Methodological guidelines

Maze Game. Part 1

🚀STORYLINE:

The ProTeam game development specialists are taking on their first large order for the creation of the Maze game. The game designers have already thought out the logic of the game, and the artists have drawn images for the background and sprites. Now, the developers have to program the functionality of the game.

While discussing the project, the developers come to the idea that they need to create a GameSprite class to display the protagonist sprite and antagonist sprite.

⚠ SUMMARY:

The **lesson goal** is to program the GameSprite class and position its instances (sprites) on the scene using a game loop.

In the first half of the lesson, the students plan the project and create a template for the game with background and music. In the second half, they program the GameSprite class, create instances of the class for the sprites, and position them on the scene.

Technical note. All work on the Maze is organized in one task in the VS Code.

💾 LINKS AND ACCESSORIES:

❏ Presentation,

❏ Exercises for the lesson: Maze Project (Visual Studio Code).

🎯 LEARNING RESULTS

After the lesson, the students will:	*The result is achieved when the students:*
● explain the purpose of the PyGame Sprite class in their own words; ● create the child class from the existing class; ● use the super().__init__() method to call a superclass constructor; ● set fields for their own class; ● create a method of their own class; ● create instances of their own class; ● apply knowledge of object-oriented programming to game creation.	● have participated in the discussions and asked clarifying questions; ● confidently listed the basic game components in PyGame; ● planned working on the project with the help of a mind map and a checklist; ● have created a child class from the PyGame Sprite class; ● have created a game template and displayed three key sprites in it; ● have answered the teacher's questions during the review stage.

RECOMMENDED LESSON STRUCTURE

Time	Stage	Stage aims
10 min	**Storyline. Discussion: Working on the project**	❏ Set the storyline-based task: to program the Maze game in a few workdays. ❏ Plan the work: draw up a mind map and a checklist with tasks for the current workday.
10 min	**Qualifications**	❏ Get the developers to confirm their qualifications by the topics: ❏ Basics of game creation in PyGame; ❏ Creating classes and inheritance.
10 min	**Brainstorming: Maze Game template**	❏ List components for the game template: game window, background, game loop. ❏ Demonstrate the mixer module commands to set the background music for the game.
15 min	**Platform: “VSC: Pygame: Maze”**	❏ Have the students complete the exercise “VSC. PyGame: Maze”.
5 min	**Break**	❏ Do a warm-up or change students’ activity.
15 min	**Brainstorming: Classes for sprites**	❏ Talk about the Sprite class implemented in PyGame and its capabilities. ❏ Explain that some of the functionality important for the game has already been implemented in it. ❏ Arrive at the idea that it is required to create the GameSprite, the child class of the Sprite class. ❏ Analyze how the GameSprite class is arranged and how to introduce it into the program.
20 min	**Platform: “VSC: Pygame: Maze”**	❏ Organize the performance of exercise “VSC. PyGame: Maze”.
5 min	**Wrapping up the lesson. Reflection**	❏ Conduct a technical interview based on the brainstorming material. ❏ Suggest that the students complete the extra exercise on the VS Code for additional practice.

Storyline. Discussion: Working on the project

(10 min)

Methodological guidelines. Save the reference solution of the project to your computer beforehand. Show it when discussing the expected appearance of the game, so that the developers can see not only the graphics but also the soundtrack.

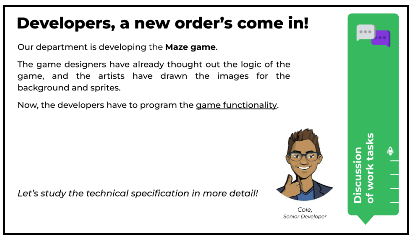
Open the presentation. The developers do not need computers yet.

> *“Hello, colleagues! Our department has received a new order – to create a maze game. The work on this project will take three workdays. Colleagues from other departments have already thought out the logic of the game and drew pictures for the background and sprites. And you need to program the functionality of the game.”*

Show the developers the slide with the technical specification and expected appearance of the game. Mention that there are new mechanics in the game, such as different controls for the sprites (one character is controlled by the keyboard, the other one is controlled automatically) and soundtrack.

