ESZTERHÁZY KÁROLY CATHOLIC UNIVERSITY

INSTITUTE OF MATHEMATICS AND INFORMATICS

AI beats a game

Author
Ahmed Mohamed M. Mahfouz Gadalla

Supervisor
Dr. Kovásznai Gergely

EGER, APRIL 2023

Contents

Dedication

This journey of three years is coming to an end, but it didn't come to this point in the period of only these years. It is the sum of my life up until this point. This is why I want to thank, by far, my parents, **Amal Hady** and **Mohamed Mahfouz**. Always putting my priorities before theirs, being the important part of me, and making me the person I am today. Thanking my mom for taking care of me with all the characteristics (good and bad) in myself. The idea of having an AI that beats the game is inspired from my dad winning over me when I was a child in racing games. I decided to make a game where I wouldn't have to touch the controller to get a higher score than him. From my heart, I thank both of you.

Now that the emotional part is over. There are people who shaped the programmer inside of me today, to the point where he is capable of developing the thesis that is in front of you, my classmates, and my supervisor, **Dr. Kovásznai Gergely**. His work with me through the studying period, sending emails for him back and forth, really can make it up to him. Not only by guiding me, but also by giving me the steps of what to do.

Chapter 1

Introduction

As much as AI technology has been evolving rapidly over the past few decades, and its applications in the gaming industry are becoming more prevalent. Starting from making the enemies more aware of their surroundings, to making an infinite game with a real-time generated environment. This paper will talk about the development process of an AI that plays an indie game and evaluate its performance in comparison to human players. It will analyze the:

1 Related work of the previous papers and projects that have been published about topics that are similar in neural network technologies.

2 Technologies that have been used to make this application come true.

3 Timeline for each (important) step was taken to make the game actually playable both by AI and a human with the physics behind particles, draw the components on the screen with a non-GUI library and creating genomes of AI.

4 Testing the AI with regard to parameters such as threshold and the highest score it can get after a specific period of training time. Explain the inputs it would get and how it will behave after changing each one.

5 Summary to see if the mentioned requirements have been satisfied or not.

In summary, this paper will talk about the steps that are needed to make an AI that is capable of playing a game. In it, you will see a step-by-step explanation of how this is thought of, starting from the reason to have it in first place, to the ending result.

The ending result along with the progress, can be viewed in the repository *Survive Line*.

Chapter 2

Related work

"God is the one who endowed man with reason, and the mind is the basis of everything. The mind is light and knowledge is a result, and so every knowledge is light" -Jābir ibn Hayyān. This work wasn't totally completed in one day, and wasn't completed from beginning to end without the help of online sources. Some of them are in the form of videos, and others are in the form of papers. Most of the sources that will be found here, are more related to the game field to make the graphics of the game and the logic for the player. While the papers have taken the shape of an implemented (Python) library to be used.

On the other hand, the related sources to the field of AI, are implemented as (Python) library to be used. It can be found in papers that require more than average knowledge about basic concepts in this field. I tried to clear most of them in the 101 AI part.

The algorithm itself have been used in other low-budget games with the same purpose of making the environment more interactive, like:

- Galactic Arms Race (GAR): The game is about a galactic war and there are weapons included as part of the game. "In GAR, all player weapons are generated by the (content-generating NEAT) cgNEAT algorithm based on weapon usage statistics. However, cgNEAT does not simply respawn weapons that people like. Rather, it creates new weapons that elaborate on the ones that have been popular in the past" *Galactic Arms Race: Research*.

- Dance Evolution: is interactive application made by the researcher behind the N.E.A.T algorithm "Kenneth O. Stanley". The game allows you to choose a song, and then a set of dancers will dance to it. When you choose a dancer, it will be the base for others to follow.

So as an implementation in the field of games by the creator, there hasn't been a single one that was made recently (in the last 10 years). While there have been

little implementation to the idea of having N.E.A.T algorithm to play a game. One is either developed from ground to up like this tutorial *Python Pong AI Tutorial - Using NEAT*. Others have made it on an open source game from another library and add their own tweaks to the library to fit the game, like *A.I. Learns to Play Sonic the Hedgehog - NEAT Explained* where he takes Sonic the Hedgehog module from Gymnasium (formally known as GymAI by openAI) and implement the N.E.A.T algorithm on it.

My implementation will be different from all of the previous examples because:

1 All the components in the game are pixel-generated, means that there isn't a single image used in the whole game.

2 A full in-depth explanation of the AI is discussed here. As this is the first AI project for me, I elaborate on the details here as if it were my first time learning it. The methodology used in the questions part is the Feynman Technique.

Chapter 3

Technologies

The project itself was purely an idea I got as an application for AI. There would still be a need for sources to help me get turn idea into something real that is applicable. The problem I had mostly with the implementation was getting the documentation of PyGame and knowing what the best implementation was for each function I wanted to implement. While making an a game for PC, you have to think wisely about the sources and how to use them in your favour.

3.1 Running library (PyGame)

The base for the game is made on *PyGame* library. It is "a set of Python modules designed for writing video games. PyGame allows you to create fully featured games and multimedia programs in the Python language". You can also control anything in the game itself, as you would have a main loop that you can think about as the frames per second of the game. Getting to know the library was easy, because of my past experience with Python, and the documentation was easy. The library has been around for some time now, which made it easy to get a 101 guide on how to get the basics.

There was one main part while making the game, developing the GUI. Making the interface for the game was a little hard as it would take much time to learn the drawing functions, such as writing a text on the screen. During the AI training, there is information that shows on the screen to make reading the log easy, like the **score** and **generation number** along side with the **genome ID**. Writing the info isn't as easy as dragging and dropping like most gaming engines, but everything has to be coded on the screen with x and y coordinates.
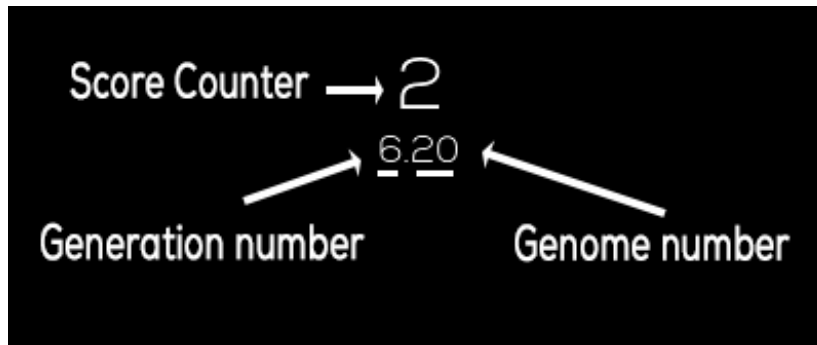
Figure 3.1: illustration for info on the score side

What is good about the PyGame library is that it allows me to have dynamic game dimensions. The screen resolution for the game while developing it was $400 * 800$, but as most of the components on the screen are made in relation to the screen resolution the user define from the code, then you wouldn't have to change every component location from the code, the game will already do it by itself.

