**Learning 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>

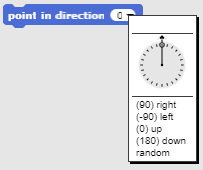
Inside you can find an image of the pacman, red ghost, star and background that represents a labyrinth:

|  |  |
| --- | --- |
|  | |
|  |  |

1. 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  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.

1. 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  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  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:  and .

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:

, ,  and .

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:  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.

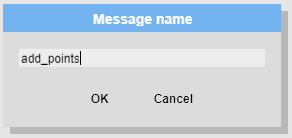
Click on a pacman and create a new variable *points* for counting points. Set it to zero, because the player doesn't have any points in the beginning of the game.

In Snap! objects can send each other messages in order to notify them that something has happened. This is done with the block: . 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:

, ,  and .

Let’s see how we can react to received message in the other object. In the “Control” group we have an event block  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:  and .

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:

, ,  and .

1. 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 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  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:

,,,,,,  and .