Work with the developers to create a mind map of the game. Discuss what tasks should be completed on the first day and make a checklist for them. We recommend doing this stage on computers in the corresponding tasks on the platform.

As you complete the tasks, show the mind map and the task checklist proposed by developer Cole. When planning the project, point out the unknown features of the game, such as the need to handle sprite collisions. Is there a ready-made tool in PyGame for that?



Developers, a new order’s come in!

Our department is developing the **Maze game**.

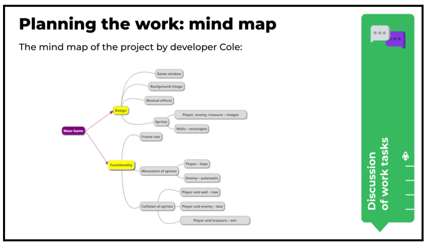
The game designers have already thought out the logic of the game, and the artists have drawn the images for the background and sprites.

Now, the developers have to program the game functionality.

Let’s study the technical specification in more detail!

Cole,
Senior Developer

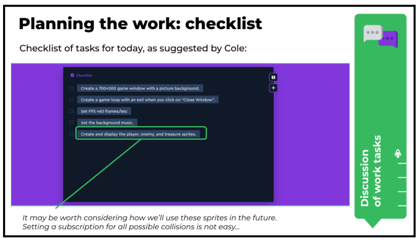
Discussion
of work tasks



Planning the work: mind map

The mind map of the project by developer Cole:

Maze Game
Design
Game window
Background image
Musical effects
Sprites
Player, enemy, treasure – images
Walls – rectangles
Functionality
Frame rate
Movement of sprites
Player – keys
Enemy – automatic
Collision of sprites
Player and wall – lose
Player and enemy – lose
Player and treasure – win



Planning the work: checklist

Checklist of tasks for today, as suggested by Cole:

Checklist
- □ Create a 700×500 game window with a picture background.
- □ Create a game loop with an exit when you click on “Close Window”.
- □ Set FPS =60 frames/sec
- □ Set the background music.
- □ Create and display the player, enemy, and treasure sprites.

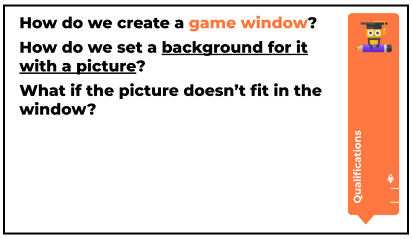
It may be worth considering how we’ll use these sprites in the future.
Setting a subscription for all possible collisions is not easy...

Set the goal for the day and announce what will need to be done.

Qualifications

(10 min)

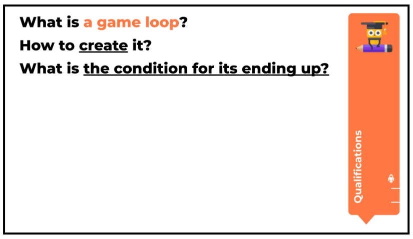
Use the presentation to get the developers to confirm their qualifications before they start working. This time it will be organized by topics: basics of creating games and object-oriented programming.



How do we create a game window?

How do we set a background for it with a picture?

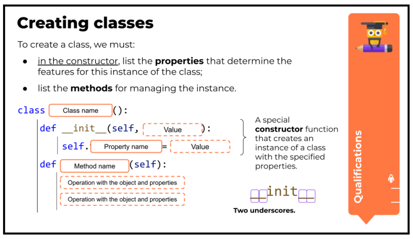
What if the picture doesn't fit in the window?



What is a game loop?

How to create it?

What is the condition for its ending up?



Creating classes

To create a class, we must:

- in the constructor, list the **properties** that determine the features for this instance of the class;
- list the **methods** for managing the instance.

```
class  Class name  ():
    def __init__(self,  Value  ):
        self. Property name = Value
    def  Method name  (self):
        Operation with the object and properties
        Operation with the object and properties
```

A special **constructor** function that creates an instance of a class with the specified properties.

__init__

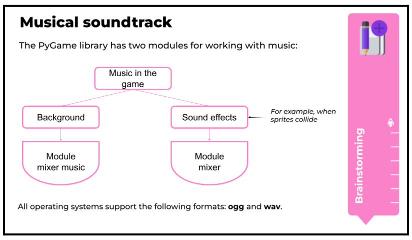
Two underscores.

