REPORT OF SNAKE AND LADDER GAME

**As a project work for Course**

ARTIFICIAL INTELLIGENCE (INT 404)

---------------------------------------------------------------------------------------------

Name : Jayant kumar

Registration Number : 11802821

Name : Sweta sengar

Registration Number : 11802761

Program : CSE B.Tech

Semester : Second

School : School of Computer Science

Name of the University : Lovely Professional University

Date of submission : 31st March 2020

 --------------------------------------------------------------------------------------------

Abstract

**It is always great to have someone to ready to play at your will and give you company, be it for an adult or child. As it is being hard for humans to meet someone else’s need robots are being implemented. While, there are AI based robots for chess and go, the simple game of snake and ladder has been over-looked. Proposed in this paper is a novel algorithm that can be implemented in a robot to play the game of snake and ladder.**

ACKNOWLEDGEMENT

**I would like to thank my mentor - Prof. Sagar Pande for his advice and inputs on this project. Many thanks to my friends and seniors as well, who spent countless hours to listen and provide feedbacks.**

Table of contents

1. Abstract

ACKNOWLEDGEMENT

1. Introduction

2.1 context

2.2 motivation

2.3 idea

3. Team members with Rules

3.1 Team leader

3.2 Members

3.3 Contributions

4. Libraries

5. Introduction

6. Methods from ai

7. progarm screensort

8. program output

9. Algorithm for snake and ladder game robot

10. conclusion

11. Refrences

INTRODUCTION

1. **Context :-**

This project has been done as part of my course for the CSE(H) at Lovely Professional University . Supervised by Sagar Pande, I have three months to fulfill the requirements in order to succeed the module.

1. **Motivations :-**

Being extremely interested in everything having a relation with the Machine Learning, the group project was a great occasion to give us the time to learn and confirm our interest for this field. The fact that we can make estimations, predictions and give the ability for machines to learn by themselves is both powerful and limitless in term of application possibilities. We can use Machine Learning in Finance, Medicine, almost everywhere. That’s why I decided to conduct my project around the Machine Learning.

1. **Idea :-**

As a first experience, we wanted to make my project as much didactic as possible by approaching every different steps of the machine learning process and trying to understand them deeply. Known as ” toy problem” the problems that are not immediate scientific interest but useful to illustrate and practice, we chose to take house price Prediction as approach. The goal was to predict the price of a given house according to the market prices taking into account different “features” that will be developed in the following .

TEAM MEMBERS

TEAM LEADER:-

**C:\Program Files (x86)\Microsoft Office\MEDIA\OFFICE12\Bullets\BD14565_.gif Jayant kumar**

Contributions:-

**1. Coding(joined)**

**2. Best first search**

**3. Reports**

**4. A\* search with forward checking.**

**C:\Program Files (x86)\Microsoft Office\MEDIA\OFFICE12\Bullets\BD14565_.gifSweta sengar**

Contributions:-

**1. Coding(joined)**

**2. Hardware impletations**

**3. A\* search**

**4. Random move**

**5. machine learning(joined)**

LIBRARIES

**Matplotlib:-**

Matplotlib tries to make easy things easy and hard things possible. We will generate plots, histograms, scatterplots, etc.,to make our project more appealing and easier to understand.

**Seaborn:-**

We will use it forstatistical data visualization as Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

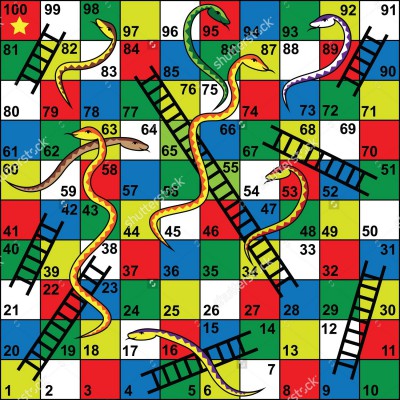
**SYSTEM:-**

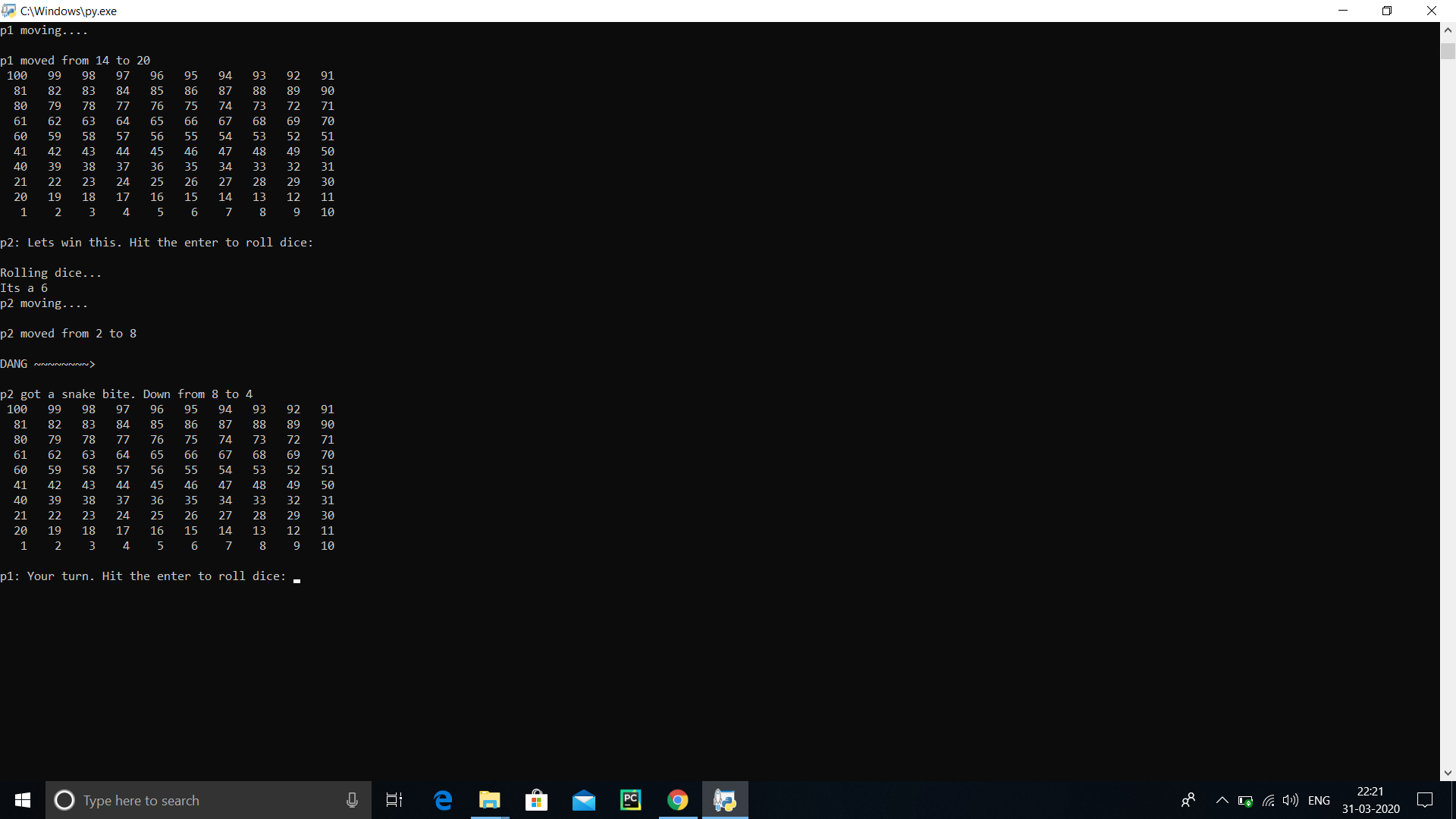
**import sys in python** is loading the module named **sys** into the current namespace so that you can access the functions and anything else **defined** within the module using the module name. On of the most common items is the list of arguments created when the program was called. This is **sys**.argv

