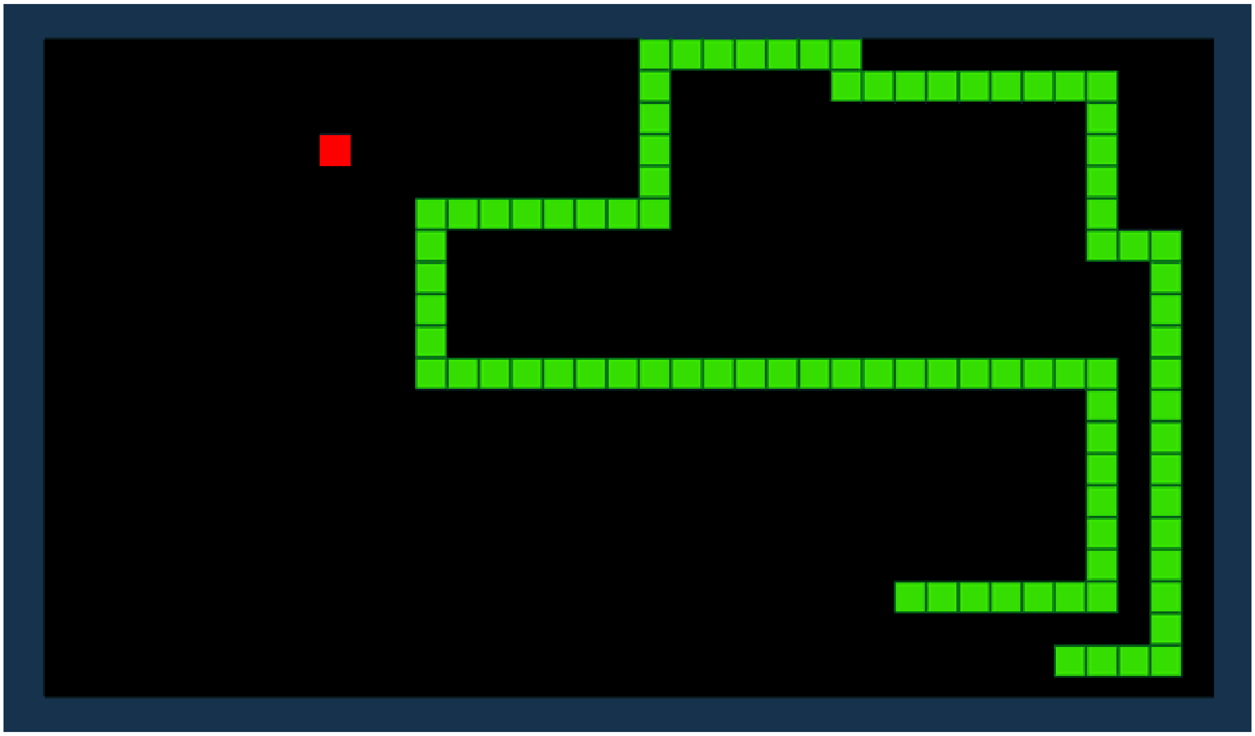
Reinforcement Learning (Deep Q)

Teaching AI to Play Snake



Intuition

Reinforcment learning(RL) is an area of machine learning concerned with how software agents should take actions in an enviroment in order to maximize the notion of cumulative reward.

Steps

```
1. Initialize your Q-table
2. Choose an action using the Epsilon-Greedy Exploration Strategy
3. Update the Q-table using the Bellman Equation
```

Necessary Formulas

Bellman Equation :

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma maxQ'(s',a') - Q(s,a)]$$

Q Update Rule Simplified :

$$Q = model.predict(state_0)$$
$$Q_{new} = R + \gamma \times max(Q(state_1))$$

$$Q_{new} = R + \gamma * max(Q(state1))$$

Loss Function :

$$loss = (Q_{new} - Q)^2$$

In [ ]:

```python
# Packages
import pygame
import time
import random
from enum import Enum
from collections import namedtuple

# Game
pygame.init()

white = (255, 255, 255)
red = (200, 0, 0)
blue1 = (0, 0, 255)
blue2 = (0, 100, 255)
green = (0, 255, 0)
blue = (50, 153, 213)

width = 600
height = 400

screen = pygame.display.set_mode((width, height))

clock = pygame.time.Clock()

snake_block = 20
snake_speed = 20

Point = namedtuple('Point', 'x, y')

def Your_score(score):
    value = score_font.render("Your Score: " + str(score), True, red)
    screen.blit(value, [0, 0])



def our_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(screen, blue1, [x[0], x[1], snake_block, snak


def message(msg, color):
    mesg = font_style.render(msg, True, color)
    screen.blit(mesg, [width / 6, height / 3])


def gameLoop():
    game_over = False
    game_close = False

    x1 = width / 2
    y1 = height / 2

    x1_change = 0
    y1_change = 0

    snake_List = []
    Length_of_snake = 1

    foodx = round(random.randrange(0, width - snake_block) / 10.0) * 1
    foody = round(random.randrange(0, height - snake_block) / 10.0) *

    while not game_over:

        while game_close == True:
            screen.fill(blue)
            message("You Lost! Press C-Play Again or Q-Quit", blue2)
            Your_score(Length_of_snake - 1)
            pygame.display.update()

            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        game_over = True
                        game_close = False
                    if event.key == pygame.K_c:
                        gameLoop()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                game_over = True
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    x1_change = -snake_block
                    y1_change = 0
                elif event.key == pygame.K_RIGHT:
                    x1_change = snake_block
                    y1_change = 0
                elif event.key == pygame.K_UP:
                    y1_change = -snake_block
                    x1_change = 0
                elif event.key == pygame.K_DOWN:
                    y1_change = snake_block
                    x1_change = 0

        if x1 >= width or x1 < 0 or y1 >= height or y1 < 0:
            game_close = True
        x1 += x1_change
        y1 += y1_change
        screen.fill(blue)
        pygame.draw.rect(screen, green, [foodx, foody, snake_block, sn
        snake_Head = []
        snake_Head.append(x1)
        snake_Head.append(y1)
        snake_List.append(snake_Head)
        if len(snake_List) > Length_of_snake:
            del snake_List[0]

        for x in snake_List[:-1]:
            if x == snake_Head:
                game_close = True

        our_snake(snake_block, snake_List)
        Your_score(Length_of_snake - 1)

        pygame.display.update()

        if x1 == foodx and y1 == foody:
            foodx = round(random.randrange(0, width - snake_block) / 1
            foody = round(random.randrange(0, height - snake_block) /
            Length_of_snake += 1

        clock.tick(snake_speed)

    pygame.quit()
    quit()


gameLoop()
```

In [ ]: