! Deleting the sprite.

Let's start new topics.

However, first of all:

https://scratch.mit.edu/projects/387341536 Eye gymnastics (45 seconds).

We don't have much time to study. Therefore, we can no longer return to the topic we

have already studied, in order to save time on getting new knowledge. The **list** and **pen** topics can be considered closed. In the new game, we will pay a lot of attention to the entertainment side of the game. It will turn out something like this:

https://scratch.mit.edu/projects/399855796 **Space Shooter.**

This game is a demonstration of Scratch's ability to create 3D images using the laws of perspective. The closer the object is to the first plan of the drawing, the larger it is. If the item is far away from us, we draw it less. All this shall be applied to both photo and video images. One more effect shall be added to the video: the further an object is from us, the slower it passes by us. The object in the foreground moves very quickly. For example, a cyclist passes over us, and a plane sails through the sky. If you combine these two effects, you will get a simple 3D effect.

You can see that in this game the asteroid increases as it moves across the screen. This gives the impression that it is moving out of the distance, from the depth of the screen to us. The fireball is moving away from us, because it is decreasing with each moment of time

Today we will see how to create such 3D effects.

Slide №4

Blaster and asteroid. Storyline. The base in deep space is located in the zone of asteroids, from which you need to shoot back to protect the Base. Blaster is the robot, and it always operates in offline mode. It detects the asteroid itself on approach, selects the charge power, and fires plasma. Today the automation is out of order and it is important for the player to replace the robot before the repair would be completed.

The Blaster is precisely drawn in the game. Use a ready-made sprite from the Scratch library for the first version of your game. Choose something that looks like a weapon. You will learn the secret of how we got the image of a real Blaster at the very end of the lesson!

In Scratch, you can make your own sprite from a photo or take it from any project you like. For example, in our game, the asteroid sprite was made for another project and used here as the most suitable.

On the left you can see drawn models of Blaster and asteroid, on the right they are the same, but in the form of photos of real objects. Is it a photo of a real asteroid? Is it real? It is known that asteroids most often consist of ice. The photo shows an ice object. In fact, it is a crumpled piece of paper, it was made by using a photograph. You should know that. Special effects are created by artists for movies and computer games. In Scratch projects, it is important for each author to be a bit of an artist to make the project interesting.

For the first version of your game, take ready-made sprites from the Materials folder. In the future, it is important to use sprites created by your own imagination and skill. Are you agree?

Pay attention to the **center of the editor**. Click 2-3 times to zoom in under the editor window. The center of the editor will become clearly visible as you approach it. It is important that the **center** of the editor coincides with the center of the drawing. How is it possible to see the center of the editor if the drawing closes it? We can see it only approximately. Move the drawing, remember the location of the editor center, and return the drawing by aligning its center with the editor center.

Upload two sprites at once (Asteroid and Blaster).

Slide №5

Blaster control keys. In the game, the Blaster is mounted on a ring mounting that has three drives (motors that turn the muzzle of the Blaster left-right, up-down, and the third drive rolls the Blaster back and forth. In real projects, you can build a model of such a Blaster using servomotors that you may have already studied in circuity. And if you haven't studied it, then it's time to sign up for a circuity course! You can also learn how to create smart devices! In our game, the drives are activated by these colored keys, and turn the barrel each in its own direction. Each key will be represented by a separate sprite, so that the touch of these sprites ensures the rotation of the barrel.

How can I draw a key? In the picture below we can see the main actions of the developer. In the **select sprite** button, **select the brush-draw**. In **vector** mode, select the **fill** in blue color, then the rectangle tool. Drag the cursor over the lower-right corner diagonally to the desired size. **Shape change** tool: pull the bottom left corner to the left. We got a trapeze. **Click** copy. Let's create another sprite, we will **draw** it. In the new sprite editor that opens, click insert. There was a blue line. Click **reflect horizontally**. **Fill** in red. The second key is ready. Third sprite-**draw-rectangle-fill** with green.

Let's arrange all the control tools at the bottom of the stage in the middle, carefully put the barrel of the Blaster. Remember that a sprite placed in any place immediately gets a registration in the command field: the "**go**" and "**swim**" commands get the **sprite's coordinates**. They are also displayed under the stage. Assign clear and expressive names to sprites. Use English words, then your projects will be clear to children from all over the world. In our game, these are the names of sprites "left"," right", "straight" and" Blaster".

Warm up. **Slide 17** Select the music.

Slide №6

Blaster: left and right keys. Use the "**go**" commands to make sure that the sprite is in place. Creating scripts for the right and left drives. They differ only in **direction** and **coordinates**. After making one script, pass it to the other two sprites, changing the **coordinates** and **messages**.