**TIME.:-**

Python has a module named time to handle time-related tasks. To use functions defined in the module, we need to import the module first.

import time



**Snakes and Ladders** is a chance game played usually by more then one player (and less then 6). However, while playing alone might seem lonely, it is still possible. In general it is played on a 10x10 board. Each square on the board is numbered from one to one hundred. Again board must not not necessary be composed of ten by ten squares as anything more than two by two is workable. However, this might make the game extremely short. First square is a starting position, the 100th square is where players finish. Furthermore, there are a number amount of snakes and ladders placed along the board. Any snake or a ladder is basically a line, whether it is curvy or not, that is connecting two game squares. Ladder starts at a square which ID is lower then the one it finishes at. Snake are exactly the opposite. In other words ladders moves move a player's piece forwards and snakes backwards. Players use single die and different counters or any other objects that fits on the square. Once players have established who starts first. The first player roles the die and moves its piece by the number the die turned out starting from the square 1. Afterwards others do the same in a loop until someone reaches 100th square or throws more than that. If meanwhile player lands on the beginning of a snake or a ladder they have to move along to the end of it and pass the turn to the others. In order to have robot play Snakes and Ladders we made few adjustments to standard board design. First of all we have decided to keep out robot as simple as possible. We had quite a few reasons for doing that:small and inexperienced team, little time to finish and task that initially seemed quite trivial. Once we established that the core idea of the project is simplicity we had to discard the though of moving actual game counters as people do while playing the game. Instead robot itself would by a counter. It is much easier to implement, on the other hand, board becomes extremely big. We had to conserve ourselves using 5x5 board and even that brings few problems.

INTRODUCTION

Work in Artificial Intelligence has shown us that things humans find hard, and so think of as requiring superior intelligence, can be programmed relatively easily. This is contrasted with work in Intelligent Robotics which has shown us that things humans find easy and even instinctive, computers have great difficulty with. So a computer can easily say which move should be made in a game of chess, where a human may struggle, but when it comes to moving the piece the robot is likely to have difficulties. It may not even be able to recognise the piece accurately.

Therefore in order to show this divide more clearly, in this project an attempt will be made to produce a robot that plays a game requiring very little of what humans would commonly see as intelligence. The game will have no strategy involved. The robot will not need to make any choices, simply follow the rules of the game. As such it is literally child’s play, one of the simplest board games: Snakes and Ladders.

The Game

Snakes and Ladders is a quite simple racing board game. You probably have played it once or twice in your childhood. It has been around for ages. First instance of the game played was recoded in 2nd century BC in India where it was know as Moksha-patamu. The game was discovered by Europeans during the colonization of India and spread widely around the world. It has been originally used to teach children about good and bad as ladders represented good deeds and snakes punishment for the bad. Nowadays, though, the game does not carry any ethical or religious meaning.

**METHODS FROM AI**

**Best First Search:- This Greedy Best-First Search algorithm has a one-move horizon and only considers moving the dice to the position on the board that appears to be closest to the goal, i.e. ladder. We use Manhattan distance to define how close the dice number is to the ladder and snake . This method has almost guaranteed that the cube number will be able to reach the ladder in a(shortest) way at least the first four places. The previous statement comes from the fact that a dice with length seven cannot construct a square on the board 2 . However, the one-step horizon also makes it easy to get stuck on local minima and plateaus. The intuitive explanation is that, the dice only looks for the next step that is assumed best or closest to the ladder or snake without considering its tail. It is easy for the dice to roll itself once it gets longer. Eventually, this method will stop being optimal after the cube reaches the optimal solution that is end point.**

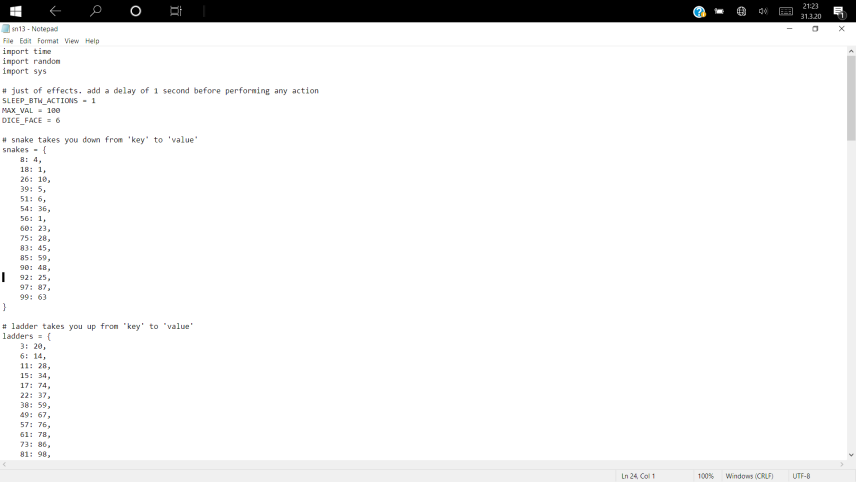
**A∗ :-** **Incorporates a heuristic in a multiple move horizon. Before taking action, it considers not only where the goal is and how far it is, but also the current state it has searched so far. This A∗ algorithm uses the Manhattan distance from the head to the ladder or snake as a heuristic and the number of steps as the “cost so far”. Each iteration of the algorithm lasts until a path is found that leads the snake to eat an apple. It improves the Best First Search algorithm by finding a full path to the apple and not stopping at the first move, this has the advantage of not getting stuck at a dead end on the way to the apple. Without memory or time restrictions, the algorithm is guaranteed to find an optimal path to the apple if one exists. There are two minor modifications compared to a standard A∗ algorithm. First, ties between nodes with same estimated total cost (that is, the sum of heuristic and cost to reach the node) are not broken randomly. Instead, one of the nodes with lowest heuristic is chosen. This helps the algorithm in finding a path with minimum cost to the goal faster. Note that there can be many optimal paths to the goal. Second, the maximum number of nodes expanded is limited. This makes the algorithm stop if a path to goal cannot be found (for any reason). In case the maximum nodes bound is reached, the algorithm will switch back to Best First Search for that iteration.**

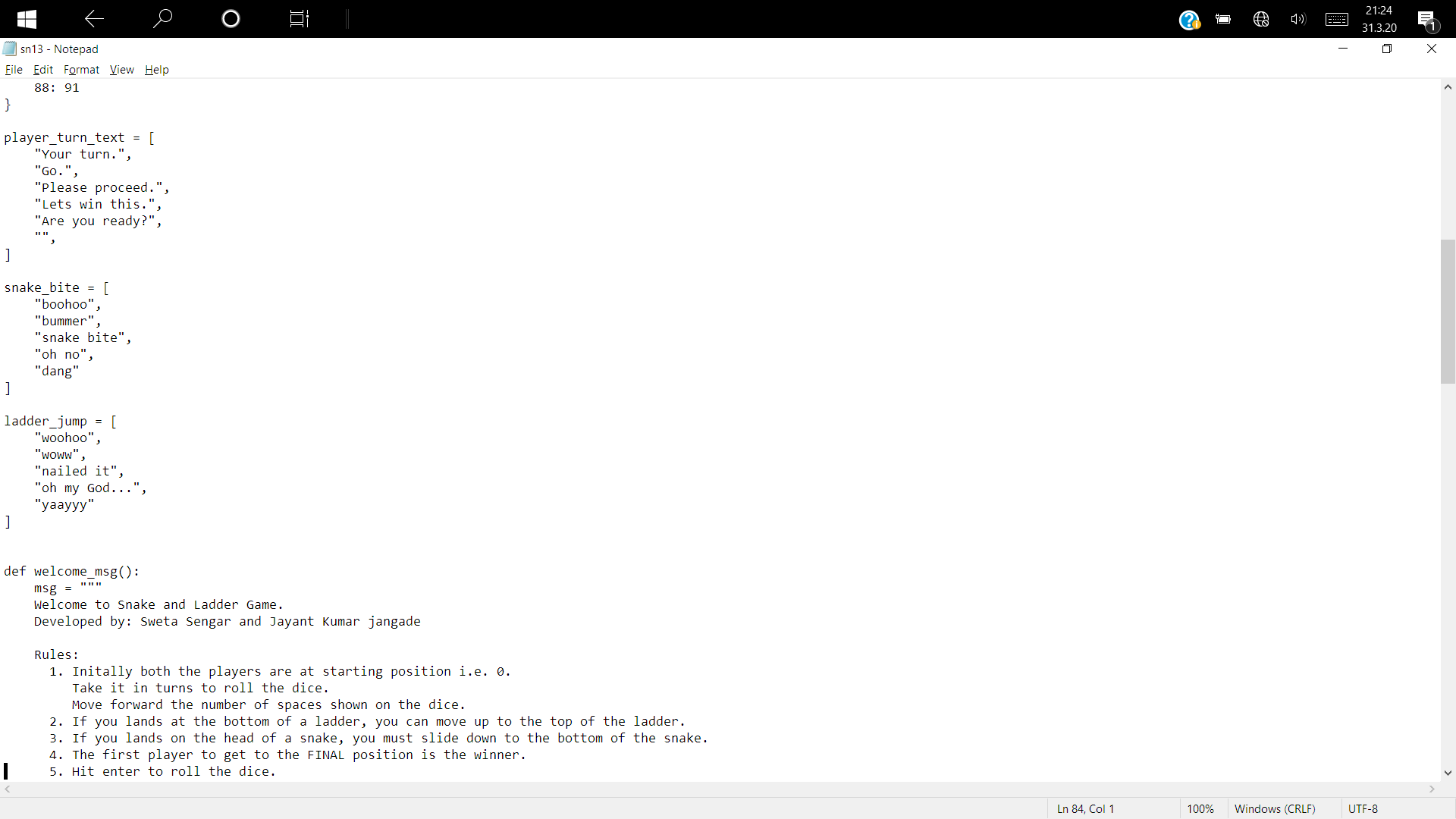
**A∗ Search with Forward Checking:-** **The A∗ algorithm introduced in the previous section still has some shortcomings. One of the shortcomings includes the fact that, once the apple is eaten, the snake can reach a dead end which can be avoided with other paths. In other words, the algorithm does not take into account the effects of the selected path once the apple is eaten. To avoid these dead ends, the A∗ algorithm is also equipped with a Breadth First Search algorithm that is used to compute if a path to a goal also leads to a dead end. Once an iteration of the A∗ algorithm ends, it will then call the Breadth First Search algorithm starting at the goal state found by the A∗ algorithm. From here, it will explore the full tree up to a certain number of nodes. If the tree is contained inside this node bound, i.e. it is a dead end, the path to the apple is rendered as not good, and the goal node from the A∗ algorithm will be discarded (the A∗ iteration will continue though). This dead end check is also used when the A∗ algorithm cannot find a path to the goal. It will then select the Best First direction that does not lead to a dead end.**

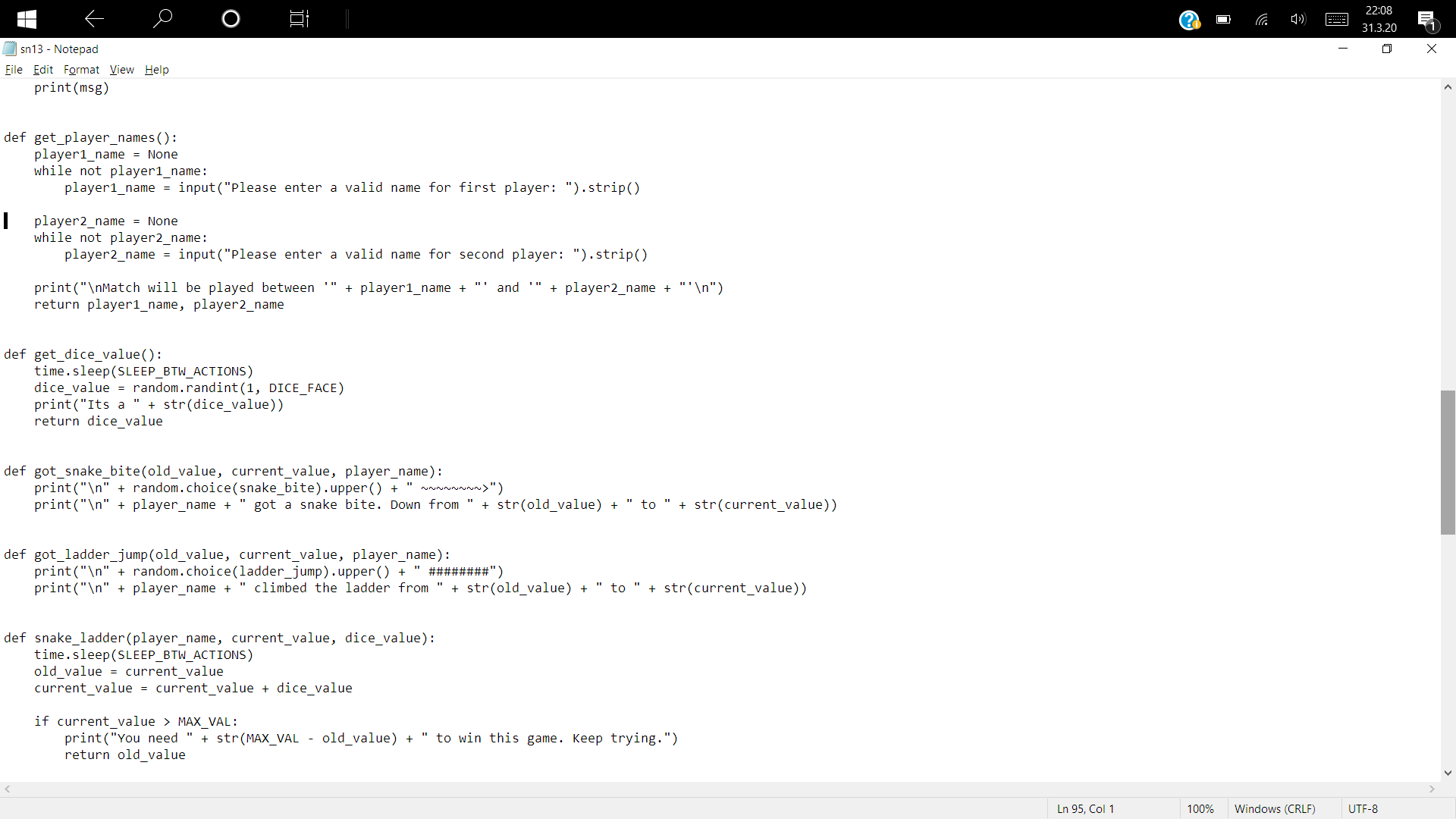
**Random Move:-**

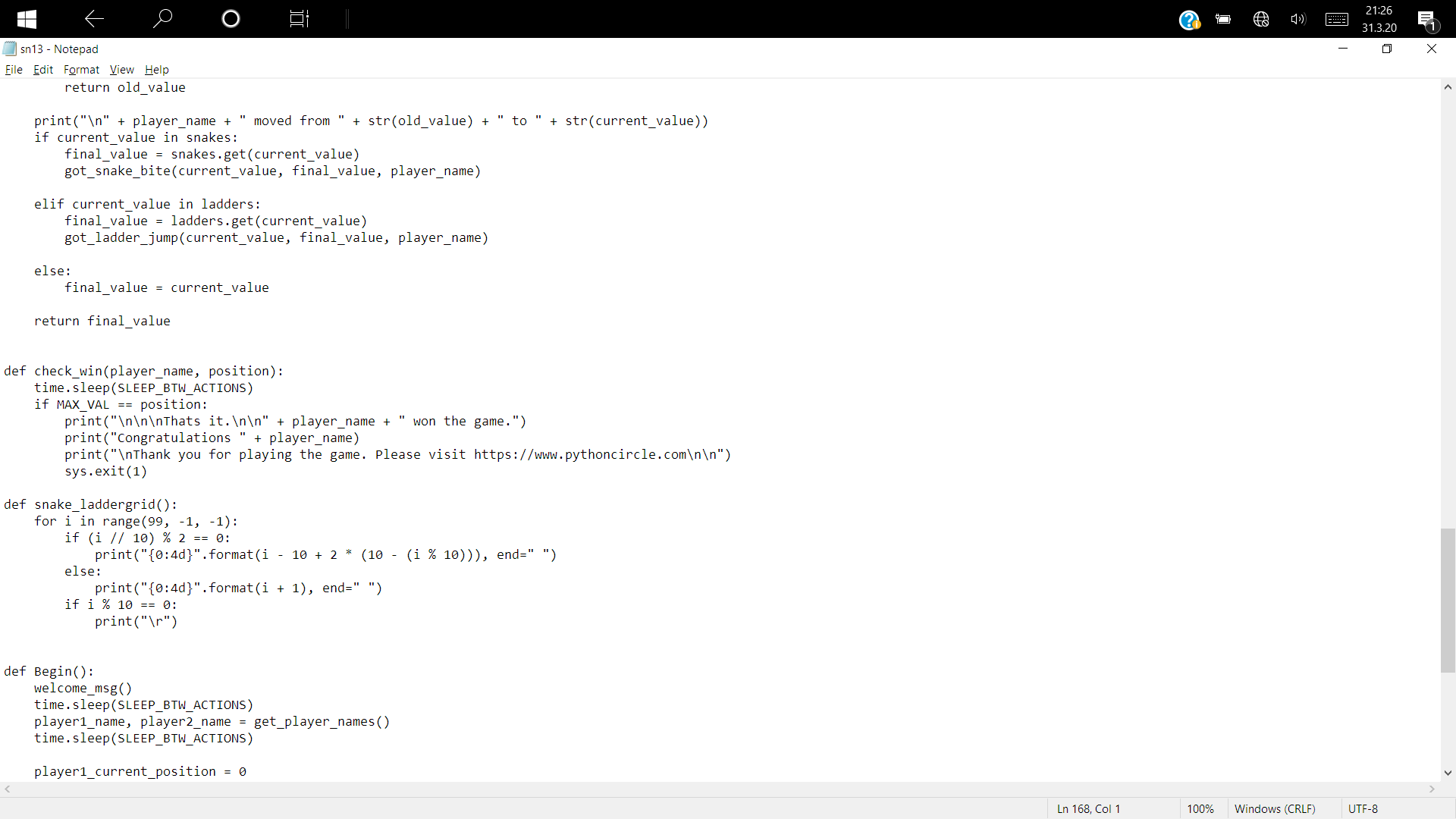
**Besides the three Artificial Intelligence methods described previously in this section, we also introduce two baseline methods for comparison. The first one of the two is Random Move. Just like the name suggests, Random Move selects the next step to move randomly. We impose only one condition: if possible, the move chosen must not end the game. As we can imagine, this method makes the snake spend a long time searching for apples as it considers nothing about the position of those. Moreover, this method can easily lead to a dead end, since it does not consider the full position of the snake on the board. Even though this method does not seem to work well, it can be a good baseline for experimental comparison with other AI methods, just as shown in Section 4.**

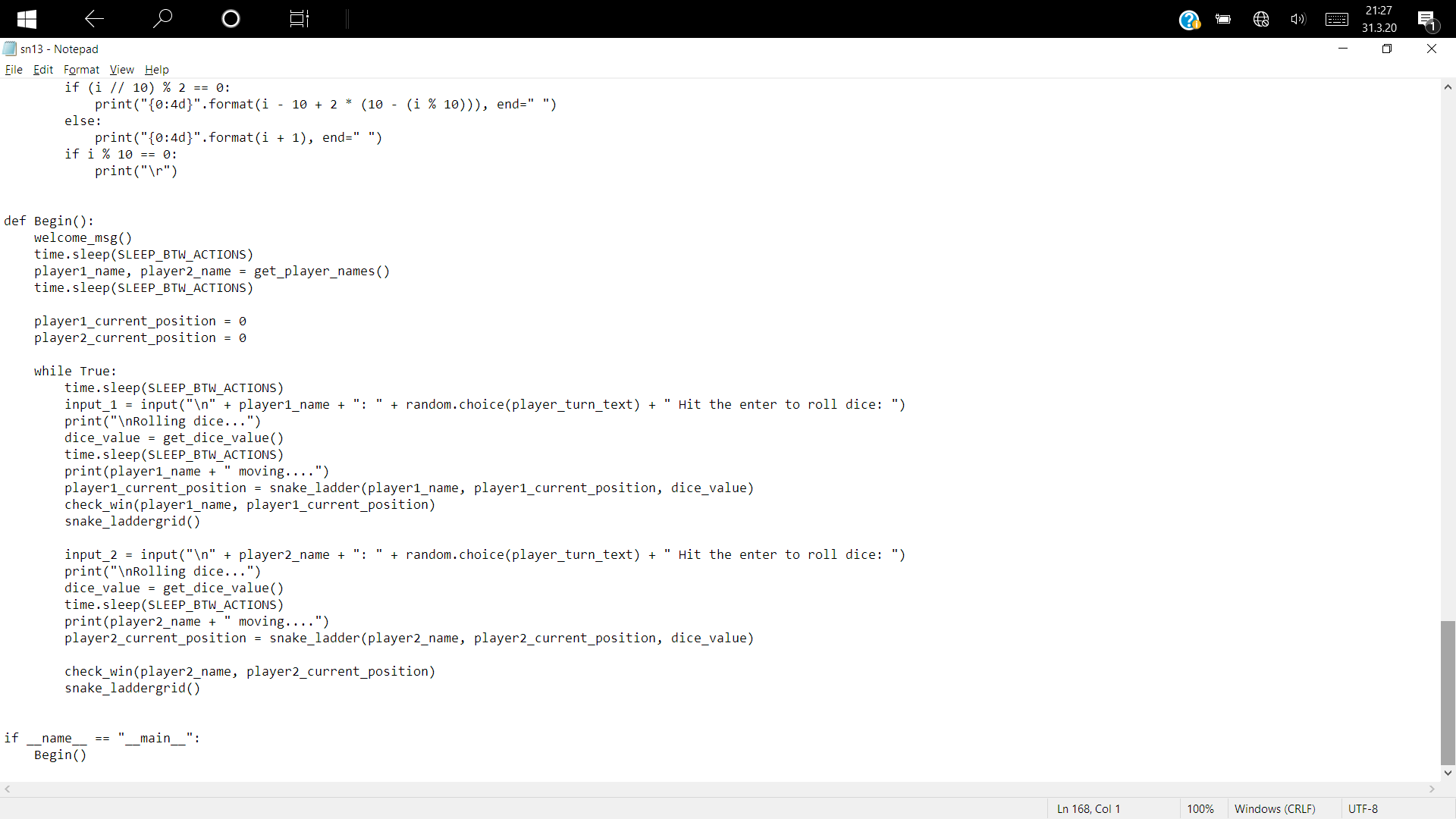
Program screensort



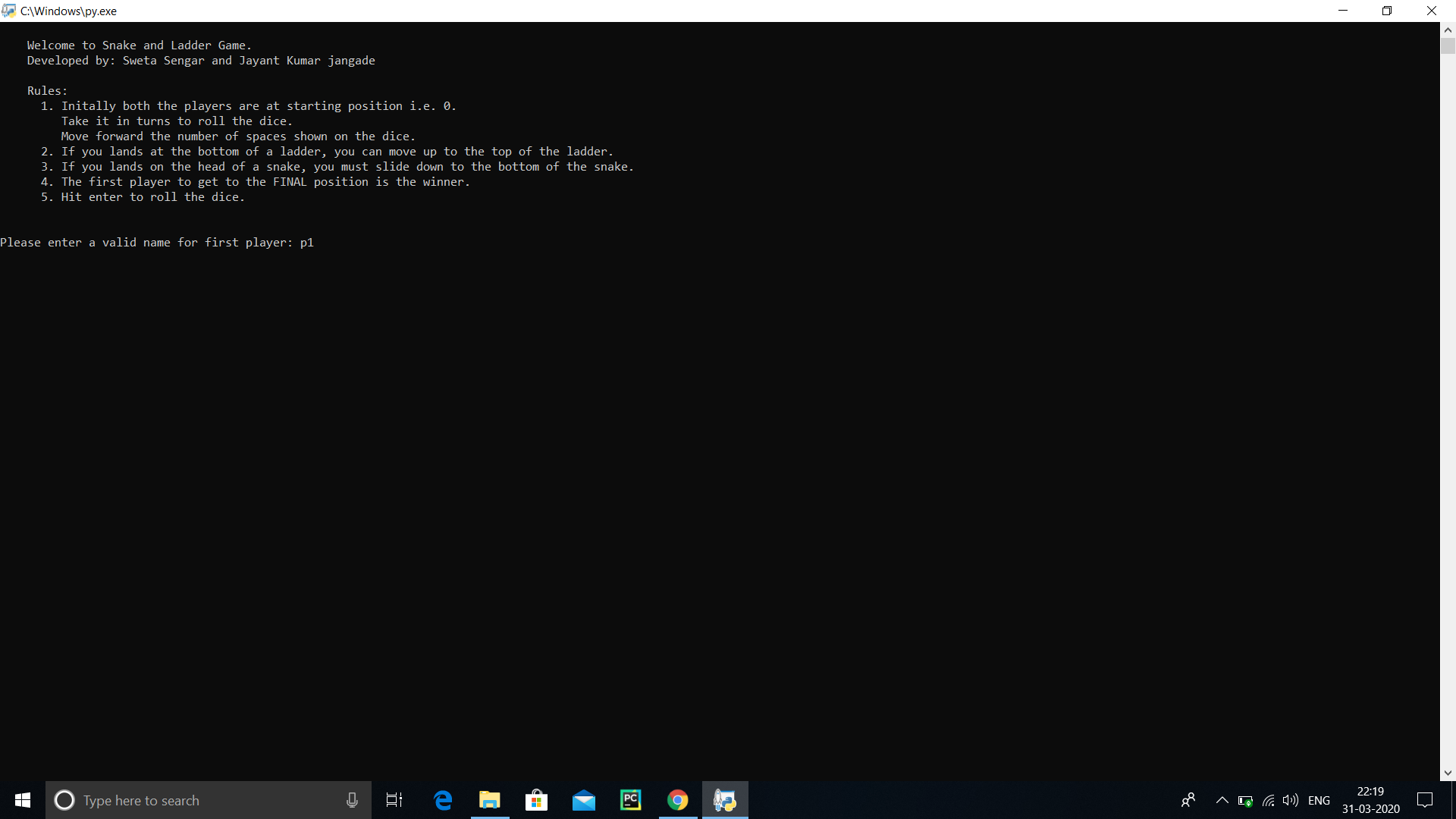


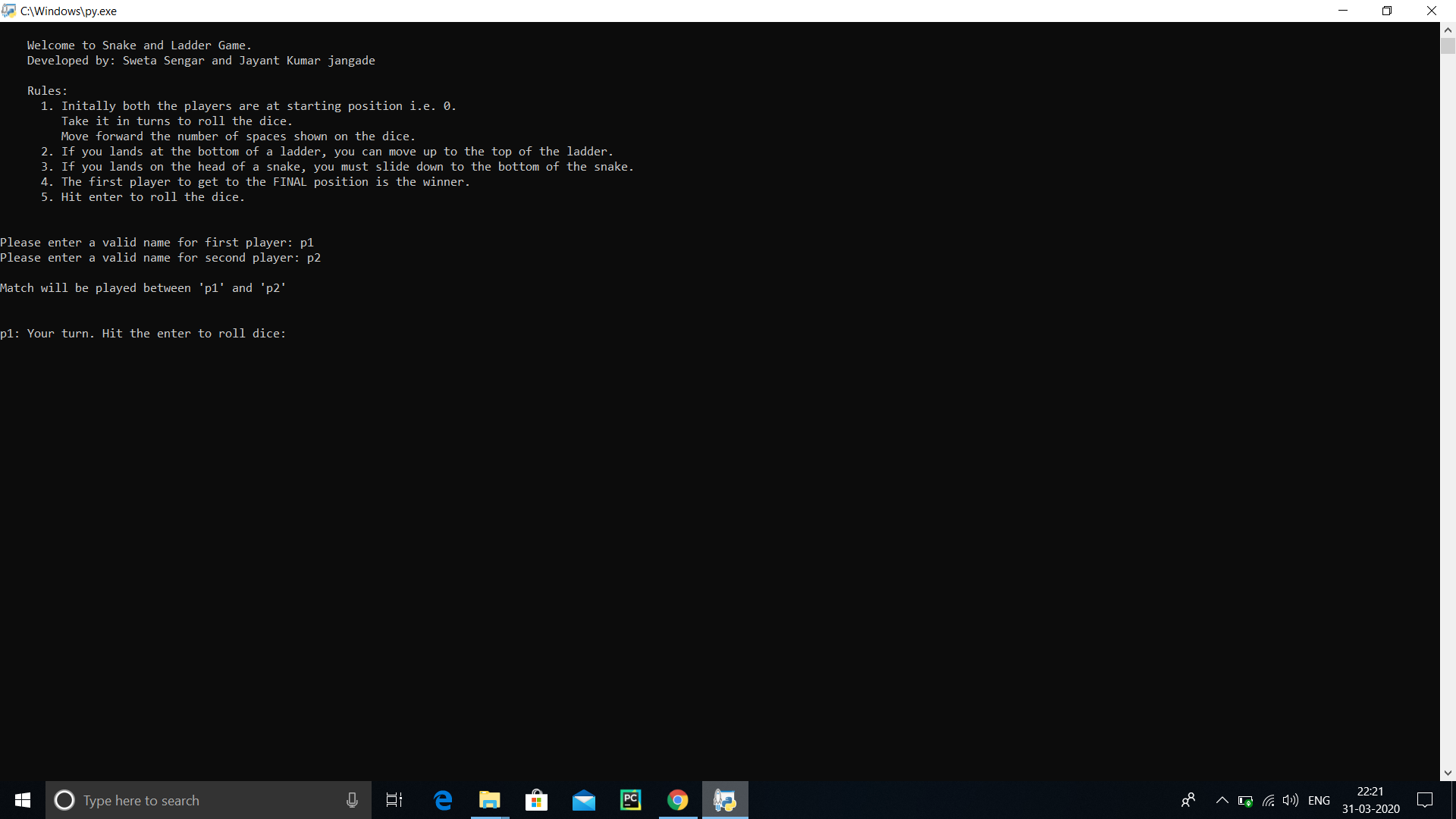


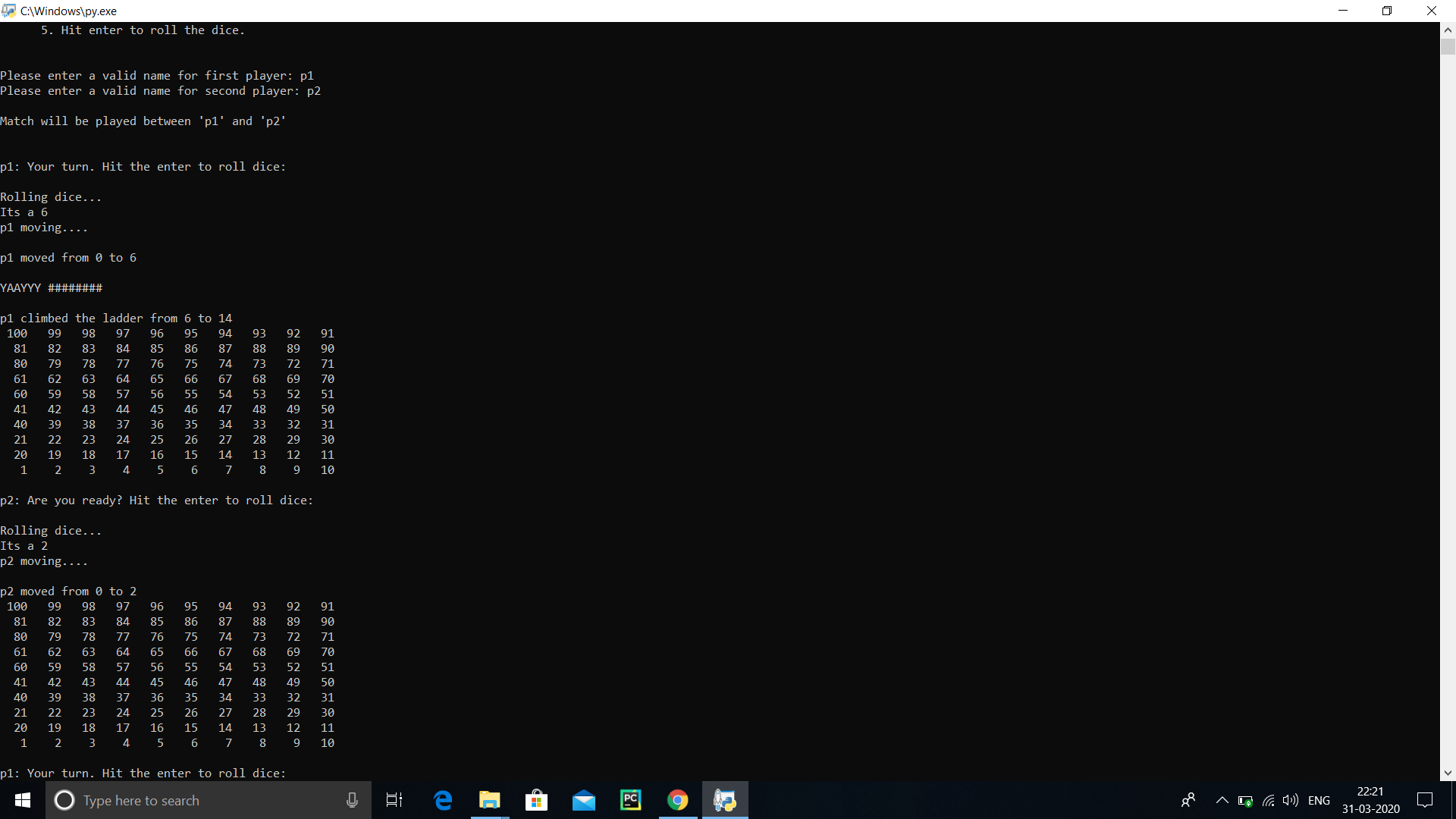


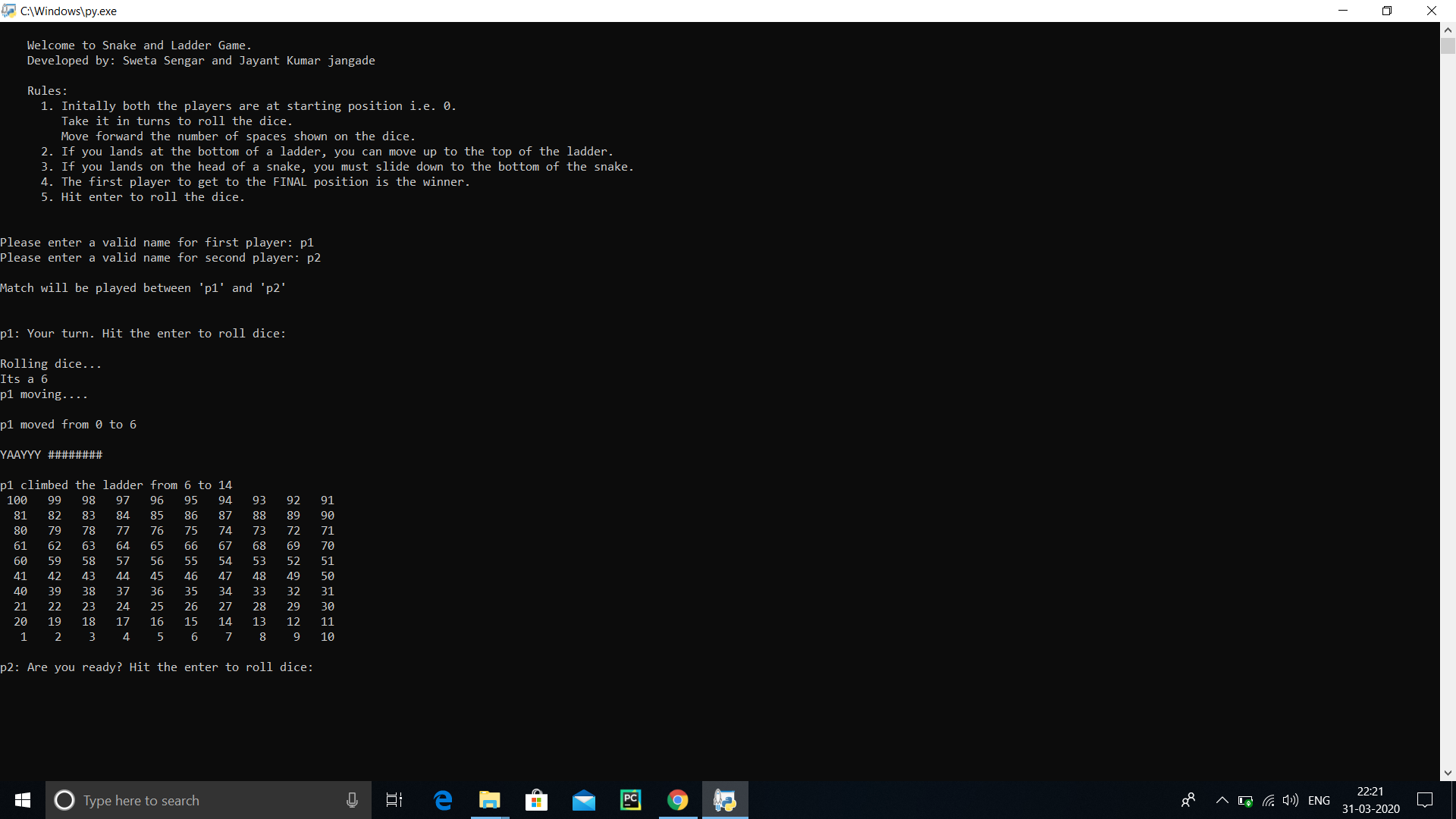


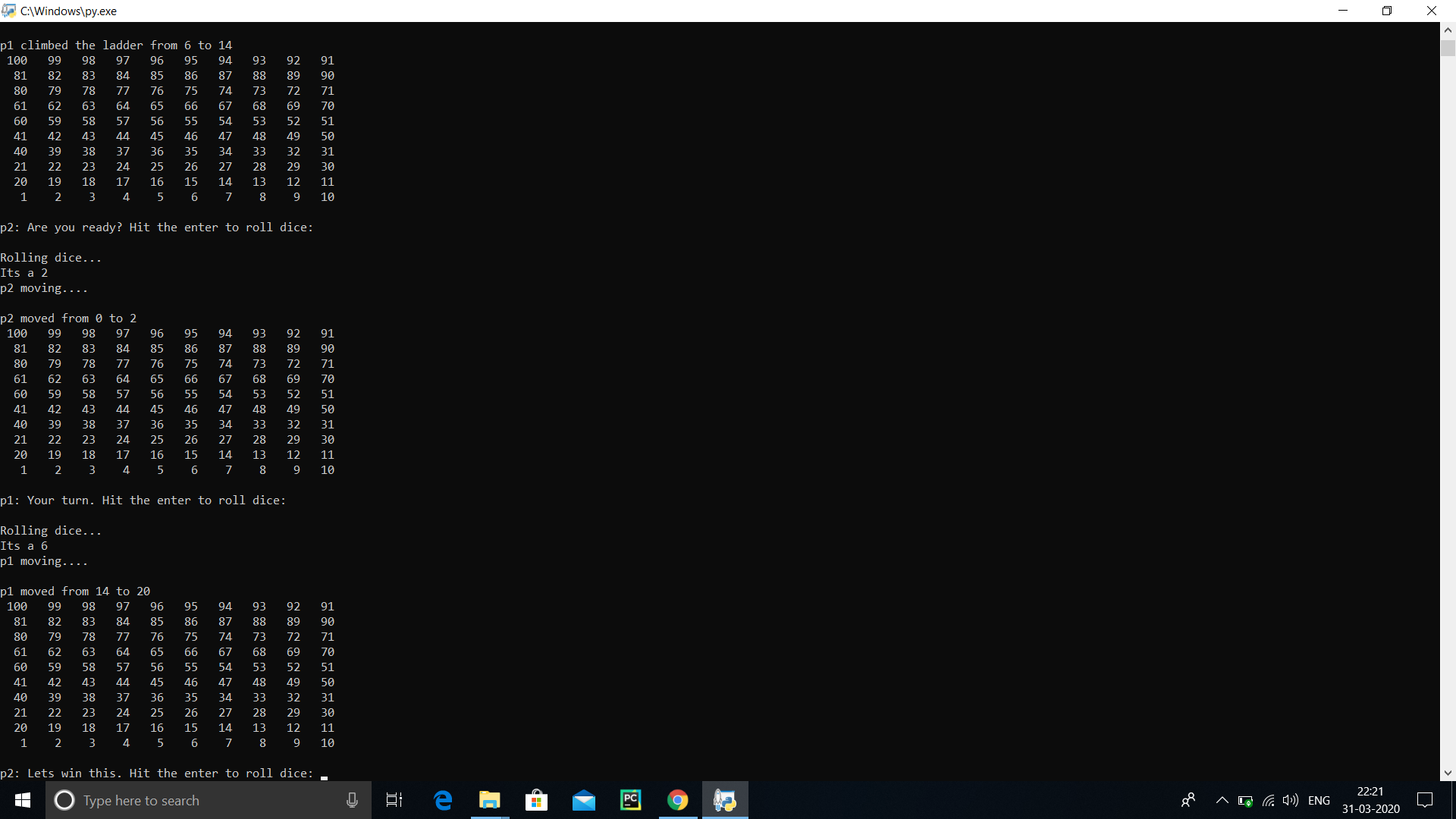
Program output











**Algorithm for Development of Snake and Ladder Robot**





ALGORITHM

**Certain requirements before coming to the main system algorithms are to be initialized.**

**a) A counter to keep track of the movement and current position of the game piece.**

**b) A constant to store the length of the side of the square on the board**

**c) Four arrays**

**LS = {x1, x2, x3. . . xn} to hold the numbers where the ladders start.**

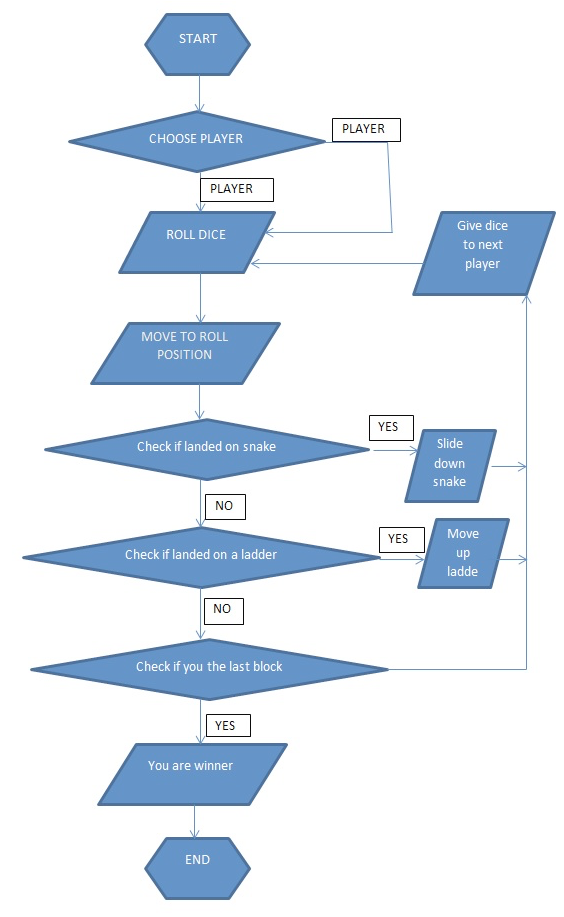
**LE = {y1, y2, y3. . . yn} to hold the numbers where the ladders end.**

**SS = {w1, w2, w3. . . wn} to hold the numbers where the snakes start.**

**SE = {z1, z2, z3. . . zn} to hold the numbers where the ladders end.**

A. **Main Algorithm**

**The Algorithm is depicted in flowchart shown in figure I. The system works by pushing the game piece from one step to another and the counter keeps track of the current location of the game piece by incrementing itself or updating as the case. Once the number has been retrieved from the die it is given to the microcontroller where it undergoes an iteration with count equal to the number on the die. In each iteration, it checks for the series beginning numbers i.e. 11, 21,31. . . and calls a function „move’. At the end of the iteration it has to check if the position is the mouth of the snake to descend or the base of the ladder to ascend. This is done by checking if the current position number exists in LS or SS array. If no, then no action is performed. If yes, then the position of that number is identified and the game piece is moved to the corresponding number in the same positon in LE or SE array. After reaching the final point it has to call another function ‘identify’. The reason to check for the series beginning number in each iteration is to differentiate between horizontal push and vertical push. Say, the game piece is to be moved from 9 to 10 then it is pushed horizontally. But if the piece is to be moved from 10 to 11, then it has to give a vertical push therefore the counter is monitored for such numbers.**

** FLOW CHART**

B. Function ‘move’

**This function will have to perform two task. The first is to move the game piece to the next square. Second to move the chassis of the robot by a distance almost equal to the side of the square. The implementation of this is in section V**

C. Function ‘identify’

**The need for the function arises while deciding if the game piece has to be moved horizontally left ways or right ways. This is done by classifying based on the digit in the ten‟s place. If the ten‟s place digits are even, then the coin is moved right ways if the digit is odd, then left ways. But this depends on the individual board whether the numbers start from left or right. This function takes the number of the current position into consideration. The function is called after the ascension of the ladder or descending down the snake to check in which direction is the next move.**

III. HARDWARE IMPLEMENTATION

A. **The Arm We have already loaded the length of each square on the board onto the microcontroller. The arm that moves the game piece is designed on a moveable chassis with a swivel base. This arm is different from the arm which throws the die which is fixed at the bottom. The size of the chassis is 12 x 7 inches or any other proportionate size. The swivel base is controlled with a servo motor and it supports a vertical arm onto which a telescopic horizontal arm is mounted. The end of the telescopic arm has two servo motors configured in such away, that one servo motor enables the game piece to be pushed sideways. The other motor turns the arm attached to it by right angles whenever there is a change in the direction of push. i.e. from horizontal to vertical. The vertical movement are enabled by the telescopic arm which moves front or back as per the need. All these movements are controlled by the microcontroller and selection of the microcontroller decides how fast the algorithm is implemented and a suitable action is performed.**

B. **Working Initially the game piece is assumed to be placed at the centre of the first square and for every step it is pushed by a distance equal to the length of the square. At the row end the servo rotates and the piece is pushed vertically upward. In the next move the servo rotates again to change the direction of movement which will be opposite to the below row. When the ladder base or snake mouth is detected, the controller takes the current position and final position numbers from the array. From these the absolute value of the difference of the ten‟s place and unit‟s place is taken and the hypotenuse and the angle made by the hypotenuse with the horizontal is calculated. The swivel is rotated by the complement of the angle thus obtained. The direction of clockwise or anticlock wise is determined by the following logic.**

**Let,**

**d= difference between current and final position unit‟s digit.**

**ad= absolute difference between the current and final position ten‟s digit.**

**IF((d>0 & ad=odd) || (d<0 & ad=even))**

**Move clockwise**

**Else**

**Move anticlockwise**

iV. NEED OF THE CHASIS :-

**As already mentioned when the function move is called the chassis is moved by a distance equal to the length of the square. This is possible with the replacement of dc motor with stepper motor for the wheels. Moving the chassis along with each move keeps the arm in alignment with the current position of the game piece. The distance to be covered by the wheels is based on the area of contact of the wheels with the ground and the step size of the motor. This should movement must be in relation to the length of the square on the game board. Thus, the amount of turn for one increment in position is calculated and is used by multiplying with an integral number as needed.**

1. When Encountering a Ladder

**When a ladder is encountered, the game piece is pushed up the ladder by the telescopic arm in the direction turned by the swivel base. The chassis covers the horizontal distance from the current position to the new position based on the difference in the digit in unit‟s place. The decision whether to move forward or backward is decided the direction the swivel had turned.**

1. When Encountering a Snake

