O3 – Instructional Support Content

INSTRUCTIONS FOR STUDENTS:

APPENDIX TO O3-A1 (COLLECTION OF GAME DESIGN BASED LEARNING SHEETS TARGETING TEACHERS)

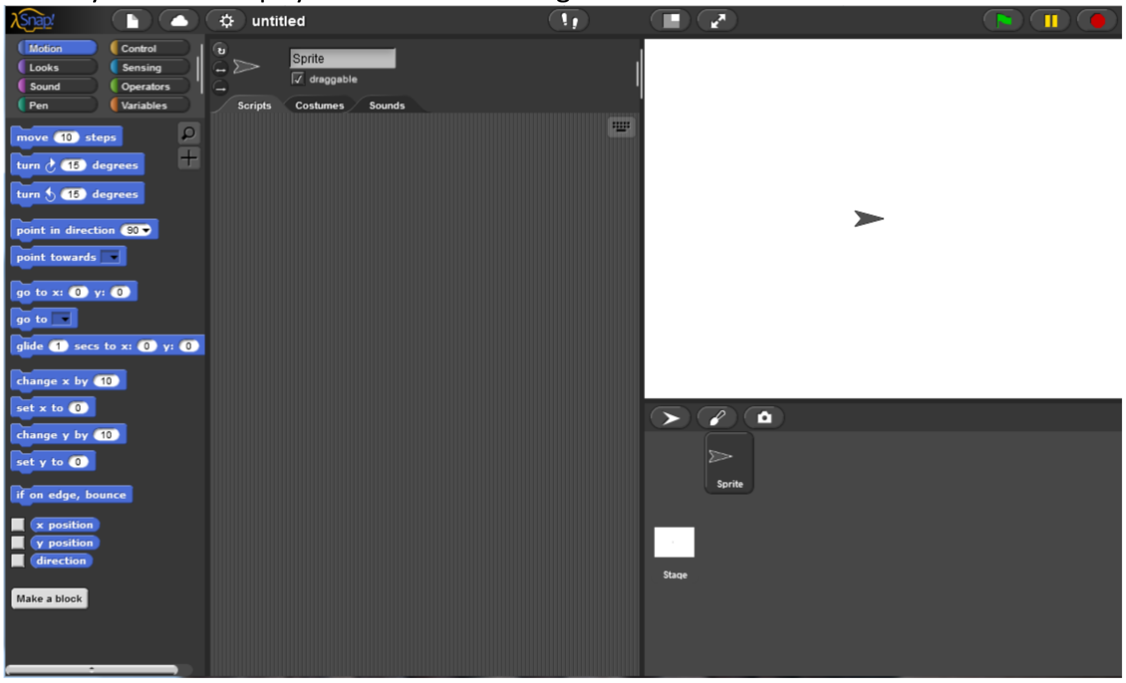
Document Data

Deliverable: Instruction for students: Appendix to O3-A1 (Collection of game design based learning sheets targeting teachers)
Partners involved: University of Ljubljana (Slovenia)

Table of contents

Scenario 1 - Introduction to Snap! interface

When you start Snap! you see the following interface:

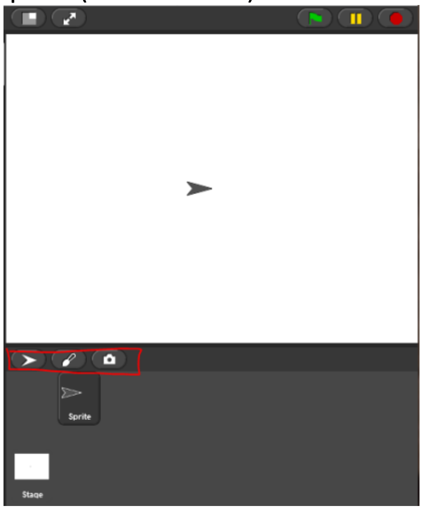


We will look into different parts of interface step by step, starting with the right side of the window.

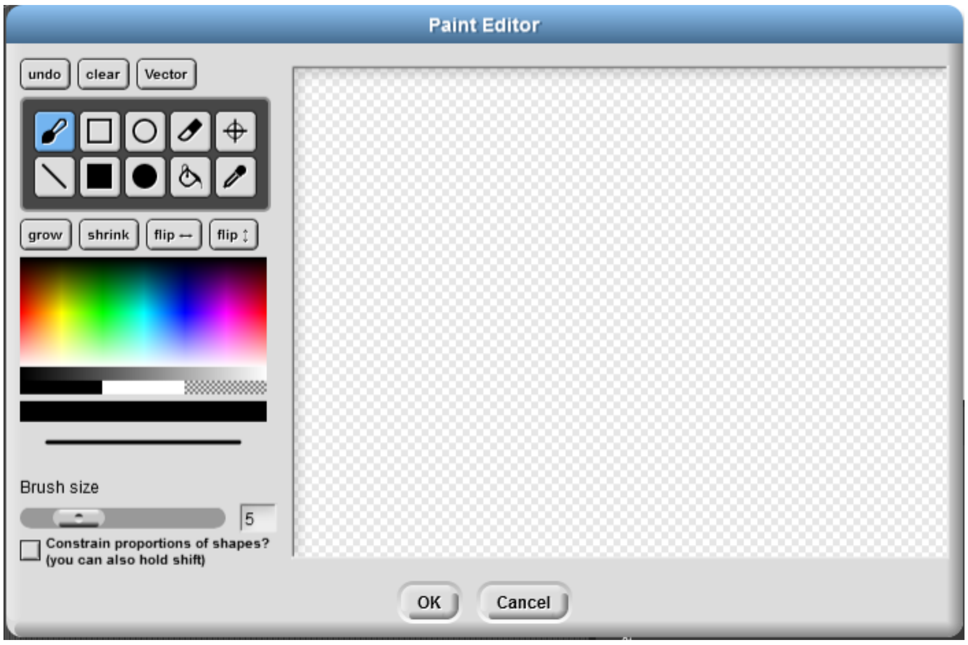
On the right side we can find a stage with a sprite on it. In the beginning the stage is plain white and the default sprite is a Turtle in shape of an arrow.

Task 1: Create a new sprite

To create a new sprite you will have three options: to add a new Turtle sprite (arrow icon), to paint a new sprite (paintbrush icon) or to take a camera screenshot and import it as a new sprite (camera icon). You can find these icons just under the stage as shown in picture.

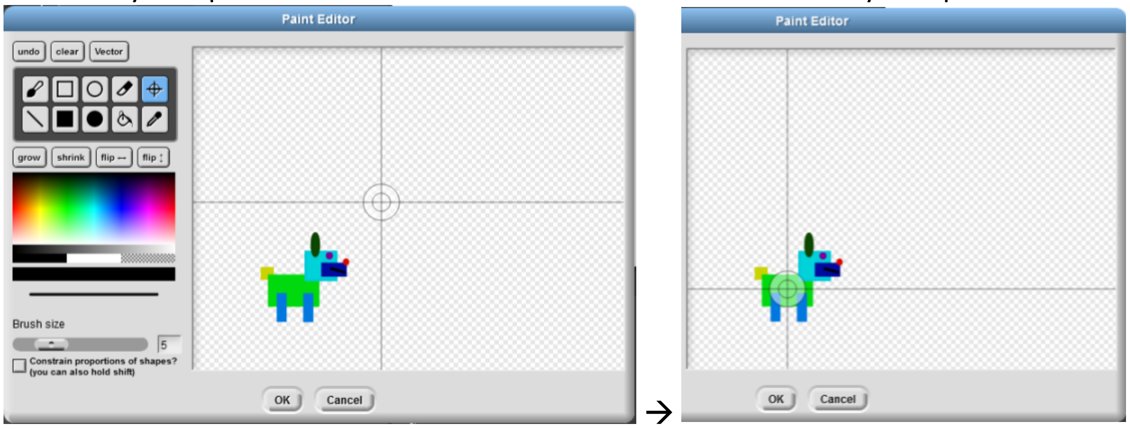


At the moment you would like to paint a new sprite, therefore click on the paintbrush icon. When you click it, the following paint editor opens:

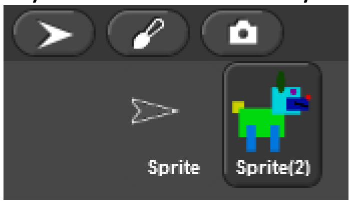


As you can see, the paint editor has an option of freehand drawing, drawing a straight line, drawing a rectangle or a circle (which can be filled with colour or not), erasing, filling a region, picking up a colour anywhere on the screen. Try it out and create your own sprite.

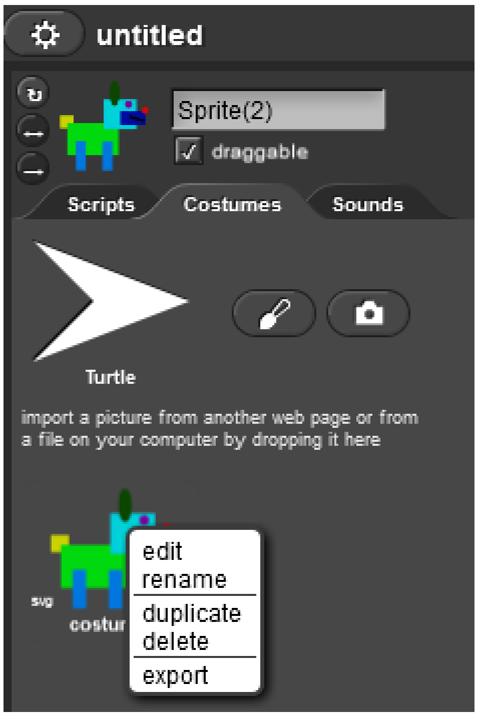
When you are done drawing use a special feature ⌖ that enables you to set the rotation centre of your sprite. If needed move it so take it is not outside of your sprite:



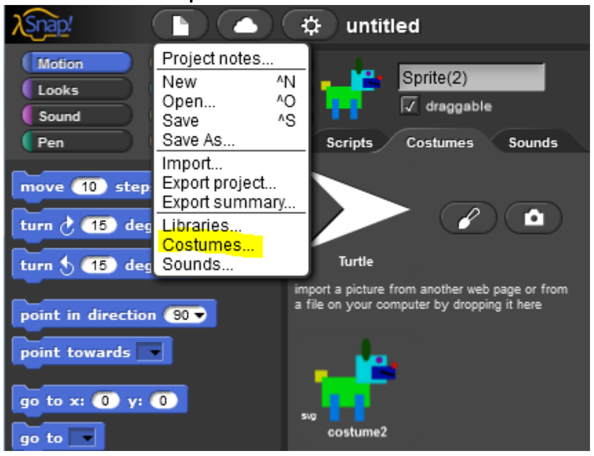
Click OK to save it.
If you still have to edit your sprite, make sure your sprite is chosen.



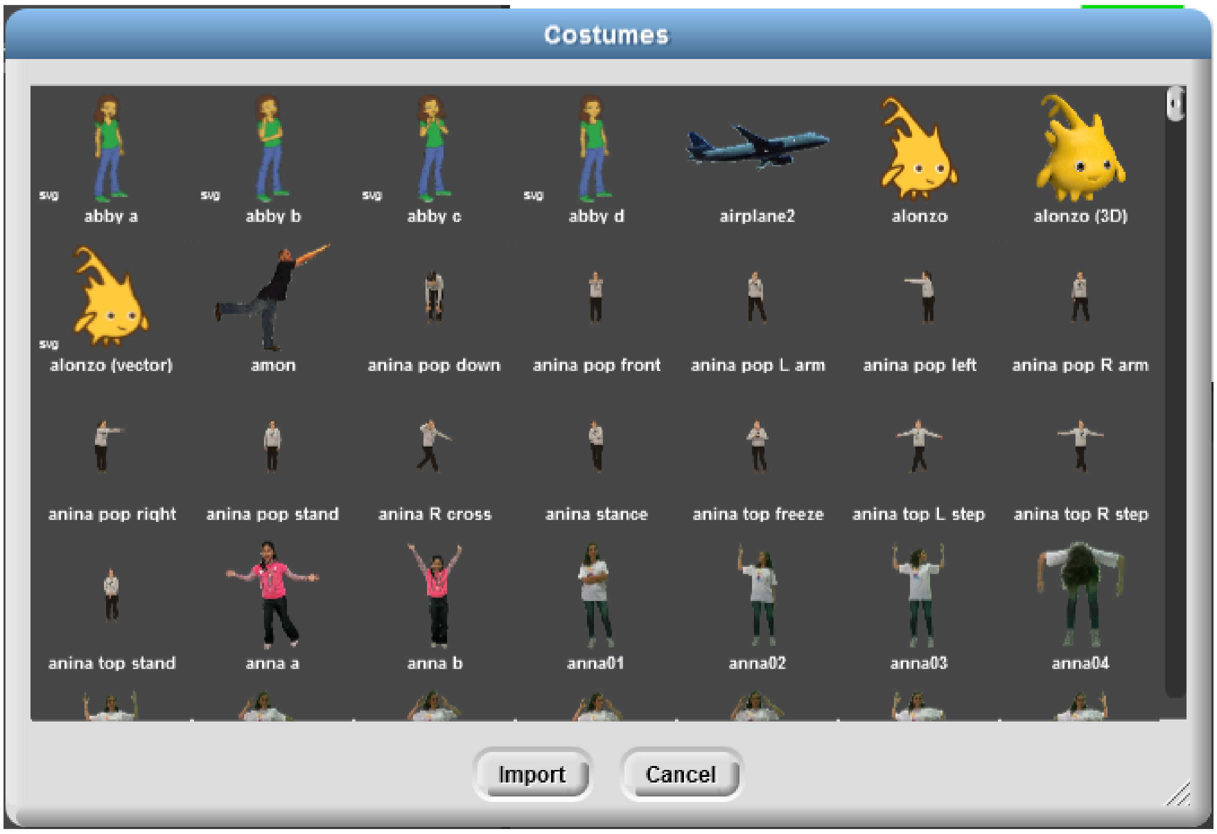
Then click on the Costumes tab below the sprite name, right-click on the costume you want to edit and chose edit to reopen the paint editor with your costume.



