

Bard

Bard College
Bard Digital Commons

Senior Projects Spring 2017


Bard Undergraduate Senior Projects

Spring 2017

Mouse vs. Machine: The Game

Cafferty Aiko Frattarelli
Bard College, cf4707@bard.edu

Follow this and additional works at: https://digitalcommons.bard.edu/senproj_s2017

 Part of the [Artificial Intelligence and Robotics Commons](#), [Digital Humanities Commons](#), and the [Graphics and Human Computer Interfaces Commons](#)



This work is licensed under a [Creative Commons Attribution-NonCommercial-Share Alike 4.0 License](#).

Recommended Citation

Frattarelli, Cafferty Aiko, "Mouse vs. Machine: The Game" (2017). *Senior Projects Spring 2017*. 150.
https://digitalcommons.bard.edu/senproj_s2017/150

This Open Access work is protected by copyright and/or related rights. It has been provided to you by Bard College's Stevenson Library with permission from the rights-holder(s). You are free to use this work in any way that is permitted by the copyright and related rights. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. For more information, please contact digitalcommons@bard.edu.

Bard

Mouse Versus Machine: the Game

Senior Project submitted to

The Division of Science, Mathematics, and Computing
of Bard College

by

Cafferty Aiko Frattarelli

Annandale-on-Hudson, New York

May 2017

Acknowledgements

I would like to thank my senior project advisor, Khondaker Salehin, for his help making this project a reality, and my academic advisor, Keith O'Hara for his help and encouragement throughout my school career.

Particular thanks to Benjamin Newman, who provided many useful tools and another set of eyes for my code, and William Flack, who let me use his dorm as a spare room to work in, and for the lunches we all spent together getting much needed relaxation.

And to my friends who I worked alongside to get our projects done, Nicolas Engst Matthews, who I must also thank for proofreading my writing, Hannah Livant, Haley Goss-Holmes, and Hayden Zahn. I am terrifically grateful for your company.

And to all my friends throughout college, making these years to remember.

And to my family, without whom I wouldn't be here at all.

Abstract

Many modern video games built by big name companies are coded by a group of people together using, and possibly modifying, an already designed game engine. These games usually have another group of people creating the artwork. In this project, I coded and designed a video game from scratch, as well as created all the artwork used in the game. The player controls a mouse character who fights a variety of monsters. In order to create the complexity of the game, I implement basic neural networks as the enemy artificial intelligence, i.e. the decision making process of the enemy. It uses this to learn how to combat a player from the player's actions, including movement and attacking. Movement is implemented through changing the player's position on the screen, and attacking creates an image which causes damage to other characters. The program is coded in Python, using the Pygame library for displaying graphics. It is currently an alpha version, with the code built and all the gameplay elements in place. With the existing foundation, this game, "Mouse versus Machine", can be extended into a full-fledged game in the future.

Table of Contents

Acknowledgements	i
Abstract	iii
Table of Contents	v
1 - Introduction	1
2 - An Overview of the Code	3
2.1 - Sprite class	4
2.2 - Character class	6
2.3 - Enemy class	8
2.4 - Inanimate class	10
2.5 - Attack class	10
2.6 - Door class	11
2.7 - Words class	12
2.8 - Main function	12
3 - Neural Networks	15
3.1 - The Basics	15
3.2 - The Code	16
3.2.1 - The Variables	17
3.2.2 - The Functions	18
4 - Art Design and Progression	21
5 - The Process	27
5.1 - The Coding	27
5.2 - The Bugs (and other technical difficulties)	29
6 - Conclusion	31
7 - Appendix	33
7.1 - Main Code	33
7.2 - Test Functions	51
7.3 - Survey	54
8 - Bibliography	55

1

Introduction

This project started with the concept of building a game from scratch. The goal was to learn about the building blocks of video games and how those can be expanded upon, as well as using these building blocks as a venue for a decision-making artificial intelligence (A.I.). The process involved research into game design and A.I. development. I wanted to use minimal outside libraries aside from one for graphics to display the game. I also wanted to incorporate different aspects of my college experience, including my computer science experiences, particularly in Object Oriented Programming, and Intelligence and Perception in Robotics, as well as my art based experiences - mainly Cybergraphics - though with a basis in the various other art classes I have taken, as well as my job on campus as a poster designer.

Game design is a complex subject involving many facets of computer programming, as well as art and design. Since I am only one person, I could not hope to rival the works of major video game companies and various designers using video game production as their sole form of

employment. This limitation caused me to focus on creating the building blocks of the game and making my own program that I could work with and continue building into a completed game. This would also allow me to have a strong understanding of what was and was not possible with the game, and why. The goal was to have complete control over the game design and structure, as well as working to optimize the game processes for minimum lag overall.

The game design was influenced by many sources, including my own experience with video games, reading various articles about video games, and discussing video game design and game mechanics with my peers. I ended up going with a relatively common overall structure, with a top-down viewpoint of a dungeon where the player controls one character and progresses through a series of rooms, fighting different enemies along the way. Some games that are comparable to this are the old Legend of Zelda games for the GameBoy, and the game Binding of Isaac, a more recent game for the PC. These are both variations of top-down two-dimensional games. Two-dimensional games are generally either top-down or side-scrolling. Viewpoints, and graphics in general, get more complicated and computer intensive once it gets to three-dimensional representations, so I kept this game two-dimensional.

The main goals behind using a neural network for the A.I. in this game was to have an amount of unpredictability and reactivity in the enemy characters' actions. The goal was not to have a perfect A.I., because that does not lead to particularly entertaining gameplay. Since this neural network starts with random weights and gathers data as the game progresses, it fulfills both of the criteria of being unpredictable and reactive.

2

An Overview of the Code

This game was coded in Python 2.7.11, implementing the Pygame and NumPy modules. Pygame is a Python library used for building video games. In this project it was used for its graphics and game time functions. The NumPy module was used for all the mathematical functions that weren't basic arithmetic, including basic trigonometric math to calculate in game distances, as well as array multiplication for the neural network, and random number generation for both the neural network and determining how the enemy characters walk when they are not aimed towards a specific location in the game.

Classes

Since this game was programmed using object-oriented programming, it is made up of several class objects, which are listed and described below, along with their initial variables and

functions; not including the basic functions that only set different variables. The code is listed in the appendix for reference. Notice the NeuralNetworkAI class is not assessed here. This is because it is more complex and will be covered thoroughly in the next section.

2.1 The Sprite class

This class is the basic class used to display any sort of image that may need to be interacted with or moved. It is the parent class of every class in the program other than the NeuralNetworkAI, since every other class represents some in-game object..

Variables:

“position”	a tuple holding the x and y coordinates of the Sprite
“size”	a tuple holding the width and height of the Sprite
“image”	loads and stores the image for the Sprite
“sprite”	stores the Sprite’s display image, which can be changed, for when multiple images are possible
“interactable”	a Boolean that says whether a Sprite can be interacted with (i.e. there is a response if the player presses the interact key while their character is looking at the Sprite)
“text”	a tuple of strings that display when the object is interacted with
“substantial”	Boolean that determines whether an object can be walked through or not
“flammable”	Boolean that determines if an object is hurt by the player’s attack
“enemy”	Boolean that determines if the object is an enemy character

“direction”	an integer value between -3 and 4 that determines which direction the Sprite is facing
-------------	--

The Main Functions:

<i>Place</i>	takes a position and a surface and uses Pygame’s blit function to display the object on the surface
<i>DoesOverlap</i>	checks if the current object overlaps with another object, provide to the function, by comparing the two objects’ position and size, used in various collision detections
<i>OutOfBounds</i>	checks if object goes off the screen
<i>CanInteract</i>	takes another object and sees if the current object (usually the player character) can interact with the other object, taking into account if the other object is “interactable” and the distance between the current object and the other object. It tests this by seeing if the player’s sprite would overlap, using the <i>DoesOverlap</i> function, with the object if moved forward by half of the Sprite’s size.
<i>SetText</i>	sets the text to the given tuple of strings and sets the Sprite’s “interactable” variable to the given Boolean, “interact”, defaulting to True, since generally if the text is set, the object can be interacted with to at least provide a text response. Will be set to False if removing the text.
<i>Delta</i>	converts a given direction and distance into change in x and y, returned as a tuple

<i>DistanceTo</i>	finds the distance from approximately the edge of one object to the edge of the other, using the size of the object to find the center as well as finding the approximate distance from the center to the edge
<i>SetDirection</i>	sets “direction” to the given direction. If given an integer outside of integer values between -3 and 4, the function converts it to inside the range. <i>SetDirection</i> also changes the sprite displayed to reflect the set direction.

2.2 The Character class

The Character class is a child class of the Sprite class. It is the base class for all characters, mainly the player character and the enemy monsters.

Variables:

“attributes”	a list with four elements, corresponding to the Character’s health, attack, defense, and speed, in that order. Health determines how many hits a Character can take before it dies. Attack determines how often a Character can attack. Defense gives the odds of an attack hitting the Character. Speed determines how far a Character can go each game tick.
“canAttack”	if “canAttack” is zero, the Character can attack, otherwise “canAttack” is an integer above zero that is manipulated by the <i>IncrementAttack</i> function to reach zero in a time frame based on the Character’s attack attribute.
“level”	an integer that determines the difficulty of the enemies the Character, applicable to the player character.

“stepNum”	an integer that keeps track of the number of ticks the Character has walked, so the image can change depending on the number of steps and looks like it’s walking.
“step1”	stores the loaded and scaled image for the first part of the walking animation.
“step2”	stores the loaded and scaled image for the second part of the walking animation.
“attack”	creates and stores an Attack object with a default attack image.

The Main Functions:

<i>GetHealth, GetAttack, GetDefense, and GetSpeed</i>	returns the value of each respective attributes from the “attributes” list
<i>AdjustAttribute</i>	takes a string “Health”, “Attack”, “Defense”, or “Speed” and adds the given change value to the attribute to adjust it. The change value can be negative.
<i>CanAttack</i>	returns True if “canAttack” is zero, and sets “canAttack” to a higher integer based on the Character’s attack attribute, otherwise it returns False.
<i>IncrementAttack</i>	decrements the “canAttack” value by one each game tick until it reaches zero.
<i>Walk</i>	moves Character in the given direction based on the speed of the Character by calling <i>IncrementalWalk</i> , as well as updating the displayed image for the character based on “stepNum”. Returns the value from <i>IncrementalWalk</i> .

<i>IncrementalWalk</i>	tries to move the Character in a given direction and distance by checking if the Character can be placed that distance and direction from where it starts using <i>DoesOverlap</i> and <i>OutOfBounds</i> . If it can be placed there, it is placed there, otherwise it calls <i>IncrementalWalk</i> again with a lower distance, until the distance reaches zero, in which case it returns False. If a successful move is accomplished it returns True.
<i>Attack</i>	displays the Character's attack sprite when necessary and if the Character can attack, it checks if the attack sprite overlaps with any object that can be damaged by the attack, and if so it decreases the health of said object, and if that object's health reaches zero it removes the object from the objects array given. Also changes the success value of the NeuralNetworkAI.

2.3 The Enemy Class

The Enemy class is a child of the Character class, and by extension the Sprite class. It holds the variables for several different Enemy monsters with the potential to add more quite easily. It also has the functions that allow the Enemy to use the programmed A.I. to make decisions and control its actions.

