



# Python Programming

## *Project 3*

# Maze Game.

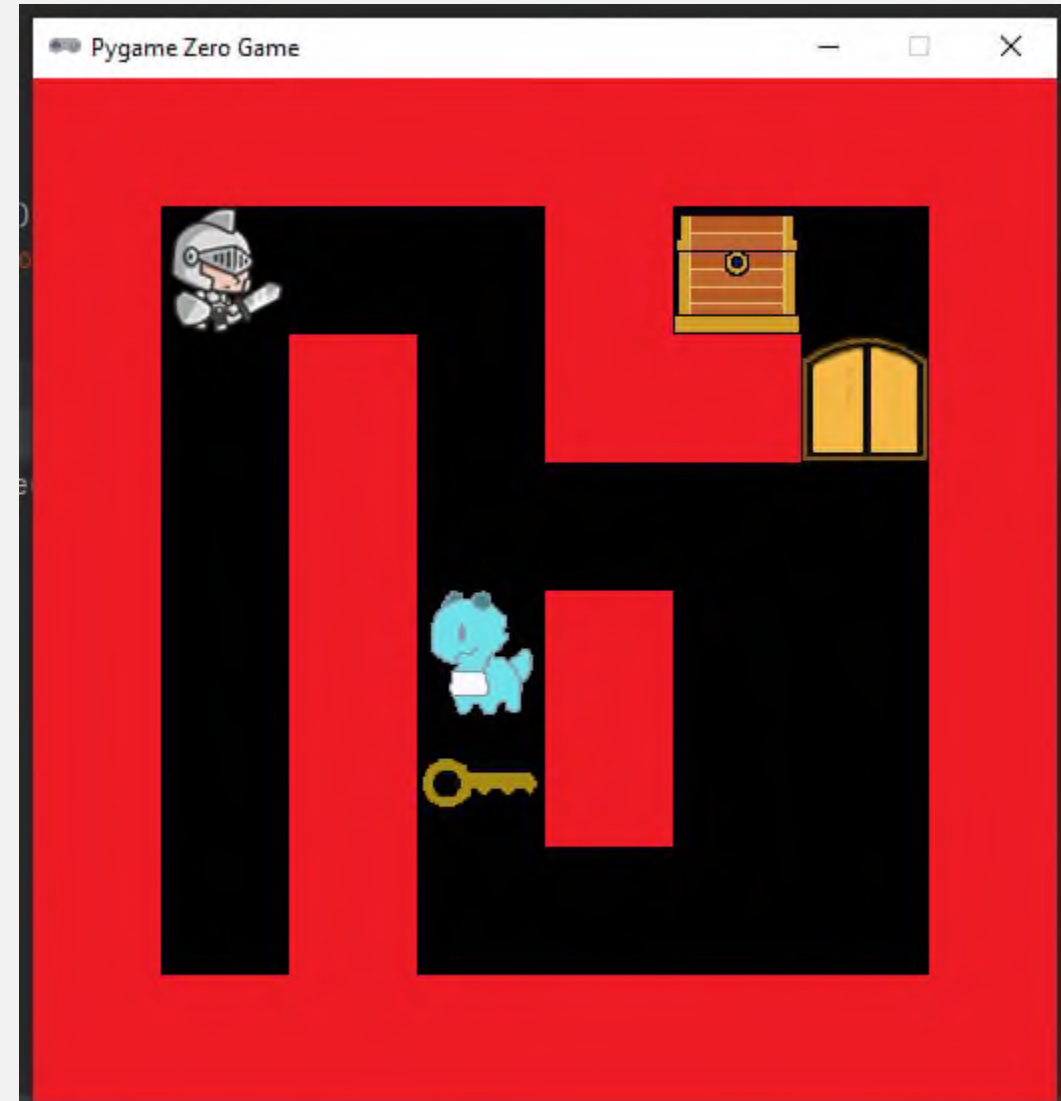
Presented by Advaspire Team



# What we are making

In this lesson, we are going to make a maze game, step by step. The technique of creating a tilemap is common in games and after seeing it here you should be able to incorporate it into your own projects.