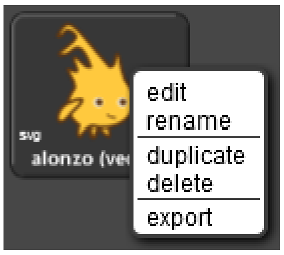
Every sprite can have one or more costumes. You can add a new one by clicking on a paintbrush icon to draw a new one or right-click an existing one and choose duplicate and then edit it for changes.
Costumes can also be exported if you need them in another project.
If you are not so much into drawing, you can import a costume by drag-and-dropping an existing picture from your hard drive to the Snap! or choosing a costume from existing ones. For the later you click on the icon that looks like a piece of paper, and choose Costumes… as shown on the picture.

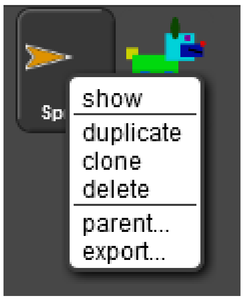


This opens a window with different costumes to choose from and import into your program.



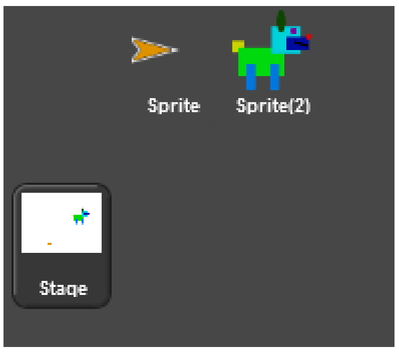
To delete a costume, right-click on it and choose delete. If you want to delete a sprite, right-click on it and choose delete.

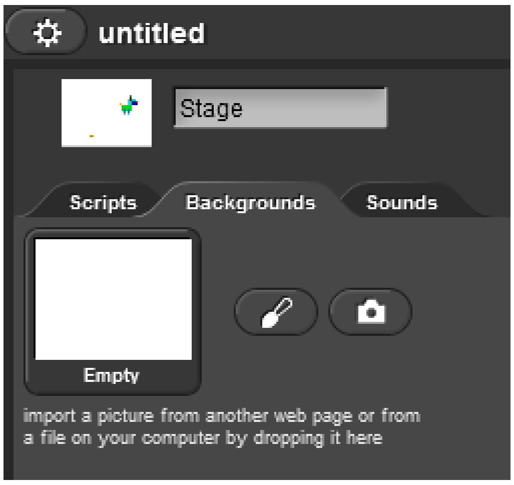




Task 2: Create a stage background

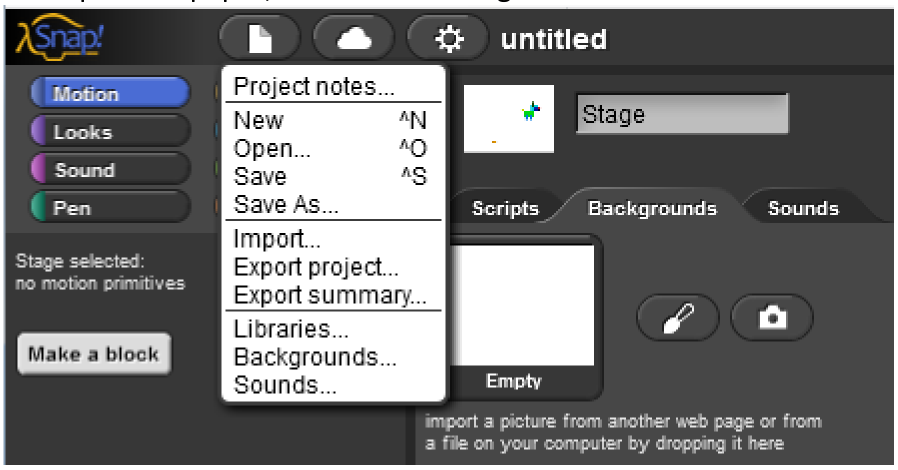
To edit or add a new a background, you first have to click on the Stage and select Background tab.





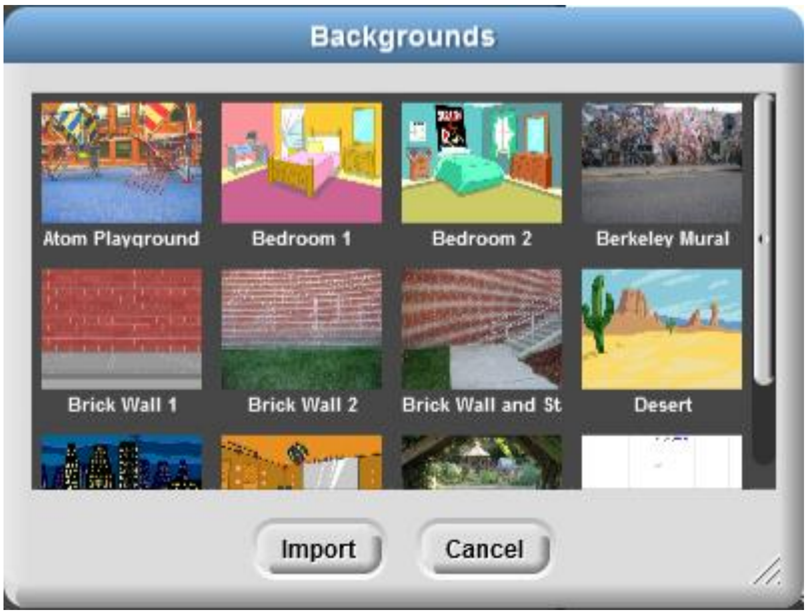
import a picture from another web page or from a file on your computer by dropping it here

To draw a new background, choose the paintbrush and the Pint editor opens. You can now draw your own background. Don’t forget to click OK when you finish.
To import an existing background drag-and-drop it into Snap! or click on the icon that looks like a piece of paper, and choose Backgrounds…



import a picture from another web page or from a file on your computer by dropping it here

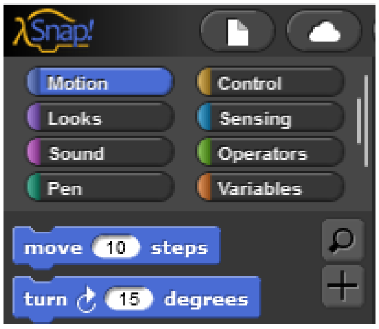
This opens another window with existing backgrounds.



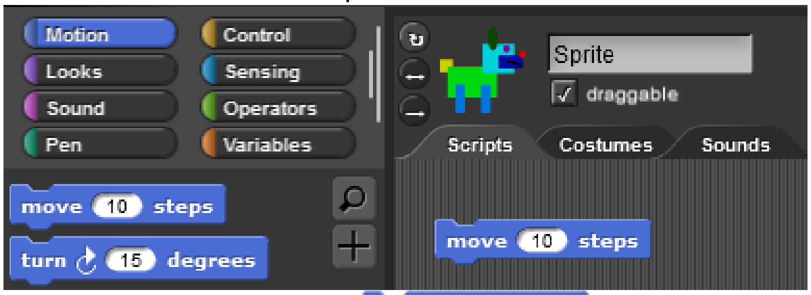
Editing backgrounds is similar to editing sprites: you right-click on it and choose edit.

Scenario 2 - Time to bring your sprite to life

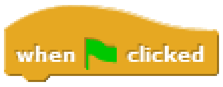
You have a dog and you want it to move. To make it move, you need to write your first program.
Look at the leftmost side of the interface. There you will find different blocks, which are divided into several categories: Motion, Looks, Sounds, Pen, Control, Sensing, Operations, and Variables.



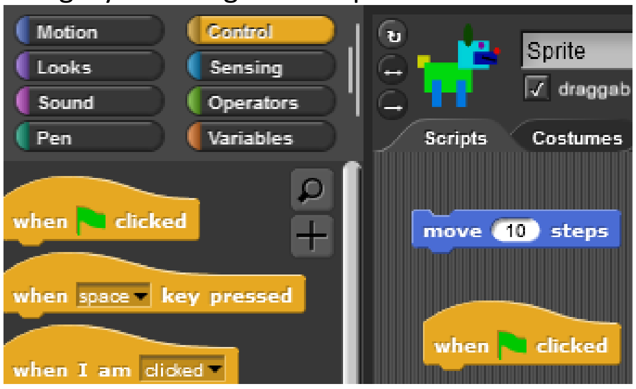
These blocks are colour-coded, which means that for example all the motion blocks are written on blue blocks.
To make your dog move forward, you first have to drag-and-drop the blocks from the left part of the interface to the Scripts section:

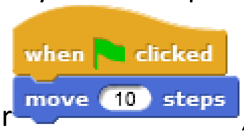


If you now double-click the

block, the dog will move for 10 steps.
You probably don’t want to double-click the blocks all the time, therefore click on Control category and drag-and-drop 

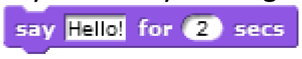
into the Scripts section:



These blocks work as bricks, which means that you can be put together to build a sequence of commands. If for example you put together 

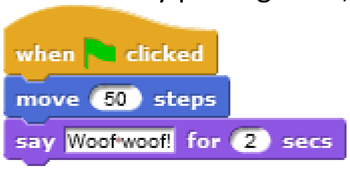
, your dog will wait for you to click on green flag on top of the stage and will then move 10 steps. You can change the number of steps dog makes by simply writing a different number in the white space provided. Try it out!

Barking

If you want your dog to bark like in comics, you have to click on Looks category and choose 

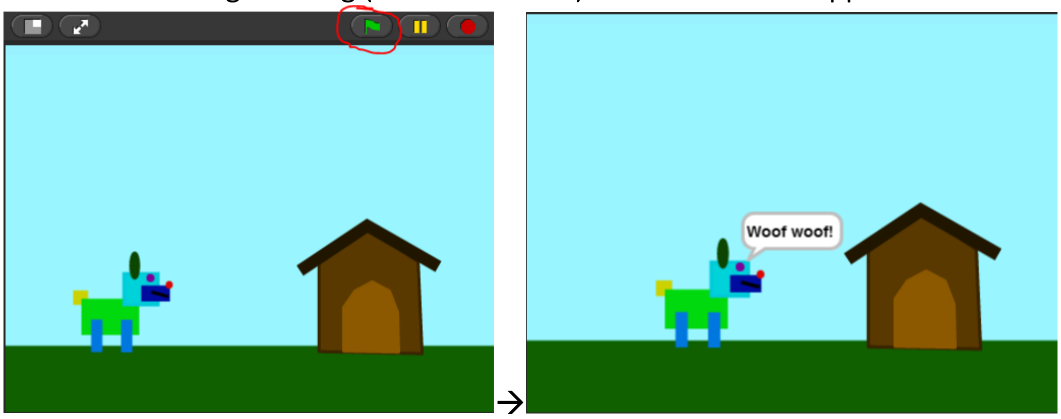
```
say [Hello!] for (2) secs
```

. Drag-and-drop this blocks to your scripts and add it to the code you have already put together, and replace "Hello!" with "Woof woof!"

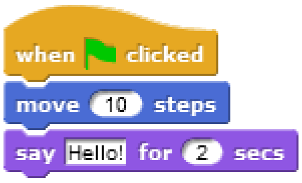


```
when flag clicked
move (50) steps
say [Woof woof!] for (2) secs
```

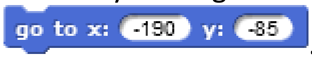
Now click on the green flag (circled with red) and this should happen:



When you click on the green flag, the dog moves 50 steps, and says "Woof woof!"
Try to write a program that will lead your dog from the left part of the stage to the doghouse on the right side of the stage and bark after every move. When it reaches the house, it barks "I'm home". Use blocks:

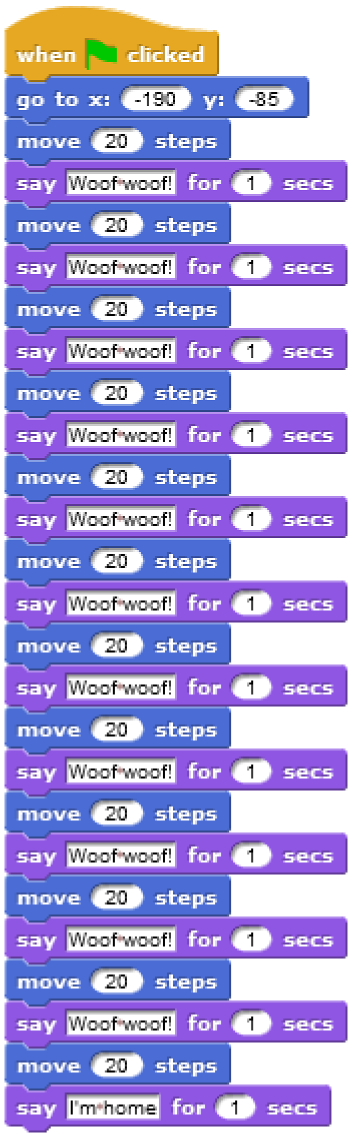


```
when flag clicked
move (10) steps
say [Hello!] for (2) secs
```

Tip 1: you can move your dog to a selected position on stage by drag-and-dropping it
Tip 2: If you want your dog to start on the same position every single time you click on green flag, use 

```
go to x: (-190) y: (-85)
```

. You can choose your own x and y position by writing a different number in the white spaces.
Possible solution:



```
when green flag clicked
go to x: -190 y: -85
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say Woof woof! for 1 secs
move 20 steps
say I'm home for 1 secs
```

Scenario 3 – Moving around the stage

[Task 1]

1) Open *Catch_the_Ball* and add code to the dog so that it catches the ball. With this you will create an animation, where you will see how the dog moves.
2) Use `change x by` and `wait secs` blocks to make an animation.
3) With click on green flag dog is moved to the starting position with `go to x: y:`
4) When you finish, don't forget to save the program.