Variables:

“type”	a string variable that determines what kind of monster the Enemy is. Currently the options are “spider”, “lizard”, and “other”. Each has a different image and different attributes.
“move”	stores the move decided by the neural network A.I., an array of two integers, zero or one.
“wander”	this variable is an integer used to count how long the Enemy has been using the <i>Wander</i> function and not changed direction.
“status”	keeps track of whether the Enemy can make it’s next move yet. Used to give a time where it has to keep using a single move so it cannot react immediately and gives the player Character a chance to act.
“action”	stores the current action the Enemy is performing or has just performed.
“lastAction”	saves the last action the Enemy performed once the current action changes.
“previousHealth”	saves the health the Enemy had last game tick.
“usingAI”	determines if the Enemy is using the neural network A.I. or the simple state-based A.I. It’s True if using the neural network and False otherwise.

The Main Functions:

<i>Act</i>	calls <i>ChooseAction</i> and executes the provided “action” by name.
<i>ChooseAction</i>	uses different procedures depending if the Enemy is using the neural network A.I. or not. If it is, <i>ChooseAction</i> creates a new data point and evaluates it with the neural network and returns the result as the action to perform. Otherwise, <i>ChooseAction</i> uses the Enemy’s distance from the player and

	direction towards the player to determine its next action.
<i>Combat</i>	the “action” called from <i>Act</i> that calls <i>Attack</i> , defined in the Character class.
<i>Flee</i>	the “action” called from <i>Act</i> that moves the Enemy the opposite direction from the player.
<i>Turn</i>	the “action” called from <i>Act</i> that moves the Enemy towards the player, and attempts to go around obstacles.
<i>Wander</i>	the “action” called from <i>Act</i> that has the Enemy <i>Walk</i> in a random direction.

2.4 The Inanimate Class

The Inanimate class is a child of the Sprite class. It is used to display inanimate objects such as walls and rocks. It also suggests the structure for objects that can be broken.

Variables:

“sturdy”	how much the Inanimate can take before being “broken”
“broken”	if the Inanimate can be used

2.5 The Attack Class

The Attack class is a child of the Sprite class. It is used to display attacks.

Variables:

“hold”	determines if the Attack image displays even if the Attack is not doing damage
“image1” and	the image displayed for the Attack and the flipped image for animating the

“image2”	Attack
“state”	keeps track of how long part of the Attack animation has been showing so it can switch back and forth to animate it.

The Main Functions:

<i>Show</i>	displays and animates the Attack image
-------------	--

2.6 The Door Class

The Door class is a child of the Sprite class. If the player interacts with the Door, they will be moved from one room to another, unless the Door is “locked.” The Door class defaults to “interactable” being True.

Variables:

“nextRoom”	the room, a collection of objects, to which the Door opens
“locked”	determines whether the Door can be opened or not. If it is “locked” it cannot be opened until otherwise determined. Defaults to being not “locked”. The Door’s “text” is determined by “locked” as well.
“opensTo”	the location the Door opens to, i.e. where the player is placed if they interact with the Door and it is not “locked”.

The Main Functions:

<i>Unlock</i>	sets “locked” to False
<i>Open</i>	returns “nextRoom” and “opensTo” to be used to move the player Character to the intended location through the Door

2.7 The Words Class

The Words class is a child of the Sprite class. It is used to display all text in the game.

Variables:

“size”	overwrites the Sprite “size” variable to one to be used for font size
“text”	stores the text to be displayed in a tuple of various lengths
“font”	loads the font to be used for the text

The Main Functions:

<i>Place</i>	overwrites Sprites’ <i>Place</i> function to render each line of the Word’s “text” in its “position”, iterating through the “text” to put each line in the Word below the previous line. Also slightly lightens the Word’s color with each line, so large chunks of text get lighter towards the bottom.
<i>SetText</i>	used to set the Word’s “text” variable so if a chunk of text is added to the Word it goes to the top of the stack and excess lines are removed from the bottom. Mainly used for the text display from interacting with objects in the game.

2.8 The Main Function

The main() function is the function outside of the classes that defines the game loop and variables carried over from one game session to another. It brings together all the different classes and functions defined above and runs the game.

It starts by initiating all the constants and objects with the desired parameters, including the game clock, window for displaying images on, the player, monsters, and a variety of inanimate objects. These objects are placed in the different room lists, which can be swapped between being used in the current OBJECTS list, which determines which objects are being displayed.

Then there is the game loop, which is a while loop that continue indefinitely until the player exits the game. The game acts different if the player Character is alive or dead. If the player is dead, meaning their health has reached zero, the game will display the death screen and give the player the option to restart the game. If the player is not dead, the game loop continues normally, logging the player's keystrokes. The "w", "a", "s", and "d" keys correspond to moving the player Character up, left, down, and right respectively, while the left shift key has the player Character try to interact with an object, and the space bar causes the player Character to attack. After these are checked, the objects in the current room are assessed to see if they should do anything, such as having enemies move and attack. Then the object images are redrawn in the window. Finally, health and other statuses of the player Character are updated and one game tick passes.

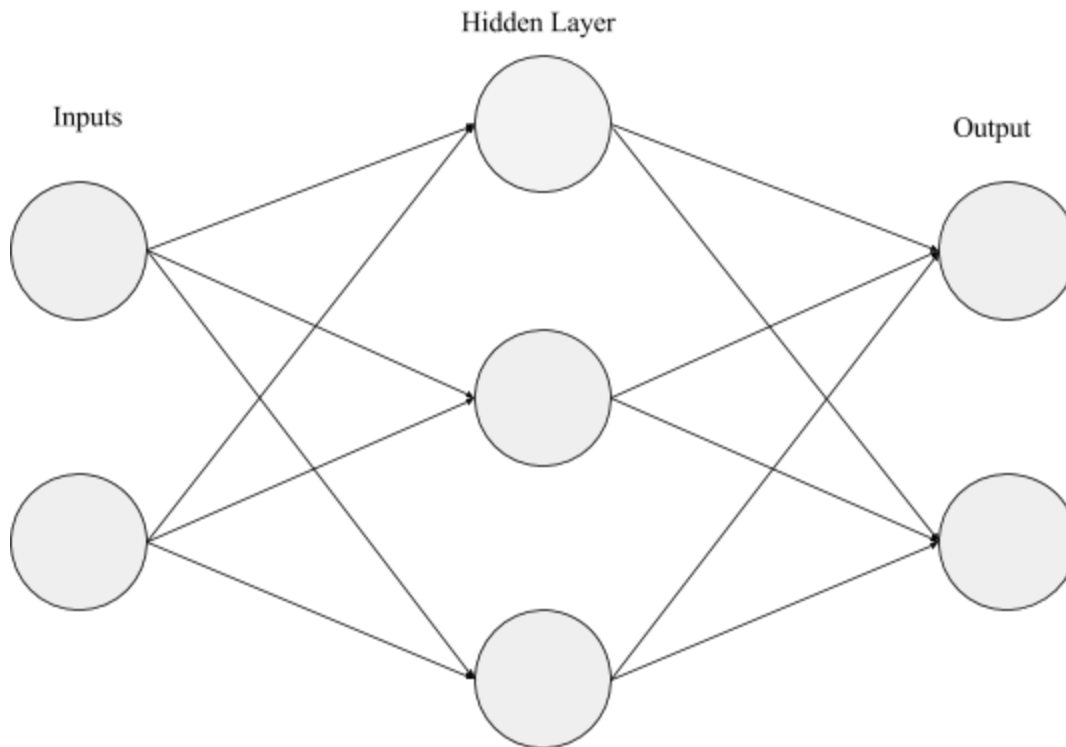
Below the main function several constants are defined, including the display size, and the input and output arrays for initializing the neural network A.I., and the main function is called.

3

Neural Networks

3.1 The Basics

Neural networks are based on neurons and the networks they create. In the case of computers, these neurons are an approximation: computational structures that take inputs and return activation values. Each neuron has a weight, learned through a training dataset, that is applied to the input, then run through a nonlinear function, such as sigmoid or hyperbolic tangent to find the activation value. When these neurons are connected together to process the data through hidden layers they can process a large variety of complex data sets.



Here is an example of a basic neural network with two inputs, one hidden layer with three neurons, and two outputs.

3.2 The Code

The neural network in this game is implemented through the class *NeuralNetworkAI*, which collects and analyzes data from every Enemy when activated. Data collection is accomplished by retrieving the values already stored in the Enemy class, or through calling a function in the Enemy class. Analyzing the data is accomplished through forward propagation and backpropagation. Forward propagation means to multiply the input through the network to get an output. Backpropagation involves finding the amount of error between the predicted result

from forward propagation and the given result and using that to update the weights for more accurate prediction.

3.2.1 The Variables

“totalData”	the total number of sample data there is to use, calculated by the length of the given “initialInputData”.
“dataLength”	the number of inputs per each piece of data, plus one to accommodate the bias, calculated using the length of one of the inputs
“weights1, 2, and 3”	randomly generated sets of weights for each layer of the neural network, to be changed to reflect the data through backpropagation
“inputData”	previously generated input data to be analyzed
“outputData”	previously generated output data corresponding to the input data to be analyzed
“unclaimedData”	a variable to store input data that is being collected but has yet to be analyzed
“unclaimedResults”	a variable to store output data that is being collected but has yet to be analyzed, corresponds to “unclaimedData”
“currentSuccessValue”	stores the current success value of the “unclaimedData” and “unclaimedResults”, a calculation of the Enemy’s action’s success
“learningRate”	the rate at which each update to weights affects the weights, useful for keeping the weights from converging prematurely

3.2.2 The Functions

-trainNeuralNetwork

This function first adds an extra 1 to the end of each array of input data to be used for the bias. Then it updates the weights the given number of times using backpropagation. This neural networks has two hidden layers.

-newDataPoint

This function collects the current direction, distance and direction to the player, edited to be a value between 0 and 1, from one Enemy and returns it to whichever function was looking for that information.

-collectData

This function collects a data point from *newDataPoint* and places it into “unclaimedData”, as well as collecting the current action from the Enemy and storing it in “unclaimedResults”. It also takes a variable “Refresh”, which defaults to False, that determines if *collectData* will attempt to see if the “currentSuccessValue” is high enough to add the unclaimed data and results to “inputData”. When adding the new data to “inputData”, it must remain the same length in order for the matrix multiplication works, so the function shuffles the new data and the old data together and takes a random sample of the two so there are the same number of

inputs in the updated array. If it does add new data, it calls the *trainNeuralNetwork* function again, with a slightly higher learning rate.

