Game Playing with Monte-Carlo Tree Search

Interim Report

Video Link:

<https://youtu.be/J0g4LYI5x3s>

Alexandra Danciu

# Aims and Objectives

The primary aim of this project is to design and implement a game-playing program using the Monte-Carlo Tree Search (MCTS) algorithm, evaluating its effectiveness through experimental analysis. The AI opponent will play Connect4 against real players, it should be able to showcase the decision-making capabilities of the algorithm well and be a good opponent.

Early objectives include creating proof-of-concept programs. One of these will be to solve bandit problems using Upper Confidence Bound (UCB) methods and compare these strategies with naive approaches. This comes into play in the simulation section of MCTS, where the issue at hand looks a lot like a bandit problem, UCB can be implemented in this section to explore the unsure parts of the tree and return the best arm.

Another proof of concept is a basic Connect4 game that allows two human players to play the game. This will be modified next term to have a GUI interface and allow players to choose to either play two-player or vs AI.

# Planning and Timescale

In terms of planning, I had my timeline and plan for both terms in my Project Plan. In said plan, I had broken up my weeks into four blocks.

The first block was met, it was to do with the project plan and spending the first two weeks doing that, which I did according to my diary and submitted the work.

The second block was not met, as you can see from the diary below, I did not do any work from October 5th to November 4th. This was due to poor time management on my part, I was doing coursework for my other classes and had neglected this course. I set up a meeting with my supervisor and so began my catch-up. I spent the next two weeks completing the bandit proof of concept code and working on the report. I then began work on the Connect4 proof of concept and reports, keeping in consistent communication with my supervisor. I did get everything done in time and managed to catch up to my original plan, completed block four as planned.

My goal for next term will be to start this course first, ensuring I follow the plan I set out for myself and that I do not get behind on my work again.

## Diary

### Thursday the 12th of December, 2024

Completed video and submitted all work for interim submission.

### Wednesday the 11th of December, 2024

Finished interim report.

### Tuesday the 10th of December, 2024

Finished work on Connect4 proof of concept code.

### Sunday the 8th of December, 2024

Continued work on Connect4 proof of concept code.

### Saturday the 7th of December, 2024

Added transitions and animations to PowerPoint and submitted. Formatted interim report and added work from proof of concept reports. Added foundation code from Connect4 UML. Fixed issues with check-style not working, configured it, fix check-style errors on all classes.

### Friday the 6th of December, 2024

Finished the PowerPoint for submission tomorrow and started thinking of the script.

### Wednesday the 4th of December, 2024

Made a plan for the final stretch. Started work on the rough draft of the presentation and putting together the proof-of-concept reports for the interim report. Formatted diary.

### Monday the 2nd of December, 2024

Attended final meeting with the supervisor, discussed final submission details, Connect4 proof of concept and presentation.

### Sunday the 1st of December, 2024

Started work on the Connect4 proof of concept, creating a UML. Made some plans for the interim deadline and prepared for a meeting with my supervisor for tomorrow.

### Thursday the 28th of November, 2024

Completed work on Bandit POC report, writing about the comparison of algorithms from the results table and explaining the code using the UMLs.

### Wednesday the 27th of November, 2024

Used debugger on bandit solver to complete code and ensure everything works as intended. Completed the results table in the report for comparison of algorithms. Updated UML to current code and added to report for explanation of code.

### Monday the 11th of November, 2024

Continued work on the Bandit Problem report.

### Sunday the 10th of November, 2024

Continued work on the Bandit Problem report.

### Saturday the 9th of November, 2024

Completed the UCB class. Updated the UML to fit the final code. Did more work on the bandit problem report.

### Friday the 8th of November, 2024

Continued work on UML and code for bandit problem proof of concept, completed the E-Greedy class.

### Wednesday the 6th of November, 2024

Continued work on UML and code for bandit problem proof of concept, completed the explore-only and exploit-only classes.

### Tuesday the 5th of November, 2024

Attended meeting with supervisor, discussed UML for bandit problem proof of concept and logistics.

### Monday the 4th of November, 2024

Started work on the bandit problem proof of concept. Created a rough outline for the bandit problem report. Created a rough UML for the program to ensure the program is planned out properly. Set up a meeting with the supervisor for tomorrow to discuss UML.

### Sunday the 27th of October 2024

Created an empty project to start work on code.

### Saturday the 5th of October 2024

Finished project plan and submitted.

### Thursday the 3th of October 2024

Continued work on the project plan. Had a meeting with the project supervisor, we discussed the plan.

