AI Agent to play the game of Object vs. Blocks

Team Members:

16BCE0752 (R.Anuraag)
16BCE0944 (Akshay Arora)
16BCE0941 (Pallav Gupta)

Report submitted for the
Final Project Review of

Course Code: CSE3013 – Artificial Intelligence

Slot: B1 + TB1

Professor: Dr. Ilanthenral

1. Abstract

Artificial Intelligence is a field of computer science where a particular function is carried out by intelligent agents acting as humans. The incorporation of AI into basic chores of life are becoming more and more prevalent. For eg. An agent predicts the song you can listen to, based on your previous preferences.

The incorporation of artificial agents into commercial games has become a trend, the agents can enrich a players' experience. Also, it can lead to development of new real-time gaming strategies.

The objective of our project is to design an agent that can play a game of an object avoiding blocks in its way, by learning about the environment without the use of pre-set datasets.

The agents used in a gaming environment are broadly known as Game AI. This AI has the job of mostly focusing on what actions need to be taken by an entity or an object should take, based on the conditions. In this case, the main objective is controlling the 'intelligent agents' where the agent is usually a character in the game – but could also be a vehicle, a robot, or occasionally something more abstract such as a whole group of entities.

The game is already designed, but the complexity arises in rebuilding the environment and setting the various rules followed by the 'think' and 'act' implementations.

The various components of developing the project are:

1. Creating the objects or the characters of the game
2. Rebuilding the environment
3. Confirming the rules to which the agent must be constrained to
4. Deciding the algorithm/model to be used
5. Building a learning agent, one that learns on its own
6. Using Reinforcement Learning and Deep Learning

In our proposed game, the object/entity needs to cross obstacles of different kind and keep on moving forward. The game ends when the object comes in contact with any obstacle in its path.

An example scenario:

A frog (entity) needs to cross certain cars (obstacles) running on the road.

Keywords: Interactive Reinforcement Learning; Deep Learning; Cognitive Systems; Game AI; TensorFlow ; Self-Learning Agent

2. Introduction

Artificial Intelligence consists of an agent acting as a human being.

Each AI agent works on three bases and in each case, the agent that needs to observe its surroundings, make decisions based on the same and implement those decision. This is simply known as Sense/Think/Act cycle:

- Sense: The agent detects parameters in the environment
- Think: The agent makes a decision about what to do in response
- Act: The agent performs actions to put the previous decision into motion

All these methodologies are useful in implementing or designing an intelligent agent.

For our project the most important methodology is the one of the Learning AI. The learning AI uses reinforcement learning and deep learning methods to complete the original objective.

Let us introduce the topics mentioned in the above statement one-by –one.

1. Learning Agent

A learning agent is a tool in AI that is capable of learning from its experiences. It starts with some basic knowledge and is then able to act and adapt autonomously,

2. Reinforcement Learning

Reinforcement learning (RL) is a behaviour-based approach which allows an agent, either an infant or a robot, to learn a task by interacting with its environment and observing how the environment responds to the agent's actions

To apply this on an artificial agent, you have a kind of a feedback loop to reinforce your agent. It rewards when the actions performed is right and punishes in-case it was wrong. Basically what you have in your kitty is:

- an **internal state**, which is maintained by the agent to learn about the environment
- a **reward function,** which is used to train your agent how to behave
- an **environment**, which is a scenario the agent has to face
- an **action**, which is done by the agent in the environment
- an **agent** which competes all his objectives

Reinforcement learning consists of a reward system that is assigned when a single iteration of the final objective is completed. The reward sends the agent in the right direction

3. Deep Q-Learning

Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms.

3. Literature Review Summary Table

Authors and Year (Reference)	***Title (Study)***	***Concept / Theoretical model/ Framework***	***Methodology used/ Implementation***	***Dataset details/ Analysis***	***Relevant Finding***	***Limitations/ Future Research/ Gaps identified***
Meng Xu ; Haobin Shi ; Yao Wang (2018)	Play games using Reinforcement Learning	Reinforcement Learning and its application	Use of Iterative Reinforcement Learning	The project gave an accurate outcome after 100 iterations	Self-Learning agent is possible	Algorithm decided using trial and error method
Daniel Fu and Ryan Houlett (2014)	Putting AI in Entertainment	Uses of AI algorithms in gaming	Decision Tree; Bayes	Gave the current trend of AI	Theoretical implications	
Xiangguang ; He Yaya Wang(2014)	Researching on AI Path-finding Algorithm in the Game Development	Path-Finding Algorithms	BFS,DFS and A* search	Implemented the algorithms on the Snakes game	Short BFS is preferred	Time consuming
Frank Dignum et al.(2009)	Games and agents: designing intelligent Gameplay	Use of AI is designing games	Done using deep learning	Agents used as character in the game	Improves the gameplay for the users	May allow the AI to control the game
Francisco Cruz ; Sven Magg ;(2017)	Training Agents Interactive Reinforcement Learning	The efficient of IRL	IRL leads to different performance efficiencies	Uses a robot on which IRL is applied for it to respond	Accurate response	Further advanced robot to be developed
Ruben Rodriguz Torrado ; Philip Bontrager (2016)	Deep Reinforcement Learning for General Video Game AI	Checking the current status of an agent	OpenAI Gym used over Keras	Using OpenGym we can classify reinforcement algorithms	OpenGym is used with TensorFLow and keras	Did not have any practical experiments
Mauro Komi (2017)	Play games using Reinforcement Learning and Artificial Neural Network	Self-Learning agent development	Using OpenGym AI, keras and FSM	FSMs are effective for learning the current state of the system	FSMs are utilized for designing agents for a game	No practical working shown

4. Innovation Component in the project

The most innovative component of the project is the implementation of the UI (User Interface) using **pygame and pymunk** (which are python libraries).

