AI-1 Systems Project Report : Play FAUhalma : A variant of Sternhalma

Akash Tambe

06 December 2023

Abstract

This report demonstrates the idea of playing a game called FAUhalma which is derived from another game known as Sternhalma or popularly known as Chinese Checkers. We will explore techniques on how to efficiently play this game when the initial board configuration is known. The main objective of any player in this game is to relocate all its pegs into the opposite corner present in front of the starting position before any other player relocates its pegs.
This report not only highlights the mechanics of the game, but also states the strategies to win the game eventually. The proposed approach states following an adversarial search to compute the initial as well as the subsequent moves of the player. Also, we aim to minimize the number of moves made so as to increase the probability of the game resulting in a win for our player.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background

In the domain of Artificial Intelligence(AI), we experience that a non-human entity, possibly an agent works intelligently similar to a human being. As an agent, it is expected to perform a task with the help of its sensors which help the agent perceive the environment. Further, the agent can perform the appropriate actions based on what state the environment is in at a given time frame.
Mainly, we can see the world of AI based on three pillars, namely Symbolic AI, Sub-Symbolic AI and their Applications. Amongst all of them, the symbolic AI part focuses on understanding and manipulation of symbols to represent the environment, the objects and their relationships between each other which eventually define human-like cognitive processes. In this report, we mainly explore the techniques revolving around symbolic AI to play the game. The search algorithms play a vital role amongst all techniques which help the agent in traversing and perceiving the environment and accordingly interact between them.
By understanding the foundations of search algorithms in the realm of symbolic AI implies developing cognitive comprehension about intelligent reasoning and effective decision making in agents. We explore this better using an example of a game play agent in a grid board environment that has location sensors which identify the locations of the pegs of every player.

1.2 Research Question

The research question helps everyone identify the main problem which is to be addressed. It also shapes the expectation of the reader making it more relevant and engaging. In context of this particular assignment report, it could be - "How can search algorithms be effectively adapted to develop an intelligent and competitive agent to play a board game?"

1.3 Contribution

The contribution sections helps everyone to set an understanding about what to expect further in the report. Also, it clearly states the individual contribution done by the author. The following contributions have been done in this report:

- A clear way of representing the non-rectilinear grid
- Implement and evaluate interactive search methods to find a move of an agent in order to maximize the win for the agent.

1.4 Overview

Initially, in chapter 2, the problem statement is described in much more detail. There are several sub-sections in the second chapter. The first section describes the actual question which we are trying to solve. Further sections describes the supporting information to understand the problem statement even more better. They describe in detail the co-ordinate system used, the moves and their representation which will be followed throughout the report.
The third chapter then focuses on the methodology of the implementation. This chapter also spans over multiple sections and sub-sections. Initially, it describes the server object used. Secondly, it describes different search methods which can be used and how are they relevant in solving this particular problem at hand. The proceeding section lists the technologies and the frameworks used when implementing the solution for the research question.
Furthermore, as the name of the section suggests, we describe an end-to-end algorithm which tries to give an overview of the full solution. The fourth chapter discusses about the results, findings and evaluations recorded throughout the experiment. The conclusion chapter summarizes the key findings and draws relevant implications and inferences. The last chapter mentions all the bibliography and cites all the references.

Chapter 2

Problem Description

This chapter highlights the context of the problem and the information associated with it.

2.1 Problem Statement

We try to build an intelligent agent which can interact with other players while playing a board game, in this case, FAUHalma. This game is played on a star-shaped board game wherein each player has same number of pegs on the board. The goal of each player is to successfully move the pegs in such a way that all of pegs reach the destination assigned to them, which we call home. The first one to do so results in winning the game. Figure 2.1 indicates a sample board configuration. Namely, A, B and C are the various players playing the game and their respective homes are highlighted. Pegs of player A and the respective home is shown using a pink colour, player B and the respective home is shown using a blue colour and player C is shown using a yellow colour. Alternatively, if there are only two players, we can also use a rhombus-shaped board to play the game. Figure 2.2 indicates a sample configuration of a 2-player variant. A similar colour scheme is followed while representing the modified variant as well. We can see that initially, all of player B's pegs are in the home of player A. Inversely, all of player A's pegs are in the home of player B.
These figures represent a configuration of the pegs and state the initial positions of all players. The pegs of a player are placed in the opposite direction of their respective homes. In the further sections, we will see how the co-ordinates of the board are represented.