You can always achieve the desired result in several ways! There are options for programming Blaster turns and shooting in our game.

This way you could program sprites to the Left and Right through the "**when the sprite is pressed** " events. Then you would have to constantly click on the keys to turn the Blaster. You could assign control to the keyboard arrows. We use the third option - the Blaster turns will be controlled by **touching the mouse**. This control is suitable for a smartphone or tablet, when the player controls turns and shooting with the touch of his fingers. Commands with the words directions inserted in the conditional operator "**if the mouse pointer touches**" do not specify these directions. These are just **messages**. Their purpose is still unclear. But nevertheless, we will create these **messages**.

The "**go to the front layer**" commands put the keys in front of the barrel. We will also define a layer for the Blaster - behind the keys. Perhaps the layers will still need to be corrected.

Tel: 02-8845-343-4, MB: 08 1019 3966, 08 7029 1414

https://scratch.mit.edu/projects/387341536 **Eye gymnastics (45 seconds)**

Slide № 7

Blaster: only straight. We will also create a **message directly** in the Return key script. We added the "**go back, go forward** " commands. These commands are intended for placing sprites that overlap each other in the correct order. This is important to do carefully and deliberately. Sometimes commands will get confused. You will have to click **stop** and start again. Sprites can be moved randomly with the mouse pointer. We've put the layers in order. To avoid accidental shifts, we inserted the "**go**" commands not in the initial conditions, but in the **"repeat always" cyclic operator**. The program will constantly monitor the sprite's position.

The Blaster script contains the initial conditions and the "**Other block all** " subroutine. We will explain its content later. Let's put the "**all**" command in place, and put the "**define all**" header on the script field for now, and we'll come back to it later.

We will complete the arrangement of Blaster control.

Slide № 8

Message to the Blaster from the keys. There are "**pass** " messages in the keyboard scripts. The Blaster itself receives **messages**. After receiving the **message** to the left, the Blaster turns to the left with the command "**turn** counterclockwise **by 1 degree**". In order not to spin endlessly in this direction, and some young programmers like to mess with this, we will limit the rotation to the sides. The **conditional if operator** will help us with this. **If** the angle of rotation exceeds **45 degrees** in one direction or another, the **"turn 1 degree in the opposite direction**" command is enabled. Why **1 degree?** It depends on the speed of rotation you selected. Try the other. Low turn speed gives the impression of massiveness of the Blaster, slowness of movement is characteristic of the underwater world and space. Turning slowly, you will target more accurately.

The **straight** direction is given to us by a hidden key- help-out: instantly puts the barrel in the up position.

Slide № 9

Plasma. The Blaster fires plasma. Who can tell what plasma is? When our plasma cloud, a clot of plasma, collides with some object, there will be a monstrous explosion.

Draw a plasma. **Circle** tool. **Gradient** of yellow and transparent. By **changing the shape,** we turn the circle into a long drop. This is how we imagine a plasma charge pushed out by an electron gun.

In the initial conditions, two commands "**go to the front layer"** and "**go back to 4 layers**". Plasma is a very mobile object that can easily confuse layers. Let the shot stand first on the **front layer**, and then count several **layers back** from it (for guarantee). The main thing is that the Plasma does not end up on top of the barrel at the most inopportune moment. It won't be nice. Having found its layer, our charge must **hide** and **wait** until the

trigger is pulled. The charge will be released only after the click command is triggered, which is a representative of the "**Other block define click**" subroutine. But the most important command is "**switch to Blaster**". This means that the plasma will be inside the Blaster and not in some random place in the barrel, but will connect its center to the center of the Blaster. **Will turn in the 0 direction** and wait for the start, which will provide **another block to wait**.

In addition, Plasma can stop the entire game (**stop everything**) under the condition **"when I get everything"**. We will see who will **give** it later.

Slide № 10

Plasma: Click. The condition "**if the mouse is pressed**" means: the command to shoot has been received. This is-first, to appear. Second, the shot is accompanied by a sound. **Select** a sound in your music library. Select the value for **"Set volume"**. After **showing**, the Plasma is removed from the Blaster by the command "**go 22 steps repeat 10 times**". But in the repetition process, the command "**change size by -8%**" also works (minus!). That is, the more Plasma moved away from the Blaster, the smaller it became, each cycle decreasing by 8%. This creates the impression that the Plasma is not just rising up, but moving away from the viewer, from the first person. So we got a simple 3D effect.