'pymunk' can basically be defined as a physics based library. So, pymunk basically allows components designed in pygame to be influenced by the concepts of physics.

For example, if we create a circle in pygame, next using pymunk we can define its working according to physics with concepts like:

1. Moment of Inertia

2. Rotation

3. Direction defined by Vec2D

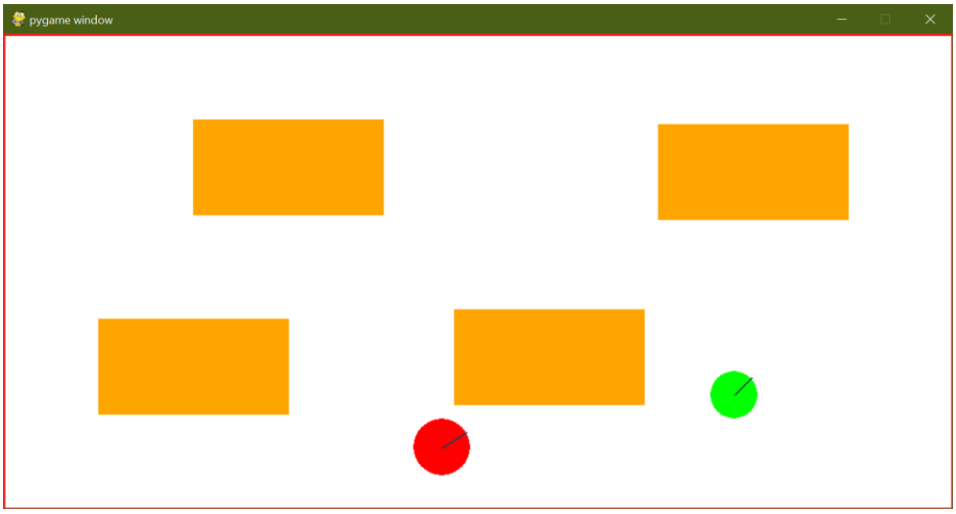
4. Collisions

There is a major difference between the results found by our approach and by various different approaches. The basic approach applied in all research papers, is to use various other tools for designing the environment to apply the Game AI on, the tools can be

- Predefined Game AI
- Unity
- OpenGym AI Library, has predefined games to apply learning on.

But, our approach uses 'pymunk' which is an approach that can be used to design scenarios using python.

The UI of our project is as follows:



5. Work done and implementation

The implementation vaguely consists of two major parts:

- The designing of the UI of the game
  - This step uses pygame and pymunk libraries on python
- Next, Training the agent by implementing Reinforcement Learning and creating a Neural Network using Keras
  - Keras and TensorFlow are used to create a Neural network with the help of simple commands

5.1. Methodology adapted:

Our objective is to make a Learning AI that can play a game of dodging obstacles by teaching itself to dodge the obstacles with the help of a neural network.

The algorithms that are proposed to be used are:

1. **Reinforcement Learning**: We decide reward values for various events that the agent performs, calculate the total reward and then take the model that gave us the maximum reward value.
2. **Deep Learning**: Building a deep Neural Network architecture.

These algorithms can be implemented using libraries such as Keras or TensorFlow

- **Tensorflow**: It is a symbolic math library, and is also used for machine learning applications such as neural networks.
- **Keras**: It is a library used for machine learning, build on top of TensorFlow to provide an easier command interface

Implementation

The steps of execution are as follows:

1. Building the objects need in the environment with pymunk and pygame
2. Designing the physics and movement of the of the objects using libraries form pymunk in python
3. Setting the rules/constraints for the game and the result of events that could happen.
4. After environment is made, start the training, we design a deep Neural network using Keras over TensorFLow.
5. Use the concept of learning, where the object learns from its surroundings one iteration at a time
6. We train the agent, in the environment for 100000 epochs, iterations with falling epsilon value, we train until we get a minimum epsilon value.
7. We divide the iteration into 4 parts, then we choose the part which contains the iteration with the maximum reward.
8. Finally, the agent starts dodging the objects

5.2. Hardware and software requirements:

Hardware Requirements:

- Graphic Card upto 2GB
- Minimum RAM: 2GB

Software Requirements:

- Google's TensorFlow
- Keras
- Python Script 3.6

5.3. Dataset used / Tools used:

a. Where from you are taking your dataset?

No dataset used, the main objective of the project is to build a learning agent that learns without the presence of a preset dataset. Use of Keras commands

b. Is your project based on any other reference project (Stanford Univ. or MIT)?