3.2   AI library (N.E.A.T Algorithm)

"**N**euro **E**volution of **A**ugmented **T**opologies. And this is what's known as a genetic algorithm" *Python Pong AI Tutorial - Using NEAT*. Think of it as the way that is used in humans to learn (refer to the example of the kid and the ball in further reading) and the natural selection of the ones that perform well as humans and are smarter, they managed to reproduce until today. Unlike the others, who were not fortunate enough to have what it takes to survive in different scenarios.

In other words, to explain it "There is a larger category called TWEANN stands for **T**opology and **W**eight **E**volving **A**rtificial **N**eural **N**etworks" *A.I. Learns to Play Sonic the Hedgehog - NEAT Explained* These are algorithms that not only evolve the strength of the connection weight for a fixed network topology but actually evolves both the topology of the network and its weights." The scientists behind the neat algorithm identified three major challenges for tweanns:

1 Meaningful crossover: by tracking genes through historical markings.
   This stops the algorithm from blindly crossing over the genomes of two neural networks and creating unnormally mutated neural networks. There are two ways to progress through a user-specified number of generations, "with each generation being produced by reproduction $either sexual or asexual$ and mutation of the most fit individuals of the previous generation".

   - Sexual: means that the new generation will be made out of the best-performing genomes from **the previous generation**.

- Asexual: algorithm will **generate a random genome** to reproduce with the genomes with the highest fitness score to make the new generation.

2 Speciation: protecting structural innervation through speciation.
That protects new structures, as they are typically low on hidden network numbers, allowing them to optimize with each other on their own. You can say category before we eliminate them. This is done by splitting up the population into several species based on the similarity of topology, connections between neurons, and their weights and biases. They only compete within their species because some of them who aren't performing well at present, can perform well in the future after the right mutation with each other without the need to eliminate them now.

3 Structure complexity: incrementally growing from minimal structure.
It prevents the algorithm from creating complex networks at the beginning of the new generation that may have to later reduce the number of nodes and connections. They did this by starting all networks with no hidden layers between the input and output layers. NN only has a series of connection genes between them, and if it is found to be useful and necessary to tweak the output a little, then it can involve complexity.

They designed N.E.A.T to specifically address each one of the above characteristics. Point (2) and (3) will be explained more in details at the explain the log from timeline chapter.

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 implementing an AI algorithm. The only option that was convenient enough was Python, as follows:

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 graphics 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. It 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). 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. 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

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 mine 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:

```python
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`.

4.1.1.1 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 ampli-
tude[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. To go extra step, there will be a
decrease in the gap between the two waves to limit the player's movement.

```
1 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          def __init__(self, wDisplay, hDisplay):
2                  self.WDisplay = wDisplay
3                  self.HDisplay = hDisplay
4                  self.ScoreCount = 0
5                  self.waveFreq = 1   # changes difficulty part
6                  self.WaveGap = 0
7                  self.GameSpeed = 2   # to increment the difference in time to
                    ↪  speed the FPS
8                  self.FPS = 60
9                  self.WaveAmplitude = 50
10                 self.PointsI = 0   # index to loop inside the points list
11                 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 OOP functionality, which means to get all the related data to a class (that is, the
wave class at this point), if it is needed in other classes, then an instance of the wave
class can be made, and 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. It is used 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. 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 conditions to make the points be generated without disorder,
like one point won't be in the other half of the screen, which means that when the 350
pixels are added to it, it will be outside the borders of the display.

---

[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.

4.1.1.2 Check gap

After generating a point, with the change in amplitude, the next point that is added to the list of points doesn't have a difference of only 1px with the one before it, so it means that there will be a different line segments in the list and a gap between the old point and the new one. To overcome this, after a point is generated, there is a for loop that checks if there is only one pixel gap between it (the new point) and the point before it, either it is minus or negative as the gap can be to left or right side.
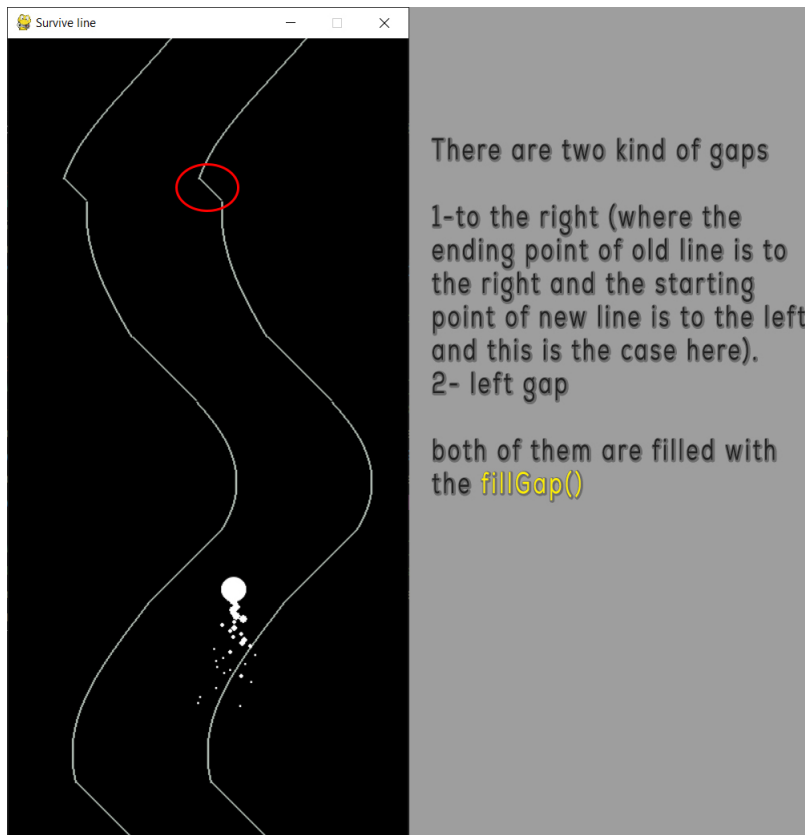
4.1.1.3 Fill gap

If there is one part that took the most time to develop, I would say it is this one, because there were different approaches to solving the problem. The first is to either move the point on the y-axis by the gap and then make a straight line from the old line segment to it. The second was to get the point just to be minus on the x-axis, then be linked to it. The first option was better for the sake of visibility and not affecting the next point, respectively. There were lots of ways (or you can say conditions) that needed to be covered in the point list, for example, what if the gap is at the end of the list? will throw an "out of index error" when trying to shift the new point by the amount of the gap. One way to cover this is by removing an amount of points from the start of the list, then adding the same amount at the end where you need it. That wasn't clear enough, let me explain more.

