Exercise DEV2

The DEV team

January 13, 2016

1 Introduction

In this series of exercises you will be guided through the assignments 1 and 2 step by step. The assignments have been broken up in the relevant parts; an exercise for each part. The exercises each contain 2 parts: in the first one you will focus on the main concept that is related to this exercise, in the second part you will integrate that concept into a pygame application. In this way you will have an aditional concrete example of the concepts you have seen in the lessons with respect to the assignments. And you will learn to handle the complexity of a larger application.

2 Exercise 1 - make a car move

As said in the introduction. you will first focus on a relevant part. Assignment 1 is about cars in a city. We already gave you the city, so let's focus on the car.

The relevant concepts are classes and methods (those are links to the slides). In (very, very) short: A class is a blueprint which defines what data can be stored. An object can be instantiated, and can then store the data you want in attributes. Methods can be used to modify that data.

Console version

- Build a `Car` class;
- Add the attribute `Position`, which will be a simple integer;
- Add the `Move` method that increments the position by one;
- Make a test program that initialises the car and moves it ten times; print the position of the car on the console at every step.

Pygame version

- Get a template to kick-start your pygame application. For example from here. Delete code you don't need, so you have a relative clean application to start with.
- Draw – Now add a shape or picture that will represent the car. For now you can put it in the game loop. Play around a bit with its parameters and make shure you understand how you can move the 'car'.
- Now add the `Car` class from part 1.
- Instantiate a car-object.
- From within the gameloop, call the move-method on the car
- Now use the car-position to change where the car is drawn on the screen.

3 Exercise 2 - make a list of cars move

In the assignment you will have more than one car. These multiple cars need to be stored in a way that makes it easy to use all these cars programatically. For this, we've introduced Lists in the lessons. In a list you typically store a lot of objects of the same type (but, it is possible to store any type). In this exercise you will combine the Car-objects with lists.

Console version

- Build the `Node` and `Empty` classes;
- Add the usual attributes `IsEmpty`, `Head`, and `Tail` to the classes;
- Make a test program that
  - initialises a list of cars,
  - moves each of them ten times,
  - print the position of each car on the console at every step.

Pygame version Expand on the pygame application from exercise 1.

- Add a `VerticalPosition` attribute to the car, so that each car has a different vertical position to distinguish it on the screen;
- Use the list of cars you just implemented to draw a pygame screen where various cars move from the left to the right of the screen.

4 Exercise 3 - moving along checkpoints

In the assignments the cars will not move along positions based on coordinates, but positions based on *tiles* are used. In this exercise we will focus on that concept, however we will use a slightly different example: metro's and metro stations. Our metro is always at a metro station. It can travel between 2 neighbouring stations, but we will not store any positions in between 2 stations. With this exercise you will gain a deeper understanding of Classes, Attributes and Lists. The theory and slides from the previous 2 exercises are applied here.

Console version

- Make a `Station` class, which contains a `Position` attribute and a `Name`; for example: Station(Position(10,40), "Kralinse zoom")
- Make a list of stations;
- Make a `Metro` class, which has an attribute `CurrentStation` and a method `Move`.
- The `CurrentStation` will store a reference to a node in the list of stations;
- In the `Metro` class, the `Move` method changes position to the `Tail`, which is the next checkpoint;
- Make a test program that initialises a list of metro's, and moves them until they all reach the final checkpoint; print the position of each metro (which is now a checkpoint) on the console at every step.

Let's reflect on what you did in the part of this exercise. The metro (or car) still stores its position, however that position is now abstracted away into a station object. This has the advantage that you can reason about station "kralinse zoom" for example, instead of (10, 40). Station still stores its position in terms of coordinates (10,40), because we will need that to draw the stations in pygame. Stations are connected to each other by the linked list. a Node's head (containing station "kralingse zoom") is connected to the tail (containing it's neighbour "Capelse brug").

Pygame version

- Draw a pygame screen with the Stations and the Metro's;
- The various metro move from one Station to the other.

5 Exercise 4 - crossings

Let's continue our voyager by car again. This gives more freedom to travel around the city.

Console version

- Make a `Node2D` class, which contains attributes `TailLeft`, `TailRight`, `TailUp`, `TailDown`, and `Final`; this is effectively the same as a list, but with four possible choices for the `Tail` (we call this a **matrix**);
- Make a class Crossing, with attributes: Position and Name.
- Make a series of crossings and put them into `Node2D`'s; For example: Rotterdam CS, Hofplein, Eendrachtsplein, Beurs and Blaak.
- You have to define which two crossings are connected using the tails. For example: Rotterdam CS's tailRight would be Hofplein.
- In the `Car`, the `Position` will now be a reference to a `Node2D` in the matrix of checkpoints;
- In the `Car`, the `Move` method changes position to one of the `Tail`s, which is the next chosen checkpoint; the choice can be random;
- Make a test program that initialises a list of cars, and moves them until they all reach a specific checkpoint with `Final == True`; print the position of each car (which is now a checkpoint) on the console at every step.

Pygame version

- Draw a pygame screen with the checkpoints and the cars;
- The various cars move from one checkpoint to the other (like the cars in the city assignment).

6 Exercise 5 - bikes

Console version

- Make a `Bike` class that has the `Move` method just like the car;
- `Bike`'s are fast, so the bike moves by two tiles at a time;

- Add a `PrintPosition` method to the `Car` and the `Bike`, which prints where the vehicle is;
- Make a test program that initialises a list contains a mixture of cars and bikes, and moves them until they all reach a specific checkpoint with `Final == True`; print the position of each car or bike (which is now a checkpoint) on the console at every step.

Pygame version

- Add a `Draw` method to the `Car` and the `Bike`, which draws where the vehicle is with the proper texture; the texture is also added as an attribute of both `Car` and `Bike`;
- Draw a pygame screen with the checkpoints, the bikes and the cars;
- The various cars and bikes move from one checkpoint to the other (like the cars and boats in the city assignment).