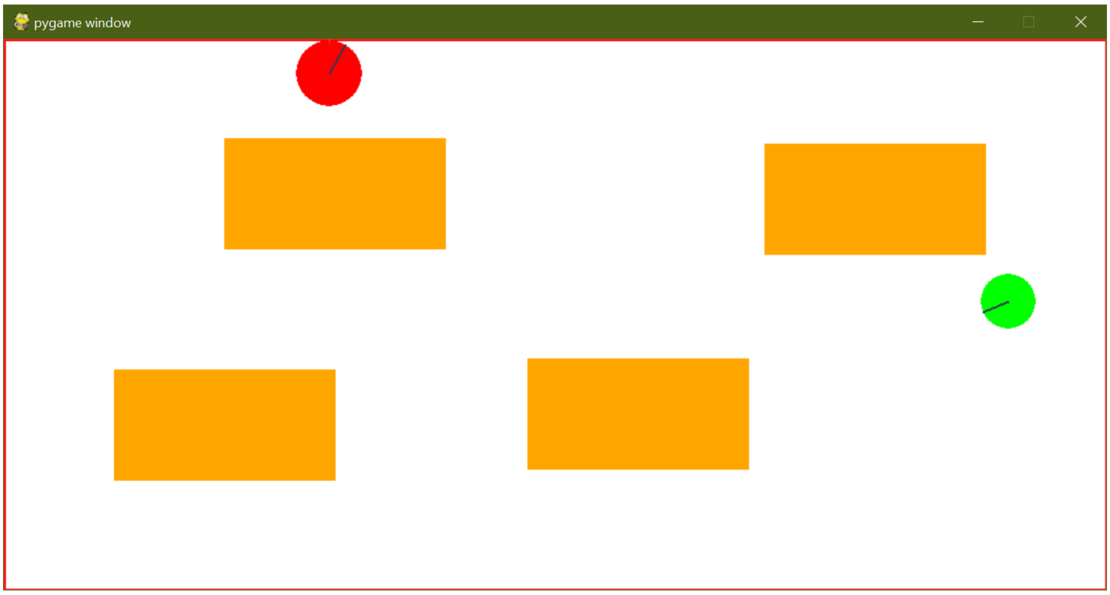
The project is based on the implementation of an agent that can play the DinoRun game. Present on the list of Stanford projects.

c. How does your project differ from the reference project?*

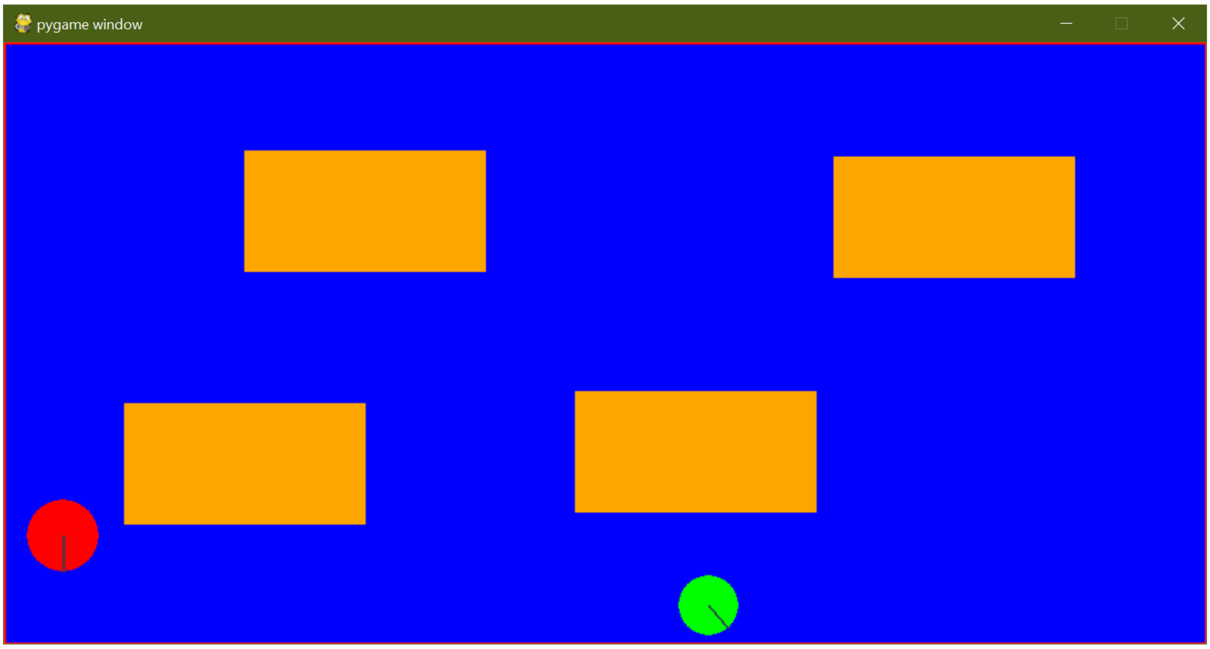
Our plan is to design an environment for a game by ourselves, and train an AI agent for the same

5.4. Screenshot and Demo:

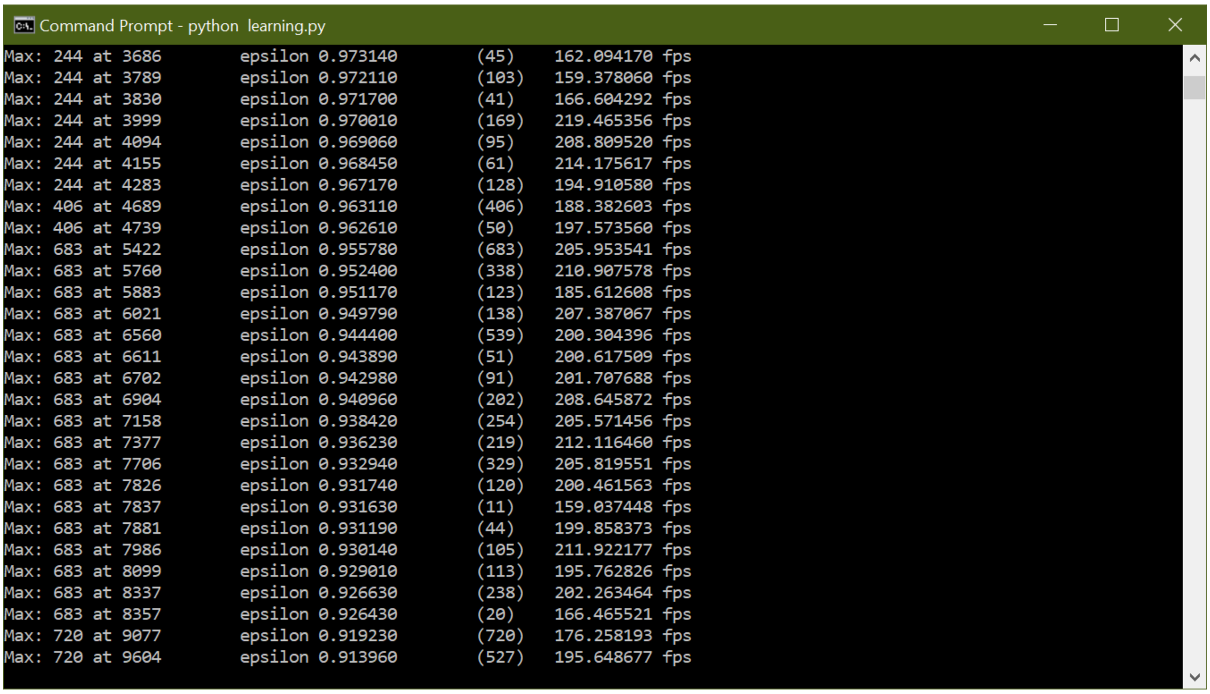
1. The UI of the game, the objects are designed using pymunk and pygame



2. When the object (AI) collides with the obstacles (orange) or with another moving object (red), the screen gives a blue tint



3. The training process, there are 100000 iterations, the reward is calculated for each iteration implying reinforcement learning, then the maximum reward iteration is stored.




```
Max: 244 at 3686        epsilon 0.973140        (45)    162.094170 fps
Max: 244 at 3789        epsilon 0.972110        (103)   159.378060 fps
Max: 244 at 3830        epsilon 0.971700        (41)    166.604292 fps
Max: 244 at 3999        epsilon 0.970010        (169)   219.465356 fps
Max: 244 at 4094        epsilon 0.969060        (95)    208.809520 fps
Max: 244 at 4155        epsilon 0.968450        (61)    214.175617 fps
Max: 244 at 4283        epsilon 0.967170        (128)   194.910580 fps
Max: 406 at 4689        epsilon 0.963110        (406)   188.382603 fps
Max: 406 at 4739        epsilon 0.962610        (50)    197.573560 fps
Max: 683 at 5422        epsilon 0.955780        (683)   205.953541 fps
Max: 683 at 5760        epsilon 0.952400        (338)   210.907578 fps
Max: 683 at 5883        epsilon 0.951170        (123)   185.612608 fps
Max: 683 at 6021        epsilon 0.949790        (138)   207.387067 fps
Max: 683 at 6560        epsilon 0.944400        (539)   200.304396 fps
Max: 683 at 6611        epsilon 0.943890        (51)    200.617509 fps
Max: 683 at 6702        epsilon 0.942980        (91)    201.707688 fps
Max: 683 at 6904        epsilon 0.940960        (202)   208.645872 fps
Max: 683 at 7158        epsilon 0.938420        (254)   205.571456 fps
Max: 683 at 7377        epsilon 0.936230        (219)   212.116460 fps
Max: 683 at 7706        epsilon 0.932940        (329)   205.819551 fps
Max: 683 at 7826        epsilon 0.931740        (120)   200.461563 fps
Max: 683 at 7837        epsilon 0.931630        (11)    159.037448 fps
Max: 683 at 7881        epsilon 0.931190        (44)    199.858373 fps
Max: 683 at 7986        epsilon 0.930140        (105)   211.922177 fps
Max: 683 at 8099        epsilon 0.929010        (113)   195.762826 fps
Max: 683 at 8337        epsilon 0.926630        (238)   202.263464 fps
Max: 683 at 8357        epsilon 0.926430        (20)    166.465521 fps
Max: 720 at 9077        epsilon 0.919230        (720)   176.258193 fps
Max: 720 at 9604        epsilon 0.913960        (527)   195.648677 fps
```

