

|  |
| --- |
| **VILNIAUS KOLEGIJA**  **ELECTRONICS AND INFORMATICS FACULTY**  **SOFTWARE DEVELOPMENT DEPARTMENT** |
| **Artificial Intelligence**  PI18E |
| Snake game  Final project for exam  6531BX028 PI18E |
|  |
| |  |  |  | | --- | --- | --- | | STUDENT | 2021-03-15 | Phumin Prachanimitchai | | LECTURER | 2021-03-15 | Dainius Savulionis | |
| 2021, VILNIUS |

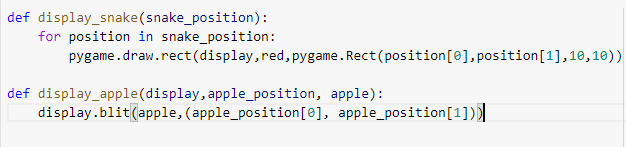
Stages of development

1. A gui to represent our game
2. A function to calculate the angle between the snake and the fruit.
3. A neural network training data , and the model
4. Applying the final model to the game.

# Stage 1: Creating the Gui

The interface is developed with the Pygame , in python.

At the start of each game, we want snake’s starting position to be fixed while the apple can take any random location**.** Starting length of the snake is 3 units where each unit is a 10×10 block.

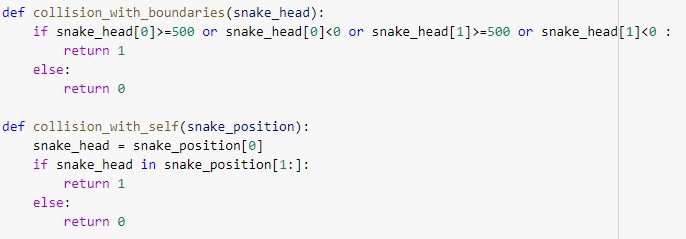


* **pygame.draw.rect()** will draw a rectangle corresponding to given arguments which will represent our snake **and display.blit()** will show the image of an apple.

## Game Logic

**Rule 1:If the snake collapses with itself or with boundaries, game over**

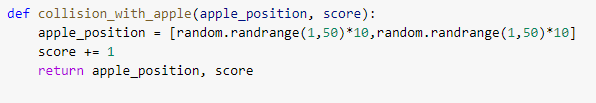
* If the X coordinate of snake’s head is more that 500 , then we can consider it as a collision with boundary , similarly if the Y coordinate is more than 500 , then it will be a collison
* Now, to detect the collision with itself , we can check if the head of the snake is not in the list of coordinates containing the (X,Y) positing of snake.

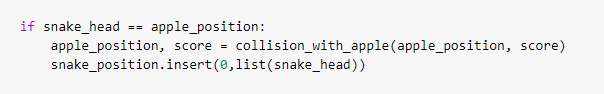
****

**Rule 2: Keep the snake moving.**

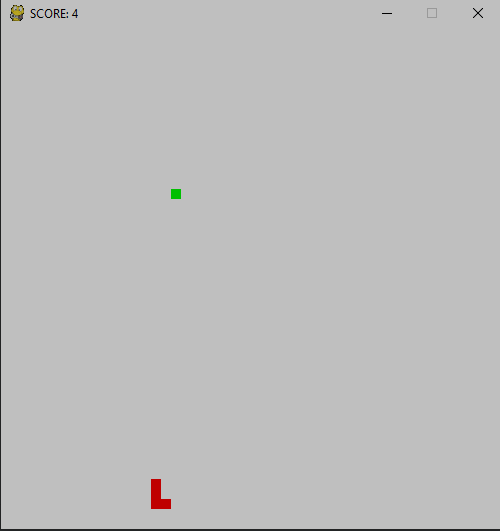
* Until any key is pressed , we will continue to move the snake in one direction, to move the snake , we will add one unit to the head , and remove one unit from tail.
* If any button is pressed , then we need to change the position of the snake’s head.