Say that the gap is over the limit of the list (800Px). Dealing with it before was just to make the gap limited to the end of the list, so if the point is at index 797 and the gap is 10 (that means there will be an "out of index" error at extra index 6), so it was just to make it limited to $\boxed{\text{gap} = \text{DISPLAY\_H - POINTS\_I - 1}}$ but the problem is that it wouldn't work on high scale when the amplitude gets higher.

To deal with it, remove the over-points in gap from the beginning of the list and add empty points of the same amount at the end then make the index go back to the new index, (back to the same example). It will remove 6 points from the beginning of the list, add 6 empty points to the end, and shift the index to 6 points in the back so it stays with the new point.

```python
if self.PointsI + (gap) >= self.HDisplay-1:
        untilEnd = self.HDisplay-self.PointsI
        toAddFromStart = abs(gap-untilEnd)
        del self.PointsList[:toAddFromStart]
        toAdd = [0]*toAddFromStart
        self.PointsList.extend(toAdd)
        self.PointsI -= toAddFromStart
        gap -= 1
```

with every line here looking weirdly by itself, you would need some explanation:

- Line 2: calculates the difference between the ending point of list and the starting point of gap.

- Line 3: get the difference in gap and the point.

- Line 4: delete the amount of point from the beginning of list.

- Line 5: create empty list with the amount of delete points from beginning of list.

Now that the condition has been fulfilled, it comes to fill the gap itself. There would be two options, if the gap is negative or positive, but I will discuss the negative gap, and the other one would have the same implementation with the difference being the sign.

```python
1  if (gapDirection):
2  # to move the point according to gap
3          self.PointsList[self.PointsI + gap] = self.PointsList[self.PointsI]
4          self.PointsList[self.PointsI] = 0
5          #the step is different for gap direction, as it would be -1 or +1
6          for x in range(self.PointsList[self.PointsI-1],
       ↪    self.PointsList[self.PointsI+gap]-1, (gap//gap)):
7                  self.PointsList[insideY] = x+1
8                  if insideY < 799:
9                          insideY += 1
```

First, it moves the first point in the new line segment by the amount of the gap, then resets the old value of it to zero (as it will be part of the straight line). Secondly, there is a for loop to fill the points incrementally, starting from the last point in the old line segment to the new point.

4.1.1.4   Second way to fill the gap

Fill the gap was basically working on the basis of shifting the point on the Y-axis, but there might be another approach to tackle this (the second way I talked about in fixing the problem of the gap).

Thinking that it will take more effort to move the point in the new line segment to the position that corresponds to the gap, then make a line between the old line segment and the new one. That is a lot to think about, there can be a different way. What if we change the point on the x-axis? just to make it close to the old one, I know it is a bit of cheating, but as long as it works, then it is good.

The idea is that, if I can calculate the gap (which I already know) then decrease the new point by the amount of the gap + 1 (if it is a positive gap) and it will be -1 if it is a negative gap. You may ask, "Why didn't you use an absolute value for the amount of the gap as left is the same as right?" because then this would mean that the wave would increment in one way, depending on whether it is +1 or -1.

The newly implemented function is called `def shiftOnXAxis(self, newPoint)` in `waveFunc.py`.

4.1.2   Ball functionality

The main focus when working on the ball was to make it as simple as it could be, so a new instance could be done from it without the need to store a self-genome variable, and every genome would have its own variables that could be changed with a new instance.

4.1.2.1 Draw ball

As the game is based on a **ball** that survives a line, then I need to display a ball and not a circle (google the difference). There isn't a function to draw a filled ball in one line, so I have to draw an empty circle then fill it. The function `pygame.gfxdraw.aacircle` will draw an anti-aliased circle and `pygame.gfxdraw.filled_circle` draw a filled circle inside of it, then draw a fake rectangle around them with `pygame.Rect` that will deal with the collision (will discuss it in the display game section).

4.1.2.2 Generate particles

This part is little on logic than the other because it was made for the visual aspect of the game. There is no output coming out of it to make the game faster or improve something, but it would add a little bit of a characteristic to the game and the vision I have for it.

The particles are made to be in the position of the ball and generate as a way to look like a combustion engine steam coming out ot it, so there are three things to notice here.

- Location: where the particles will start and their ending point.

- Velocity: the amount of particles that will be generated in a second.

- Time: how long they will last on the screen.

With this in consideration, we can start writing a function for it

```python
def generateParticles(self):
        Loc =[self.ballCordX, self.ballCordY]
        Vel = [random.randint(0, 20) / 10 - 1, -3]
        Timer = random.randint(4, 6)
        self.Particles.append([Loc, Vel, Timer])
        for particle in self.Particles:
                particle[0][0] -= particle[1][0]
                particle[0][1] -= particle[1][1]
                particle[2] -= 0.1

                pygame.draw.circle(self.GameDisplay, (255, 255, 255),
                ↪  [int(particle[0][0]), int(particle[0][1])],
                ↪  int(particle[2]))
                if particle[2] <= 0:
                        self.Particles.remove(particle)
```

In the `Vel` variable deceleration part, it makes sure that the value we would get, would be a random number between {-1, 1}. The `Timer` to give chaos to the particles so not all of them are released at the same time.

The code would add to the list of particles a new particle with these random starting values, then the `for loop` process each value on its own.

- Line 7: it process the position on X-axis to the velocity also on the X-axis, same would happen to the Y-coordinates.

- `particles[2]` is to reduce the particle radius by 0.1 in every frame (which is every loop then).

