Università degli Studi di Salerno

Intelligent Agents Course

Final project

AI plays Hex

Hex is a two player game with simple rules. Since a universal winning strategy is only known for small boards (up to 9 x 9 cells), due to the complexity of the game itself (comparable to that of Chess and Go), the design of an intelligent agent capable to play to larger boards, is a particular interesting topic. This applies especially for AI courses students, like I am, who need to practise with game playing algorithms such as **mini-max** and $\alpha - \beta$ **pruning**. Furthermore, the complexity of these kind of games, allows to gain some experience in the field of **heuristic function design**, but also in a very useful **set of techniques** typically used to address these genres of problems.

Author
L. Schiavone

Professors
A. Della Cioppa
A. Marcelli

A.A. 2022-2023

1 Introduction to Hex

1.1 Game playing

Since it's a non-trivial activity, game playing is often considered as an interesting testbed for research fields like Artificial Intelligence. This is due to the fact that games are ideated from humans to make themself playing and therefore it's clear that games non-triviality is naturally associated to a *human degree of intelligence.*

Moreover, compared to other intelligent tasks, games have the advantage of beeing confinated to a specified **abstract** domain. Consequently they are characterized by clearly defined behavioural and also success rules.

1.1.1 Two player game definition

Since Hex is a two-player game, i.e. a game with only two players, we give a formal definition of two-player game here. A two-player game is a tuple $\{C, c_0, M, L, S\}$ where

- C is a set of board states, also known as *positions.*
- $c_0 \in C$ is the *initial position.*
- $M : C \to P(C)$ is the *successor function*, which given a position, lists all the positions in which the game state can go applying an action.
- $L \subset C$ is the set of *leaf positions*, i.e. the positions in which the game is considered finished $(M(l \in L) = \emptyset)$
- $S : L \to \mathbb{R}$ is the *score function* which associates a score to each game terminal position.

A game is a sequence of board states $\{c_0, c_1, ..., c_m\}$ where $c_i \in M(c_{i1})$ for $1 \leq i \leq m$ and $c_m \in L$. The *outcome of the game* is $S(c_m)$. In a two-player game the score of a position can be positive, negative or zero. We establish that one of the two player wants to maximize the score and the other wants to minimize it.

1.2 The game of Hex

Hex is a famous board game (a games which consists of placing pieces on a board), with simple rules (only pieces positioning rules) but a complex strategy behind. In fact winning strategies are only known for board sizes up to 7×7. Instead the game is commonly played on board of 10×10 or larger. The game is a particular case of a more general game, a graph coloring game, known as the Shannon switching game.

1.3 Brief history

Hex was first invented in 1942 by Piet Hein, a Danish engineer, but also a poet and mathematician. He called the game "Polygon" and published a series of articles on it. The game was rediscovered John Nash, a mathematics graduate student in 1948 at Princeton. Here its name became "Nash". The name "Hex" was introduced by Parker Brothers, who put it on the market in 1952. In the 90' a Hex playing community emerged on the Internet and Hex continued to gain popularity, until the publication of the first book dedicated to the game, written by Bro00. Hex was also included in the Olympic list of games to feature at the 2000 Computer Olympiad.

1.4 The rules

Hex is played on a rhombic-shaped board, made of hexagons. When we talk about a 7x7 Hex we are saying that each side of the board is 7 hexagons long. However the game can be played on baords of any sizes. There are two players, commonly white and black, or also red and blue. Also the board has two white (or red) sides (edge cells), which are one in front of the other, and two black (or blue) sides, also one in front of the other. Note that the four corner edge cells belong to both the players.

Play proceeds in rounds.The two players take turns placing a piece of their colour on an empty cell. There is no standard convention on which colour gets the first move. A player wins the game by connecting his edge cells border borders with a chain of his pieces, before the other player.