Catch the Ball:
https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Catch_the_ball

[Task 2]

1) Open *Help monkey climb the tree* add code to the monkey to fetch the bananas.
2) Use `change y by` and `wait secs` to make an animation of a monkey climbing on a palm tree.
3) Move monkey to starting position.
4) When monkey reaches bananas, it shall climb back to the starting point. The movement should look like an animation.
5) When you finish, don't forget to save the program.

Help monkey climb the tree:
https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Help_monkey_climb_the_tree

[Task 3]

In both tasks you had to interchangeably use two blocks. How many times did you have to **repeat the code?**

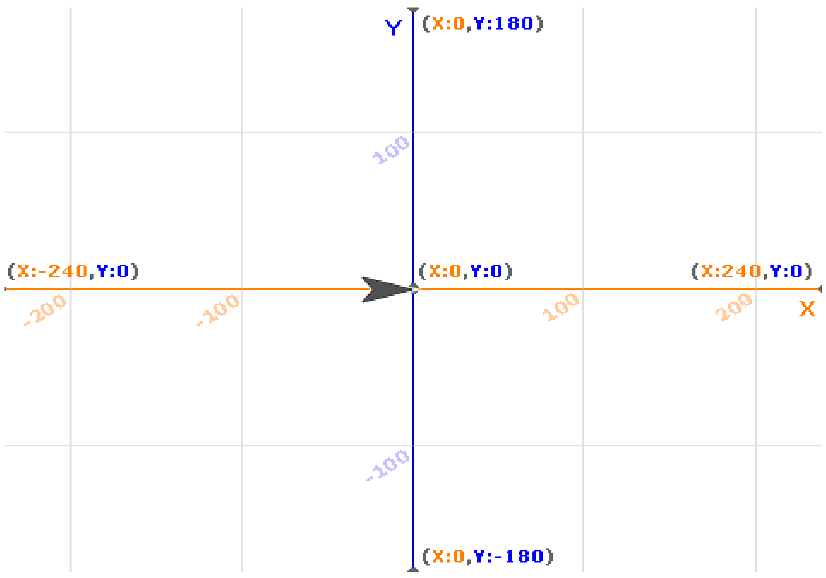
There is a shorter way of writing this code by telling the computer to **repeat** your code a given number of times.

1) Find a suitable block of code, that enables you to repeat the code.
2) Update your code for the monkey with the block you have found.
    a. Make a code for monkey to climb up the tree
    b. Make a code for monkey to climb down form a tree (to the starting position)
3) Update the code for the dog
    a. Make a code for the dog to run to the ball
    b. Make a code for the dog to return
    c. Additionally: try to figure out how to turn the dog around, so it doesn't run backwards

The picture below shows the *stage*.

For moving left/right you move in x direction (right +, left -)

For moving up/down you move in y direction (up +, down -)



Remark:

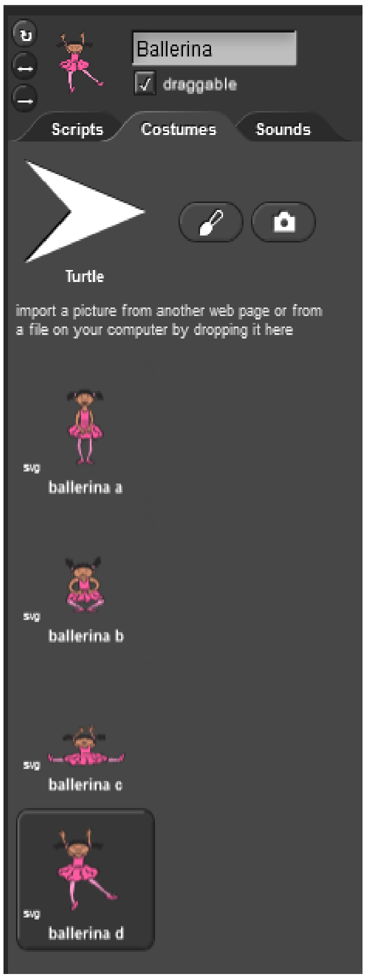
For moving right or up, you ***never*** *write the* ***+*** *sign in front of the number, e.g. it is* ***not +240****, it is just* ***240****.*

Picture : XY Grid Background to help you with moving in x and y direction

Scenario 4 – Changing costumes and turning

[Task 1]

1) Open a new empty project, click on *icon* that looks like a *white piece of paper*, and select *Costumes…*
2) Click on **ballerina a**, and click on *Import*. Do the same with *ballerina b*, *ballerina c*, and *ballerina d*. Then click *Cancel*



In Costumes tab of your sprite, you now have 4 ballerina costumes.

You can rename Sprite to Ballerina, by changing the text above the Costumes tab.

3) Now go back to Scripts tab and try to create a code, that:

a. will start when the green flag is clicked

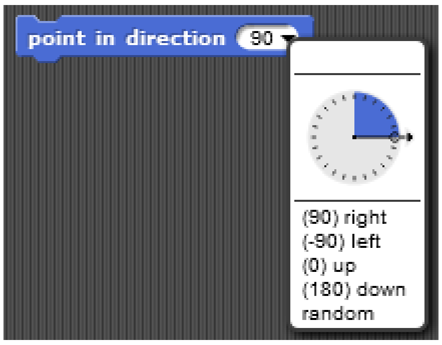
b. our ballerina will dance so that she will change her appearance 15 times. Use next costume and wait ( ) secs.

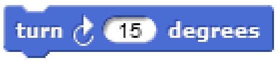
c. Character ends her dance by changing appearance to *ballerina a*.

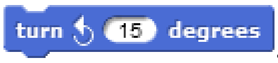
[Task 2]

a. Open a new empty project. Repeat all the steps from [Task 1], except that you import *avery walking a*. As before add also costumes *avery walking b*, *avery walking c* and *avery walking d*.
b. Add a suitable background for Avery to walk on, so that in animation it will seem as if Avery is walking from left side of the stage to the right side of the stage.
c. Create an animation of Avery walking. The code includes:
    a. Start when green flag is pressed
    b. Starting position
    c. 14x repeating the change of costumes. Don't forget to add wait _ secs block to see the animation.
    d. The girl is now walking on spot. Punca sedaj hodi na mestu. Try to figure out, how to animate Avery in a way, that her steps will look connected as in real life and she will move from left to right.

[Task 3]

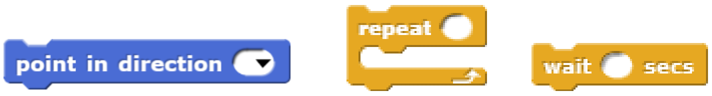
1) Open a file Find cheese.
2) Until now, you always wrote a program where a sprite only moved in one direction. In this task, you will have to turn the mouse, in order to reach the cheese. To make her turn, you can either choose:
    a. 

    where you tell the mouse in which direction she has to point or
    b. you can tell her to turn for a certain angle clockwise 

    or counter clockwise 

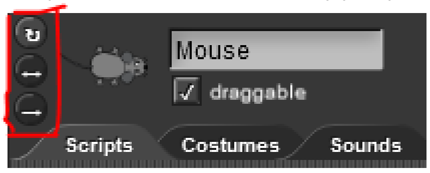
    .
    A full circle has 360 degrees, so if you want to turn in the opposite direction from where you are now, you turn for 180 degrees. If you want to turn to your left you turn 90 degrees counter clockwise. If want to turn to your right you turn 90 degrees clockwise.

3) Write a program that mouse has to follow to reach the cheese if she has to walk only on the green area
    a. Use the following blocks:



b. Now try to write a program using `turn ↺ 15 degrees` and `turn ↻ 15 degrees` with 90 degrees.

4) As you have seen, the mouse has turned in different directions to reach the cheese. Sometimes you don't want your sprite to turn upside down, but to just turn to the left or to the right so it doesn't walk on its head. To make sure your sprite turns like you want it to, you have to click on appropriate icon left of your sprite:



   a. The *circular arrow* means, that your sprite can turn in any direction (like your mouse)
   b. The *left-right arrow* means that your spirit will only turn to the left or to the right (this is what you would use for the dog not to walk on its head
   c. The *right arrow* means that the sprite will always look as it is (you could use this for the monkey)

c. Try to rewrite your programs for the dog and the monkey so that they first go the object and back by turning. Make sure you change their rotation style properly.

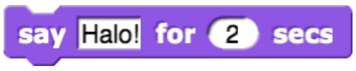
Find cheese:
https://snap.berkeley.edu/snap/snap.html#present:Username=spelac&ProjectName=C4G_Find_cheese

Scenario 5 – Sounds of the farm

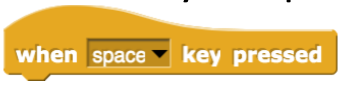
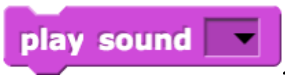
1. First you have to open the *Sounds of the Farm* program. In it you find a program template with background and the main character – the woman farmer.

2. There are different animals on the farm that are advertised under certain conditions. How the animals advertise tells us the woman farmer. **Now you have to program the woman farmer to tell the instructions: "If you want to hear the dog, click on the key "D"!".**

   You have to do the same for other farm animals.

   TIP: Help yourself with the block 
   say Halo! for 2 secs
   .

3. If you want animals on the farm to be advertised, you need to add sound to them.
Now you have to add a sound from the sound library and program the sound of the dog which will be played when the key "D" is pressed.

TIP: Help yourself with the blocks 
when space key pressed
and 
play sound
.

4. You have to **import sounds** to other animals and give them a similar **code** as in Task 3.
   TIP: You can import sounds by dragging them to the Sounds tab. Help yourself with blocks from Task 3.

5. Next step is to **program the woman farmer's welcome greeting**. When player start the game the woman farmer has to say: "Welcome to my farm". First, you have **to record** the woman farmer's welcome greeting and than you have **to add the sound** in woman farmer's scripts.

   TIP: You can record sound by clicking the red button on the Sounds tab.

[**ADDITIONAL TASK**]
You can upgrade the farm as you like by adding new sprites (farmer, hen, tractor, ...) and sounds.
WHEN YOU FINISH, **SAVE THE PROGRAM**!
Sounds of the Farm:
https://snap.berkeley.edu/project?user=tadeja&project=Sounds%20of%20the%20farm_0

Scenario 6 – Chameleon’s summer vacation

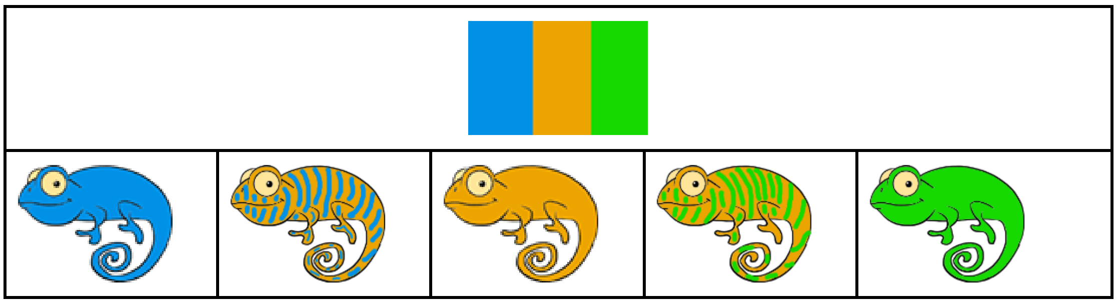
Task: Create a game where the player will be able to move the main character - chameleon around the screen with arrow keys. Chameleon will change its looks (costumes) based on the color of the background. Background is divided in three parts with uniform color, each representing a different place: blue represents the sea, sandy color for the beach and green for the forest.