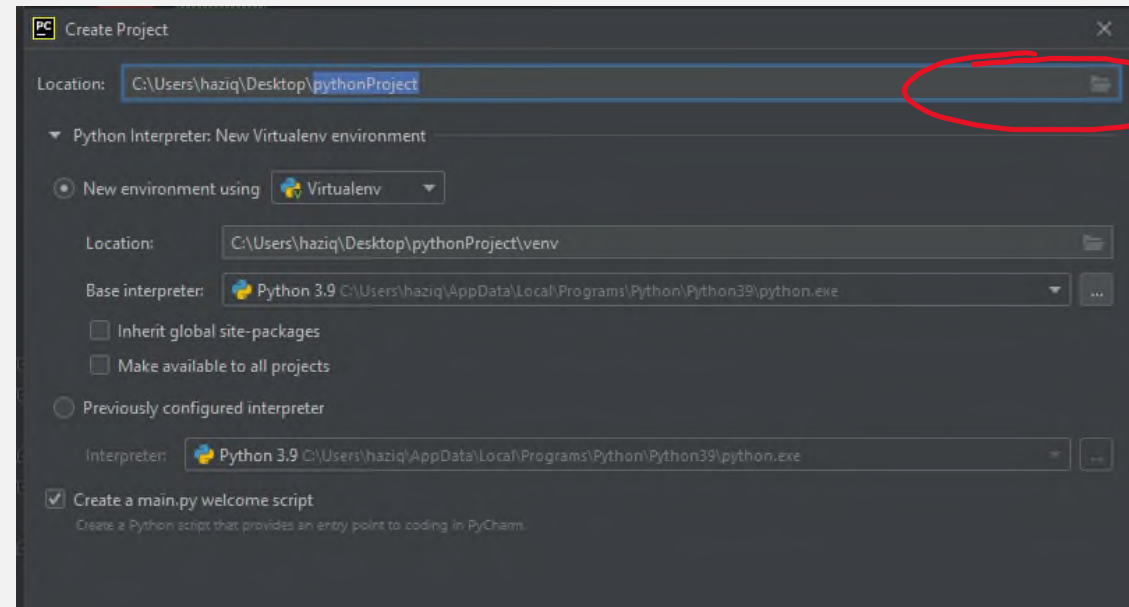
The Python we will use in this project is quite simple: mostly just conditionals and loops.





# Setting Up

Create a new project. This time, I recommend to create it at your desktop for easy access. Do the same thing we did last time.





# Setting Up Tilemap

A tilemap uses a small number of images (the tiles) and draws them many times to build a much larger game level (the map). This saves you from creating a lot of artwork and makes it very easy to change the design of the level on a whim.

The way we setup the display this time is a bit different as we are using a tilemap.

```
import pygame

TILE_SIZE = 64
WIDTH = TILE_SIZE * 8
HEIGHT = TILE_SIZE * 8
```



# Tilemap Cont

We need 3 image files for the tiles : empty, wall and goal. If you are using your own sprites, make sure it is 64x64 pixels, just like our tilesize.

```
tiles = ['empty', 'wall', 'trechest']
```

Once you have prepared the sprites, create a list called tiles. Inside, write as strings the sprites in this order : empty, wall and goal.

Note: I will use the treasure chest as the goal.



## Level Design

The level design is stored in a list of lists, more commonly called a *two dimensional array*. There are 8 rows and 8 columns in the array.

If you change the size of the array you will need to change the Width and Height values.

1	1	1	1	1	1	1	1
1	0	0	0	1	2	0	1
1	0	1	0	1	1	0	1
1	0	1	0	0	0	0	1
1	0	1	0	1	0	0	1
1	0	1	0	1	0	0	1
1	0	1	0	0	0	0	1
1	1	1	1	1	1	1	1

The numbers in the maze array refers to elements in the tiles array. It kind of use the index position to assign. That is why I lined it that way in the list previously.



# Tilemap code

```
maze = [  
    [1, 1, 1, 1, 1, 1, 1, 1],  
    [1, 0, 0, 0, 1, 2, 0, 1],  
    [1, 0, 1, 0, 1, 1, 0, 1],  
    [1, 0, 1, 0, 0, 0, 0, 1],  
    [1, 0, 1, 0, 1, 0, 0, 1],  
    [1, 0, 1, 0, 1, 0, 0, 1],  
    [1, 0, 1, 0, 0, 0, 0, 1],  
    [1, 1, 1, 1, 1, 1, 1, 1]  
]  
  
player = Actor("alien", anchor=(0, 0), pos=(1 * TILE_SIZE, 1 * TILE_SIZE))
```

Make sure to use the right player sprite. The rest of the code will be the same.

The number here represent the index position of the items inside the tile list before. So based on it, 0 = empty, 1= wall, 2 = chest.



# Drawing Maze

To draw the maze we use a for loop within another for loop. The outer loop iterates over the rows and the inner loop iterates over the columns, i.e. the elements of the row.

```
def draw():  
    screen.clear()  
    for row in range(len(maze)):  
        for column in range(len(maze[row])):  
            x = column * TILE_SIZE  
            y = row * TILE_SIZE  
            tile = tiles[maze[row][column]]  
            screen.blit(tile, (x, y))  
    player.draw()
```





# Moving The Character

The control for this game will be a bit different than previous game. Because the game uses column and row as the map, the character should only move one space at a time, according to the tiles.

```
def on_key_down(key):  
    row = int(player.y / TILE_SIZE)  
    column = int(player.x / TILE_SIZE)  
    if key == keys.UP:  
        row = row - 1  
    if key == keys.DOWN:  
        row = row + 1  
    if key == keys.LEFT:  
        column = column - 1  
    if key == keys.RIGHT:  
        column = column + 1  
    player.x = column * TILE_SIZE  
    player.y = row * TILE_SIZE
```



# Restrict Where You Can Move

When you tried the code just now, you can see that the sprite can move to places it should not. So to solve it, delete the last 2 line from the code, then add these on. It will prevent the sprite to move in or across the wall.


```
tile = tiles[maze[row][column]]
if tile == 'empty':
    player.x = column * TILE_SIZE
    player.y = row * TILE_SIZE
elif tile == 'goal':
    print("Well done")
    exit()
```



# Animate Player Movement

Make sure if you are using your own sprite, the size is 64x64 pixels. It will fit nicely in the tiles. The movement of the Actor is sudden and jerky. Luckily Pygame includes a function to do smooth movement for us automatically. Find these lines of the program and replace it with the second.

```
if tile == 'empty':  
    player.x = column * TILE_SIZE  
    player.y = row * TILE_SIZE
```



```
if tile == 'empty':  
    x = column * TILE_SIZE  
    y = row * TILE_SIZE  
    animate(player, duration=0.1, pos=(x, y))
```

You should see the player moves smoothly than before.



## Create an Enemy

Now we are going to make it a bit difficult for the player. Using the Dragoons as the enemy, we are going to make it move around in a specific motion. So first you need to create the variable and velocity for the enemy.

```
player = Actor("knightright", anchor=(0, 0), pos=(1 * TILE_SIZE, 1 * TILE_SIZE))  
enemy = Actor("dragoons", anchor=(0, 0), pos=(3 * TILE_SIZE, 6 * TILE_SIZE))  
enemy.yv = -1
```

We use the velocity so it will move up and down in a smooth motion in the empty space, then once it hit a wall, it will reverse the velocity .