### Wednesday the 2nd of October 2024

Started work on the project plan. Created a rough template mainly focussing on abstract and risk assessment. Preparing for a meeting tomorrow with a supervisor.

# Proof of Concepts

## Bandit Problem

What are Bandit Problems?

Bandit Problems are statistical problems that involve trying to gain as many points over a limited time as possible. There will be multiple places to visit that include an unknown number of points each. The goal is to find a balance between exploring and exploiting.

Exploring means going out to new places and figuring out how many points they can give you, most of the time the points will vary at different times so the average number of points would be best.

Exploiting is going to the place where you know the average number of points given is high and you are likely to gather the most points here.

One example would be going to a restaurant to eat. Say you had 300 days, and you wanted to go out to eat each night at 3 different restaurants. Each restaurant will give you a certain amount of happiness to go eat there each night. One restaurant could give you an average of 10 happiness with a deviation of 2.5 each way, a second restaurant could give you 8 happiness on average with a 2 deviation each way, and the third restaurant offers an average of 5 happiness with a 12.5 deviation.

You want to have a balance between going to a restaurant and trying its menu and seeing how much happiness it gives you and going to your favourite restaurant that you are more confident will make you happy.

Naïve Strategies

There are a number of different algorithms you can use to solve the bandit problem, however, the efficiency of some of them compared to other, better, algorithms are lacking.

Explore Only

One of these less efficient algorithms that still gets the job done is Explore Only. With this algorithm the entire 300 days is spent exploring and picking random restaurants to dine at. By the end, you have spent a considerable amount of time at the wrong restaurants and have lost some points on the way. Each restaurant will end up having an equal 100 days spent at them.

Exploit Only

Another inefficient, naïve algorithm would be Exploit Only, which is similar to Explore Only where it does only one-half of the explore/exploit in the problem.

With this algorithm you will use the first three days to visit each restaurant once and which restaurant gave you the highest happiness level you will visit for the rest of the 297 days.

While this may seem like an ok algorithm, the happiness points given by each restaurant have a deviation. On day one you may visit the best restaurant, number 1, and get a low deviation that gives you 7 instead of the average 10, and then you go to restaurant 2 on day 2 and get an 8. You would then continue to go to Restaurant 2 for the remainder of the days and would only be visiting the second-best restaurant, lowering your chances for higher total happiness points.

E-Greedy

This is one of the better algorithms that could be used to generate decent results.

With this algorithm you start by denoting an E value, say E = 10%.

What this means is that each day, there is now a 10% chance that you will visit a random restaurant and a 90% chance that you will visit the restaurant that has historically given you the most happiness points. These exploit days, the 10% chance that you visit a random restaurant, gives you more evidence of which restaurant is the best. It will either ensure that you switch restaurants to the best one in the case that the first time you went to one a deviation happened, or it will make you more confident that the restaurant you are currently going to is the best one.

The issue with this algorithm is that its performance depends on the chosen epsilon value and the value of the possible deviations.

Upper Confidence Bound (UCB)

This is one of the best algorithms used to solve bandit problems. It is great at balancing the exploration of new arms or arms with low confidence and exploiting arms with known high values or arms with high confidence values as they have been visited frequently. This algorithm works similarly to E-Greedy. The strategy is that:

At each time T, pick point R, such that

A math equation with numbers and symbols

Description automatically generatedmean of R + √ 2 \* current time T / number of times visited R

With this algorithm, each time an arm needs to be selected, each arm’s UCB value is calculated using the equation above, and the highest value is selected. This works similarly to E-Greedy in the way that the arm with the highest value is visited more frequently as it gives back the most points but the calculation of the UCB also means that other arms are visited at regular points and in a fair way; it is not left to random chance. The success of E-Greedy depends highly on the value of E, which UCB does not struggle with.

A screenshot of a computer screen

Description automatically generatedCode

The coding for this proof of concept started with the UML design above. We have the BanditArm class that creates instances of the arms equipped with points and numbers of visits and some methods to get and set variables. These arms represent the restaurants in the example used above. Then, we have the four BanditSolver…() classes that represent the four different algorithms used to solve the bandit problem. In this basic version of the UML, all of the classes had the same basic variables and methods that later changed as I coded. Finally, we have the main class that initialises all of the classes sets up the arms and prints out the results of running each solver class.

A screenshot of a computer

Description automatically generated

Above is the final version of the UML that copies the exact layout of the final code for the proof of concept. The main structure change was in the solver classes, the original UML had two classes for selectArm() and updateArm(), however, I decided to break this down further into three classes, selectArm(), round() and runRound().

I chose to do this because I thought I was a better general fit for all the different algorithms. selectArm() contains the code for the actual algorithm within each arm. round() simply runs selectArm() and then increments the visits on the arm and adds the points to the total points. And runRound() checks whether the current round is equal to the total number of rounds before running another round.

Aside from the main structure changes more changes were made on a class basis, depending on what the algorithms needed. Such as the E variable for the E-Greedy class, firstRound() for the exploit only class and random was implemented for the explore only and E-Greedy class.

The Main was changed appropriately to accommodate the initialisation of all the classes and instances of the arms. Main was mostly used to change the values in the arms and other variables such as the total rounds and E for the tests done for the comparison of the efficiency of the algorithms under different states.

Results Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Round | Parameters | 1. Explore  Only | 2. Exploit Only | 3. E-Greedy | 4. UBC | Total Points |
| 1 | Total Rounds: 100  Arms: 4  Arm1: 10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  Base,  Normal parameters | Arm1: 24  Arm2: 23  Arm3: 35  Arm4: 18 | Arm1: 1  Arm2: 97  Arm3: 1  Arm4: 1 | Arm1: 83  Arm2: 8  Arm3: 6  Arm4: 3 | Arm1: 96  Arm2: 2  Arm3: 1  Arm4: 1 | 1: 645  2: 444  3: 857  4: 984 |
| Arm1: 26  Arm2: 24  Arm3: 23  Arm4: 27 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 84  Arm2: 5  Arm3: 4  Arm4: 7 | Arm1: 90  Arm2: 1  Arm3: 7  Arm4: 2 | 1: 700  2: 883  3: 932  4: 983 |
| Arm1: 21  Arm2: 31  Arm3: 25  Arm4: 23 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 88  Arm2: 3  Arm3: 6  Arm4: 3 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | 1: 649  2: 759  3: 964  4: 931 |
| Arm1: 23  Arm2: 25  Arm3: 22  Arm4: 30 | Arm1: 1  Arm2: 97  Arm3: 1  Arm4: 1 | Arm1: 68  Arm2: 6  Arm3: 2  Arm4: 24 | Arm1: 95  Arm2: 1  Arm3: 2  Arm4: 2 | 1: 660  2: 411  3: 842  4: 907 |
| Arm1: 23  Arm2: 21  Arm3: 25  Arm4: 31 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 92  Arm2: 3  Arm3: 2  Arm4: 3 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | 1: 662  2: 724  3: 918  4: 949 |
|  |  |  |  |  |  |  |
| 2 | Total Rounds: 500  Arms: 4  Arm1:10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  A lot more rounds run. | Arm1: 119  Arm2: 123  Arm3: 124  Arm4: 134 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 459  Arm2: 12  Arm3: 11  Arm4: 18 | Arm1: 495  Arm2: 1  Arm3: 1  Arm4: 3 | 1: 3522  2: 3777  3: 4672  4: 4640 |
| Arm1: 122  Arm2: 146  Arm3: 99  Arm4: 133 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 110  Arm2: 47  Arm3: 315  Arm4: 28 | Arm1: 492  Arm2: 2  Arm3: 1  Arm4: 5 | 1: 3448  2: 3827  3: 3569  4: 4693 |
| Arm1: 105  Arm2: 134  Arm3: 147  Arm4: 114 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 182  Arm2: 134  Arm3: 147  Arm4: 37 | Arm1: 478  Arm2: 2  Arm3: 2  Arm4: 18 | 1: 3488  2: 3712  3: 3622  4: 4698 |
| Arm1: 105  Arm2: 134  Arm3: 147  Arm4: 114 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 497 | Arm1: 182  Arm2: 134  Arm3: 147  Arm4: 37 | Arm1: 478  Arm2: 2  Arm3: 2  Arm4: 18 | 1: 3488  2: 3712  3: 3622  4: 4698 |