Chameleon has to change its looks and also tell where he is in five different situations:
1) when swimming in the sea, he has to change his color to blue and say “I am swimming in the sea”;
2) when he is between the sea and the beach his skin turns half blue-half sandy color and he says “I am between the sea and the beach”;
3) on the beach, he takes on a sandy color and says “I am relaxing at the beach”;
4) between the beach and the forest, he turns half green-half sandy color and says “I am between the beach and the forest”;
5) in the forest, his skin turns green and he says “I am cooling off in the tree shade”;

The process of creating the game:

1. Open the template file:
   https://snap.berkeley.edu/project?user=zapusek&project=chameleon_template

   There is a background divided into three unicolor parts. You will also find five different looks for a chameleon. You can optionally place on a background other items to make a scene more realistic, such as: waves, sea shells, trees… You have to be careful not to choose items that are entirely different color than background and are at the same time bigger than the chameleon. In this case the sensing block won’t be able to correctly detect the color of the background if the player will place chameleon there.



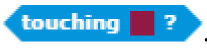
2. Write a code for moving the chameleon with arrow keys (up, down, left, right). Set his step length to 10 pixels. Remember to rotate him in the appropriate direction before each step. Chameleon should not move over the edge of the screen so use the block `if on edge, bounce` to make him bounce back.

3. Chameleon changes his looks according to the color of the background at his current position. We can move him freely across the screen, so it is impossible to predict his position beforehand. This is the reason why we have to check his position repeatedly. In this kind of situation where we have to constantly check if something has happened we use a forever loop.

```
forever
```

4. What are the possible locations of the chameleon on the screen? We noticed there are five:

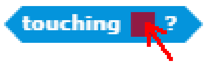
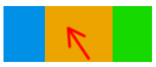
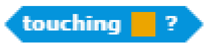
a. he is entirely on the blue color
b. he is entirely on the sandy color
c. he is entirely on the green color
d. he is partly on the blue color and partly on the sandy color
e. he is partly on the sandy color and partly on the green color

5. In *Snap!* we have a special block that tells us what color the object is currently touching. We can find it in the Sensing group: 

.

6. This block gives us data about whether it's: *true* or *false* that object is touching a specified color. We can specify a color by clicking on a little square and then select a color. We can choose it from a color palette or by clicking on a part of the screen with desired color.

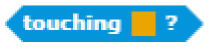
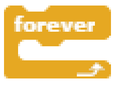
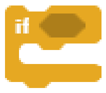
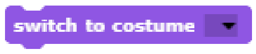
   Below is the example showing us how to get data: *true* or *false* from a sensing block if object is touching the sandy color:

7. Block

is not a command block. It represents a logical expression that can be put into a control block. Control blocks execute other blocks (that we put in their body) only if the value of the logical expression (in their head) is *true* and don't do anything if the value is *false.* Examples of control blocks are: if block, repeat until loop and event "When". In our case we will have to use if control block.

8. Combine blocks below to check if the chameleon is touching the sandy color. In that case switch to a sandy chameleon look:

9. Now you can use similar blocks and then combine all of them together to also check the other two conditions: if the chameleon is touching blue and green color.

10. **Remember!** If there are multiple if blocks with fulfilled condition in a sequence, only the commands of the last one will have the effect in the game. Use that knowledge when creating the code.

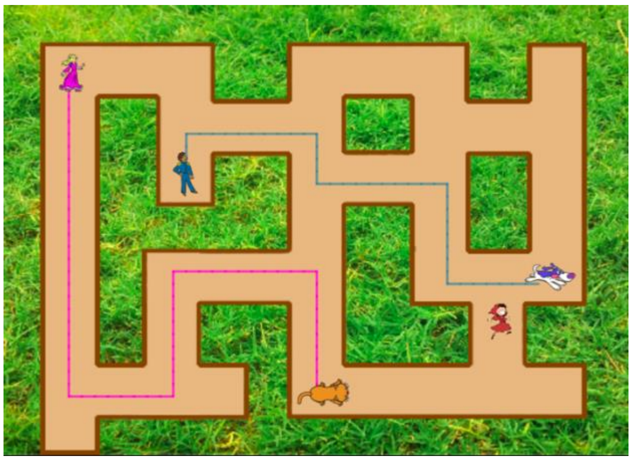
11. Now we have to take into consideration other two cases that can happen in game, that is when chameleon is touching two colors of the background at the same time. If you want to check if multiple conditions are true at the same time you have to combine them using a logical operator "and". You can find logical operators in the "Operators" group.

12. Combine two blocks for sensing color: 2x `touching ?`, logical operator: and `and`, if block: `if` and block for switching costumes: `switch to costume` to implement detecting those additional cases.

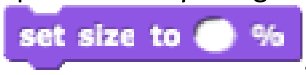
13. Complete the game with the use of block `say Hello!` to program a chameleon to say on which part of the screen is he currently positioned.

Chameleon summer vacation: https://snap.berkeley.edu/project?user=zapusek&project=chameleon

Scenario 7 – Helping Prince and Princess to find their animals

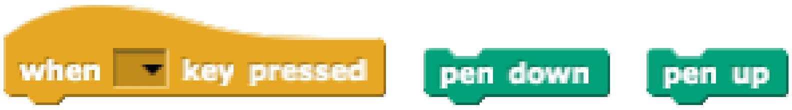


1) Open the program *Helping Prince and Princess to find their animals*. You will add sprites, write a code for the girl's movement and drawing a path.
2) Create 4 new sprites (a Prince, a Princess and eg. a dog and a cat). Reduce the sprites' size by using this block

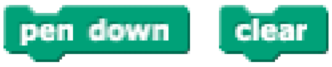
   ```
   set size to ( ) %
   ```
   .

   The sprites have to be small enough to move inside the maze (as shown in the picture).
   Put the sprites in the desidered position in the maze.
3) Now you will write a code for girl's movement with the keys. You already have the code for moving with the right arrow. Write a code for moving in other three directions.
4) The girl can only move along the path, so you have to stop her from stepping on the grass. Think about how will you do this.
5) Now you will write a code for drawing with a pen.
   Use these blocks:

   ```
   when [ v] key pressed    pen down    pen up
   ```

   Then move the girl around the maze and you wll see what happens.
6) The player will have to connect a Princess and her cat with one color and a Prince and his dog with another color. Think about how will you do this.
7) In the last step you have to think how will the game begin. What will happen when the green flag is clicked?
   a. The girl will move to her starting position.
   b. Then player who will play the game does not know that he has to draw a path from a Princess to her cat with one color and from a Prince to his dog with another color and that the paths must not cross. Write the instructions that will tell all this.
      Make sure that the player has enough time to read the instructions.
   c. When creating a game, we should always test it and look for possible errors. Repeatedly start the game by clicking on the green flag and see if everything works as it should.
   d. If/when you find out that something is missing in the code, think about what the next blocks do and where you need to put them.

      ```
      pen down    clear
      ```

[Additional tasks]

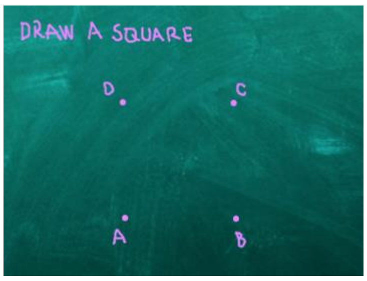
You can add additional tasks according to your wishes or follow the tasks below:

- Set starting coordinates for the Prince and the Princess and write a code for their movement. Set the appropriate size for them. They should draw a path to their animals.
- Add another sprite (animal) for the girl.
- Each sprite should draw with a different color.
- Adjust the initial instructions.
- Add instructions for moving a sprite and drawing by clicking a sprite. E.g. the Princess says: "You move me with pressing the keys W, S, A and D. I draw the path by pressing the key 3. I stop drawing by pressing the key 4. Help me to find my cat!"

Helping Prince and Princess to find their animals:
https://snap.berkeley.edu/project?user=mateja&project=Helping%20Prince%20and%20Princess%20to%20find%20their%20animals%20-%20Part

Scenario 8 – Drawing with a chalk

[Task 1]



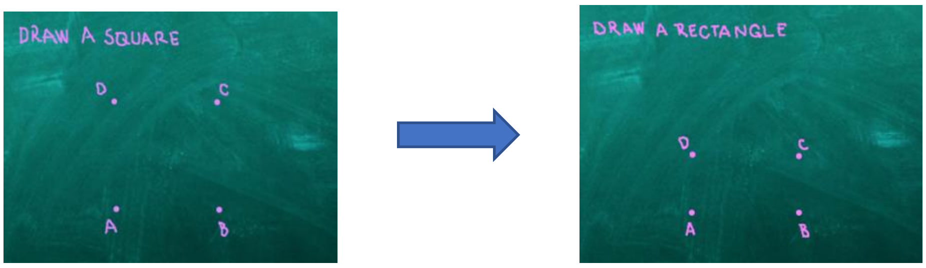
1) Open *Drawing with a chalk* and write a code so that the chalk will draw a square by pressing the »S« key.
2) The chalk has to connect the vertex A and B, B and C, C and D and D and A.
   Hint: The distance between A and B is 150 steps.
   Every angle of a square is 90°.
   Use `wait ( ) secs` so you can see the chalk's movement.
   If the steps are repeated, use the *loop repeat*.
3) Before using the loop, we have to add a following code:
   a. Set pen color.
   b. Set starting coordinates for the chalk in vertex A (x: -80, y: -105).
   c. Put the pen down.

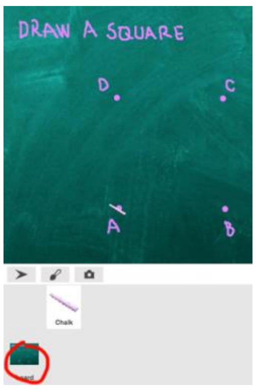
   Consider whether it makes sense so use blocks *pen up* and *clear* as in the previous activity and where to put them.

   Why is good to use the block `point in direction (90 ▾)` ?

[Task 2]

1) In this task you will write a code for drawing a rectangle. Firstly, you will have to change a background.

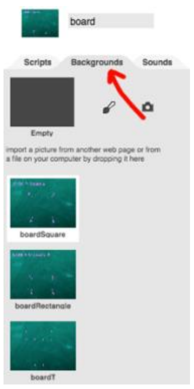




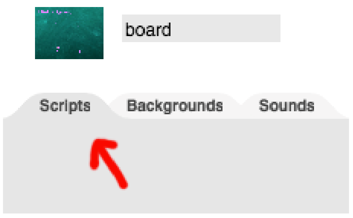
Clicking on »board« (left picture) you open backgrounds.

Clicking on »Backrgounds« (right picture) you can see 3 prepared backgrounds for this activity:

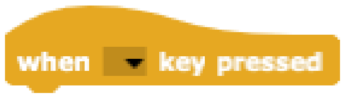
boardSquare, boardRectangle and *boardT*.

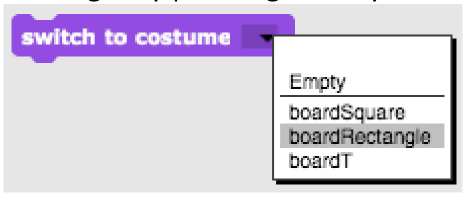

board
Scripts
Backgrounds
Sounds
Empty
import a picture from another web page or from a file on your computer by dropping it here
boardSquare
boardRectangle
boardT

2) For writing a code click on Scripts.



We want switch the background to boardRectangle by pressing the key "R".

Use blocks 

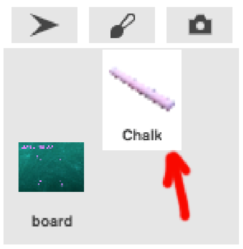
and 

.

Do the same for switching backgrounds to *boardSquare* with the key "S" and to *boardT* with the key "T".

3) By clicking on the Chalk you go back to writing code for the chalk.

You have to add one more block of code to the [Task 1].



You already wrote a code for switching to boardRectangle by pressing the key "R", but the player does not know that so you have to tell him. For writing instructions use block *say*.

Now you can continue with writing the code for a rectangle. You will use a similar procedure as when drawing a square. Use the loop when possible!

Same hints:

a. Vertex A has the same coordinates as before x: -80, y: -105.
b. All angles are 90°.
c. The distance between the vertex A and B is 150 steps and between the vertex B and C is 75 steps.

Add instructions for switching to a new background.

[Task 3]

1) Here you will connect the vertices to the letter T.
2) Some hints:
   a. Coordinates of the starting vertex are x: -56, y: -138.
   b. The distances between vertices are 60, 185 and 180 steps.
   c. All angles are 90°.
3) Add instructions for playing again from the beginning.

[Additional tasks]

You can add additional tasks according to you wishes or follow the tasks below:

- Add a new background and draw some dots.
- Write a code that connects the dots. You can draw a background or you can use a given one.

Drawing with a chalk:
https://snap.berkeley.edu/project?user=mateja&project=Drawing%20with%20a%20chalk%20-%20Part

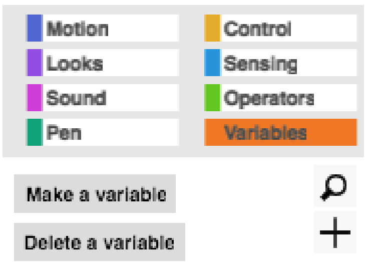
Scenario 9 – Picking up trash and cleaning the park

1) Open the program *Picking up trash and cleaning the park*. The code for moving the girl is already made. Sprites for waste (a bottle and a paper) and a trash bin are also given.
   You will create more sprites (waste) that a girl will have to pick up and dump into a trash bin at the end.
2) Select the starting position for the girl and set the x and y coordinates. You can move the trash bin if you want (trash bin will always be in the same position, so you do not have to set the starting coordinates).
   As always, write initial instructions.
3) In order to check if the girl picked up all the trash, we have to count how many items has she picked up. To help with the counting, we will use a **variable**.

