Fundamentals of Software Engineering

Tetris Learning Bot

Progress Report

November 30, 2015

Team #5
Contributors:
Matthew Durant
Trae Hurley
Jarod Hart
Khalid Alqammash

Tetris Learning Bot

Table of Contents

Introduction

About the software:

This program is an artificial intelligence(bot) that learns how to play a flash game. The bot records every move to increase accuracy of playing. Each game will improve the bot's ability to make decisions. In this software we will use Python language because it is easier to do machine learning with Python because a lot of members of the machine learning community have already written code, reducing our workload. Python also allows complex ideas to be presented in simple code. This bot will be playing Tetris by reading the screen, analyzing the inputs it receives from screen

reading, and making decisions based on what it sees as well as previous experienced (this is the machine learning part).

We chose this project mostly because we felt it would be a challenge to each of the team members' various levels of skill. It also helped us gain experience in a language which was new to all of us, Python. This project was also applicable to our daily lives, as well as many other young people, because we play video games frequently.

By the completion of this project, we expect that our software will be able to learn how to play various games. We would like for it to be able to learn quickly and eventually work for games that have a faster pace or a higher skill level, more than just simple flash games. Realistically, however, we believe reaching the goal of playing a single, slower-paced game would be more appropriate for our given time frame.

We have learned many things throughout the software development process for our project. We are learning a new language, Python, which is useful for scripting. We are also learning the importance of appropriately planning our time usage and the importance of firm leadership. We initially estimated that our project would take the entire semester to complete, but with multiple projects, tests, and homework for other courses, this became much less likely to happen. We have also learned that such a massive difference in skill level can cause issues between the work load. At

this point in our software development, we also have learned the true importance of completing design and analysis before beginning to actually code.

About the team:

We meet regularly twice a month and by today we have met more than five times. During these meetings we discussed all what we need to finish this project on the principles of software engineering. For example, we divided the work between the members depends on the strengths of each member. Who good at coding will focus more on implementation and who good at analyzing will focus on requirements and design analysis. However, everyone contributes in each phase by discussing the whole project. So, we have communicated to understand the needs and determining the plan.

Project Plan

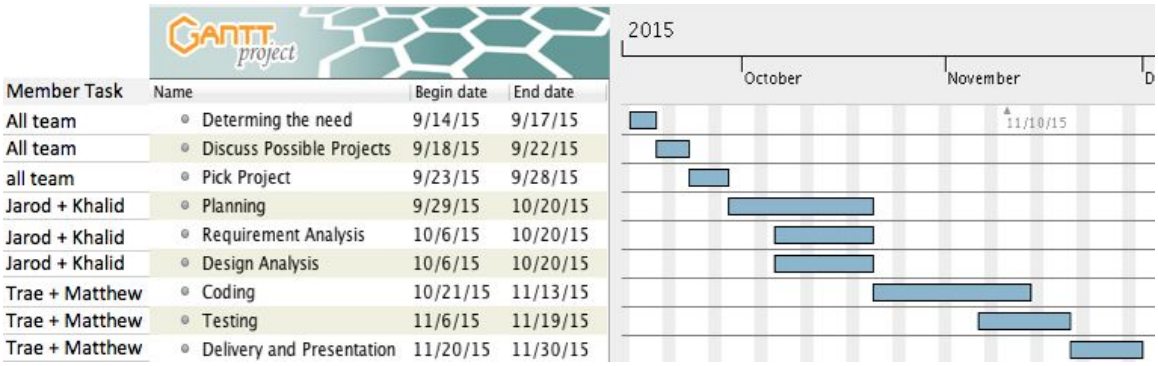
The scope of this project was to: create a software that will be able to read a screen and make decisions on its own based on what it sees on the screen. The main tasks are as follows:

- Build a user interface

- This interface will consist of a menu allowing the user to select from different options such as: start bot, define input keys, select area, record, pause, and a visible reward.
- There will be a submenu within the “select area” consisting of: new bounds, modify area, and display current areas.

- Ability to extract feature vector
  - The bot will take a screenshot of an area and extract the score from the game.
  - It will also extract the RGB values from the screen to be able to play the game.
- Ability to learn from previous attempts and experiences
  - The bot will play through a game and improve its play style based on a reward system that will reward it for doing well based on the score it extracts from the screenshot, as mentioned above.

Project Schedule



Member Task	Name	Begin date	End date
All team	Determing the need	9/14/15	9/17/15
All team	Discuss Possible Projects	9/18/15	9/22/15
all team	Pick Project	9/23/15	9/28/15
Jarod + Khalid	Planning	9/29/15	10/20/15
Jarod + Khalid	Requirement Analysis	10/6/15	10/20/15
Jarod + Khalid	Design Analysis	10/6/15	10/20/15
Trae + Matthew	Coding	10/21/15	11/13/15
Trae + Matthew	Testing	11/6/15	11/19/15
Trae + Matthew	Delivery and Presentation	11/20/15	11/30/15

Team members and duties

The team working on this project consists of: Trae Hurley, Khalid Alqammash, Matthew Durant, and Jarod Hart. Their duties are as follows:

- Trae Hurley
    - Develop the screen reader and main GUI
    - Integrating the GUI into the system itself
    - Integrating the system components
- Khalid Alqammash
    - Plan schedule
    - Create diagrams for processes
- Matthew Durant
    - Developing main learning algorithm
    - Assisting with integration of components
    - The portion of the software that handles outputs
- Jarod Hart
    - Create documentation
    - Assist in creating diagrams for software processes

Risk

A majority of the risk of our project lies in the coding and the people working on it. As students, the timeline itself is a huge risk for any project; given only a few months to complete an entire project can be difficult for anyone. Another risk based on being students is other course work. Taking five or more classes a semester can lead

to a large workload for the student, lowering their chances of completing a full-semester project. Luckily, the chance of this project not being completed on time is lowered for each skilled member of the group. The best way to mitigate this risk entirely would have been to work on the project periodically throughout the semester, for 3-5 hours a week.