| Arm1: 132  Arm2: 105  Arm3: 128  Arm4: 135 | Arm1: 497  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 82  Arm2: 12  Arm3: 390  Arm4: 16 | Arm1: 492  Arm2: 2  Arm3: 2  Arm4: 4 | 1: 3564  2: 4581  3: 3567  4: 4777 |
|  |  |  |  |  |  |  |
| 3 | Total Rounds: 20  Arms: 4  Arm1:10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  A lot fewer rounds run. | Arm1: 9  Arm2: 3  Arm3: 4  Arm4: 4 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 17 | Arm1: 13  Arm2: 1  Arm3: 4  Arm4: 2 | 1: 126  2: 154  3: 149  4: 174 |
| Arm1: 7  Arm2: 3  Arm3: 1  Arm4: 9 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 3  Arm2: 2  Arm3: 14  Arm4: 1 | Arm1: 5  Arm2: 2  Arm3: 12  Arm4: 1 | 1: 179  2: 172  3: 136  4: 152 |
| Arm1: 6  Arm2: 5  Arm3: 5  Arm4: 4 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 1  Arm2: 1  Arm3: 17  Arm4: 1 | Arm1: 14  Arm2: 1  Arm3: 1  Arm4: 4 | 1: 120  2: 146  3: 144  4: 185 |
| Arm1: 3  Arm2: 5  Arm3: 6  Arm4: 6 | Arm1: 1  Arm2: 1  Arm3: 17  Arm4: 1 | Arm1: 8  Arm2: 2  Arm3: 2  Arm4: 8 | Arm1: 17  Arm2: 1  Arm3: 1  Arm4: 1 | 1: 125  2: 139  3: 150  4: 175 |
| Arm1: 4  Arm2: 6  Arm3: 8  Arm4: 2 | Arm1: 1  Arm2: 1  Arm3: 17  Arm4: 1 | Arm1: 16  Arm2: 2  Arm3: 1  Arm4: 1 | Arm1: 1  Arm2: 1  Arm3: 2  Arm4: 16 | 1: 119  2: 109  3: 158  4: 139 |
|  |  |  |  |  |  |  |
| 4 | Total Rounds: 100  Arms: 8  Arm1:10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  Arm5: 6, 4  Arm6: 3, 3  Arm7: 4, 2  Arm8: 1, 5  Double the number of arms. | Arm1: 16  Arm2: 15  Arm3: 12  Arm4: 15  Arm5: 13  Arm6: 9  Arm7: 4  Arm8: 16 | Arm1: 93  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | Arm1: 82  Arm2: 3  Arm3: 3  Arm4: 3  Arm5: 1  Arm6: 1  Arm7: 2  Arm8: 5 | Arm1: 90  Arm2: 2  Arm3: 1  Arm4: 3  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 570  2: 897  3: 887  4: 912 |
| Arm1: 16  Arm2: 12  Arm3: 8  Arm4: 14  Arm5: 16  Arm6: 11  Arm7: 13  Arm8: 10 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 93  Arm8: 1 | Arm1: 58  Arm2: 1  Arm3: 29  Arm4: 2  Arm5: 2  Arm6: 5  Arm7: 2  Arm8: 1 | Arm1: 91  Arm2: 1  Arm3: 1  Arm4: 3  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 552  2: 341  3: 818  4: 846 |
| Arm1: 10  Arm2: 9  Arm3: 9  Arm4: 16  Arm5: 16  Arm6: 10  Arm7: 16  Arm8: 14 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 93  Arm8: 1 | Arm1: 81  Arm2: 2  Arm3: 3  Arm4: 3  Arm5: 3  Arm6: 2  Arm7: 3  Arm8: 3 | Arm1: 74  Arm2: 2  Arm3: 2  Arm4: 16  Arm5: 3  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 558  2: 350  3: 873  4: 877 |
| Arm1: 13  Arm2: 15  Arm3: 11  Arm4: 15  Arm5: 13  Arm6: 12  Arm7: 11  Arm8: 10 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 93 | Arm1: 13  Arm2: 1  Arm3: 1  Arm4: 2  Arm5: 4  Arm6: 76  Arm7: 2  Arm8: 1 | Arm1: 77  Arm2: 6  Arm3: 4  Arm4: 6  Arm5: 3  Arm6: 1  Arm7: 1  Arm8: 2 | 1: 488  2: 185  3: 419  4: 863 |
| Arm1: 10  Arm2: 8  Arm3: 17  Arm4: 12  Arm5: 21  Arm6: 12  Arm7: 12  Arm8: 8 | Arm1: 93  Arm2: 1  Arm3: 1  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | Arm1: 2  Arm2: 1  Arm3: 2  Arm4: 88  Arm5: 3  Arm6: 1  Arm7: 1  Arm8: 2 | Arm1: 88  Arm2: 1  Arm3: 6  Arm4: 1  Arm5: 1  Arm6: 1  Arm7: 1  Arm8: 1 | 1: 512  2: 968  3: 708  4: 894 |
| 5 | Total Rounds: 100  Arms: 4  Arm1: 10, 4  Arm2: 20, 5  Arm3: 30, 3  Arm4: 40, 2  Greatly increase the distance between points given. | Arm1: 17  Arm2: 30  Arm3: 31  Arm4: 22 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 5  Arm2: 5  Arm3: 5  Arm4: 85 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2546  2: 3917  3: 3654  4: 3895 |
| Arm1: 28  Arm2: 21  Arm3: 26  Arm4: 25 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 4  Arm2: 2  Arm3: 6  Arm4: 88 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2419  2: 3914  3: 3744  4: 3901 |
| Arm1: 26  Arm2: 23  Arm3: 25  Arm4: 26 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 1  Arm2: 2  Arm3: 5  Arm4: 92 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2499  2: 3915  3: 3831  4: 3869 |
| Arm1: 25  Arm2: 23  Arm3: 29  Arm4: 23 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 2  Arm2: 2  Arm3: 3  Arm4: 93 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2502  2: 3907  3: 3812  4: 3870 |
| Arm1: 29  Arm2: 24  Arm3: 21  Arm4: 26 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 3  Arm2: 7  Arm3: 3  Arm4: 87 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 2407  2: 3875  3: 3733  4: 3881 |
|  |  |  |  |  |  |  |
| 6 | Total Rounds: 100  Arms: 4  Arm1: 10, 20  Arm2: 5, 15  Arm3: 7, 18  Arm4: 8, 21  Greatly increase the deviation. | Arm1: 24  Arm2: 29  Arm3: 28  Arm4: 19 | Arm1: 1  Arm2: 97  Arm3: 1  Arm4: 1 | Arm1: 74  Arm2: 4  Arm3: 4  Arm4: 18 | Arm1: 79  Arm2: 11  Arm3: 1  Arm4: 9 | 1: 1085  2: 271  3: 967  4: 643 |
| Arm1: 22  Arm2: 25  Arm3: 26  Arm4: 27 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 72  Arm2: 2  Arm3: 4  Arm4: 22 | Arm1: 1  Arm2: 2  Arm3: 1  Arm4: 96 | 1: 655  2: 1003  3: 883  4: 592 |
| Arm1: 20  Arm2: 27  Arm3: 28  Arm4: 25 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 57  Arm2: 23  Arm3: 4  Arm4: 16 | Arm1: 74  Arm2: 1  Arm3: 22  Arm4: 3 | 1: 769  2: 444  3: 915  4: 949 |
| Arm1: 35  Arm2: 22  Arm3: 17  Arm4: 26 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | Arm1: 4  Arm2: 3  Arm3: 3  Arm4: 90 | Arm1: 1  Arm2: 1  Arm3: 76  Arm4: 22 | 1: 614  2: 870  3: 875  4: 762 |
| Arm1: 24  Arm2: 26  Arm3: 29  Arm4: 21 | Arm1: 97  Arm2: 1  Arm3: 1  Arm4: 1 | Arm1: 46  Arm2: 3  Arm3: 5  Arm4: 46 | Arm1: 1  Arm2: 1  Arm3: 1  Arm4: 97 | 1: 624  2: 1125  3: 686  4: 884 |
|  |  |  |  |  |  |  |
|  | Parameters | Round 1 | Round 2 | Round 3 | Round 4 | Round 5 |
| Total Rounds: 100  Arms: 4  Arm1: 10, 4  Arm2: 5, 5  Arm3: 7, 3  Arm4: 8, 2  E = 10 | Arm1: 3  Arm2: 4  Arm3: 2  Arm4: 91  Total: 747 | Arm1: 3  Arm2: 3  Arm3: 91  Arm4: 3  Total: 662 | Arm1: 87  Arm2: 3  Arm3: 5  Arm4: 5  Total: 846 | Arm1: 52  Arm2: 3  Arm3: 2  Arm4: 43  Total: 832 | Arm1: 69  Arm2: 2  Arm3: 26  Arm4: 3  Total: 870 |
| E = 20 | Arm1: 26  Arm2: 4  Arm3: 11  Arm4: 59  Total: 799 | Arm1: 71  Arm2: 5  Arm3: 10  Arm4: 14  Total: 875 | Arm1: 90  Arm2: 3  Arm3: 3  Arm4: 4  Total: 917 | Arm1: 77  Arm2: 14  Arm3: 2  Arm4: 7  Total: 850 | Arm1: 41  Arm2: 4  Arm3: 51  Arm4: 3  Total: 749 |
| E = 30 | Arm1: 62  Arm2: 3  Arm3: 14  Arm4: 21  Total: 913 | Arm1: 63  Arm2: 7  Arm3: 22  Arm4: 8  Total: 766 | Arm1: 50  Arm2: 9  Arm3: 9  Arm4: 32  Total: 825 | Arm1: 60  Arm2: 5  Arm3: 7  Arm4: 28  Total:917 | Arm1: 72  Arm2: 13  Arm3: 9  Arm4: 7  Total: 840 |
| E = 40 | Arm1: 9  Arm2: 60  Arm3: 20  Arm4: 11  Total: 563 | Arm1: 68  Arm2: 11  Arm3: 10  Arm4: 11  Total: 780 | Arm1: 61  Arm2: 17  Arm3: 14  Arm4: 8  Total: 824 | Arm1: 65  Arm2: 7  Arm3: 13  Arm4: 15  Total: 821 | Arm1: 56  Arm2: 10  Arm3: 19  Arm4: 15  Total: 855 |
| E = 50 | Arm1: 36  Arm2: 20  Arm3: 7  Arm4: 37  Total: 790 | Arm1: 57  Arm2: 14  Arm3: 6  Arm4: 23  Total: 874 | Arm1: 45  Arm2: 15  Arm3: 23  Arm4: 17  Total: 758 | Arm1: 61  Arm2: 12  Arm3: 14  Arm4: 13  Total: 657 | Arm1: 32  Arm2: 47  Arm3: 12  Arm4: 9  Total: 672 |

Comparison of Algorithms

Explore only is an algorithm where an arm is selected at random for each round, meaning each arm is visited roughly equally overall. This is arguably the worst algorithm. In all the tests conducted above explore only collected around half of the maximum points within each test and all the runs. It beat a few of the other algorithms in test 6 where the deviation on the arms was greatly increased as the point values given would differ greatly from the main value meant for the arm, meaning some of the other algorithms would fail and explore only would work better simply because it explored all arms equally, however, the results for test 6 were wildly different for each run accounting for the wildly different values available.

Exploit only is the second worst algorithm. Its performance greatly depends on which arm gives it the best results during the first round. This algorithm's best performance was test 5 where the values for each arm differed greatly and the deviation was low, meaning it was easy for the algorithm to find the highest value arm and stick to it. It scored close to max points on each run. The worst performance for this algorithm was test 4 where more arms with low values and low deviance were added. The algorithm picked the wrong arm on multiple runs and scored extremely low.

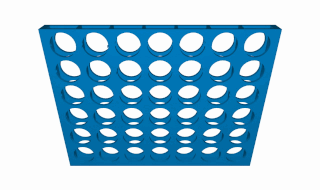
E-Greedy is the second-best performing algorithm. Its performance stayed consistent over all the tests conducted, however, this algorithm is highly dependent on the value used for E. As the second table shows, for the base values used, when E is equal to 10 the values are around 800, but when E is equal to 50 the values end up around 600. When E is 20-30 the best results are produced, giving points around 900. Of course, this is just with the base stats, which E value is best will also depend on the other values such as the number of arms, difference in point values and difference in deviance.

The Upper-Confidence-Bound (UCB) algorithm is the best-performing algorithm, over all the tests it was near-perfect scoring over all the runs. This is a great algorithm that weighs the values of each arm before choosing, not leaving it to random chance or external values. The test that this algorithm faltered on was test 6, where the deviance values were greatly increased, but even then, its lowest value was about half of the max, and all of the algorithms seemed to struggle with this test.

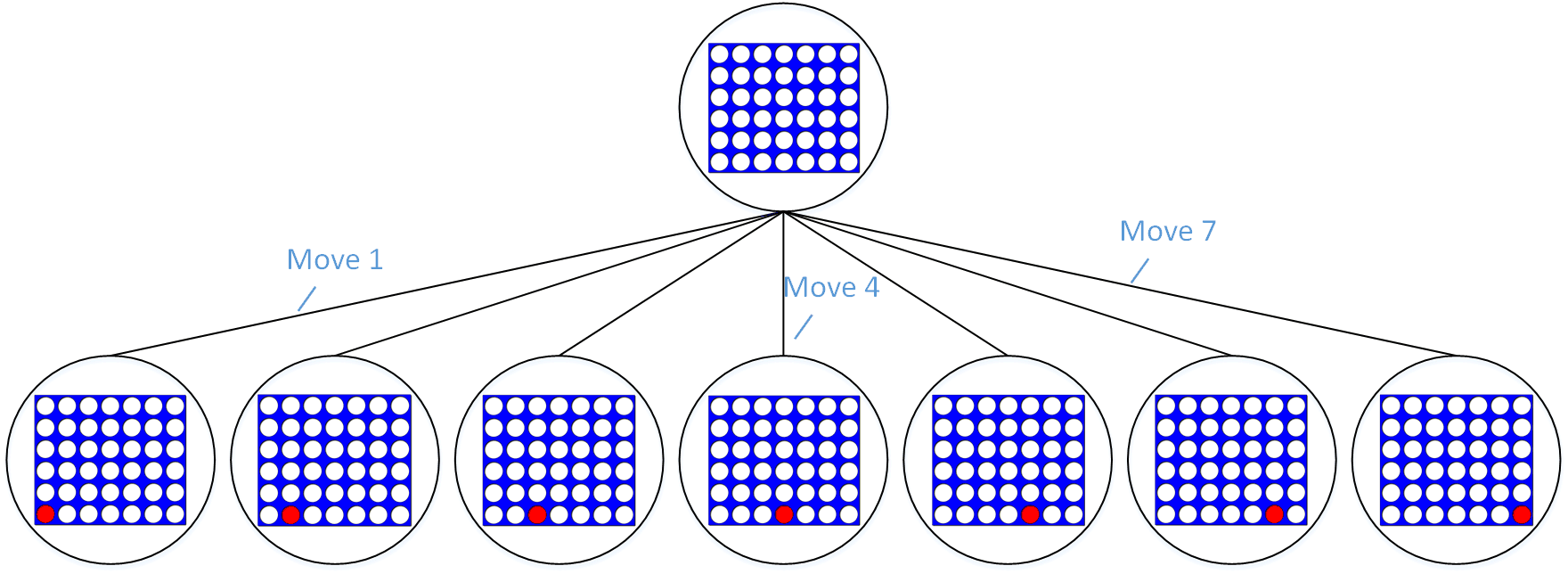
## Connect4

What is Connect4?

Connect4 is a two-player board game. The board is a grid of six rows and seven columns. Players can either be on the red team and play with the red-coloured discs or on the yellow team and play with the yellow-coloured discs. Players will take turns dropping one of their discs into a column, the disc will drop down to the lowest available slot in the column. The game aims to get four discs of your colour in a row either horizontally, vertically or diagonally. Draws can happen if the board is filled but not one player has four discs in a row.