- If condition at the end to remove the particle from the list so it wouldn't take much of space with more runtime.

This function is possible thanks to *Particles - Pygame Tutorial*

4.2 Display the game

Getting all of the parts coming together is based on logic, as there are lots of changes in it with variables in each game component `waveFunc` and `ballFunc` but the basics are made in the game. Now about displaying them from the `game.py` file that will deal with all game components as an instance, like the string that will keep all of them tied up.

4.2.1 Draw

There are basic things in the game that a user would need to always see during the runtime of it, like the ball to know its position and the wave so the ball won't hit it, maybe the particles to add some visuality to the game, and the score on top. Most of these functions are already in the components class, but they need to be called here, in the `game.py` file.

4.2.1.1 Update label

In order to type anything on the screen in PyGame, you need to:

1 Instantiate first the font you will use, how big it will be.

2 Decide what to write then make a position for it on the screen.

3 Blit (show on screen).

17

It can be thought of as a harder way just to write a simple text, but it gives more freedom to customize the text before writing it and it's also CPU consuming wise when it comes to graphics options that you might not need, like anti aliasing.

4.2.1.2 Display score

There is a need for a counter in the game, to be incremented each time the main `While` loop in the game makes a new round, so it would work as a way to measure the score of the game (also fitness for AI). This `ScoreCount` will be the way to measure it, but as every loop, that is +60 (FPS of game) points in every second. I think it might be a little bit over, so I will divide it by 200 and place it on the middle top part of the screen. Now the first part of the game requirement is over, with the display of the score.

4.2.1.3 Display AI number

A way to keep track of the training sessions of AI is to find the generation number and genome number, but if you don't understand these expressions, it is ok, I don't either. I will read about them once I reach the AI part and tell you more there. During the learning period, I won't be staring at my screen the whole time, so there has to be some way I can know which genome did better and the time for it. The log will be in CMD after every genome and will use OBS to record the game window only, so I can return to the video at anytime.

4.2.2 Collision

Basics of collision coming ahead. When there is an object that overlaps another one and they shouldn't. To implement this in the game, a collision would be triggered if the ball overlaps any point of the wave points that corresponds to the same x-axis points with it.

In the Ball component class there is a `ballRect` functions, that returns an invisible rectangle that surrounds the ball with lots of important functionality in it. Basically, I can access all the coordinates of every edge in this rectangle that would help in the collision, and this will be the way of detecting the ball.

As for the collision function's working mechanism, it loops through every point in the wave with the range of the ball, in this case, it is calculated by the full height of the screen (800 px) minus the bottom of the ball (238 x-cord for it) $800 - 562 = 238$ this is the starting range. Ending range will be the top of the ball, $800 - 538 = 262$, for the right side of the ball. If the x value for it is over or equal to the x-axis of the points list with the same axis as -50 - the wave gap then there will be a collision. On

18

the left side, if the x-axis of the left side of the ball is less than or equal to the x-axis of the points list with the same axis - 350 - the wave gap then there will be a collision.

I see a hand rising from the back of class. "Why there is a difference of 24 pixels between 238 and 262?" Because the ball radius is 12 pixels that makes the diameter 24 pixels.

```
1  #                              238           ,              262
2  for YCord in range(self.HDisplay-ball.bottom, self.HDisplay-ball.top):
3  if (Wave.PointsList[YCord] != 0) and (ball.right >=
   ↪  Wave.PointsList[YCord]-50-Wave.WaveGap):
4  return runLoop == False
5
6  if (Wave.PointsList[YCord] != 0) and (ball.left <=
   ↪  Wave.PointsList[YCord]-350+Wave.WaveGap):
7  return runLoop == False
```

You might ask, "Why make a rectangle be responsible for the detection of a ball object?" The problem is a limitation in the library itself. To choose an object that you want to have detection on, even pixel-wise, it is 5 pixels on the top for each side that decreases way down until the middle of the ball, so it isn't that much to think about.
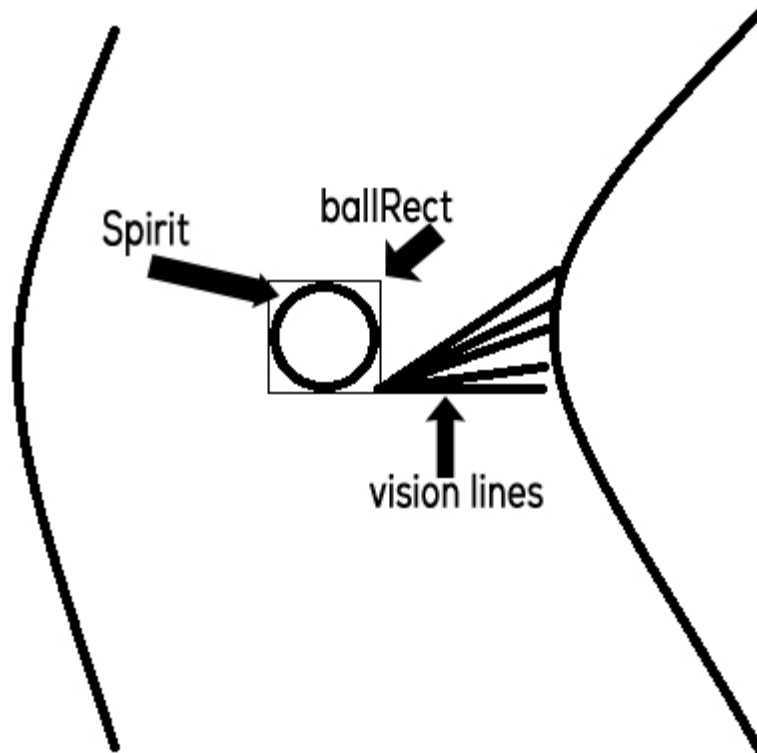
Figure 4.1: Relation between the ballRect and the normal wave

4.2.3   Move ball

Remember what I said about OOP and every class should make what is limited to? that is when it comes in handy, as here I can set the limitation I want for the ball movement, and not in the ball class.

There will be buttons assigned to each movement in the class to move the ball around the x-axis only. If the function gets "right" as a parameter, then it will call the `moveball` function in `ballFunc.py` file to add to the x-axis of the ball, the amount of steps being 5px, also the same with the left side. The limitation is to not get the ball outside the borders of the screen and an extra half of the radius of the ball because it wasn't looking good having the whole ball stuck to the screen side as a limit.