**Rule 3: If the snake eats the fruit then , the apple moves to a new position and the length of the snake is increased.**

****

****

**The final gui of the snake game will be as below.**

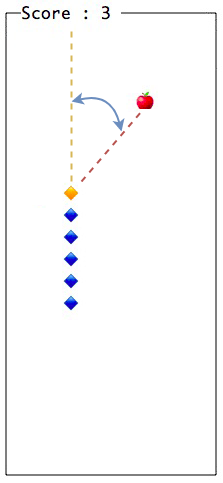
****

# **Stage 2 : Determining the angle between the snake and the fruit , and a function which will decide where the snake should move.**

Measuring the angle between snake and apple.

The current position of snake and apple is required first.

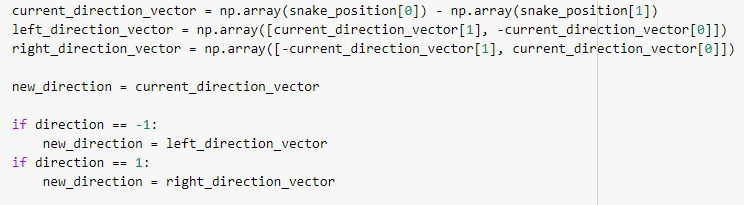
* First calculated the snake’s current direction vector and Apple’s direction from the snake’s current position.
* Snake’s direction vector can be calculated by simply subtracting the 0th index( i.e snake’s head ) of the snake’s list from the 1st index (i.e snake’s tail ).
* Now, the apple direction from the snake can be measured by subtraction of Snake’s head coordinates for the apples coordinates.
* Normalizing these values , will give us the required values.



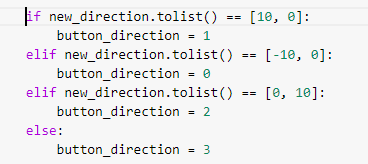
Now , we can decide in which direction the snake can move.

* The calculated angle can be used to determine the angle.
  + If the angle > 0 , then apple is on the right of the snake.
  + If the angle < 0, then apple is on the left of the snake
  + If the angle = 0 , the apple is in the same direction of the snake.

After this we need to calculate the left , right vector with respect to the snake.

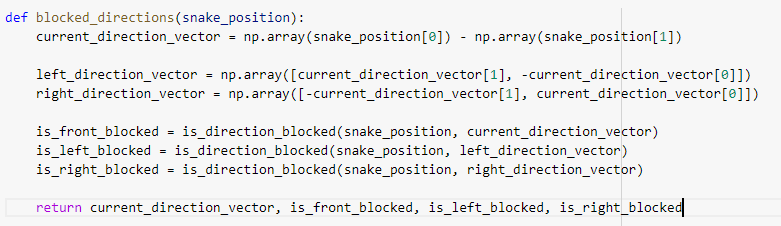
****

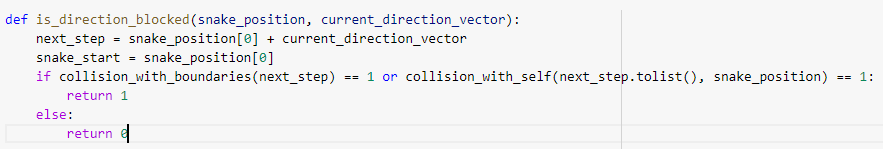
**With the direction\_vector we can now combine this with the GUI event to move the snake.**

****

**Also , we need to check if the next step is valid , or if there are any violations of rule that will affect the game or score.**

* Check if the current direction is incorrect , for this we can create a look ahead function in every direction.