# Enemy Movement Code

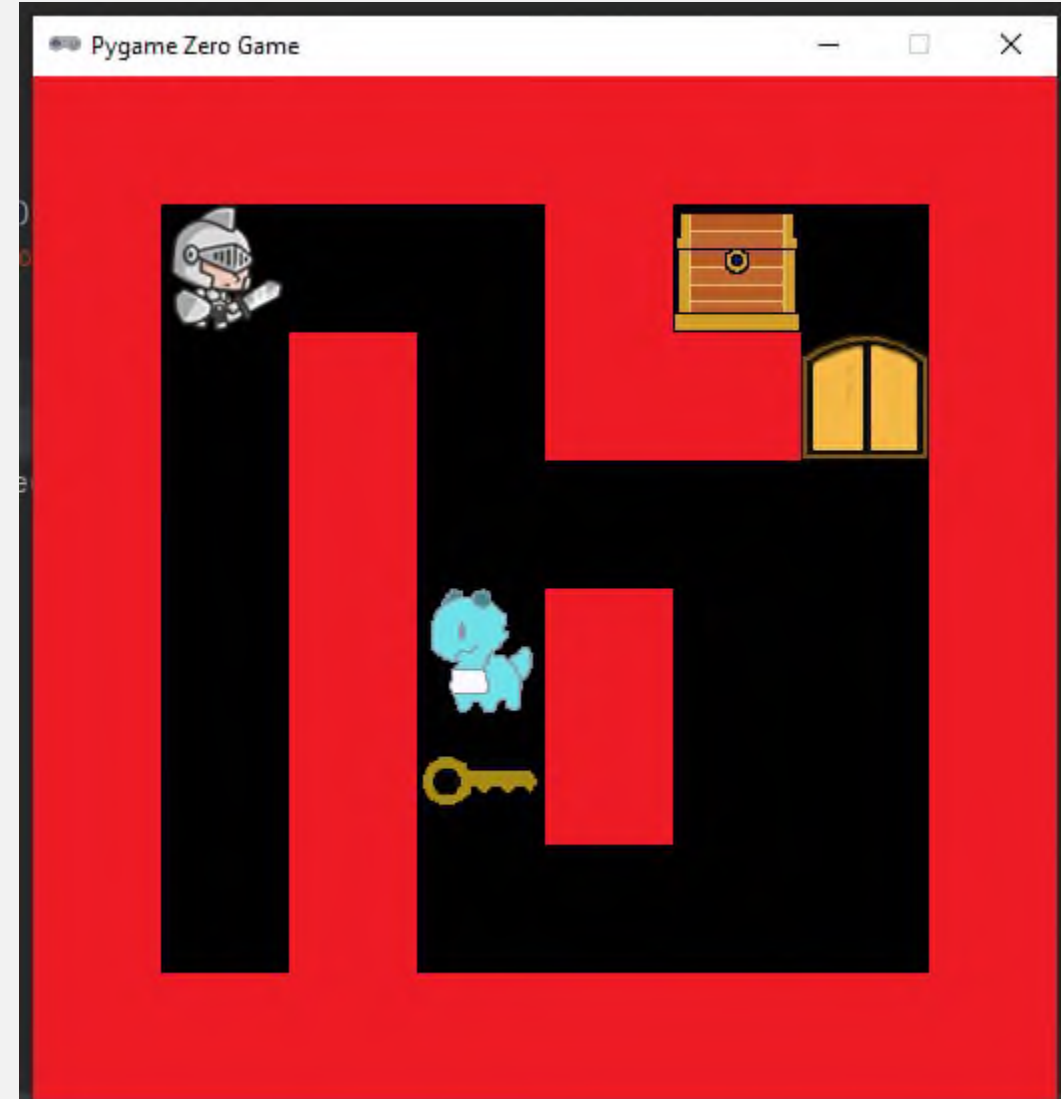
This code is still under the on\_key\_down function.

```
# enemy movement
row = int(enemy.y / TILE_SIZE)
column = int(enemy.x / TILE_SIZE)
row = row + enemy.yv
tile = tiles[maze[row][column]]
if not tile == 'wall':
    x = column * TILE_SIZE
    y = row * TILE_SIZE
    animate(enemy, duration=0.1, pos=(x, y))
else:
    enemy.yv = enemy.yv * -1
if enemy.colliderect(player):
    print("You died")
    exit()
```



# More Challenge

Right now, we don't have any winning condition for our game. So we are going to add 2 more tiles inside which will be our key and door. The way to end the game is by first getting the key which will have the enemy patrolling, then unlock the door to get the chest.





## Add to the Tiles

Inside the tiles list, add these two new sprites. Doors and key. Make sure it is already inside the image folder. After that, create a variable called unlock. This will store the number of keys the player is carrying.