The **conditional operator "if it touches a Meteorite"** is inserted in the **cyclic operator "repeat 10 times"**. It causes the Plasma to bounce off the met Meteorite, changes the trajectory to a random direction in the range from -33 to 33. But the length of the Plasma path remains the same: 10x22 steps. On this path, the Plasma also decreases. When the "**repeat 10 times**" loop stops, the script proceeds to execute the next command - the "**wait** " subroutine. Let's look at this subroutine.

Complete Task 1 in Activity Books. And then:

Complete Task 2 in Activity Books.

Warm up. **Slide 17** Select the music.

Slide № 11

To wait and to go. The subroutine is simple. These are 3 commands that require you to return to the **initial conditions** after completing **repeat 10 times cycle**. Subroutines and **other blocks** adorn programs and make them more visible. In this game, we encountered a typical situation for complex projects - when one subroutine fits into another subroutine: the script contains a **click** subroutine, which in turn contains a **wait** subroutine. Note that subroutines - **other blocks** - can be moved from one script to another.

Top right is the first option for subroutines.

Why does the Plasma go in the direction of the gun? Plasma checks the **direction** of its flight with the angle of rotation of the Blaster the same **messages** from the keys that control the rotation of the Blaster itself. Therefore, both the Blaster and the Plasma will rotate completely synchronously. The keys were able to **send messages** to two addresses at once.

Question to the chat. What does synchronous mean?

Complete Task 3 in Activity Books.

Slide № 12

Dome. Drawing stages. **This is your homework**. At home, you can test your artist skills. First, use the **circle** tool to draw an empty oval inside-the **contour** is **6** mm thick, and the color is blue. Use the **shape change** to give the oval the shape of a dome. **Fill with a gradient** from azure to full **transparency**.

Our Dome sprite will actually perform the function of protection. From the First asteroid hit, the Dome will change the costume to its version with cracks. Draw them dark, **brush tool**, thickness **5**. The next costume is a Dome with an image of a hole. We will draw it using **shape change**. The third suit-the Dome will split into two parts. This is also done by **changing the shape**. But you will have to change the shape of the Dome so that it will remain half. You need to **copy-paste-reflect it horizontally**, then move it and **change its shape** a little, because the fragments are not the same. This is your homework. You will have a lot of time. You can even draw the interior of the cabin, or a glove. The script of the glove will move it behind the mouse (**go to the mouse pointer**) and "click" on the control keys will not be the **mouse pointer**, but the glove, i.e. the pilot's hand.

Slide № 13

Dome. 1-st script. The state of the dome depends on the number of Asteroids missed. To count such events, you need a **variable**. Let's call it **bang**. Let's set it to **0** in the **initial conditions**. In them, all other commands are clear to you. In the **repeat always** loop, we will assign the **frontmost layer** to the Dome forever. And now we will observe our instruments only through a semi-transparent Dome. This is not a first-person position, but an outside view

Question to the chat. How can we make the image appear in the first person? Yes, we need to be inside the dome itself. Look into space through a transparent vault. Can this be achieved in Scratch? You can try it yourself.

The dome changes only when using the conditional operator "**if it concerns an asteroid**". What, from the point of view of programming, should happen when an asteroid is detected? Answer: our variable is changing. Everything else is a **consequence of this change**. Including both subroutines **shudder** and **dome**. Let's create these subroutines and **other blocks**. We will deal with the **shudder** block now, and we will return to the **dome** block later, this will be the most complex script.

Slide № 14

Dome shuddered. The dome subroutine shuddered. After receiving a blow from above, the Dome will fall diagonally down. It will go straight back up. It will immediately go up diagonally and then back down again, and the Dome will shudder into place.

Let's pause for a moment and check how we understand the organization of the shot?

Complete Task 3 in Activity Books.

Slide № 15

Blaster and Dome. Blaster and Dome are linked by teams. We remember that the body of the Blaster script has the **all** subroutine. It determines what happens to the Blaster when an Asteroid or Meteorite hits it. The meteorite is not dangerous to the Dome, but a direct hit to the Blaster blows it up and with it the Dome. The dome falls apart when you receive the **boom message**, which comes just from the **all** subroutine. Visually, this is a **change of 2 costumes in about 0.1 seconds.**

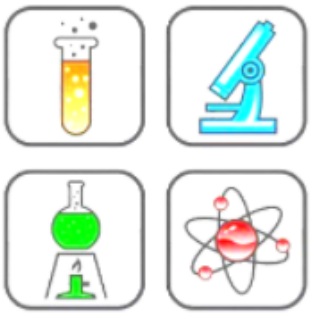
The **explosion** subroutine is also hidden there, which shows an animation of a Blaster explosion by **changing 5 costumes in about 0.1 seconds**. Both events occur simultaneously. In addition, the **Other block** should transmit the **message all**. It will get a Plasma that will **stop everything**, and an Asteroid only to make **hide** at the end of the game. This is done so that it doesn't stay in the middle of the screen when the game ends. The slide turned out to be rich and you can trace the relationship of 4 sprites at once.