-extrapolate

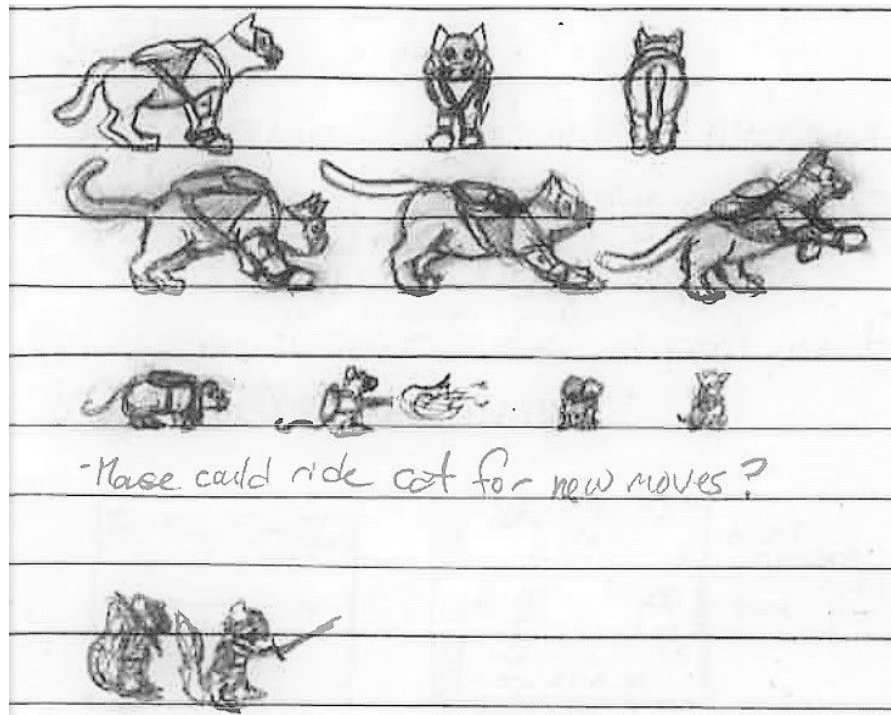
This function takes the current state of the Enemy and uses forward propagation with the current weights to choose the next action the Enemy takes.

-sigmoid

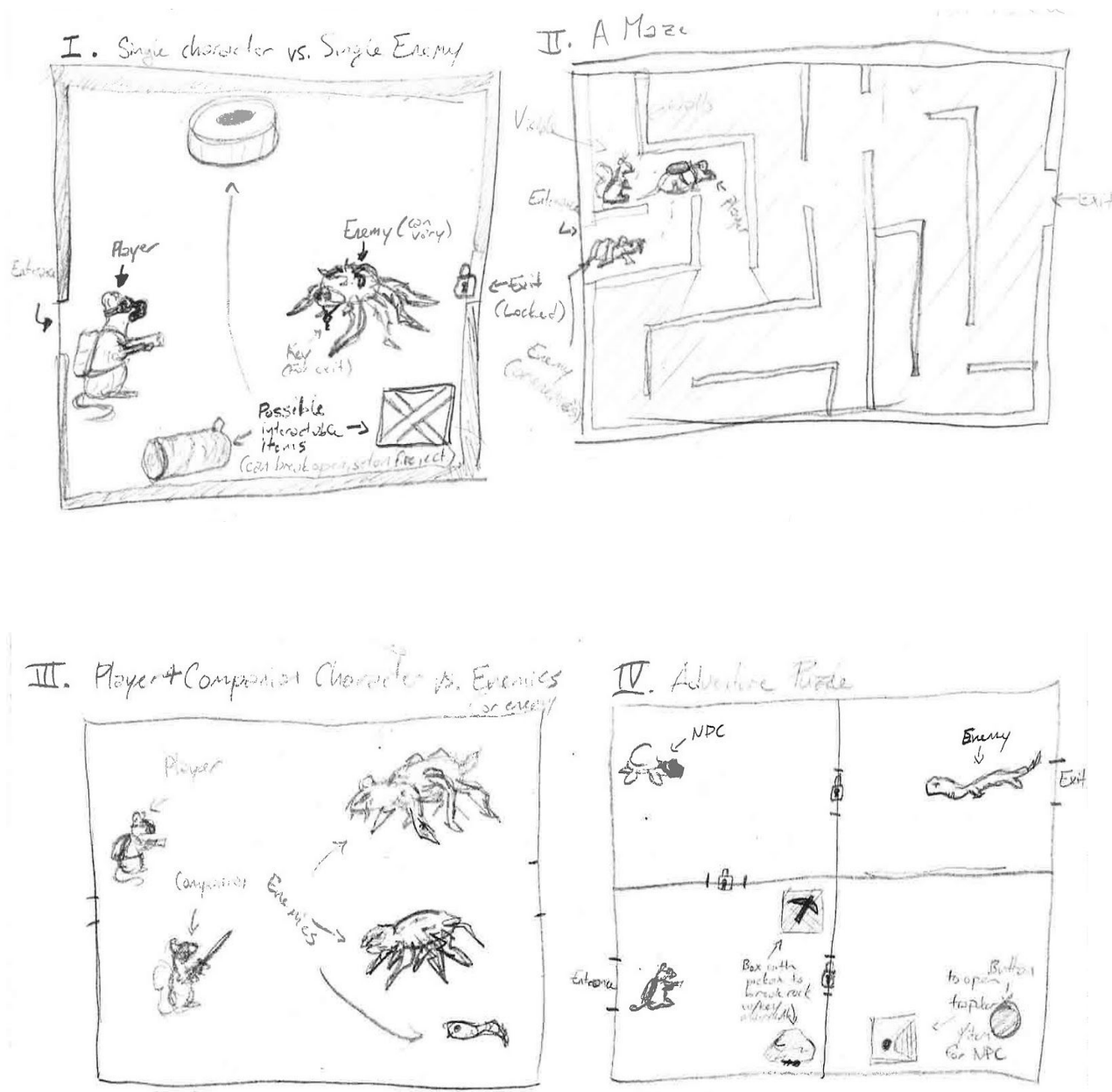
This function converts the outputs from each layer nonlinearly to a number between 0 and 1, as well as being able to be used to find the derivative. The derivative is used to find how certain the predicted value was. If the derivative, and thus the slope, is high, then it is not a very certain prediction, so the change in weight will be more drastic when the weights are updated.

4 Art Design and Progression

The art of the game and how it progressed through the game design



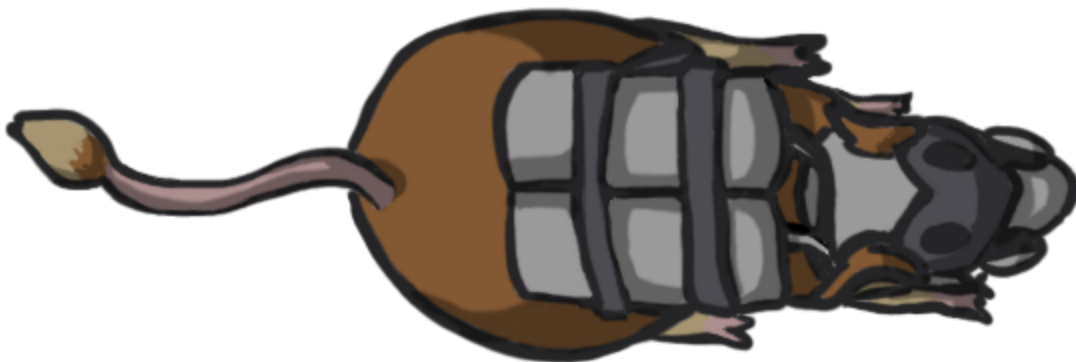
First sketches of possible characters for the player character. The mouse was eventually chosen to keep the scale reasonable and the complexity down.



Sketches done of possible game scenarios and enemy designs. Several of these can be implemented using the data structures and objects created.



Original digital main character mouse design, scrapped due to decision to have the game view from the top down.



Second mouse design, scrapped due to the logistical difficulties caused by a rectangular sprite design. Mainly the problems occurred in corners and near objects where it would either clip through them or not fit in places where it fit before when it was sideways.



Final mouse design, showing differences in walking animation.



The sprite for the spider and the lizard-like enemy characters.



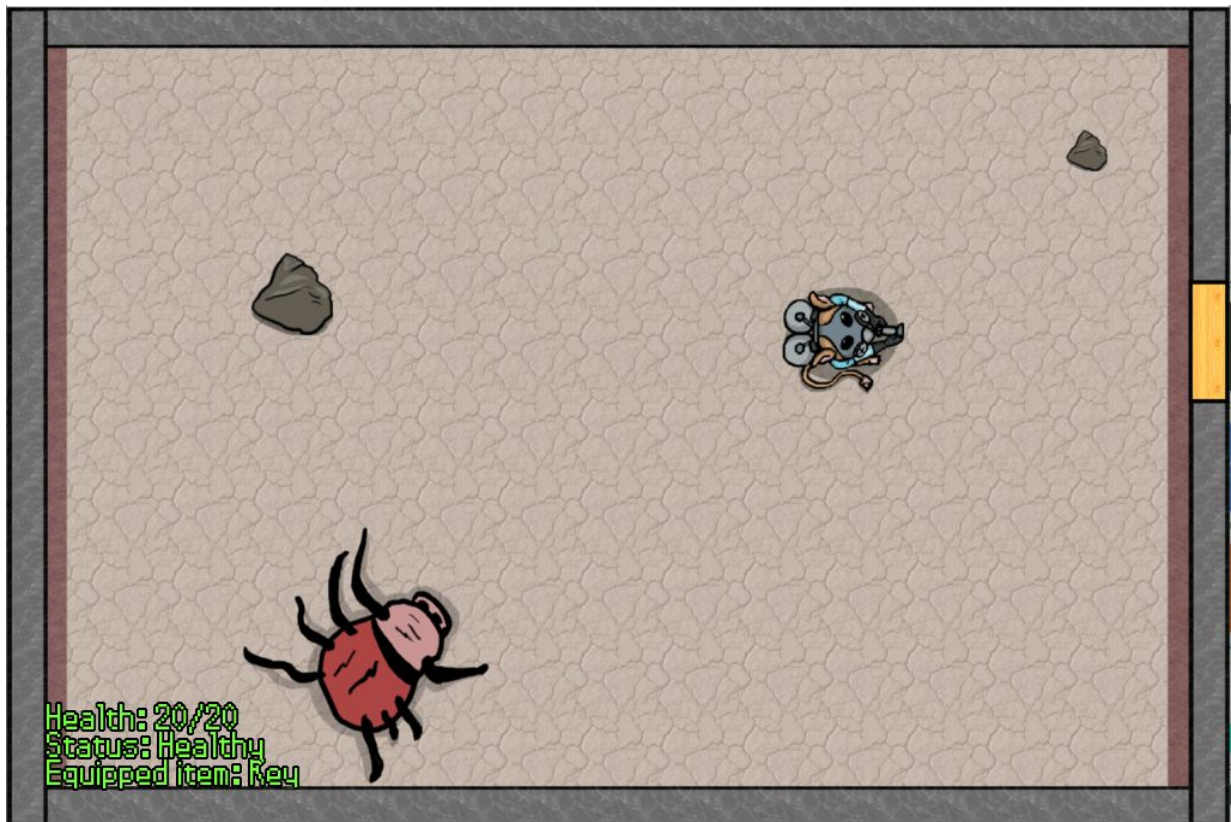
The sprite for the rock object, also used for the morph enemy as an enemy to catch the player off-guard.



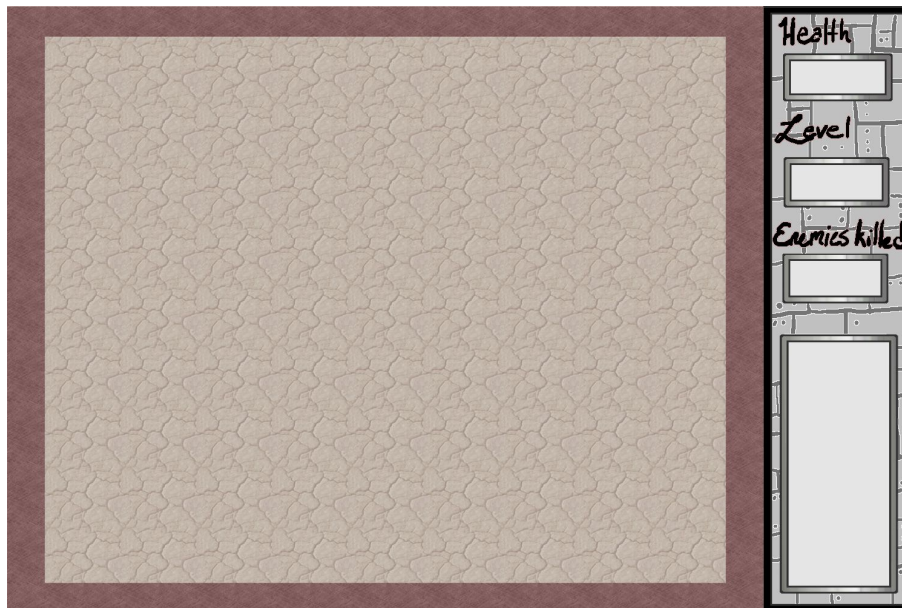
The attack sprites used by the player and the enemy characters respectively.



The wall and door sprites. They define the dimensions of the room and show the way out.



Initial mock-up of gameplay.



The final background image, with spaces for displaying health, level, and enemies killed, as well as any in-game description or dialogue.



An example of a game scene where the player character is being attacked by a morph.

5

The Process

5.1 The Coding

Coding this game from the ground up, I started by figuring out which programming language to use. I settled on Python because it was the language I was most familiar with, particularly in regards to object oriented programming and in relation to artificial intelligence similar to what was covered in the Intelligence and Perception in Robotics class. I decided to use the Pygame module to supplement Python due to its graphics and timing library.

Next I started coding the rough building blocks of the game, fleshing out which classes were and were not necessary. In this step there was a lot of writing and re-writing code for efficiency, so the code did not work unnecessarily hard and cause the game to slow down. This

would cause frustration both on the player's level and further down the line cause difficulties with training the A.I., which would only serve to slow the program further.

Once the code was to a point where I could introduce graphics, I began to digitally draw out the characters using my drawing tablet and the art program GIMP 2. Some of them had already been designed on paper, as seen in the previous section. After several redesigns I found what worked best for the game and drew up final versions.

Finally, I started implementing the A.I. At first the goal was to have the A.I. fully control the Enemy characters, that, however, turned out not to be feasible for a number of reasons. First, it slowed down the program a significant amount due to the number of relevant inputs, as well as the number of test problems that would be necessary to give the A.I. accurate instruction. Second, the problem with giving the A.I. full control of the Enemy characters was that the computer has much quicker reactions than a human player does, so if it did work fully, it would be incredibly difficult, if not impossible, to play the game and win, or even have fun. Thus, I decided on a partially hybridized A.I., where the neural network chooses between a few different actions with a time delay.

Implementing the neural network involved a variety of different tests, including ones to generate a training data set, as well as finding the right balance of number of iterations and learning rate for the neural network to produce accurate results.

5.2 The Bugs (and other technical difficulties)

There are always some bugs and other difficulties in any coding project, however these were some that were particularly difficult or stood out in some way, and some which still persist.

The Stuck Mouse

This was the bug where the player Character, the mouse, could not move because it was seeing itself as an obstacle to where it could walk. This was a result of a previous bug fix where the Enemy characters could walk through the player Character due to not perceiving it as an obstacle. This was eventually solved by reformatting the *Walk* function so it could check whether the object it was trying to walk through was itself.

There was also a similar bug where, if the player Character was facing the wrong direction, its attack could hit itself and burn it to death, which was also solved by making it check whether the object it was attack was itself.

The Wall Approach Problem

This was a problem where any of the Characters did not always fully approach the wall while walking due to the Character only being able to move its full distance or not at all. This was solved by creating the function *IncrementalWalk*, which calls itself recursively until it's certain there's no space to move into between the Character and the other object.

Beta testing

It was a goal to have some people beta test this game and fill out a survey based on their experiences. However, sending the game to others proved challenging in the time left, and when I did send it to some people. They proved to be unable to run the game on their computers due to various factors, such as what software they had on their computer, and what kind of computer they had. For example, one person had one version of Python downloaded that was not compatible with my version of Python, Also, Mac computers were particularly troublesome because of their differing file system. Thus I was unable to have the game beta tested as of this report. Some more research into creating an executable file of the game will likely make it possible in the future.

Diagonal Rotation

One problem that still remains in the code is the problem where rotating an image diagonally makes said image larger, and thus slightly changes how the image interacts with obstacles, sometimes causing the images to overlap. It is not a major issue, however it is still an issue I will continue to investigate. It wouldn't be as much of a problem if Pygame had an image cropping function, however it does not so it requires a bit of a work around, and will most likely involve rewriting how the *Place* function works.

6

Conclusion

This game is the alpha version of the game, which I intend to complete down the line by adding more assets and features. Some features I will implement include other friendly characters that the player can interact with, as well as more varied objects and new monsters to fight in new rooms. Once these features are implemented I can move onto expanding the game into a full story, possibly with sound and dialogue.

On the more technical side of things, I intend to make building objects and storing the training array for the neural network less cumbersome by reading it from a file instead of having all the values directly in the code. I will also look into an improved training set for the neural network, with a more sophisticated use of the success value that was implemented in this version

- possibly storing the success value of each input. This will involve some research into effectively calculating success in the game scenario. Possible contributing factors I have considered are taking less damage from the player, dealing more damage to the player, how long the Enemy character survives, and a ratio of the damage done over time versus the damage taken. The most complex aspects of figuring this out will be balancing the Enemy's survival with defeating the player character, as well as how to calculate these values and over what amount of time.

Overall, however, this project was successful in what it set out to do, namely, creating an program where the basics of the game were implemented and the actions of the Enemy characters depended on the neural network, which I coded. I have come out of this project with a much more solid understanding of how neural networks work, as well as a product that I am proud of . Despite some errors and frustrations in the process, at the end the work I did over the past year it came together into a successful program.

7

Appendices

7.1 Main Code

```

1. #Cafferty Frattarelli
2. #Mouse Vs. Machine: the Game
3. #Senior Project - May 2017
4.
5. import pygame, sys
6. from pygame.locals import *
7. import numpy
8.
9. #Sprite class - base class for all game objects
10. class Sprite:
11.     def __init__(self, sprite, size = (100, 100)):
12.         self.position = (0,0)
13.         self.size = size
14.         self.image = pygame.image.load(sprite)
15.         self.image = pygame.transform.scale(self.image, (self.size[0],
self.size[1]))
16.         self.sprite = self.image
17.         self.interactable = False
18.         self.text = " "
19.         self.substantial = True
20.         self.flammable = False
21.         self.enemy = False
22.         self.direction = -2
23.         #Up=0, UpRight=-1, Right=-2, DownRight=-3, UpLeft=1, Left=2,
DownLeft=3, Down=4
24.
25.     #Displays the sprite in assigned position (takes position tuple (x,y) and
surface to put it on)
26.     def Place(self, position, surface):
27.         surface.blit(self.sprite, position)
28.         if self.GetPos() != position:
29.             self.SetPos(position)
30.
31.     #Returns current position
32.     def GetPos(self):
33.         return self.position
34.
35.     #Sets current position (takes a tuple with two values, x and y)
36.     def SetPos(self, position):
37.         self.position = position
38.
39.     #Returns Sprite size (returns a tuple with two values, x and y)
40.     def GetSize(self):
41.         return self.size
42.

```

```

43.     #Sets Sprite size (takes a tuple with two values, x and y)
44.     def SetSize(self, size):
45.         self.size = size
46.         self.sprite = pygame.transform.scale(self.image, (self.size[0],
self.size[1]))
47.
48.     #Returns if something can be walked through or not (returns True or False)
49.     def GetSubstantial(self):
50.         return self.substantial
51.
52.     #Sets if something can be walked through or not (takes True or False)
53.     def SetSubstantial(self, sub):
54.         self.substantial = sub
55.
56.     #Sees if current object overlaps other object, used for collision detection
(takes and Sprite object, returns True or False)
57.     def DoesOverlap(self, other):
58.         if other.GetSubstantial():
59.             selfx = self.GetPos()[0]
60.             selfx2 = selfx + self.GetSize()[0]
61.             selfy = self.GetPos()[1]
62.             selfy2 = selfy + self.GetSize()[1]
63.
64.             otherx = other.GetPos()[0]
65.             otherx2 = otherx + other.GetSize()[0]
66.             othery = other.GetPos()[1]
67.             othery2 = othery + other.GetSize()[1]
68.
69.             if selfx >= otherx2 or selfy >= othery2 or selfx2 <= otherx or selfy2
<= othery:
70.                 return False
71.             else:
72.                 return True
73.         else:
74.             return False
75.
76.     #Used to make sure we haven't placed an object off screen (returns True or
False)
77.     def OutOfBounds(self):
78.         selfx = self.GetPos()[0]
79.         selfx2 = selfx + self.GetSize()[0]
80.         selfy = self.GetPos()[1]
81.         selfy2 = selfy + self.GetSize()[1]
82.         if selfx < 0 or selfy < 0 or selfx2 > DISPLAY_X or selfy2 > DISPLAY_Y:
83.             return True
84.         else:
85.             return False
86.
87.     #Checks if something is interactable (returns True or False)
88.     def Interactable(self):
89.         return self.interactable
90.
91.     #Sets if something is interactable (takes True or False)
92.     def SetInteractable(self, interact):
93.         self.interactable = interact
94.
95.     #Checks if this object can interact with supplied object,
96.     #given it can interact with something up to it's own size in front of it
97.     #(takes another Sprite object, returns True or False)

```

```

98.     def CanInteract(self,other):
99.         direction = self.GetDirection()
100.        delta = self.Delta(direction,self.GetSize()[0],self.GetSize()[1])
101.        tempPos = self.GetPos()
102.        self.SetPos((tempPos[0]+delta[0],tempPos[1]+delta[1]))
103.        result = self.DoesOverlap(other)
104.        self.SetPos(tempPos)
105.        if other.Interactable() and self != other:
106.            return result
107.        else:
108.            return False
109.        #Sets text that displays when this object is interacted with, and sets the
        object as interactable
110.        #(takes a tuple of strings)
111.        def SetText(self, text, interact = True):
112.            self.text = text
113.            self.interactable = interact
114.
115.        #Returns interaction text (returns a tuple of strings)
116.        def GetText(self):
117.            return self.text
118.
119.        #Converts direction to change in x and y (returns a tuple with two values,
        x and y)
120.        def Delta(self,direction,distance1,distance2=-1):
121.            if distance2 == -1:
122.                distance2 = distance1
123.            delta = (distance1*numpy.cos(numpy.pi*direction/4),
        distance2*numpy.sin(numpy.pi*direction/4))
124.            return delta
125.
126.        #Finds distance to another object from center minus half of size
127.        def DistanceTo(self, other):
128.            here = (self.GetPos()[0]+self.GetSize()[0]/2,
        self.GetPos()[1]+self.GetSize()[1]/2)
129.            there = (other.GetPos()[0]+other.GetSize()[0]/2,
        other.GetPos()[1]+other.GetSize()[1]/2)
130.            return
        numpy.sqrt((there[0]-here[0])**2+(there[1]-here[1])**2)-(self.size[0]+self.size[1]+
        other.size[0]+other.size[1])/4
131.
132.        #Returns direction from this to other object (takes a Sprite object and
        returns direction as defined in Sprite.__init__() )
133.        def DirectionTo(self,other):
134.            xDiff = self.GetPos()[0]-other.GetPos()[0]
135.            yDiff = self.GetPos()[1]-other.GetPos()[1]
136.            direction = round(4 + numpy.arctan2(yDiff,xDiff)/numpy.pi * 4)
137.            while direction > 4:
138.                direction = direction - 8
139.            while direction < -3:
140.                direction = direction + 8
141.            return direction
142.
143.        #Returns direction (an integer)
144.        def GetDirection(self):
145.            return self.direction
146.
147.        #Sets direction to an integer between -3 and 4 so it's useable by other
        functions (takes an integer)

