Guidelines

My game. Lesson 3

The main concepts discussed at the lesson

- Coordinates
- Drawing
- Arrow controls
- Other blocks

Electronic educational resources used at the lesson

- Internet-enabled computer
- preinstalled RobboScratch3, Zoom or Blue Jeans
- Own account in the community on https://scratch.mit.edu/

Features of the lesson's representation

This course has been developed by the ROBBO Methodological Department specially for distance learning.

A special thing about the course is that the material is presented visually in the form of slides. Explanations for each slide follow below.

Greetings.

Hello, everyone! Welcome again to our RobboClub.
This is our third lesson. A reference to a game that has already been made. You can take a look at her, refresh the clones in memory. We 're gonna need them again.
https://scratch.mit.edu/projects/387914131
Some of you trained at home and fixed your scripting skills.
Open every game you play. Start. Make sure everything works.

Revision.

Show on your screens how it works and what doesn 't work. Discussion. Who works everything, chat.
Let us repeat the basic concepts learned in the last lesson.

Slides №2 and №3

What is a clone? Clone is a copy of the sprite. But unlike a sprite's take, a clone can have its own individual characteristics: its color, size, and location on the stage.

A clone is always set using two scripts. What script do you need? The start script creates clones. **When I start as a clone** command controls clones.

Slide №4

How can a variable appear in the program? It should be created.

Slide №5

How does the program know that it has a variable? The program should be introduced to the variable. Set the initial value and set it under the checkbox. This is called "initialize a variable".

Move on to new topics.

But first of all:

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

You can close your game. We are starting to create a new game. Click File-New.

Slide № 6

Coordinates. Everyone must have played sea battle on pieces of squared paper. It is important to understand that each square has its own address, and there cannot be another square with the same address. In the game sea battle (chess), the address is encoded by two characters: a letter and a number. D; 6 B; 8, etc. It is desirable to separate

the signs with a semicolon, as is customary in mathematics.

The scene in Scratch is also divided into squares, but there are a lot of them and they are not visible to the ordinary eye. If the cells in the Sea Battle are only 10 in width and 10 in height, then in the Scratch they are 480 in width and 360 in height.

The location of a point on the Stage is also set by two characters, but both are numbers. How not to get lost in them? It is possible to say: 133 to the right, 27 to the bottom, as we have already tried. There is another way in mathematics. Those numbers that change from right to left are designated by the Latin letter **X**. and those that change from bottom to top are designated by the letter **Y**.

Point 0; 0 as you already know, is located in the center of the stage. If you move from the center to the right, you will make 240 steps to the edge, or you will pass 240 cells. You already know that the sprite can't go further than the edge. You can see that in the picture the submarine has coordinates 211; 0, because it did not reach the right edge a little, and from the top edge and from the bottom it is at the same distance, which means y=0.

The sprite receives the command to stay in such a place via a command from the Movement blocks, **go to x: y:**. If such stickers were in the game sea battle, they would look something like on the slide:

(Go to letter: (B) digit: (8)), I.e. Go to B; 8, for example.

Slide № 7

Coordinates 2. If you go to the left from the center of the stage, it will also be 240 steps to the left edge. To distinguish them, it is used to count movement to the left with a minus, and to the right with a plus. in Other words, x changes from -240 to 0 and from 0 to 240, a total of 48. The same is true for y: up from 0 180 steps, down also 180 steps. And these 180 steps down also have a minus sign.

You are already familiar with the concept of negative and positive numbers.

Question to chat. Here are the numbers: 33, -3, 3, and - 33. Write to t:

- the least number;
- the largest number;
- the least positive number;
- the largest negative number.

The point in the center of the stage has coordinates 0; 0.

It is possible to draw two lines crosswise through this point 0;0. More precisely, perpendicular. They are called axes. The one drawn from right to left is the x-Axis, and the one drawn from top to bottom is the y - Axis.

The coordinates of the sprite are always visible under the stage in two oval windows. If it moves, the coordinates change. When the game is stopped, you can record the desired coordinates in these boxes, and the sprite will move to the specified point. To start, enter 0 and 0 in both boxes.

Slide № 8

Coordinate system. Stage dimensions. Let's see. Remember the dimensions. Remember the coordinates of the two corner points.

Question to chat. Name the coordinates of the other two corners of the stage.

Warm up. **Slide 22** Choose the music.

Slide № 9

Move to random location. Select the sprite. Place it in any place. Immediately note that the command field in the windows already has the value x and y. Drag sprite a little. The values will immediately change. I.e. the coordinates of the sprite are determined automatically. It is very convenient. It is possible when the **go to** command is in use. When it is in the script field, the x and y values in the sticker do not change, but in the command field they will change again if you drag the sprite.

Random numbers.

Enter **Operators** block. In the script field, pull out the longest green **pic random 1 to 10** command. Call the command, see what numbers are obtained, now enter instead of 1 - 30, and instead of 10-77. Then enter -10 instead of 1 on the left, and 10 on the right. See what numbers will appear when you click on them. You can see that both negative and positive numbers appear randomly.

Select the **go to random position** command in the **Motion** block. Click it. The sprite is here and there. The disadvantage of this command is that the sprite may partially go off the screen. It's not good. A programmer is always looking for beautiful solutions. We can set the x and y values for the sprite so that it doesn't go over the edge of the screen.

Select the command **go to x: y:.** Let **Y** = 22, and let **X** change randomly from the left edge of the screen to the right. We know that the largest value is x = 240, and the smallest value is x = -240. Let's make it less so that the sprite doesn't touch the edge with its rays. For example, from -222 to 222. Click. The sprite moves to different locations, but remains at the same height.

Now let's make random not only x, but y as well. How will y change? Let it will change from -166 to 166, so that the sprite doesn't cling to the edge. Click. The sprite appears at different points without touching the edges. Getting random coordinates x and y, the sprite moves to a random location.

Now let's play this game and try to guess (approximately, +-20, the coordinates of the starfish). Remember about Minus and Plus!

https://scratch.mit.edu/projects/392273105

Let's start a new game.
What kind of game will it be? I suggest you make adventure with obstacles and enemies. Play it..

https://scratch.mit.edu/projects/394270172 Circle against squares. The circle must cross the stage and touch another circle in the lower right corner. It is hindered by evil squares and curbs.

Slide № 10

Background. The background can be used in materials. Or it is possible to draw it in

the image editor in raster mode. The programmer must be able to use graphic editors and draw in them. This develops motor skills, taste, and imagination. If you draw, you can make the background of any color, plain or with a gradient - this is a smooth transition from one color to another. It is important to place obstacles properly. You can experiment at home. The color is obtained by changing the position of the engine. Let the brightness and saturation be the highest-on the right. Click on the tilted can to fill. It is better to draw obstacles using the Line tool. The thickness can be changed. You can make two more backgrounds at once.

Slide № 11

Circle control. Let's create a sprite circle. It should be created in raster mode. We do not take a ready-made one, because our plan is to "deflate" in case of failure. We'll do that later. Just draw a circle with the circle tool. Choose the color to your taste. But this slide is not about drawing a circle (it's too easy), it's about controlling the circle.
We will control it with arrows. Go to the Event block. There are so-called Headlines here, they look like hats or caps. Scripts start from them. This is, for example, a checkbox. We now need the header **when..key** pressed. The keyboard keys are available in the drop down list in this command.

Let's take the **Up Arrow** key first. Here the main thing is that under the cover of the up Key was the command Turn in the direction of 0 (up). The direction is set, and the speed must be set. We adjust it to go 10 steps. Check. Each time you click the up arrow, the circle goes up 10 steps. Similarly, activate three more keys to move to the right, left, and down.

But our controls turn out to have one unpleasant feature: the circle does not want to walk diagonally. Although, it would seem that when you press two arrows simultaneously Up and to the Right, why not take a shortcut? No, it doesn't work.

But there is a way out. You need to use the Conditional operator **if...then**.

Let's make a construction like: in the Conditional **if...then** operator, insert the command **Key Up arrow pressed**? (from the Sensor block), and inside the conditional operator of the **Point in direction** and **move 4 steps command** (Choose exactly 4 steps). This will give us more maneuverability! Repeat for all other arrows. Let's create a script like on the right side of the slide. Start the script with the command **when space key pressed** - this will come in handy later. We shall determine the initial coordinates of the circle - its position at the beginning of the game, its size, costume number, and display Let all initial conditions be set. You will agree with them later. Throw out the first four scripts When the arrow is pressed, they are no longer required.

Warm up. Slide №22

Slide № 12

Obstacle. So, the circle is well controlled by arrows, but it ignores obstacles. Make it harder for him. Let the circle feel the curb as something solid and impenetrable When it touches an obstacle, it bounces back to the starting point of its journey. Since our obstacles differ from the background in color, we will use the **touching color command .?** from the **Sensors** block to organize the rebound

The bounce algorithm provides a conditional **If** operator. **If touching the Color**, go to the initial coordinates. The color is selected with a pipette, which will appear if you click on

the window with the color sample. The obstacles are now insurmountable for the circle. Let's add sound so that we cannot only see the touch, but also hear it.

Slide № 13

Square. Drawing. In addition to stationary obstacles, the movement of the circle is complicated by mobile units of its opponents. They can be called enemies, antipodes.

Let's create a square sprite in the image editor. Select vector mode. It is equally easy to draw a square in any mode. But you need to master the vector one. Click Go to vector mode under the drawing window. Select a color using the engine. Switch the outline to transparent. Select the Square tool (just the namesake of our sprite). Click near the center and drag the mouse to the desired size. Don't forget about the center of the object. It should be combined with the center of the editor window.

Slide № 14

Programming enemies. The enemy must interfere, attack and build other schemes. In our idea, the circle, as soon as it appears on the scene, as some predators' rush to it, and meeting them does not bode well for the circle.

Retreat - do not forget that you are game developers and that you can actively use your Scratch account. Any game in any account can be copied to your account - make Remix with one click. In your account you can change the game, as you like, adjust for yourself, understand the algorithm of the game.

So we drew a square. Let's start reviving it.

One enemy is too easy for us. Let it be 5 of them. We will multiply them using the cloning method. Space again instead of a flag. It is more comfortable to start with a space: you do not need to reach for the flag with the mouse. Next, use the **Create a clone of yourself command, Repeat 5 times.**

Let's give the circle a 1 second head start in the clone program (using the **wait 1 second** command) and then immediately rush to the circle: the **point towards Circle** command is selected the same way as **Turning to the mouse**. We set the speed in 2 steps - we will select it if necessary. And finally, our favorite - If the edge...

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

Slide № 15

Subprogrammes. Other blocks. Programs can be very complex. You have already come to those. You can put books, clothes, dishes, shoes in one pile. Even if it will be neat, but it will look strange, and it will be inconvenient to choose, and you will not remember where you put it. But you can structure this stack, divide everything into logical groups, put them in different places (Cabinet, table, drawer) and to be sure, also sign each storage location.

So in our complex or just long scripts, we can notice that some elements of the script do one job, some another. Therefore, you just want to combine performers of one type into one group and put them in one Cabinet, and another group, say, in a bag.

Yes, while your programs are not so complex that there is an urgent need to structure

them into subprograms. And subroutines are just groups of teams with similar tasks. Moving is one task, changing color is another. But we learn, master the Scratch apparatus, its interface. And constantly increasing our knowledge.

The Command block **Other blocks** are located next to the **Variables** block. We click. Already familiar proposal Create. Click, Create again. You have already guessed what to create means to name. We print the name. We simply call it: **Movement**. OK. You made your first Other block!

Slide № 16

Subprogrammes. Place in the scripts. You have already figured out for yourself where Other blocks fit in the scripts. And they understood why they were called Subroutines. Because they are only part of the main program, the script. Without the main script, they simply won't work. The subroutine itself is located under the cover of Another block, and in order for the main script to execute it, the subroutine leaves a sticker with its name in the script body instead. The script, executing its commands, will reach the Subroutine sticker, jump to it, execute the subroutine, and return to its other commands. Or will it do it all at the same time.