These are the only rules (excluding the swap rule, which, for sake of simplicity, I'm not considering in this work)

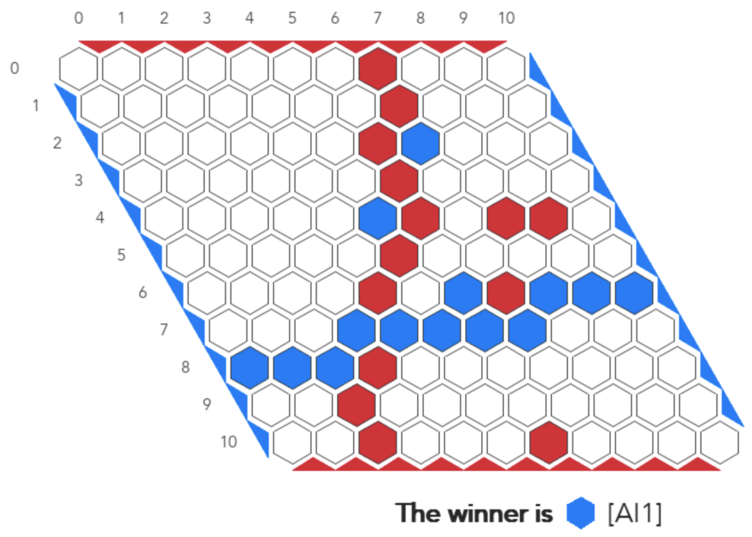

Figure 1: Hex board and a game state

1.5 Analysis of the game

Even if Hex rules are very simple, Hex strategy is a surprisingly complex. Let's analyze some properties of this game.

- Hex is a **zero-sum game**, meaning that an increase in one player's payoff corresponds to a decrease in another player's payoff (sum of payoffs is zero; A wins means B loses).
- In Hex, **the first player wins**, obviously playing rationally, that is not true for a human, because of the search complexity of the game itself. This was proved by John Nash. In practice, with human players, starting for first is a large advantage. That's why sometimes a equalization move is played, which is called the *swap rule*: after the starting played has played his move, the second player then decides to take or not possession of this move. The rationale is to make the first player play a move which is not too much strong, so that the second player won't change it. Anyway, for sake of simplicity I don't consider it in this discussion.
- The previous proof is very simple, but it is only a proof of existence. *No general winning strategy is known.* Even and Tarjan proved that **the game is PSPACE-complete**. In fact, Hex can be solved a simple depth-first search, i.e. the minimax algorithm. The amount of memory needed for an exhaustive search is $O(n^2 \log n)$, where n is the number of vertices of the game graph and this is PSPACE-complete. [1]
- Hex 11x11 has **121 board positions** and a **average branching factor** $b = 96$ - lower than Go (250) but higher than Chess (35).

1.6 Objectives and motivations

The objective of this project is to put in practise the application of search and evaluation techniques to a game, Hex, which shows challenges to both of these factors.

The hardness in searching in Hex can be identified with the large branching factor. About the difficulty with evaluation function, it can be seen that is not immediately clear how to identify concepts to compute that are strategically relevant to Hex.

Playing of games with large branching factors put the designer in front of the dilemma of choosing between a more exhaustive search or a more selective one. Clearly there are risks of unreliable results using both strategies.

Motivations to study Hex resides in the difficulties of finding a computable solution, as already said above, which contrapposes with the simplicity of the rules, which allows to think how these simple rules hide an unexpectedly deep and rich strategy. As Jack van Rijswijck says in his thesis, the game is a prime example of *"a minute to learn, a lifetime to master"*.

[1] A decision problem is PSPACE-complete if it can be solved using an amount of memory that is polynomial space complex and if every other problem that can be solved in polynomial space can be transformed to it in polynomial time. Note that PSPACE is known to be more difficult of NP class.