2.2 Co-ordinate System

In order to represent this board, we make use of non-rectilinear co-ordinate assignment. A non-rectilinear grid refers to a mesh which does not consist an arrangement of regular rectangular cells. Unlike a rectilinear grid where

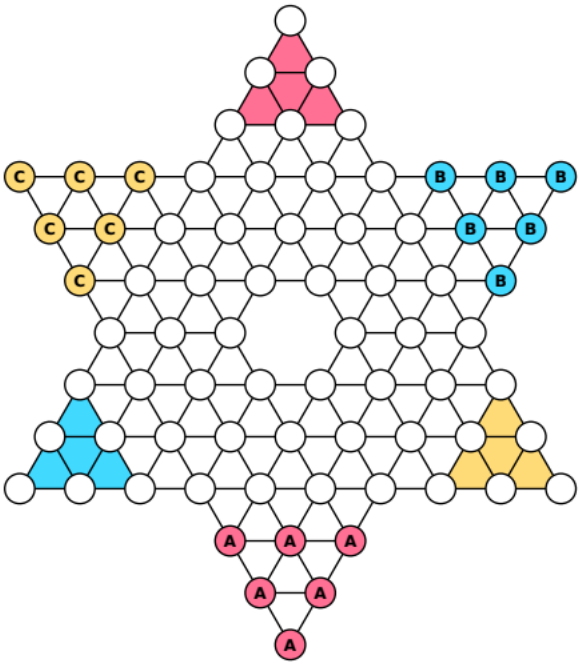

Figure 2.1: Board Configuration - 3 Players

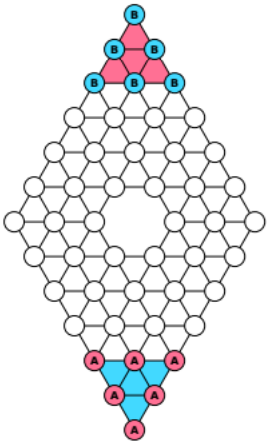

Figure 2.2: Board Configuration - 2 Players

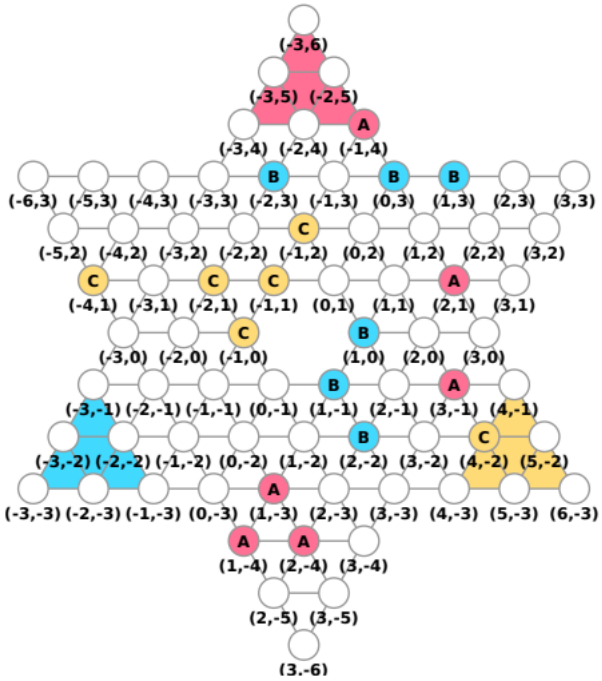

Figure 2.3: Co-ordinate System of Board

co-ordinate axes are orthogonal, a non-rectilinear grid have them aligned at a different angle and the co-ordinates are also aligned accordingly.
Figure 2.3 shows how the co-ordinates are assigned for a 3 player variant board configuration using a non-rectilinear co-ordinate assignment. For instance, home co-ordinates for player A are {[-3,6],[-3,5],[-2,5],[-3,4],[-2,4],[-1,4]}.
It is also important to note that the origin [0,0] is removed when representing the board. Legally, there is no way any player's peg can access that location at any given point of time.

2.3 Moves

To specify the dynamics of any game, we need to define the state of the game at a given point of time. Defining the state includes specifying all the peg positions and their respective home positions as well. In order to achieve this, we need to specify all legal and valid moves a player can play. Totally, there are four possible moves which are listed and explained below.

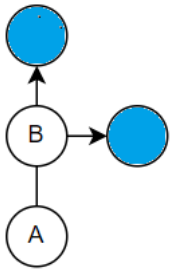

Figure 2.4: Simple Move

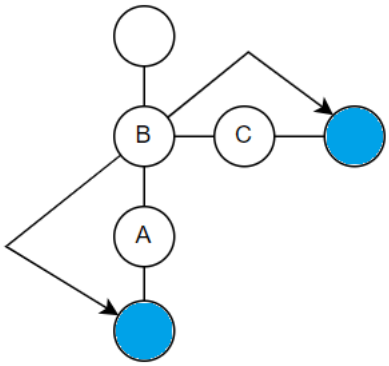

Figure 2.5: Simple Hop

- **Simple move** : The simple move indicates moving the peg to an adjacent position in any direction. The pre-condition for the move to happen is that the position where the peg is to be moved should be empty. Figure 2.4 shows a sample of a simple move. It indicates that peg of player B can be moved to the locations highlighted with the blue colour but cannot be moved to the location where peg of player A is placed.
- **Simple hop** : If any of the adjacent positions are occupied with a peg, the peg we are trying to move can jump over the adjacent peg and can be placed behind the adjacent pegs position as long as it is empty. This is more elaborately shown in the Figure 2.5. Peg of player B can be moved to locations highlighted with blue colour as their adjacent positions are not empty.
- **Hop Chain** : We can visualize this move as a series of simple hops. There is no limit on number of hops made in this move as long as it is a valid

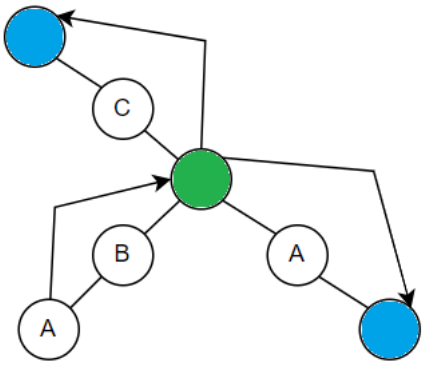

Figure 2.6: Chain hop

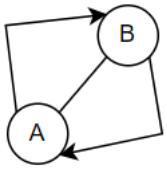

Figure 2.7: Swap move

move. Figure 2.6 illustrates a possibility of this move. The blue locations are the end locations where a peg can be placed ultimately, whereas the green location indicates intermediate positions of the peg.

- **Swap move** : If any adjacent position is occupied by another player's peg, we can interchange the positions of the two pegs. Importantly, this move can only be made when both of the pegs are in the home area of moving peg. Figure 2.7 explains this move more intuitively. Assuming that both pegs A and B are in the home area, we can say that their positions can be interchanged.

2.4 Representation of Moves

The moves are represented as a list of pairs wherein the first pair indicates the location of the peg you want to move and the second pair indicates the end location of the peg. If there is a hop chain move, it will also contain all the intermediate locations the peg moves to.

- **Simple move** : [1, -3], [1, -2]
- **Chain hop** : [-1, 4], [1, 2], [3, 0], [3, -2]

In case of a simple move, [1, -3] indicates that there is a peg at that location which is to be moved which we can call as the source location. Further, [1, -2] indicates the location where the peg is to be moved which can be named as destination location.
For the chain hop move, [-1, 4] is the source location and [3, -2] is the destination location. All the location which are mentioned in between are the intermediate locations the peg at source location visits. In words, the peg at [-1, 4] moves to [1, 2], further to [3, 0] and reaches the final destination location at [3, -2].

Chapter 3

Methodology

This chapter highlights the methods, algorithms and techniques used to solve the problem.

3.1 Server

For an interactive behaviour, we need a server object which can interact with agents of all players. The server is able to read the agent input, modify the respective peg positions as requested by the agent, and finally return a response back to the agent stating the current configuration of all pegs on the board. Before modifying the peg positions, the server also makes sure whether the move requested by the agent is a valid and legal move or not. If yes, the server modifies the peg locations and returns a response to the agent. If not, it returns an error message to the agent stating it is not a valid move. If there are no moves requested by the agent, the server always notifies back the agent that there is no change in the board configuration as a response.
To interact with the server, we need to explicitly notify it with the details which include :

- **Name** : Indicates name of the agent.
- **Environment** : Indicates whether the board should be configured according to a 2-player or a 3-player variant.
- **Password** : Indicates the password of the agent.
- **Server URL** : Indicates a https location where the server is hosted.

All the above information is sent to the server in a JSON Object when the agent tries to interact with the server. We used the following JSON Object :

```
{
 "agent": "Agent259",
 "env": "ss23-2.2-easy-2-player",
```

```
 "pwd": "<agent password>",
 "url": "<server url>"
}
```

The response agent gets from the server is formulated as below :

```
{
 "errors" : [],
 "messages" : [],
 "action-requests" : [
    { "run" : 20, "percept" : ...}
 ]
}
```

The action requests is a list of actions that the agent is expected to return the actions for in the next iteration. Errors and messages are self explanatory. The implementation of the server was done in python mainly using frameworks such as requests and json.

3.2 Search

Now that we have an understanding of the valid moves, their representation and the interaction with the server, we will try to understand how these moves are computed. For any agent to move the pegs, it needs to sense the environment(i.e. the surrounding neighbour positions and their values(empty or pegs) in this case). Once it has perceived the environment, it needs to take corresponding action. In order to generate an action, it needs to find a possible move from the list of moves mentioned in section 2.3. This sub process of finding a move requires the agent to execute a step-by-step procedure is termed as "Search".
There can be two approaches for performing a search, namely Greedy and Adversarial. In the below sections, we will understand how both of these search methods work.

3.2.1 Greedy Search

Greedy search is an algorithmic paradigm that makes locally optimal choices at each stage with the hope of finding a global optimum. In other words, at each step, the algorithm selects the best available option without considering the overall consequences. Greedy algorithms are often used for optimization problems, where the goal is to find the best solution from a set of possible solutions. The optimal behaviour of a greedy algorithm is subject to the problem that one is trying to solve. In some cases, it may yield the optimal solution, in other cases it may fall short. A classic example of greedy search is Dijkstra's algorithm. As in the Figure 3.1, we can see that the greedy algorithm fails to maximise sum of nodes along a path from top to bottom because it lacks the

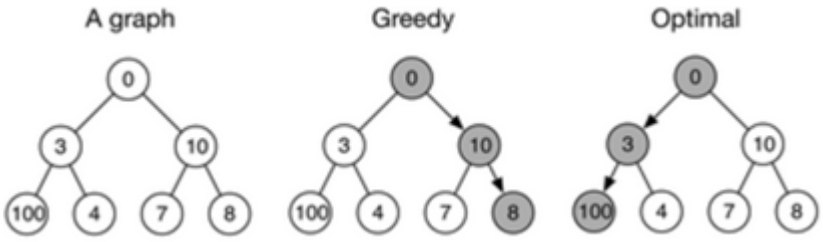

Figure 3.1: Example : Greedy Search

foresight to choose sub-optimal solutions in the current iteration that will allow for better solutions later.

3.2.2 Adversarial Search

Adversarial search, also known as game-playing search, is a field of study within artificial intelligence (AI) and computer science that focuses on designing algorithms to make decisions in competitive environments. The term "adversarial" refers to situations where multiple agents or players have conflicting interests. The mini-max algorithm is a fundamental approach when using adversarial search. The algorithm involves recursively evaluating the possible outcomes of moves, assuming that the opponent makes optimal moves. In almost all cases, the algorithm computes the optimal solution. Let us explore some of the examples of adversarial search algorithms for game playing in the below sections.

Minimax Search

The minimax algorithm is a decision-making algorithm used in game theory and artificial intelligence for minimizing the possible loss for a worst-case scenario. The basic idea behind minimax is to recursively explore the game tree, where each node represents a possible game state, and each edge represents a possible move. The algorithm alternates between two players: maximizing player (MAX) and minimizing player (MIN).

The maximizing player (MAX) aims to maximize its chances of winning by choosing the move that leads to the highest possible score. The minimizing player (MIN) aims to minimize the chances of the maximizing player winning by choosing the move that leads to the lowest possible score. These scores are assigned based on some evaluation function, which determines the desirability of a particular game state. By recursively exploring the game tree, minimax determines the optimal move for MAX. Figure 3.2 shows an example of Minimax Search.

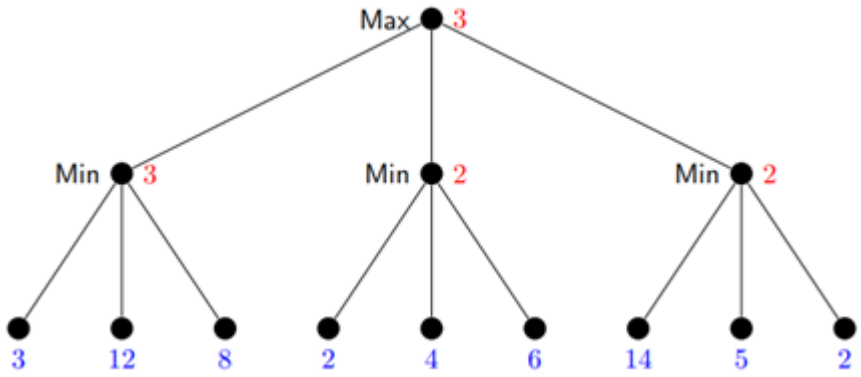

Figure 3.2: Example : MiniMax Search

Alpha Beta Search

Alpha-beta pruning is an optimization technique applied to the minimax algorithm to reduce the number of nodes evaluated in the search tree. It prunes branches of the search tree that cannot possibly influence the final decision, thus reducing the computational effort required to find the optimal move. Alpha-beta pruning exploits the fact that as the search progresses, the bounds of the possible scores for each node become clearer. It maintains two values, alpha and beta, which represent the minimum score that the maximizing player (MAX) is assured of and the maximum score that the minimizing player (MIN) is assured of, respectively.

During the search, when evaluating a node, if the current player (MAX or MIN) determines that the node's score is worse than what the opponent can already achieve elsewhere, it prunes the sub tree rooted at that node. This is done by comparing the node's score with the alpha-beta bounds. If the score is worse than the alpha value for MAX or better than the beta value for MIN, then further exploration of that sub tree is unnecessary, and the search moves on to other branches. Alpha-beta pruning eliminates the need to evaluate certain nodes in the search tree, leading to a significant reduction in the number of nodes explored and thus improving the efficiency of the search. Figure 3.3 shows an example of Alpha Beta Search.

3.3 Algorithm

Using the above two concepts, this section describes the end-to-end naive algorithm used to solve the problem. This algorithm will helps to understand how the intelligent agent plays the board game with either two or three players and aims to win the game.
The algorithm can be described as below:

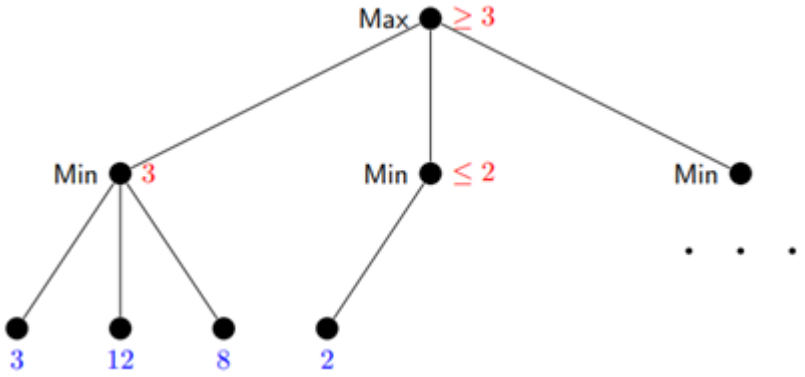

Figure 3.3: Example : Alpha Beta Search

- Agent sends a first HTTP request to the server including the agent details such as name and password of the agent. It should also contain an empty set of actions.
- The server sends a HTTP response back to the agent which has information regarding the connection. This also indicates the agent to send the next possible actions.
- The response we receive contains the initial arrangement of all the pegs of all the players on the board. The agent then computes the possible move.
- To compute a possible move, the agent tries to map all the positions returned by the server on to a NumPy array.
- Further, in case of a greedy approach, the agent finds the peg which has minimum distance to their respective homes. Once found, the agent moves the peg in the direction of the home of the peg.
- If we follow an adversarial approach, the agent maximises the probability of a chain hop in the direction of home of the peg ensuring a fast jump over pegs of other player(s).
- Once a valid move is computed, the agent forms an action request and sends it to the server object.
- The server updates the positions and computes the next moves of other player(s) which are sent back to the agent in form of a server response.
- The agent again computes the next move and this process continues until a player has won the game or the game has reached an end state.