You might ask me. "Why make the parameter for function a string and not a Boolean variable as there are only two types of movement?", I'm not going to lie, I had it during the whole game as a boolean variable, but with the implementation of AI, I changed it to a string. Why? because in the first place I thought the neural network would have two outputs (left and right movement), but there would be a third one, as not moving, or 'centre' in other words, to make the ball steady during the training and not wiggling around.

4.2.4   Count distance

As of the early implementation of the AI, the input was (the order from left to right):

- Distance between the centre of ball to the point in wave with the same x cord both left and right.

- Ball x cord.

But the problem is that the collision covers the whole ballRect (that is 24 pixel) and the NN can only have input of the centre, to solve it, you would need to pass a list of the distance between the points on ballRect and the wave on the side. Here is a big dive into it.

To work on the right distance, there is ballRect that covers the whole ball and is responsible for the collision detection, and there are points with the corresponding x axis on the wave. If any point on the ballRect overlap one of the points (with the same x-axis) on the wave, the collision will be triggered, and this genome will be over. To go more into details, I need to check first with the point on the bottom right side of the ballRect with the same as the x axis on the wave, then move one point up on the wave and calculate it (using Pythagoras). When the points on the wave in the vision for the ball are over, then move one pixel up the ballRect and repeat the same on the wave.

```python
def countDistance(self):
    """
    make a list for distance between the bottom edge of ballRect to the
      corresponding point on the wave
    with the same x-axis, then one step up on wave and repeat, and make another loop
      to do the same with the left side
    """
    ball = self.Ball.ballRect()
    Wave = self.Wave
    rightList = []
    leftList = []
    self.showedLines.clear()
    # the starting point here is the bottom of ball and the end is the top +
      20 px for prediction
    # increased the step size because there was lag for the whole process to
      be handled
    for YCord in range(self.HDisplay-ball.bottom, self.HDisplay-ball.top+20,
      5):
        for YBall in range(0, ball.width, 5):
            dxR = pow(Wave.PointsList[YCord] - 50 - Wave.WaveGap -
              ball.bottomright[0] + YBall, 2)
            dxL = pow(Wave.PointsList[YCord] - 350 + Wave.WaveGap -
              ball.bottomleft[0] + YBall, 2)
            dy = pow(abs((800-YCord) - ball.bottomright[1]+YBall),
              2)

            rightList.append(int(math.sqrt(dxR+dy)))
            leftList.append(int(math.sqrt(dxL+dy)))
            self.showedLines.append(YCord)
    return rightList, leftList
```


Figure 4.2: keep in mind that the starting point is in the middle of the ball in the image, but it is just a simple way to say that there is a vision

Sounds good, but the problem is the two for loops were CPU consuming. Imagine there are 24 pixels, then repeat it again for the wave ones (a two inner for loops) $24^2 = 576$ and another one for the left side $576 * 2 = 1152$, and to process them again on the AI input, that is $1152 * 2 = 2304$ times. All of this is done in one time only in one while loop (not to multiply it by 60 for the FPS of the game), not to add the particles that are being made behind the ball, all of this is a lot. To make it a little better, I reduced the step size to be 5 for each loop, made it 45 in each list, and got

the inserted value to be an integer.

576 lines on each side	45 on each side
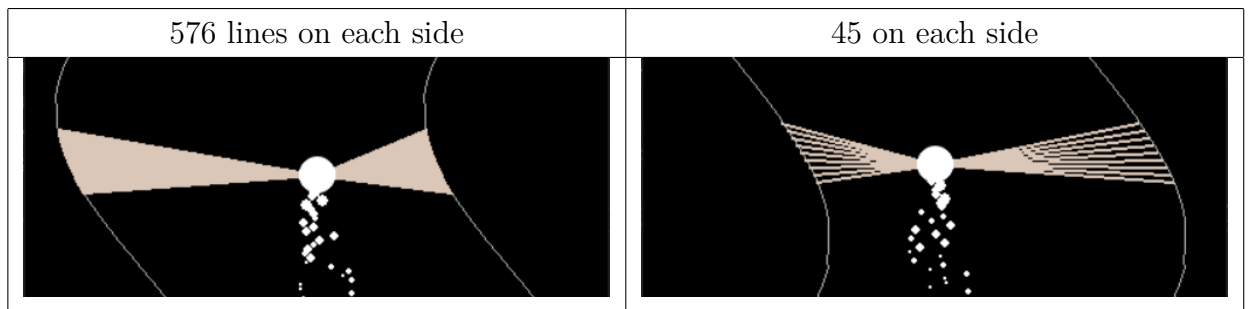
You might ask "Will this reduce the vision of the AI?", Quite frankly, I think if it could handle itself with only one point in the past, then it can do better with 45 lines on each side. It even gave more space to add an extra point, so it can see the future now (an extra 20 points on the wave).

4.2.5 Show Vision

The speed of learning during the AI would be faster than a human could play a game, and due to the amount of loops that are made every time in the main loop `while`, the calculation time is a lot to take, so to make the things smoother, there are vision lines of the ball that would help in viewing what the ball is actually seeing (or detecting) on the wave, but there would be a control for it in the `draw()` method so it doesn't slow the process of the game, and by pressing the `v key` it would switch it on or off.

4.2.6 Loop

There are some functions that need to always work during the game. Changing the wave amplitude, increasing the `ScoreCount` by one, and changing the game speed as in FPS. `ScoreCount` is just a holder for them so it doesn't get messy in the AI part.

4.2.7 Reset

Calls the reset function in the `Wave` and `Ball` components classes. It is triggered once the collision happens.

4.3 Create AI

As the progress of the thesis can be separated into two parts. I can say now that the first part is over with making a fully functional game that satisfies the requirements. Now the second part, making an AI that can reach a high score in it, despite the difficulties

that are made with the ball surviving through the game more, such as changing the amplitude and increasing the speed.

There are a lots of algorithms, that can be used to develop such a neural network that is capable of this task. Changing the weights and biases without me interfering in them. I feel like I'm going over lots of expressions here that need to be clarified first.

4.3.1   101 AI

The way for AI to work is to try to mimic the way that the human brain works, but this means that the human brain itself is going to develop such an intelligence that it overcomes it ?!?! It raises some red flags here, but the closest it can get (at least on my humble machine) is to make a neural network capable of solving only one task. That is totally different from the human brain being capable of doing multiple tasks in a short period of time, like taking input from the senses and giving the output in action, without mentioning controlling your breathing and heart beat without you thinking (now you are thinking about them?).