2 Playing by search

2.1 Mini-max and alpha-beta pruning

The basic algorithm which an agent uses to choose the best move to play in a turn two-player game consists in the exploration of all the game search tree, starting in the current position (the current game state). This is called mini-max algorithm. The idea is to choose the best move knowing that also the adversary will do the his best move. In two-player zero-sum games, the mini-max solution is exactly the *Nash equilibrium.*

The algorithm is recursive: at each level of the three the algorithm plays the role of one of the two players. Starting from the root (the current game state) with the role of the player who is choosing the move, the algorithm choose his best move between all the possible available moves. At the next level the algorithm will do the same, except for the fact that now he is choosing the best move according the adversary (since he knows that the adversary will play it), until the leaves of the tree are reached. These are the terminal game states. The value associated to these moves are their scores.

```
def minimax(node, maximize, depth=inf):
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximize
        value := -inf
        for each child of node do
            value := max(value, minimax(child, depth-1, FALSE))
        return value
    else
        value := +inf
        for each child of node do
            value := min(value, minimax(child, depth-1, TRUE))
        return value
```

The number of nodes to be explored usually increases exponentially with the number of *plies*, which are the number of explored levels. The number of nodes to be explored for the analysis of a game is therefore approximately the branching factor raised to the power of the number of plies. It is therefore impractical to completely analyze games such as Hex using the mini-max algorithm.

The performance of the basic mini-max algorithm can be improved dramatically, without affecting the result, using alpha–beta pruning. Alpha–beta pruning consists in the mini-max algorithm, with the benefit that portions of the game search tree are not visited - they are pruned - since it is not useful to visit them. This optimization reduces the effective depth to a bit more than half that of the basic mini-max if the nodes are evaluated in a approximately optimal order.

```
def a_b_pruning(node, maximize, depth=inf, a=-inf, b=inf):
    if depth == 0 or node is terminal then
        return the heuristic value of node
    if maximize
        value := -inf
        for each child of node
            value := max(value, a_b_pruning(child, depth-1, a, b, FALSE))
            if value > b
                break (* b cutoff *)
            a := max(a, value)
        return value
    else
        value := +inf
        for each child of node
            value := min(value, a_b_pruning(child, depth-1, a, b, TRUE))
            if value < a
                break (* a cutoff *)
            b := min(b, value)
        return value
```

2.2 Enhancements

Even if the alpha-beta pruning is used, for game with a large average branching factor, like Hex, the exploration of the game search tree until reaching his leaves is not possible. The choice of the move to play will require too much time, also on very powerful machines. That's why it is common to visit the tree until a certain depth is reached, in a way that the visited nodes are not so much. But this requires to evaluate nodes that are non terminal. This is done injecting some **heuristics** i.e. a priori information about the game. In particular, in this specific case, the **heuristic function** is used to evaluate how good, in terms of winning, is a move.

Other enhancements, which also concerns with heuristics injection, is the use of **opening moves** i.e. moves that are considered good as first moves, according previous game plays.

Still other heuristic pruning methods can be used - e.g. transposition tables - but we don't use them.

3 Building an Hex AI player

3.1 The search algorithm

The intelligent agent I realized uses alpha-beta pruning algorithm, with a specified maximum search depth. Literature is full of material about Hex playing, therefore I collected a set of node evaluation and node ordering functions and I tested them, to understand

- their ability to win (how many times they win)
- the time they take to play
- the difficulty they encounter to win (how many moves are required to win)

3.2 Proposed node evaluation functions

Node evaluation functions are a fundamental part of a heuristic game search algorithm. These are the ones that I decided to shortlist.

3.2.1 Connected Heuristic

The idea is to count *how many pieces of the player are connected.* Therefore a higher number of connected pieces corresponds to a higher value of the evaluation for the player.

To evaluate the node value of the game state we have to take in charge the value of the heuristic function $k_{p_i}(game)$ for both player p_1 and p_2. We subtract them:

$$J(game) = k_{p_1}(game) - k_{p_2}(game) \tag{1}$$

3.2.2 Shortest Path Heuristic

Connected heuristics is too much simple: it doesn't take in charge how much good the connection between player pieces. Here the idea is to count *the least number of pieces required to connect p_i's edge pieces.* The computation is straightforward: first the shortest path between the player's edge pieces is computed, then we count the number of unoccupied cells lying on the shortest path.

The total heuristic value is:

$$J(game) = k_{p_2}(game) - k_{p_1}(game) \tag{2}$$

3.2.3 Y-Reduction Heuristic

The Game of Y, by Craig Schensted, inspired an interesting heuristics for Hex. This is because Hex is a special case of it. We'll use a method called *Y reduction.* The game of Y is played on a triangular board composed by hexagons. The goal is to establish a chain that connects all three sides of the board. Hex is a special case of Y.

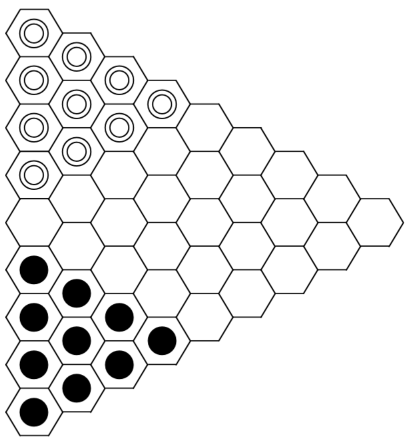

Figure 2: Hex board and a game state

Playing Y in this diagram is equivalent to playing Hex in the empty region.

Consider a Y board completely filled with pieces. This board can be reduced to a size n-1 board , where each cell on the n-1 board corresponds to a group of three neighbouring cells on the n board. In the n1 board the colour of a cell is determined by the majority of the colours of the three corresponding cells on the bigger board. It happens that he colour of the lone piece on the size-1 board indicates who the winner is on each of the preceding boards. This is because every path of one colour is preserved by the reduction step. If a path touches one side, then the corresponding chain in the reduced board does. A winning path will therefore produce corresponding winning chains in all of the smaller boards, including the final trivial size-1 board on which a winning path is the single piece of the board.

A heuristic evaluation method for Y and for Hex can be designed extending micro reduction to empty cells [3]. One way to do this is to assign a ownership probability to each cell on the board. Cells that already contain pieces have probability 1 or 0 according the owner. Empty cells have probability 1/2. Therefore the probability of owning at least two of the cells of a triangle can be computed However Y is not a game of chance. Moreover, the probabilities we are considering are not independent. Anyway this method can provide a good heuristic evaluation of a Y position.

In practice, the reduction method generates a pyramid of values, starting with originals cells of the game board and going down to the single value of the size-1 board that represents the final evaluation.

3.2.4 Two Distance Heuristic

Using shortest path to compute an heuristic value for an Hex position is not the best choice so far. This is because it doesn't consider alternative paths: if the adversary blocks the shortest path on their next move then the metric says nothing about how well-connected the rest of the board is. Here we present an alternative metric that implicitly takes this into account. Such a metric can be achieved using a distance measure called the **two-distance** [2]. In brief it is defined as follows: if u and v are neighbours then d(u, v) = 1, but otherwise d(u, v) = 1 + d(u", v), where u" is a neighbour of u with the minimum distance to v after first removing u' from its neighbours. Note that here neighbours means that either they are adjacent or adjacent to the paths of the considered player.

The total heuristic value is the difference of the two-distances between the player's edges:

$$J(game) = k_{p_2}(game) - k_{p_1}(game) \tag{3}$$

3.2.5 Max-Flow Heuristic

The following heuristics is one of the ones that uses, for his computation, a network representation of the hex board. In particular a weighted graph for each player is built. This graph has edges between two nodes if they are adjacent and they are either empty cells of cells occupied by the considered player. In my implementation the weight of the edges is infinite if the two cells belong to the considered player, else it's 1. Furthermore all the edge nodes are represented as unique source and sink nodes, and source-sink edges are infinite weighted. The heuristic value for a player consists in executing a max-flow algorithm on the player weighted graph. This can be done, for example, with the Ford-Fulkerson algorithm.