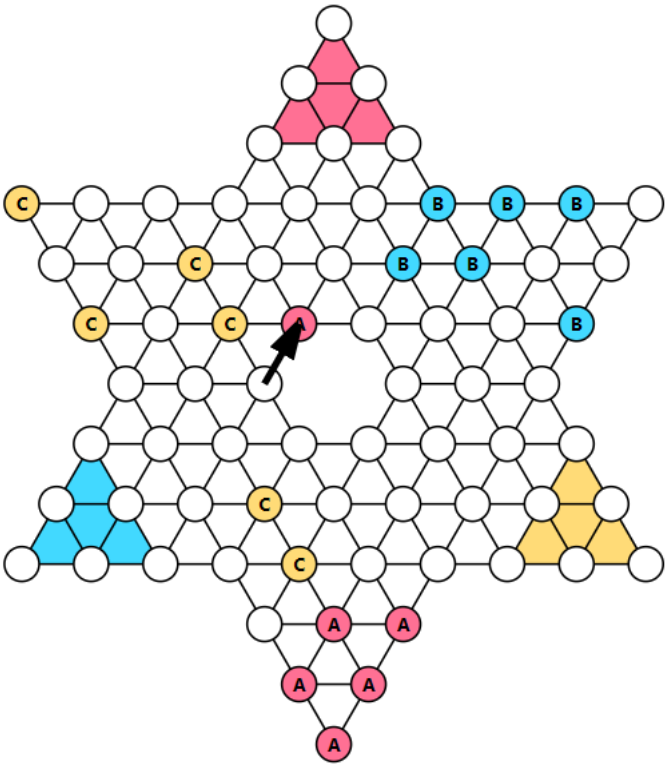

Figure 3.4: Moves : Greedy Approach

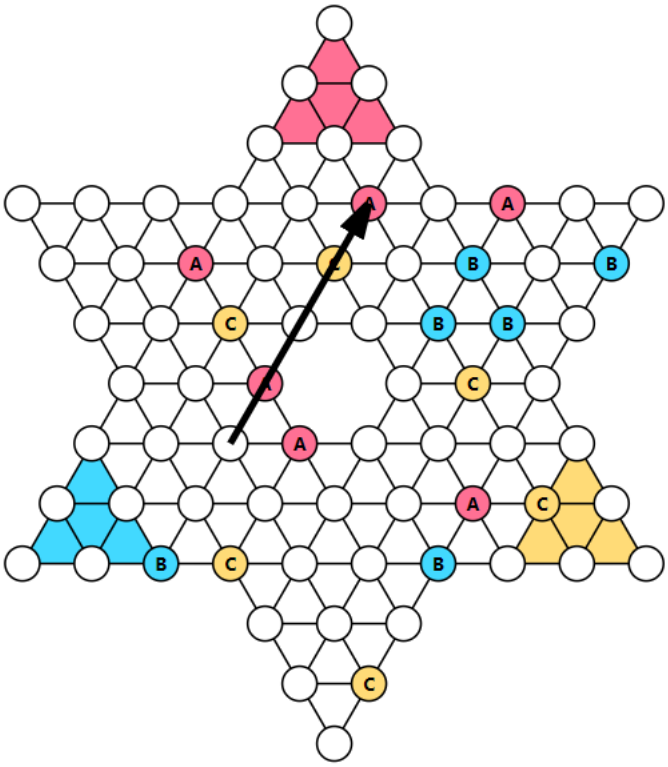

Figure 3.5: Moves : Adversarial Approach

Two possible approaches for the moves listed in Section 3.2 are depicted with the help of Figure 3.3 and Figure 3.4.

In the Figure 3.3, we can see that only one peg of Player A is nearer to its home and has a minimum distance between its own location and location of its respective home. Hence, in a greedy approach, the agent tries to move that peg.

In figure 3.4, we can see that the agent tries to increase a probability of a chain hop in order to move the peg more fast than in the greedy approach.

3.4 Technologies Used

As a part of implementing the proposed solution and its methodologies, we used the following technologies:

- **Python** as the base programming language
- Framework **itertools** for effective traversal
- Framework **json** to create, access and pass adequate information as json objects
- Framework **requests** to implement a client-server kind of architecture, also send and receive requests and responses respectively.
- Framework **logging** to log messages for creating a better understanding to the end user
- Framework **NumPy** for representation of the non-rectilinear co-ordinates and compute the effective moves.