The other biggest risk associated with our project is the complexity of our design. We wanted to make a program that could automatically play a set of different games with minimal user interaction. This project was much more difficult than some of the other ideas we could have done, which is why we chose it and also why there is so much risk in the project idea alone. Many high-functioning algorithms are difficult to program by themselves, but we are also adding a machine learning aspect to our software. This increases the difficult even more, leading to more time required to complete the project and increasing the likelihood of any errors. Again, the best way to mitigate this risk would have been to plan ahead and periodically work on the project throughout the semester. This would allow us to troubleshoot the code periodically, meet to discuss any progress or difficulties that presented themselves, and it would have allowed for each person to get more deeply involved in the coding.

Requirements Analysis

This system will have multiple functions. The first two functionalities are prior to starting the bot. They consist of defining the area for the screen to read and actually clicking the button to begin the bot.

The first function of the bot once it has started will be to read the screen. After it receives the feature vector from the screen read it will estimate the most likely action to take based on previous experiences. It will continue to do this indefinitely, until the game finishes, the bot is paused, or the bot loses the game. After the termination of the bot's process, it performs gradient descent to improve the estimation of the actions it will make for the next attempted playthrough. Once the analysis is finished, the bot may resume playing at the discretion of the user.

Nouns:

- AI: This will be what controls the software as it plays the games
- Player: User (or Player) will input the basic commands needed to play the game, as well as operate the menus
- Menu:
    - Start AI: This initializes the software
    - End AI: This ends the software

    - Record Player: This is one way the software can learn how to play the game
- Game: Tetris, required before the software can begin working
- Controls: How the software can play the game
- Record: The software/player will be recorded through the software so the machine can learn from them
- Karma Chart: A reward system used to tell the software if it performing well or poorly

Primary Classes: Screen reader, neural network, output, save state, main menu.

Optional Classes: Recording, pre-registered games, player input, karma system.

Use-Case Development:

1. User loads game.
2. User loads AI.
3. AI registers game.
4. User trains the program.
5. AI plays game until it fails.

Full Requirements:

The system must be able to play Tetris with minimal user interactions. It will do so by taking an image of the screen every frame, analysing the current positions of each piece, and deciding how it should move the current piece in order to most efficiently

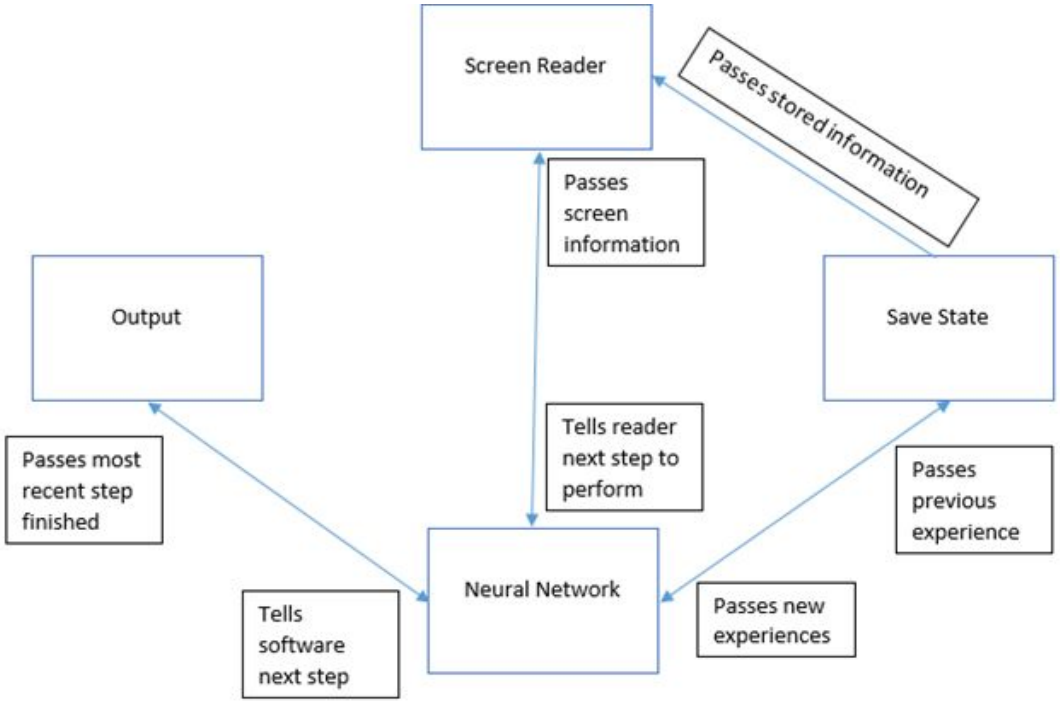
complete the level. The system should also have a menu that can be used to start, stop, pause, record, and set the area of the screen to get images taken from. We would like the software to be able to improve upon itself as it progresses through the levels or makes mistakes. This is to help reduce user interactions with the software. It would be preferable if the machine could learn the controls of the game on its own, but allowing users to input the controls is also acceptable. Finally, we would like for the machine to be able to learn from a user playing as well as from playing by itself.