To simplify the process of a human brain. Since a young age when you were a child, you got to learn that something is dangerous or safe, right or wrong, by trying and then learning from your own mistakes, that is called **Reinforcement learning** *Supervised vs Unsupervised vs Reinforcement Learning*, and this type is the main point in making the AI.

4.3.1.1   Life example

Given the example of a child trying to kick the ball, this is the first time a child sees a ball and doesn't know **yet** what to do with it. The child here is called an agent, and the football is the environment that is trying to solve, or in this case, kick it as far as it can be (which they don't know yet). The agent try to touch the ball, but it isn't the goal they are seeking to reach (trying over time). After some time of hitting it harder, they realize it won't hurt them (as a punishment), and if they score a goal, it is good (as a reward).

That is the same way the AI is learning in this game. The ball (which is the agent) is trying to survive as much as it can between the two waves (environment), but it doesn't know if touching any of the waves will end it or not. The ball tries to use one of the three controls it has (left, centre, right) and if it goes totally left, it dies, just like if it goes totally right, but if it stays in the middle, it will survive the most, at least for now.
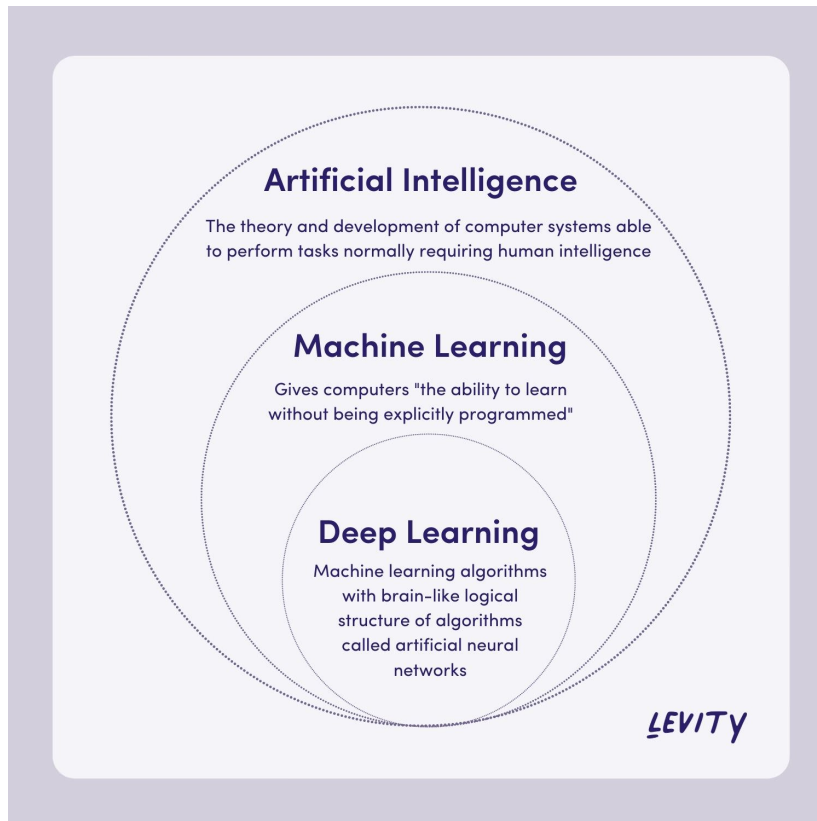
Figure 4.3: how deep learning is connected to machine learning

As mentioned in counter part, there is a counter in the game, it will be used as what is called fitness function . A Fitness function measures how well our genome is doing with a good input for the ball to give a proper output of movement, so the ball survives as far as it can. Here is another unknown expression.

4.3.1.2 Fitness function

There must be a way to reward or punish the AI according to how it behaves during the game. The more the ball survives the wave, the higher its fitness score will be (better its quality). Meanwhile, if it dies in the early stages, its fitness score for it will be drastically reduced as a punishment, and it will also be lower in the hierarchy of genomes in the generation. The possibility for a low fitness-genome to be mutated into a future generation will be lowered.

By applying this method to the game here, there is a way to reward. If the ball stays in the middle between the two waves, that is for the sake if there is an immediate change in the wave curve, the ball is less likely to hit into it.

```
1 leftDis = int (output[0])
2 rightDis = int (output[2])
3 if(len(str(leftDis)) and len(str(rightDis)) > 2 ):
4         if round(leftDis, -1) == round(rightDis, -1):
5                 fitness += 2
```

Explained from the inside out. The fitness is increased by 2, if the rounded output for left distance == rounded output for right distance (the ball is in the middle), but first it checks if both of them are more than 2 two-digit numbers so it can round if the numbers are only hundreds.

As for the punishment. If the genome dies at the beginning of where it was without any movement from it to survive, the fitness score will be reduced. The point here is to try to find a good punishment for it and not overdo it, so it can be balanced.

```
1 if(fitness < 300):
2         fitness -= 20
```

There was an early implementation, to add a reward in case the ball takes the output of staying in the middle and not wiggle around with the right and left only. But the ball managed to overcome it and also waggle (as it sought a change in the input more than the reward it would get), so the other way was to make the distance equal on both the left and right sides is the reward.

4.3.1.3   Neural network

In order to take a decision, there is an input, and then a certain amount of processing is made in between to have an output. This is what is called a neural network, it consists of layers that modify the input along the way to decide what it will be at the end of it, as an output. The small fundamental building piece for NN is called a **neurons** , same as in the human brain (but not the same amount!). There are input neurons that are found in the first layer, then processed in the second layer and modified in the third one, with the summation being in the output layer.