Chapter 4

Results and Evaluation

The purpose of this chapter is to present the findings recorded throughout the experiment, validate our objectives and emphasize on the contributions we stated earlier in the Section 1.3.
This chapter describes the performance of the agent in various player environments. It also gives an overview of the rating of the agent evaluated by the server in different game board configurations.
In order to evaluate the agent, we make it play in several game board configurations which may be two or three player with increasing difficulty. For the agent to be evaluated and assigned a rating, it is expected by the server that the agent plays a particular board configuration at least 50 times. Once this condition is sufficed, be it resulting in any state(i.e. irrespective of which player is winning), the server then assigns a rating to the agent indicating how strong the agent is. This value is an average value based on what outcomes the game results in at the end state. The outcome indicates the number of points which are assigned to the player when a single iteration of a game halts. It is also important to note that if at all at any given point, the agent has no legal move remaining, the blocking player loses the game and gets an outcome 0 indicating zero points for that game iteration.
The outcome assigned to each game iteration depends on the number of players. Table 4.1 and table 4.2 show the finishing places of a player(i.e. when all pegs are relocated in their home locations) and the outcome that they are assigned to in a three player and two player game board configuration respectively.

After testing the agent to play in both two player as well as three player con-

Finishing Place	Outcome
First	2
Second	1
Last	0

Table 4.1: Outcome : 3 Player Game Configuration

Finishing Place	Outcome
First	1
Otherwise	0

Table 4.2: Outcome : 2 Player Game Configuration

Environment	Rating
Two player environment	1.0
Three player environment	0.86

Table 4.3: Performance Evaluation of the agent

figuration environments, we recorded the relevant results. Table 4.3 illustrates the performance of the agent in both the scenarios. The rating is described as the average points (i.e. outcomes) of 50 consecutive games played by the agent in a particular environment.

It can be believed that, the agent has shown on par performance in a two player configuration as it was evaluated with a rating of 1.0. In case of a three player variant, we see a decline in the rating which evaluates to 0.86 as we followed a greedy approach while implementing our agent. Had we followed a more adversarial approach, we would have also seen the agent performing quite well in the three player variant as well.

In the Figure 4.2, it can be seen the results of a two player environment where our agent (Agent259) competed and it got a rating of 1.0

Similarly, in the figure 4.3, it can be seen how our agent performed in a three player environment. As the agent did not include some adversarial part implementation, it performed with a slightly lower rating as compared to others where our rating was 0.86 and the highest rating was 1.84.

Addition of these heuristics to the algorithm would make the algorithm more robust which can be taken up as future prospects of the proposed approach. It is recommended to apply some other techniques such as Iterative Deepening to minimax search which can implement complex heuristics. Alternatively, new methods could be explored such as combining existing approach with neural networks or solving the problem using Monte Carlo Tree Search approach.

Agent	Best rating	Current rating
Agent259	1.0	1.0
Agent707	1.0	1.0
Agent569	0.9	0.9
Agent663	0.84	0.7
Agent980	0.54	0.54

Figure 4.1: Results : 2 Player Environment

Agent	Best rating	Current rating
Agent707	1.84	1.84
Agent980	1.54	1.48
Agent259	0.86	0.84
Agent663	0.68	0.56

Figure 4.2: Results : 3 Player Environment

Chapter 5

Conclusion

As per the abstract stated, the system implements effective exploration techniques to play a board game FAUHalma and achieves the main goal of the peg relocation in their homes. This system also proves that we can implement many such agents to play various board games which have several configurations and evaluate them.
In this report, we also explored how to effectively work with a non-rectilinear grid. A key challenge whilst working with these type of grids is data representation and efficient mapping. We can conclude that the way we represented the non-rectilinear grid co-ordinates mapping to a NumPy array and its corresponding co-ordinates, turned out in favour of the agent as we got a very high and relevant rating when the agent was tested in both the stated environments. In future, combining the adversarial approach would also contribute in betterment of the agent rating.

Bibliography

[1] Jan Frederik Schaefer *Assignment SS23.1.2:FAUHalma* URL : *https://kwarc.info/teaching/AISysProj/SS23/assignment-1.2.pdf*

[2] Micheal Kohlhase *Notes : Artificial Intelligence I* URL : *https://kwarc.info/teaching/AI/notes.pdf*