****

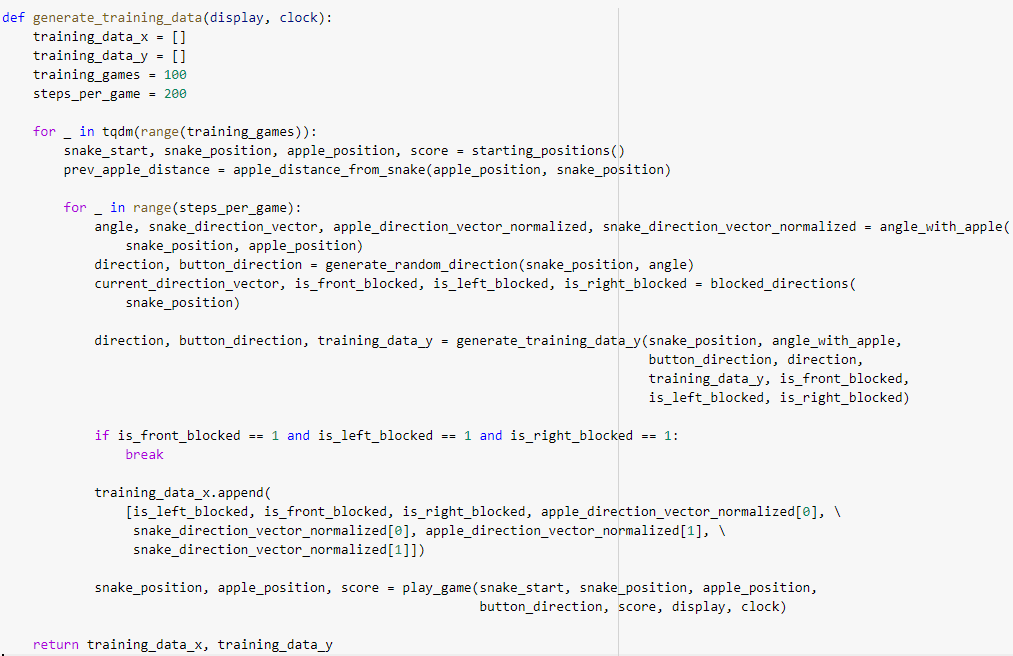
****

## 

# Stage 3 : Develop training data and neural network.

For , every step in the game generated we do:

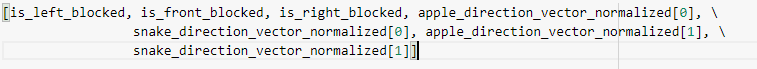
* Calculate the vector of the snake and the apple
* Convert the direction into ( -1, 0 ,1 ) respectively to feed the functions.
* To predict the next direction , check if the direction is blocked or not , and according to that create the output for the model.



**Developing the Neural Networks.**

For Neural Network , we have used **Tensorflow 2.0 with Kears** as a High-end api above tensorflow.

Our input for the network will be a numpy array of dimension (7,0) as shown below:



To decide the architecture of the neural network m we opted for hit and trial method.

We have tried 5 models , for the neural network.

* **1 hidden layer with 9 neurons.**
* **1 hidden layer with 15 neurons.**
* **2 hidden layer with both 9 neurons**
* **2 hidden layers, one of 9 neurons and other of 15 neurons.**
* **2 hidden layers, both with 15 neurons**

From all the proposed architecture , 2 hidden layers, one with 9 neurons and other with 15 neurons, performed the best in average score during the training process.

Network architecture Details:

* Used Sequential mode , where we can stack layers on top of each other.
  + The sequential model each layer has exactly one input tensor and one output tensor.
* The input layer has 7 neurons and is densely connected to next layer.
* The 1st hidden layer in having 9 neurons , and the activation function is a nonlinear function RELU .
  + As RELU does not activate all the neurons at the same time , which will increase our model performance.
* The 2nd hidden layer has 15 neurons and is densely connected to the next layer.
* The output layer is having dimension 3 , as there are only 3 moves possible, and the activation function is SOFTMAX.

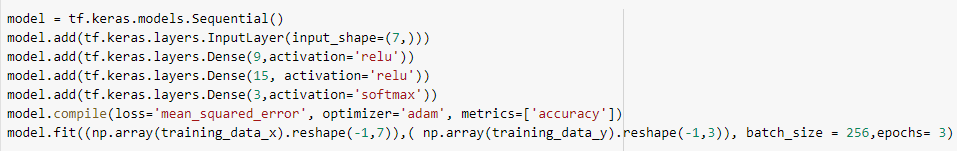
**LOSS Function:**

* The model uses the **Mean Squared Error** to calculate the loss during the training process.