System Design

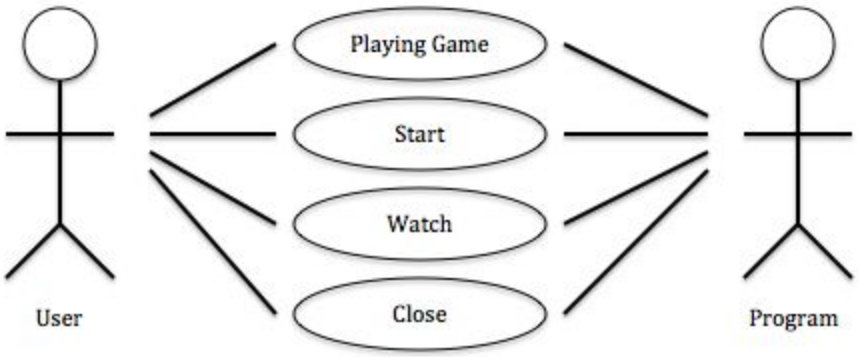
Product Design

The data the bot receives from the screen will be stored in the Neural Network. The user interface will consist of multiple buttons and a submenu. The buttons will be linked to certain parts of the code to perform various actions. These actions include: running the bot, pausing the bot, defining the area (which leads to a submenu), defining input keys for use, recording the run, terminating the bot, and displaying a reward at the bottom. The submenu (define area) will have options for creating new bounds, altering the existing area, viewing existing areas, and a button to return to the main menu screen.

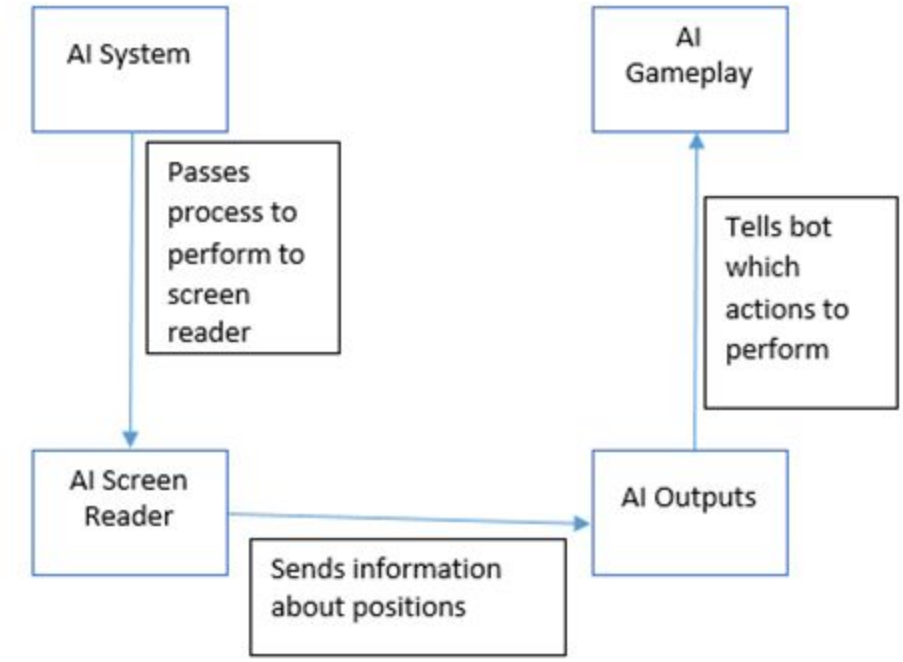
Class Diagram


Screen Reader
Passes stored information
Passes screen information
Output
Save State
Tells reader next step to perform
Passes most recent step finished
Passes previous experience
Neural Network
Tells software next step
Passes new experiences

Use Case Diagram



Interaction Diagram



Testing Plan

When the program is complete, we plan on testing each part of the program separately first to make sure each part works, and then finally test the whole thing to see if the program is learning at all. The part we will be individually testing are:

- Screen Reader to make sure it can get an area, and get an image from the area selected
- Number reader to make sure it can read numbers
- Menu to make sure the GUI works as planned

We will be using a flash based tetris game that can be found on tetris.com as our testing material for testing the whole program.

