Code Tutorial
Different Environments in Pygame

Embodied AI, VU Amsterdam

Overview

In the second assignment you are asked to simulate the spread of COVID-19. As you might already know, the spread is greatly influenced by our behaviors, but our behaviors in turn by the environment we live in (if there is only one grocery store open, will this increase the spread?). Hence, in order for you to be creative and see how different environments might change the way how the agents behave, we have written for you this tutorial in which you can find information on how to customize the environment for your needs.

Designing Your Own Environment

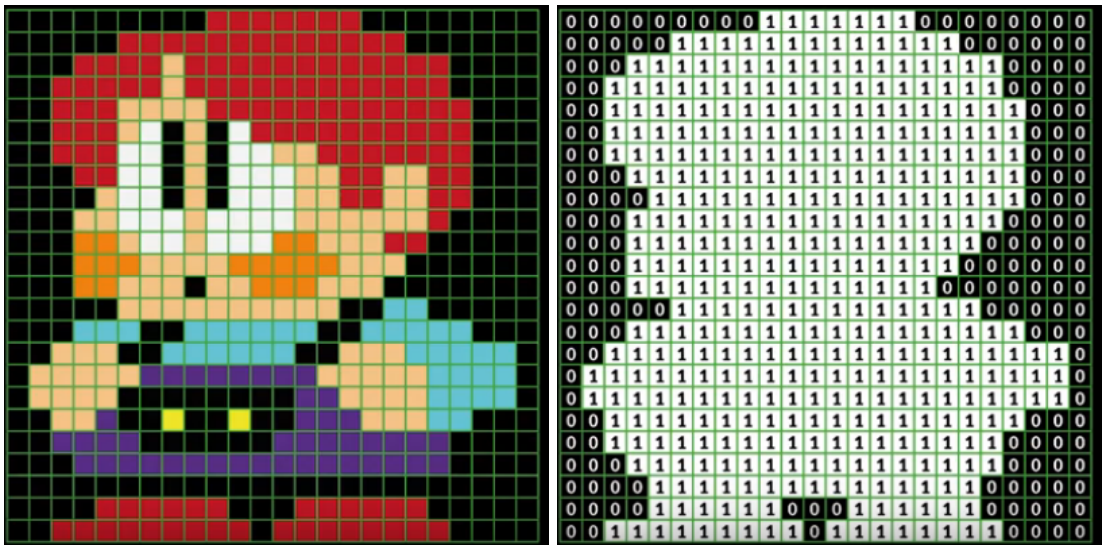
A simple way how to load a complex environment without manually defining every data point, is by creating an image (.png file) and loading the entire image within Pygame. We recommend using an online tool, such as draw.io (link). In this web interface you can create your own environments, and export them to your device as .png files with a transparent background. These conditions are necessary if you want to use the image loading function and the obstacle avoidance function provided by us (*Feel free to do it your own way, but then you might have to spend some hours investigating pygame manual, which is not a focus of this course).

Pygame Functionality

The bullet points below briefly explain the functionality of the code under *class Object* (file objects.py).

- `image_with_rect()`: loads the image in the pygame environment, and scales it if needed
- `get_rect()`: creates a rectangular object of the image, centered at a given position
- `mask_from_surface()`: creates a mask of an image using bits: 1 represents part of the image, and 0 represents transparency. In principle, converts any image to a white-black image (fig 1)

This last feature is of great importance, because based on this mask definition a correct agent initialization (avoid initializing agents on obstacles), as well as obstacle avoidance is achieved (pixel perfect collision).

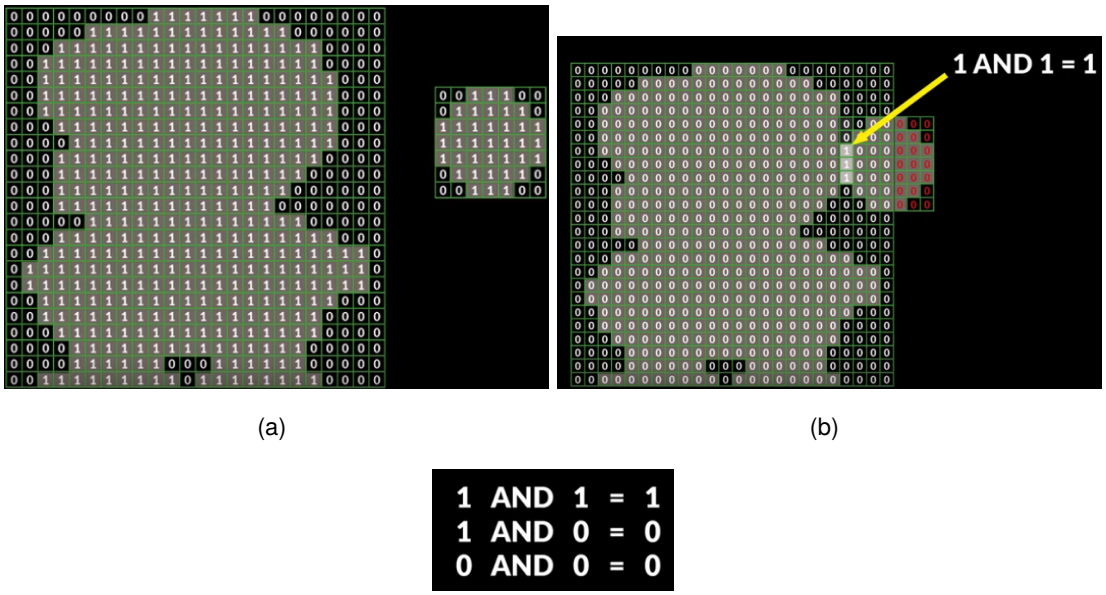


(a) (b)

Figure 1: (a) Pixel image (b) Bitmask

Obstacle Avoidance

To test for collision between an agent and an obstacle in the environment, we can use the created masks (figure 2 (a)). The created masks allow Pygame to perform a simple bitwise operation (figure 2 (b)) by applying simple Boolean logic (figure 2 (c)). Therefore, Pygame allows a relatively fast and simple way for collision detection based on the underlying pixel distribution. For more detailed explanation check pygame documentation (link).



(a) (b)

(c)

Figure 2: (a) Two masks, (b) Collision Detection,(c) Bitwise Logic