# Student Details

Name: Johanz Te

Deakin Student ID: 223528029

Tutor: Mr Xiangwen Yang

Class: SIT320 – Advanced Algorithms

Intended Grade: Credit

**Instructions: Please fill in the module name, along with submission and discussion deadlines from Ontrack website.**

I will adhere to following timetable for submitting tasks, and will come to class for task discussion with my tutor.

# SIT320 - Time Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Module** |  | **Tasks** | **Submission Deadline** | **Discussion Deadline** |
| 0 | **Introduction** | Must | Module 1 Task | 14 July | 21 July |
| 1 | Trees | P |  | 21 July | 28 July |
| 2 | Distributed Algorithms | P |  | 28 July | 4 Aug |
| 3 | Algorithm Analysis | C |  | 4 Aug | 11 Aug |
| 4 | Graphs | P |  | 11 Aug | 25 Aug |
| 5 | Dynamic Programming | P |  | 18 Aug | 25 Aug |
| 6 | Greedy Algorithms | P |  | 25 Aug | 1 Sep |
| 7 | Linear Programming | P |  | 1 Sep | 8 Sep |
| 9 | Flow-based Algorithms | C |  | 8 Sep | 15 Sep |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | Portfolio due |  |  | 13 Oct |  |

Discussed with your Tutor (Circle one): No

Signatures: Johanz

SIT320 PLANNING FORM 1

Tic-Tac-Toe Psuedocode

Design a solution to win the game

In this Psuedocode we assume that there are three positions left and the computer has to make the best decision.

A game with green and black crosses

Description automatically generated

Like this for example (gotten from lecture slides)

#Minimax function pseudocode inspiration gotten from <https://github.com/Cledersonbc/tic-tac-toe-minimax>

Class main

Board = 3x3 board (made up of some function)

X = computer

O = user

Spaceleft=9

Function minimax (state, depth, player)

if (depth == 0 or gameover) then

score = evaluate(state, player)

return [null, score]

if (player == computer) then

#Initialize best move and score for computer

best = [null, -infinity]

for each valid move m for player in state s execute move m on s #iterate over each valid move

[move, score] = minimax(state, depth - 1, -player) #Looks at what the opponent might do next

undo move m on s #Undos the move

#Updates the best move depending on which score is higher

if score > best.score then

best = [move, score]

else

best = [null, +infinity]

for each valid move m for player in state do

execute move m on state

[move, score] = minimax(state, depth - 1, player)

undo move m on state

if (score < best.score) then

best = [move, score]

return best

Function main

For Spaceleft i>0 and not gameover

{

Print board

[move, score] = minimax(board, depth, computer)

Execute move on the board

Decrease Spaceleft by 1

If there is a winner

Computer wins

Gameover

User moves

Decrease Spaceleft by 1

If there is a winner

Gameover

User wins

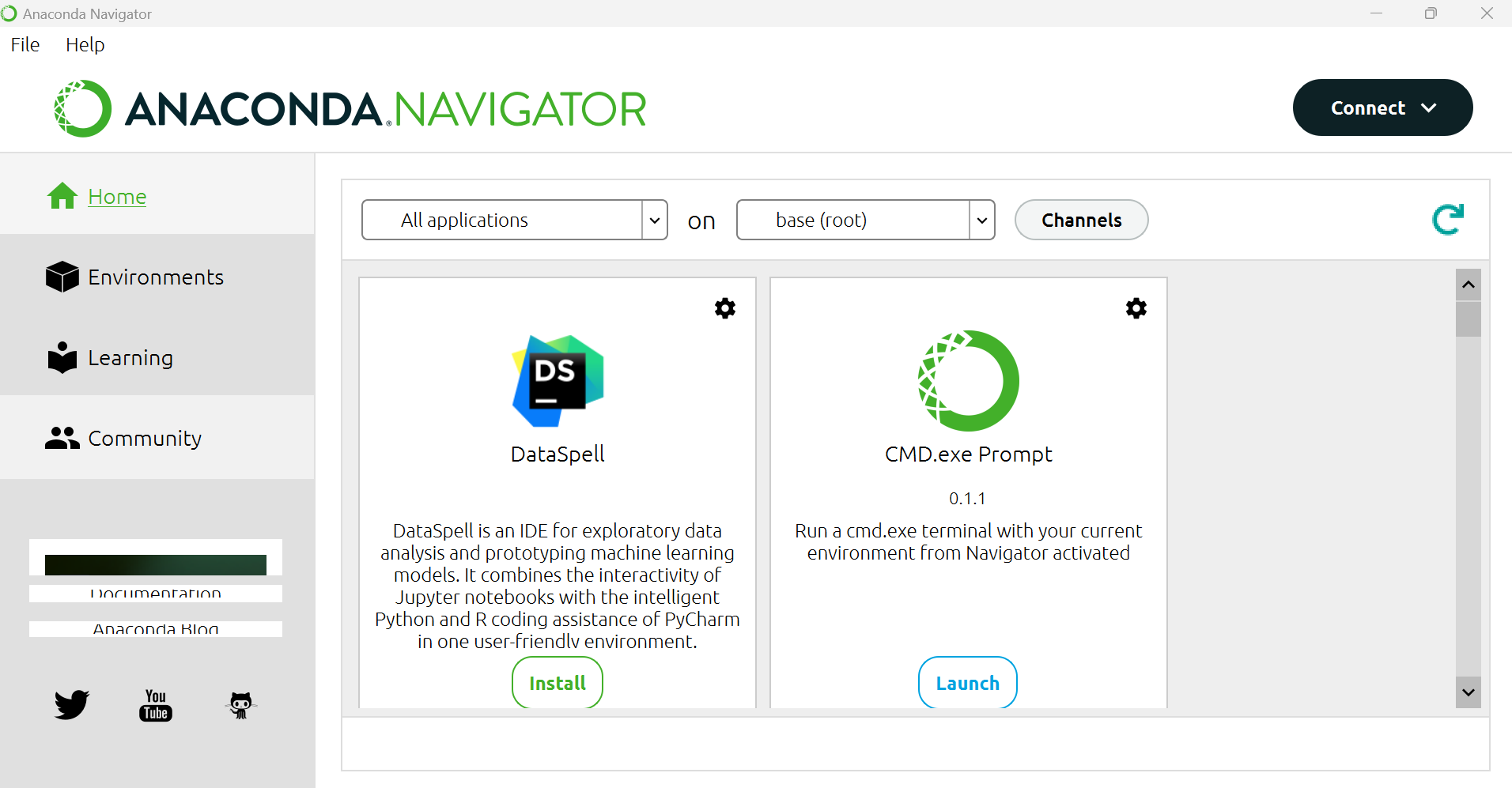
Else if spaceleft = 0

Draw

}

Draw

End

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The Minimax Algorithm has the time complexity of O(b^d), where b is branching factor of possible moves and d is depth for future moves. Usually the branching factor starts of at 9 (as there are nine possible moves at the start of the game), however the branching factor will decrease to 1 as the game progresses. The depth is 9 as there are only 9 possible positions in the board. The algorithm has an exponential time complexity of b^d because for every b moves, there a b more possible moves. Tic-tac-toe is a class P problem as the game board only has a fixed size of 9, meaning there is a finite number of moves.