Brainstorming: Maze Game template

(10 min)

Use the presentation to tell the students about modules for working with music. Mention that different PyGame library modules and different methods are used to add background music and sound effects.

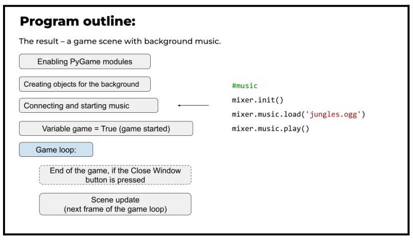
Discuss how to incorporate background music into the game loop.



Musical soundtrack

The PyGame library has two modules for working with music:

All operating systems support the following formats: **ogg** and **wav**.



Program outline:

The result – a game scene with background music.

Enabling PyGame modules
Creating objects for the background
Connecting and starting music
Variable game = True (game started)
Game loop:
End of the game, if the Close Window button is pressed
Scene update (next frame of the game loop)

```
#music
mixer.init()
mixer.music.load('jungles.ogg')
mixer.music.play()
```

Specify that sound files must be in ogg or wav format, as they are supported by all operating systems, and stored in the project folder.

Technical comment. The background music may also be set in .mp3 format, but when running such a program on the Linux operating system, problems are possible.

Wrap up the discussion and begin working on the VS Code.

Platform: "VSC. Pygame: Maze"

(15 min)

Arrange the work on the VS Code. The task requires the developers to program a game template with a picture background and set background music.

Link to the project archive is available at the end of the methodological guidelines.

Brainstorm: Classes for sprites

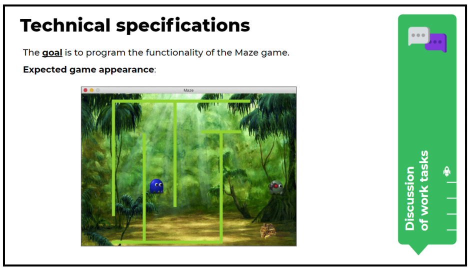
(15 min)

Procedural comment. In this lesson, it is crucial to achieve a conscious use of inheritance to create a game class. In the next two lessons, the children will continue working with this topic and will create new classes, for example, Enemy as a child of the GameSprite class. The slow pace of this lesson is chosen due to this specific feature.

Remind the developers that according to the technical specification, the player loses when colliding with the enemy or walls of the maze. Demonstrate the game again:

> *“How to program the collision of the player and enemy sprite? (Students’ answers). Possible solution: compare coordinates of the current locations of the player and enemy sprites. Guess what is the disadvantage of this approach? (Students’ answers). Right, handling a collision with walls would be very cumbersome. We need a different solution, and PyGame has it – these are tools of the ready-made Sprite class!”*

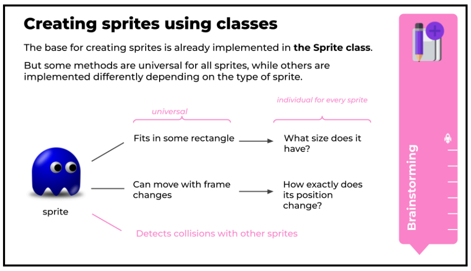
Explain to the developers that the foundation for creating sprites is already implemented in the Sprite class, but they will not be able to create characters as the Sprite instances: some methods are universal for all sprites, while others are implemented differently depending on the type of sprite.



Technical specifications

The **goal** is to program the functionality of the Maze game.

Expected game appearance:



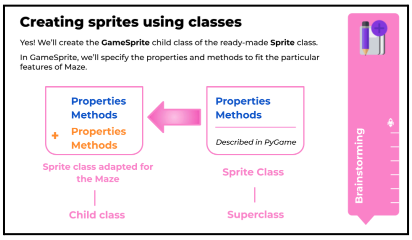
Creating sprites using classes

The base for creating sprites is already implemented in **the Sprite class**.