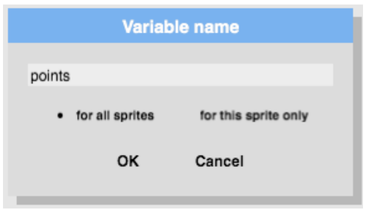
What is a variable?

A variable is like a box where we store some information. The name variable derived from the fact that its value may vary during the implementation.
We will make a variable *points* and with *points* we will count how many items the girl picked up.

How do we create a variable?



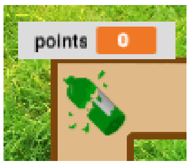
We click on the orange block *Variables*, then we select *Make a variable*, write its *name* and click *OK*.



IMPORTANT! The name of the variable should:

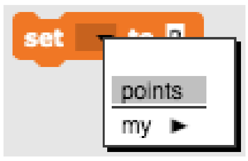
- make sense and name what the variable will represent – eg. *points*, *noOfTrash* etc.
- **not** contain non-English characters (eg. **č, š, ž** etc.),
- **not** include **spaces.** If we want to name the variable number of trash, we can name it *number_of_trash*, *numberOfTrash* or shorter *no_of_trash* or *noOfTrash* etc.

When we make a variable,
the variable appears on the left.

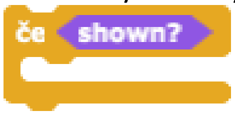


A check mark in front of a variable indicates that the variable name and value appear in the background.

Since we have not picked up any trash at the beginning, the value of the variable *points* should be 0. We set this with this code:

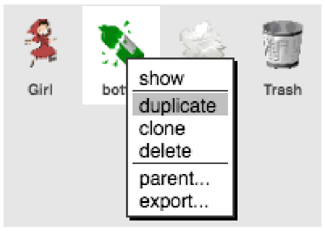


4) Now you will write a code for a *bottle*. When the girl touches the bottle, the bottle will disappear.
   Think about:
   a) How can you check if the girl came to the bottle?
   b) What happened to the bottle, when the girl touches it?
   c) What happens with points?
      Test if number of points increases correctly.
   d) Hint: Why would you use the following command?



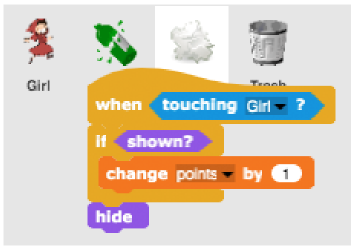
   e) Test again, click on the green flag. What happens?

5) When a code for one bottle is finished, you can copy the sprite bottle.



Right click on the bottle and select duplicate. A duplicated sprite appears somewhere in the background. Move it somewhere inside the maze. Copy the sprite bottle several times, so you will have more trash in your maze.

6) Now you have to write a code for a sprite *paper*. The code is the same as it was for the bottle, so you can simply copy the entire code. Left click on the entire code, drag it on the sprite *paper* and drop it.



   Copy the other parts of the bottle code in the same way.

7) Repeat the step 5) and copy the sprite *paper* like you copied the *bottle*.

8) The last thing is a code for the *trash*.
   When the girl comes to the trash bin, the trash bin will tell her if she collected all the trash or not.

[Additional tasks]

You can add additional tasks according to you wishes or follow the tasks below:

- Add another type of waste (e.g. bio-waste).
- The trash can says e.g. “You picked up X bottles, Y papers and Z watermelons”.
- If a player picks up all the trash, the trash can says: “Congratulations! You picked up all the trash!”
- If a player does not pick up all the trash, the trash can tells him which trash has not been picked up, e.g. “You did not pick up all the bottles. You did not pick up all the watermelons.” and “Come back when you pick up all the trash”.

Picking up trash and cleaning the park:
https://snap.berkeley.edu/project?user=mateja&project=Picking%20up%20trash%20and%20cleaning%20the%20park%20-%20Part

Scenario 10 – Feeding the cats

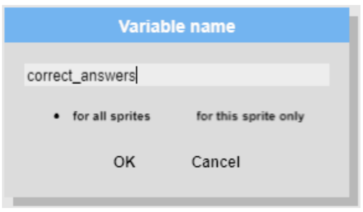
Task: Program the game in which the shelter keeper will repeatedly ask the player for the number of cats she can feed in a certain room. The number depends on the number (2 to 10) and size (2 to 5) of the bowls. For each room those two numbers have to be assigned randomly. The size of the bowl tells how many cats can eat from it, for example if bowl size is 3 that means 3 cats can eat from it.
We also have to have a counter that will count the right answers. Create a game in which the player will have to guess the right number of cats that we can feed in each room. After the activity give a feedback about how many times the player answered correctly and how many times she was wrong.

The process of creating the game:

1. Open the template file:
   https://snap.berkeley.edu/project?user=zapusek&project=cat_feeding_template

   There is an image for the background and the main character - cat shelter keeper.

2. We want to count the correct answers. If we want to store a value in a program we have to use a variable. We can access commands regarding variables in the "Variables" group. New variable is created when we click on: Make a variable .
   Variables assign a name to a value, so when we make a new one, we have to name it first. It is suggested that we use a mnemonic name, this means that we can tell what kind of value is stored in a variable from its name. For counting correct answers we can choose the name "correct_answers".



Variable name

correct_answers

• for all sprites   for this sprite only

OK   Cancel

The value of the variable can be set to or changed by some value. If we want to set the value of the variable we use the block: set ▾ to 0. Everytime we set a value to a variable, the previous value is overwritten.
If we want to change the current value by some other value, we have to use block: change ▾ by 1. Current value of the variable will be changed by the value we specify in a white space.

3. Now we will start to actually code the game. First think about how activities in each room will actually be very similar, the same even. In each room we have to:
   a. assign a random value for the number of bowls and for the bowl size.

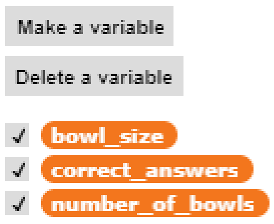
      In the “Operators” group we can find the block that returns the random value from an interval that we can specify. For example: `pick random 1 to 10` will return the random number between 1 and 10.

      We will need the values of the number and size of the bowls in each room stored for later when we will have to calculate the right answer. This can be achieved by creating new variables and assigning those values to them.

      Create two variables:
      1) for the number of bowls in each room
      2) for the size of the bowl in each room

      Assign them random values according to the specification of this game. Use the blocks: `set ▾ to 0` and `pick random 1 to 10`.

   b. Player has to know which random values computer selected, so we have to inform her. Use blocks: `say Hello! for 2 secs`, `join hello world ◂▸` and the values of the variables that store the number and size of bowls: `number_of_bowls` and `bowl_size`
      .
      You can find references to variables in the “Variables” group:



      Make a variable
      Delete a variable
      ✓ bowl_size
      ✓ correct_answers
      ✓ number_of_bowls

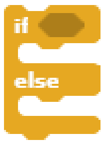
   c. Now we have to prompt the player to input her answer. If we want to get the player input we have to use input block from “Sensing” group: `ask what's your name? and wait`.
      When the player write her answer into input field it is stored in the “answer” variable: `answer`.

   d. Next we have to check if the player’s answer is correct or not. Think how you can calculate the right answer from the number and size of the bowls. If we compare the value stored in `answer` with the number of cats we can feed (we calculate this value), we can find out if the answer is correct or not.
      Calculate the correct number of cats we can feed in each room using these blocks: `( ) × ( )`, `number_of_bowls` and `bowl_size`.

      i. if the answer is correct: congratulate the player and add one to the value of the correct_answer variable.
      ii. if the answer is wrong: give the appropriate feedback.

         We can differentiate between exactly two possible situations with the use of if-else block:

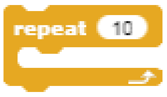
Med dvema situacijama (pravilen ali napačen odgovor) ločimo z uporabo bloka za pogojni stavek "če-sicer".



If the answer will be correct it will be equal to the calculated value, otherwise it won't be. Complete the code using the block that checks equality:

.

4. We implemented the code for one room, but there are ten rooms in a shelter. We could copy the above code ten times and place it sequentially but that would not be an optimal way to do it. Instead of doing that we can use a loop that will repeat the same code ten times. The most simple loop for achieving that is repeat [n] times loop:



5. When the game is over you have to provide the feedback: number of correct and wrong answers. The number of correct answers is stored in a variable and the number of wrong answers can be calculated. If we know, that player will input the number 10 times and we also know how many times he answered correctly, we can find out how many times she was wrong. Complete the game using these blocks:
6. Na koncu izpiši število pravilnih in napačnih odgovorov. Število pravilnih je shranjeno v spremenljivki *tocke*. Napačne pa lahko izračunamo. Če vemo, da je možno zbrati največ deset točk, prav tako pa kolikokrat je igralec odgovoril pravilno, lahko število napačnih odgovorov izračunamo:
say Hello! for 2 secs
,

and
correct_answers
.

7. You can upgrade the game using these suggestions:
   a. Shelter keeper includes the number of each room in her question. For example: "Guess the number of cats I can feed in room 5".
   b. If the player's answer is wrong, she tells her the correct number.
   c. Instead of ten rooms, the room number is random or the player inputs the number of rooms at the beginning of the game.
   d. Add your own ideas and upgrade the game to your likings.

Scenario 11 – Guessing the number of cats in a shelter

Task: Cat shelter keeper Martha wants you to guess the exact number of cats that she has in her shelter. The number can be anywhere between 1 and 100. When the player types the number she answers if current input number is less, more or equal to the right number of cats.

1. Open the template file:
   https://snap.berkeley.edu/project?user=zapusek&project=cats_in_a_shelter_template

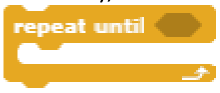
   There is an image for the background and the main character - cat shelter keeper.

2. First we have to randomly choose a number of cats in a shelter. The number must be selected from the interval from 1 to 100. We will need that value later in a game for comparing it to the player's answers, so it has to be stored in a variable.
   Create a new variable *number_of_cats* and assign it a random value from 1 to 100. This number will represent a number of cats in a shelter.
   Use the blocks: `set [ ▾] to 0` and `pick random 1 to 10`.

3. Think about: 1) which actions will be repeated in a game and 2) how many times we will have to repeat those actions. Can we predict in advance how many guesses will the player have to make in order to figure out the correct number?

4. The following actions will be repeated in a game:
   a. Player will enter a number. We can get an input from a player with the use of `ask what's your name? and wait` block.

   b. We have to check if the number is:
      i. Greater,
      Combine the blocks below to find out if the entered number is greater than the value stored in a variable *numbe_of_cats:*
      `if`, `[ ] > [ ]`, `answer` and `number_of_cats`.
      ii. Smaller,
      Combine the blocks below to find out if the entered number is greater than the value stored in a variable *numbe_of_cats:*
      `if`, `[ ] < [ ]`, `answer` and `number_of_cats`.
      iii. Same?
      To detect if the player answered correctly we will use a small "trick" that will be explained in the next chapter.

5. How many times will the code from 4) repeat? Well..until the player won't guess the right number. Think about how we cannot predict how many tries a player will need to guess the right number. In such situations where we have to repeatedly execute the same actions until a sentinel event occurs (in our case, the sentinel event occurs when the correct number is entered), we use a repeat until loop.


"Repeat until loop" will repeat all the blocks in its body until the condition in its head is true. If the condition is evaluated as false it will go into another iteration. When the "repeat until loop" exits, blocks that are placed below will be executed.

   In our case we will have to ask a player to input her guess and compare that value to the value stored in the number_of_cats variable. We will have to stop when the player's input will be equal to the number of cats, if not, we will have to do it again.

   Define the condition for checking if the input is equal to the number of cats using these blocks: [ ] = [ ], answer and number_of_cats.

6. **Let's summarize:** What will happen if the entered number will be different than the number of cats? Loop will go into its next iteration.

7. **Let's summarize:** What will happen if the entered number will be equal to the number of cats? The loop will stop and the blocks placed below the loop will be executed.

8. This is the "trick" we were talking about earlier and that enables us to detect the right answer without checking the condition explicitly inside the loop.
   Let's see what will happen when the player enters the right number. The two conditions inside the loop that check if the value if greater or smaller will not be satisfied and the loop will go check the condition in its head. This condition will be true, so the loop wont go into the next iteration and will stop executing. The program will start to execute the blocks placed below the loop.

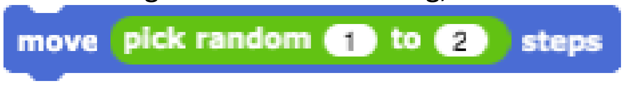
9. If we know that "repeat until loop" is behaving like this, we can use that knowledge to our advantage. We can conclude that if the blocks placed below the loop are executing, the player must have guessed the right number. Then we can congratulate the player.

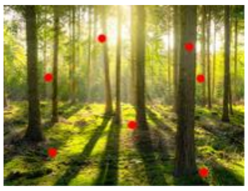
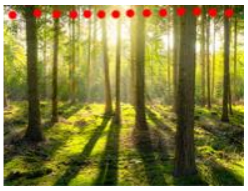
10. In Snap! we can show or hide the value of the variable to the payer. This can be done by clicking on the checkbox next to the variable name: ✓ number_of_cats .
    Consider if it is good to have the value of the *number_of_cats* variable visible to the player?