Let's create two more **Other blocks**.

Slide № 17

Subprogrammes. All blocks are assembled. First of all, the programmer must decide what he wants from the sprite. It would be wise to diversify our clones as much as possible. What can we offer them within the framework of the technologies we have studied? As usual, this is the appearance at the start of each of them in a random place. The command **Go to x ... y ...** Set it with a margin of freedom, so that they do not fly off the screen. Setting randomness limits for size. The size is measured in%%. 100% = your drawing. You should not make a large spread in size. what I suggested (77 - 111%), I consider optimal - tested. Perhaps your %% will have its own standard, its own values. Experiment. And of course - a variety of colors. The number of shades in the Scratch you remember.

Question to chat. How many shades of colors offers the command to Set the color effect?

We have put together three commands that provide a variety of clones. How do we name this assembly? Let it will be Variations.

The good news: obstacles for a circle are also insurmountable for a square, on a square when it hits an obstacle - just bounce off it a little and turn slightly. So, on the slide, all the blocks are assembled. Make sure that changes in Other block affect the entire program. Replace Go -5 with 0 in the command. What happened? Don't forget to return -5.

Perhaps in the square sprite structuring, simplification, ordering is not very noticeable, but see how it looks in the circle.

Slide № 18

Subprogrammes. Circle. Here's how elegant the circle script looks. We did not add anything to it, so that nothing would prevent us from admiring it. Imagine what a pile of scripts would be in the sprite, if not for Other blocks. However, you understand that the pile has not gone away, we just removed it from sight. But no one will do them for us, so let's continue.

Slide № 19

The other blocks of the circle. Movement. Obstacle. We parted with the circle, giving it the ability to walk on arrows and straight, and diagonally, as well as strictly instructing him not to climb over the fence.

Then everything is simple. Creating Other Movement and **Obstacle** blocks. The good news is that Other blocks can be named the same in different sprites. the program will not be confused with them. And most importantly, we ourselves are less confused: we chose the word **obstacle** for a square and for a circle - now we know exactly what these blocks are responsible for in different sprites.

We connect the corresponding sets of commands to the Headers of Other blocks, removing them from the body of the main script, and repeating them always from the Cyclic operator. Instead, we insert stickers of Other blocks into the empty body (see slide 18).

Also for clarity, and on the left side of the slide, place the entire Other block called Movement. Here it is, a pile-up, really has not gone anywhere.

Slide № 20

Subprogrammes. Circle and other sprites. There are two more sprites in the game. In addition, the circle has 4 suits. Let's dress up the circle first. We'll add three more suits to it. Draw in raster mode. These are very simple drawings. As we know, the circle is deflated when we lose. To see the air coming out of the circle, it was slightly colored, in General it is white, against the background of the scene is clearly visible.

Creating two sprites by drawing. Drawings on the slide. We will make a small script for everyone. Sprites are stationary. Let's look at their tasks now.

Slide № 21

Subprogrammes. Circle-Sprites. As in any game we have two endings: victory and oops! And oops will be the real oops. The balloon deflates. Squares attacking the circle with each touch additionally inflate it. The poor circle, if it cannot escape, swells so that it bursts or deflates. At the same time, it gradually disappears, having managed to admit: - Deflated.

We have just created 2 more sprites. The circle sometimes concerns them, but the consequences are different.

If the circle touches the squares, it is thrown back by 33 steps, the squares make some terrible sounds after it, and its size increases by 5%. (You can immediately add the enable sound champ command).

If the Finish sprite is touched, it is a victory. The circle floats 1 second to the starting place, a joyful rap sounds, the background changes, the score of victories is announced (we also add the sound of dance celebrate). (variables for the Victory will be programmed in the next lesson)

Slide 22

Warm up. Used at the discretion of the teacher several times during the lesson.

Complete tasks in the Activity Book

Task № 1

1320 steps.

Task № 2

All.

Task № 3

You don't need a Cyclic operator.

Task № 4

No, it is created for each sprite.