The total heuristic value is the difference of the two max-flows for the two players:

$$J(game) = k_{p_1}(game) - k_{p_2}(game) \tag{4}$$

3.2.6 Resistance Heuristic

Another heuristics using a network representation of the Hex board is the one used by Claude Shannon to solve this game in [1]. Here the idea is opposed to the max-flow heuristics: each edge of the graph constitutes a resistance between cells. Then the equivalent resistance for each player can be computed. In my implementation this is done by a circuit simulator library. Note that this computation is time expensive, even more than the max-flow one. Shannon only used this heuristics building a physical circuit!

The total heuristic value is the difference of the two equivalent resistances for the two players:

$$J(game) = k_{p_2}(game) - k_{p_1}(game) \tag{5}$$

3.3 Proposed node ordering functions

As said before, alpha-beta pruning optimization is very effective when the nodes of the search three are evaluated in a approximately optimal order. That's why I searched also for node ordering heuristics.

3.3.1 Random Order Heuristic

The simple ordering heuristics is to consider the nodes at random. This doesn't help in the search so much, but it's better than having a prefixed order on nodes, since this will make the game predictable in some cases.

3.3.2 Charge Heuristic

This is a position ordering heuristic that allows to evaluate how good are the moves. Clearly also the above node evaluation functions can be used, but they are too slow and we are using node ordering just to speed up the search.
I found this on the Github of @rjewsbury, unfortunately I didn't found any paper talking about it. Anyway, the idea is to treat stones as positive and negative charges, and try to find saddle points in the electrical field represented by these charges. These points are computed using physics analytic models.

3.4 Opening moves

Another improvement I applied to my Hex intelligent player is the use of opening moves. In reality, in Hex without the swap rule, the best move is known to be a central cell. Therefore the first move for the first player is always the central hexagon in my algorithm. However, using the swap rule, the choice of good opening moves becomes more interesting: a good opening move will prevent the adversary to swap it. This can be done playing a move which is good but not too much to be swapped.

3.5 Experiments and results

To evaluate the performances of these heuristics with the game search algorithm I ran several experiments opposing different AIs. For each experiment I recorder:

- the number of wins
- the number of wins when playing as first player
- the average and the standard deviation of move time during each play
- the number of moves required to terminate the play.

Also I ran, for each experiment, a Mann-Whitney test to conclude if the agents have same or different performances in terms of move time.

Note that since I used, in each of the experiment, a stochastic component (the node random ordering), I ran the experiments for N trials, in a way that I can average all the results I obtained. Clearly, since this is an exam project and not a thesis, an due to my limited computational resources, N is not too high. Anyway the results I obtained are so explicit to allow making robust considerations, despite the not so high N.

3.5.1 Experiment #1: Connected-2-R VS Shortest Path-2-R (11x11)

In this task I made the following two AIs to face, playing on a 11x11 board: alpha-beta pruning with max-depth = 2 using ConnectedValueHeuristic and RandomNodeOrderingHeuristic VS alpha-beta pruning with max-depth = 2 using ShortestPathValueHeuristic and RandomNodeOrderingHeuristic.

The experiment is done on N = 30 trials.

Table 1: Statistics for the test

	ConnectedValueHeuristic	**ShortestPathValueHeuristic**
N of wins	0	30
N of wins as starting player	0	15
E[MoveTime]	0.2668	0.30299
Std[MoveTime]	0.21649	0.15837
Avg move times with Mann-Whitney test	same performances!	
Number of moves to win	24.3	

From the results I deduced the following. Shortest Path Heuristic is too much stronger than Connected one. Even when the latter starts for first it loses. Also, even if Connected heuristics is a little bit faster, both heuristics have statistically the same move time performances. Note also that the number of moves required to win is a but above the minimum: that means the game is very unbalanced!