4.3.1.4   Weight and biases

In the early stages of the ball's life, when it is learning how to control its movement, the first steps are either going totally to the left or right. Which isn't really paying off to make it go far. Each neuron has to tweak the parameters it gets from the one before it, the weights and biases are responsible for this, going back to the child example. The
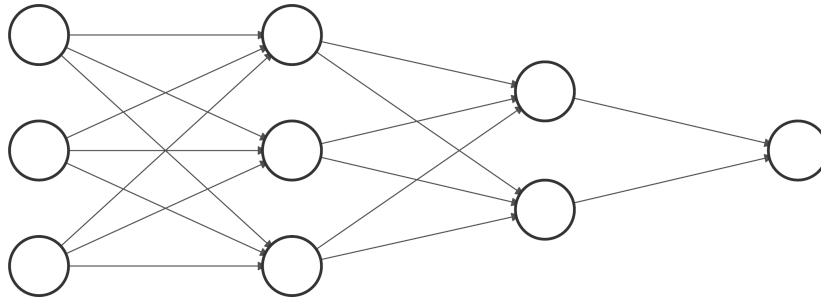
Figure 4.4: relation between input, hidden and output layers

brain consists of little neurons that are responsible for making every decision the child take, it change over time, as is the case here. The brain changes a little bit on the decision and tries a different approach to the problem. One time hit the ball hard to the left side, one time with little power to the right side. "**Weights** control the signal (or the strength of the connection) between two neurons. In other words, a weight decides how much influence the input will have on the output." *Weights and Biases*.

4.3.1.5 Activation function

The connections between neurons and each other have weights and biases that can be altered. Let's say that there is one neuron whose value isn't important and can be used when the input is different, or even if it doesn't exist at all, but only to one specific neuron in the layer after it. If it were to be removed, it would change the value of all the neurons after it.

That is what the activation function is for. Change the value for one specific neuron and decide if it will have more (or less) impact on the neuron that is after it with the connection in between. As the name states, "Activation", the value for it is between 0 and 1 for each neuron.

You might ask "Why use an Activation function, isn't it changing in the numbers in the same way the weights and biases are doing?" Yes and no. The weights and biases change the number **that is going from one neuron** to another, but activation function **change the effect** of the neuron to the one after it.
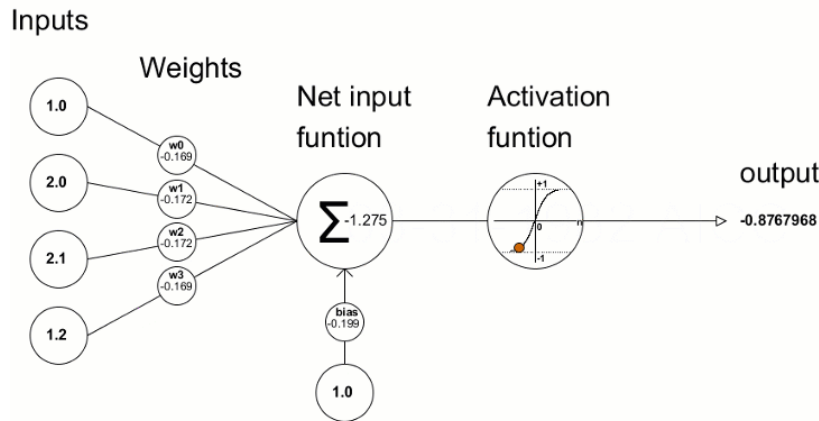
Figure 4.5: how weights, biases and activation function effect each neurons

4.3.1.6 Genome

All the info about the previous parts above can be collected into one thing here. The genome is the collected part for all of this, it works on a neural network that relies on weights and biases to change the input until the output and uses a fitness function. One population exists with a pre-defined number of genomes inside it, it can be called the children of the generation.

4.3.2 Tweak AI

After the base was made for the algorithm to work properly, there were tweaks that had to be made to it. The algorithm itself takes the input from the `config.txt` file, there are some expressions that are needed to be explained in the file to know what to change.

```
1 [NEAT]
2 fitness_criterion     = max
3 fitness_threshold     = 1000000
4 pop_size              = 30
5 reset_on_extinction   = False
```

These are the most important values to look for in this file. They define when the algorithm will stop, and the size of it.

1 `fitness_criterion` state when the algorithm will stop regarding the threshold. There are three values for it `min`, `max`, and `mean`. If it is `max` then it will

stop once there is at least one genome that managed to reach the threshold and define it as the winner, then terminate the process.

2 `fitness_threshold` is the high score that the algorithm is learning to reach. The value written in the file equals 5000 points in the game.

3 `pop_size` how many genomes can be in one population, in case it is more, the learning time for the algorithm will be increased.

```
[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 1.0
activation_options      = relu
```

Going to the activation function, the one that defines the effect from one neuron to another, the value is between 0 and 1. More info about it in Activation function section.

4.3.3 Explain the log

During the runtime of an AI training session, the algorithm displays statistics at the end of every generation. To have a better insight of what is going on behind the curtains, the one here is taken from a training session on the 2nd of January 2023.

```
****** Running generation 99 ******
14 reached the threshold
genome number 15 = 57S with fitness 82
highest fitness now is 20206 generation 37 genome 1
Population's average fitness: 3039.20000 stdev: 4602.73985
Best fitness: 20201.00000 - size: (4, 7) - species 1 - id 2564
Average adjusted fitness: 0.140
Mean genetic distance 1.803, standard deviation 0.759
Population of 30 members in 2 species:
ID   age  size  fitness  adj fit  stag
====  ===  ====  =======  =======  ====
1    99    16   20201.0   0.146    62
2    99    14   8545.0    0.135    78
Total extinctions: 0
Generation time: 324.005 sec (364.069 average)
Saving checkpoint to neat-checkpoint-99
```

There are only **three** lines in this log that I made to be printed during the process. It would make it easier to check the genome's behaviour from the recorded video, and they are:

- line 2: says that from this generation, only genome number 14 managed to reach the threshold, which was 101 points.

- line 3: there is an if statement to output genomes with running time that exceed a specific score (which was 50).

- line 4: store the highest fitness of the genome from the beginning of the session. It got the first threshold, so I can know if my input is being optimized with every generation or not.

Starting from line 5, all that comes, is made by the `reporting()` class in the algorithm source code. What will be written here is an attempt to explain every part of it (and heavily taken from *NEAT Overview - Glossary*):

- line 5 → `stdev` : is the standard deviation of each genome to the over of all mean fitness in the generation, so higher it is, the more difference there is.

- line 6 → `species 1` "Subdivisions of the population into groups of similar (by the *NEAT Overview - Genomic distance* measure) individuals (*NEAT Overview - Genome*), which compete among themselves but share fitness relative to the rest of the population. This is, among other things, a mechanism to try to avoid the quick elimination of high-potential topological mutants that have an initial poor fitness prior to smaller "tuning" changes". More can be found in the AI library (N.E.A.T Algorithm)

- line 8 → `Mean genetic distance` is measurement to the difference (or tweaks) that have been made in the genomes of this generation to their parents from previous generation. As they might have been populated from parents that aren't in the previous generation exactly, check "Meaningful crossover" in the AI library (N.E.A.T Algorithm) section.

- line 16 → `Saving checkpoint to neat-checkpoint-99` : so I can come back and get a live feed from the same generation. It is only valid with the same settings that were used during running it first time, any changes on the configuration will make it unusable. That is why recording the sessions with OBS was useful.

Chapter 5

Testing

At first time running the algorithm, the input vision for the ball to the wave was from the centre of the ball to both sides of the wave as a single point. That made it hard for the ball to find its way (or have a futuristic vision, as you can say). There would be a change in the wave curve and the ball couldn't detect it, so the idea came of having more range for the ball to see. It would calculate the distance between the ball rectangle and the left or right side of the wave in more than one point.

To get more into it with numbers, there would be a notice of the ball having a weird sense of getting to know its "new" sense of wider vision diameter. The ball would take about 20 generations just to start moving more randomly left and right. This is made when the ball had only the vision of its 24-pixel diameter (12 px as radius) and then the extra step of plus 50 pixels, getting this info in a nutshell.

- One point vision: good as start and better CPU wise.

- Diameter vision + 20: best one in score yet (117 points in generation 89).

- Diameter +50 points: No learning even after nearly 300 generation.

The reason to increase the vision for the ball (even though it was working fine) is that I wanted to test how long it would take for the ball to get used to the new (increased) amount of lines. I can tell you that it took long enough.

Figure 5.1: Number of training sessions

At this point in the game, I implemented the increased speed of wave *4 that improved the learning speed. With normal fps, it would take 8 hours and 15 minutes for 37 generations, but the new one (with a limitation of 4 times the speed, not more) takes 5 hours and 15 minutes for 100 generations to work. That is four times the normal running time of a normal pace of game for a human to play it, and the generation threshold to be 300 generation instead of 100. Let the laptop run as much as it needs. It took more than 15 hours to finish 294 generations and 11 genomes, when I went back to check the log, none of them managed to pass the 3000 fitness score. That means that none of them had a good intuition about the lines to move left or right and at least overcome one curve in the wave. From this, the amount of numbers increased = more time in training.

There is a small box that is shown around the ball, it is called ballRect and is mentioned a lot in the Count Distance and Collision. ballRect is shown to check if the genome did terminate for an actual collision or because it reached the threshold, like the case here in this video.

In order to save as much CPU power as possible during the learning process, the box is shown only when fitness is over 50. In addition to some extra visuals in the game, such as the particles behind the ball, all of them can be viewed again with a key for each one:

- v Key to shown **v**ision

- b key to show the **b**allRect

- p key to show the **p**articles

31

Chapter 6

Summary

Here, I will review the findings from the previous sections. I will consider the implications of AI in this paper regarding its performance compared to a human playing a game. I will also include a discussion of any lessons learned from the project and any future work that can be done.

6.1 The game development

The progress of the game took most of the time due to the lack of sources in the PyGame library with the right functions that I was looking for then. Generating the function was the part that took most of the time. I remember that I spent more than a month trying to figure out a way to make the wave have a random amplitude so the player wouldn't cheat in the game. The problem was finding a way to make the wave one sequence after generating with the new amplitude, as it would shift the new sequence by a specific amount of pixels on the x-axis, either positively or negatively, in relation to the last point in the old sequence.

Figure 6.1: the start of game repository on GitHub

Dealing with the other parts, such as, making the game follow the principle of encapsulation, took more than 3 days of continued work. Something I had before was the problem of having one file that did the functionality for all components of the game. For instance, the reset function that existed in ball functions is responsible for resetting the whole game, including wave-related variables. It is preferable to create a class for the wave and then include a function in it to reset the associated variables. You can call the wave functions whenever you want.

In my opinion, this is preferable, because I would be confused about which variables to call and would benefit from a better file management for each component of the game.

Figure 6.2: Time it took to implement the new layout in encapsulation

The repository can be accessed from this link *Survive Line*. The game, along with running footage, can be viewed on my website *Survive Line - ahmedmahfouz.me* for more visuality and the learning curve about it.

6.2   The AI tweaking

This part didn't require much tweaking, as it was just to leave the laptop to do the training sessions for the night and check the log in the morning. To compare between each generation, I had to add extra `print()` sentences in the log to keep track of them. It was also important to record the sessions with OBS, as I could navigate easily from the output code to the part in the video and then tweak it as much as I could for the next generation.

Bibliography

DaFluffyPotato. *Particles - Pygame Tutorial*. 2020. URL: https://www.youtube.com/watch?v=F69-t33e8tk.

Intellipaat. *Supervised vs Unsupervised vs Reinforcement Learning*. 2022. URL: https://intellipaat.com/blog/supervised-learning-vs-unsupervised-learning-vs-reinforcement-learning/.

Kenneth O. Stanley, Erin Hastings et al. *Galactic Arms Race: Research*. 2014. URL: http://galacticarmsrace.blogspot.com/p/research.html.

Knight, Forrest. *A.I. Learns to Play Sonic the Hedgehog - NEAT Explained*. 2020. URL: https://www.youtube.com/watch?v=5RR1T_-zVws&t.

Mahfouz, Ahmed. *Survive Line*. Version 1. Feb. 2023. URL: https://github.com/Ahelsamahy/Survive-Line.

— *Survive Line - ahmedmahfouz.me*. 2023. URL: http://ahmedmahfouz.me/?p=thesis.

Shinners, Pete. *PyGame*. 2011. URL: http://pygame.org/.

Smith, Allen W. *NEAT Overview - Genome*. 2020. URL: https://neat-python.readthedocs.io/en/latest/glossary.html#term-genome.

— *NEAT Overview - Genomic distance*. 2020. URL: https://neat-python.readthedocs.io/en/latest/glossary.html#term-genomic-distance.

— *NEAT Overview - Glossary*. 2020. URL: https://neat-python.readthedocs.io/en/latest/glossary.html.

Tim, Tech With. *Python Pong AI Tutorial - Using NEAT*. 2022. URL: https://youtu.be/2f6TmKm7yx0.

Wiki, AI. *Weights and Biases*. URL: https://machine-learning.paperspace.com/wiki/weights-and-biases.