```

```

148.     def SetDirection(self,direction):
149.         if direction >= -3 and direction <= 4:
150.             self.direction = direction
151.             self.sprite = pygame.transform.rotate(self.image,
152. 45*(-direction-2))
152.         elif direction > 4:
153.             self.SetDirection(direction-8)
154.         elif direction < -3:
155.             self.SetDirection(direction+8)
156.
157.     #Character class - base class for all characters: monsters, player, ect.
158.     class Character(Sprite):
159.         def __init__(self, img1, img2, size=(100,100)):
160.             Sprite.__init__(self, img1,size)
161.             self.attributes = [20,10,10,15]
162.             # Health, Attack, Defense, Speed
163.             self.flammable = True
164.             self.canAttack = 0
165.             self.level = 1
166.             self.stepNum = 0
167.             self.step1 = pygame.image.load(img1)
168.             self.step1 = pygame.transform.scale(self.step1, (self.size[0],
self.size[1]))
169.             self.step2 = pygame.image.load(img2)
170.             self.step2 = pygame.transform.scale(self.step2, (self.size[0],
self.size[1]))
171.             self.attack = Attack("../Art/Fire.png", True, (50,50))
172.
173.             #returns character's current health (returns an integer)
174.             def GetHealth(self):
175.                 return self.attributes[0]
176.
177.             #returns character's current attack stat (returns an integer)
178.             def GetAttack(self):
179.                 return self.attributes[1]
180.
181.             #returns character's current defense stat (returns an integer)
182.             def GetDefense(self):
183.                 return self.attributes[2]
184.
185.             #returns character's current speed stat (returns an integer)
186.             def GetSpeed(self):
187.                 return self.attributes[3]
188.
189.             #takes a string with the attribute name and adds the change to the stat
(change can be negative)
190.             def AdjustAttribute(self, att, change):
191.                 if (att == "Health"):
192.                     self.attributes[0] = self.attributes[0] + change
193.                 elif (att == "Attack"):
194.                     self.attributes[1] = self.attributes[1] + change
195.                 elif (att == "Defense"):
196.                     self.attributes[2] = self.attributes[2] + change
197.                 elif (att == "Speed"):
198.                     self.attributes[3] = self.attributes[3] + change
199.                 else:
200.                     print("Error: No such attribute")
201.
202.             #returns boolean of True if the character can attack and False if it cannot

```

```

203.     def CanAttack(self):
204.         if self.canAttack == 0:
205.             self.canAttack = 13 - self.GetAttack()
206.             return True
207.         else:
208.             return False
209.
210.     #counts down to when the character can attack again, based on the attack
    stat
211.         #(would more accurately be called attack speed, but is not for reasons
    of clarity)
212.     def IncrementAttack(self):
213.         if self.canAttack > 0:
214.             self.canAttack = self.canAttack - 1
215.         elif self.canAttack < 0:
216.             self.canAttack = 0
217.
218.     #Moves character in given direction if possible, using IncrementalWalk
219.     def Walk(self, direction, objects):
220.         self.stepNum +=1
221.         speed = self.GetSpeed()
222.         if self.stepNum >= 2*speed/10:
223.             self.image = self.step1
224.             self.stepNum = 0
225.         elif self.stepNum >= speed/10:
226.             self.image = self.step2
227.
228.         return self.IncrementalWalk(direction, objects, speed)
229.
230.     #Moves character based on direction and speed, gets as close to an obstacle
    as possible, returns how far it went
231.     def IncrementalWalk(self, direction, objects, distance):
232.         self.SetDirection(direction)
233.         tempPos = self.GetPos()
234.         delta = self.Delta(direction, distance)
235.         self.SetPos((tempPos[0]+delta[0], tempPos[1]+delta[1]))
236.
237.         if distance <= 0:
238.             self.SetPos(tempPos)
239.             return False
240.         if self.OutOfBounds():
241.             self.SetPos(tempPos)
242.             return self.IncrementalWalk(direction, objects, distance-1)
243.         else:
244.             for i in objects:
245.                 if self != i:
246.                     if self.DoesOverlap(i):
247.                         self.SetPos(tempPos)
248.                         return self.IncrementalWalk(direction, objects,
    distance-1)
249.             return True
250.
251.     #Draws a sprite that damages any enemy it touches if the character can
    attack
252.     def Attack(self, objects, display, AI):
253.
254.         if self.attack.hold:
255.             self.attack.Show(self.GetPos(), self.GetSize(), self.GetDirection(), display)

```

```

256.
257.         if self.CanAttack():
258.             if not self.attack.hold:
259.
self.attack.Show(self.GetPos(),self.GetSize(),self.GetDirection(),display)
260.         for i in objects:
261.             if i.flammable:
262.                 if self.attack.DoesOverlap(i) and i != self:
263.                     i.AdjustAttribute("Health", -1)
264.                     i.sprite.fill((4,0,0), None, BLEND_RGBA_MULT)
265.                     if i.enemy:
266.                         AI.currentSuccessValue += - 1
267.                     else:
268.                         AI.currentSuccessValue += 2
269.
270.                 if i.GetHealth() == 0:
271.                     objects.remove(i)
272.
273.             return(("Something", "burned.", ""))
274.         return False
275.     else:
276.         return False
277.
278. #The class the holds each enemy character's stats and actions
279. class Enemy(Character):
280.     def __init__(self, type):
281.         self.type = type
282.         if self.type == "spider":
283.             img1 = "../Art/Spider.png"
284.             img2 = "../Art/Spider.png"
285.             Character.__init__(self, img1, img2)
286.             self.attributes = [10,10,10,18]
287.         elif self.type == "lizard":
288.             img1 = "../Art/AquaLizard.png"
289.             img2 = "../Art/AquaLizard.png"
290.             Character.__init__(self, img1, img2, (150,150))
291.             self.attributes = [13,10,13,13]
292.         else:
293.             img1 = "../Art/Rock1.png"
294.             img2 = "../Art/Rock1.png"
295.             Character.__init__(self, img1, img2)
296.             self.attributes = [20,5,15,5]
297.         self.attack = Attack("../Art/Slash.png", False, (50,50))
298.         self.move = 0
299.         self.enemy = True
300.         self.wander = 0
301.         self.status = 0
302.         self.action = ""
303.         self.lastAction = ""
304.         self.previousHealth = self.attributes[0]
305.         self.maxHealth = self.attributes[0]
306.
307.         #Whether this is using the neural network AI or using the basic AI
308.         self.usingAI = True
309.
310.     def Act(self, player, OBJECTS, AI, display):
311.         #choose action and perform it
312.         #Actions: attack, flee, turn towards and approach the player, and
wander

```

```

313.         self.lastAction = self.action
314.         self.action = self.ChooseAction(player, OBJECTS, AI)
315.         if self.action == "combat":
316.             self.Combat(player, OBJECTS, AI, display)
317.         elif self.action == "flee":
318.             self.Flee(player, OBJECTS)
319.         elif self.action == "turn":
320.             self.Turn(player, OBJECTS)
321.         else:
322.             self.Wander(OBJECTS)
323.
324.     def ChooseAction(self, player, OBJECTS, AI):
325.         #returns action to do
326.         if self.usingAI:
327.             if self.status == 0:
328.                 move = AI.extrapolate(AI.newDataPoint(self, player, OBJECTS))
329.                 if move[0] > .5:
330.                     self.status = 5
331.                     return "combat"
332.                 elif move[1] > .5:
333.                     self.status = 10
334.                     return "turn"
335.                 elif move[2] > .5:
336.                     self.status = 5
337.                     return "flee"
338.                 else:
339.                     self.status = 10
340.                     return ""
341.             else:
342.                 self.status=self.status-1
343.                 return self.action
344.
345.         else:
346.             distance = self.DistanceTo(player)
347.             if self.type == "rock":
348.                 if self.status == 0:
349.                     if distance >= 50 or
self.GetDirection() !=self.DirectionTo(player):
350.                         self.status = 10
351.                         return "turn"
352.                     else:
353.                         self.status = 10
354.                         return "combat"
355.                 else:
356.                     self.status += -1
357.                     return self.action
358.             elif self.status == 0:
359.                 if distance <= 500:
360.                     if self.previousHealth > self.GetHealth():
361.                         self.status = 5
362.                         self.previousHealth = self.GetHealth()
363.                         return "flee"
364.                     elif distance >=50 or
self.GetDirection() !=self.DirectionTo(player):
365.                         self.status = 5
366.                         return "turn"
367.                     else:
368.                         self.status = 10
369.                         return "combat"

```



```

370.         else:
371.             self.status = 10
372.             return ""
373.         else:
374.             self.status += -1
375.             return self.action
376.
377.     def Combat(self, player, OBJECTS, AI, display):
378.         #attack player
379.         self.Attack(OBJECTS, display, AI)
380.
381.     def Flee(self, player, OBJECTS):
382.         #move away from player
383.         self.Walk(self.DirectionTo(player)+4, OBJECTS)
384.
385.     def Turn(self, player, OBJECTS):
386.         #changes angle and walks /towards player
387.         if self.status == 0:
388.             self.SetDirection(self.DirectionTo(player))
389.         for i in range(7):
390.             if not self.Walk(self.GetDirection(),OBJECTS) and not
self.CanInteract(player):
391.                 if i%2 == 1:
392.                     self.Walk(self.GetDirection()-i,OBJECTS)
393.                 else:
394.                     self.Walk(self.GetDirection()+i,OBJECTS)
395.             else:
396.                 break
397.
398.     def Wander(self,OBJECTS):
399.         #move around randomly
400.         self.move+=1
401.         if self.move==10:
402.             self.move=0
403.         for i in OBJECTS:
404.             if self.CanInteract(i):
405.                 self.move = 0
406.                 break
407.         if self.move==0 or self.move==5:
408.             self.wander = numpy.random.randint(-3,3)
409.         if not self.Walk(self.wander,OBJECTS):
410.             self.move = -1
411.
412.     #Makes decisions for the enemy characters
413.     class NeuralNetworkAI():
414.         def __init__(self, initialInputData, initialOutputData, learningRate = .1):
415.             self.totalData = len(initialInputData)
416.             self.dataLength = len(initialInputData[0]) + 1
417.
418.             numpy.random.seed(1)
419.             self.weights1 =
2*numpy.random.random((self.dataLength,self.totalData)) - 1
420.             self.weights2 =
2*numpy.random.random((self.totalData,self.totalData)) - 1
421.             self.weights3 = 2*numpy.random.random((self.totalData,4)) - 1
422.             #takes an array of weights, initially randomly generated, then
calculated in the game process
423.             self.inputData = initialInputData
424.             #takes an array of input data, changed to values between 0 and 1