3.5.2 Experiment #2: Shortest Path-2-R VS YReduction-2-R (11x11)

In this task I made the following two AIs to face, playing on a 11x11 board: alpha-beta pruning with max-depth = 2 using ShortestPathValueHeuristic and RandomNodeOrderingHeuristic VS alpha-beta pruning with max-depth = 2 using YReductionValueHeuristic and RandomNodeOrderingHeuristic.

The experiment is done on N = 30 trials.

Table 2: Statistics for the test

	ShortestPathValueHeuristic	**YReductionValueHeuristic**
N of wins	23	7
N of wins as starting player	13	5
E[MoveTime]	0.4762	0.92552
Std[MoveTime]	0.35577	0.79969
Avg move times with Mann-Whitney test	different performances!	
Number of moves to win	38.8	

From the results I deduced the following. Shortest path is also better than Y-Reduction: it won about 75% of the times and the wins for Y Reduction happens mostly when it is the first to play. About move time Y Reduction takes in averafe about twice the time of Shortest path (Mann-Whitney test confirms they have different performances). In this case the plays are more balanced, with an average move to win of 39.

3.5.3 Experiment #3: Shortest Path-2-R VS Two Distance-2-R (11x11)

In this task I made the following two AIs to face, playing on a 11x11 board: alpha-beta pruning with max-depth = 2 using ShortestPathValueHeuristic and RandomNodeOrderingHeuristic VS alpha-beta pruning with max-depth = 2 using TwoDistanceValueHeuristic and RandomNodeOrderingHeuristic.

The experiment is done on N = 30 trials.

Table 3: Statistics for the test

	ShortestPathValueHeuristic	**TwoDistanceValueHeuristic**
N of wins	1	29
N of wins as starting player	1	15
E[MoveTime]	1.02298	0.96353
Std[MoveTime]	0.69462	0.62549
Avg move times with Mann-Whitney test	same performances!	
Number of moves to win	29.6	

From the results I deduced the following. Two distance heuristics won approximately all the times and the only time in which SP won is when it starts as first player. Move time performances are the same. In this case the number of moves to win is about 30 (between the 2 previous experiments).

Therefore, I take the most promising heuristics so far and I test it against the other ones I proposed.

3.5.4 Experiment #4: Two Distance-2-R VS Max Flow-2-R (8x8)

In this task I made the following two AIs to face, playing on a 8x8 board: alpha-beta pruning with max-depth = 2 using TwoDistanceValueHeuristic and RandomNodeOrderingHeuristic VS alpha-beta pruning with max-depth = 2 using MaxFlowValueHeuristic and RandomNodeOrderingHeuristic.

The experiment is done on N = 20 trials.

Table 4: Statistics for the test

	TwoDistanceValueHeuristic	**MaxFlowValueHeuristic**
N of wins	19	1
N of wins as starting player	10	1
E[MoveTime]	0.24377	3.54548
Std[MoveTime]	0.14609	2.1812
Avg move times with Mann-Whitney test	different performances!	
Number of moves to win	20.65	

From the results I deduced the following. Also in this case Two Distance is the best. Again Max Flow only won 1/20 times - which was when it started as first player. Performances are different: Max-flow heuristics based AI takes about 10x the time of Two Distance. In this case the heuristic strength ration is so unbalanced: about only 20 moves are required on average to win.

3.5.5 Experiment #5: Two Distance-2-R VS Resistance-2-R (8x8)

In this task I made the following two AIs to face, playing on a 8x8 board: alpha-beta pruning with max-depth = 2 using TwoDistanceValueHeuristic and RandomNodeOrderingHeuristic VS alpha-beta pruning with max-depth = 2 using ResistanceValueHeuristic and RandomNodeOrderingHeuristic.

The experiment is done on N = 20 trials.