Slide № 16

Starry sky. The lesson is coming to an end. The remaining sprites (Asteroid and Metorite) are complex enough to start considering them from the script now. We will continue it in the next lesson. Before the end of the lesson, we will have time to make a starry sky. It could be created as a background. But we have something more spectacular in mind. So let's create the sky as a sprite.

Choose to **draw** a sprite. This time we choose **raster** mode. At the bottom it is suggested to convert to vector graphics, so we are in **raster** mode. This mode is intended for drawing with dots. And the stars are just points. Select the **fill too**l (bucket). Choose two colors: black and dark blue. **Gradient from top to bottom**. Click to make the black one at the top.

Create stars! Feel like a Creator! Take the **brush too**l and select the **fill** white, the **size of the brush**, choose the place where you will place the stars in the sky with the **brush**. It took a long time. Each trace is an asterisk. You can sometimes change the **brush size** and color.



Today, we can see 88 constellations in the sky. Many of them you could see during a clear night - Orion, Cassiopeia, and the Bears. Now scientists also continue to discover new stars. Names of new stars are chosen by voting in the International Astronomical Union. For example, one of the new stars in the constellation Pegasus, at a distance of about 160 light years from Earth, similar in weight and size to our Sun, received in 2019 the official name-Solaris, in honor of the famous novel by Stanislav Lem. (https://www.bbc.com/russian/news-50811879).

Setting the initial conditions for the Sky. It is important for him to go to the back layer, because all the others look against the sky. **Set the size to 122%**. What is the aim? This is our know-how. Our Sky will twinkle, wink at us with stars. The astronomer will rightly be told that the stars twinkle only in the earth's sky - because of the atmosphere, and in space they

emit a constant light. That's right! But we deliberately violate the laws of physics for the sake of entertainment. The blinking of the Stars will be realized due to the fact that our Sky will slide. It will be very subtle, very slow, but ... beautiful. To do this, we set the scale to 122%. If it was 100%, the Sky would open up empty corners of the stage as it moved. And at 122% scale, the Sky is bigger than the stage.
Click the checkbox. We are looking at a corner of our galaxy where our thirst for travel has brought us. Yes, no shooting yet. Just a trip. We will continue to repel attacks of Meteorites and Asteroids in the next lesson.

Creating music for our game. Enter the Stage. Creating a new sprite. The music is taken from the music library; the section is named **Space**. You can choose the **volume** yourself! Let's explain what it means **to wait**. You can insert the **play sound** command to the end, and it will play. But! There is a short pause at the very end of the sound. Therefore, when the audio file is played in full, this pause makes the melody intermittent.
The first way to avoid intermittent music playback is to cut off this pause in the editor.
The second method is to use the **wait** command to shorten the duration of the sound to the specified **wait** value. The length of the melody is exactly 10 seconds. We left her **9.9 seconds**.
The melody became continuous!
Reception: r**epeat always-turn on the sound-wait** for seconds as long as the melody lasts allows you to play the melody. If to remove "**wait**", then it will not work. It will start over every moment, according to the instructions of the cyclic operator. But the command "**play the sound to the end**" inserted in "**repeat always**" will be played as it should be. After all, it is prescribed to finish first, and then only start again.
Check that your music is playing and the stars are twinkling. A Blaster with a control panel is visible under the dome. You can control the turning of the Blaster. You can shoot. You can pull out "**put boom**", click and ... ruin the program. For this situation, click **"stop, check box"** and everything fell into place.

The lesson is over! Thanks!

You can stay for 1 minute if you are interested in finding out the secret of why our Blaster looks like a real one! We just took a photo of a real Blaster. It is made of real plasticine by a student of the ROBBO Club from Russia. We took a picture of the craft in the **Scratch image editor** by selecting the **camera** option. The photo immediately goes to the editor window. Next, you need to carefully remove the surrounding **background** with an **eraser**. This work requires accuracy and patience. As a result, we got a new sprite!

See you in the next lesson.

Complete tasks in Activity Books.

Task № 1

Retire. The timer is always greater than 0, so the difference 1t - 3t is always < 0 and the size will decrease.

Task № 2

2D: nowhere. 3D: straight.

Task № 3

Yes no.

Task № 4

2 - 5 - 3 - 1 - 4 - 6