Final System Analysis

The Final system in place has a GUI class that is the main class. This can run the screen reader, and starts the neural network for the bot to play the game. The menu uses the QAlgorithm class as a learning class to teach the program how to play the game,make the bot play the game, and get the desired outputs. This algorithm is based off of the Deep Q-learning algorithm with Experience Replay (Minh). The Algorithm calls the neural network that has a Q learning function in it already, and expands it so that it can attempt to predict the future. The neural network we used was PyBrain, and is a recurrent neural network that can remember past events

more easily without our implementation. We also had to make an environment and a task class so the neural network can do most of the algorithm for us by setting up how the game works. Based on what the algorithm outputs, it will send an output that the bot converts to a key press. Sometimes this output is random if there has not been enough data in any given situation to see what rewards it would give. This program also pickles itself so it can be in the same state for the next use, and can gather a history over previous uses. We also have 2 instances of the screen reader running as seen in the menu class in appendix A, so the program can tell the difference between the score and the actual game, and run the number reader through the score.

This program has not been tested much as of the time this paper has been written, and general outputs have not been tested. Because this is a learning bot, it would be hard to test extensively other than making sure the gui works with the output. The screen reader works as it gets an area for recording, and sends it to the neural network. The number reader can read certain numbers, but needs to be trained more for it to be used effectively.

Issues that occurred were figuring out the Algorithm, and implementing it with the neural network. We had problems figuring out transferring the algorithm listed from the paper in the reference section into code, but ended up deciding to use the neural network's version of the Q algorithm to help us out. Because all of us have

other classes, we took too long to try and get the program done as work/classes/personal issues got in the way. The code is done, but we needed more time to test the learning thoroughly to get the right numbers to make it as optimized as possible(like minibatch, frames a second, ect.). We tested the individual parts other than the Algorithm separately, and they all work the way it was designed.

Conclusion

The end goal of this system was for it to be able to completely play a game of Tetris by itself with little to no user interference. In order to do this, the software would need to read a screen, gather information from what it has read, relay the information to an internal algorithm, and determine the next move to make in the game through the algorithm. As of the writing of this paper the program runs, but does not seem to learn well. A reason for this could be that the rewards function is not well defined, and we would need to have a formula to make it better. We have learned many things from this project. For instance, we have learned time management is an essential skill in the real world and it will take practice to develop it. We chose to use Python for this project, which some of us had no prior experience in, so it was a new language and environment for most of us. We also learned the inner workings of how machine learning worked, and how neural networks work as well (Trae was the only one with prior experience in machine learning).

Appendix A: Class Skeletons

class ScreenReader:

def checkEndGame(self,x,img): #Checks to see if a saved area has changed

def readNumber(self,img): #Reads an area to find out what number is in that area

def GetIntensityImage(self,image): #returns an array describing the rgb intesnsites of an image

def RGBToInt(self,p): # turns a list of RGB values into a single number

def grabArea(self,bbox): # turns a defined area into (x0-x1,y0-y1) into an image

def SaveAreaCoordinates(self,e): #used to get the Coordinates of an area of interest

def veiwArea(self, area): #Views the current selected area

def selectArea(self, prevar = None): #makes a window to be used to select an area of interest

def TestSelectArea(self): #test select area function

def TestRGBToInt(self): #tests rgb to int conversion

def TestGrabScreen(self): #test screen grabing

def TestCompare(self):#test hashing compasions

class Output:

def PressKey(hexKeyCode): # function that presses the key

def ReleaseKey(hexKeyCode): # function that lifts the key up

def getKeyforOutput(string): #gets a string that converts a key name to hexa for output

def getKeyforRelease(string): #gets a string that converts a key name to hexa for release

class QAlgorithm:

def Pause(self):#if menu says pause pause execution

def Quit(self):#if menu says quit stop running

def Start(self):#starts the Bot

def CheckState(self):#checks to see what state the menu says to be in

def GameOver():#checks to see if state requires bot pause, quit or if the game is over

def runBot():#runs the bot for a single Episode

class menu:

#this starts the bot for learning
def StartBot(e):

#This sets a new box bounds for the game area
def NewBox(e):

#this gets the score area, and sets it up to be recorded
def ScoreArea(e):

#this pauses the learning bot
def PauseBot(e):

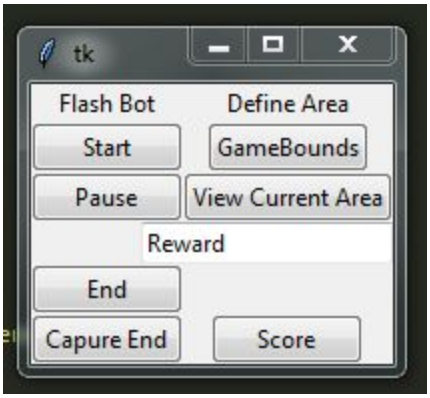
#this ends the learning bot
def EndBot(e):

#This captures the game over screen
def CaptureEnd(e):

Appendix B: References

Minh, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (n.d.). Playing Atari with Deep Reinforcement Learning. Retrieved November 30, 2015, from https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