5.5. Code

UI.py

```
import random
import math
import numpy as np

import pygame
from pygame.color import THECOLORS

import pymunk
from pymunk.vec2d import Vec2d
from pymunk.pygame_util import
DrawOptions

# PyGame init
width = 1000
height = 500
pygame.init()
screen = pygame.display.set_mode((width,
height))
clock = pygame.time.Clock()

# Turn off alpha since we don't use it.
screen.set_alpha(None)

# Showing sensors and redrawing slows things
down.
show_sensors = True
draw_screen = True


class GameState:
    def __init__(self):
        # Global-ish.
        self.crashed = False

        # Physics stuff.
        self.space = pymunk.Space()
        self.space.gravity = pymunk.Vec2d(0., 0.)
        self.draw_options =
pymunk.pygame_util.DrawOptions(screen)

        # Create the car.
        self.create_car(100, 100, 0.5)

        # Record steps.
        self.num_steps = 0

        # Create walls.
        # static=
[pymunk.Segment(self.space.static_body, (50,
50), (50, 550), 5)
        #
,pymunk.Segment(self.space.static_body, (50,
550), (650, 550), 5)
        #
,pymunk.Segment(self.space.static_body,
(650, 550), (650, 50), 5)
        #
,pymunk.Segment(self.space.static_body, (50,
50), (650, 50), 5)
        #    ]
        static = [
            pymunk.Segment(
                self.space.static_body,
                (0, 1), (0, height), 1),
            pymunk.Segment(
                self.space.static_body,
                (1, height), (width, height), 1),
            pymunk.Segment(
                self.space.static_body,
                (width-1, height), (width-1, 1), 1),
            pymunk.Segment(
                self.space.static_body,
                (1, 1), (width, 1), 1)
        ]
        for s in static:
            s.friction = 1.
            s.group = 1
            s.collision_type = 1
            s.color = THECOLORS['red']
        self.space.add(static)

        self.obstacles = []

self.obstacles.append(self.create_obstacle(30
0, 360, 100))

self.obstacles.append(self.create_obstacle(79
0, 355, 125))

self.obstacles.append(self.create_obstacle(57
5, 160, 35))

self.obstacles.append(self.create_obstacle(20
0, 150, 35))
```

```
        # Create a cat.
        self.create_cat()

    def create_obstacle(self, x, y, r):
        c_body =
pymunk.Body(body_type=pymunk.Body.STATI
C)
        c_shape =
pymunk.Poly.create_box(c_body,(200,100,r))
        c_shape.elasticity = 1.0
        c_body.position = x, y
        c_shape.color = THECOLORS["orange"]
        self.space.add(c_body, c_shape)
        return c_body

    def create_cat(self):
        inertia = pymunk.moment_for_circle(1, 0,
14, (0, 0))
        self.cat_body = pymunk.Body(1, inertia)
        self.cat_body.position = 50, height - 100
        self.cat_shape =
pymunk.Circle(self.cat_body, 30)
        self.cat_shape.color = THECOLORS["red"]
        self.cat_shape.elasticity = 1.0
        self.cat_shape.angle = 0.5
        direction = Vec2d(1,
0).rotated(self.cat_body.angle)
        self.space.add(self.cat_body,
self.cat_shape)

    def create_car(self, x, y, r):
        inertia = pymunk.moment_for_circle(1, 0,
14, (0, 0))
        self.car_body = pymunk.Body(1, inertia)
        self.car_body.position = x, y
        self.car_shape =
pymunk.Circle(self.car_body, 25)
        self.car_shape.color =
THECOLORS["green"]
        self.car_shape.elasticity = 1.0
        self.car_body.angle = r
        driving_direction = Vec2d(1,
0).rotated(self.car_body.angle)

self.car_body.apply_impulse_at_local_point(d
riving_direction)
        self.space.add(self.car_body,
self.car_shape)

    def frame_step(self, action):
        if action == 0:
            self.car_body.angle -= .2
        elif action == 1:
            self.car_body.angle += .2

        # Move obstacles.
        if self.num_steps % 100 == 0:
            self.move_obstacles()

        # Move cat.
        if self.num_steps % 5 == 0:
            self.move_cat()

        driving_direction = Vec2d(1,
0).rotated(self.car_body.angle)
        self.car_body.velocity = 10 *
driving_direction

        # Update the screen and stuff.
        screen.fill(THECOLORS["white"])

self.space.debug_draw(self.draw_options)
        self.space.step(1./10)
        if draw_screen:
            pygame.display.flip()
        clock.tick()

        # Get the current location and the
readings there.
        x, y = self.car_body.position
        readings = self.get_sonar_readings(x, y,
self.car_body.angle)
        normalized_readings = [(x-20.0)/20.0 for