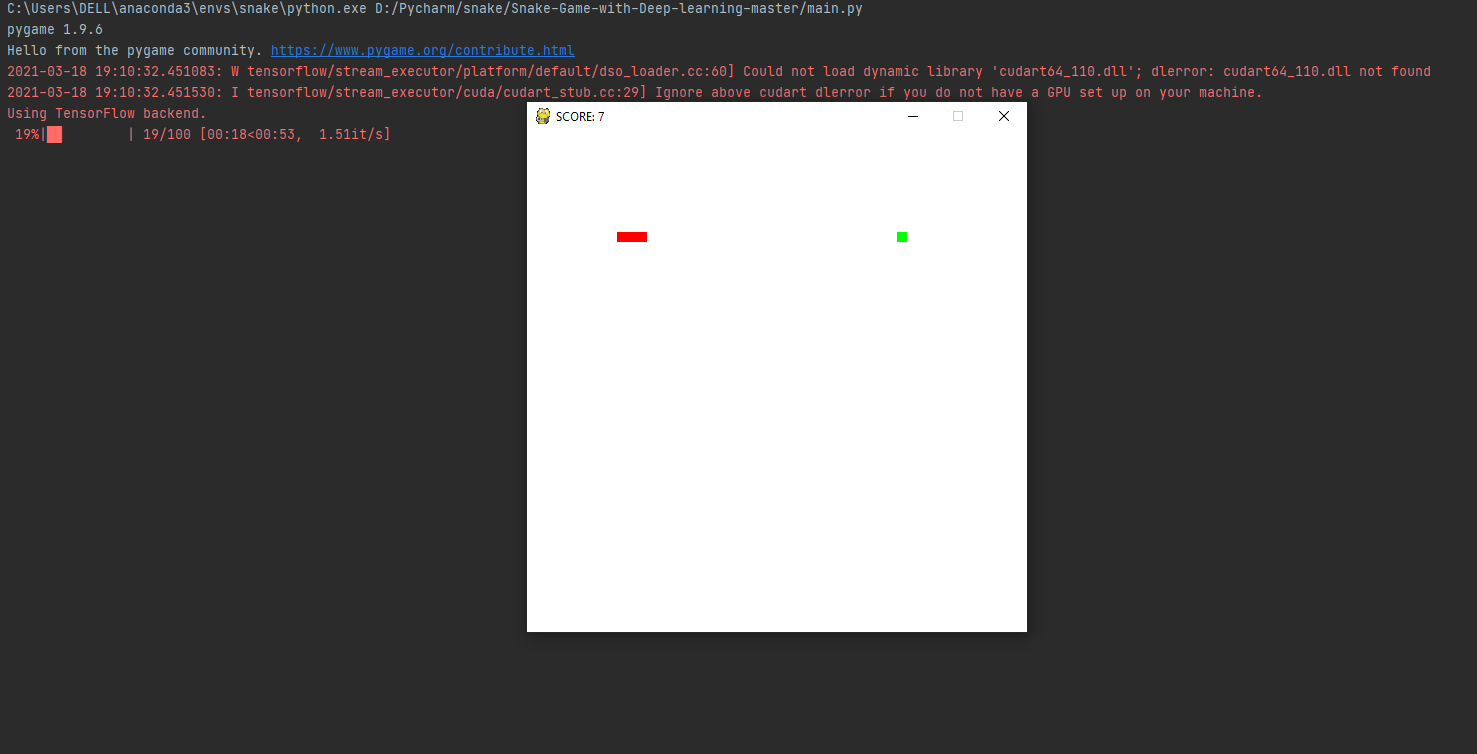
**Optimizer function:**

* The model used **ADAM** function as it is vastly used and performed best in our test cases.

We have set **Epoches = 3** and **training\_game for 1000**.



Training the Neural network.



With **1000 training games** , and **2000 steps per games** , we have generated **1622161 training samples**, and with the final model we got a **higher score of 69** and **average score of around 22**.

Now, we can save the model and its weights , to apply it to our game.

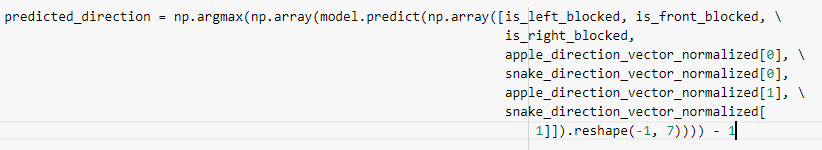


# Stage 4 : Applying the final model to the Game

The trained model can be loaded as:

**model = load\_model('model.h5', compile = False)**

We can predict the direction as:



Now , our game will be controlled by our trained neural network.: