This is a small game written in python with the pygame module. Help MacGyver escape !

Installation

Follow the instruction below to install the game. Make sure you have python-pip installed. Below is the command line installation for Linux.

```
git clone https://github.com/2pie/mac_gyver.git
cd mac_gyver/
pip install -r requirements.txt
```

Launch the game with the following command

```
python game.py
```

My process

1. **Starting with Python.** I did the first three video courses on Python: "Démarrez votre projet avec python", "Découvrez la programmation orientée objet" et "Découvrez le fonctionnement des algorithmes". I also did small projects: a "guess the number" game and a random password generator (not in github). This gave me some basics skills in python.
2. **Starting with pygame.** I followed some tutorials to discover pygame. I learned how to create a window, add a rectangle that can be moved with keyboard arrows, and I also learned to add walls that stops the movement of the rectangle. This last part was not easy, after some research, I found a method in a tutorial that worked. The idea of the method is, after a vertical (resp. horizontal) coordinate change, test if there is a collision with a wall. If there is a collision, set the moving vertical (resp horizontal) coordinates equal to the coordinates of the vertical (resp. horizontal) edge of the wall. From there I structured my code in an object oriented framework. The code was composed of three parts:
    - Building the classes
    - Creating the instances
    - Main loop of the game
3. **Design of the labyrinth.** Once I had an idea of how pygame works, I designed the labyrinth. I used an spreadsheet software to draw the labyrinth in a grid, with wall blocks represented by ones. I then wrote a small R script to generate a csv file from this table that gives the x,y coordinates of each block (not in github, because not necessary to run the

program). I imported this file in my program to create the labyrinth. I added walls at the edge of the screen using the same class.

4. **Creating guardian and note on classes** I created the Guardian using the same class as for the walls. These object are not moving and in my program they don't "do" anything, they are just generated but have no methods in their classes. It is the player that does all the action, and therefore has several method in its class. This way of doing thing has the drawback that the code is concentrated in the player class, and therefore harder to read. However, I find it more intuitive to put all the interactions with other elements of the game in one function "update", than to split it in all classes. Intuitively, the player has the active part, he stops in front of a wall, he kills the guardian he picks up objects, while the other element have a passive role (they are picked, they are killed). It seems therefore more logical to let the player do the action.
5. **Item position algorithm.** The next step is to place the three items needed to kill the guardian. They are randomly placed on the board, and of course, cannot be placed in a wall. I generated random numbers from 1 to 14 (as the number of sprites) and multiplied them by 40 to get the coordinates. I compare this set of x,y coordinates with the coordinates of other blocks (walls, guardians and other items). If there is a match I regenerate a new item position, and so forth until there is no match (i.e. it is not in an existing block). I repeat this for the three items. I also updated the win condition, added a score tracking text, and a win/lose message.
6. **Adding pictures.** Since my basic sprites is 40 pixel large and high at maximum, I cropped a bit the MacGyver picture so that it is 40 pixel high. The guardian pictures already fits in a 40x40 pixels square. I used a tile from the "floor-tiles-20x20.png" file, duplicate it four times to create a 40x40 block for the walls. I used another tile from the same file for the floor. I drew it on every sprites using a loop.
7. **Cleaning code and structuring code** My mentor pointed out some issues with the formating and the structure of the code. Following this discussion I made the following changes:
   - Putting defintion of constants and classes in separate files
   - Using a linting tool to apply the PEP8 guidelines

Classes

Name	Methods	Attributes	Interactions	Notes
Player	Init, move, update	x, y, width, height, picture, vel, change_x, change_y, walls, guard, item, score, victory, defeat	Interact with Block and Item instances in the update method	
Block	Init	x, y, width, height, picture		Fixed objects with deterministic position (walls and guardian)
Item	Init	other_blocks, picture		Fixed object with random position

Design of the board

The board must be 15 sprites long. I make it also 15 sprites wide to have a square. The basic sprite is based on the size of Macgyver's picture, slightly cropped to make it 40 pixel long. 40x15 = 600. So the screen should be 600x600 pixels, composed of a 15x15 grid of sprites of size 40x40 pixels.

References

Starting with pygame

https://www.youtube.com/watch?v=i6xMBig-pP4

https://www.youtube.com/watch?v=2-DNswzCkqk

Walls

https://www.youtube.com/watch?v=1aGuhUFwvXA

http://programarcadegames.com/python_examples/show_file.php?file=move_with_walls_example.py

Using OOP

https://www.youtube.com/watch?v=xfnRywBv5VM