Table 5: Statistics for the test

	TwoDistanceValueHeuristic	**ResistanceValueHeuristic**
N of wins	18	2
N of wins as starting player	10	2
E[MoveTime]	0.2112	46.0627
Std[MoveTime]	0.10933	16.5942
Avg move times with Mann-Whitney test	different performances!	
Number of moves to win	20.2	

From the results I deduced the following. The same considerations of Max-Flow heuristics apply in this case, with the only difference of even greater average move time.

3.5.6 Experiment #6: Two Distance-2-C VS Two Distance-2-R (11x11)

In this task I made the following two AIs to face, playing on a 11x11 board: alpha-beta pruning with max-depth = 2 using TwoDistanceValueHeuristic and ChargeNodeOrderingHeuristic VS alpha-beta pruning with max-depth = 2 using TwoDistanceValueHeuristic and RandomNodeOrderingHeuristic.

The experiment is done on N = 30 trials.

Table 6: Statistics for the test

	TwoDistanceValueHeuristic	**TwoDistanceValueHeuristic**
N of wins	19	11
N of wins as starting player	11	7
E[MoveTime]	0.587	0.89951
Std[MoveTime]	0.43694	0.46589
Avg move times with Mann-Whitney test	different performances!	
Number of moves to win	43.0	

From the results I deduced the following. Two Distance heuristic combined with Charge Ordering heuristic won about the 60% of the times. Time performances are different: node ordering reduces of about %40 the search time. And this also results in a better win performance.

3.5.7 Experiment #7: Two Distance-2-C VS Two Distance-3-C (8x8)

In this task I made the following two AIs to face, playing on a 8x8 board: alpha-beta pruning with max-depth = 2 using TwoDistanceValueHeuristic and ChargeNodeOrderingHeuristic VS alpha-beta pruning with max-depth = 3 using TwoDistanceValueHeuristic and ChargeNodeOrderingHeuristic.

The experiment is done on N = 20 trials.

Table 7: Statistics for the test

	TwoDistanceValueHeuristic	**TwoDistanceValueHeuristic**
N of wins	20	0
N of wins as starting player	10	0
E[MoveTime]	0.1179	2.10226
Std[MoveTime]	0.07674	1.44927
Avg move times with Mann-Whitney test	different performances!	
Number of moves to win	38.5	

From the results I deduced the following. In this case we increased the search depth, but the results are interesting! The faster agents - which uses only two levels of search - overperforms the the slower agents - which uses 3 levels of search. How this can be explained? When we increase the search depth we are making the model of our intelligent agent more complex. Note also that heuristics are simply previous knowledge of the game injected into the system. The consequence of complexity is that the algorithm becomes more focused on specific sequences of moves that lead to favorable outcomes. This can make the algorithm less capable of handling unforeseen strategies from the opponent - which, in a certain sense, the shallower search adversary produces.

Also note that search depth is an hyper-parameter of the agent. Therefore, a good design strategy, if I had the computational resources, would be to grid-search the search depth. In this case I refer to design choices found on literature simply for sake of simplicity since *this is not a research paper!* Therefore I can conclude that the two distance-based agent with charge ordering and a max depth of 2 is the agent which performed the best in the limited setting I imposed for this work.

3.6 Future enhancements

Clearly it would be interesting to conduct large-scale testing by exposing the agent to a variety of players, both human and AI-controlled, for example the best known players for Hex like Hexy and Queenbee. This will help evaluate my agent performances in a broader context.

Enhancements to my agents can include the introduction of the swap rule, which will allow to study the effectiveness of the best known opening moves, and also the use of different pruning approaches to game search.

References

[1] C. E. Shannon. Computers and automata. *Proceedings of Institute of Radio Engineers 41: 1234-1241*, 1953.

[2] Drew Mellor Stephan K. Chalup and Fran Rosamond. The machine intelligence hex project. 2005.

[3] Jack van Rijswijck. Search and evaluation in hex. 2002.