**The case is a little different when the snake is encountered. Using the same logic as in the case of ladder, the angle, the hypotenuse length, the direction of rotation is found out. But here the chassis first moves to align itself in the final positon and the pulls the game piece from its current position downward to final place along the hypotenuse. Thus, the piece reaching its required position after descent.**

V. CONCLUSION

**The algorithm and technique proposed effectively covers the tracing of the path, ascending the ladders and descending down the snakes on any given board. However, it is not completely independent as the die is required to be placed in its cupped arm slot and a switch needs to be pressed for each move. Efforts can be made to eliminate this. Further development to this can be made by being able to move the game piece along the curve of the body of the snake which would seem more graceful to the onlooker.**

**Machine Learning :-**

**Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of Computer Programs that can change when exposed to new data.**

**Supervised Learning :-**

**Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.[1] It infers a function from labeled training data consisting of a set of training examples. • Supervised learning is when the model is getting trained on a labelled dataset. • Labelled dataset is one which have both input and output parameters.**

**REFERENCES**

**[1] Cynthia Matuszek et all, “Gambit: An Autonomous Chess Playing Robot Systems” 2011 IEEE International Conference on Robotics and Automation**

**[2] van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yarger N,Gouillart E, Yu T, the scikit-image contributors. (2014) scikit-image: image processing in Python. PeerJ 2:e453** <https://doi.org/10.7717/peerj.452>

**[3]** <https://in.mathworks.com/help/images/ref>

**[4] Rosebrock , Adrian. How-To: Python Compare Two Images https://www.pyimagesearch.com/2014/09/15/pythoncompare-two-images/**