x in readings]
        state = np.array([normalized_readings])

        # Set the reward.
        # Car crashed when any reading == 1
        if self.car_is_crashed(readings):
            self.crashed = True
            reward = -500

self.recover_from_crash(driving_direction)
        else:
            # Higher readings are better, so return
the sum.
            reward = -5 +
int(self.sum_readings(readings) / 10)
        self.num_steps += 1

        return reward, state
```

```
    def move_obstacles(self):
        # Randomly move obstacles around.
        for obstacle in self.obstacles:
            speed = random.randint(20, 50)
            direction = Vec2d(1,
0).rotated(self.car_body.angle +
random.randint(-2, 2))
            obstacle.velocity = speed * direction

    def move_cat(self):
        speed = random.randint(10, 30)
        self.cat_body.angle -= random.randint(-1,
1)
        direction = Vec2d(1,
0).rotated(self.cat_body.angle)
        self.cat_body.velocity = speed * direction

    def car_is_crashed(self, readings):
        if readings[0] == 1 or readings[1] == 1 or
readings[2] == 1:
            return True
        else:
            return False

    def recover_from_crash(self,
driving_direction):
        while self.crashed:
            # Go backwards.
            self.car_body.velocity = -10 *
driving_direction
            self.crashed = False
            for i in range(10):
                self.car_body.angle += .2
                screen.fill(THECOLORS["blue"])

self.space.debug_draw(self.draw_options)
                self.space.step(1./10)
                if draw_screen:
                    pygame.display.flip()
                clock.tick()

    def sum_readings(self, readings):
        """Sum the number of non-zero
readings."""
        tot = 0
        for i in readings:
            tot += i
        return tot

    def get_sonar_readings(self, x, y, angle):
        readings = []

        # Make our arms.
        arm_left = self.make_sonar_arm(x, y)
        arm_middle = arm_left
        arm_right = arm_left
        # Rotate them and get readings.

readings.append(self.get_arm_distance(arm_l
eft, x, y, angle, 0.75))

readings.append(self.get_arm_distance(arm_
middle, x, y, angle, 0))

readings.append(self.get_arm_distance(arm_r
ight, x, y, angle, -0.75))

        if show_sensors:
            pygame.display.update()

        return readings

    def get_arm_distance(self, arm, x, y, angle,
offset):
        # Used to count the distance.
        i = 0

        for point in arm:
            i += 1

            rotated_p = self.get_rotated_point(
                x, y, point[0], point[1], angle + offset
            )

            if rotated_p[0] <= 0 or rotated_p[1] <=
0 \
                    or rotated_p[0] >= width or
rotated_p[1] >= height:
                return i  # Sensor is off the screen.
            else:
                obs = screen.get_at(rotated_p)
                if self.get_track_or_not(obs) != 0:
                    return i

            if show_sensors:
                pygame.draw.circle(screen, (255,
255, 255), (rotated_p), 2)

        # Return the distance for the arm.
        return i
```

```
    def make_sonar_arm(self, x, y):
        spread = 10
        distance = 20  # Gap before first sensor.
        arm_points = []
        # Make an arm. We build it flat because
we'll rotate it about the
        # center later.
        for i in range(1, 40):
            arm_points.append((distance + x +
(spread * i), y))

        return arm_points

    def get_rotated_point(self, x_1, y_1, x_2,
y_2, radians):
        # Rotate x_2, y_2 around x_1, y_1 by
angle.
        x_change = (x_2 - x_1) *
math.cos(radians) + \
            (y_2 - y_1) * math.sin(radians)
        y_change = (y_1 - y_2) *
math.cos(radians) - \
            (x_1 - x_2) * math.sin(radians)
        new_x = x_change + x_1
        new_y = height - (y_change + y_1)
        return int(new_x), int(new_y)

    def get_track_or_not(self, reading):
        if reading == THECOLORS['white']:
            return 0
        else:
            return 1

if __name__ == "__main__":
    game_state = GameState()
    running=True
    while running:

game_state.frame_step((random.randint(0,
2)))
        for event in pygame.event.get():
            if event.type ==
pygame.MOUSEBUTTONDOWN:
                running = True
            if event.type == pygame.QUIT or
event.type == pygame.KEYDOWN:
                running = False
```