```

```

425.             #each array is length 3 in the form of:
426.             #[current_direction, player_distance, player_direction]
427.
428.             #How to calculate each input: current_direction =
429.             (self.GetDirection()+4)/8
430.             # player_distance =
431.             self.DistanceTo(player)/1500
432.             # player_direction =
433.             (self.DirectionTo(player)+4)/8
434.             self.outputData = initialOutputData
435.             #takes an array of output data for each input point, of values 0 to
436.             1
437.             self.unclaimedData = numpy.array([[5]])
438.             #will be used to store collected data with undetermined success
439.             value
440.             self.unclaimedResults = numpy.array([[5]])
441.             self.currentSuccessValue = 0
442.             self.learningRate = learningRate
443.
444.             def trainNeuralNetwork(self, times):
445.                 l0 = []
446.                 for i in range(self.totalData):
447.                     if i==0:
448.                         l0 = numpy.array([numpy.append(self.inputData[0], 1)])
449.                     else:
450.                         l0 = numpy.append(l0, [numpy.append(self.inputData[i], [1],
451. 0)], 0)
452.
453.                 for i in xrange(times):
454.
455.                     l1 = self.sigmoid(numpy.dot(l0, self.weights1))
456.                     l2 = self.sigmoid(numpy.dot(l1, self.weights2))
457.                     l3 = self.sigmoid(numpy.dot(l2, self.weights3))
458.
459.                     l3_error = self.outputData - l3
460.                     l3_delta = l3_error*self.sigmoid(l3,deriv=True)
461.
462.                     l2_error = l3_delta.dot(self.weights3.T)
463.                     l2_delta = l2_error*self.sigmoid(l2,deriv=True)
464.
465.                     l1_error = l2_delta.dot(self.weights2.T)
466.                     l1_delta = l1_error*self.sigmoid(l1,deriv=True)
467.
468.                     self.weights3 += l2.T.dot(self.learningRate*l3_delta)
469.                     self.weights2 += l1.T.dot(self.learningRate*l2_delta)
470.                     self.weights1 += l0.T.dot(self.learningRate*l1_delta)
471.
472.             def newDataPoint(self, char, player, OBJECTS):
473.                 current = numpy.array([(char.GetDirection()+4)/8])
474.                 current = numpy.append(current, [(char.DistanceTo(player))/1500,
475. (char.DirectionTo(player)+4)/8])
476.
477.                 return current
478.
479.             def collectData(self, character, player, OBJECTS, Refresh=False):
480.                 if self.unclaimedData[0][0] == 5:
481.                     self.unclaimedData = numpy.array([self.newDataPoint(character,
482. player, OBJECTS)])

```

```

476.         else:
477.             self.unclaimedData = numpy.append(self.unclaimedData,
[ self.newDataPoint(character, player, OBJECTS)], axis = 0)
478.
479.             if character.action == "combat":
480.                 act = [1,0,0,0]
481.             elif character.action == "turn":
482.                 act = [0,1,0,0]
483.             elif character.action == "flee":
484.                 act = [0,0,1,0]
485.             else:
486.                 act = [0,0,0,1]
487.
488.             if self.unclaimedResults[0][0] == 5:
489.                 self.unclaimedResults = numpy.array([act])
490.             else:
491.                 self.unclaimedResults = numpy.append(self.unclaimedResults, [act],
axis = 0)
492.
493.             if Refresh:
494.
495.                 if self.currentSuccessValue < 0:
496.                     self.unclaimedData = numpy.array([[5]])
497.                     self.currentSuccessValue = 0
498.                     self.unclaimedResults = numpy.array([[5]])
499.                 else:
500.                     tempInputs = numpy.append(self.unclaimedData, self.inputData,
axis = 0)
501.                     tempOutputs = numpy.append(self.unclaimedResults,
self.outputData, axis = 0)
502.
503.                     tempOrg = numpy.random.choice(len(tempInputs), size =
self.totalData, replace = False)
504.                     self.inputData = numpy.array([[5]])
505.                     for i in range(self.totalData):
506.                         if self.inputData[0][0] == 5:
507.                             self.inputData = numpy.array([tempInputs[tempOrg[i]])]
508.                             self.outputData =
numpy.array([tempOutputs[tempOrg[i]])]
509.                         else:
510.                             self.inputData = numpy.append(self.inputData,
[tempInputs[tempOrg[i]]], axis = 0)
511.                             self.outputData = numpy.append(self.outputData,
[tempOutputs[tempOrg[i]]], axis = 0)
512.
513.                     self.unclaimedData = numpy.array([[5]])
514.                     self.unclaimedResults = numpy.array([[5]])
515.
516.                     self.trainNeuralNetwork(50)
517.
518.             #forward propagation
519.             def extrapolate(self,newInput):
520.                 l0 = numpy.append(newInput, 1)
521.                 l1 = self.sigmoid(numpy.dot(l0, self.weights1))
522.                 l2 = self.sigmoid(numpy.dot(l1, self.weights2))
523.                 l3 = self.sigmoid(numpy.dot(l2, self.weights3))
524.
525.                 return l3
526.

```

```

527.     #sigmoid function - converts the outputs from each layer non-linearly into
    a number from 0-1
528.     #if using the derivative, a high derivative indicates more uncertainty
529.     def sigmoid(self, x, deriv=False):
530.         if (deriv==True):
531.             return x*(1-x)
532.
533.         return 1/(1+numpy.exp(-x))
534.
535.     #Inanimate Objects Class - class for things that won't move on their own -
    rocks and trees and such.
536.     #Might be able to break them.
537.     class Inanimate(Sprite):
538.         def __init__(self, img, sturdy, size = (150,150)):
539.             Sprite.__init__(self, img, size)
540.             self.sturdy = sturdy
541.             self.broken = False
542.
543.     #Attack Class - Used to display attacks
544.     class Attack(Sprite):
545.         def __init__(self, img, hold, size = (50,50)):
546.             Sprite.__init__(self, img, size)
547.             self.substantial = False
548.             self.hold = hold
549.             self.image1 = self.image
550.             self.image2 = pygame.transform.scale(self.image1, (self.size[0],
    self.size[1]))
551.             self.image2 = pygame.transform.flip(self.image2, True, False)
552.             self.state = 0
553.
554.         def Show(self, source, sourceSize, direction, display) :
555.             self.SetDirection(direction)
556.             delta =
    self.Delta(direction, (sourceSize[0]+self.size[0])/2, (sourceSize[1]+self.size[1])/2)
557.             self.Place((source[0]+delta[0]+(sourceSize[0]-self.size[0])/2, source[1]+delta[1]+(s
    ourceSize[1]-self.size[1])/2), display)
558.             self.state += 1
559.             if self.state >= 4:
560.                 self.image = self.image1
561.                 self.state = 0
562.             elif self.state >= 2:
563.                 self.image = self.image2
564.
565.     #Door Class - Interacting with one passes you to another room, unless a key is
    required.
566.     class Door(Sprite):
567.         def __init__(self, img, nextRoom, opensTo, size = (50,150), locked =
    False):
568.             Sprite.__init__(self, img, size)
569.             self.nextRoom = nextRoom
570.             self.interactable = True
571.             self.locked = locked
572.             self.opensTo = opensTo
573.             if self.locked:
574.                 self.text = ("Door is", "locked.", "")
575.             else:
576.                 self.text = ("Opening", "door...", "")
577.

```

```

578.     def IsLocked(self):
579.         return self.locked
580.
581.     def Unlock(self):
582.         self.locked = False
583.
584.     def Open(self):
585.         return self.nextRoom, self.opensTo
586.
587.     #Words class - Used to display text
588.     class Words(Sprite):
589.         def __init__(self, text, position, size):
590.             Sprite.__init__(self, "../Art/Words.png", (size,size))
591.             self.size = size
592.             self.position = position
593.             self.substantial = False
594.             self.text = text #pass a tuple as text to look through it for each line
595.             self.font = pygame.font.SysFont('bradleyhanditc', size, True)
596.
597.         def Place(self, position, surface, color = (0,0,0)):
598.             placement = 0
599.             self.SetPos(position)
600.             for i in self.text:
601.                 words = self.font.render(i, True, color)
602.                 surface.blit(words, (position[0],position[1]+placement))
603.                 placement += self.size+10
604.                 color = (color[0]+15, color[1]+15, color[2]+15)
605.
606.         def SetText(self, text):
607.             text = list(text)
608.             text.reverse()
609.             for i in text:
610.                 self.text = [i]+self.text
611.                 if len(self.text)> 11:
612.                     self.text.pop()
613.
614.     def main(AI, refreshCount, passedLevel = 1, enemies_killed = 0):
615.         pygame.init()
616.
617.         FPS = 30
618.         fpsClock = pygame.time.Clock()
619.
620.
621.         DISPLAYSURF = pygame.display.set_mode((DISPLAY_X, DISPLAY_Y), 0, 16)
622.
623.         pygame.display.set_caption("Mouse Vs. Machine")
624.
625.         BGCOLOR = (100,100,100)
626.
627.         bg = pygame.image.load('../Art/Background1.png')
628.
629.         player = Character('../Art/MouseStep1.png', '../Art/MouseStep2.png')
630.         player.Place((50,50), DISPLAYSURF)
631.         player.AdjustAttribute('Speed', 20)
632.         player.SetText(("How did you","get here?", ""))
633.
634.         rock1 = Inanimate('../Art/Rock1.png', 10, (100,100))
635.         rock1.SetInteractable(True)
636.         rock1.SetText(("It's a rock",""))

```

```

637.     rock1.Place((300,300), DISPLAYSURF)
638.
639.     rock2 = Inanimate('../Art/Rock1.png', 10, (150,150))
640.     rock2.SetInteractable(True)
641.     rock2.SetText(("It's a ", "large rock",""))
642.     rock2.Place((400,400), DISPLAYSURF)
643.
644.     rock3 = Inanimate('../Art/Rock1.png', 10, (50,50))
645.     rock3.SetInteractable(True)
646.     rock3.SetText(("It's a", "small rock",""))
647.     rock3.SetDirection(-3)
648.     rock3.Place((1100,150), DISPLAYSURF)
649.
650.     wall1 = Inanimate('../Art/Wall1.png', 100, (1250,50))
651.     wall1.Place((0,0), DISPLAYSURF)
652.
653.     wall2 = Inanimate('../Art/Wall1.png', 100, (1250,50))
654.     wall2.Place((0, DISPLAY_Y - 50), DISPLAYSURF)
655.
656.     wall3 = Inanimate('../Art/Wall2.png', 100, (50,1000))
657.     wall3.Place((0, 0), DISPLAYSURF)
658.
659.     wall4 = Inanimate('../Art/Wall2.png', 100, (50,1000))
660.     wall4.Place((DISPLAY_X - 300, 0), DISPLAYSURF)
661.
662.     wall5 = Inanimate('../Art/Wall2.png', 100, (50, 700))
663.     wall5.Place((900,0), DISPLAYSURF)
664.
665.     spider = Enemy("spider")
666.     spider.SetInteractable(True)
667.     spider.SetText(("AAAAAA", "AAAAAA", "SPIDER",""))
668.     spider.Place((700,500), DISPLAYSURF)
669.
670.     spider2 = Enemy("spider")
671.     spider2.SetInteractable(True)
672.     spider2.SetText(("AAAAAA", "it's another", "spider",""))
673.     spider2.Place((500,700), DISPLAYSURF)
674.
675.     spider3 = Enemy("spider")
676.     spider3.SetInteractable(True)
677.     spider3.SetText(("AAAAAA", "it's another", "spider",""))
678.     spider3.Place((400,800), DISPLAYSURF)
679.
680.     morph = Enemy("rock")
681.     morph.SetInteractable(True)
682.     morph.SetText(("It's a rock", "... or", "is it?",""))
683.     morph.Place((700,500), DISPLAYSURF)
684.
685.     lizard = Enemy("lizard")
686.     lizard.SetText(("Uh... It's", "a lizard...", "... maybe...", ""))
687.     lizard.Place((600,500), DISPLAYSURF)
688.
689.     intro = Words(['Use W A S D keys to move', 'Press LEFT SHIFT to interact',
'Press SPACE to attack'], (250,100), 40)
690.
691.     room1 = [intro, morph, rock1, rock3, wall1, wall2, wall3, wall4]
692.     room2 = [spider, rock1, rock2, wall1, wall2, wall3, wall4, wall5]
693.     room3 = [lizard, rock2, rock3, wall1, wall2, wall3, wall4]
694.     room4 = [spider2, spider3, rock3, wall1, wall2, wall3, wall4]

```

```

695.
696.     door1 = Door('../Art/DoorVertical.png', room2, (50, DISPLAY_Y/3))
697.     door1.Place((DISPLAY_X - 300, DISPLAY_Y/3), DISPLAYSURF)
698.     room1.append(door1)
699.     room1.append(player)
700.
701.     door2 = Door('../Art/DoorVertical.png', room1, (DISPLAY_X - 400,
DISPLAY_Y/3))
702.     door2.Place((0, DISPLAY_Y/3), DISPLAYSURF)
703.     room2.append(door2)
704.     room2.append(player)
705.
706.     door3 = Door('../Art/DoorVertical.png', room3, (50, 2*DISPLAY_Y/3))
707.     door3.Place((DISPLAY_X - 300, 2*DISPLAY_Y/3), DISPLAYSURF)
708.     room2.append(door3)
709.
710.     door4 = Door('../Art/DoorVertical.png', room2, (DISPLAY_X - 400,
2*DISPLAY_Y/3))
711.     door4.Place((0, 2*DISPLAY_Y/3), DISPLAYSURF)
712.     room3.append(door4)
713.
714.     door5 = Door('../Art/DoorHorizontal.png', room4, (DISPLAY_X/3, 50),
(150,50))
715.     door5.Place((DISPLAY_X/3, DISPLAY_Y-50), DISPLAYSURF)
716.     room3.append(door5)
717.     room3.append(player)
718.
719.
720.     door6 = Door('../Art/DoorHorizontal.png', room3, (DISPLAY_X/3,
DISPLAY_Y-150), (150,50))
721.     door6.Place((DISPLAY_X/3, 0), DISPLAYSURF)
722.     room4.append(door6)
723.     room4.append(player)
724.
725.     OBJECTS = room1
726.
727.     enemies = [morph, spider, lizard, spider2, spider3]
728.     for i in enemies:
729.         i.AdjustAttribute("Attack",1)
730.
731.     player.level = passedLevel
732.
733.     health = Words([str(player.GetHealth())], (1325, 95), 40)
734.     level = Words([str(player.level)], (1325, 270), 40)
735.     enemies_killed_str = Words([str(enemies_killed)], (1325, 425), 40)
736.     text = Words([""], (1310, 560), 25)
737.
738.     STATS = (health, level, enemies_killed_str, text)
739.
740.     DISPLAYSURF.fill(BGCOLOR)
741.
742.     dead = False
743.     justdied = True
744.
745.     while True: # game loop
746.
747.         if dead:
748.             DEATH_SCREEN = Words([' You have died',"Click anywhere to play
again.'], (250,100), 50)

```

```

749.         if justdied:
750.             DISPLAYSURF.fill(BG_COLOR, None, BLEND_RGBA_ADD)
751.             justdied = False
752.             DEATH_SCREEN.Place(DEATH_SCREEN.GetPos(), DISPLAYSURF)
753.             for event in pygame.event.get():
754.                 if event.type == QUIT:
755.                     pygame.quit()
756.                     sys.exit()
757.                 elif event.type == MOUSEBUTTONDOWN:
758.                     main(AI, refreshCount, player.level, enemies_killed)
759.
760.             else:
761.                 mouseClicked = False
762.                 INTERACT = False
763.                 DISPLAYSURF.blit(bg, (0,0))
764.
765.                 for event in pygame.event.get():
766.                     if event.type == QUIT:
767.                         pygame.quit()
768.                         sys.exit()
769.                     elif event.type == MOUSEBUTTONDOWN:
770.                         mouseClicked = True
771.                     elif event.type == KEYDOWN:
772.                         if event.key == K_LSHIFT:
773.                             INTERACT = True
774.
775.                 if mouseClicked:
776.                     print(pygame.mouse.get_pos())
777.
778.                 keys = pygame.key.get_pressed()
779.
780.                 LEFT = keys [K_a]
781.                 RIGHT = keys [K_d]
782.                 UP = keys [K_w]
783.                 DOWN = keys [K_s]
784.
785.                 ATTACK = keys[K_SPACE]
786.
787.                 if UP:
788.                     if LEFT:
789.                         player.Walk(-3, OBJECTS)
790.                     elif RIGHT:
791.                         player.Walk(-1, OBJECTS)
792.                     else:
793.                         player.Walk(-2, OBJECTS)
794.                 elif DOWN:
795.                     if LEFT:
796.                         player.Walk(3, OBJECTS)
797.                     elif RIGHT:
798.                         player.Walk(1, OBJECTS)
799.                     else:
800.                         player.Walk(2, OBJECTS)
801.                 elif LEFT:
802.                     player.Walk(4, OBJECTS)
803.                 elif RIGHT:
804.                     player.Walk(0, OBJECTS)
805.
806.                 if INTERACT:
807.                     for i in OBJECTS:

```



```

808.             if i.interactable:
809.                 if player.CanInteract(i):
810.                     text.SetText(i.GetText())
811.                     if isinstance(i, Door):
812.                         if not(i.IsLocked()):
813.                             OBJECTS = i.Open()[0]
814.                             player.SetPos(i.Open()[1])
815.                             break
816.
817.         for i in OBJECTS:
818.             if i.enemy == True:
819.                 i.Act(player, OBJECTS, AI, DISPLAYSURF)
820.                 i.IncrementAttack()
821.                 refreshCount += 1
822.                 if refreshCount >= 60:
823.                     AI.collectData(i, player, OBJECTS, True)
824.                     refreshCount = 0
825.                 elif refreshCount%10 == 0:
826.                     AI.collectData(i, player, OBJECTS)
827.                 i.Place(i.GetPos(), DISPLAYSURF)
828.
829.         if ATTACK:
830.             attack = player.Attack(OBJECTS, DISPLAYSURF, AI)
831.             if attack != False:
832.                 text.SetText(attack)
833.                 enemies_killed += 1
834.                 enemies_killed_str.text= [str(enemies_killed)]
835.                 if enemies_killed % 5 == 0:
836.                     player.level += 1
837.                     level.text = [str(player.level)]
838.                     for i in enemies:
839.                         i.AdjustAttribute("Attack",1)
840.                 for i in enemies:
841.                     if i.GetHealth() > 0:
842.                         break
843.                     elif i == enemies[-1]:
844.                         print i
845.                         for j in enemies:
846.                             j.AdjustAttribute("Health", j.maxHealth)
847.                             room1.append(morph)
848.                             room2.append(spider)
849.                             room3.append(lizard)
850.                             room4.append(spider2)
851.                             room4.append(spider3)
852.                             text.SetText(('Enemies have respawned.', ''))
853.
854.             if player.GetHealth()<=0:
855.                 text.SetText(('You died.', ""))
856.                 dead = True
857.
858.             health.text = ((str(player.GetHealth()), ""))
859.
860.             for i in STATS:
861.                 i.Place(i.GetPos(), DISPLAYSURF)
862.
863.             player.IncrementAttack()
864.             pygame.display.update()
865.             fpsClock.tick(FPS)
866.