11. You can upgrade the game using these suggestions:
    a. Count the guesses.
    b. In the beginning ask the player to enter her name and greet him. Use her name when giving feedback or asking to input the next try.
    c. Give feedback that will depend on the player's success. If the player guesses the correct number in five or less tries, give her a cat as a reward. If she guess the correct number on the first try, provide a special feedback..
    d. Add your own ideas and upgrade the game to your likings.

Scenario 12 – Catching healthy food

1) Open the program *Catching healthy food*: https://snap.berkeley.edu/project?user=mateja&project=C4G12_Catching%20healthy%20food%20-%20Part.
A bakground and a sprite (Girl) is alerady given. You will create a game, where healthy (+1 point) and unhealthy (-1 point) food will fall from the top. The player will have to click on the healthy food and collect a certain number of points. The girl will tell the initial instructions and then she will hide. The instructions have to tell that the game continues by pressing the key »S«.

2) Add a food sprite. Choose one healthy sprite, eg. an *apple*.
   a. Write a code for apple's movement. Think about the direction of movement.
   b. To make a game more interesting, instead of *move 2 steps* use
   
   ```
   move (pick random 1 to 2) steps
   ```
   .

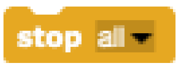
3) Think: what will happen when the apple comes to the bottom of the screen?
What do the pictures below mean?



4) What do you need for counting points? Make it and set it properly (write a code on the girl sprite).

5) How will you realize that the apple moves constantly?
Hint: The game ends when the player reaches e.g. 5 points.

6) When girl talks at the beginning, we want that the apple stays hidden. When the girl reappears, the apple hides again.
Think: why do we have to hide an apple on *when the green flag clicked*?

7) What happens when we choose (click on) a healthy food – an apple? Think and write the code.

8) The code for the apple is (almost) finished. Go back on writing the code for the girl. The girl will reappear when the player reaches 5 points and say e.g. »Congratulations, ...«. The program must constantly check if the player has reached 5 points. How will you do this?
Check – what does this block  `stop all` do and where will you put it?

9) When the player will play the game again, he will already know that he can skip initial instructions by pressing the key »S«. This will cause confusion as the girl will

still talk and the food will already fall.
You can prevent this by making a new variable (named e.g. *start*). At the beginning you set the *start*'s value to 0 (which means *food does not appear*). When the girl ends giving instructions, you set the *start*'s value to 1 (which means *food can appear*).

To make this work, you have to add a block of code to the sprite *apple*. What do you need to do?

10) In the last step you will duplicate the apple sprite few times to have more food. Change the sprite's costume so you will have healthy and unhealthy food (e.g. also a banana, a donut, a cake).

    The codes of healthy and unhealthy food are different only in one thing. Which one?

[Additional tasks]
Add additional tasks according to your wishes or follow the tasks below:

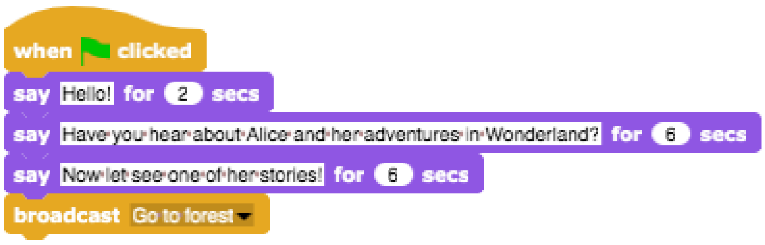
- Change the game so that a bowl sprite is catching food.
- Add a new sprite (a bowl). Draw it, find it online or use attached picture/s of the bowl.
- Set the starting position of the bowl (e.g. at the bottom of the screen) and write a code for the bowl’s movement (left and right, if you want also up and down). Food sprites have to disappear and reappear at a random location by touching the bowl (and not on mouse-clicking the food as before).
- Change the rules – let the game end when a player scores 20 points (he wins) or when he picks up 3 unhealthy foods (he loses).
- Add more food sprites to make the game more interesting.
- Change the bowl costume when a player scores e.g. 5, 10, 15 points.

Scenario 13 – Storytelling

1) Let's take a look at *Alice1*:
   https://snap.berkeley.edu/project?user=ddureva&project=Alice_1

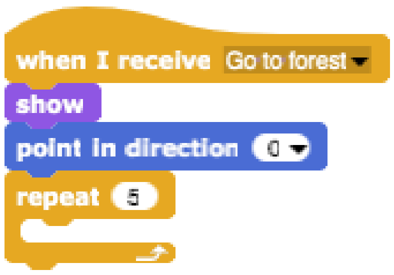
   and *Alice2* program together:
   https://snap.berkeley.edu/project?user=ddureva&project=Alice_2

2) We use broadcasting e.g. when we want that an event 2 happens after the event 1 ends.



```
when [flag] clicked
say [Hello!] for (2) secs
say [Have you hear about Alice and her adventures in Wonderland?] for (6) secs
say [Now let see one of her stories!] for (6) secs
broadcast [Go to forest]
```

E.g. at the beginning the Rabbit starts telling the story (event 2). When he ends, he sends the message *Go to forest.*

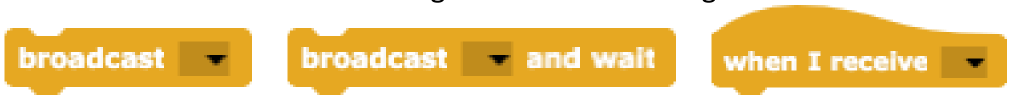


```
when I receive [Go to forest]
show
point in direction (0)
repeat (5)
```

When Alice receives the message *Go to forest*, the event 2 starts.

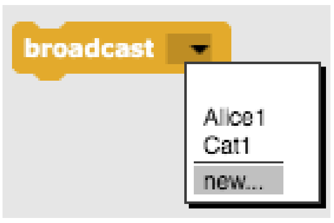
Shorter instructions for broadcast, receive and create messages

3) For broadcast and receive messages we have following commands:

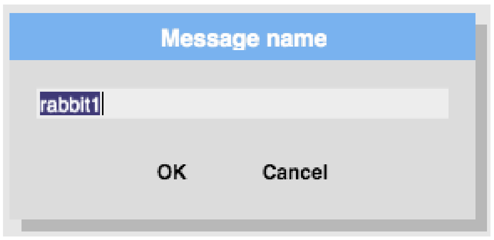


```
broadcast [ ]
broadcast [ ] and wait
when I receive [ ]
```

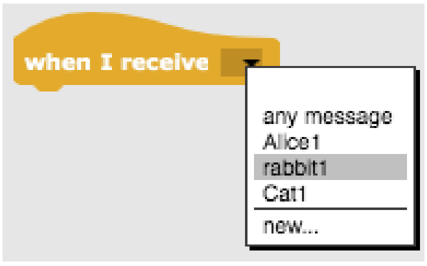
4) Creating messages:



Right-click on block *broadcast* and write the name of the message.



Then click on the event block *when I receive* and choose the message.



For creating a story, you always need a plot (scenario). In the table **Story plots/Scenarios** a scenario for the story is written and the second table **Sprites** provides a detailed scenario for each sprite / background.

Continue the program *Alice 2* and create a story!

Optionally, you can add some scenes and also change anything.

Story plots/Scenarios

Name	Design	Actions	Notes
1. Start		The story starts with the scene (When the green flag is clicked)	Against this background, the Rabbit introduces the story.
2. Forest		The scenery appears when the Rabbit rounds up his introduction (A *Go to forest* message has been sent)	Against this background, Alice appears positioned in the center of the stage. She starts moving, wondering *"Where am I?"*. The sprite gradually reduces its size 5 times by 10%. When it reaches the end of the path (at a crossroads), the scene changes to *Meeting*. (Alice sends message -broadcast *Meeting with Cheshire Cat*)
3. Meeting		Appears when *Alice's* message *Meeting with Cheshire Cat* is received.	Here Alice and the cat are part of the background. To use Alice's sprite, prior to the message, she is positioned so that she covers her image from the decor. The Cat sprite appears at a later stage. As the scene changes, the Rabbit continues to tell the story. Later a conversation takes place between Alice and the Cheshire Cat.

Sprites

Sprite	Actions	Background
Rabbit	At the Start: Says: Hello! (For 2 sec.) Says: Have you heard about Alice and her adventures in Wonderland? (For 6 sec.) Says: Now let's see one of her stories! (for 6 sec.) Sends the *Go to forest* message.	start
Alice	At the Start: Hides from stage; at centre stage position and 100% size, ready to be displayed against the new background.	start
Cat	At the Start Hides from the stage; positioned at x: -74, y: 113 (Positions are predetermined after the Cat sprite has been set on the *Meeting* stage.)	start
Alice	Receives a *Go to forest* message: The sprite shows on stage. Repeated 5 times: waiting for 1 sec .; moving 5 steps; size reduction (change by -10); wondering: *Where am I?* Preparing for next decor: waiting 5 sec; restoring the sprite's size (100% change) and positioning at x: -187, y: -67 Sends Message: *Meeting with Cheshire Cat*.	forest
Rabbit	No action. Just becomes visible from previous decor.	forest
Rabbit	Receives the message: *Meeting with Cheshire Ca*t. Resizes to 80% He says: *"Alice stops at the crossroads and wonders were to go."* (for 10 seconds). He says, *"She saw the Cheshire Cat on the three."* (for 8 sec.) Sends a message *Alice1*	meeting
Alice	Receives the *Alice1* message. Moves to the front (This is necessary because the Cat appears after her, which prevents Alice's lines from appearing in a dialogue box if she is not in the front layer). She says: *"Hi!"* (for 2 sec.) She says: *"Would you tell me please, which way I ought to go from here!"* (for 10 seconds). Sends a *broadcast* message to the Cat: *Cat1*.	meeting

	Receives the *Cat1* message. The sprite shows on stage. It says: *"That depends a good deal on WHERE you want to get to!"* (for 10 seconds). Sends an *Alice2* message.	meeting
	Receives the *Alice 2* message. Says: ………………………………………………………………………. Sends a *Cat2* message.	meeting
	Receives the *Cat2* message. Says: ………………………………………………………………………. Sends a *Rabbit1* message.	meeting
	Receives the *Rabbit1* message. Says: “What’s the moral of the story?” (for 8 sec.) Says: “To know which way to go, one has to determine his or her goal first.”	meeting

Scenario 14 – Improve the Climate

The air has been heavily polluted by the industry and needs to be cleaned. Program the game so that the player will improve the air by planting trees.

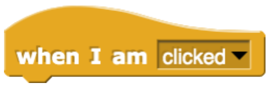
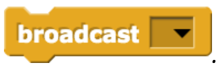
1. First **you have to open** *Improve the Climate* **program**. It contains template of backgrounds (industry and grass) and sprites (a pencil, a pine, an oak and sprite named clear).

2. Next you have to program that **the pencil will be positioned in the middle of screen** and that **the industry will be in the background** when the green flag is clicked.

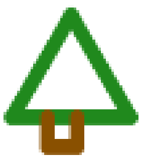
3. Now you have to add the code that **the pencil will draw a pine** when the player clicks it.

The following note:

- When the pine is clicked, **it sends a message** to the pencil to draw the pine.

TIP: Help yourself with the blocks 
when I am clicked
and 
broadcast
.

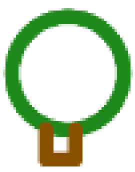
- **The pencil draws a pine** – you have to draw a green triangle and below it a brown square. You will need maths skills here.



TIP: Use the blocks in the Pencil tab.

- Once the pine is drawn, you have to **lift the pencil** and **place it in a random position**.

4. Next you have **to add the code for drawing oak** in the same way as in Task 3. When the player clicks in the sprite named oak, it is drawn to the screen. You have to draw an oak as a green circle and a trunk as a brown square. You will also need math skills here.



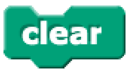
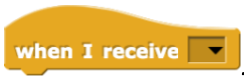
TIp: Help yourself with the steps in Task 3.

5. Next you have to add the code that **all the drawn trees are deleted when the player clicks on the *Clear sprite***.

   The following note:
   - When the *Clear sprite* is clicked with the mouse, it sends a message to clear all trees.

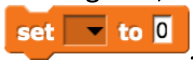
     TIP: Help yourself with blocks from TASK 3.

   - When the Pencil sprite receives a message, it deletes all drawn trees.

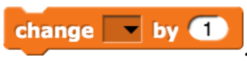
     TIP: Help yourself with blocks  *and* .

6. Next you have to add **a new variable named *Clean Air*** and add the code that the player will earn **2 points for each drawn pine and 3 points for each drawn oak** (the oak contributes more to the purity of the air than the pine). When the player reaches a certain points (eg 10), the air is no longer polluted.

   The following note:
   - At the beginning of the game, **the *Clear Air* variable must be set to 0**.

   TIP: Help yourself with block .

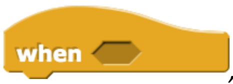
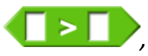
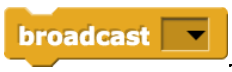
   - Each time the *Pencil* sprite receives a message for drawing pine, **the *Clear Air* variable must change by 2**.

     TIP: Help yourself with block .

   - Each time the *Pencil* sprite receives a message for drawing oak, **the *Clear Air* variable must change by 3.**

   TIP: Help yourself with the block above.