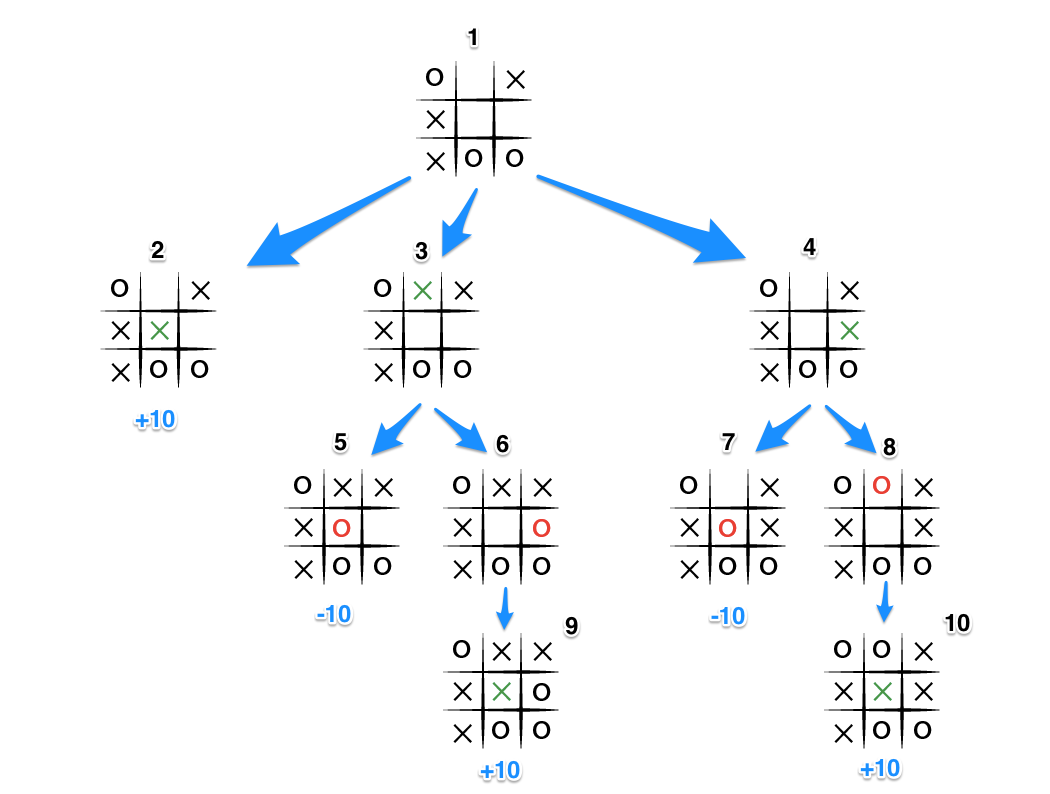


What are the game states of Connect4?



Connect4 is a game with a large playing space and has a massive, yet still finite state of game spaces it can be in. I chose this game for the project as it would be able to show off the algorithm's complex decision-making capabilities while still being fairly easy to code and work with. The initial game state starts with an empty board, each disc that is dropped into a column creates a new game space.

Games like tic-tac-toe have a much smaller playing area and therefore have a much smaller game tree.



Code

A screenshot of a computer

Description automatically generated

Above is the original rough UML I created to begin the code with. I had a class for the board, the player and the actual game, I also created a class for the game logic to keep these methods separate as I thought putting them in the main or board classes would add unnecessary functionality to those. The idea here was that the main class would run the flow of the game, initialising the players and asking for their names, player one having the red disc and player two having the yellow disc, then creating the board and displaying it, asking the players to drop discs and switching players, checking for a win or draw condition each turn.

A screenshot of a computer program

Description automatically generated

A few things have been changed while coding from the original UML. The names of the classes have been changed as this proof of concept code has been added with the bandit problem proof of concept code and I changed the class names to reduce confusion as to which class belongs to which program.

The next big change is moving the dropDisc() method into the board class, I decided to do this to make writing the method easier, as this method directly interacts with and changes the board I thought it would be better to have it in the board class. I removed the isMoveValid() method as it was only checking where the column input was in range and not full and so I added this code straight into the dropDisc() method instead.

# Bibliography

[ritvikmath](https://www.youtube.com/@ritvikmath)(2020) Multi-Armed Bandit: Data Science Concepts. Available at: <https://www.youtube.com/watch?v=e3L4VocZnnQ>

[ritvikmath](https://www.youtube.com/@ritvikmath)(2020) Best Multi-Armed Bandit Strategy? (feat: UCB Method). Available at: <https://www.youtube.com/watch?v=FgmMK6RPU1c>

[Nikolay Manchev](https://domino.ai/blog/author/nikolay-manchev)(2023) Reinforcement learning: The K-armed bandit problem. Available at: <https://domino.ai/blog/k-armed-bandit-problem>

[Nikolay Manchev](https://domino.ai/blog/author/nikolay-manchev)(2022) Semi-uniform strategies for solving K-armed bandits. Available at: <https://domino.ai/blog/semi-uniform-strategies-for-solving-k-armed-bandits>

samishawl(2020) Upper Confidence Bound Algorithm in Reinforcement Learning. Available at:<https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/>

Wikipedia contributors. (2024) Connect Four. Available at: <https://en.wikipedia.org/wiki/Connect_Four>

[Udacity](https://www.youtube.com/@Udacity)(2016) Monte Carlo Tree Search p1. Available at: <https://www.youtube.com/watch?v=onBYsen2_eA>

[Udacity](https://www.youtube.com/@Udacity)(2016) Monte Carlo Tree Search p2. Available at: <https://www.youtube.com/watch?v=4rVXxSD2DIM>

[Udacity](https://www.youtube.com/@Udacity)(2016) Monte Carlo Tree Search p3. Available at: <https://www.youtube.com/watch?v=EGN1KAjtNS4>

[Udacity](https://www.youtube.com/@Udacity)(2016) Monte Carlo Tree Search p4. Available at: <https://www.youtube.com/watch?v=2SeEQMfM3Ug&t=12s>

[Udacity](https://www.youtube.com/@Udacity)(2016) Monte Carlo Tree Search p5. Available at: <https://www.youtube.com/watch?v=H__-7zzNiPk&t=1s>