nn.py

```
from keras.models import Sequential

from keras.layers.core import Dense,
Activation, Dropout
from keras.optimizers import RMSprop
from keras.layers.recurrent import LSTM
from keras.callbacks import Callback


class LossHistory(Callback):
    def on_train_begin(self, logs={}):
        self.losses = []

    def on_batch_end(self, batch, logs={}):
        self.losses.append(logs.get('loss'))


def neural_net(num_sensors, params,
load=''):
    model = Sequential()

    # First layer.
    model.add(Dense(
        params[0], init='lecun_uniform',
input_shape=(num_sensors,)
    ))
    model.add(Activation('relu'))
    model.add(Dropout(0.2))

    # Second layer.
    model.add(Dense(params[1],
init='lecun_uniform'))
    model.add(Activation('relu'))
    model.add(Dropout(0.2))

    # Output layer.
    model.add(Dense(3, init='lecun_uniform'))
    model.add(Activation('linear'))

    rms = RMSprop()
    model.compile(loss='mse', optimizer=rms)

    if load:
        model.load_weights(load)

    return model
```

6. Results and Discussion

Our objective was to finally design an AI agent that can dodge obstacles and traverse the environment that we created without any external interference. We can conclude by saying that we were successful in doing so.

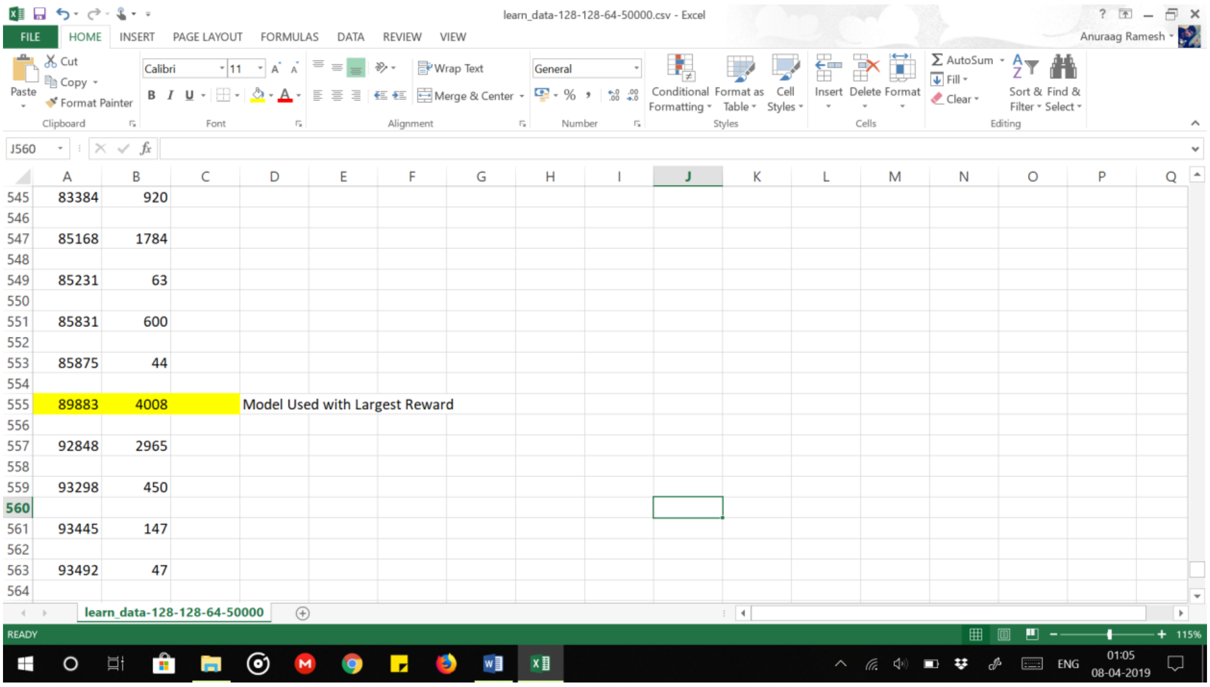
Our final result was an AI Game agent trained using Deep Reinforcement Learning, that was successful in dodging obstacles in its way, and traverses the environment.

We train the agent i.e. the green object, according to our environment, during the training we get the reward values for each iteration. This training is done using the ‘learning.py’ and nn.py’ files. The main function that does the training is

train_net(model,params).

where train_net is function; model is the training mould; params are the constraints defined

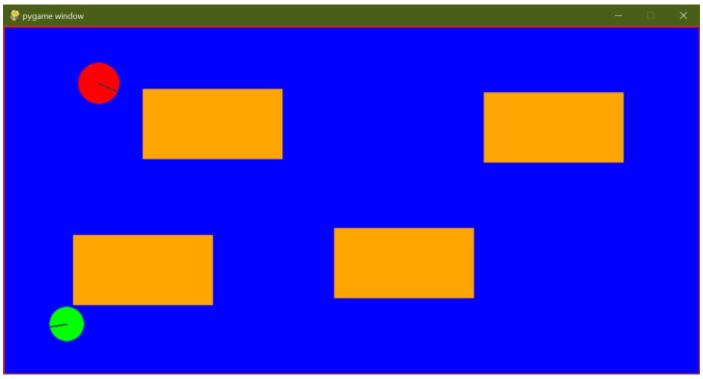
In the image below, we can highlight the iteration with the highest value.



