Project Report

Introduction to computing using python

Moe Myint Myat (1155165771)
Rasmus Høgh Gregersen (1155190871)

June 24, 2023

1 Snake!

In the final programming project of 1100 AIST Introduction to Computing Using python, our group has made our own version of the classic Snake game. In the game, a player controls a snake slithering it's way around a board, consuming food objects that respawn upon consumption, regenerating in a different spot in the map. The goal is to eat as many pieces of food without dying.
The game is fairly well known, and can be divided into following possible actions:

Change direction of snake: The snake always moves a set speed in a set direction - the player has the possibility to change the direction of the snake.

Eat food: The food is randomly generated whenever eaten, and is consumed by the snakes first index (the mouth/head) landing on the coordinate of the food. Consuming this food allows the snake to grow in length, and the score of the game is incremented.

Eat the tail: In case the head of the snake hits any part of its body, the snake dies.

Hit the wall: In case the head of the snake hits the walll, it dies.

The game is designed with the Snake being its own class, and the food being its own class. These are used in the class SnakeGame, which inherits the Gym subclass, and has functions for main game and menu - beyond the gym functions implemented.

1.1 Classes

1.1.1 Snake

The snake class:

```
1    class Snake:
2
3    #Possible directions of movement
4    DIRECTIONS={"UP":(0,-1),"DOWN":(0,1),"LEFT":(-1,0),"RIGHT":(1,0)}
5
6    #Lenght (sausage links haha) of snake and position of body
7    links=None
8    body=None
9
10   #Direction of snake
11   direction=None
12
13   #Size of snake parts
14   pixel_size=None
15
16   #Color of snake
17   color=(0,0,255)
18   borders=None
```

Has attributes describing the current/allowed length of the snake, the position of it's body parts, the direction of the snake, and the pixel size and borders it shares with the game and the food.
The snake class __init__ function intializes the snake object with the given borders and pixelsize. Also it calls the respawn function, where the rest of the snake is defined

```
    #Iinitialize new snake
    def __init__(self, pixel_size, borders):
        self.pixel_size=pixel_size
        self.borders=borders
        self.died()

    #respawn function, three long
    def died(self):
        self.links=3
        s=self.pixel_size
        self.body=[(s,s),(2*s,s),(3*s,s)]
        self.direction=self.DIRECTIONS["RIGHT"]
```

The snake has a move function, where the head is placed on the next square in the direction of the movement, and if there is no food on the square, the tail is removed - creating the illusion of movement. The control function changes the direction of the snake. Furthermore functions are created that are controlled every loop of the game, whether the snake has hit it's own tail, whether it's hit the boundaries, or whether it has hit the food - hitting the food increments the links attribute, meaning the "tail" isn't removed, allowing the snail to increase in size.

1.1.2 Food

The food class:

```
    class Food(Object):

    def __init__(self,pixel_size,borders):
        super().__init__(pixel_size,borders)
        self.color=(255,0,0)

    def eaten(self,snake):
        x_pixels=(self.borders[0])/self.pixel_size
        y_pixels=(self.borders[1])/self.pixel_size
        self.x=random.randint(0,x_pixels-1)*self.pixel_size
        self.y=random.randint(0,y_pixels-1)*self.pixel_size
        while (self.x,self.y) in snake.body:
            self.x=random.randint(0,x_pixels-1)*self.pixel_size
            self.y=random.randint(0,y_pixels-1)*self.pixel_size
```

Has pixel size and border attributes shared with game and snake. If eaten, the food gets a different location, that is within the confines of the game, but not in the snake.

1.1.3 Snake Game

The snake game class imports snake and food. Initializes the food and the snake, and can be timed, and have different difficulties (For human player - speed changes with difficulty). There are two menu functions implemented for use with human player - first menu lets you exit game or start a new game, if start a new game is selected, the following menu function is called. This function allows you to choose difficulty and timed mode, and start the new game. Once the snake dies, you get the option to go back to main menu, or start again.

2 Q learning

2.1 Description of the q table

First, we identify the states the snake is in using its position relative to the food and the future states. The relative position includes the food being above or below the snake or on the same horizontal level as the snake, and being at the left or the right to the snake or on the same vertical level as the snake. The future states are defined by the states the snake will achieve by moving one step in any possible directions (up,down,left,right).
In terms of the relative position to the food, we vectorized the right state and the above state as 1, the left state and the below state as 0, and the same horizontal level and the same vertical state as '-'.
For the future states, we check the conditions whether the snake hits the vertical borders or the horizontal borders or the snake itself. If the conditions are true, we identify with "1", but "0" otherwise. Since there are four digits, we categorized the states into a string of 4 digits.
The states serve as a key in the Q table and the values are the scores on the actions taken in each state. Since there are 4 possible directions to make, there will be a list of 4 scores.

2.2 How the Key of the AI Player is chosen

Key here refers to the direction key.
The key of the computer player is chosen based on the values of the Q table but to introduce some randomness and explore, we will create one-tenth probability of randomly choosing a key in a given state. We set an epsilon limit of 0.1 and generate a random epsilon (within 0 and 1). When the random epsilon is smaller than the epsilon limit, the key will be chosen randomly but otherwise, the key will be chosen from the Q table. After 100 games, the epsilon is reduced to 0 to prioritize the learnt values in choosing the key. Given the state, the key with the maximum score will be chosen as the key and the move will be made accordingly.

2.3 Reward Policy

According to the game, when the snake hits the borders or hits itself, it dies. Given any state, if the snake dies, the action (key) that the player made during the state is given a negative reward (-1). Similarly, when the snake goes further from the food, it is a negative reward. In contrast, when the snake goes nearer to the food or ate the food, it is a positive reward.

2.4 Calculation of Score Depending on Reward

The score is calculated using the Bellman equation. It calculates the expected return of an action.

```
    Score of the Action of Previous State = (1-learning_rate) * (previous_state_action) +
learning_rate * (reward + discount*max(current_state_action) # Bellman equation
```

2.5 Choice of Parameters in score calculation

The learning rate refers to how much the new value is accepted as the old value. The discount rate refers to how much we prioritize the long term reward over the short term reward. Small discount rate indicates that we prioritize the short term reward.

Chosen Discount: 0.2

Chosen learning rate: 0.5

Chosen epsilong limit: 0.1

The current default parameters are chosen based on experimentation. Currently, after 100 games, the highest scores become stagnant although the total scores over each consecutive 50-game span mostly increases. When we lower the discount rate, the highest score over each 50-game span achieves the highest in comparison to when we increase the discount rate. According to experimentation,when the discount rate is higher, the highest score over each 50-game span is not as high as the highest score of the experimentation with a smaller discount rate.

2.6 Score Plots

We have chosen to report on the total scores and the highest score over each 50-game span. From observation, we can see that the total scores and the highest score always increases until 150 games. However, it becomes stagnant after 150 although the total score occasionally achieves higher than the previous total scores. The total 50-game span scores after 150 games are always higher than those before 150 games.

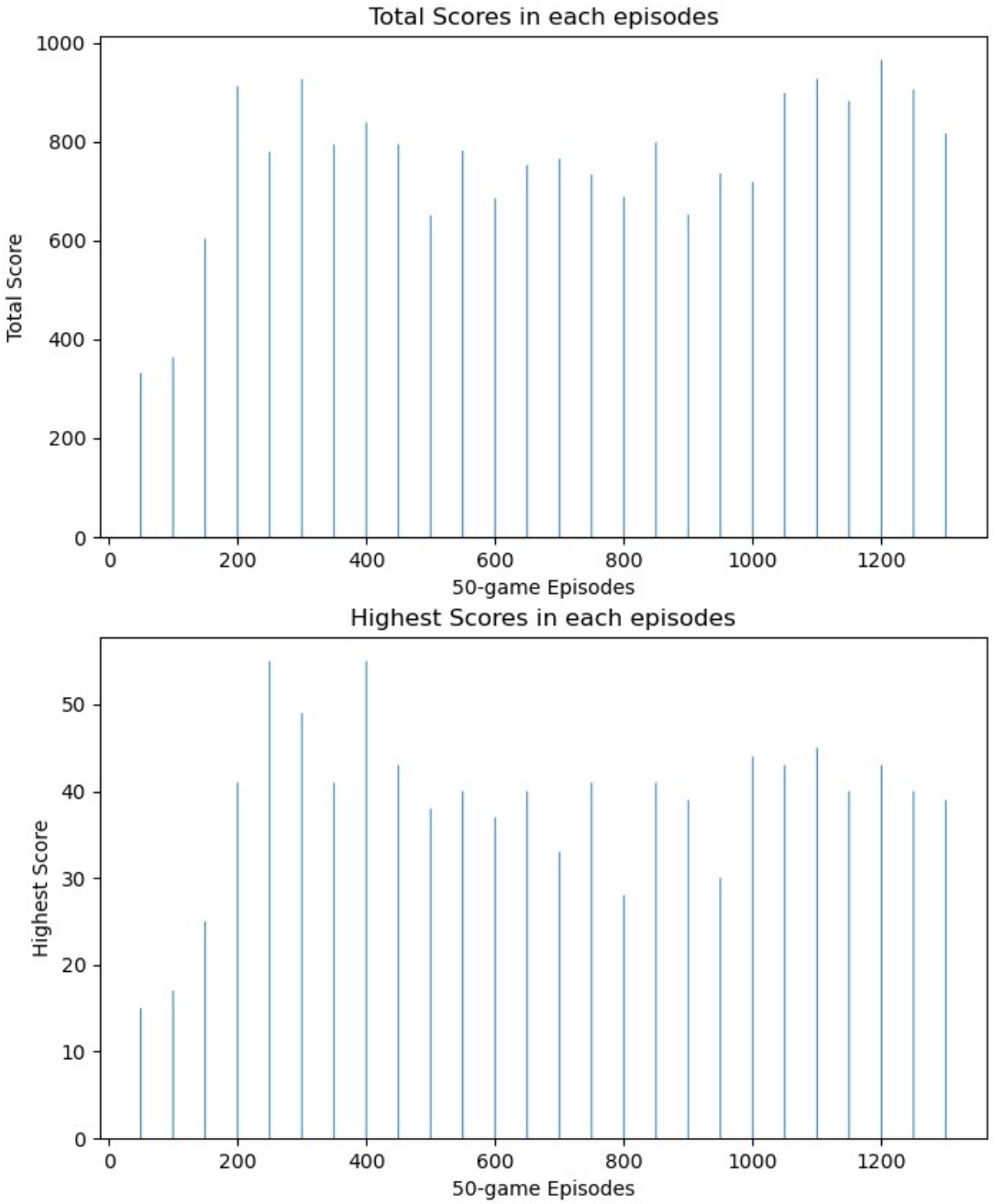

Figure 1: Sum and highest score of each episode