Appendix C: Sample User Interfaces



Appendix D: Final Code

TTask:

```
from pybrain.rl.environments.episodic import EpisodicTask
import Output as opt
from screenReader import ScreenReader
#keycodes
a= ["LEFT","UP","DOWN","RIGHT"]
sc = ScreenReader()
class TTask(EpisodicTask):
    def __init__(self, environment,ScoreBox,GameOver):
        """ All tasks are coupled to an environment. """
        self.env = environment
        self.N = 1500000000000000
        self.t = 0
        # limits for scaling of sensors and actors (None=disabled)
        self.sensor_limits = None
        self.actor_limits = None
        self.clipping = True
        self.scoreBox = ScoreBox
        self.discount = .1# Discount Rate for actions
        self.batchSize = 32#how many actions to learn over
        self.gameOver = GameOver
```

```
    def isFinished(self):
        if self.t >= self.N:
            self.t = 0
            return True
        else:
            self.t += 1
            return False
    def getObservation(self):
        self.state = self.env.getSensors()
        return self.state

     #Sets an action
    def performAction(self, ac):
        self.env.performAction(ac)

    def getReward(self):
        # sleep(0.01)
        # print(self.state, self.action)
        score = sc.GetScore(sc.grabArea(self.scoreBox))
        reward = 0
        for x in range(len(score)):
            reward += score[x]*10**x
        return reward
```

TEnvironmnet:

```
from scipy import *
import sys, time
from screenReader import ScreenReader
import Output as opt
from pybrain.rl.environments.environment import Environment
from PIL import Image
from numpy import int64
#keycodes
a= ["LEFT","UP","DOWN","RIGHT"]
sc = ScreenReader()
class TEnviroment(Environment):
   def __init__(self, area):
      self.indim = 4
      self.outdim = 50*50

      self.reset()
      self.area = area
      self.state = None
```

```
        self.sequence = None
        self.target = None
        self.size = 10



    #sees if state has been updated, and returns it
    def getSensors(self):
        time.sleep(1)
        self.update()
        return self.state
    #Sets an action
    def performAction(self, ac):
        self.updated = False
        self.action = ac
        opt.getKeyforOutput(a[int64(ac[0]).item()])
        time.sleep(.005)
        opt.getKeyforRelease(a[int64(ac[0]).item()])

    #An update method that gets called every number of frames that updates the state
    def update(self):
        im = sc.grabArea(self.area).resize((50,50))
        self.state = sc.GetIntensityImage(im)

    #resets the varriables
    def reset(self):
        self.state = None
        self.updated = True
        self.sequence = []
```

ScreanReader:

```
from PIL.ImageGrab import grab
import imagehash
import time
import pickle
from PIL import Image,ImageOps
try:
    # for Python2
    from Tkinter import *
except ImportError:
    # for Python3
    from tkinter import *

class ScreenReader:
    def Segment (self, img):#seperates the digits out of raw score
```

```
    im = img.resize((52,64))#resizes image so it the happens properly
    im = ImageOps.grayscale(im)
    colThresh = 255*.817*im.size[1]#calculates threshold of white space in collums
(white intesnity)*81.7%*numcollums
    rowThresh = 255*.66*im.size[0]#calculates threshold of white space in rows
(white intesnity)*66%*numrows

    width = im.size[0]#gets number of collums
    height = im.size[1]#gets number or rows

    colSum = [ 0 for x in range(width)]#creates an array to hold sum of collum
intesnisty
    rowSum = [0 for y in range(height)]#creates an array to hold sum of row
intesnisty
    for x in range(width):#sums along collums and rows
      for y in range(height):
        colSum[x] += im.getpixel((x,y))
        rowSum[y] += im.getpixel((x,y))
    cols = []#holds a list of cordinants for start and end of didgit collums
    rows = []#holds a list of cordinants for start and end of didgit rows
    testprev, testcurrent = False,False#finds the start and end of digits collums
    for x in range(len(colSum)):
      if(colSum[x]<colThresh):
        testprev = testcurrent
        testcurrent = True
      else:
        testprev = testcurrent
        testcurrent = False
      if(testprev == False and testcurrent == True):#from higher than threshold to
below start of digit
        cols.append(x)
      if(testprev == True and testcurrent == False):#from lower than threshold to
higher end of digit
        cols.append(x)

    testprev, testcurrent = False,False
    for x in range(len(rowSum)):#finds the start and end of digits rows
      if(rowSum[x]<rowThresh):
        testprev = testcurrent
        testcurrent = True
      else:
        testprev = testcurrent
        testcurrent = False
```

```
        if(testprev == False and testcurrent == True):#from higher than threshold to
below start of digit
            rows.append(x)
        if(testprev == True and testcurrent == False):#from lower than threshold to
higher end of digit
            rows.append(x)
    crop =[]#holds cropped images
    deltax = width*0.023529411764705882 #slight padding to get whole digit
collums
    deltay = height*0.037037037037037035 #slight padding to get whole digit
rows
    rowMin = min(rows)
    rowMax = max(rows)
    while(len(cols)%2 != 0):
      cols = cols[:len(cols)-2]
    for x in range(0,len(cols),2):

crop.append(img.crop((cols[x]-int(deltax),rowMin-int(deltay),cols[x+1]+int(deltax),
rowMax+int(deltay))))
    return crop

  def GetScore(self,img):#reads score from screen
      digits = self.Segment(img)
      score = []
      for x in digits:
        temp = x.resize((32,32))
        im = ([int('0',16) for z in range(32*32)])
        for y in range (32):
          for x in range(32):
            im[x*32+y] = (self.RGBToInt(img.getpixel((x,y))))
        temp = self.net.activate(im)
        l = temp.argmax()
        if l == 9:
          l = 0
        else:
          l = l+1
        score.append(l)
      return score

  def checkEndGame(self,x,img):#Checks to see if a saved area has changed
       return imagehash.average_hash(img) ==
imagehash.average_hash(self.grabArea(x))

  def readNumber(self,img):#Reads an area to find out what number is in that area
    img = img.resize((32,32))
```

```
        im = ([int('0',16) for z in range(32*32)])
        for y in range(32):
            for x in range(32):
                im[x*32+y] = (RGBToInt(img.getpixel((x,y))))
        temp = number.activate(img)
        index = temp.index(max)
        if temp == 10:
            return 0
        else :
            return temp

    def GetIntensityImage(self,image):#returns an array describing the rgb
intesnsites of an image
        intensityMap = [None for y in range(image.width*image.height)]
        for x in range(image.width):
            for y in range(image.height):
                intensityMap[image.width-1*image.height-1+x] =
self.RGBToInt(image.getpixel((x,y)))
        return intensityMap

    def RGBToInt(self,p):# turns a list of RGB values into a single number
        return int('%02x%02x%02x' % (p[0], p[1], p[2]), 16)

    def grabArea(self,bbox):# turns a defined area into (x0-x1,y0-y1) into an image
        return grab(bbox)

    def SaveAreaCoordinates(self,e):#used to get the Coordinates of an area of intrest
        temp = self.root.winfo_geometry()
        #print(temp)
        p1 = temp.find("x")
        p2 = temp.find("+",p1+1)
        p3 = temp.find("+",p2+1)
        x0 = int(temp[p2+1:p3])
        y0 = int(temp[p3+1:])
        x1 = int(temp[0:p1]) +x0
        y1 = int(temp[p1+1:p2]) +y0 + 5
        self.selectedArea = [x0,y0,x1,y1]
        self.root.destroy()
        self.quit = True
        return self.selectedArea

    def veiwArea(self, area):
        root = Tk()
        root.geometry(area)
        root.mainloop()
```

```
    def selectArea(self, prevar = None):#makes a window to be used to select an area
of interst
        self.root = Tk()
        self.quit = False
        self.root.geometry(prevar)
        self.root.bind('<Return>',self.SaveAreaCoordinates)
        self.root.wm_attributes("-topmost", 1)
        while not self.quit:
            self.root.update_idletasks()
            self.root.update()
        return self.selectedArea

    def TestSelectArea(self):#test select area function
        print(self.selectArea())

    def TestRGBToInt(self):#tests rgb to int conversion
        print("Enter r")
        r = int(input())
        print("Enter g")
        g = int(input())
        print("Enter b")
        b = int(input())
        print(self.RGBToInt((r,g,b)))

    def TestGrabScreen(self): #test screen grabing
        area = [0,0,0,0]
        for i in range(4):
            print("Enter point cordinante")
            area[i] = int(input())
        image = self.grabArea(tuple(area))
        print(image)
        return image

    def Quit(self):
        sys.exit(0)

    def TestCompare(self):#test hashing compasions
        i1 = grab()
        i2 = i1.copy()
        print("Change screen")
        time.sleep(15)
        i3 = grab()
        hash1 = imagehash.average_hash(i1)
        hash2 = imagehash.average_hash(i2)
```

```
        hash3 = imagehash.average_hash(i3)
        print("image 1 vs image 2")
        print(hash1 == hash2)
        print("image 1 vs image 3")
        print(hash1 == hash3)

    def switch(self,x):#navagates testing menu
        return {
            '1': self.TestSelectArea,
            '2': self.TestGrabScreen,
            '3': self.TestCompare,
            '4': self.TestRGBToInt,
            '0': self.Quit,
        }[x]()

    def test(self):#unit test for Screan reader
        select = -12
        while(select != '0'):
            print("=====Testing Mode=======")
            print("1)select area")#stores an area so operations can be done on it
            print("2)sceen shot")#takes a screen shot of area so it can be loaded to
memory
            print("3)get Screen Hash")#makes a hash of image for comparison
            print("4)get rgb single value")#returns the rgb as a single value
            print("0)exit")
            select = input()
            self.switch(select)

    def __init__(self,mode=0):#creats a screen reader object and loads neural network
that was taught to read numbers
        self.net = pickle.load(open("Number.txt","rb"))
        self.root = None
        self.selectedArea = None
        if(mode != 0):
            self.test()
```

QAlgorithm:

```
from PIL.ImageGrab import grab
import imagehash
import time
import pickle
import numpy as np
import random
import Output
from screenReader import ScreenReader
```

```
from TEnviroment import TEnviroment
from TTask import TTask
from pybrain.rl.learners.valuebased import ActionValueNetwork
from pybrain.rl.learners import Q
from pybrain.rl.agents import LearningAgent
from pybrain.rl.experiments.episodic import EpisodicExperiment
import multiprocessing
import sys

class QAlgorithm:
  def Pause(self):#if menu says pause pause exicution
    while self.state == 1:
      time.sleep(.05)
    return True

  def Quit(self):#if menu says quit stop running
    self.process.terminate()
    return False

  def Start(self):#starts the Bot
    if self.process == None:
      self.runBot()
      #self.process = multiprocessing.Process(target=self.runBot, args= [])
      #self.process.start()
    return True

  def CheckState(self):#checks to see what state the menu says to be in
    if self.state == 0 :
      self.Start()
    elif self.state == 1:
      self.Pause()
    elif self.state == 2:
      self.Quit()

  def GameOver(self):#checks to see if state requires bot pause, quit or if the game is
over
    return self.CheckState() or self.sr.checkEndGame(self.endBox,self.gameOver)

  def __init__(self,rewardBox,box,gameOver,endGame,scoreArea):
    self.reward = rewardBox
    self.bbox = box
    self.environment = TEnviroment(box)#Custom environment class
    self.controller = ActionValueNetwork(50**2,4)#Arguments
(framerate*maxPlaytime, Number of acitons)
    self.learner = Q()
```

```
    gf = {0:self.GameOver}
    self.agent = LearningAgent(self.controller, self.learner)
    self.task = TTask(self.environment,scoreArea,gf)#needs custom task
    self.experiment = EpisodicExperiment(self.task, self.agent)
    self.process = None
    self.gameOver = gameOver
    self.endBox = endGame

  def runBot(self):#runes the bot for a single Episode
    while(True):
      self.experiment.doEpisodes()
      self.agent.learn()
      self.agent.reset()
      time.sleep(5)
```

Output:

```
import ctypes
import time

SendInput = ctypes.windll.user32.SendInput
#Hex Keycode
DictionaryofKeys = {
'KEY_LEFT' : 0x25,
'KEY_UP' : 0x26,
'KEY_DOWN' : 0x27,
'KEY_RIGHT' : 0x28,
'KEY_0' : 0x30,
'KEY_1' : 0x31,
'KEY_2' : 0x32,
'KEY_3' : 0x33,
'KEY_4' : 0x34,
'KEY_5' : 0x35,
'KEY_6' : 0x36,
'KEY_7' : 0x37,
'KEY_8' : 0x38,
'KEY_9' : 0x39,
'KEY_A' : 0x41,
'KEY_B' : 0x42,
'KEY_C' : 0x43,
'KEY_D' : 0x44,
'KEY_E' : 0x45,
'KEY_F' : 0x46,
'KEY_G' : 0x47,
'KEY_H' : 0x48,
'KEY_I' : 0x49,
```

```
'KEY_J' : 0x4A,
'KEY_K' : 0x4B,
'KEY_L' : 0x4C,
'KEY_M' : 0x4D,
'KEY_N' : 0x4E,
'KEY_O' : 0x4F,
'KEY_P' : 0x50,
'KEY_Q' : 0x51,
'KEY_R' : 0x52,
'KEY_S' : 0x53,
'KEY_T' : 0x54,
'KEY_U' : 0x55,
'KEY_V' : 0x56,
'KEY_W' : 0x57,
'KEY_X' : 0x58,
'KEY_Y': 0x59,
'KEY_Z' : 0x5A
}

# C struct redefinitions
PUL = ctypes.POINTER(ctypes.c_ulong)
class KeyBdInput(ctypes.Structure):
   _fields_ = [("wVk", ctypes.c_ushort),
               ("wScan", ctypes.c_ushort),
               ("dwFlags", ctypes.c_ulong),
               ("time", ctypes.c_ulong),
               ("dwExtraInfo", PUL)]

class HardwareInput(ctypes.Structure):
   _fields_ = [("uMsg", ctypes.c_ulong),
               ("wParamL", ctypes.c_short),
               ("wParamH", ctypes.c_ushort)]

class MouseInput(ctypes.Structure):
   _fields_ = [("dx", ctypes.c_long),
               ("dy", ctypes.c_long),
               ("mouseData", ctypes.c_ulong),
               ("dwFlags", ctypes.c_ulong),
               ("time",ctypes.c_ulong),
               ("dwExtraInfo", PUL)]

class Input_I(ctypes.Union):
   _fields_ = [("ki", KeyBdInput),
                ("mi", MouseInput),
                ("hi", HardwareInput)]
```

```
class Input(ctypes.Structure):
   _fields_ = [("type", ctypes.c_ulong),
            ("ii", Input_I)]

# Actuals Functions

def PressKey(hexKeyCode):

   extra = ctypes.c_ulong(0)
   ii_ = Input_I()
   ii_.ki = KeyBdInput( hexKeyCode, 0x48, 0, 0, ctypes.pointer(extra) )
   x = Input( ctypes.c_ulong(1), ii_ )
   SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))

def ReleaseKey(hexKeyCode):

   extra = ctypes.c_ulong(0)
   ii_ = Input_I()
   ii_.ki = KeyBdInput( hexKeyCode, 0x48, 0x0002, 0, ctypes.pointer(extra) )
   x = Input( ctypes.c_ulong(1), ii_ )
   SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))

def getKeyforOutput(char):
   keycode = 'KEY_'+char
   PressKey(DictionaryofKeys[keycode])

def getKeyforRelease(char):
   keycode = 'KEY_'+char
   ReleaseKey(DictionaryofKeys[keycode])
```