Once the variables and list is done, lets change the tile map so the key and door will appear on the map.

```
tiles = ['empty', 'wall', 'trechest', 'door', 'key']  
unlock = 0
```

```
maze = [  
    [1, 1, 1, 1, 1, 1, 1, 1],  
    [1, 0, 0, 0, 1, 2, 0, 1],  
    [1, 0, 1, 0, 1, 1, 3, 1],  
    [1, 0, 1, 0, 0, 0, 0, 1],  
    [1, 0, 1, 0, 1, 0, 0, 1],  
    [1, 0, 1, 4, 1, 0, 0, 1],  
    [1, 0, 1, 0, 0, 0, 0, 1],  
    [1, 1, 1, 1, 1, 1, 1, 1]  
]
```



# Test for Key and Door

First, find the if statement for the goal. Replace it with the second code here. So now the game will also test for the key and door tiles.

Since we are using the global variable(unlock), make sure to write the code at the start of the function.

1

```
elif tile == 'goal':  
    print("Well done")  
    exit()
```

2

```
if tile == 'goal':  
    print("Well done")  
    exit()  
elif tile == 'key':  
    unlock = unlock + 1  
    maze[row][column] = 0 # 0 is 'empty' tile  
elif tile == 'door' and unlock > 0:  
    unlock = unlock - 1  
    maze[row][column] = 0 # 0 is 'empty' tile
```

3

```
def on_key_down(key):  
    global unlock  
    row = int(player.y / TILE_SIZE)  
    column = int(player.x / TILE_SIZE)
```

Note : replace the goal with 'trechest' if you are using my images for the goal.





# Complete Code.

```
import pgzrun
TILE_SIZE = 64
WIDTH = TILE_SIZE * 8
HEIGHT = TILE_SIZE * 8

tiles = ['empty', 'wall', 'trechest', 'door1', 'key']
unlock = 0

maze = [
    [1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 1, 2, 0, 1],
    [1, 0, 1, 0, 1, 1, 3, 1],
    [1, 0, 1, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 1, 0, 0, 1],
    [1, 0, 1, 4, 1, 0, 0, 1],
    [1, 0, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1]
]

player = Actor("knightright", anchor=(0, 0), pos=(1 * TILE_SIZE, 1 * TILE_SIZE))
enemy = Actor("dragons", anchor=(0, 0), pos=(3 * TILE_SIZE, 6 * TILE_SIZE))
enemy.yv = -1
```

```
def on_key_down(key):
    global unlock
    row = int(player.y / TILE_SIZE)
    column = int(player.x / TILE_SIZE)
    if key == keys.UP:
        row = row - 1
    if key == keys.DOWN:
        row = row + 1
    if key == keys.LEFT:
        column = column - 1
    if key == keys.RIGHT:
        column = column + 1
    tile = tiles[maze[row][column]]
    if tile == 'empty':
        x = column * TILE_SIZE
        y = row * TILE_SIZE
        animate(player, duration=0.1, pos=(x, y))
    if tile == 'trechest':
        print("Well done")
        exit()
    elif tile == 'key':
        unlock = unlock + 1
        maze[row][column] = 0 # 0 is 'empty' tile
    elif tile == 'door1' and unlock > 0:
        unlock = unlock - 1
        maze[row][column] = 0 # 0 is 'empty' tile
```

```
# enemy movement
row = int(enemy.y / TILE_SIZE)
column = int(enemy.x / TILE_SIZE)
row = row + enemy.yv
tile = tiles[maze[row][column]]
if not tile == 'wall':
    x = column * TILE_SIZE
    y = row * TILE_SIZE
    animate(enemy, duration=0.1, pos=(x, y))
else:
    enemy.yv = enemy.yv * -1
if enemy.colliderect(player):
    print("You died")
    exit()
pgzrun.go()
```



## Complete + Improvement tips.

By the end of this, you should have a Maze game.

You can add much more feature to the game like the player will face the direction it moves( move left , face left and vice versa), door opens up animation, coins that act as scores like in pacman game and much more.

Let me know if you have added some new features inside the game and I will give some reward.



You can direct message your teacher and ask your question through [Slack Robotene Community](#) or arrange a [One-to-One Consultation](#) with your teacher.



# Any Questions?



Thank you :)