Simple Book Example

TeXstudio Team

January 2013

Contents

Chapter 1

Introduction

Abstract

It is about the process of starting the game in an idea, to laying it down to be an actual project that can run, it will be separated into main 3 section, the first one is the wave class, second is ball and third is combine the game altogether in the main class

Choosing python as a programming language had a lots of good options in it:

1 Most of AI libraries are made in python

2 The numbers start at 1 with each use of the `enumerate` environment.

3 Another entry in the list

the first step of the game was was to generate a wave and duplicate it on both sides of the screen, not to be hard-codded as the game is made to be infinite function that can generate a wave for me and at the same time i can change in the variables of wave to make it harder for the players, these variable are the wave amplitude[1] or the wave frequency[2] with all of this in calculation, it means that i can make the game harder by making the behaviour unexpected for the next move, also to go extra step, the wave will decrees the gap between each other to limit the player movement

[1]is the maximum or lowest height the wave can go in one point to up or down
[2]number of waves that can go through a fixed distance in amount of time

Chapter 2

Related work

Chapter 3

Technologies

Chapter 4

Timeline

The first step of developing the game was choosing a programming language that would be helpful in both showing graphics on screen and implement an AI algorithm. The only option that was convenient enough was python as:

1 The most famous programming language with AI libraries that have a good documentation.

2 It is OOP, which means I can have a class for each component of the game easily to make an instance for the AI to train from.

3 A library to draw graphic on screen while it won't need heavy CPU usage that won't throttle the process of AI learning

Choosing a library for the game was the first step as it would define the characteristics. I went for **PyGame** because it was nearly the only one that is good enough with documentation to start with, and quit enough, the main focus isn't about making a game that will have that much of physics in it and 3d animation. For now, the imagined picture of the game is a rectangle as a screen that will have two main component of the game, a wave that is on one side of the screen vertically and one on the other side with the only difference is 350px (the starting amplitude is 50px and the screen width is 400px) and a ball that the **only purpose for it is to survive as much as it can, without hitting any of the sides of the wave**.

I'm minimalist when it comes to developing things, like there wouldn't be splash screen all over, and hard controls, not even hard rules, and it will be the approach I'm following with the game, to break it into steps:

1 The accent colour of the game will be black as a background

2 White will be used to show elements

3 Score counter on top

    3.1 AI mode: show generation and genome number

    3.2 AI mode: show total runtime

    3.3 AI mode: show the vision of the ball

During the writing here, I will raise some questions that might come to your mind while working on some parts (as they might have came to me too) and will try to answer them at the end of every section in the chapter.

4.1 Develop the game

The files layout of the game will be an `AI.py` file in the root folder, then subfolder named `SurviveLine` with 3 files in it `ballFunc.py`, `waveFunc.py` and game.py . To make it easy to make instance of the game, will create a file named __init__.py that will only have one line it it from .game import Game that means we will have a Class Game(): in the game.py and it is used to call the game function as a library in the AI.py file (as it is in another folder) and make instance from it. Every major component will have its own class in file to refer later.

4.1.1 Wave Functionality

To get the base function of wave, there would be a lot of functions to cover like:

```
def draw(self, Display):
        #increase the FPS of game
def changeSpeed(self):
        #change the wave aplitude and increase the wave gap
def changeWave(self):
        #generate a new point on Y axis
def generateWave(self):
        #add point to the list of points
def addPoint(self, index, point):
        #check if there is a gap
def checkGap(self):
        #function to fill it
def fillGap(self, gap, gapDirection):
        #reset all the self. variable that are made in __init__ class
def reset(self):
```

most of them are self explanatory, but the ones that need more dive into details are the generateWave , checkGap and fillGap .

Generate wave

The starting point of the game, in the waveFunc.py to make a main class Class Wave(): with an equation that can generate a wave and at the same time I can change in the variables of the wave to make it harder for the player. These variable are wave amplitude[1] or wave frequency[2].

With all of this in calculation which means that I can make the game harder by making the behaviour unexpected for the next move, also to go extra step, there will be a decrease in the gap between the two waves to limit the player's movement.

```
pointsList_XCord = int((self.HDisplay/2) + self.WaveAmplitude*
↪  math.sin(self.waveFreq * ((float(0)/-self.WDisplay)*(2*math.pi) +
↪  (time.time()))))
```

as you can see, there are some variables that have the self. before, that are defined as:

[1]is the maximum or lowest height the wave can go in one point to up or down.
[2]a number of waves that can go through a fixed distance in amount of time.

```
def __init__(self, wDisplay, hDisplay):
        self.WDisplay = wDisplay
        self.HDisplay = hDisplay
        self.ScoreCount = 0
        self.waveFreq = 1  # changes difficulty part
        self.WaveGap = 0
        self.GameSpeed = 2  # to increment the difference in time to
        ↪  speed the FPS
        self.FPS = 60
        self.WaveAmplitude = 50
        self.PointsI = 0  # index to loop inside the points list
        self.PointsList = [0]*800
```

Listing 1: the self variable in waveFunc file.

These are the variables that are only (and not specifically) linked to the wave functions, and this is where an important functionality of OOP comes in. Encapsulation is the OOP functionality which means to get all the related data to a class (which is wave class at this point), if it is needed in other classes, then an instance of class wave can be made then the new variable can be used from it.

The equation in figure 1 is going to store the X axis coordinates in a list called `PointsList` for the sake of adding points to it once they are generated and show them on screen one by one as if it is loading, because if there isn't a list, then the wave would be a steady visual sine wave (without changing amplitude or frequency yet), this part of code is placed in `def generateWave(self)` function.

There have to be more condition to make the points be generated without disorder, like one point won't be in the other half of screen, which means that when the 350px is added to it, it will be out of the borders of the display.

4.2 Display the game

4.3 Create AI

4.3.1 101 AI

4.3.2 What is N.E.A.T?

4.3.3 Tweak AI

4.3.4 Observation

4.3.5 Explain the log

Chapter 5

Testing

Chapter 6

Summary