   - When the *Clean Air* variable is greater than e.g. 10, the *Pencil* sprite sends a message to background **to turn it into grass**. At that point the air is not polluted anymore.

   TIP: Help yourself with blocks , , .

[ADDITIONAL TASK]
You can upgrade the game by adding animals that they appear when the air is not polluted anymore.

WHEN YOU FINISH, **SAVE** THE PROGRAM!
Improve the Climate:
https://snap.berkeley.edu/project?user=tadeja&project=Improve%20the%20climate

Scenario 15 – Catch the mouse

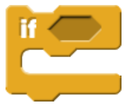
There is a mouse in Nina's room. The cat named Muri is released into the room with the intention of catching the mouse. Muri is unsuccessful in hunting. The mouse escapes several times to the cat. How many times has Muri almost caught the mouse?

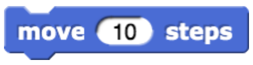
1. First you have **to open** *Catch the mouse* **program**. In it you find a template with background – Nina's room.

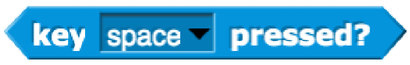
2. Next you have **to add a new sprites** – a cat and a mouse.

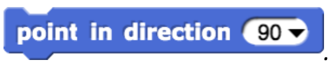
3. The cat is moved by the player with the arrow keys. The player tries to catch the mouse. Now you have **to add the code that the cat is moved by the arrow keys** (up, down, left, right). Also you have to consider what happens if the cat is on edge.

   TIP: Help yourself with the blocks







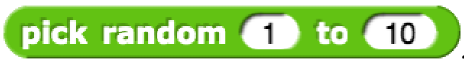


.

4. The mouse runs around Nina's room. You have **to program the mouse to move randomly**.

The following note:

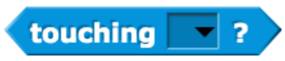
- The mouse moves by a random number of steps.
- The mouse turns by a random number of degrees.
- The mouse bounces if it is on the edge.

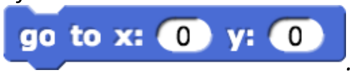
TIP: Help yourself with the block 

.

5. Every time the cat catches a mouse, it escapes. Now you have to program that **the mouse hides and appears in a random location in the room when it touches the cat**.

TIP: Help yourself with the blocks from TASK 4 and with the blocks



and 

.

6. We wonder how many times a player catches the mouse. You have **to add a counter** that **it will count the number of times the cat touches the mouse.**

The following note:

- You have to create a new variable named *Result* (you have to go to *Variables* and click on *New Variable button*).
- No mouse is caught at the beginning of the game, so the *Result* variable must be set to 0 (help yourself with block `set ▾ to 0`).
- Each time the cat catches (touches) the mouse the *Result* variable must be increased by 1 (*help yourself with the block* `change ▾ by 1`).

7. The game ends after a certain amount of time (e. g. after 30 s). You have **to add a timer** to determine the end of the game.

The following note:

- You have to add the timer (you can find it in the *Sensing* tab).
- After a while (eg 30 s) the mouse and the cat hide (help yourself with the block `hide`).
- You have to reset the timer at the end of the game (help yourself with the block `reset timer`).

8. Next you have to **add a new sprite** – a girl. She tells the score of the player how many times she or he has caught the mouse.

TIP: Help yourself with the block `join hello world ◂▸`.

[ADDITIONAL TASKS]
You can add any elements to the game.

9. For example, you can upgrade the game by adding the girl who is afraid of mice and she jumps every time when she touches a mouse.

   TIP: Help yourself with the blocks `touching ▾ ?` *and* `switch to costume ▾`.

10. For example, you can upgrade the game by adding the sound of the cat which it plays when the cat caught the mouse.

TIP: Help yourself with the blocks `touching ▾ ?` *and* `play sound ▾`.

WHEN YOU FINISH, **SAVE** THE PROGRAM!

Catch the mouse:
https://snap.berkeley.edu/project?user=tadeja&project=Catch%20the%20mouse

Scenario 16 – Buying food for a picnic

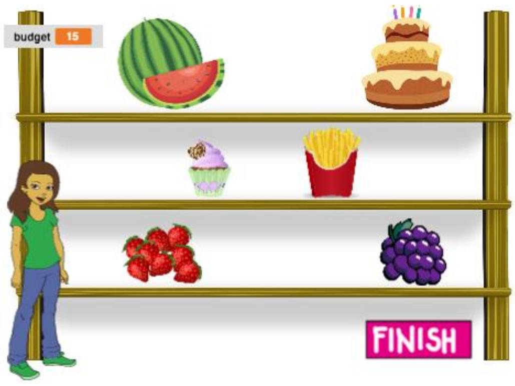
Instructions: Create a game where a player buys healthy and unhealthy products (food). The game should include:

- Initial instructions, given by a sprite (girl).
- The amount of money a player has at the beginning.
- Healthy and unhealthy products and a price for each product.
- When mouse pointer hovers a food product, the product's price shows.
- When mouse pointer hovers a girl, she says how much money is still available.

1) Open a new project, select a background and a sprite (e.g. a girl). The girl gives initial instructions to the player.

2) You will need more variables. Think about why do you need next variables:
   a. *budget*,
   b. *healthy_food*,
   c. *unhealthy_food*,
   d. »*price_of_particular_food*« - you can add this later, when you will know which food products will you have.

   Set a starting value for each variable.



3) Add a food sprite, e.g. a watermelon.
   a. The watermelon shows at the beginning. Set a price for this product.
   b. When mouse pointer hovers the watermelon, the watermelon says its price. E.g. Watermelon costs 4 EUR.
   c. What happens when you click on the watermelon (and you want to buy it)? Think:
      i) In which case can the player buy the watermelon and in which case he can not buy it?
      ii) What happens with the variable *budget*, if the player buys the watermelon?
      iii) How can we count the bought products?
      iv) What happens with watermelon?

4) You can duplicate the watermelon and change its costume, so you will have more products on the shelf.
   How will the watermelon and e.g. the cake code differ?

5) Create a sprite to end shopping (e.g. FINISH).
   Clicking on this sprite, the sprite broadcasts a message for finishing with shopping.

6) The girl says e.g. *You chose 2 healthy and 3 unhealthy products!*

7) Add a code which will, when mouse pointer hovers a girl, say how much money is still available.

[Additional tasks]

Add additional tasks according to your wishes or follow the tasks below:

- Change the game so you can buy each food 3 times.
- Give more money to the player at the beginning.
- At the end the girl tells also how many products you bought. E.g. “You bought 2x watermelon, 1x grapes, 2x fries”.

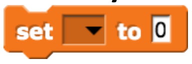
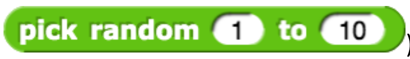
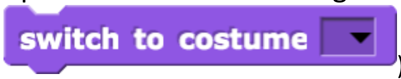
Scenario 17 – Operations

You have to program the game with which the player solves simple mathematical operations.

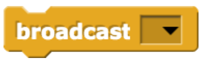
1. First you have **to open the** *Operations* **program**. In it you find a program template with background and sprites – the numbers.

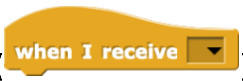
2. Next you have to add the code that **the backgrounds (mathematical operations) will be randomly changed 10 times**.

The following note:

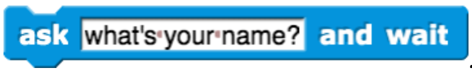
- You have **to add a new variable named *Operation*** in which you will store the mathematical operation - the background (you go to Variables tab and click on the New Variable button).
- You have **to randomly set the value of the Operation variable** (help yourself with the blocks 
set to 0
and 
pick random 1 to 10
) .
- You have **to switch background** to the randomly selected mathematical operation – another background (help yourself with the block 
switch to costume
).

3. You have to add the code that it **randomly change the number** (the sprite). First you have to create a new variable named Number. Then you have to add the code that the number is changed, when the mathematical operation is changed.

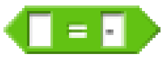
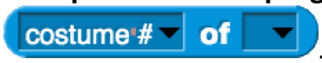
You do this by sending a message using the 
broadcast
block.

When the number receives a message (
when I receive
), the number is changed randomly (similar to task 2).

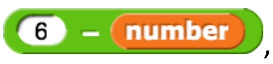
4. You have to add the code that **the player can enter the result**.

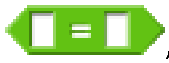
TIP: Help yourself with the block 
ask what's your name? and wait
.

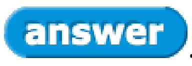
5. Now you have to add code that you can check if the player has entered the correct result.
   The following note:
   - You have to create a new variables named *Correct* and *Incorrect* to count how many correct and incorrect answers the player has given.
   - **You have to check which mathematical operations is in progress.**

   TIP: Help yourself with the blocks 
=
and 
costume # of
.

- **You have to verify that the player's answer is correct.**

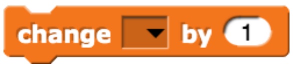
TIP: Help yourself with the blocks 

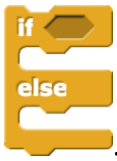
, 

, 

.

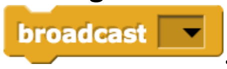
- **If the player's answer is correct, you have to increase the Correct variable by 1, otherwise you have to increase Incorrect variable by 1.**

TIP: Help yourself with the blocks 

and 

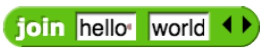
.

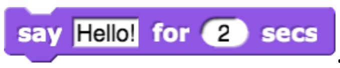
6. Finally you have to add the code that let the player know how many points he has scored. The following note:
   - **You have to send a message:** *sum the points*.

TIP: Help yourself with the block 

.

- When the number receives the message, it calculates **how many points the player has scored**. The points are calculated by subtracting the number of incorrect answer from the number of correct answer.

TIP: Help yourself with the blocks 

and 

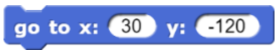
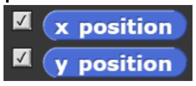
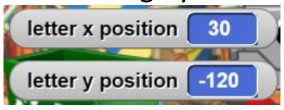
.

WHEN YOU FINISH, **SAVE** THE PROGRAM!

Operations: https://snap.berkeley.edu/project?user=ddureva&project=operations_half

Scenario 18 – Recycling

Robot Binny has noticed that there are pieces of paper and glass object on a playground and that kids cannot play there. Because it wants to help kids and their parents to clean up the playground, it brought two bins to sort the waste into: a green one for glass, and blue one for paper waste. Program a game to teach kids to sort the waste into appropriate bin to sort and collect the trash from the playground.

1. Open a file C4G18_Recycling_Part. In this file you will find robot Binny, two bins and waste, that needs to be sorted. Make sure that the waste will be scattered on the playground at the beginning of the game. You can use 
go to x: 30 y: -120
. To find out where on the stage are the objects, you can check the checkboxes next to x and y position for each object separately

☑ x position
☑ y position
and on stage you will see the value of x and y position of the object:

letter x position 30
letter y position -120
When you set the starting position, uncheck the boxes so that they will not interfere with your work and play.

2. When green flag is clicked robot Binny, both bins and all the trash have to be shown.

3. Binny shall ask the player, what is her name, and give her instructions: the player has to put the waste in appropriate bins, green for glass and blue for paper. When Binny ends his instructions, it shall broadcast a message for the beginning of the game and hide itself.

4. Determine for each piece of waste to which bin does it belong: if player drags it to appropriate bin the piece is hidden, otherwise is says that it does not belong to that bin. Checking shall start when the message for the beginning of the game is received.

5. Make a variable, that will count, how many pieces of waste have been successfully disposed or how many still need to be collected – it is up to you to decide. Think when the value of the variable should be displayed on the stage and when it should be hidden. You show or hide it also during the game:

6. The game ends, when the player has correctly sorted all the waste. Think how to check if all the waste was collected.

7. When all the pieces of waste are correctly disposed, Binny will show up, and congratulate you for the finished task. Binny will address you by your name, that you have written in the beginning, e.g. Anna, congratulations on cleaning the playground. Now you can go and play.

Additional task: add another bin for plastics and plastic waste to the game

C4G18_Recycling_Part: https://snap.berkeley.edu/project?user=mateja&project=C4G18_Recycling_Part

Scenario 19.1 – Play a Piano

1) Teacher showed you the program *Play a Piano 1* and explained how the Sound group commands can be used.

2) Divide into groups. Each group will create a game like *Play a Piano 1*.

3) Open the pogram *Play a Piano Half baked*:
https://snap.berkeley.edu/project?user=ddureva&project=Play_a_Piano_Half_backed

4) Discuss about the game scenario and describe the game plan in the description sheet (page 2).

Additional: you can add a condition for the dinosaur to dance while playing (the dinosaur has several costumes in the pre-prepared file).

Describe your project for computer game
Title ....
Team members: ....

1. Describe game plot.

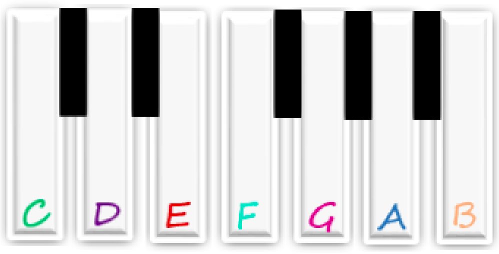
2. Sprites, Backgrounds, Variables (Describe: sprites – name, costumes, etc.; backgrounds; Variables – name, purpose, area of activity – for all sprites, for current sprite (please write name of the sprite if Variable is related only to one sprite)

3. Detailed plot

Sprite (name or picture)	Events	Duration

4. Used commands (blocks):

Scenario 19.2 – Play a Piano

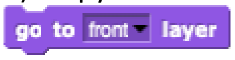


60 62 64 65 67 69 71

1) Open the program *Play a piano*:
https://snap.berkeley.edu/project?user=ifrankovic&project=Play%20Piano

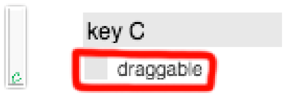
You will assemble the keys as you see in the picture above.