But some methods are universal for all sprites, while others are implemented differently depending on the type of sprite.

universal
individual for every sprite
Fits in some rectangle
What size does it have?
Can move with frame changes
How exactly does its position change?
sprite
Detects collisions with other sprites
Brainstorming

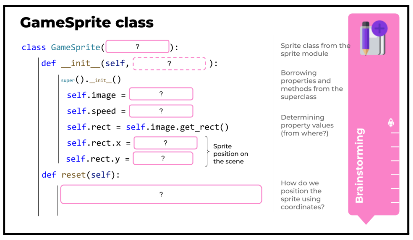
While discussing, suggest a way to solve this difficulty: to take useful properties and methods from the ready-made Sprite class and add new details to them using inheritance. To do this, they can create a child class GameSprite of the ready-made Sprite class and specify the properties and methods to fit the particular features of Maze.



Creating sprites using classes

Yes! We’ll create the **GameSprite** child class of the ready-made **Sprite** class.

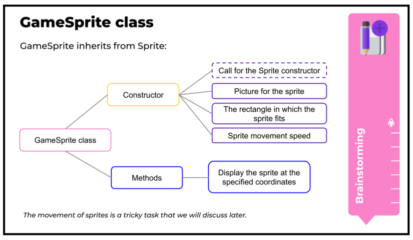
In GameSprite, we’ll specify the properties and methods to fit the particular features of Maze.



GameSprite class

```
class GameSprite( ? ):
    def __init__(self, ? ):
        super().__init__()
        self.image = ?
        self.speed = ?
        self.rect = self.image.get_rect()
        self.rect.x = ?
        self.rect.y = ?
    def reset(self):
        ?
```

Sprite position on the scene
Sprite class from the sprite module
Borrowing properties and methods from the superclass
Determining property values (from where?)
How do we position the sprite using coordinates?
Brainstorming



GameSprite class

GameSprite inherits from Sprite:

GameSprite class
Constructor
Call for the Sprite constructor
Picture for the sprite
The rectangle in which the sprite fits
Sprite movement speed
Methods
Display the sprite at the specified coordinates
Brainstorming

The movement of sprites is a tricky task that we will discuss later.

Discuss the scheme of the GameSprite class. Suggest that the developers fill in the blanks using information from the previous slides. Discuss how to introduce the class into the finished program code.

Wrap up the discussion and begin working on the VS Code.

Platform: “VSC. PyGame: Maze”

(20 min)

Arrange the work on the VS Code. The task requires the developers to implement the GameSprite class and create instances of this class for the sprites: player, enemy, and treasure. Position the sprites on the scene.

Link to the project archive is available at the end of the methodological guidelines.

Wrapping up the lesson

(5 min.)

Have the developers turn away from their computers, then organize a technical interview about the brainstorming material. Announce that on the next workday they will continue working on the Maze game and will set the movement of sprites.

Suggest that the developers complete the additional exercises to improve their skills and provide learning materials.

Exercises answers:

Exercise "VSC. Pygame: Maze", part 1.

```
from pygame import *
"Required classes''

#parent class for sprites
class GameSprite(sprite.Sprite):
   #class constructor
   def __init__(self, player_image, player_x, player_y, player_speed):
       super().__init__()
       #every sprite must store the image property
       self.image = transform.scale(image.load(player_image), (65, 65))
       self.speed = player_speed
       #every sprite must have the rect property - the rectangle it is fitted in
       self.rect = self.image.get_rect()
       self.rect.x = player_x
       self.rect.y = player_y

   def reset(self):
       window.blit(self.image, (self.rect.x, self.rect.y))

#Game scene:
win_width = 700
win_height = 500
window = display.set_mode((win_width, win_height))
display.set_caption("Maze")
background = transform.scale(image.load("background.jpg"), (win_width, win_height))

#Game characters:
packman = GameSprite('hero.png', 5, win_height - 80, 4)
monster = GameSprite('cyborg.png', win_width - 80, 280, 2)
final = GameSprite('treasure.png', win_width - 120, win_height - 80, 0)

game = True
clock = time.Clock()
FPS = 60

#music
mixer.init()
mixer.music.load('jungles.ogg')
mixer.music.play()

while game:
   for e in event.get():
       if e.type == QUIT:
           game = False

   window.blit(background,(0, 0))
   packman.reset()
   monster.reset()

   display.update()
   clock.tick(FPS)
```

Link to the archive with a complete solution: archive.