```

```
867.DISPLAY_X = 1500
868.DISPLAY_Y = 1000
869.
870.     #Initializing A.I. with tested parameters
871.
872.inputArray = numpy.random.random((100,3))
873.outputArray = numpy.random.choice(numpy.array([0,1]),(100,4))
874.
875.inputArray = numpy.array([[0.25, 0.155688448415, 0.75],
876.     [0.0, 0.359467375886, 1.0],
877.     [0.875, 0.205917339058, 0.875],
878.     [0.375, 0.0244215663285, 0.375],
879.     [0.25, 0.122981080927, 0.75],
880.     [1.0, 0.0445640717722, 0.5],
881.     [0.75, 0.00174918873308, 0.75],
882.     [0.125, 0.40978027009, 0.125],
883.     [0.25, 0.00162555084805, 0.25],
884.     [0.125, 0.31369290117, 0.125],
885.     [0.75, 0.0169856590862, 0.75],
886.     [0.125, 0.250979347249, 0.125],
887.     [1.0, 0.069356639673, 0.625],
888.     [1.0, 0.0183547828482, 1.0],
889.     [0.0, 0.442144584082, 0.125],
890.     [1.0, 0.0283425991062, 1.0],
891.     [0.0, 0.137838199269, 0.25],
892.     [0.625, 0.0859784836301, 0.625],
893.     [0.0, 0.442144584082, 0.125],
894.     [0.125, 0.148542886138, 0.125],
895.     [0.625, 0.023392154901, 0.625],
896.     [0.0, 0.248126210978, 0.25],
897.     [0.75, 0.0842253858378, 0.5],
898.     [1.0, 0.18073343682, 1.0],
899.     [0.0, 0.360933873884, 0.125],
900.     [0.875, 0.010614069676, 0.875],
901.     [0.0, 0.061383174512, 0.25],
902.     [0.875, 0.0647056672056, 0.75],
903.     [0.625, 0.0939244492384, 0.625],
904.     [0.5, 0.0185975645382, 0.75],
905.     [0.25, 0.0692665914232, 0.125],
906.     [0.125, 0.435661582631, 0.125],
907.     [0.5, 0.0044696169736, 0.5],
908.     [0.5, 0.0044696169736, 0.5],
909.     [0.125, 0.180053798161, 0.125],
910.     [0.5, 0.136807806483, 0.625],
911.     [0.125, 0.377559289827, 0.125],
912.     [0.25, 0.255604542668, 0.75],
913.     [0.375, 0.00162766355362, 0.5],
914.     [0.125, 0.345515258608, 0.125],
915.     [0.625, 0.109662788442, 0.5],
916.     [0.75, 0.106793408048, 0.75],
917.     [0.0, 0.377233098106, 0.25],
918.     [0.75, 0.0449779700156, 0.625],
919.     [0.75, 0.112535012735, 0.75],
920.     [0.0, 0.167801257161, 0.125],
921.     [0.0, 0.314401177048, 0.875],
922.     [0.375, 0.0942999943698, 0.875],
923.     [0.75, 0.000450780762508, 0.75],
924.     [0.625, 0.0349267322222, 0.625],
925.     [0.75, 0.0828922756579, 0.625],
```

```

926. [0.0, 0.26769766435, 1.0],
927. [0.875, 0.113014916672, 0.875],
928. [0.625, 0.138438807039, 0.625],
929. [0.125, 0.28215289183, 0.125],
930. [0.25, 0.0713065461066, 0.375],
931. [0.375, 0.0244215663285, 0.375],
932. [0.0, 0.451074152234, 0.125],
933. [0.0, 0.347025310428, 1.0],
934. [0.0, 0.160463252102, 0.25],
935. [0.75, 0.162589285435, 0.875],
936. [0.375, 0.0201029300908, 0.875],
937. [0.0, 0.408786333698, 0.125],
938. [0.75, 0.00701060533436, 0.75],
939. [0.75, 0.00349146234724, 0.75],
940. [0.125, 0.311702641574, 0.125],
941. [0.125, 0.0612697932898, 0.125],
942. [0.75, 0.0945784306709, 0.625],
943. [0.25, 0.127738036617, 0.75],
944. [0.0, 0.279044327474, 0.875],
945. [0.5, 0.0594702950469, 0.625],
946. [0.625, 0.077533744631, 0.75],
947. [0.75, 0.0813965039013, 0.5],
948. [1.0, 0.127826580322, 1.0],
949. [0.25, 0.231612342578, 0.375],
950. [0.0, 0.323314276329, 1.0],
951. [0.875, 0.139011621748, 0.625],
952. [0.625, 0.165587331318, 0.625],
953. [0.75, 0.176877234685, 0.25],
954. [0.0, 0.296431289407, 0.25],
955. [0.5, 0.000702536573322, 0.5],
956. [0.75, 0.0011766863617, 0.75],
957. [0.625, 0.184832894578, 0.625],
958. [0.125, 0.108839791347, 0.625],
959. [0.875, 0.0471397508853, 0.875],
960. [0.875, 0.00746178804382, 0.875],
961. [0.625, 0.00482345524075, 0.625],
962. [0.25, 0.150248491452, 0.5],
963. [0.75, 0.000311347174161, 0.75],
964. [0.75, 0.00384111081162, 0.75],
965. [0.75, 0.0054213382633, 0.625],
966. [0.25, 0.000178748436944, 0.25],
967. [0.0, 0.100013551051, 0.375],
968. [0.75, 0.0236468897784, 0.5],
969. [0.0, 0.392372426319, 1.0],
970. [0.625, 0.00482345524075, 0.625],
971. [0.25, 0.0518126120541, 0.75],
972. [0.0, 0.23754116619, 0.875],
973. [0.625, 0.132427127748, 0.5],
974. [0.0, 0.321672246026, 0.25]]
975.
976. outputArray = numpy.array([[0, 0, 1, 0], [0, 0, 0, 1], [0, 1, 0, 0], [1, 0, 0,
0], [0, 1, 0, 0], [0, 1, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1,
0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [1,
0, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [1, 0, 0, 0], [1,
0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1,
0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [1,
0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0, 1, 0, 0],
[0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0,
0],

```

```

0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1,
0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0, 0, 0, 1], [0, 1, 0, 0], [0,
0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0],
[0, 1, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0], [0, 1, 0,
0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1,
0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0,
1, 0, 0], [1, 0, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [0, 1, 0, 0], [1, 0, 0, 0],
[0, 1, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [1, 0, 0, 0], [0, 0, 0, 1], [1, 0, 0,
0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 0, 0], [0, 0, 0, 1]]
977.
978.AI = NeuralNetworkAI(inputArray, outputArray,.05)
979.AI.trainNeuralNetwork(10000)
980.refreshCount = 0
981.
982.main(AI, refreshCount)

```

7.2 Test Functions

Test Networks 1 and 2 - used for comparing the efficacy of different numbers of hidden layers

```

1. def testNetwork1(self):
2.
3.     numpy.random.seed(1)
4.
5.     testInitial = numpy.array([[0,.5,1,0],
6.                                [0,0,1,0],
7.                                [1,0,1,0],
8.                                [1,1,.5,0]])
9.
10.    testResult = numpy.array([[.75],
11.                               [1],
12.                               [1],
13.                               [.83]])
14.
15.    randomWeights1 = 2*numpy.random.random((4,4)) - 1
16.    randomWeights2 = 2*numpy.random.random((4,1)) - 1
17.
18.    for i in xrange(10000):
19.        l0 = testInitial
20.        l1 = self.sigmoid(numpy.dot(l0,randomWeights1))
21.        l2 = self.sigmoid(numpy.dot(l1,randomWeights2))
22.
23.        l2_error = testResult - l2
24.
25.        if (i% 1000) == 0:
26.            print "Error:" + str(numpy.mean(numpy.abs(l2_error)))
27.
28.        l2_delta = l2_error*self.sigmoid(l2,deriv=True)
29.
30.        l1_error = l2_delta.dot(randomWeights2.T)
31.
32.        l1_delta = l1_error*self.sigmoid(l1,deriv=True)
33.
34.        randomWeights2 += l1.T.dot(l2_delta)
35.        randomWeights1 += l0.T.dot(l1_delta)

```

```

36.
37.     print l2
38.
39.     def testNetwork2(self):
40.
41.         numpy.random.seed(1)
42.
43.         testInitial = numpy.array([[0, .5, 1, 0],
44.                                     [0, 0, 1, 0],
45.                                     [1, 0, 1, 0],
46.                                     [1, 1, .5, 0]])
47.
48.         testResult = numpy.array([[.75],
49.                                   [1],
50.                                   [1],
51.                                   [.83]])
52.
53.         randomWeights1 = 2*numpy.random.random((4,4)) - 1
54.         randomWeights2 = 2*numpy.random.random((4,4)) - 1
55.         randomWeights3 = 2*numpy.random.random((4,1)) - 1
56.
57.         for i in xrange(10000):
58.             l0 = testInitial
59.             l1 = self.sigmoid(numpy.dot(l0, randomWeights1))
60.             l2 = self.sigmoid(numpy.dot(l1, randomWeights2))
61.             l3 = self.sigmoid(numpy.dot(l2, randomWeights3))
62.
63.             l3_error = testResult - l3
64.
65.             if (i% 1000) == 0:
66.                 print "Error:" + str(numpy.mean(numpy.abs(l3_error)))
67.
68.             l3_delta = l3_error*self.sigmoid(l3, deriv=True)
69.
70.             l2_error = l3_delta.dot(randomWeights3.T)
71.
72.             l2_delta = l2_error*self.sigmoid(l2, deriv=True)
73.
74.             l1_error = l2_delta.dot(randomWeights2.T)
75.
76.             l1_delta = l1_error*self.sigmoid(l1, deriv=True)
77.
78.             randomWeights3 += l2.T.dot(l3_delta)
79.             randomWeights2 += l1.T.dot(l2_delta)
80.             randomWeights1 += l0.T.dot(l1_delta)
81.
82.         print l3

```

Test Neural Network - used to compare different learning rates and network sizes

```

1. def testNN(inputArray, outputArray):
2.     testing = NeuralNetworkAI(inputArray, outputArray, .05)
3.
4.     testing.trainNeuralNetwork(5000)
5.
6.     print "Extrapolated: " + str(testing.extrapolate(inputArray[0]))
7.     print "Expected: " + str(outputArray[0])

```

```
8.     print "Difference: " + str(outputArray[0] - testing.extrapolate(inputArray[0]))
      + "\n"
9.
10.    print "Extrapolated: " + str(testing.extrapolate(inputArray[1]))
11.    print "Expected: " + str(outputArray[1])
12.    print "Difference: " + str(outputArray[1] -
      testing.extrapolate(inputArray[1]))+ "\n"
13.
14.    print "Extrapolated: " + str(testing.extrapolate(inputArray[6]))
15.    print "Expected: " + str(outputArray[6])
16.    print "Difference: " + str(outputArray[6] -
      testing.extrapolate(inputArray[6]))+ "\n"
17.
18.    print "Extrapolated: " + str(testing.extrapolate(inputArray[8]))
19.    print "Expected: " + str(outputArray[8])
20.    print "Difference: " + str(outputArray[8] -
      testing.extrapolate(inputArray[8]))+ "\n"
21.
22.    print "Extrapolated: " + str(testing.extrapolate(inputArray[93]))
23.    print "Expected: " + str(outputArray[6])
24.    print "Difference: " + str(outputArray[6] -
      testing.extrapolate(inputArray[6]))+ "\n"
25.
26.    print "Extrapolated: " + str(testing.extrapolate(inputArray[95]))
27.    print "Expected: " + str(outputArray[8])
28.    print "Difference: " + str(outputArray[8] -
      testing.extrapolate(inputArray[8]))+ "\n"
```

7.3 Survey

1. How did you find fighting the enemies in the first version?

Too hard Somewhat hard Just right Somewhat easy Too easy

2. In the second version?

Too hard Somewhat hard Just right Somewhat easy Too easy

3. How long did you play the game for? _____

4. How would you score the game play?

Poor 1 2 3 4 5 Excellent

5. How would you score the graphics design?

Poor 1 2 3 4 5 Excellent

6. Do you have any suggestions for future progression of this game or any other comments?

8

Bibliography

- Barnson, Jay. "How To Build a Game In A Week From Scratch With No Budget." *How To Build a Game In A Week From Scratch With No Budget*. GameDev.net, 06 July 2005. Web. Fall 2016.
- Bourg, David M., and Glenn Seemann. "Four Cool Ways to Use Neural Networks in Games." *ONLamp.com*. O'Reilly Media, Inc, 30 Sept. 2004. Web. Apr. 2017.
- Britz, Denny. "Implementing a Neural Network from Scratch in Python – An Introduction." *WildML*. Wordpress, 10 Jan. 2016. Web. 20 Apr. 2017.
- Brownlee, Jason. "How to Implement the Backpropagation Algorithm From Scratch In Python." *Machine Learning Mastery*. Machine Learning Mastery, 02 Jan. 2017. Web. 20 Apr. 2017.
- Graft, Kris. "When Artificial Intelligence in Video Games Becomes...artificially Intelligent." *Gamasutra: The Art & Business of Making Games*. UBM Technology, 22 Sept. 2015. Web. Nov. 2016.
- Iamtrask. "A Neural Network in 11 Lines of Python (Part 1)." *A Neural Network in 11 Lines of Python*. N.p., 12 July 2015. Web. 03 Mar. 2017.
- Sweigart, Al. *Making Games with Python & Pygame: A Guide to Programming with Graphics, Animation, and Sound*. 1st ed. Charleston, SC: Creative Commons, 2012. *Invent with Python*. Web. Oct. 2017.