2) The key C is already in the right place.
   a) Duplicate the key C.
   b) Move it to the correct position.
   c) Rename it.

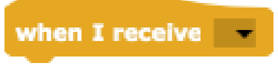
3) Copy the black key. It the black key is hidden behind the white keys, use the code 

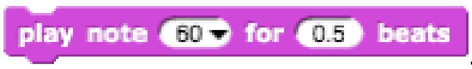
.

4) Uncheck the »draggable« button, so the key sprites can not be moved while playing. Do this for all sprites.



5) To play a note we will use those commands:



and 

.

6) We will write a code for clicking on a sprite/key (*when I am clicked*) and for pressing a key (*when __ key pressed*). Will the codes be the same? What will happen when you click on the sprite/key or when you press a key on the keyboard?

7) By clicking the sprite *violin key* the background will change to background *chords*.

8) By clicking the sprite *reset* the background will change to background *blank*.

9) By clicking the sprite *note* the whole song will be played. Write a code that plays the whole song. Use a loop where possible.

[Additional tasks]
Add additional tasks according to your wishes or follow the tasks below:

- Duplicate the sprite Note (and change its position on the background) and write a program for another song.
- Add a background with chords for the new song.

Scenario 20 – Test

Abby would like to test classmate's knowledge about Snap! She created a test with questions, but she has lost the code for checking the correctness of the answers. Help her create a program, which will change background before each question, and let Abby tell the possible answers to her classmate at the same time. Abby will also count correct and wrong answers.

8. First you have to open C4G_20_test_en_tmp program, where you will find Abby, start button, and the backgrounds you will need for the test. Abby first gives instructions to her classmate, which are already written for you. Each correct answer brings 1 point, while each wrong answer deducts a point from a total score.
9. Start button: First make sure, that the game will start, when the start button in pressed. On click the button has to be hidden. How will you inform other characters that the game has begun? What do you have to do with the start button on the beginning of the program?
10. Changing stage background: when the game starts when background has to be changed with the next one. The easiest way to do this is that Abby broadcasts a message to change, and the stage changes costume to the next one when it receives this message. Try it out!
11. Create 2 variables: one for correct and one for wrong answers
12. Seting questions: As the questions are written on the stage background, Abby has to ask a question so that she tells the classmate what are the possible answers, e.g. You can answer with yes or no
13. When classmate answers, Abby must tell him if the answer was correct or not. If it was she also adds a point for correct answer, otherwise she adds a point to wrong answers
14. After each question the background is changed to the next one. Do you still know how to do it?
15. At the end of the test, Abby has to tell her classmate how many answers were correct, how many were wrong and what is his score (= number of correct – number of wrong answers)
16. Based on the total score Abby tells her classmate if he successfully passed the test or if he has to retake it

Additional task 1: Abby can change appearance during the game

Additional task 2: add your own questions. The do not need to be written on the background - Abby can tell them to the classmate

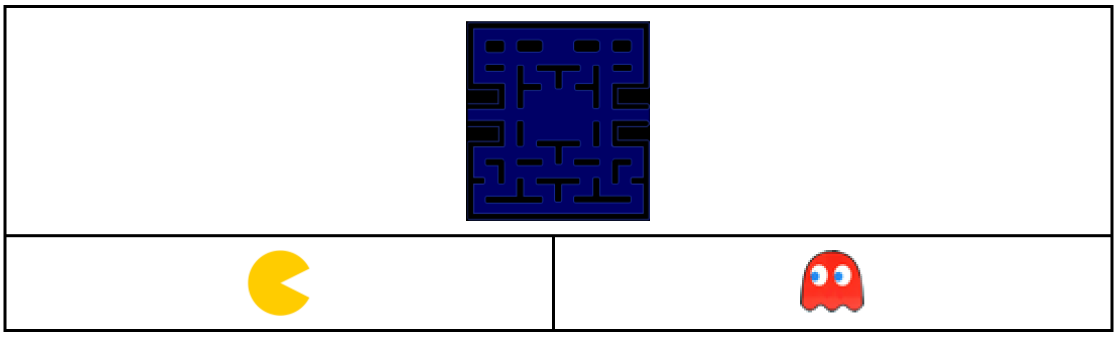
Template: https://snap.berkeley.edu/project?user=spelac&project=C4G_20_test_en_tmp

Scenario 21 – Simplified PacMan

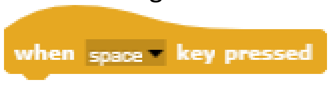
Task: Program a game of simplified Pacman. The main character (pacman) moves around the labyrinth and collects stars. In the beginning there is one star on the random position, when pacman collects it, new star appears in a new random location inside the labyrinth. Pacman is chased by a ghost who is randomly moving. The game is over when the player collects 20 stars or if pacman touches the ghost.

1. Open the template file: https://snap.berkeley.edu/project?user=zapusek&project=pacman_template

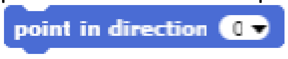
   Inside you can find an image of the pacman, red ghost, star and background that represents a labyrinth:



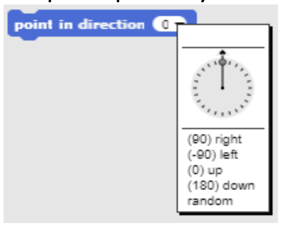
2. First we have to program the movement of the pacman. Player can control its movement with arrow keys. We have to consider that pacman can not move through the walls.

   We are going to code the movement using multiple 

blocks. We will define the appropriate keys.

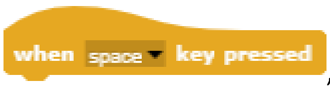
   At each step we will have to turn the pacman to the corresponding direction and then make a move. We can turn an object using 

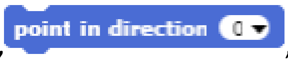
block, and select the direction using the drop down menu. We don't have to type in the exact angles of directions because Snap! helps us by naming them.

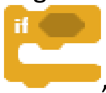


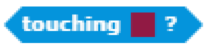
   Pacman cannot move arbitrarily, because he is limited by walls. Walls are painted black, and the legal paths are painted in blue. This can be used in order to implement such movement. If pacman is on blue he can move, but if he touches the black color, he has to stop. Each step must be combined with this condition.

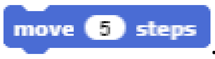
   Code the movement on the blue area of the labyrinth using these blocks:



, 

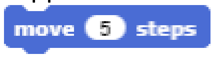
, 

, 

and 

.

If we find out that he touches the black color, we know he touched the wall. In this case we have to move him back in an opposite direction, so he doesn't get stuck on the wall.

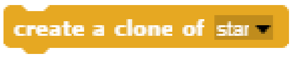
Positive values inside block  `move (5) steps` will move the object forward, negative values will result in moving the object backwards.

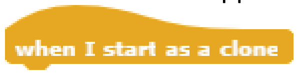
Add another if block to check if pacman is touching the black color. In that case we move him 5 steps in an opposite direction.

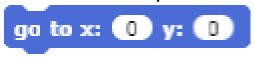
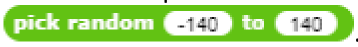
3. Next we want to code the behavior of the star. In a simplified pacman game there is only one star at the time. When the player collects it, a new star appears in a random location. For each star player gets one point, when she has a total of 20 points the game ends.

   In the game there will be a total of 20 stars. We are going to implement stars as clones of the original star object.

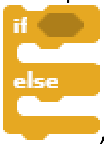
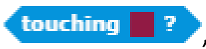
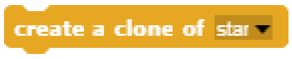
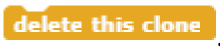
   We use clones in situations where we want to use several instances of the original object.

   New clone is created with the  `create a clone of [star]` block. In a drop down menu we select the object we want to clone.

   Now we have to click on a star and code what happens when the clone from the original star is created. This is done using a  `when I start as a clone` block.

   When a new instance of the star starts as a clone it has to move on a random position on a screen. The default background height and width in Snap! is from -140 to 140 pixels. If we assign a random value between those two numbers for x an y position of clone, we achieve the random placement of the star. Use combination of blocks:  `go to x: (0) y: (0)` and  `pick random (-140) to (140)`.

   The next problem that arises is when the computer selects such numbers from the interval that the star is placed on a wall and pacman can't get to it.

   We know that the wall is represented by a black color, we can use that to check if a star is placed on a wall or not. If this is true we can make another clone and delete this one. This will be repeated until the clone will be placed in a legal location. Code this using these blocks:
    `if ⟨⟩ else`,  `touching [■] ?`,  `create a clone of [star]` and  `delete this clone`.

   Checking if the pacman touched the star could be programmed within the pacman, or in the clone code. Because we want to introduce the idea of sending the messages, we will make it inside a clone.

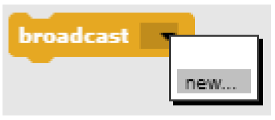
Player can move the pacman freely around the labyrinth, so we have to check if he touched the clone in a forever loop. To detect if the object touched another object we can use the block: `touching ▾ ?` and from the drop down menu select the other object.

When the pacman collects the clone the player will be awarded one point. Let's implement this by setting a point counter variable inside a pacman. When the clone and pacman collide, the clone will send a message to the pacman, so he will know that he has to increment the point counter.

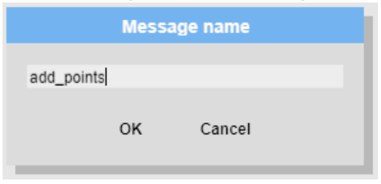
Click on a pacman and create a new variable *points* for counting points. Set it to zero, because the player doesn't have any points in the beginning of the game.

In Snap! objects can send each other messages in order to notify them that something has happened. This is done with the block: `broadcast ▾`. When the message is broadcasted, all the objects on a scene receive it. In each object we can decide what will be its response. If we don't program anything, nothing will happen.

We create a new message by clicking on a small arrow inside the block's input field and select "new".



Then we have to name a message. We want to choose a mnemonic name, so the meaning of the message will be recognized from its name.



Complete the code of the star clone in a way that it will constantly check if the clone has collided with the pacman. When this happens, it should send a message *add_points*. After sending a message, the clone must send a command to create a new clone and delete itself. To program this functionality use those blocks:
`touching ▾ ?`, `broadcast ▾`, `create a clone of star ▾` and `delete this clone`.

Let's see how we can react to received message in the other object. In the "Control" group we have an event block `when I receive ▾` that starts its execution when a specified message is received. We can select a specific message using a dropdown menu.

In a pacman code, program the incrementation of the point counter, when the message is received by using these blocks: `when I receive ▾` and `change ▾ by 1`.

Here we can also code the end game condition. If we increment points by 1 and the total score is equal to 20, the game is over. Code this functionality using these blocks:

if, = , points and stop all.

4. Now we have to program a ghost. The ghost must move randomly throughout the labyrinth, it must not go beyond the wall, but must bounce off and continue in a random direction. If he touches the pacman, the game is over.

   Let us consider how can we make a ghost move in a random direction after touching the wall. He can move in the following directions: left, right, up and down.

   Using a point in direction 90 we can turn the object. We can choose the desired direction by entering the angle of the turn:

   - 0 degrees = UP,
   - 180 degrees = DOWN,
   - 90 degrees = RIGHT,
   - 270 degrees = LEFT.

   We want to find a way in which a point in direction block would be randomly set to one of those values: 0, 90, 180 or 270. Problem is that in Snap! we can get a random value from an interval not from a list of values. If we study these numbers, we can notice that all are multiples of the number 90, because: 0*90 = 0, 1*90 = 90, 2*90 = 180 and 3*90 = 270. We can get a random number from interval 0 to 3 with the use of pick random 1 to 10 block, if we multiply this randomly selected number with 90, we get one of those numbers: 0, 90, 180, 270. This is exactly what we want!

   The movement of the ghost and testing if it has touched the wall must take place throughout the game, until the player reaches 20 points or the ghost touches the pacman. We already programmed the first option that ends the game now let's take care of the second.

   The exit condition for moving the ghost is when the ghost collides with the pacman. If we use the repeat until loop we can set the condition to when it touches the pacman. Until this is not true, the ghost will move around, when he will hit the pacman, the loop will stop and blocks placed after the loop will be executed.

   The ghost movement is quite simple. In every iteration of the loop he must move by 1 step in a given direction.

   We can test if the ghost touched the wall with the sensing color block. Walls are black and similar as we did with pacman, we can repeatedly test if the ghost is maybe touching the black color. If this occurs, we have to move it 1 step backwards (so it doesn't get stuck on the wall) and change direction randomly as described above.

   The loop will stop when the condition in its head will be true. This will happen when a ghost collides with pacman. We have to provide feedback and end the game.
   Code this functionality using these blocks:
   repeat until, if, stop all, point in direction 90, move 1 steps, touching ?, × and pick random 0 to 3.