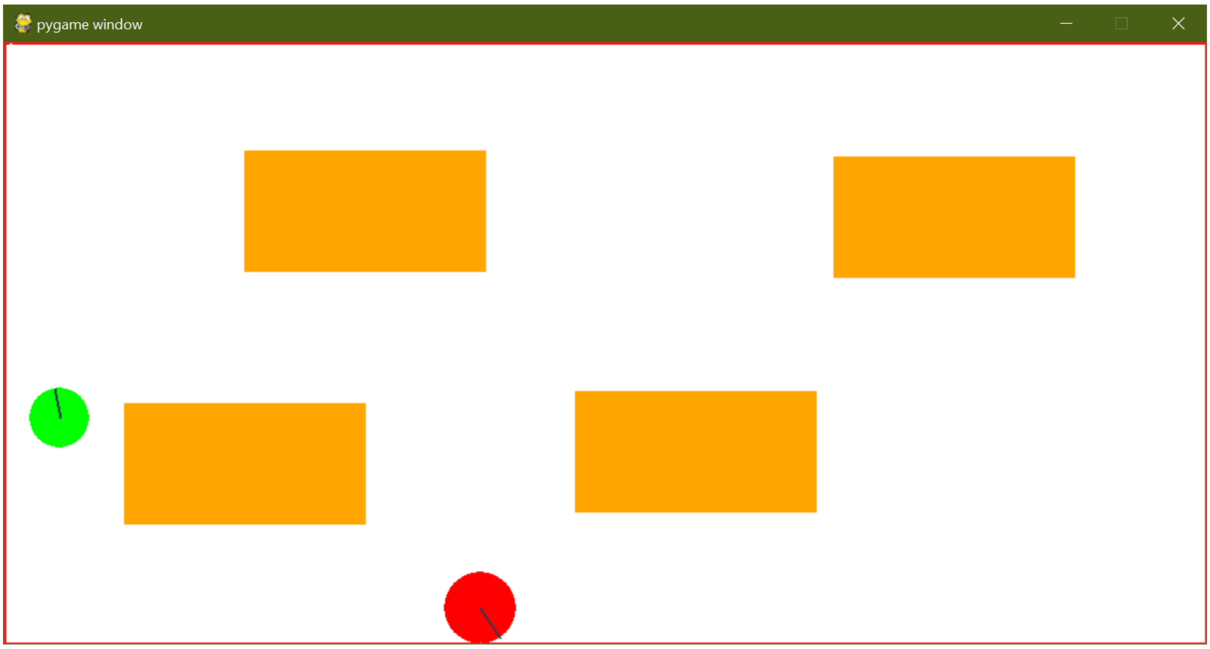
After, creating the models in training and saving them in a folder, as four batches/ sets.

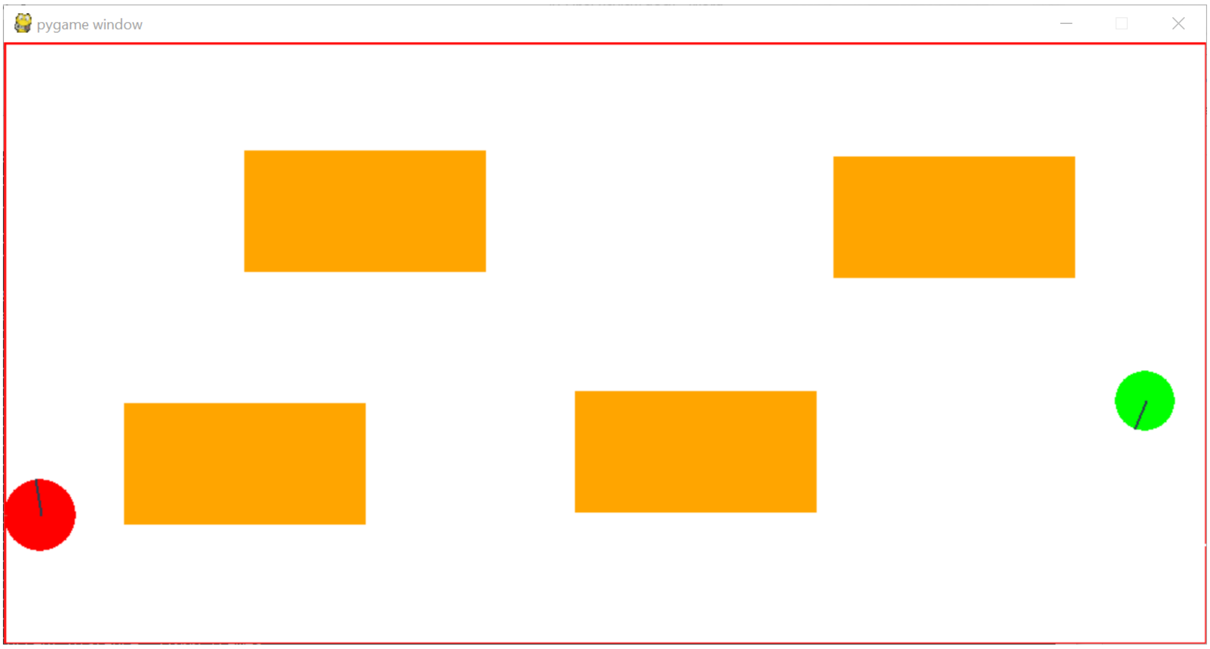
We run the ‘playing.py’ file which runs the environment using the model we designed using the training process. Hence getting the final output where the agent (green object) has learnt to dodge the obstacles.

We have placed a blue screen as an indicator for collisions.



Movement of the green object without colliding, form one end to another.





7. References

1. R. S., & Barto, A. G. *Reinforcement learning: An Introduction*. Cambridge, MA: Bradford Book.

2. http://web.stanford.edu/class/cs221/2018/restricted/posters/nwrubin/poster.pdf

3. Artificial Intelligence and Games; Georgios N. Yannakakis and Julian Togelius

4. An approach to interactive deep reinforcement learning for serious games: Aline Dobrovsky; Uwe M. Borghoff; Marko Hofmann

5. https://keras.io/

6.https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning

7. Distributed Deep Reinforcement Learning using TensorFlow; Ajay Rao Department of Computer Science & Engineering, RVCE, Bengaluru – 59; Navaneesh Kumar B

8. http://outlace.com - On Deep Tensor Networks and the Nature of Non-Linearity