Menu:

```
import pygubu
import tkinter as tk
from screenReader import ScreenReader
from QAlgorithm import QAlgorithm

builder  = pygubu.Builder()#loads Ui Builder
#
builder.add_from_file("FinalGUI.ui")#loads main menu from file
window = tk.Tk()#creates window object
window.mainwindow = builder.get_object("MainWindow")#copies the main frame
for ui in file
#this section creates instances of attibuites in Main windows
StartButton = builder.get_object("Start")
```

```
Pause = builder.get_object("Pause")
End = builder.get_object("End")
Reward = builder.get_object("Reward")#passed to q-learning class to update
current reward desplayed in menu
#DELETE LATER ALONG WITH PART IN GUI
ViewCurrent = builder.get_object("Current")
NewBBox = builder.get_object("Bounds")
EndGame = builder.get_object("EndGame")
Score = builder.get_object("Score")


sr = ScreenReader()#instanciates screen reader
#important game areas
scoreArea = None
GameArea = None
endGame = None
gameImage = None

bot = None #instanciates game playing bot
#starts the bot playing the game
def StartBot(e):
	bot =
QAlgorithm(Reward,GameArea,gameImage,endGame,scoreArea)#instanciates game
playing bot
	bot.state = 0
	bot.CheckState()
#saves a new area for main game play
def NewBox(e):
	global GameArea
	#load sup window or area selection
	#save array of bound boxs of interest
	if not (GameArea == None):
		x = GameArea[0]
		y = GameArea[1]
		h = abs(GameArea[0] -GameArea[2])
		w = abs(GameArea[1] -GameArea[3])
		GameArea = sr.selectArea('%dx%d+%d+%d' % (w, h, x, y))
	else:
		GameArea = sr.selectArea()
def ScoreArea(e):
	global scoreArea
	#load sup window or area selection
	#save array of bound boxs of interest
	if not (scoreArea == None):
		x = scoreArea[0]
```

```
                y = scoreArea[1]
                h = abs(scoreArea[0] -scoreArea[2])
                w = abs(scoreArea[1] -scoreArea[3])
                scoreArea = sr.selectArea('%dx%d+%d+%d' % (w, h, x, y))
        else:
                scoreArea = sr.selectArea()
#pauses bot execution
def PauseBot(e):
        bot.state = 1
#ends bot
def EndBot(e):
        bot.state = 2

#DELETE LATER ALONG WITH PART IN GUI
def ViewBox(e):
        if(GameArea == None):
                NewBox()
        x = GameArea[0]
        y = GameArea[1]
        h = abs(GameArea[0] -GameArea[2])
        w = abs(GameArea[1] -GameArea[3])
        sr.veiwArea('%dx%d+%d+%d' % (w, h, x, y))


#This captures the game over screen
def CaptureEnd(e):
        global gameImage
        #Thiscapures the end screen
        global endGame
        if not (endGame == None):
                x = endGame[0]
                y = endGame[1]
                h = abs(endGame[0] -endGame[2])
                w = abs(endGame[1] -endGame[3])
                endGame = sr.selectArea('%dx%d+%d+%d' % (w, h, x, y))
        else:
                endGame = sr.selectArea()
        gameImage = sr.grabArea(endGame)



#needs work not implmented right
#window.resizeable(0,0)#disables window resizing

#adds handlers to main attributes in main window
```

```
StartButton.bind("<ButtonPress-1>",StartBot)
#DELETE LATER ALONG WITH PART IN GUI
ViewCurrent.bind("<ButtonPress-1>",ViewBox)
NewBBox.bind("<ButtonPress-1>",NewBox)
Pause.bind("<ButtonPress-1>",PauseBot)
End.bind("<ButtonPress-1>",EndBot)
EndGame.bind("<ButtonPress-1>",CaptureEnd)
Score.bind("<ButtonPress-1>",ScoreArea)
#starts bot's main menu
window.mainloop()
```

Appendix E: User Manual

Introduction

This program is an AI bot that tries to learn how to play Tetris. It is only currently supported using windows and requires Python 3.5 and or higher, and the libraries imagehash, numpy, pillow, pybrain, pygubu, visual c++ redistributable x86, and scipy. Binaries for installation can be found in the install folder. The “whl” files must be installed by running “pip install (File of interest)”. All other dependencies, exluding the “exe”, must be installed by going into their root folder and running “python setup.py install”. The application menu in the root directory should be run to start the application.

Instructions

Before starting the program, make sure that the game is up to the playing stage so the bot does not crash. This bot does not go through menus, so go through them to get to the desired output. Would recommend the site http://tetris.com/play-tetris/ for the bot to play. To start the program, run the menu to bring up the GUI. Once the GUI is up, First Click the New Bounds Button and

a blank window should pop up. Resize, and put this window in the designed gameplay area. When the window is to the correct bounds, press enter. This window does not have to be perfect, but should at least get the the whole area where the blocks are. The area this window gets is from the top of the window where the label is, to whatever the area of the white space is, so adjust accordingly. After the game window is set, press score and another pop-up window should appear. Set this so it can be over the score, or lines depending on your preference. Both give different results, and when ready press enter again. Make sure that only ONE of the numbers are in the screen and no words, or errors might happen. The bot will crash as well if you do not get the screen to see the whole number, so make sure to have a little extra on the sides, and bottom. When both windows are ready, press start, and click on the game so the button presses go to the game and not somewhere else as this bot uses the system key commands for outputs. After that all you need to do is observe the bot play the game. Other features seen on the GUI are not finished, and do not need to be used in its current state.